Special Feature 01
関数型プログラミング
Special Feature 02
2014年IT業界潮流
2014 February 02
[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

Software Design

2014年2月18日発行
毎月1回18日発行
通巻346号
（発刊280号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
1,280円

Special Issue
京セラコミュニケーションシステム社の実現力
本音を語るコンパで企業・組織を活性化！

Special Feature 02
目利きによるトレンド予測
2014年IT業界はどうなるのか？
原点に戻りながら次の一手を考える

Special Feature 01
λ（ラムダ）式からはじめませんか？
関数型プログラミング再入門

Lisp | OCaml | Haskell | Python | Erlang | Java | Ruby |

Re-Introduction
Functional
Language

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62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。

NaCL/PNaCL

Webブラウザ用のネイティブコード実行基盤「PNaCL」

米Googleが開発するWebブラウザ「Google Chrome（以下、Chrome）」は、常に最新のWeb技術を取り入れながら成長するその進化の速度が強みの1つとなっています。そのChromeの新機能として話題になっているのが、Chrome 31より正式サポートされた「PNaCL（Portable Native Client）」です。PNaCLはWebブラウザ内でネイティブコードを実行するための技術で、生成したプログラムを再コンパイルすることなく、さまざまなプラットフォームで動作させられる点が大きな特徴です。

Chromeでは従来よりPNaCLの前身となる「NaCL」によってネイティブコードの実行をサポートしていました。NaCLもWebブラウザ向けのネイティブアプリケーション実行基盤で、C/C++のソースコードからWebブラウザで実行できるネイティブコードを生成して使用します。ただしNaCLの場合には、コンパイル後のネイティブコードがCPUに依存しており、CPUの種類が異なる環境間では互換性がないという問題がありました。

PNaCLの場合は、LLVM（Low Level Virtual Machine）の中間コード（LLVM-IR/bitcode）を採用することで、このCPU依存問題を解消しています。すなわち、PNaCLのコンパイラは直接ネイティブコードを生成するのではなく、いったん中立的なフォーマットの中間コードを出力します。ChromeのPNaCL実行環境はこの中間コードを読み込み、クライアント側でネイティブコードに変換したうえで実行するしくみになっています。中間コードまでは環境に依存しないため、PNaCLのプログラムは単一のコードでさまざまなプラットフォームで実行できるというわけです。

NaCL/PNaCLアプリの開発

Googleでは、NaCLおよびPNaCLのアプリケーションを作成するための開発環境として「Native Client SDK」を提供しています。プログラミング言語としてはC/C++がサポートされており、NaCL用にはgccベースのツールチェーン（nacl-gcc）が、PNaCL用には中間コードの生成をサポートするツールチェーン（PNaCl tool chain）が用意されています。

NaCL/PNaCLのプログラムからWebブラウザの機能やその他のクライアント機能にアクセスするためには、SDKに含まれている「Pepper API」と呼ばれる独自APIを使用します。Pepper APIでは、ローカルファイルやリモートファイルの入出力、マウスやキーボードをはじめとする各種入力機能、2D/3Dグラフィックス、オーディオ再生、JavaScriptインターフェースなどが利用できます。

セキュリティの確保のために、NaCL/PNaCLのクライアント側の実行環境はサンドボックス化され、メモリ空間の利用や命令の実行などについてさまざまな制約が設けられています。また、コード検証機を備えることで、システムコールなどの危険な命令の呼び出しを防止しています。

作成したアプリケーションを配布するうえでの両者の大きな違いは、NaCLの場合はChrome Web Storeでの配布が原則となっているのに対して、PNaCLは一般のWebサイト内に埋め込んで使用することができるという点です。そのほかの興味深い点としては、PNaCLのほうは原理的にはLLVMの中間コードにコンパイルできれば、どんな言語でも対応できるということが挙げられます。現在サポートされているのはC/C++のみですが、将来的には多言語対応というシナリオも出てくるかもしれません。

ネイティブコードサポートの背景

GoogleがNaCL/PNaCLの開発を推進する背景には、JavaScriptの最適化によるパフォーマンス向上が限界に近づいているという事情が見え隠れしています。もちろん、同社が依然としてJavaScriptの性能向上に積極的であることはV8エンジンの開発実績などからも明らかです。しかしユーザ体験のさらなる充実を目指すために、JavaScriptとは異なるアプローチも重要だということでしょう。

一方で、このアプローチに懐疑的な意見もあります。とくに強く懸念されるのは、NaCLはWeb標準からは大きく外れるためWebアプリの移植性が犠牲になり、Webの断片化につながるのではないかということです。事実、現状ではNaCLやPNaCLはChromeでしかサポートされていません。これらの懸念の払拭は、NaCL/PNaCLがスタンダードになるために不可欠な要素と言えるでしょう。SD

Native Client
https://developers.google.com/native-client/

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

Contents

Software Design Feb. 2014

1

λ（ラムダ）式からはじめませんか?

関数型プログラミング再入門

技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

http://gihyo.jp/dp

GIHYO Digital Publishing 未来の"普通"を、今。

Google play

amazon.co.jp

※販売書店は今後も増える予定です。

法人などまとめてのご購入については
別途お問い合わせください。

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

ソフトウェア・デザイン［目次］

Contents

Software Design Feb. 2014

2

第2特集　Special Feature 02

目利きによるトレンド予測

2014年IT業界はどうなるのか?　069

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/

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

ソフトウェア・デザイン［目次］

Contents

Software Design Feb. 2014

3

Column

Development

OS/Network

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

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

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

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

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

定平誠、須藤智　著
A5 判／ 544 ページ
定価 1,764 円（1,680 円＋税）
ISBN978-4-7741-6093-1

栢木厚　著
A5 判／ 456 ページ
定価 1,869 円（1,780 円＋税）
ISBN978-4-7741-6091-7

きたみりゅうじ　著
A5 判／ 464 ページ
定価 1,974 円（1,880 円＋税）
ISBN978-4-7741-6180-8

山本三雄　著
B5 判／ 544 ページ
定価 1,554 円（1,480 円＋税）
ISBN978-4-7741-6095-5

あなたを合格へと導く一冊があります！

大滝みや子、岡嶋裕史　著
A5 判／ 736 ページ
定価 3,129 円（2,980 円＋税）
ISBN978-4-7741-6111-2

加藤昭、芦屋広太、矢野龍王　著
B5 判／ 464 ページ
定価 1,764 円（1,680 円＋税）
ISBN978-4-7741-6110-5

大滝みや子　著
A5 判／ 608 ページ
定価 3,129 円（2,980 円＋税）
ISBN978-4-7741-5940-9

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

岡嶋裕史　著
A5 判／ 656 ページ
定価 3,024 円（2,880 円＋税）
ISBN978-4-7741-6099-3

エディフィストラーニング株式会社　著
B5 判／ 392 ページ
定価 3,024 円（2,880 円＋税）
ISBN978-4-7741-6100-6

山平耕作　著
A5 判／ 720 ページ
定価 3,465 円（3,300 円＋税）
ISBN978-4-7741-6089-4

エディフィストラーニング株式会社　著
B5 判／ 496 ページ
定価 3,360 円（3,200 円＋税）
ISBN978-4-7741-6006-1

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

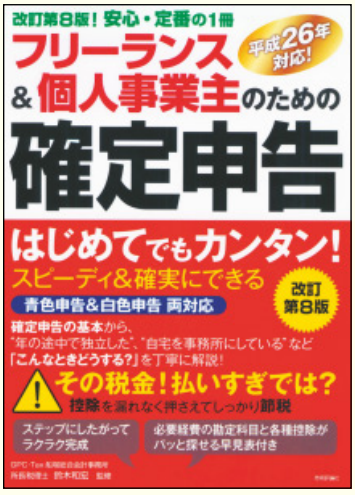
技術評論社の 確定申告本

平成26年3月締切分

初めてでも大丈夫！ マネして書くだけ確定申告 平成26年3月締切分

山本宏　監修／A4判／176ページ
定価1,449円（1,380円＋税）
ISBN 978-4-7741-6059-7

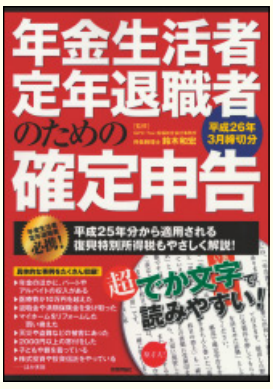
家族が働いていたり副業があるために確定申告が必要なサラリーマンのために、確定申告に必要な書類の作成方法を、簡単にわかりやすくまとめました。サラリーマンだけでなく、フリーランスや不動産オーナー、年金で生活している方などが確定申告を行う場合の書類の作成方法についても、個別のケースごとに詳しく解説しています。本書を参考にすれば、該当するケースを探して、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などに、特におすすめしたい1冊です！



フリーランス＆個人事業主のための確定申告 改訂第8版

鈴木和宏　監修
A5判／240ページ
定価1,554円（1,480円＋税）
ISBN 978-4-7741-6012-2

フリーランス＆個人事業主として働く方の確定申告はこの1冊で決まり！確定申告・帳簿付けの基本から、所得税・税額控除の計算法、申告書への記入の仕方まで、ステップにしたがって、確実にスピーディーに手続きができます。医療費や保険・共済など各種控除の情報も充実しており、節税ポイントもしっかり解説。青色申告・白色申告両対応です。支出や収入から勘定科目をすばやく検索できる早見表や、法人成りをした場合のメリット・デメリットなど、確定申告シーズン以外も、事業を営む皆さまのお役にたてる情報が満載です。



年金生活者 定年退職者 のための 確定申告 平成26年3月締切分

鈴木和宏　監修／A4判／144ページ
定価1,554円（1,480円＋税）
ISBN 978-4-7741-6027-6

世界一ラクにできる確定申告

全自動クラウド会計ソフト「freee」で仕訳なし・入力ストレス最小限！

原尚美・山田案稜　著
A5判／264ページ
定価1,659円（1,580円＋税）
ISBN 978-4-7741-6119-8

恐怖の3月15日 ―確定申告の締切を目前に、「ただでさえ忙しいのにもっとラクにできないの？」と思いませんか？そんな悩みを解決できるとして話題なのが、全自動クラウド会計ソフト「freee」。「めんどうな仕訳なし」「データを自動的に同期して入力ストレス最小限」という同ソフトで最大限ラクする方法はもちろん、「経費で落とせる取引と、落とせない取引の違いは？」といった悩みどころもフォローした、日本初の書籍です！

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

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9回 環境構築のイロハ

イラスト：高野涼香

はじめに

ラック、電源、ネットワーク機器選び、サーバ選び、ケーブリングとやってきました。

一通りの環境がそろったところで、実際に環境を作ってみましょう。

環境があっても、どのようにレシピするか、決まっていないと宝の持ち腐れどころか、重たい文鎮になってしまいます。

まずは自分の中で目標を決めましょう。たとえば、こんな感じでしょうか。

1. 買いそろえた環境で、生活環境のネットワークを構築してみる
2. 公開サーバなどを立ち上げて、ブログなどを立ち上げる
3. 実験して、勉強してみたい

結局のところ、世の中の大半の回線はPPPoE（PPP over Ethernet）を使用する環境でしょうから、ルータの設定（PPPoE、NAT、フィルタなど）、スイッチの設定がほぼ必須になるでしょう。

生活環境のネットワーク環境の構築

生活環境となると、無線LAN環境も欲しくなりますね。自分の生活用とゲスト用のネットワークをVLANで分けて、フィルタを変更するなど試してみても良いでしょう。となると、少なくともDHCPサーバやDNSサーバも欲しくなるので、ルータに任せてしまうか、サーバを立ち上げるか、それらを検討しましょう。LAN内で使用するプライベートIPのセグメントも設計が必要です。

定番の192.168.0.0/24や192.168.1.0/24、192.168.11.0/24などをLAN内で使用すると、結構はまってしまうので、なるべく別のアドレス帯にしたほうが良さそうです。なぜハマるのか……？　それは、端末からVPNなどを使うとき、自分がつないでいるネットワークと自宅がかぶってしまうと、非常に困るからです。なるべく別のセグメントにしましょう。

また、筆者の経験上、192.168.0.1とかをルータに割り当てると、親が勝手にルータを設定のためにつないだらIPがぶつかった、ということもありました。トラブルを防ぐためにも変更しておいたほうが無難です。

DHCP、DNSを立ち上げるのであれば、どんなOS上で稼働させるのか、仮想化するのか、なども検討の必要がありますね。

公開サーバをたててみる

固定IP対応のISPを検討するか、DDNSでやるのか、まずはプロバイダおよび回線選びから始まります。最近は「IPアドレス枯渇だー」とかいいつつも、けっこう安価に固定IPや複数IPを割り当ててくれるプロバイダもありますので探してみてください。単なる公開サーバを立ち上げるのが目的でしたら、最近では各社VPSが1,000円前後～とお手軽な値段

でやっていますので、単純に公開サーバを作りたいというのであれば、価格面ではまったくメリットがありません。やっぱり、ハードウェアから触ってみたい！　VPSじゃスペックが……という方は、ぜひ試してみてください。筆者は、自宅環境＋VPSなどで用途ごとに使い分けています。最近であれば、すべて自前で持つことを考えると、電気代や固定IPのプロバイダ料金の面で、実はVPSのほうが安かったりするケースも多々あります。VPS、自宅に限らず、公開サーバをたててみる際にはセキュリティにはしっかりと気をつけましょう。公開サーバに至らずとも、自宅内のファイルサーバなどから始めてみても良いでしょう。

実験して勉強してみたい

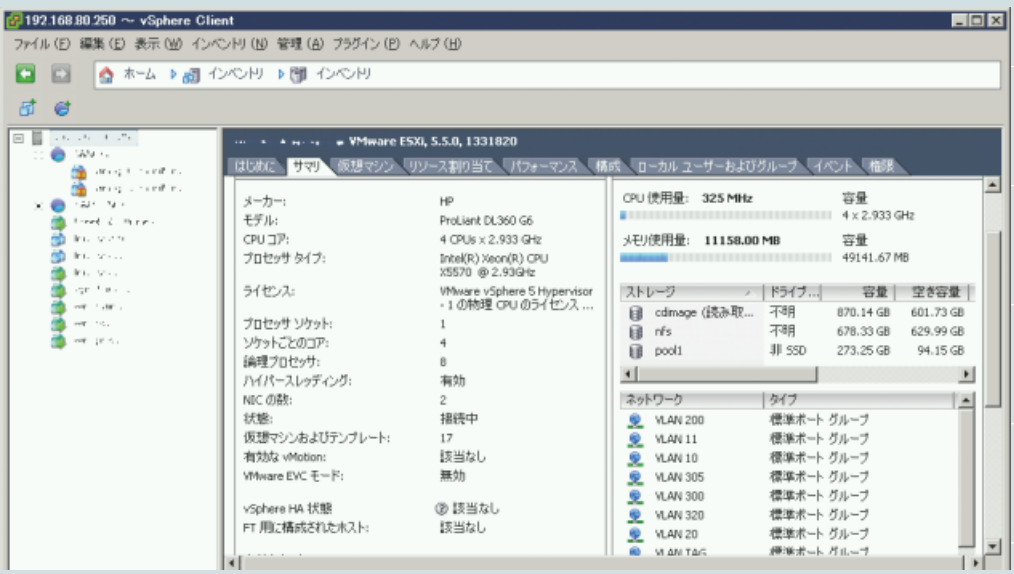
これは、完全に趣味や業務目的といろいろと環境が混在すると思います。筆者は勉強してみたい、趣味も兼ねてAS番号まで取得してしまい、BGPの世界にまで足をつっこんでしまいました。おそらく本稿を愛読されている方は、すでに、スイッチやルータ、ファイアーウォールなどそろっていると思いますので、まずは公開サーバをたててみて、それに付随するネットワーク環境、ファイアーウォールをしっかりと設けて、環境を構築します。次に、ネットワークの冗長化やダイナミックルーティングをやってみたい、VPNを併用して拠点間でやってみたいことなどを考え始め、最後にはWANの冗長化など、勉強してみたいってことで、BGPをやってみるのも良いでしょう。BGPのお勉強をしてみるには、IHANet[注1]などが楽しいかもしれません。

終わりに

結局は、自宅ラックでネットワークを勉強してみようと思ったら、DNSやDHCPなど、基本的な知識、構築、運用。そして、サーバの設定などが必要になってきます。用途ごとにサーバを立ち上げていたらいくら物理リソースがあっても足りませんし、サーバもスペックがよくなっているので、筆者は大半のサービスをVMWare上に環境構築をしています（図1）。気付いたら相当な台数が動いていますね。物理サーバはファイルサーバと公開サーバの一部くらいでしょうか。SD

注1）http://www.ihanet.info/

▼図1　筆者の環境



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 182 >>>>

香港にてSIGGRAPH ASIA 2013開催 ～躍進するアジアのCG～

CGの祭典SIGGRAPHのアジア開催

2013年11月19日から22日の4日間、コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会SIGGRAPHのアジア版「SIGGRAPH ASIA 2013」が香港で開催されました。香港は街中の標識が英語と中国語で並記され、最新の高層ビルが並ぶすぐ横には、古いビルの改修のために竹で組まれた足場が見られる混沌とした街でした。

SIGGRAPH ASIAとしては6回目、香港での開催は2回目となります。今回は世界52ヵ国から7,000人弱の参加者を集めました。夏に米国で開催される本家のSIGGRAPHと比較すると約半分の規模です。規模は小さいながらも内容は充実しており、各セッション・各種展示と、大変クオリティが高いものばかりでした。テーマは“SENSE The Transformation”「変化を感じる」です。

今年はBusiness Symposiumが併設セッションとして開催されました。このシンポジウムはCG/VFX業界でのビジネス視点で催されたもので、テーマは“Hito/Kane/Mono”。そのとおり、人材・資金調達・機材やキャラクタ・マーケットに関するもので、現状の課題や解決方法、今後の課題など、数多くのパネルセッションが行われました。

また、展示会場やギャラリー会場には、地元の中高生も団体で訪れ大変盛況でした。

SIGGRAPH ASIA 2013 公式ページ
http://sa2013.siggraph.org

SIGGRAPH ASIA 2013 論文集一覧（Ke-Sen Huang 氏がまとめている非公式なもの）
http://kesen.realtimerendering.com/siga2013Papers.htm

昨今、SIGGRAPH ASIAが注目される背景としては、大手CGプロダクションが軒並みアジア拠点を設けていることにあります。たとえば、ILMシンガポール、Double Negativeシンガポール、Pixomondo上海、インドMPCなどです。業界全体で人材の確保と、高騰する人件費を抑える展開が行われ、しかも時差を感じさせず、逆にうまく活かす。世界的に環境が大きく変化してきていることが感じられます。

では、SIGGRAPH ASIAの数多くの論文・展示から、興味深いものをいくつか紹介しましょう。

論文より

Inverse Dynamic Hair Modeling with Frictional Contact Paper Abstract Author Preprint

物理的に再現した髪CGの表現。モデルの髪の毛を切ったときの動きもシミュレーションできます。未来の美容院向けの技術（写真A）。

SIGGRAPH ASIA 2013の会場となった香港コンベンションセンター

展示会場より。頭の位置を検知できるヘッドマウントディスプレイと3Dマウスを操作する様子。クリスティデジタル社のブース

※本記事で紹介しているものは国内未発表・未発売のものを含んでおります。

Cost-effective Printing of 3D Objects with Skin-Frame Structures Paper Abstract Author Preprint Paper Video Paper Data

3Dプリンティングも素材によって、または無駄なモデル形状データのためにコストがかさむことあります。本研究は3D形状データを工夫して、より低コストで作成できる方法の提案です。中身が全部詰まったモデルに比べ、必要な強度を保ったまま、約15%の材料で済むように節約できるそうです(写真B)。

Designing and Fabricating Mechanical Automata from Mocap Sequences Paper Abstract Author Preprint Paper Video

Automata(オートマタ)と呼ばれる機械人形を自動生成する方法。動きをデータ化したモーションキャプチャデータから、実物モデルが作れる技術(写真C)。

3-Sweep: Extracting Editable Objects from a Single Photo Paper Abstract Author Preprint Paper Video

2D画像の中のものを3D化し、形状の編集を平易にできるようにする技術。今年、最も注目されていた論文です(写真D)。

The Line of Action: an Intuitive Interface for Expressive Character Posing Paper Abstract Author Preprint Paper Video

CGで描かれた人物モデルのポーズを、線一本で指定するだけで、とても簡単に創り出す方法(写真E)。

Halftone QR Codes Paper Abstract Author Preprint Paper Video Demo Program or Source Code

細かなグラデーション表現ができるハーフトーンQRコードの提案(写真F)。

Real-time 3D Reconstruction at Scale using Voxel Hashing Paper Abstract Author Preprint Paper Video

Kinectカメラを持って部屋の中を移動し、3D空間データを再構成する手法。とても高速です。

3D Wikipedia: Using Online Text to Automatically Label and Navigate Reconstructed Geometry Paper Abstract Author Preprint Paper Video

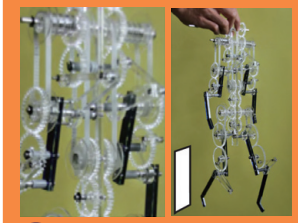
Wikipediaを3D立体化したもの。Wikipediaの単語の説明に3D表示が使える方法。

Interactive By-example Design of Artistic Packing Layouts Paper Abstract Author Preprint Paper Video

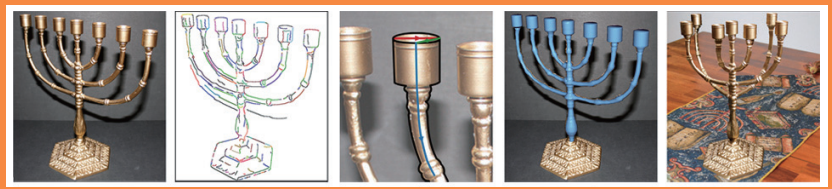
大量の大きさの違う物体の適切なレイアウト手法の提案。1つ動かしただけでも全体が最適化される。

写真A

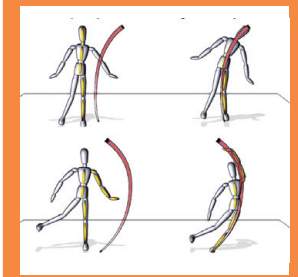
写真B



写真C



写真D



写真E

写真F

SIGGRAPH ASIA 2013の特徴とCG研究の行方

今回のSIGGRAPH ASIA 2013では、とくにシンガポール、インド、上海のプロダクションに勢いが感じられました。アジア圏のCGプロダクションではハリウッドの大作映画の特殊効果を請け負う量も増え、徐々にハリウッドと同じような分業スタイルを受け入れるようになってきています。

映像制作に利用するツールに関しても、多少ライセンス料金が高額であっても業界スタンダードのツールやワークフローを採用し、多少はオープンソースのツール、社内開発したツールなどを駆使しながら北米のCG/VFX仕事を請け負う前提で環境構築がなされています。

また、時差のある遠隔地で仕事をするデメリットは減ってきており、膨大なデータの送受信や大容量のストレージ、プレビュー環境、コラボレーションツールの活用も一昔前とは大違いなほどに整備されてきました。

CG研究の分野も、基礎研究が尊重されつつも、実用分野の研究が増えています。大作映画の特殊効果によって現場での技術が進化しつつ、映像分野、デジタルカメラ分野、各種デジタルデバイス、テーマパークのアトラクション、インタラクティブ広告など、すぐに世の中に役立ちそうなものの比率が増えてきています。

また、コンピュータパワーが安価に大量に使えるようになってきたことから、データ駆動の手法や、旧来は遅くて使えなかったような物理演算など、新しい可能性も広がってきました。

2014年夏のSIGGRAPH 2014はカナダのバンクーバーで8月10日から14日に、次回のSIGGRAPH ASIA 2014は中国深圳(しんせん)で12月3日から6日に開催される予定です。日本を含めたアジアでのCG業界、CG研究の広がりがますます期待されることでしょう。SD

gadget 1

BrainLink

http://macrotellect.com/

脳波コントロールデバイス

BrainLinkは脳波を受信するヘッドギアとiOSアプリにより、ゲームやヘルスケアなどに活用するためのデバイスです。iOSデバイス上の専用アプリとBluetoothで接続します。頭部2個所と耳たぶのセンサーで検知します。価格は1,999元（約33,000円）。「ブレインコンピュータ」と銘打っていますがそれほど多くのことができるわけではなく、脳波がリラックスしている状態なのか、集中している状態なのか程度がわかるだけということでした。

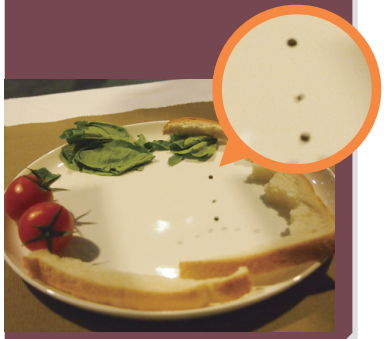
gadget 2

Lapillus Bug

http://star.web.nitech.ac.jp/pdf/2013EC.pdf

虫の空中浮遊

Lapillus Bugはアート作品として展示されものです。アレイ状に配置された小型超音波集束装置による音響浮揚によって、ごくごく小さな物質を、小さな虫のように動かすしくみです。本当の虫ではない、単なる黒い物体がレーザーポインタの赤点を追いかけたり、野菜のトマトを追いかけたりする様は、本物の小バエがいるのかと認識するほどのものでした。デジタル制御された虫とインタラクションできる作品です。



gadget 3

ARAtouch

http://www.cyber.t.u-tokyo.ac.jp/

背面認識の感覚タブレット

ARAtouchは、タブレット端末の裏側を操作している指を鏡に映し、その映像から得た情報を表側の画面にフィードバックすることで、画面に映し出された物体の柔らかさや堅さを感じ取ることのできる研究です。実際にタブレットの裏面の固さが変わるわけではありませんが、操作によって映像が柔らかく動くのか、なかなか動かないのか、映像から触覚を喚起することのできるデバイスになっています。写真はTボーンステーキの部位によって固さが異なることを感じ取るデモです。

gadget 4

Phonejoy

http://phonejoy.com/

スマートフォン専用ゲームコントローラ

Phonejoyは各種スマートフォンを挟んで、モバイルゲーム機化するデバイスです。ボタンコントロールはBluetooth経由で行い、ゲーマーには慣れた十字キーでスマートフォンのゲームアプリを楽しむことができます。筐体は黒のみで69ドル。バッテリー搭載で6〜8時間使い続けられます。単にスマートフォンを挟むだけの機器なので機種を問わず、iPhoneやAndroidスマートフォンの多くの機種に対応します。

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

結城浩の再発見の発想法 9

Default

Default——デフォルト

デフォルトとは

デフォルト(default)とは、もともとは「完全に不足している」という意味で、分野ごとに異なる意味を持ちます。デフォルトという1つの単語が「テニスの試合を棄権すること」を意味したり、「法廷へ出頭しないこと」を意味したりします。金融用語のデフォルトは「なすべき義務を果たさないこと」として「債務不履行」の意味になります。

この記事では「技術用語としてのデフォルト」について書きます。技術用語としては「暗黙の値」を意味します。暗黙の値というのはわかりにくい表現ですが、「この場合はこの値になります」と明示されて**いない**値のことです。

C言語とその仲間のプログラミング言語では、ずばり“default”というキーワードがあります。式の値によって処理が多方向に分岐するswitch文で、明示的に値が書かれていないときの処理を行う個所にdefaultというキーワードを使うのです。たとえば「変数nの値が1の場合、2の場合、3の場合にはそれぞれこんな処理を行い、それ以外の場合にはこんな処理を行う」という場面での「それ以外の場合(明示的に値が与えられていない場合)」のところに、defaultと書きます。「デフォルトの処理」と呼ばれることもあります(**図1**)。

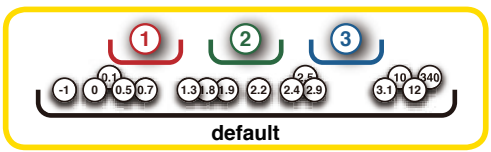
プログラマの会話では「デフォルト」という用語はよく使われます。日常会話で使われるとき、「普通の場合はそうする」や「特別に何も言わなかったらそうなる」というニュアンスになります。たとえばプログラマの宴会で幹事さんが「デフォルトはビールとして、特別に何か頼みたい人いますか」と使っても自然に通じるはずです。「私はウーロン茶」「僕はジンジャエール」のように明示的に頼んだ人以外は、みんなビールになるということです。

安全装置としてのデフォルト

プログラマは、コンピュータが行うべき処理をよく考えてプログラムを書きます。ある変数が取り得る値の範囲にはとくに注意を払う必要があります。さもないと、プログラムは予想外の振る舞いをしてしまいますから。

そんなとき、デフォルトの場合を記述するのが一種の安全装置として働くことがあります。プログラマは変数の取り得る値を列挙して処理を記述しますが、万一そこに漏れがあった場合、すべてデフォルトの場合として捕捉することができるからです。デフォルトの処理としてエラー処理を書いておけば、プログラマの考えに漏れがあった場合でも最後の安全装置としてエラー

▼図1　デフォルトのイメージ図



処理を行うことができるでしょう。

例外なのか?

さてこのような「暗黙の値」を定めるデフォルトですが、ちょっと考えると「その他の例外」のように感じられます。そう考えても悪くはないのですが、実は逆です。「デフォルトの場合」というのは、本来の意味からすると「明示的に言われなかった場合」ですから、そちらのほうが「普通の場合」「あたりまえの場合」なのです。デフォルトじゃない場合、すなわち明示的に示さなければいけないほうがどちらかといえば例外に相当するでしょう。

マナーとデフォルト

考えてみますと、私たちはすべてを明示的に表現して日常生活を送るわけではありません。「普通はそうする」や「何も言わなかったらそうなる」というデフォルトの処理がたくさん決まっているのです。

たとえば公共生活における「マナー」を考えてみましょう。濡れた傘を振り回したりしない。電車の中では大声で話さない。そのようなマナーは明示的に指示されることは少ないですが、多くの人があたりまえのこととして行動します。マナーは一種のデフォルトの処理と言えるでしょう。

たとえば「落とし物があったら交番に届ける」というのはどうでしょうか。落とし物を見つけたときに多くの人が交番に届けるならば、それはデフォルトの処理と言えるでしょう。

このようなマナーやエチケットに相当するものは、大げさに言えば、コミュニティや集団や国の「文化」と呼べるかもしれません。それは「いちいち明示的に言われなくても普通はそうする」という行動が集まったものだからです。

異文化とデフォルト

海外旅行で別の国に行くと、基本的な生活習慣が異なっていて驚くことがあります。たとえば駅で電車を待つときに1列に並ぶかどうか。電車に乗る前に切符の改札があるかどうか。逆に、海外から日本に来た人に(日本人には)あたりまえのことで驚かれることもあるでしょう。

あまりにもあたりまえのことですので、明示的に表現されることのないもの。明示的に表現されなくても多くの人があたりまえとして行うこと。それがデフォルトです。「異文化に出会う」というのは、大げさに言えば「デフォルトの違いに出会う」ということなのかもしれません。

サービスとデフォルト

Webサービスにユーザ登録をするとき、多くのサービスが「あなたに宣伝メールを送ってもいいですか」というオプションを用意しています。自分のメールアドレスとパスワードなどを記入して「登録」ボタンを押す。そのときに「あなたに宣伝メールを送ってもいいですか」のチェックボックスがデフォルトでどうなっているかはサービス側にとっては重要です。デフォルトがオンのほうがずっと多くのメールを送ることができるでしょう。

人間関係とデフォルト

もっとプライベートな場面でもデフォルトが重要な場合があります。たとえば仲良くなりかけている異性のお友達同士でメールのやりとりをしているとき。こちらからいつもより親密なメールを送ったところ、返事が返ってこなかったとします。この「返事がない状態」というのはいったいどう解釈すればいいのでしょう。「うれしいな、ありがとう」や「いやちょっとそういうのは違うから」のように明示的な返事をもらえばはっきりします。でも「返事がない状態」——つまり、デフォルトではどう解釈したらいいのでしょうね。「返事がない」ことがどちらの意味になるかは、2人が作り出す人間関係の小さな「文化」と言えるでしょう。

あなたの周りを見回して、明示的に言わなくても決まっているもの——デフォルト——は見つかりますか。それは誰とどんな文化を共有しているのでしょう。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10回 enchantMOONの誕生［前編］

レーサーミニ四駆の企画者、前田靖幸と共に開発した「前田ブロック」を引っさげ、新しいプログラミングスタイルへの模索が始まりました。

子供のためのプログラミング端末

僕が小学生の頃、世はまさにマイコン全盛期。PC-8800シリーズやMSX、ファミリーベーシックといった、子供がプログラミングを学ぶことを目的にしたハードウェアが数多く発売されていました。しかし同時に、多くの子供たちがプログラミングに挑戦し、挫折を経験した時代でもありました。ごくわずかな子供たちだけが、プログラミングの魅力に目覚め、それを職業にしていくことを考え始めます。そしてそれ以外の子供たちは、「プログラミングって難しい」というニガテ意識を強く植え付けられることになるのです。

子供は誰でも、最初は自分の中に無限の可能性があると信じています。それが子供向けに作られたものであればなおのことです。しかし実際には、BASICは思いどおりのことをやらせるには難しすぎました。子供が自発的にプログラミングの世界にはまり、やがてより複雑で本格的なプログラミングの世界にステップアップするという道筋をもっと丁寧にデザインする必要があったと思うのですが、あの時代の非力なCPUとRAMでは、BASICを動かすのがやっとでした。

そして現代、今や高性能なコンピュータが一般家庭に普及しましたが、“実用品”になってしまいました。かつてのホビーとしてのプログラミングは、子供たちの手から離れてしまったのです。

子供のためのプログラミング環境として、たとえばRaspberry PiのようにPythonのIDLE[注1]やScratch[注2]を使うことを前提にした環境があります。しかし僕がそのとき想像したのは、もっと自由な端末でした。マウスもキーボードもいらない、それ単体が独立したゲーム機のように動き、子供たちが創造性を最大限発揮することにフォーカスした端末です。もっと安価なタブレット端末で構わないのです。お絵描きの延長上にあるプログラミング。そういうものを実現したい。

子供のためのプログラミング端末、そういうものがたとえば1万円程度で発売できれば、子供たちが再びプログラミングに夢中になる時代が来るかもしれない。その中から未来のビル・ゲイツやマーク・ザッカーバーグが生まれるかも——そんなふうに考えていました。

次世代の紙を目指す。しかし障害が……

そうしたアイデアの一方で、僕は「Zeptopad」

注1）Pythonデフォルトの統合開発環境。
注2）MITメディアラボのライフロング・キンダガーデン・グループによって開発された、教育向けビジュアルプログラミング言語。

というアプリの開発をずっと行ってきました。当時このソフトに興味を持ってくれたNECの依頼で、Zeptopadをもとに同社の端末にプリインストールするノートアプリの開発をしていました。しかしiPadでもAndroidでも、僕たちが思うような書き心地には程遠い感覚がありました。どれだけアプリケーションを最適化してもOSが手書きを重視していないので、努力が報われる気がしない時代が長く続きました。

僕たちとしては、ベクトルベースで、なおかつ手書きの線を忠実に再現できるようなものを作りたかったのですが、当時のキャパシティブタッチスクリーンの解像度では充分とは言えませんでした。NECは抵抗皮膜をかなり頑張って最適化したのですが、キャパシティブにしろ抵抗皮膜にしろ、紙と同じように画面に手をつくととたんに筆跡が乱れます。これではとても実用的に使うことはできません。

結局、どれだけ最適化を頑張ってもOSやハードの制約がある以上、紙の書き心地には決して追いつかないという事実は、僕たちを大いに落胆させました。

そのうえ、大メーカーと組んだ協業体制にも落胆がありました。なにしろ、ハードがなければ我々の実現したいソフトは作れないのです。しかし、誰も僕たちが求めるハードウェアを作ってはくれませんでした。

こうなったら、いっそのこと自分たちでハードを作ろう。僕はそう心に決めました。

enchantMOONの誕生

僕は子供向けに、低価格のAndroidタブレットをプログラミング環境とセットで製品化するというアイデアを具体的に検討することにしました。これはハードウェアは非力であっても、ソフトウェアによって付加価値を付け加えることができないかという挑戦でもありました。ちょうどスティーブ・ジョブズが復活したときのAppleが採っていたような戦略です。WindowsPCとほとんど同じハードウェア構成で、値段は高いけれどもMac OSを使いたいがためにMacを購入する、という路線です。

2012年4月から具体的な検討を始め、9月にはエンジニアリングサンプルをいくつか手に入れました。その一方で筐体のデザインをイラストレーターで漫画家の安倍吉俊さんに依頼し、監修を映画監督の樋口真嗣さんにお願いしました。さらに確実を期すために、Androidの足回りに強いエンジニアを国内最大手キャリアからヘッドハンティングしました。

そうして体制を整える傍(かたわ)ら、僕はまったく未経験のハードウェアビジネスにどう取り組んでいくべきか頭を悩ませていました。

実のところ、そもそも最初はAndroid端末を作ろうと思っていました。ただし不安がありました。なぜなら、Android端末として販売する場合、ライバルがあまりにも多いのです。我々のような会社が単なるAndroid端末を発売しても埋れてしまうし、せっかくソフトで付加価値を付けようとしてもその意図がうまく伝わらない可能性もあります。

圧倒的に違うものを作る必要がありました。そうでなければ我々が産み出す意味がないのです。そのために、コンセプトを極限まで尖らせ、他の追従を許さない独創的な商品にする必要があると考えたのです。

しかし、一体どのように独創的な製品を作り出すか。僕にはさっぱり見当がつきませんでした。

その問いに答えを出したのが辻秀美でした。

魔法の紙、NO UI。そしてプログラミングまで可能な万能端末。Zeptopadとenchant.jsという2つのプロジェクトが初めてつながりました。

これは圧倒的に違うモノでした。それどころか、これまでの我々がやってきたことの道のり、そのすべてが凝縮されたと言っていい企画でした。

僕は、自分が生まれてきた意味はここにあった、と思いました。これほど独創的な製品を産み出すことができるのは、世界でUEIしかあり得ないと。

なぜなら、大企業なら確実に潰されるほど斬新な企画であり、仮にやったとしても、他社なら膨大な努力をしてこのまったく新しいパラダイムに向けたアプリを開発してくれる人材を集めなければなりません。たとえば世界的ベストセラー端末のGalaxy Noteでさえも、専用アプリを作ろうと思う開発者は世界に何人いるのでしょうか。

しかし僕たちには2年間に渡って育ててきたenchant.jsのコミュニティがすでにあります。プログラミングの楽しさをもっと大勢の人に伝えたいという活動が、すでに十分広まっていました。僕たちはenchant.jsでプログラムすることの楽しさを知った人々に、こう伝えるだけでいいのです。「こっちに来て。もっとおもしろい世界があるよ」とね。

新しいハードウェアやプラットフォームが成功するためには開発者コミュニティの存在が不可欠です。そしてこの端末は、アラン・ケイが提唱するエンドユーザプログラミングというコンセプトにまで接続されます。コンピュータの歴史について充分な教養を持った人物ならば、この端末を買ってくれるかどうかはともかく、無視し続けるのは難しいはずです。そこに商機があると思いました。そんなニッチな市場は、いまや世界の誰も狙っていません。ブルーオーシャンです。しかしニッチであるがゆえに、大きな会社ではそこに向けた商品を開発するなんて計画にGOサインは出ません。

つまり、これをやるためには、1)アイデアがあり、2)充分な資金があり、3)開発能力があり、4)経営者にプログラミングについての教養があり、5)なおかつ大企業ではない、という5つの条件を同時に満たす必要があります。

当初の目標販売台数は1,000台。これは工場の主張する最小ロットでした。1,000台くらいなら、そうしたニッチ市場には充分なニーズがあるだろうと予測しました。

この新しいコンセプトは、考えれば考えるほど自信が湧いてくるという、僕の仕事の中でも珍しい部類のものでした。しかし同時に、この製品を理解するのはとてつもなく難しいものであることを僕は幾度も痛感させられることになります。これほど本質を理解するのが難しく、作るのが難しい製品はほかにないだろうと思い知ることになったのは、製品を発表するより遥(はる)か以前でした。

僕はまず、自分の組織の中にいる敵と闘わなくてはならなかったのです。

四面楚歌

「まあ夢の話は、もういいからさ」

武市智行は取締役会で熱っぽくこの計画について語る僕をそう制しました。

会社の中長期経営計画について説明しているときの話です。僕はこのハードウェアとそこから生まれる周辺技術を成長ドライバにして、この会社を長期的に成長させていくべきと主張したのです。元銀行員でゲーム畑出身の武市にしてみれば、ゲーム以外の事業の成長性を適正に評価するのは難しかったのでしょう。彼はあくまでゲーム事業を主力として成長を目指すべきだという主張を繰り返しました。

「素人なので申し訳ありませんが、私には、これが到底、売れるとは思えません」

他の役員も口々に不安を述べました。

「個人的な意見ですが、私は賛成できませんね。まったく現実感を持ってこのビジネスの将来性を感じることができないんですが……」

大株主であるベンチャーキャピタル、ジャフコの坂本氏も苦言を呈します。

なんて愚かな連中だ、と僕は思いました。彼らはコンピュータの歴史も、意義も、教養も、何もわかっていないのです。この製品がどうして革命的なのか、理解するのが難しいのはわかります。これだけ詳しく説明してもわからないのだから、もうわかるわけがありません。アラン・ケイが誰なのかもよく知らないような人たちのために時間を費やすのは本当に腹が立ちま

した。

この時代、ソーシャルゲームはもちろん重要です。それは世の中の流れがソーシャルゲームに来ているからです。それもネイティブアプリへのシフトが起きていることもわかっています。しかし、今、この瞬間だけしか見ていなかったら、正確な未来を予測することも、もちろん創りだすこともできません。

僕はハードウェアによって会社を成長させることこそが最終的には最も会社の将来価値を高める方法だと説明を繰り返しましたが、彼らはまともに取り合うことをしませんでした。妥協案として、僕の部署が稼ぐ想定利益を上限として研究開発に投入する、という条件で、取締役会はプロジェクトを渋々承認しました。

その中でも無条件で応援してくれていたのは、大手ゲーム会社から引き抜いて来たゲーム部門のトップ、相原将也と、UEIの基幹事業であるソリューションビジネス部のトップ、水野拓宏でした。相原は僕を全面的に信頼してくれていたし、水野はコンピュータに対する深い教養があったため、僕の意図を理解してくれたのです。ほかの全員が敵に回っていたのですが、稼ぎ頭の彼らが応援してくれたお陰で、このプロジェクトをなんとかスタートすることができたのです。

反対派を説得するためには、デモンストレーションとしてInternational CESへ出展することは必須条件でした。何が出ようが、製品の開発に失敗しようが、会社の宣伝にさえなればいい、ということで彼らはこの無謀とも言えるプロジェクトに目をつぶることにしたのです。

しかし次の裏切りは、意外にも、僕の腹心の部下たちでした。

裏切り

「なんだこれは」

気がつくと僕は会議室で怒鳴っていました。

ポカンとした表情で僕を見つめたのは、このプロジェクトを監督する増田哲郎でした。彼は大学卒業後、新卒からずっと僕の直下で働き、大きな案件は常に彼が事業のフロントとなって実際に回してきた、いわば腹心中の腹心でした。最も信頼できる部下であり、彼の長所も短所も知り尽くしていると思って重用していました。

しかしそのときの僕は、増田に対し強い怒りを感じていました。

「あのう……なにか問題がありましたか？」

最終的な製品仕様が決まった、と連絡を受けた僕は会議室に呼び出され、増田とプランナーの辻から最終の仕様書のプレゼンを受けていました。開発は暫定的な仕様書をもとに進行しており、この開発がこのままスケジュールどおりにいくとこのような製品になる、というプレゼンです。

ところが当初の勢いはどこへやら。僕にとってこの製品の最大の魅力であるシールアプリや、プログラミングという機能が完全に抜け落ちたものでした。

目の前にあるそれは、一風変わった手書きノート端末の企画に成り下がっていました。アプリの追加さえもできず、僕にはまったく魅力を感じることができませんでした。

「これ、君は59,800円で売ってたら買うの？」

増田は答えました。

「それは……プログラマと相談した結果、実装を考えると、現実的にはこういう仕様しかないと……」

それで僕はああ、と思いました。“病気”が始まったな、と思ったのです。どんなモノ作りの現場にもある“病気”です。

コンセプトがどれだけ尖っていても、否、尖っているからこそ、プログラマが何を作ればいいのか焦点が定まらなくなり、仕様書を策定する段階で、「それはできない」「それは間に合わない」と拒絶し、最終的に出来上がるのが、無難そのもののどこにでもある平凡な製品になってしまうという病です。SD

コレクターが独断で選ぶ!

偏愛キーボード図鑑

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

第10回

写真1　Comfort Keyboard Original

腕に優しい姿勢でタイピングできる

Comfort Keyboard Original & SafeType

はじめに

今回は久々にエルゴノミクスキーボードを紹介します。通常、タイピングをする際は、机の表面に対して手のひらが並行になるように手を置くのが一般的です。しかし、それでは腕をひねることになります。手のひらは机の表面に対して垂直となるのが腕に優しいスタイルだ、という考え方も存在します。エルゴノミクスキーボードの中には、手のひらが机の表面に対して垂直に近い状態でタイピングできるように設計されているものも存在します。今回はそんなタイピング方法が可能なキーボードを取り上げます。

Comfort Keyboard Original

Comfort Keyboard Company ,Inc.が販売しているエルゴノミクスキーボードです(**写真1**)。公式ページ[注1]を見る限りでは、本体色のバリエーションが黒と白、接続がUSBとPS/2というバージョンがあるようです。また、筆者は見たことがありませんが、専用のフットペダルも存在するようです。

特徴

数々のおもしろい特徴を備えた立体型で分離可能なキーボードです。

- **3つのパーツに分離でき、好きに組み合わせられる**
- **キーボードの角度を変更できる**
- **キー配列を変更できる**

キーボードを3つの部位に分離できます。それぞれ左手で打つ部分、右手で打つ部分、テンキーの部分です。これらは鉄板に固定してずれないようにしてありますが、鉄板から取り外して順序を入れ替えることもできます(**写真2**)。「左利きだからテンキーは左側に置きたい」「テンキーを中央に置き左手で打つ部分と右手で打つ部分を離して肩が窮屈にならないようにしたい」といったことが実現できます。

また、3つの部位の角度は自由自在に変更できます(**写真2**)。単純に角度をつけるだけでなく、回転させることもできます。ホームポジションで打つキーを縦にならべて、腕をひねらずに打つことも可能です(**写真3**)。

キー配列も自由自在に変えられます。変更した内容はキーボード側に記憶されるので、どのような場面でも変更した配列が使えます。マクロも使用可能で、複数のキー入力を1

写真2　順序入れ替えて、角度を変更した様子

注1）http://www.comfortkeyboard.com/keyboards_comfort.html

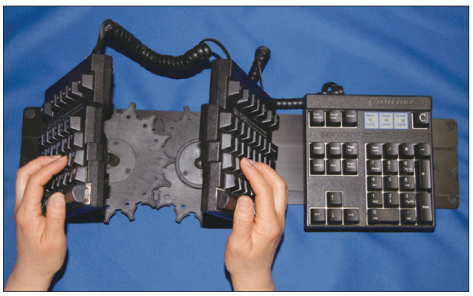


写真3　腕をひねらずにタイピングできる

つのキーに割り当てられます。マクロを定義すると既存のキーを1つ潰すことになると思われるかもしれませんが、同製品の場合、必ずしもそうではありません。マクロを実行するキーは好きなキー（たとえばPage Upなど）に定義するのですが、通常はそのキーを押してもマクロは実行されません。同製品にはComfort Keyというキーが存在し（**写真4**）、そのキーを押した直後にマクロを定義したキーを押すと、マクロが実行されるようになっています。キー配列の変更もマクロの定義もキーボードから行え、特別なソフトウェアを必要としないため、設定したいと思ったときすぐに実施できます。

入手方法

筆者の知る限り、日本で取り扱っている店はありません。海外のAmazon.comでは取り扱っており、値段は$389です。Amazon.comの場合、キーボードは日本に直接発送してくれないことも多いので、一番簡単な入手方法はオークションサイトを使うことです。eBayでは比較的よく見かけ、落札価格は$100を切ることもあれば、$200を超すこともあります。$200以下であれば狙い目です。

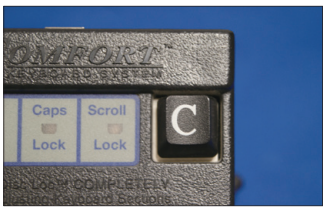


写真4　Comfort Key

SafeType ergonomic keyboard

ErgoType BVが販売するエルゴノミクスキーボードです[注2]（**写真5**）。黒色と白色のカラーバリエーションが存在します。

特徴

特徴は見た目のとおりその形状です。腕をひねらずにタイピングが行えるように、通常のキーが縦に配置されています（**写真6**）。ただ、その形状の結果として、キーが見づらくなっています。通常のキーボードでは見下ろせば良いだけですが、SafeTypeでは顔を横に向ける必要があります。その対策として鏡が付いており、鏡の反射で縦に配置されたキーが見えるようになっています。

形状以外は一般的なキーボードととくに変わらず、普通のメンブレンキーボードです。コンセプトに沿ったシンプルなキーボードですが、値段が高く入手しづらいのが難点です。

入手方法

日本で取り扱っている店はないようです。Amazon.comでは取り扱っており、$280ほどです。オークションサイト（eBay）で比較的よく見かけるキーボードで、落札価格は$100～200ほどです。

◆ ◆ ◆

手のひらを机の表面に対して垂直にかまえて操作するスタイルは、キーボードだけでなくマウスでも存在します。Vertical Mouseと言われています。何種類か製品が出ており、日本でも秋葉原などの店頭で入手可能です。まずはマウスで試してみて、自分のスタイルに合うようなら、Comfort Keyboard Originalなどの利用を考えてみると良いでしょう。SD


写真5　SafeType ergonomic keyboard


写真6　SafeTypeの使用例

注2）http://safetype.com/index.php

秋葉原発!

第40回

はんだづけカフェなう

Makerスペースを始めてみた

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

はんだづけカフェ

この連載のタイトルにも含まれている「はんだづけカフェ」[注1]は、電子工作のための道具や場所をシェアできるオープン・スペースです。秋葉原の端にある末広町という場所で、電源やネット環境、ハンダごてといった、ちょっとした作業ができる環境と、そこにいる誰かとちょっとした会話をしたりする場を提供しています（**写真1**）。

はんだづけカフェは、ArduinoなどをWebショップで販売しているスイッチサイエンスが、「より多くの人に、電子工作の楽しさを知ってもらいたい」ということで始め、場所代や店員さんの人件費などを負担することで運営しています。

はんだづけカフェは、筆者が電子工作という趣味を再開してしばらくした2010年5月9日にオープンし、いろいろなことを学ばせてもらっている場所です。工作をするには多くの知識が必要になるのは言うまでもありませんが、活字にはならないノウハウが多く存在します。そういった情報をやりとりしたりできるスペースはとても有用です。はんだづけカフェで、FPGA-CAFE/FabLab Tsukuba[注2]やTokyo hackerspace[注3]といったほかのスペースの存在も知りました。

そうこうするうちに筆者は電子工作という趣味にどっぷりハマリ、海外のMaker Faireに参加したり、そのときに海外のhackerspaceを訪問してみるのも楽しみになってきたのです。

注1）　http://handazukecafe.com/

▼写真1　はんだづけカフェ（2010年ころ）

東海のスペース

一昨年、筆者は仕事の都合で、東京から名古屋に生活の中心を移しました。そうなると、こういったおもしろいスペースが自分の近くにも欲しくなりました。名古屋にはソフトウェアではgeek bar[注4]という集まりがありますし、隣の岐阜県の大垣市にはIAMASイノベーション工房[f.Labo][注5]もあります。しかし、電子工作中心で、比較的行きやすい場所はないだろうかという思いは消えません。なければ作るしかないということで、こうして筆者は、はんだづけカフェを名古屋に作ることができないかと考え始めました。

スペースの運用形態

ここまでFabLabやhackerspaceなどのスペースを挙げて来ました。日本には上陸してい

注2）　http://www.fpga-cafe.com/
注3）　http://www.tokyohackerspace.org/
注4）　https://www.facebook.com/nagoyageekbar
注5）　http://f-labo.tumblr.com/

ませんが、TechShop[注6]というものもアメリカには生まれています。これらのスペースの違いを簡単に紹介しましょう。

TechShopでは、月$175の会費で、木工、機械工作や溶接といったさまざまな設備のそろった工房を提供してくれます。同社は2006年創業で、このスペースをチェーン展開しているベンチャー企業です。TechShopを起業したマーク・ハッチ氏はKinko's出身ということで、そう聞くとTechShopはKinko'sの工房版とも言えることに気づかされます。

FabLabは、「ほぼあらゆるもの」を作ることを目標とした工房です。MIT（マサチューセッツ工科大）メディアラボのThe Center for Bits and Atomsで行われていた研究から生まれました。世界50ヵ国250ヵ所以上のFabLabが存在し、国内には現在6ヵ所のFabLabが存在します。FabLabには「ファブラボ憲章」[注7]という基本理念や運営のガイドラインが存在し、この憲章が世界中のFabLabで掲示されているはずです。ほかにFabLabとなるための条件も存在し、その中では週一度は無償（か、バーターで）一般に公開する日を設けることなどが定められています。FabLabは教育機関や地方自治体、非営利機関などによって運営されているケースが多く存在します（**写真2**）。

Hackerspaceは、おもにテクノロジに興味がある人々が集まるコミュニティによって運営されるスペースです。1990年代にヨーロッパのhackerたちが集まり、こういったスペースを持ち始めたのがHackerspaceの始まりです。もともとhackとかhackerから連想されるように、ソフトウェアやネットワーキング方面の集まりでした。

21世紀に入ってからHackerspaceはアメリカに輸入され、NYC ResistorやNoisebridgeなど数多くのHackerspaceが立ち上がりました。Hackerspaceにもよりますが、それぞれソフトウェア方面に関心のある人が多いところや、ハードウェア方面に強いところといった特色があったりもします。

TechShopやFabLabは場所を管理する側と利用者側が分かれていますが、Hackerspaceはもっとコミュニティ寄りです。本連載でも何度か紹介しましたが、メンバ費を支払っている人はスペースの鍵を持っていて、公開日に一般の人々にスペースを公開してメンバを集めるという運営方法をよく見かけます。筆者がHackerspace巡りをするときには、公開日を狙ったり、アポを取ったりしてから行くようにしています（**写真3**）。

はんだづけカフェは、これらのどれにも当てはまりません。運用形態はFabLabに近いのですが、ファブラボ憲章を掲げてはいないのでFabLabではありません。

筆者がスペースを開くにあたって、気になったのはおもに店番の人件費です。もちろん家賃

注6） http://www.techshop.ws/
注7） http://fablabjapan.org/fabcharter/

▼写真2　FabLabつくば

▼写真3　London Hackerspace

などの固定費も気にはなるのですが、店番としてアルバイトなどを雇うとなると、必要になるコストは家賃の比ではありません。そして、これらのコストを捻出するために、利用料をいただくというのは考えられませんでした。僕はさまざまな人と一緒に趣味を楽しみたいのであって、店舗経営がしたいわけではないからです。こういったことから、Hackerspaceのような運営形態を採ろうという結論に達しました。

名古屋でニーズはあるのか

スペースを始めたいと考えたころ、筆者は「はんだづけカフェ 名古屋」を始めたいということで仲間を探していました。Hackerspace形式で行くのであれば、一緒に立ちあげる仲間がいなければ筆者1人ですべての手続きや費用の捻出をする必要があります。仲間を探すつもりで少しずつ動いていたところ、名古屋にそういうスペースがほしいと思っている人々と知り合うこともできました。

スペースの継続性を考えると月々の家賃は10万円以下に収めたいと考え、物件を探し始めたりもしました。しかし、交通アクセスの良いところで、かつ、人が出入りしたり、工作機械を使っても良い場所となるとなかなか見つけるのは難しいものです。そうこうするうちに1年近くの月日が経ってしまいました。

仲間

やがて、名古屋にFabLabを作りたいと考えている人たちとも知り合いました。現在、私たちのスペースの代表者をしてくださっている朝尾さんとその友人たちです。朝尾さんは子供たちがものを作る経験ができる場所を作りたいと考え、筆者はものを作ることを楽しむ仲間と出会って一緒に遊べる場所がほしいと考えていました。モチベーションは違えど、人と関わりたいという大きなくくりでは想いは一緒です。

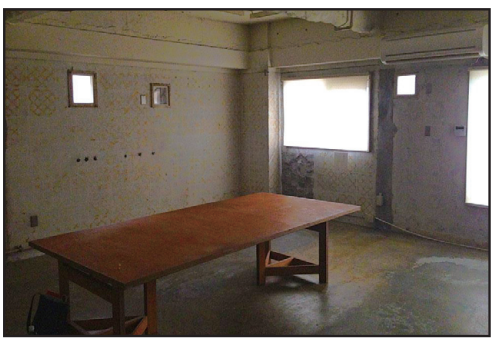
朝尾さんたちと一緒に物件探しなどをした結果、昨年の7月によさそうな物件を見つけることができました。大家さんが、パルル[注8]という多目的スペースを始めた方で、こういったスペースへの理解が深い方だったのです。また、名古屋市内でも新栄という比較的交通アクセスの良い場所にあるということもとても好都合でした。家賃も予算内です(**写真4**)。

工具

場所が決まったので、次は工具などです。私物を持ち寄ったり、知り合いのツテで置かせてもらった工具などが少しずつそろい始めてきました。もちろん3Dプリンタもあります。ボール盤一揃えを持ってきてくださる方がいたり(**写真5**)、大型のCNCミリングを持って行っ

注8) http://www.parlwr.net/about.html

▼写真4 借りたての状態の物件



▼写真5 ボール盤で3Dプリンタの部品に穴を開ける筆者

ていいよと言ってくださる方がいたりと、当初考えていたよりも多くの人が助けてくださいました。工具も一通りそろってきた9月、ささやかなオープニングパーティーを開いてお披露目もしました。また、最近では自分で購入したレーザー加工機（**写真6**）をスペースに置いてくださる太っ腹な協力者も現れました。

運営形態

運営形態ですが、最終的にはHackerspace的な運営になっています。鍵を持っている「コアメンバ」と私たちが呼んでいる人々で家賃などの維持費を負担しています。まだまだ、「来てくれれば確実に楽しんで帰ってもらえる」と胸を張って言える状態ではないのですが、コアメンバになっても良いよと言ってくださる人が見つかったのも驚きです。

皆、それぞれ仕事があるので平日にスペースを開けておくのは難しいのですが、祝祭日にはできるだけ開けるようにしています。スペースを来て下さった方にはとくに費用負担を設けてはいませんが、ドネート（寄付）を受け付けるという形で運営をしています。開いている日は、はんだづけカフェを真似て、Googleカレンダーに書き込んで見てもらえるようにしました。

最近の状況

先ほどの写真のように、借りた段階ではまったく内装もされていませんでしたので、少しずつ自分たちでスペースをよくしようと工夫をしています。たとえば、壁。プロジェクターで壁に何かを映したいと思っても、スクリーンがありませんので、壁から作り始めました（**写真7**）。もともと大工だった方に教えてもらって、ホームセンターで買ってきた材料を使って壁を少しずつ作り始めています。パテで補修などをしてから、最終的には壁をスクリーンにできる塗料を塗ろうとしています。

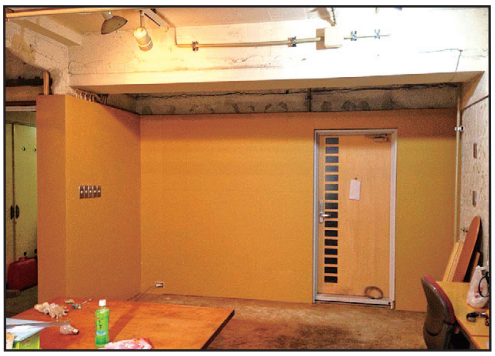
また、まだ私たち自身も加工機を使いこなせている段階ではないため、ワークショップなどを開催しながら加工機を使ってもらえる人を増やそうと思っています。

こんな、始まったばかりのMaker Lab Nagoya[注9]（**写真8**）ですが、名古屋近郊の方はぜひとも遊びにきてください。最近、なかなか忙しくて顔を出せていないのですが、お声がけをいただければ筆者もスペースでお話できるように行こうと思います。SD

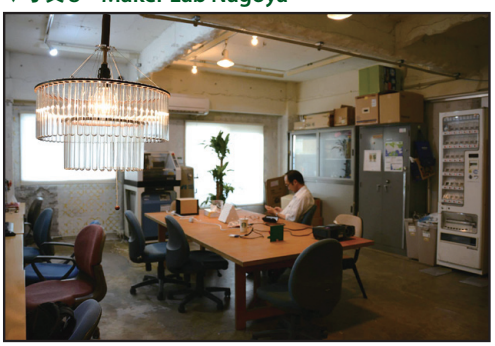
注9） http://makerlab.jp/

▼写真6　レーザー加工機

▼写真7　作った壁



▼写真8　Maker Lab Nagoya



PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは 2014 年 2 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 ノートPC タブレット デュアルアーム DN-10313

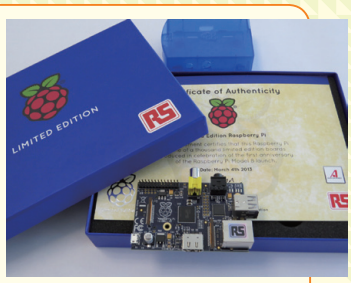
1名

17 インチサイズまでのノート PC と、iPad や 7 ～ 10 インチのタブレットを同時に設置できるアームセットです。それぞれのアームは使いやすい角度に調節でき、デスク上を効率良く使えます。支柱に取り付けるアームの位置は任意に変更が可能。タブレットホルダーは、横向き、縦向きに向きを変えられます。　※本製品に iPad と MacBook Pro は付属しません

提供元 上海問屋　**URL** http://www.donya.jp/

02 Blue Raspberry Pi

3名



Raspberry Pi 誕生1周年記念限定モデル。One-Nine-Design 社が特別にデザインした専用箱に、基盤部分が青色のシリアルナンバー入り Raspberry Pi「Model B Revision 2」、専用ケース、Raspberry Pi 財団代表エベン・アプトン氏の署名入り証明書を同封したパッケージです。

提供元 アールエスコンポーネンツ　**URL** http://rswww.co.jp/

03 ESET FAMILY SECURITY 2014

5名

Windows、Mac、Android の端末を合計 5 台まで利用可能な総合セキュリティソフトの最新版。ホスト侵入防止システム機能や不正侵入対策機能を強化しました。プレゼント提供の製品は 1 年間のみ使用可能。その後の契約更新はできませんが、機能は製品版と同じです。

提供元 キヤノンITソリューションズ　**URL** http://www.canon-its.co.jp/

04 人生って、大人になってからがやたら長い

5名

著者サイン入り！

きたみりゅうじ 著／
A5 判、176 ページ／
ISBN = 978-4-344-02472-2

大人になってからの人生は結構イベントが多い。そんな果てしなく長いように思える「大人になってからの人生」の苦労と喜びを、ときにユーモラスに、ときにセンチメンタルに描くコミックエッセイ。

提供元 幻冬舎、きたみりゅうじ氏　**URL** kitajirushi.jp

05 知識ゼロから学ぶ ソフトウェアテスト 改訂版

2名

高橋 寿一 著／
A5 判、240 ページ／
ISBN = 978-4-7981-3060-6

各種テスト手法の基礎とポイント、アジャイルなど新しい開発手法に対応したテストの考え方など、テスト技術者にとって不可欠な知識と情報を親しみやすい記述や例示でわかりやすく解説します。

提供元 翔泳社　**URL** http://www.shoeisha.co.jp/

06 過負荷に耐えるWebの作り方

2名

株式会社パイプドビッツ 著／
A5 判、224 ページ／
ISBN = 978-4-7741-6205-8

秒間 10,000 ものアクセスに耐えることが求められる国民的アイドルグループ選抜総選挙の Web 投票システム。そのシステムを開発したエンジニア自ら過負荷に耐えるシステム作りについて解説します。

提供元 技術評論社　**URL** http://gihyo.jp/

07 HTML5 ハイブリッドアプリ開発[実践]入門

2名

久保田 光則、アシアル株式会社 著／
A5 判、384 ページ／
ISBN = 978-4-7741-6211-9

フレームワーク「Apache Cordova」による開発から、ストレージの使い分け、タッチジェスチャなどの活用など、HTML5 ハイブリッドアプリ開発に必須の知識をわかりやすく解説します。

提供元 技術評論社　**URL** http://gihyo.jp/

第1特集

Lisp/OCaml/Haskell/Python/Erlang/Java/Ruby

関数型プログラミング再入門

λ式からはじめませんか?

関数型プログラミング言語について、プログラマ・エンジニアが再注目しています。CPUのマルチコア・マルチスレッド化が進んでいるのに、既存の言語ではその機能を十分に活かせないからだと言われています。それがふたたび脚光を浴びている理由です。関数型プログラミング言語は、スクリプト言語に慣れた目では、一見わかりにくい特殊な記述方法が多いので面くらいますが、一度その魅力に取り憑かれると、ファンになってしまうことが多いのです。誕生から数十年を経てもLispプログラマを名乗る方もたくさんいます。本特集では、元祖Lispから、OCaml、Haskellへと解説を進め、Pythonでスクリプト言語の場合を考え、Erlangで実装例を知り、Java SE 8での関数型の導入の状況を押さえ、最後にRubyでの関数型を探求します。抽象度が高いプログラミング言語なのでいきなり習得するのは難しいですが、本特集を足がかりにして再入門してください。

CONTENTS

Lispでウォーミングアップ

Writer るびきち（http://www.rubyist.net/~rubikitch/）／Twitter@rubikitch

関数型プログラミングを再入門する際に避けて通れないのが、Lispです。本章では、関数型プログラミングの元祖であるLispの基礎文法を押さえます。その特徴である、式の評価、リスト、関数定義、ラムダ式の基礎的な考え方を解説し、最後にリスト処理の方法のバリエーションを紹介します。

Lispとは

名前の由来

Lispという言語があることを聞いたことがありますか？　関数型といえばLispについて話しておく必要があります。Lispとは1958年生まれのFORTRANの次に古いプログラミング言語です。

LispとはLISt Processingの略であり、リスト（list）処理を基本としています。ガーベッジ・コレクション、高階関数などの先進的な特徴があの時代からあったのは驚くばかりです。初めての「パソコン」が登場する1974年の15年以上も昔にLispが登場したのは、まさに古代の神秘と言えるでしょう。

Lispの子供達

Lispは容易に実装できるため、たくさんの方言が生まれました。それらを標準化するため、Common Lispが登場しました。ほかにも最小主義のScheme、JVMで動くClojure、Emacsの拡張言語としてEmacs Lispが現在おもに使われているLisp方言です。本稿で取り扱うコードはEmacs LispとCommon Lisp（処理系CLISP）で動作します。

現存する多くのプログラミング言語がLispの影響を少なからず受けています。第7章で取り上げるRubyはMatzLispと言われているほど強く影響を受けています。

()だらけのLispのコード

そして、Lispは関数型言語の元祖と言われています。関数型言語がもてはやされている今、Lispを知っておくことは大きな意味があります。Lispプログラミングをしないとしても、知っておくことで現在使っているプログラミング言語を違った視点でとらえられるようになります。

Lispのコードを見たらほぼ全員が大量の括弧に圧倒されることは間違いありません（**リスト1**）。LispはS式と呼ばれるこのシンプルきわまりない文法から驚くべき表現力を発揮します。

関数型と言われているLispは実はマルチパラダイム言語で、命令型やオブジェクト指向プログラミングにも対応しています。Emacs LispはEmacsを操作するのが目的なので、もはや命令型言語になっています。

▼リスト1　Lispのサンプルコード

```
;;; nの階乗を求める関数
(defun fact (n)
  (if (= n 0)
      1
    (* n (fact (- n 1)))))
;;; 関数呼び出し
(fact 1)    ; => 1
(fact 4)    ; => 24
(fact 5)    ; => 120
```

プログラミング環境の準備

本稿のコードを試すには、当然Lisp処理系が必要です。Emacsを使用しているのであれば、**M-x ielm**を実行すればプロンプトが出てきます(**図1**)。Common Lispの機能を使うためにclライブラリを読み込んでおきます。

そうでなければCommon Lisp処理系CLISPをお勧めします(**図2**)。フリーで手軽に使えるCommon Lispインタプリタです。インストールしたらclispを実行すればプロンプトが出てきます。

Common Lispを常用するならばEmacs上でSLIMEを導入するともっと快適になります。paredit.elを使えば括弧の対応に悩まされることがなくなります。

評価

Lispコードの最大の特徴は、いらいらするほど括弧にあふれるこの構文ですね！　S式と呼ばれるこの構文こそLispの表現力の源です。S式はとても解析しやすく、木構造を表現するのにうってつけの構文になっています。プログラミング言語は処理系によって構文木に変換されますが、Lispの場合は構文木そのものを書いていると言えます。

S式は

(命令　引数　引数...)

という形式になっています。そして引数も、S式を取ります。たとえば、数式

```
1+3*4
```

をS式で表現すると、

```
(+ 1 (* 3 4))
```

となります。S式での数式表現は慣れるまで多少時間がかかると思いますが、見てわかるようにS式は評価の優先順位が明確になります。乗算が加算より優先されるのは常識ですが、Lisp以外でビット演算するには優先順位を知っておく必要があります。

Lispプログラムを実行することを評価といいます。評価には次の規則があります。

1) 数値、文字列、**t**(真)、**nil**(偽・空リスト)を評価すると、そのまま評価結果になる
2) クオートされたS式はクオートが取れてそのまま評価結果になる
3) シンボルを評価すると、その変数の値が評価結果になる
4) リストを評価すると、関数・スペシャルフォーム・マクロを呼び出す
5) 関数は引数をすべて評価してから呼び出す
6) スペシャルフォームとマクロには独自の評価方法がある

▼図1　Emacsの場合(M-x ielmと入力)

```
*** Welcome to IELM ***  Type (describe-mode) for help.
ELISP> (require 'cl)
cl
```

▼図2　CLISPの場合

```
  i i i i i i i       ooooo    o        ooooooo   ooooo   ooooo
  I I I I I I I      8     8   8           8     8     o  8    8
  I  \ `+' /  I      8         8           8     8        8    8
   \  `-+-'  /       8         8           8      ooooo   8oooo
    `-__|__-'        8         8           8           8  8
        |            8     o   8           8     o     8  8
  ------+------       ooooo    8ooooooo  ooo8ooo   ooooo   8

Welcome to GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010

Type :h and hit Enter for context help.

[1]>
```

Lispプログラムはリストの集合体であり、たくさんの関数と少数のスペシャルフォーム、そしてユーザ定義のスペシャルフォームともいえるマクロで構成されています。とくにLispのマクロは超強力でオリジナルの制御構造が定義できるどころか、まったく新しい言語をも構築できてしまうほどです。マクロ定義は高度な話題になってしまうので、とても誌面では伝えきれません。

では、さきほどの式を評価してみましょう。図3の過程をたどります。

Lispではコードとデータが同じように書けます。今評価した式(+ 1 (* 3 4))は+、1、(* 3 4)を要素とするリストでもあります。これをプログラムとみなして評価した結果13という値が得られました。評価器にデータとしてこのリストを渡すには'(+ 1 (* 3 4))のように「'」をつけてクオートします。日本語でいう「」みたいなものです。シンボルもクオートすればシンボルそのものを渡せます(図4)。

コンスセル・リスト・アトム

リスト(list)はLispの代名詞と言えるオブジェクトです。これまで出てきたLispコードもリストです。リストはコンスセル(cons cell)と呼ばれるペアのデータ構造を数珠つなぎにしてつくられます。コンスセルは(CAR(カー) . CDR(クダー))のように表記します。たとえば、図5のように1と2で構成されるコンスセルは(1 . 2)となります。歴史的事情によりコンスセルの左はcar、右はcdrと呼ばれています。

そして、それぞれのコンスセルのcdrを次のコンスセルに設定すれば数珠つなぎになり、リストとなります。(1 2 3)は(1 . (2 . (3 . nil)))と同じです。リストはコンスセルで構成されており、それ自身がコンスセルです(図6)。

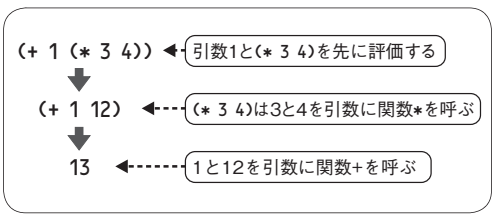
コンスセルとリストを評価器に渡してみましょう。コンスセルをそのまま渡すにはクオートが必要です。carとcdrを取り出す関数、各要素を評価してコンスセルやリストを作成する関数が用意されています(図7、図8)。

コンスセル以外の要素をアトムといいます。数値、文字列、シンボルはアトムです。atom、consp関数で判定できます。

関数定義・ラムダ式

関数定義はdefunスペシャルフォームを使います。ラムダ式はlambdaで定義します。ラムダ式はリスト処理関数で大活躍します。関数を呼ぶ関数funcallはラムダ式も呼び出せます(図9)。

▼図3　式の評価過程

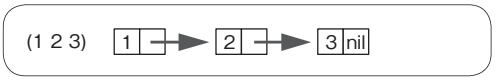


▼図4　評価例

```
ELISP> (+ 1 (* 3 4))
13

ELISP> '(+ 1 (* 3 4))
(+ 1
   (* 3 4))

ELISP> (eval '(+ 1 (* 3 4)))
13

ELISP> pi
3.141592653589793

ELISP> 'pi
pi
```

▼図5　コンスセル

▼図6　コンスセルを数珠つなぎにする



リスト処理

簡単なリスト処理の例として、リストの各要素を2倍にするリストを作成するいろいろな方法を紹介します。

まずは命令型の考え方として、結果のリストを空リストnilで初期化→各要素を2倍にした値を追加→ひっくり返すという方法をとります。consとsetqでリストを更新すると、先頭にどんどん追加されるので、ひっくり返す必要があります。リストの各要素について式を評価する関数mapcがあります。lambdaを書きたくないならばdolistを使います(図10)。

やはり命令型ではLispらしさがまったく感じられません。各要素において同じ計算を行った新しいリストを返すmapcarを使いましょう。これで余計なローカル変数も除去できます。さらに、(lambda (x) (twice x))はtwiceと同じなのでそのまま関数名を渡してしまいましょう。とても短く、関数型という雰囲気が出てきました(図11)。

最後は、究極のLispマクロといえるloopマクロによる方法です。loopマクロは、mapcar以外にも内部にループが使われている処理ならばほぼ何でも書ける「ループ用ミニ言語」というべき代物です(図12)。Emacs LispではCommon Lisp関数(マクロは可)が使いづらい状況にあるのでloopマクロを覚えることは重要なことです。

loopマクロの全容を一度に把握しようとすると複雑すぎて挫折するかもしれません。関数として提供されている処理と同じことをloop

▼図7　コンスセルとリストの例

```
ELISP> '(1 . 2)
(1 . 2)
ELISP> (car '(1 . 2))
1
ELISP> (cdr '(1 . 2))
2
ELISP> (cons 1 2)
(1 . 2)
ELISP> '(1 . (2 . (3 . nil)))
(1 2 3)
ELISP> (car '(1 2 3))
1
ELISP> (cdr '(1 2 3))
(2 3)
ELISP> (list 1 2 3)
(1 2 3)
```

▼図8　N番目の要素、最後のコンスセル

```
ELISP> (nth 0 '(1 2 3))
1
ELISP> (nth 1 '(1 2 3))
2
ELISP> (last '(1 2 3))
(3)
```

▼図9　関数定義・ラムダ式

```
ELISP> (defun twice (x) (* x 2))
twice
ELISP> (twice 100)
200
ELISP> (lambda (x) (* x 2))
(lambda
  (x)
  (* x 2))
ELISP> (funcall (lambda (x) (* x 2)) 100)
200
ELISP> (funcall #'twice 100)
200
```

▼図10　命令型(mapc/dolist)の例

```
ELISP> (setq l '(1 2 3))
↑グローバル変数lを設定する
(1 2 3)

ELISP> (let ((result nil))
          (mapc (lambda (x) (setq result
(cons (twice x) result)))
↑ローカル変数resultをsetqで更新
                l)
         (reverse result))
(2 4 6)

ELISP> (let ((result nil))
         (dolist (x l)
           (setq result (cons (twice x)
result)))
         (reverse result))
(2 4 6)
```

▼図11　関数型(mapcar)の例

```
ELISP> (mapcar (lambda (x) (twice x)) l)
(2 4 6)

ELISP> (mapcar #'twice l)
(2 4 6)
```

で書いてそれを熟語として覚えるのが習得のコツです(図13)。

終わりに

Lispのほんのさわりを紹介しましたが、いかがだったでしょうか？　Lispは奥深い世界です。letの亜種としてlet*があります。条件分岐はif、cond、when、unlessがあります。ループはほかにもdotimes、doなどがあります。リストの要素を分解してローカル変数に設定するdestructuring-bindも面白い存在です。

もしLispを面白いと感じられたのならば、いっそのことエディタをEmacsに乗り換えてしまうのもアリではないでしょうか。Emacsを使っていると否が応でもEmacs Lispに触れることになるのですから。SD

▼図12　loopマクロの例

```
ELISP> (loop for x in l
             collect (twice x))
(2 4 6)
```

▼図13　他のリスト処理とloopマクロの例

```
ELISP> (find-if #'evenp l)
↑最初の偶数を取り出す
2

ELISP> (loop for x in l if (evenp x)
return x)
2

ELISP> (remove-if-not #'oddp l)
↑奇数をすべて取り出す
(1 3)

ELISP> (loop for x in l if (oddp x)
collect x)
(1 3)

ELISP> (reduce #'+ l)
↑合計を求める
6
ELISP> (loop for x in l sum x)
6
```

今熱い！ 快進撃のOCaml

Writer 五十嵐 淳（いがらし あつし）京都大学、Jacques Garrigue（ジャック ガリグ）名古屋大学、
古瀬 淳（ふるせ じゅん）SCB Singapore

航空機エアバスの内部システムでの採用、Webフレームワークへの応用など、OCaml（オーキャメル）を巡る状況は着実に変化し、ふたたび注目を集めています。本稿は、そんなOCamlを再入門（再履修）するために、OCamlの誕生から現在に至るプロセスを見直しながら、その特徴的なしくみや文法を体験していただきます。その中でこの言語の熱気を見つけてください。きっと啓発されるはずです！

はじめに

OCamlは長い伝統を持った型付き関数型プログラミング言語です。前から関数型言語に興味を持つ人なら、Objective Camlという名前で知っているかもしれません。2012年にバージョン4.00がリリースされたとき、1997年から冠していたObjectiveを1文字に短縮してOCamlとなりました。1980年代から続いている本来の名前がCamlなので、2文字目も大文字です。**くれぐれも小文字で書かないでください**。“Objective”が示すように、OCamlにはオブジェクト指向機能も備わっています。しかし、最大の特徴は実行時型エラーを許さない型システムとほぼ完全な型推論（変数や関数の型は書かなくて良い）、そして高速で効率の良いコンパイラです。その3つによって、安全で早いプログラム開発が可能になっています。OCamlでは自由で自然な形でプログラムが書けることが哲学になっています。操作が自然に書ける関数型言語の核を基本にして、高度なモジュールシステムやオブジェクトの構造をより細かく推論する機能が自由に取捨選択できるのが特徴といえます。

エアバスはOCamlで飛んでいる!

OCamlは昔から大学や研究所でコンパイラ開発やプログラム生成・解析で広く使われてきています。こうした学術的な成功例には、定理証明支援系のCoq[注1]やマイクロソフトリサーチが開発したデバイスドライバ解析ツールSLAM[注2]などが挙げられます。しかし、最も印象的なのはエアバスの飛行機開発での利用でしょう。エアバスA340とA380では、飛行制御システムの開発はSCADE[注3]というOCamlで開発された形式手法のツール群を利用していて、飛行機の中で実行されているのコードの7割がこのツールによって生成されています。Cなどを使って直接書かれたコードについても、OCamlで書かれたAstrée[注4]という解析ツールによって実行時エラーの欠除が保障されています。今やOCamlなしにエアバスは飛べないと言っても過言ではないのです！

OCamlの利用は前述の得意分野に限定されていません。最近目立っているのは、金融とWeb開発での利用です。Lexifi社[注5]の先物記述言語やJaneStreet社[注6]の高頻度トレーディングシステムは異なる形で金融への応用を代表している

注1）The Coq Proof Assistant　http://coq.inria.fr/
注2）http://research.microsoft.com/en-us/projects/slam/
注3）http://www.esterel-technologies.com/products/scade-system/
http://www.esterel-technologies.com/success-stories/airbus/
注4）http://www.astree.ens.fr/
注5）http://www.lexifi.com/
注6）https://ocaml.janestreet.com/

といえます。Web言語のOPA[注7]やHaxe[注8]、Webフレームワーク Ocsigen/Eliom[注9]もOCamlで作られています。こういう新しい分野から新しいニーズと協力が集められたことがOCamlをより使いやすいものにしています。

OCamlエコシステムの充実

この数年、OCamlのコミュニティが非常に活性化されてきています。その象徴とも言えるのがパッケージマネージャOPAM[注10]の登場です。協力をスムーズにするためには、皆が開発したソフトウェアを簡単に導入できなければなりませんが、OPAMにより、それが可能になりました。OPAMの利用は急激に広がっていて、今ではOCamlで書かれたほとんどのソフトウェアがOPAM経由でインストールできるようになっています。新しいソフトウェアの公開、提供が非常に簡単になったため、新しく公開されるソフトウェアの数も増えています。

OPAMはOCamlProというOCamlのサポートを提供する会社が開発しています。アカデミアでも関数型プログラミング総合環境としてのOCamlシステムの普及開発を目指すOCaml Labsがケンブリッジ大学に設立され、精力的な活躍を始めました。

このようなユーザベースの広がりから、石橋を叩いて渡ることで有名なOCamlコンパイラ開発元であるINRIAも利用者が求める新しい機能を早く取り込むようになっています。最近では、安全性と型推論を壊さない形で、普通の値のように使える第一級モジュールや、プログラムの不変量を型にエンコードできるGADT（Generalized Algebraic Data Type：一般化代数的データ型）が追加されています。この好循環が今OCamlを「熱い言語」にしているのです。

OCaml入門（もしくは再履修）

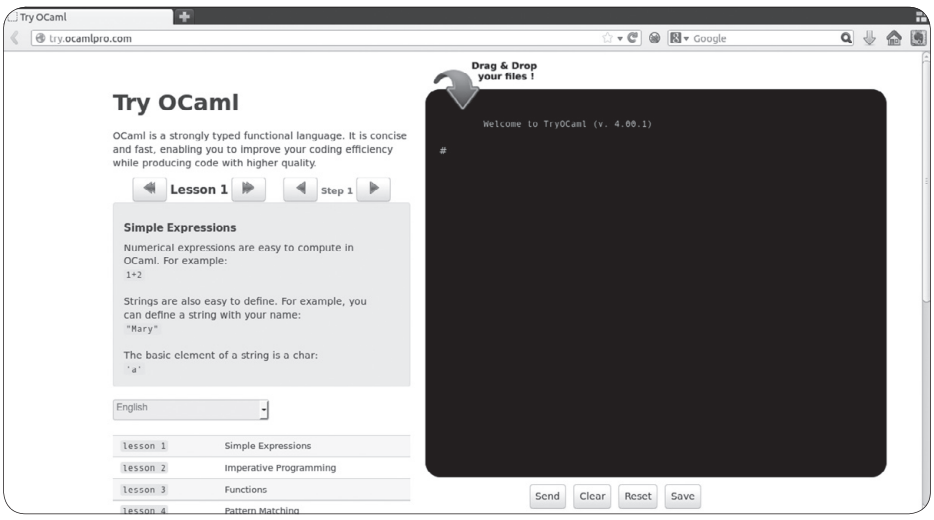
ではOCamlの雰囲気を感じてみましょう。え？　処理系をインストールするのが面倒くさい？——そんなあなたにピッタリなのがTry OCamlです。

Try OCaml

図1のTry OCamlは、OCamlの対話的処理系のブラウザ版で、後述するOCamlから

注7）http://opalang.org/
注8）http://haxe.org/
注9）http://ocsigen.org/
注10）http://opam.ocamlpro.com/

▼図1　Try OCaml（http://try.ocamlpro.com）



JavaScriptへのコンパイラjs_of_ocamlを使って実装されています。

ページ右上の黒い領域が入力／応答です。この領域をクリックしてから「1+2」と入力してEnterキーを打ってみると、「- : int = 3」と入力式の型であるintと計算結果の3が表示されます。ページ左側には、簡単な、そして未完成の ;-) チュートリアルがあって——英語ですが——説明を読みながら式を打ち込んでいくだけである程度OCamlの雰囲気がつかめるしくみになっています（このチュートリアル、ちょっと面白いのは入力画面と連動していることです。たとえばstep()と入力してみましょう。次の項目に進めます。マウスに手を伸ばしたくないという方はどうぞ）。

以降の機能紹介での例もTry OCamlでの動作を確認していますから、試しながら読んでみるといいでしょう。Try OCamlはEnterを押すといきなり実行されてしまうので、複数行にわたる入力はShift + Enterで改行してください。

（INRIAの）OCaml

「ふつうの」OCamlコンパイラは、OCamlの総本山であるフランスINRIAのWebサイト[注11]で配布されています。可搬性の高いバイトコードへコンパイルするocamlc、機械語にコンパイルするocamlopt、Try OCamlのような対話的コンパイラocamlなどが含まれています[注12]。

OCamlの機能紹介

紙幅の都合もあり、OCamlの主な特徴が現れているプログラムを1つ示して超特急で雰囲気をつかんでみましょう（図2）。題材は、二分木を定義して、葉っぱの数を数える関数を書く、です。

データ型定義（2〜5行目）と木の定義（7行目、10行目）

二分木の型（itree と呼びます）をtype宣言で定義します。ここでの二分木は、

- （データを保持しない）葉っぱ（Leafで表す）
- 整数を保持し、ふたつの部分木を持つノード（Nodeで表す）

から作られるとします。Nodeのof以下には、ノー

注11）http://caml.inria.fr/

注12）詳しいインストール方法は「http://ocaml.jp/インストール方法」に書かれています。

▼図2　二分木の定義itreeと葉の数を数える関数count_leaves

```
1  (* 二分木の型定義 *)
2  # type itree =
3      Leaf (* 葉っぱ *)
4    | Node of itree * int * itree (* ノード *)
5  ;;
6  type itree = ...
7  # let t1 = Node(Leaf, 1, Node(Leaf, 2, Leaf));;
8  val t1 : itree = ...
9
10 # let t2 = Node(Node(Leaf, 5, Leaf), 6, Node(Leaf, 7, Leaf));;
11 val t2 : itree = ...
12
13 # let rec count_leaves t =
14   match t with
15     Leaf -> 1
16   | Node(left, a, right) -> count_leaves left + count_leaves right;;
17 val count_leaves : itree -> int
18
19 # count_leaves t1;;
20 - : int = 3
21 # count_leaves t2;;
22 - : int = 4
```

※注　(* ... *)部分はコメント、処理系からの応答は*斜体*、行頭の＃は入力不要

ドを構成するデータの型が*で区切られて順番に書いてあります。二分木をいくつか作ってみたのが、7行目、10行目です。let t1 = ...は値の定義の構文で、右辺(の計算結果)にt1という名前を付けています。図3にt1、t2がどういう木を表しているかを示しました。

OCamlは、プログラムの実行前に型検査をするいわゆる静的型付き言語ですが、型推論という機能があり、変数や関数の型宣言をしなくても、処理系が補ってくれます。処理系からの応答のコロン(:)以下がt1、t2の型がitreeであることを示しています。

パターンマッチを使った関数定義(13〜16行目)

関数の定義もlet(再帰的な場合はrecをつけます)を使って行います。tが仮引数で=の後に関数の返値を計算する式が書いてあります。この定義では、t中の葉っぱの数は

- tが葉っぱならば、葉っぱの数は1
- tがノードならば、求める葉っぱの数は(左の部分木の葉っぱの数)＋(右の部分木の葉っぱの数)

で計算することを示しています。

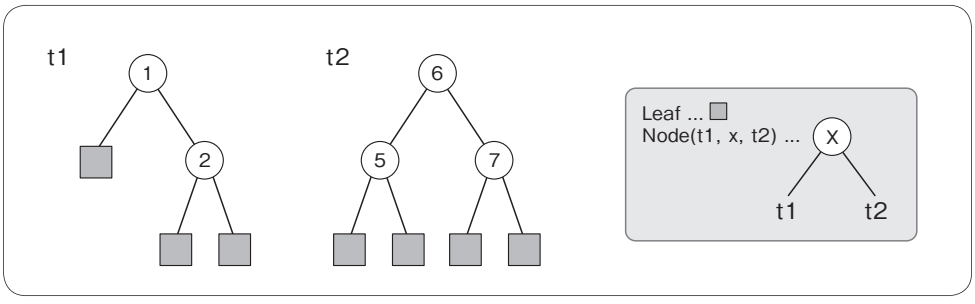
具体的には、match構文と呼ばれる場合分けとデータの取り出しを同時に行う機能を使って書かれています。match t withを「tが」、->を「ならば」、|を「または」と読むと直観的に読めるかもしれません。->の左側のLeafやNode〜はパターンと呼ばれ、たとえばNode(left,a,right)の場合、対象となるデータがNode〜という形をしているならば3つのデータにそれぞれleft、a、rightという名前をつけた上で->の右側の計算を行うことを意味しています。

これがパターンマッチと呼ばれる機能で、OCamlではなくてはならないものです。単にプログラムが簡潔に書けるだけでなく、OCaml処理系によって、パターンがtのとりうる値のすべての可能性を尽くしていることまでチェックしてくれますから、**場合分けの漏れなどによるバグも関数を定義した時点で見つかります**。また、上でもふれたように、プログラマはtの型や返値の型を書かなくても、処理系がcount_leavesはitreeを受け取ってintを返す関数であることを推論してくれています(17行目)。

count_leavesの呼び出し例は19行目と21行目になります。

同じように、ノードに格納された整数の和を計算する関数は**リスト1**のように定義できます。

▼図3　t1、t2の構造



▼リスト1　ノードに格納された整数和を求める関数例

```
# let rec sum_nodes t =
  match t with
    Leaf -> 0
  | Node(left,a,right) -> sum_nodes left + a + sum_nodes right;;
val sum_nodes : itree -> int
```

多相的データ型定義

図2の二分木の定義はノードに格納できるデータが整数に限られていましたが、型定義を少し変更するだけで(木全体で統一されている限り)さまざまなデータを格納できるようになります。それがリスト2の型定義です。

'aは型を動くパラメータ(C++のテンプレートの引数のようなものです)で、木に格納するデータの型を表しています。ですので、intと書いてあったところが'aになっています。

この定義のもとで、先ほどのt1を再定義してみましょう。

```
# let t1 = Node(Leaf, 1, Node(Leaf, 2, Leaf));;
val t1 : int tree = ...
```

ここでは'aが具体化されてint treeとなっている、つまり、t1が整数をノードに格納した二分木であることを、プログラマが何も言わなくても型推論によってわかったことに注目してください。要素の種類を変えるとそれに応じて型が変わります。

```
# let t3 = Node(Leaf, 'a', Leaf);;
val t3 : char tree = ...
```

また、この定義のもとでcount_leaves関数を再定義してみると、

```
# let rec count_leaves t = ...同上...;;
val count_leaves : 'a tree -> int = ...
```

となります。このcount_leavesの型は、任意の種類の木から整数への関数であることを意味しています。これは、count_leavesがノードのデータを使うことなく計算をしていることと対応しています。ですから、この関数はt1だけでなくt3にもそのまま使うことができます。

```
# count_leaves t3;;
 - : int = 2
```

一方、sum_nodesはノードに格納された数の和を計算しますから、木の種類が限定されて、

```
# let rec sum_nodes t = ...同上...;;
val sum_nodes : int tree -> int
```

と整数の木しか引数にとれないことが型から明らかです(OCamlでは+はintの足し算に限られます)。

ですから、sum_nodesをt3で呼び出そうとしても「t3はchar treeなのに、sum_nodesはint treeを欲しがっているよ」という型検査時のエラーになります。

```
# sum_nodes t3;;
File "", line 1, characters 10-12:
Error:This expression has type char tree
    but an expression was expected of type
int tree
```

count_leavesのような、定義の一部の型をいろいろ変えられたり、関数がいろいろな型の引数を取れることを「多相性」といいます。

▼リスト2 型定義

```
# type 'a tree =
    Leaf                          (* 葉っぱ *)
  | Node of 'a tree * 'a * 'a tree (* ノード *)
;;
```

▼リスト3 tree_map関数の例

```
# let rec tree_map f t =
  match t with
    Leaf -> Leaf (* 葉っぱはそのまま *)
  | Node(left, a, right) -> Node(tree_map f left, f a, tree_map f right)
    (* a は f a に、左右の部分木は再帰的に変換 *)
;;
val tree_map : ('a -> 'b) -> 'a tree -> 'b tree
```

OCamlでは多相性がありながら型推論が可能なので、簡潔さを保ちつつ再利用性の高いコードを書くことが可能になります。

高階関数

OCamlは関数型言語ですので、ほかの関数型言語と同じく、関数を引数として関数に渡したりするようなことも**当然**できます。

リスト3は、関数**f**に従って二分木**t**のノードのデータを変換した木を返す**tree_map**関数です。こういった関数を受け取る関数を高階関数といいます。

この型は、関数**f**(**'a -> 'b** 型)と、その引数型(**'a**)を要素とする二分木(**'a tree**) **t**を引数として渡すと、**f**の返り値型を要素とする二分木が返ってくることを示してみます。**tree_map**を使って**t1**の各ノードを1増やしてみましょう。

```
# tree_map (fun x -> x + 1) t1;;
- : int tree = Node(Leaf, 2, Node(Leaf, 3, Leaf))
```

fとして渡している**fun x -> x + 1**は匿名関数と呼ばれるもので「**x**を引数として**x + 1**を返す(無名の)関数」を表しています。

参照

OCamlの変数は、Javaのfinal変数のように基本的には宣言時に初期値を与えたら中身を書換えることはできませんが、参照という機能を使えば書き換え可能な変数の真似事をすることができて、**for**文、**while**文なども使えます。以下は、1から10までの和を計算して**sum**に格納するプログラムです(応答は省略しています)。

```
# let sum = ref 0;;
# for i = 1 to 10 do sum := !sum + i done;;
# !sum;;
```

ref 0が参照と呼ばれる「初期値0の入った箱」を作るものです。**for**はなんとなくわかるとは思いますが**do**と**done**ではさまれた部分を、**i**を増やしながら繰り返していきます。**:=**が箱への代入、**!**が箱からの読み出しです。読み出し時にわざわざ**!**をつけるのが面倒ですが、OCamlプログラマは**!**を書くたびに「面倒くさい……こんなことなら参照を使わないプログラムに書き換えた方がましだ」と思います(ちょっとだけウソです)。

モジュールシステム

OCamlには先進的なモジュールシステムがあり、OCamlの魅力の1つなのですが、すべてを紹介するのは無理ですので基本的な部分のみふれたいと思います。モジュール上の関数であるファンクタなどの上級者機能については、コラムで紹介する書籍などをあたってください。

モジュールはインターフェースと実装の記述が分離していて、さらにモジュールが1ファイル(正確にはインターフェースファイルと実装ファイルがひとつずつ)に対応しています。たとえば、ライブラリの文字列モジュールは、**string.mli**という文字列関数の型(と機能に関するコメント)だけが書かれたファイルと**string.ml**という関数の定義が書かれたファイルで構成されています(**図4**)。このモジュールを使う人は**mli**ファイルだけを見てプログラムを書きます。自分のプログラムのコンパイルは、依存するモジュールの**mli**ファイルだけあればできるので、極端な話、(開発途中段階で)実装されていないモジュールを使っていてもオブジェクトファイルへのコンパイルまではできることになります。OCamlプログラムは型検査に通すまでがデバッグという感じですから、実装が一部未完成でもデバッグができる、というのは、複数人で分担して開発を行うときには大変ありがたいです。

OCamlのメリット

完全な型推論

OCamlプログラムで型を書くのは基本的に型の定義だけです。後は省略したければ一切省略できます。「完全」というのは一切省略できる、というくらいの意味です(正確には、モジュールのインターフェースや一部の機能を使うために型の記述が多少求められますが、書かなければな

らない場所がきちんと決まっているのがよいです。一方、ScalaやHaskellなど他の言語の型推論は完全でなく、高階関数や型クラスを使うときに型の記述が必要な場合があります)。

多相性

上の例でも見たように、多相性を利用することで1つの型や関数の定義をさまざまな場面で使うことができます。

厳密な型検査と型安全性

OCamlは型安全性といって「いったんコンパイラの型検査に通れば値の種類に起因するバグ(たとえば文字列で割り算をするとか)は発生しない」ということが保障されています。また、場合分けの漏れなども、型定義とパターンマッチをうまく使うことで防ぐことができるため、(応用分野にもよりますが)コンパイラの型検査に通れば、その時点で「かなり正しい」プログラムが得られる、というのが筆者の実感です。

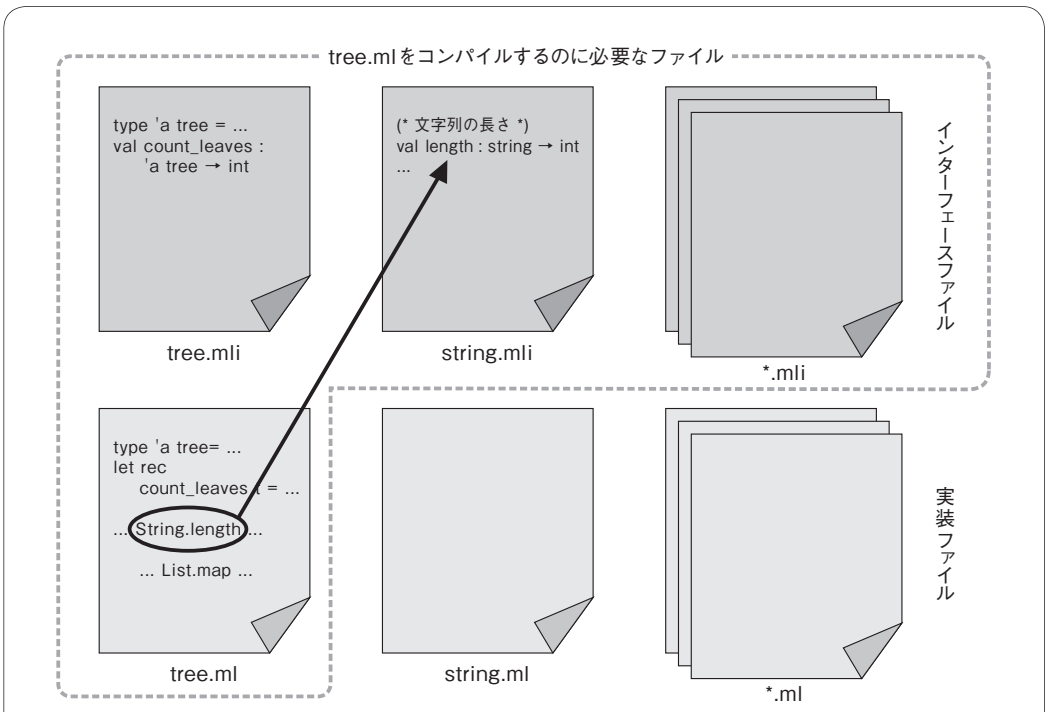
モジュール機能の紹介でもふれたように型検査に通すまでがデバッグ作業になっているとも言えます。プログラムを走らせることなくデバッグをするわけですから、慣れないうちは戸惑う人も多いようですが、その分プログラムを書き終えるころには、自分のプログラムが何をするかの理解が深まっています。

欲しい機能はだいたいある

紙幅の都合上紹介できませんが、機能はかなりそろっています。

- オブジェクト、クラス
- 多相バリアント
- ラベル付き引数、オプション引数
- GADT
- スレッド
- マーシャリング(シリアライズ)
- 構文拡張(camlp4)

▼図4 モジュールシステムの例



OCamlにおける分割コンパイル。実装(.mlファイル)をコンパイルするためには依存するモジュールのインターフェース(.mli)のみが必須で実装は必要ない。

・実行時コード生成（BER MetaOCaml）

実行時に型情報が欲しかったり、Haskellの型クラスが欲しくなったり、スレッドがマルチコアで並列実行できなかったりするのに悔しい思いをすることがないとは言えませんが満足感は高いです（マルチコアを有効活用する実装は、OCamlLabsで現在鋭意開発中とのことです。乞御期待）。

ポータビリティ

Web（JavaScript）やスマホ（iOS、Android）を含めて、ほぼすべての環境で動作し、クロスコンパイルも可能です。

プログラムが何をするのか見える

コンパイラが素直な作りをしていて、コードを勝手に生成したりすることはなく、最適化はわりと自明なものに限られています。これはダサいようで実は大きなメリットです。ある意味「プログラムは書いたとおりに動く」ので、ソースコードを変更した際の性能変化の予想がたてやすいです。

真のOCaml riderへ向けて

OCamlプログラミングが複雑になってくると対話型コンパイラだけを使っていたり、標準システム一式をインストールしただけの状態では開発がたいへんになってきます。そこで、OCamlプログラミング初級を脱出し、中、上級"OCaml rider"へ向かうための必須ツールの数々を紹介します。

コマンドラインコンパイラ ocamloptとOCamlFind

対話式コンパイラでOCaml感をだいたいつかんだら、ひとまず対話式は卒業してコマンドラインコンパイラocamloptに移行しましょう。ocamloptは機械語を生成するため実行速度も高速です。さらに、ライブラリを使用しはじめるとocamloptもOCamlFindでラップして使用するのが普通です。OCamlFindは面倒なライブラリのインクルードパスやライブラリファイルの列挙作業を自動化してくれます。たとえば

```
ocamlfind ocamlopt -packages lwt -linkpkg -o xxx.exe xxx.ml
```

とするとlwtパッケージに必要なスイッチをocamloptコマンドに自動的に付け加えてコンパイルを行います。

なお、バイトコードコンパイラocamlcはocamloptの生成する機械語より速度が劣るので現在ではあまり使いません。

ビルドシステム

複数のソースファイルを扱い始めると必要になるビルドシステム。古き良きMakefileを使うのも良いのですが、OCamlに特化したビルドシステムを使うとルール記述がとても楽になります。

OCamlBuild

OCaml標準システムに同梱のOCamlに特化したocamlbuildコマンドを使う方法。公式周辺ツールのビルドにも使われています。

OMake

OMake[注13]はOCamlのための強力なマクロが備わっているMake似のツール。百万行を超えるOCamlプログラムのビルド実績もあります。

OPAM：GitHubを利用したパッケージシステム

2012～2013年のOCaml界隈で最も熱かったものと言えばOPAMの登場でしょう。OPAMはOCamlの各種ソフトウェアをソースパッケージとして提供する新しいシステムで、

```
opam install <パッケージ>
```

とすると依存するソースコードをネットからダウンロードし、自動的にインストールを行います。現在すでに500を超えるライブラリ、アプリケーションがパッケージとしてGitHub[注14]上で管理されており、インストール時の問題点などもそこ

注13）http://omake.metaprl.org/
注14）https://github.com/OCamlPro/opam-repository

で一元的に議論されています。後述のツールもすべてopam installコマンド一発でインストール可能です。

WindowsではOPAMは現在移植中で少し待つ必要があるようです。

プログラミング環境強化

OCaml 4よりコンパイラ内部APIが登場しました。これによりツール開発がとても楽になり、サードパーティの開発したOCamlプログラミング環境強化ツールも充実してきました。現時点での注目株を挙げておきます。

強化対話型コンパイラ「utop」

生のocamlコマンドを試してみてイライラした方はutopをお勧めします。ラインエディタや補完機能が付いている強化版対話型コンパイラです。

TypeRexやMerlinでエディタをIDE化

TypeRexやMerlinはEmacsとVimの機能を拡張し、OCamlプログラミングの効率を飛躍的に高めるツールです。ソースコード上の部分式の型情報の表示や定義へのジャンプ、文脈を意識した補完機能などが提供されています。

API検索エンジン「OCamlOScope」

OCamlOScope[注15]は検索文字列に似た型や名前を持つAPIを探し出すWebサービスです。この型を持つ関数は何という名前だったか、どんなライブラリが提供しているのか、といった「逆引き」的疑問解決に威力を発揮します。

注目ライブラリ、フレームワーク

強化標準ライブラリ「Batteries/Core」

OCamlの「標準ライブラリ（stdlib）」は貧弱だ、というのは有名です。これにはさまざまな理由があるのですが、コミュニティレベルで「強化標準ライブラリ」を作っていくという動きに今は落ち着きました。

有名な強化ライブラリにはOCaml Batteries Included（略称：Batteries）とJane Street Core（略称：Core）があります。BatteriesはCoreと比べるとstdlibの雰囲気をよく残しているので、stdlibからの移行は比較的簡単です。一方CoreはOCamlを使用して金融取引を行っているJane Streetが内製していたものです。開発に膨大な資源が投資されているので性能、信頼性ともに非常に高いのですが、stdlibと非互換な慣習を導入しているので良くも悪くもJane Street流OCamlプログラミングを行うことになります。

非同期演算ライブラリ「Lwt/Async」

OCamlは現時点ではマルチコアを簡単に有効利用する方法がなく並列（parallel）計算は不得意とされています。しかしこれはOCamlでは並行（concurrent）計算が難しい、ということではありません。非同期並行計算ライブラリLwtとAsyncを使うと、無名関数を駆使した関数型言語らしい、並行しつつも関連しあった複雑な仕事を簡潔に書くことができます。

OCamlのWebフレームワーク「Ocsigen/Eliomとjs_of_ocaml」

EliomはOCamlのWebフレームワーク。Eliomでは専用Webサーバ Ocsigenと、Webブラウザの間でOCamlで書いた共通のコードを走らせることができます。ブラウザ上ではOCamlのコードはjs_of_ocamlコンパイラによってJavaScriptに変換されるため特殊なブラウザ拡張も必要ありません。Eliomでは、HTML、CSS、SQLにいたるすべてのWeb開発もOCamlの強力な型システムのもとで完結してしまいます。

js_of_ocamlは単体だけでもOCamlプログラムをJavaScriptに変換するコンパイラとしてとても面白いツールです。Try OCamlもこのjs_of_ocamlでOCamlコンパイラ自身をJavaScriptに変換したものです。

注15）http://ocamloscope.herokuapp.com/

終わりに

以上、駆け足でしたがOCamlの特徴を紹介しました。Try OCamlでゆるゆると初体験でもよいですし、一気にCoreで最前線OCamlに挑戦してもよいでしょう。ぜひOCamlのコミュニティに参加し、熱気にふれてください。SD

Column 「OCamlをもっと知りたいあなたに」

日本語で書かれたOCaml情報サイト（http://ocaml.jp/）にアクセスください。その他の日本語によるOCaml情報へのリンクも充実しています。

書籍はこれまでに次の3冊が出版されています。

- 『プログラミングの基礎』
  浅井健一（著）、サイエンス社
- 『プログラミング in OCaml—関数型プログラミングの基礎からGUI構築まで』
  五十嵐 淳（著）、技術評論社
- 『入門OCaml—プログラミング基礎と実践理解』
  OCaml-Nagoya（著）、毎日コミュニケーションズ

『プログラミングの基礎』はOCamlを使った（静的型付き）関数型プログラミングの入門書で、OCamlの機能を網羅しているわけではありませんが、たいへんわかりやすくよい本だと思います。残りの2冊については、残念ながら現在、新品の入手が困難です。手前味噌で恐縮ですが『プログラミング in OCaml』は、モジュール、オブジェクト、多相バリアント、ラベル付き引数あたりまでふれていて、機能の網羅度は高めです。現在は、電子版が技評オンラインで（冊子版よりもずっと安く!（笑））入手できます。『入門OCaml』は、標準ライブラリの関数やMySQLとの連携についても書かれていますので、応用面に興味のある方にも役に立つと思います。

英語の書籍は（親戚のStandard MLの本は何冊かあるものの）、実は最近まで良いものが少なかったですが、最近になって状況が改善されつつあります。

- 『OCaml from the Very Beginning』
  John Whitington. Coherent Press. 2013.

この本は、筆者（五十嵐）は未確認ですが、タイトルのとおりプログラミングの初心者用にていねいに書かれているという評判を聞きます。ただし、オブジェクト指向などの機能の一部は扱われていないようです。

- 『Introduction to Objective Caml』
  Jason Hickey
  http://files.metaprl.org/doc/ocaml-book.pdf

これも、OCamlプログラミング入門で、中核となる部分だけではなくモジュール・オブジェクトあたりまでカバーしています。

- 『Unix system programming in OCaml』
  Xavier Leroy and Didier Rémy
  http://ocamlunix.forge.ocamlcore.org/

通常はC言語で行われるUnixシステムプログラミングをOCamlコンパイラ開発者たちがOCamlを使って解説した教科書です。ポインタやメモリ管理などC言語のはまりどころにとらわれずUnixプログラミングを習得できます。

- 『Real World OCaml: Functional programming for the masses』
  Yaron Minsky, Anil Madhavapeddy, Jason Hickey
  https://realworldocaml.org/

OCamlをまったく知らない人を対象としながらも、そこから現在のOCamlの最前線まで触れることを目指した非常に意欲的な形の入門書です。そのため、入門時点から、標準のOCaml環境ではなく、強化対話型コンパイラutopや強化標準ライブラリCoreなどをOPAMパッケージシステムでインストールした状態から始まります。OCamlランタイムの解説や、新しい外部関数呼び出しインターフェースCtypesについて触れられているなど、OCamlのベテランプログラマにも参考になる内容です。

コマンド作りで知るHaskell

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

Haskellは、誰か1人が作り上げたのではなく、学者が委員会を組織して仕様を決めたユニークな言語です。その仕様は数学に裏打ちされた厳格なもので、一見すると、他の頑固な関数型言語と比較しても、実用性という観点を受け付けない頑固者に見えます。しかし、厳格な理論は合理性をもたらし、合理性は、最終的には実用性をもたらします。本稿では、実際に仕事をするコマンドを作ることで、頑固者Haskellの内面を覗いてみることにします。

はじめに

こんにちは、産業技術大学院大学の上田です。と言うと、筆者をご存じの方だと「シェルだ逃げろ」と思うかもしれません。確かに「シェルスクリプトは関数型だ」などと不規則発言を繰り返しているのでそう思われてもしかたないのですが、今回はHaskellの話です。

いわゆる「こわいひと」がたくさんいるHaskell魔界の一番底辺にいる筆者が何でというところですが、実は現在、USP研究所[注1]で使っているコマンドのフリー版「Open usp Tukubai[注2]」をHaskellで書き直す1人プロジェクトを行っています。

本特集は初心者向けということで、Haskellのおもしろいところをなるべく多く紹介したほうが良いのですが、一方でHello Worldみたいなコードばかり並べてもつまらず、それをどうやって実用するのか想像できません。その点、コマンドは短いプログラムでも一人前に仕事をしますし、オプション解析、データの読み込み、データの加工、出力と4つの機能をこなす中でHaskellを代表する特徴がいろいろ出てきます。そこで、コマンドのコードを読むことでHaskellの特徴を洗い出すのがおもしろいのではないかと考えたしだいです。

Haskellを学習するにはそれなりの順序があり、おそらく、本稿はその順序にはなっていません。本稿は読み物です。「Haskellのコードや考え方とはどんなものか」を力まずに眺めていただければと存じます。

Haskellって何だ？

Haskellの生い立ちは冒頭に書いたとおりです。詳しい経緯については http://www.haskell.org/onlinereport/preface-jfp.html にありますが、関数型言語の乱立に歯止めをかけ、フリーかつきっちりとした仕様の純粋関数型言語を作ろうということで、1987年に最初の集まりがあったようです。名前はHaskell B. Curryという数学者からとられましたが、1982年にすでに他界しており直接開発に携わったわけではありません。

学者の頭の中

こういう生い立ちですので、Haskellを一言で表すと「学者の頭の中」と言ってしまってもいいかもしれません。数学者、物理学者はもちろん、あるいは筆者のように専門不明学者でさえ、研究には数学を使います。その、学者の数学には「関数」というものがあります。

```
y = f(x)
```

注1）https://www.usp-lab.com

注2）https://github.com/usp-engineers-community/Open-usp-Tukubai

というやつです。ただし、難しい数学をやっている人は、

```
f: x ↦ y
```

と書きます。

どちらの書き方でも同じことですが、ここでのfは、xという入力をyに変えて出力するものを表します。腕組みしたままうなっている学者がいたら、**図1**のような、「何かを入れたら何かが出てくる」ものが頭の中に一杯なのです。

「不純な関数」とは?

さて、研究[注3]で使っている数式をプログラムして何か計算しようとすると、普通の言語(ここでは手続き型の言語の意)を使うとちょっと困ったことが起こります。たとえばC言語の「関数」の宣言を見ると、

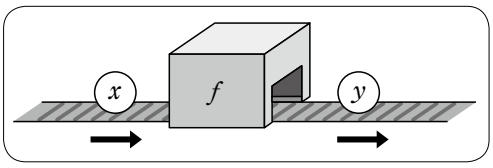
```
int f(char ch);
```

というように、さも「1文字入力して整数を出力」するように見えますが、C言語の関数は、その中でグローバル変数やファイルを好き勝手にいじれてしまいます。**図2**のように、入出力だけのはずの関数にあろうことか横に手が生えたようなものになっています。こういうものは「副作用」と呼ばれます。数学の関数にはないものです。

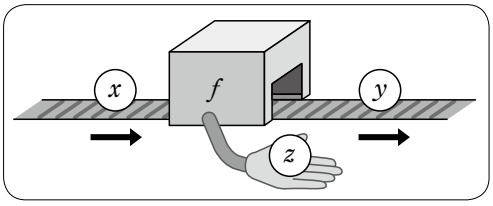
何か融通が利いて良いような気もしますが、副作用は学者の頭の中では数学の世界を破綻させる悪魔になってしまいます。実際C言語の「関数」は、数学で言うところの「関数」とは違うものなのです。

注3) あくまで研究というのは一例で、Haskellは研究以外にも使えます。

▼図1 関数(xを入れたらyが出てくる「箱」)



▼図2 副作用のある純粋でない関数



Haskellは純粋

学者がこういう「関数もどき」を排除しようとするのは自然なことです。実は筆者も学生のときに、同じことができないかとC++のテンプレートで頑張ってみた経験があります。もちろん、情報系の学生でもなく、何のスキルもなかったので頓挫しました。それゆえに最近になってやっとHaskellを知ったときの衝撃は計りしれませんでした。

一方、慣れていないと、純粋な関数の世界には不自由さを感じます。たとえば「変数に値を代入する」という概念は、入出力だけ扱う関数の世界にはありません。何かをHaskellのプログラムに放り込んだら最後、数珠つなぎになった関数を通って何か出てくるだけで、「中間のもの」がないのです。

これから読んでいくコードは、そういうわけのわからないものです。ただ、学者のエゴを押し通したことで、かえって言語の仕様がスッキリしました。そのおかげで、現在のHaskellは「わかってしまえばすごい道具」と言えるものになっています。

コマンド読み開始!

さてコードを読んでいきましょう。Open usp Tukubaiのselfコマンド[注4]のミニチュアを作っていきますので、みなさんはダラダラと読んでください。

selfの動作は**図3**のようなものです。selfの後

注4) https://uec.usp-lab.com/TUKUBAI_MAN/CGI/TUKUBAI_MAN.CGI?POMPA=MAN1_self

ろにオプションで出力したいフィールド(列)を指定します。最初の指定1は1列目を出力しろでOKです。次の3.3は「3列目の3文字目以降」を出力せよということです。その次の2.3は「2列目の3文字目以降」ですが、selfの「3文字目」は半角文字数にして3文字目という定義で、全角では2文字目ということになります。

▼図3　selfの動作

```
$ cat hoge
aaa 濱田 ﾊﾏﾀﾞ      1列目
bbb 鎌田 ｶﾏﾀ        ↓
$ cat hoge | self 1 3.3 2.3 ←2列目の3文字目以降
aaa ﾀﾞ 田             ↑
bbb ﾀ 田              3列目の3文字目以降
```

▼図4　GHCのインストール

```
$ sudo apt-get install haskell-platform
$ ghc --version
The Glorious Glasgow Haskell Compilation System, version 7.6.3
```

▼図5　HaskellのインタプリタGHCiの使い方

```
$ ghci
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
(略)
Prelude> Data.List.sort [1,4,3] #コードを試す
[1,3,4]
Prelude>        #Ctrl+dでインタプリタを抜ける
Leaving GHCi.
```

▼リスト1　first.hs

```
1  import qualified Data.ByteString.Lazy.Char8 as B
2
3  main :: IO ()
4  main = B.getContents >>= print . B.lines
```

※行頭の行番号は説明上入れたものです。プログラムリストには含まれません。

▼図6　first.hsのコンパイル

```
$ ghc first.hs
[1 of 1] Compiling Main             ( first.hs, first.o )
Linking first ...
```

▼図7　./firstの実行

```
$ cat hoge | ./first
["aaa ¥230¥191¥177¥231¥148¥176 ¥239¥190¥138¥239¥190¥143¥239¥
190¥128¥239¥190¥158","bbb ¥233¥142¥140¥231¥148¥176 ¥239¥189¥
182¥239¥190¥143¥239¥190¥128"]
```

環境について

本稿で試したコードの動作確認には、Ubuntu 13.10を使いました。Ubuntu 13.10の場合、**図4**のようにapt-get一発で環境が整います。本稿ではコードはすべてコンパイルして実行します。コンパイラにはGHC(Glasgow Haskell Compilar)を使います。

インストールした環境にはGHCだけでなく、インタプリタGHCiが付いてきます。GHCiは、**図5**のように使います。端末でちょっとコードを試したいときに便利です。本稿では、型を調べるときにGHCiを使用します。

標準入出力(いきなり最大の山場)

最初に提示するのは**リスト1**のようなファイルです。本稿で最初のHaskellのコードですが、動作は標準入力から文字列を読み込んで、行に分解して出力するというものです。

もうすでに難しい概念がたくさんあり、Haskellを知っている人からすると「そこから始めるんかい!」とツッコミが入りそうですが、本稿は読み物ですのでと言いわけをしておきます。あまり深く考えないでください。

説明は後回しにしてコンパイルして動かしてみましょう(**図6**)。

これでfirstというファイルができるので、これを実行します(**図7**)。

出てくる数字はUTF-8の文字のバイナリですのでびっくりしないでください。この出力を見ると、[x,y]という形式になっていますが、これは「**リスト**」というもので、ここでは行が2つ入ったリストが標準出力に出さ

れたという理解で大丈夫です。リストについては、後述の型の話と合わせてコラムに例を加えました。

ここで**リスト1**の解説をしておきましょう。Haskellのコードを読み解くには、「**型**」を手がかりにします。CやC++だとintやdoubleやcharがありますが、その型です。Haskellでは多種多様な型が登場しますが、関数に値を渡すときや、ある関数の返す値を別の関数に渡すときは、C/C++と同様、いやそれ以上に厳密に型がピッタリ一致している必要があります。

1、3行目の説明は保留しておいて、4行目がmainという関数の「定義」です。

```
main = B.getContents >>= print . B.lines
```

これを大雑把に意訳すると「mainというものは、B.getContentsの出力を「print . B.lines」という関数に入力した出力」という意味になります。また、「print . B.lines」は、printとB.linesという2個の関数が組み合わさったものです。要はmainという関数は、3個の関数をくっつけたものと言えます。そして、Haskellではこのような「**関数の定義**」をダラダラ並べてコーディングします。プログラムは全部定義でできています。代入ではなく定義です。

B.＜関数名＞のBというのは、Data.ByteString.Lazy.Char8の別名です。1行目は、Data.ByteString.Lazy.Char8という「**モジュール**(関数の定義の詰まったもの)」を読み込んでBという別名で定義を使いたいという意味になります。ちなみに、Haskellでは関数の名前の一文字目を小文字で書き、型などの場合は大文字で書くというルールがあります。

型を合わせて関数をくっつける

もうちょっと細かいところを見て型を理解しましょう。「print . B.lines」は2つの関数がくっついたもので、「**合成関数**」と言います。数学のg(f(x))という式を想像してください。この場合、先にf(x)が計算され、その値をg(x)に入力して計算しています。これをHaskellでは、「(g . f) x」と書きます。数学でもg○f(x)と書きます。

このように関数を合成するには、fの返す型とgの受け入れる型が一致している必要があります。ghciで型を調べると、**図8**のように出てきました。

「:t hoge」で、関数hogeの型を表示せよという命令になります。「:t」は「:type」と書いてもかまいません。型は、GHCiのほか、Hackage[注5]というサイトでも調査できます。

図8の4行目の「B.lines :: B.ByteString -> [B.ByteString]」という表記は、「::」の左側に関数の名前を書いて、右側に入力と出力の型を書いたものです。この後のソースコード中にも、関数の型を指定するためにこの記法が出てきます。

入力が複数あるときは、

```
<関数名> :: <引数> -> <引数> -> ... -> <出力>
```

と書きます。出力は1つしか許されません。

次に、printについてはもうちょっとややこしくて、とりあえず「Show a =>」を無視すると、「a -> IO ()」となっています。このaですが、実はいろんな型になれるので仮にaと書いているだけです。「print . B.lines」と書いた場合、aはB.linesの出力の型[B.ByteString]に化けます。化けるのは、コードをコンパイルするときです。インタプリタであるGHCiの場合はコンパイルしませんが、次のように合成関数の型を認識します。

```
Prelude B> :t print . B.lines
print . B.lines :: B.ByteString -> IO ()
```

Haskellは、このように不定な型をコンパイル

注5) http://hackage.haskell.org/

▼図8　ghciで型を調査

```
$ ghci
Prelude> import qualified Data.ByteString.Char8 as B
Prelude B> :t B.lines
B.lines :: B.ByteString -> [B.ByteString]
Prelude B> :t print
print :: Show a => a -> IO ()
```

時に合わせる「**型推論**」というしくみを持っています。型推論のおかげで、printとB.linesという別々の関数が「B.ByteString -> IO ()」という型を持つ1つの関数となりました。めでたしめでたし。

ところでaはどんな型にでも化けることができるわけではなく、制限があります。「Show a =>」がその制限を表しています。この場合のShowは「型クラス」を表しています。ただ、ここで言う「クラス」はオブジェクト指向言語のソレではなく、演算するときのルールが同じ型のグループという意味です。

Haskellを書いていると何度となくコンパイラに「型が合ってない」と叱られます。コードを書いているときはこのエラーにイライラするのですが、結局型があってないものを書いているということはロジックの整合性がとれていないということですので、先に教えてくれるだけ親切というものです。

このように型はHaskellを支配する重要な概念です。**リスト1**のような実践的なコードだと説明に限界があるので、本章末のコラムにGHCiとリストを使って少し例を示しておきます。

モナドで純粋性を保つ

今度は**リスト1**の4行目のB.getContentsを見てみましょう。

```
main = B.getContents >>= print . B.lines
```

この関数の型は、IO B.ByteStringです。IO B.ByteStringは出力の型です。あれ？ なんか変です。

1）B.getContentsの入力は？
2）「print . B.lines」の入力は「IO B.ByteString」ではなくB.ByteString

文字だけで説明するのが到底無理ですので、**図9**のような下手な絵を描いてみました。①～④は、mainで行う文字列の加工の流れを示したものです。

まず①ですが、これはgetContentsが標準入力という「外」から文字列を入手している様子です。標準入力は、関数getContentsの入力ではありません。Haskellの中での入出力は、Haskellの中で完結していて外とはいっさい関係ありません。外とやりとりするのは「副作用」です。

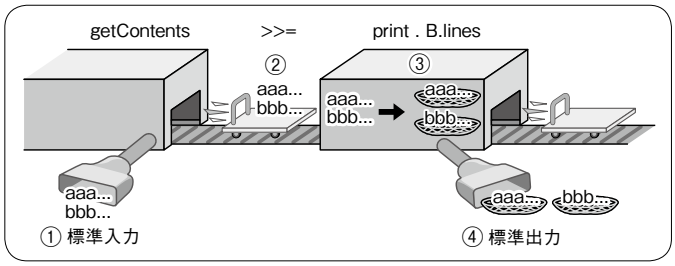
じゃあどうやって標準入力を読むのでしょう？

getContentsの型は「IO B.ByteString」ですが、これは、「標準入力を読んでB.ByteString型にするというIOに関する行為（アクション）」を出力するという意味になります。「一連の副作用となりそうなものをひっくるめてアクションにして出力しているので、これは副作用にならずに済む」という、普通の人にはどう考えても屁理屈なのに数学的には正しいという、驚異のテクノロジでHaskellは純粋性を保っています。

使うぶんにはそこまで考えなくても、B.ByteString型の文字列が、IOという名前の台車に乗っているという解釈で十分だと筆者は考えます。要は「不純な文字列」と関数の世界を台車で分けているわけです。Haskellにはこのような台車「**IOモナド**」というものがあります。IOモナドは、副作用である標準入出力の操作を純粋性を持って扱えるようにデザインされたしくみです。モナドはほかにもMaybe、Etherなどいろいろ種類があります。言うと混乱するだけですが、実はリストもモナドでできており、Haskellのコードには陰に陽にモナドが出現します。

我々はモナドの恩恵を受ける代わりに、モナドを操作しなければ

▼図9　mainで行う文字列の加工の流れ



いけません。それで、③にいきますが、「print . B.lines」である②の文字列を受け入れるには、台車を取って文字列だけにしなければなりません。これをやっているのが「>>=」です。「>>=」が台車から文字列を降ろし、「print ...」に入力します。生身の文字列は、もうモナドを気にせず関数に入力できるようになります。B.linesが行に分割し、printにその出力が入ります。printの出力の型は「IO ()」で再びアクションです。このアクションが出力されたとき、関数の世界とはあずかり知らぬところで、標準出力から処理結果が出力されます。()というのは空という意味です。「>>=」は、右側の関数の出力がきちんと台車つきでないと使えないので、この空の台車が必要になります。

型を作る

さて、少し書き進めて**リスト2**のようになりました。4行目に「data Field = ……」とありますが、これはFieldという型を新たに作るという意味です。また、この1行で「Fieldという型はFullF Intか、SubF Int Intのどちらかである」という意味になります[注6]。この2つの型は、たとえばself 2 3.3などと書いたときの、2(フィールドだけ指定)や3.3(文字列の切断も指定)にそれぞれ対応します。

ここではコマンドのオプションの読み込みは実装せず、7行目で[FullF 1,SubF 2 3,SubF 3 3]というオプションのリストをハードコーディングして先に進みます。

リストには、同じ型のものしか入れることができないのですが、これで2や3.3というオプションを、どちらもFieldとしてリストに入れることができます。Haskellのこのような型のしくみは、「**代数的データ型**」というタイプのもので、硬派なHaskellが見せる唯一(?)の柔軟な部分で、超強力な部分です。

木構造なども、

```
data Node = Leaf Int | Branch [Node]
```

などと左辺の定義を右辺の定義に再帰的に用いることで、ポインタなしで簡単に表現できます。

関数の書き方あれこれ

次に9、10行目のslfにいきましょう。やっとまともな関数が紹介できます(やれやれ)。

9行目はslfの型を指定したもので、「[Field], B.ByteStringを入力として受け付けてB.ByteStringを返す」という意味になります。「::」は型宣言、「->」で区切られた一番右が戻り値で、その前は入力です。2つの入力があるのがわかります。

```
slf :: [Field] -> B.ByteString -> B.ByteString
```

10行目はslfの定義で、左辺のfsとcsは9行目で型宣言されています。fsはそれぞれフィールドのリスト、csは標準入力からの文字列になります。

```
slf fs cs = B.unlines ➡
( map (slfLn fs) ➡
(B.lines cs) )
```

右辺を見てみましょう。まず、「B.lines :: B.ByteString -> [B.ByteString]」は、先ほども出てきたように文字列を行で分割します。「slfLn fs」は、12行目に型の指定がありますが、1行を読み込んでフィールドの指定に合わせた行に

注6) FullFやSubFは、型コンストラクタという関数で、FullFは1個、SubFは2個の整数を引数にとり、どちらもField型の値を返します。

▼リスト2　少し書き進める(second.hs)

```
 1  import Data.Char
 2  import qualified Data.ByteString.Lazy.Char8 as B
 3
 4  data Field = FullF Int | SubF Int Int
 5
 6  main :: IO ()
 7  main = B.getContents >>= B.putStr . slf [FullF 1,SubF 2 3,➡
SubF 3 3]
 8
 9  slf :: [Field] -> B.ByteString -> B.ByteString
10  slf fs cs = B.unlines ( map (slfLn fs) (B.lines cs) )
11
12  slfLn :: [Field] -> B.ByteString -> B.ByteString
13  slfLn = undefined
```

変換して返す関数のつもりで型を定義しました。ただ、まだ書いていないので13行目でundefinedとしています。これでコンパイルは通るので、型が合っているかどうかを調べることができます。ただし、プログラムを実行するとエラーが返ります。

それで、もう一度10行目を見てみましょう。mapという関数の型は、

```
map :: (a -> b) -> [a] -> [b]
```

です。これは型[a]のリストの要素ひとつひとつに、「a -> b」という型の関数を適用し、それぞれの出力を型[b]のリストで返すという関数です。このように、Haskellでは関数を入力に指定することがあります。mapのように関数を引数に取る関数のことは「**高階関数**」と呼びます。

もう1つ、mapの引数になった「slfLn fs」ですが、これは2つ必要なslfLnの引数のうち、1つしか指定されていません。この中途半端なものは何か？　実はこれも関数で、型は、

```
slfLn fs :: B.ByteString -> B.ByteString
```

となります。slfLn fsに入力されるのはB.linesの出力したB.ByteStringのリストのひとつひとつですから、つじつまが合っています。

このように引数の一部を関数に適用して新しい関数を得ることを、「**部分適用**」と言います。実は7行目の「>>=」の右側も、部分適用された関数になっています。この話を深堀りすると、「**カリー化**」という言葉に突き当たりますので、興味のある人は調べてみてください。

10行目では、mapの出力をB.unlinesに入力して、行ごとにリストになった文字列を再び1列の文字列にして、返します。このように括弧が多いと読みにくいので、

```
B.unlines $ map (slfLn fs) (B.lines cs)
```

という書き方もできます。

リスト3に、slfLn以下をちゃんと書いたものを示します。ここも見所がたくさんあり過ぎて勘弁してくれと思われる方も多いと思いますが、これは読み物です。我慢して読んでみてください。コンパイルして動かすと**図10**のようになります。このコード、エラー処理は適当ですのでご注意を。また、UTF-8の文字列を処理するときに1バイト文字と3バイト文字しか入力されないことを仮定しています。

まず、14行目のwhereから。13行目で、定義されていない「p、ws」を使っていますが、これをwhere以下で定義できます。where以下は、whereの範囲を示すためにインデントを合わせて書く必要があります。

14行目のwsは、1行を空白で切ったフィールドのリストになります。単語の分割にはB.wordsという関数がありますが、欲しい動作と微妙に違うので自前で分割処理を書きました。filterは、mapと同じく第1引数にBool値を返す関数、第2引数にリストをとります。そして、第1引数の関数がTrueを返すものだけを残したリストを作ります。ここでのfilterは、「B.split ' ' ln」で1行を空白で区切ってリストにしたもののうち、空の文字列以外を残すために使っています。第1引数の関数は、「/= (B.pack "")」です。「B.pack ""」は空のB.ByteString文字列のことですので、意味的には/= ""という関数ですが、これって関数なんでしょうか？

これも部分適用です。関数/=に""を適用したものと解釈できます。

ところで14行目は「/= (B.pack "")」を、

```
\x -> x /= (B.pack "")
```

と書き換えても動きます。このように\と->を使って書かれた関数は、「**無名関数**」と呼ばれます。冒頭で、

```
f: x ↦ y
```

という関数の表現を見ましたが、これの「f」に相当する名前がないという意味で無名です。また、このような方法で定義された無名関数を「**ラムダ抽象**」や「**ラムダ式**」と呼びます。ASCIIコードでλと書けないので、代わりに\を使います。

この関数には、まだまだ別の書き方があります。

```
ws = [ w | w <- B.split ' ' ln , w /= ➥
B.pack "" ]
```

とも書けます。この書き方は、「**リスト内包表記**」と呼ばれます。wsを「B.split ' ' ln」でできた各単語wのうち、「w /= B.pack ""」を満たすものでリストを作る、と読めます。

場合分けはパターンマッチとガードで

ところで、whereの中にpという関数の定義が2つあるのがわかるでしょうか？

パターンマッチ

引数を見るとわかるのですが、上のpは、Fieldのうち、FullFで作られたもの、下のpは、SubFで作られたものに適用されます。関数の定義が2つ以上あると、引数が書いてあるものと条件が一致するか上から順にチェックされ、一致したものが適用されます。このような書き方を「**パターンマッチ**」を使うと言います。

15行目のf、16行目のf、sにはInt型の数字が入るので右辺で使えます。15行目の「ws !! (f-1)」は、wsからf-1番目の要素を表します。C言語でいうws[f-1]ですね。

```
    p ws (FullF f ) = ➥
ws !! (f-1)
    p ws (SubF f s) = ➥
B.pack $ cutW ➥
(B.unpack $ ws !! ➥
(f-1)) (s-1)
```

16行目では、B.packとB.unpackという操作を「ws !! (f-1)」に適用しています。これは些細なことなのですが、B.ByteStringは単なるバイト列扱いで、そのままだと文字列処理が難しいので、一度B.unpackという関数を通して文字列型[Char]にしてcutWに入力しています。cutWの出力はB.packに通されて再度B.ByteStringに戻されます。

cutWは文字を切り出す関数で18行目以下に定義されています。文字列(リストになったバイト列)と、あと半角いくつ分字を削れば良いかを引数にとります。この関数の19、20、21、25、28行目はパターンマッチですが、22行目などの縦棒「|」で始まる行は何でしょう？

▼リスト3　標準入力の処理部分を最後まで書いたもの（selfproc.hs）

```
 1  import Data.Char
 2  import qualified Data.ByteString.Lazy.Char8 as B
 3
 4  data Field = FullF Int | SubF Int Int
 5
 6  main :: IO ()
 7  main = B.getContents >>= B.putStr . slf [FullF 1,SubF 2 ➥
3,SubF 3 3]
 8
 9  slf :: [Field] -> B.ByteString -> B.ByteString
10  slf fs cs = B.unlines $ map (slfLn fs) (B.lines cs)
11
12  slfLn :: [Field] -> B.ByteString -> B.ByteString
13  slfLn fs ln = B.unwords $ map (p ws) fs
14    where ws = filter (/= (B.pack "")) (B.split ' ' ln)
15          p ws (FullF f ) = ws !! (f-1)
16          p ws (SubF f s) = B.pack $ cutW (B.unpack $ ws !! ➥
(f-1)) (s-1)
17
18  cutW :: [Char] -> Int -> [Char]
19  cutW []         _  = error "invalid cut pos"
20  cutW bs         0  = bs
21  cutW (a:b:c:bs) n
22   | a < '¥128'      = cutW (b:c:bs) (n-1)
23   | isHanKana a b c = cutW bs       (n-1)
24   | otherwise       = cutW bs       (n-2)
25  cutW (a:b:[])   n
26   | a < '¥128'      = cutW [b]      (n-1)
27   | otherwise       = error "invalid string"
28  cutW _          _  = error "invalid cut pos"
29
30  isHanKana a b c = x >= 0xEFBDA1 && x <= 0xEFBE9F
31    where x = (ord a)*256*256 + (ord b)*256 + (ord c)
```

▼図10　コンパイルして動作確認

```
$ ghc selfproc.hs
[1 of 1] Compiling Main             ( selfproc.hs, selfproc.o )
Linking selfproc ...
$ cat hoge | ./selfproc
aaa 田 ﾀﾞ
bbb 田 ﾀ
```

ガード

これは「**ガード**」というものです。関数の左辺と右辺の間に「| <条件> =」と書くことで場合分けができます。パターンマッチと似ていますが、パターンマッチはおもに型やデータ構造を条件に使いますが、ガードの条件にはBool値を返す関数を指定します。

これをふまえて18行目から下に向かってcutW関数を読んでみましょう。まず、19行目のパターンは、[Char]のリストが空になってしまったらエラーを返すという式です。文字列を空にするとフィールドがずれるのでエラーにします。「error :: String -> a」はかなり乱暴な関数で、文字列をエラー出力に出してプログラムを終わらせます。Haskellの世界ではあまり使わないほうが良いとされていますが、コマンドは小さいプログラムですのであまり気にすることはないかなと考えています。もちろん、すぐ終わるからといってメモリリークするわけではありません。それから、15行目などの「ws !! (f-1)」も、fに変な数字が入っていたら、その場でプログラムは終わります。

20行目は、削るときの数が0だったら、削る必要がないので単にリストを返すという意味です。21行目は「リストの先頭の3つのバイトが取り出せたら」という意味になります。

このパターンにマッチするのは、長さが3以上のリストです。この行にマッチした場合、次に22～24行目でガードを使ってさらに場合分けします。ここでは、aが1バイト文字だったら、aを削ったリスト「b:c:bs」を、自分自身であるcutWに渡します。また、削らなければならない字数を半角1字分、減らします。このような再帰呼び出しをHaskellでは多用します。

ちなみに、正常に処理が進めば、20行目にマッチして再帰が終わります。23行目では、「a,b,c」を関数isHanKanaに渡して半角カナなのかどうか判別しています。半角カナなら、「a,b,c」をリストから除去し、半角1字分数字を減らしてcutWを呼び出します。24行目は、1バイト文字でも半角カナでもなければ、全角文字と判断し(決め付け)、cutWに3バイト削ったbsと、2文字分減らした数字を渡します。

25～27行目は、バイト列が2バイトの場合にマッチしますが、説明は割愛します。

29行目の「cutW _ _」については、「_」は何にでもマッチします。ですから、27行目以上にマッチしないと必ず28行目にマッチし、エラーが呼ばれてプログラムは終わります。

自然な感じでパーサを書く

さて今度は**リスト4**に、オプションを解析するプログラムを示します。このプログラムは本稿でぜひ触れたかった「**パーサコンビネータ**」を使ったパーサです。実行すると、**図11**のような出力でオプションがField型に変換されていることがわかります。

リスト4について、文法はかなり端折って流れだけ説明すると、7行目のgetArgsでselfコマンドのオプションを読んでいます。オプションはgetFieldsに渡ります。getFieldsはオプションを1つずつ、fieldというパーサ(文字列を解析する関数)に食わせています。

15行目のfieldは、「まずパーサsubFを適用してみて、失敗したらsimpleFを適用し、それでも失敗したらエラーを返せ」という意味です。4行目でdata FieldにError Stringという型コンストラクタが追加されていますので、エラーはこれに乗せて返します。型は型推論に任せて省略しています。

17～20行目のsubF関数は、まず数字を読み込みドットを読み込んで、もう一度数字を読み込みます。これがうまくいったら20行目でSubFで読み込んだ数字をField型にします。ドットがなくて失敗したら、失敗した旨を返します。subFの中のdigitも数字1字を読むパーサです。

このように、パーサの関数fieldは、パーサの関数subFやsimpleF、subFはパーサの関数digitなどを組み合わせて構成されます。また、digitは1字の数字のパーサですが、「many1 digit」と

することで、1字以上の数字の文字列というパーサになります。

こうやって小さいパーサを組み合わせて大きなパーサを作るということがパーサコンビネータという言葉の意味です。BNF(バッカス・ナウア記法)を表現するような特別なものを持ち込まなくても、Haskellの関数をブロック遊びのように組み合わせるだけでパーサが書けるのです。実際のselfのオプションはもっと複雑ですが、それでもこんな調子で書いていけばあまり悩まなくて済みます。

22行目のsimpleFは、わざとsubFと違う書き方をしました。再び「>>=」が登場しました。実は裏にはモナドがいます。

17行目のdoは何か手続き型のプログラミングみたいに見えますが、「>>=」で一直線にモナドが操作できないときの書き方で、基本、同じものです。

最後に、作ったselfproc.hsとselffield.hsを合体してself.hsを完成させます。冒頭の部分を**リスト5**のように書き換えて、あとは作った関数をselfproc.hsとselffield.hsのmainとdata Field以外をコピー&ペーストしてください。

「え？　それで動くのか？」さすがに正しくコピー&ペーストしないといけませんが、動きます。なぜなら、Haskellのコードは、副作用のない関数をつなげたものですので、同じ名前の関数でもない限り、互いに悪さはいっさいしないことが保証されています。

結局、このように互いに関数の独立性が高くなることは、副作用を徹底的に嫌ったことの成果とも言えます。

▼リスト4　オプションを解析する(selffield.hs)

```
 1  import System.Environment
 2  import Text.ParserCombinators.Parsec
 3
 4  data Field = FullF Int | SubF Int Int | Error String ➤
deriving Show
 5
 6  main :: IO ()
 7  main = getArgs >>= print . getFields
 8
 9  getFields :: [String] -> [Field]
10  getFields as = map f as
11    where f x = case parse field "" x of
12                    Right opt -> opt
13                    Left err  -> Error (show err)
14
15  field = try(subF) <|> try(simpleF) <|> (return $ Error ➤
"option error")
16
17  subF = do f <- many1 digit
18            char '.'
19            s <- many1 digit
20            return $ SubF (read f) (read s)
21
22  simpleF = many1 digit >>= return . FullF . read
```

▼図11　selffieldの実行

```
$ ghc selffield.hs
[1 of 1] Compiling Main             ( selffield.hs, selffield.o )
Linking selffield ...
$ ./selffield 1.2 3.3 2
[SubF 1 2,SubF 3 3,FullF 2]
```

▼リスト5　self.hs(ただし冒頭だけ)

```
 1  import System.Environment
 2  import Data.Char
 3  import Text.ParserCombinators.Parsec
 4  import qualified Data.ByteString.Lazy.Char8 as B
 5
 6  data Field = FullF Int | SubF Int Int | Error String ➤
deriving Show
 7
 8  main :: IO ()
 9  main = do as <- getArgs
10            cs <- B.getContents
11            B.putStr $ slf (getFields as) cs
```

最後はLazyな話

ところでこれ、ちゃんとまともに動くのでしょうか？標準入力から何GBも読み込んだら、getContentsはメモリに全部抱え込むのではないでしょうか？そういう心配がないことは、**図12**のような実験で確認できます。入力を始めたらすぐ出力が始まります。

Haskellのプログラムは、「ある処理を行うために必要になったら必要な処理を行う」という動作をします。標準入力も必要な分だけチビチビ読まれます。このような動作は遅延評価というプログラムの評価(実行)順序で実現されています。ByteStringの定義は、Data.ByteString.Lazy.Char8で行っていますが、遅延評価を行わないようにしたData.ByteString.Char8というモジュールもあります。

これを使ってself.hsをコンパイルすると、**図13**のようにメモリをバカ食いした挙げく終わらないコマンドができてしまいますので、ご注意を。

▼図12　大きなデータを読ませてもすぐ処理が始まる

```
$ ghc -O2 self.hs
[1 of 1] Compiling Main             ( self.hs, self.o )
Linking self ...
$ time seq 1 10000000000 | ./self 1 | head -n 3
1
2
3

real 0m0.023s
user 0m0.017s
sys 0m0.011s
```

▼図13　Data.ByteString.Char8の場合

```
$ time seq 1 10000000000 | ./self 1 | head -n 3
(終わらない)
```

おわりに

本稿では、selfというコマンドを例に、実際に動くHaskellのコードを読んで特徴を洗い出しました。おそらく、わけのわからない概念がたくさん出てきて読む方はたいへんだったと思いますが、筆者が考えるHaskellの比類なき武器であるParsecまで紹介できました。もし、「こういう世界」に何か感じるところがあれば、もう、あなたは関数の世界に足を突っ込んでいるかもしれません。SD

Column GHCiを使って型の例をもう少し

selfが動いたので「実用コードでHaskellを説明」は達成したものの、リストと型についてはもうちょっと簡単な例で示した方がよいのでここで補足をします。

たとえば、**図A**のようにGHCiでtehaiという「**定数**」を定義します[注7]。letは、GHCiで関数や定数の定義を書くときに使うものです。コンパイラを使うときは不要です。

▼図A　GHCiでtehaiという「定数」を定義

```
Prelude> let tehai = ["東","南","南",↵
"南","西","西","西","北","北","北","II",↵
"II","3"]
Prelude> tehai
["¥26481","¥21335","¥35199",...(略)
```

注7) 引数ゼロの関数じゃねえか！と思った方はかなり関数型に毒されています。

右辺がリストになっています。リストは、このように同じ型の要素をカンマ区切りで並べて[]で囲って定義します。

GHCiで単に「tehai」と打つと、「print tehai」と打ったことになりますが、printはあくまでデバッグ用です。tehaiを日本語でちゃんと表示するには、putStrLnなどの関数を使います。putStrLnとtehaiの型を調べてみましょう。

```
Prelude> :t tehai
tehai :: [[Char]]
Prelude>  :t putStrLn
putStrLn :: String -> IO ()
```

tehaiが「文字のリストのリスト」、putStrLnが、文字

列を受け入れて「IO ()」を返す（そして標準出力に文字列を出力）する関数です。実は[Char]とStringは同じ型を指すのでtehaiの型は[String]と等価ですが、それにしても型が合っていないので、「[[Char]] (= [String])」をStringに変換しないとputStrLnが使えません。

Haskellにそこそこ慣れると、じゃあ何か型を変換できる関数はないかという発想になります。こういうときは、たとえばunwordsという関数を使います。型はこうなってます。

```
Prelude> :t unwords
unwords :: [String] -> String
```

unwordsの動作はまだ説明していませんが、型が合ってたらつなぐことはできます。つなげて使ってみましょう。

```
Prelude> putStrLn (unwords tehai)
東 南 南 南 西 西 西 北 北 北 II II 3
Prelude> ( putStrLn . unwords ) tehai
東 南 南 南 西 西 西 北 北 北 II II 3
```

ちゃんと日本語が表示されました。unwordsはリストの文字列を空白区切りでつなぐ関数で、その出力はString、つまりputStrLnで扱える型になります。同じ出力が2つの方法で得られましたが、前者の例は「unwords tehai」でtehaiをStringに変換してからputStrLnに引数として渡す、後者の例は「putStrLn . unwords」という「[String] -> IO ()」という型を持つ合成関数にtehaiを渡しています。やっていることは同じですが意味的には違うということになります。

そうこうしているうちに、東をツモりました。話が麻雀にヨレてきましたが、とりあえずtehaiに組み入れましょう。++あるいは:という演算子を使います。演算子といってもこれらも関数扱いでき、

```
Prelude> :t (++)
(++) :: [a] -> [a] -> [a]
Prelude> :t (:)
(:) :: a -> [a] -> [a]
```

というふうに型を調べられます。型だけではわからないので動作を補足説明すると、++はリストとリストをつなぎ、:はリストの要素とリストをつなぎますので、

```
Prelude> ( putStrLn . unwords ) ↵
("東":tehai)
東 東 南 南 南 西 西 西 北 北 北 II II 3
Prelude> ( putStrLn . unwords ) ↵
(["東"]++tehai)
東 東 南 南 南 西 西 西 北 北 北 II II 3
```

というように微妙な型の違いに気をつけて使います。通常は:を使います。さらに3を捨てましょう。filter関数に「/= "3"」という関数を渡して"3"を除去します。これだと"3"が2つ以上あると小牌しますが……。

```
Prelude> let tehai' = filter (/= "3") ↵
("東":tehai)
Prelude> ( putStrLn . unwords ) tehai'
東 東 南 南 南 西 西 西 北 北 北 II II
```

※Haskellではもとのものからちょっと違う変数や関数を作る時に、もとの名前に「'」をつける習慣があります。

filterの型はこうなってます。

```
Prelude> :t filter
filter :: (a -> Bool) -> [a] -> [a]
```

つまり、「a -> Bool」という型の関数とリストを引数にとって、リストを返すわけです。型で見ると高階関数も型の掟に従っているのがわかります。さらに、部分適用で作った関数「filter (/= "3")」の型を見ると、文字列"3"を手がかりに型aが[Char]に確定する様子が観察できます。

```
Prelude> :t filter (/= "3")
filter (/= "3") :: [[Char]] -> [[Char]]
# (参考) 別の型になる例
Prelude> :t filter (/= True)
filter (/= True) :: [Bool] -> [Bool]
```

さあ、四喜和（スーシーホー）[注8]をテンパりましたが、これ以上書くと叱られそうなのでここでやめときます。言いたかったことは、とにかく型を理解して型を合わせるように書けばHaskellのコードが書けるようになるよということでした。次号、詰むや、詰まざるや[注9]。

注8） 麻雀の役満。
注9） 将棋の決め台詞な上に次号もありません。

Pythonにおける関数型プログラミング

Writer 柏野 雄太(かしの ゆうた)バクフー株式会社代表取締役／Twitter@yutakashino

Pythonにおける関数型プログラミングの位置づけと実践的な使い方を雑感的に述べます。

Pythonと関数型プログラミング

Pythonは次のように、関数型プログラミング言語の多くの特徴を持っています。

- 関数がファーストクラスオブジェクトである
- 制御構造として「ループ」を再帰で表現する(ことができる)
- データや関数の処理がリストの処理となる
- データを再代入するなどの副作用を避けることができる(純関数型では許さない)
- 宣言を避けることができる(純関数型では許さない)
- 高階関数をサポートする

このような点から「Pythonは関数型プログラミング言語である！」的な主題で本稿を「無理やり」書いてもそれはそれで面白いとは思いますが、日々多くの現実的な課題に直面する大人として、やはりプラクティカルな面に着目するほうがよさそうです。

結論から述べますと、元来Pythonは関数型プログラミングを目指して作られていないために、通常の関数型プログラミング言語に比べて、関数型プログラミングは不得意です。Pythonのクリエイターであるグイド・ファン・ロッサムもはっきりとPythonは関数型プログラミング言語でないと次のように述べています[注1]。

> いろいろな人が指摘をしているが、Pythonが関数型プログラミングの影響を強く受けているというのはまったくありえない。僕は関数をファーストクラスオブジェクトとして実装したけれど、CやAlgol 68のような手続き型プログラミングのほうが全然得意だし、Pythonを関数型プログラミングとしてみなすことはない。

関数型プログラミング言語として必要な機能を、Pythonは言語として実装していないことがその不得意さの理由となっています。たとえばStackOverflowの議論の1つ[注2]では、次の点によりPythonを純粋な関数型プログラミング言語として扱うには無理があるとしています。

- 末尾再帰がない
- パターンマッチングがない
- リスト操作が貧弱
- 遅延リストがない
- ラムダに式しかかけない
- ifが式でなく文である

本稿では、通常の関数型プログラミング言語の特徴のうち、Pythonで利用するべきでないもの、その反対に積極的に利用するべきものを、網羅的には紹介できないですが、肩肘張らずに雑感的に述べることにします[注3]。

注1) "Origins of Python's 'Functional' Features"(http://python-history.blogspot.jp/2009/04/origins-of-pythons-functional-features.html)

注2) "Why isn't Python very good for functional programming?"(http://stackoverflow.com/questions/1017621/why-isnt-python-very-good-for-functional-programming)

注3) 以降の説明ではいまだに利用者が圧倒的に多いPython 2系をターゲットにしますが、Python 3系もほとんど同じように当てはまると思います。

Pythonでの再帰

普通の関数型プログラミングでは、**for**や**while**などの手続き型ループを使わず再帰でループを表現します。Pythonでも再帰で関数ループを表現することもできますが、末尾再帰が言語として実装されていないので、手続きループに比べて遅く、メモリを使い果たし、とても非効率です。このことを具体例で示します。

今、ループを**for**文と再帰で表現した2つの関数を次のように記述します。

```
# forvsrecurse.py
def forloop(n):
  for i in xrange(int(n)):
    x = 1

def recloop(n):
  if n <= 0: return
  x = 1
  recloop(int(n) - 1)

if __name__ == '__main__':
  pass
```

ここで、それぞれの関数の引数として1,000を入れてみた計算を、IPythonの**timeit**マジック[注4]でそれぞれの関数の時間計測をします。結果は**図1**のように、再帰が15倍以上も遅いことがわかります。

実はPythonにおける再帰は、速度が遅いだけではありません。この例では**sys.setrecursionlimit**[注5]によってあらかじめ最大再帰スタックを拡張していますが、デフォルトのPythonの設定ではスタックがオーバーフローしてクラッシュしてしまいます。

このように、Pythonにおいてループを再帰で表すのは、とても効率が悪く、関数型プログラミングをしているという自己満足やほかの人への意地悪くらいの効果しかないです。Pythonにおける再帰の乱用はやめたほうがいいと思います。

注4）http://ipython.org/ipython-doc/dev/interactive/tutorial.html
注5）http://docs.python.jp/2/library/sys.html

▼図1　関数処理の実測値

```
% ipython

In [1]: import forvsrecurse

In [2]: %timeit forvsrecurse.forloop(1e3)
10000 loops, best of 3: 21.5 µs per loop

In [3]: import sys

In [4]: sys.setrecursionlimit(int(1e6))

In [5]: %timeit forvsrecurse.recloop(1e3)
1000 loops, best of 3: 325 µs per loop
```

Pythonでのλ（ラムダ）

λ（ラムダ）計算は関数型プログラミングの最重要のビルディングブロックの1つです。しかし、Pythonにおけるλは、コールバック関数として利用するのに便利なほかは、あまり使い勝手がよくないので、二級市民扱いです。それは次の理由からです。

- 無名関数としても限定的な利用に限られる
- 式のみで、宣言では使えない
- Pythonでは通常の関数の定義も2行増える程度であるように、たいして難しくないので、λ式のメリットがあまりない
- Pythonの最大の特徴である人間にわかりやすい言語という側面が失われる
  - →λ式では関数名が使えない
  - →λ式ではdocstringが使えない

ただし、Pythonのλ式はコールバックとしては使い手があります。

```
>>> def somefunc(callback, arg):
...   return callback(arg)
…… (略) ……
>>> somefunc(lambda x: x**2, 10)
100
```

定番モジュール：itertools、functools

Pythonにおける関数型プログラミングの定番モジュールといえば、**itertools**と**functools**になります。しかし、この2つのモジュールの解説をすると、それだけで紙幅が尽きてしまいます。一方で、これらのモジュールにはとても

良いチュートリアルがありまして、それを紹介することにして解説に変えさせていただきます。

itertools

・ダグ・ヘルマン氏の"Python Module of the Week"のitertools解説[注6]

・ローリ・ゲーガン氏のMontrealPythonにおけるitertools講演[注7]

functools

・ダグ・ヘルマン氏の"Python Module of the Week"のfunctools解説[注8]

リスト内包表記は積極的に

Pythonにおけるスタンダードライブラリの**map**、**apply**、**filter**、**reduce**は非常に多く利用される関数型プログラミングフレーバーの関数群です。そのなかでも**map**や**filter**は多用されると思いますが、Pythonでは一般的にmapを利用するよりもリスト内包表現を利用したほうが、効率が良く、わかりやすいです。具体例を示します。

注6) Doug Hellmann "Python Module of the WeeK: itertools - Iterator functions for efficient looping"(http://pymotw.com/2/itertools/index.html)

注7) Rory Geoghegan "Module of The Month: itertools"(http://www.youtube.com/watch?v=VxPoJbpqAzI), "Module of The Month: itertools 2"(http://www.youtube.com/watch?v=nsLOzpwYII8)

注8) Doug Hellmann "Python Module of the WeeK: itertools - Iterator functions for efficient looping"(http://pymotw.com/2/functools/index.html)

▼図2　内包表現の実測値

```
% ipython

In [2]: %timeit map(lambda x: x**2, xrange(1, 10))
1000000 loops, best of 3: 1.76 µs per loop

In [3]: %timeit [x**2 for x in xrange(1, 10)]
1000000 loops, best of 3: 1.23 µs per loop
```

▼図3　filterのほうが処理が遅い

```
In [6]: timeit filter(lambda x: x > 10, rxange(1, 20))
100000 loops, best of 3: 2.55 µs per loop

In [7]: timeit [x for x in xrange(1, 20) if x > 10]
1000000 loops, best of 3: 1.39 µs per loop
```

まず**map**とリスト内包表現の実行速度をIPythonの**timeit**マジックで速度測定したものが**図2**、**図3**です。結果はわずかではありますが、mapよりリスト内包表記が速いことがわかります。

Pythonでの遅延評価：ジェネレータ

PythonにはHaskelなどが持つ純粋な関数型プログラミングとしての網羅的な遅延評価の機能がありません。しかし、Pythonにおいてジェネレータは遅延評価される数少ない機能の1つで、メモリリソースとCPU時間の効率的利用のために積極的に利用するべきです。

遅延評価を用いないリスト内包表現とジェネレータ表現を比べてみます。リスト内包表現はすぐにリストを返しますが、この返すリストが巨大になる場合、ここで大きなリソース消費が発生してしまいます。

```
>>> [x**2 for x in xrange(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

一方でジェネレータ表現は、次のようにすぐにはリストを返さず、ジェネレータオブジェクトを返します。つまりジェネレータではすぐに評価は行われないのです。ジェネレータオブジェクトは**next()**メソッドが呼ばれることで初めて評価を行い値を返します。このために巨大なオブジェクトであってもこの方法ならリソースの無駄遣いがなく効率よく処理できます。

```
>>> gen = (x**2 for x in xrange(10))
>>> gen
<generator object <genexpr> at 0x10047fd70>
>>> gen = (x**2 for x in xrange(10))
>>> gen.next()
0
>>> gen.next()
1
>>> gen.next()
4
```

別の例を見てみます。これは2009年のPyCon USにおけるデビッド・ビーズリー氏の講演[注9]か

注9) David Beazley "PyCon 2009: A Curious Course on Coroutines and Concurrency"(http://pyvideo.org/video/213/pycon-2009--a-curious-course-on-coroutines-and-co, http://pyvideo.org/video/215/pycon-2009--a-curious-course-on-coroutines-and-c1, http://pyvideo.org/video/214/pycon-2009--a-curious-course-on-coroutines-and-c0)

ら取った例ですが、カウントダウンを関数として実現する例です。

```
>>> def cd(n):
...   while n > 0:
...     yield n
...     n -= 1
...
>>> x = cd(10)
>>> x
<generator object cd at 0x10056b820>
>>> x.next()
10
>>> x.next()
9
>>> for i in x:
...   print i,
...
8 7 6 5 4 3 2 1
```

Pythonにおいて遅延評価のポイントとなっているのが**yield**文で、この**yiled**文があるために関数は値をまとめてすぐに返すことをしないで、**next()**メソッドにより1つずつ生成するようになるのです。これがPythonにおける遅延評価のやり方です。

ジェネレータを利用したパイプライン処理

ジェネレータを利用すれば、関数型プログラミングでお馴染みのパイプライン処理を簡単にコーディングできます。これも上記のビーズリー氏の講演からの例ですが、UNIX系のシェルではお馴染みの、ログファイルを**tail -f**をして**grep**をパイプでつなぐような処理を簡単に書くことができます(**リスト1**)。

▼リスト1　パイプライン処理の例

```
def follow(thefile):
  thefile.seek(0,2)
  while True:
    line = thefile.readline()
    if not line:
      time.sleep(0.1)
      continue
    yield line

def grep(pattern, lines):
  for line in lines:
    if pattern in line:
      yield line

if __name__ == '__main__':
  # Set up a processing pipe : tail -f | ➥
grep python
  logfile = open("/var/log/system.log")
  loglines = follow(logfile)
  greplines = grep("Safari", loglines)
  # Pull results out of the processing
pipeline
  for line in greplines:
    print line,
```

リスト1の例はMac OS Xのシステムログファイルを**tail -f**して"**Safari**"という文字列を探す簡単な例です。**follow**関数が**tail -f**で、**grep**関数が**grep**です。どちらも**yield**文によりジェネレータ関数となっていまして、2つの関数が順次呼ばれることにより、値が1つずつ評価され、パイプラインが実行できるようになっています。

上記の例では、**yield**文がジェネレータ関数を作り出し、遅延評価として**next()**で値を**'pull'**する例をみましたが、Python 2.5から**yield**を式を使うことで、ジェネレータに値を**'push'**できるようになりました。この**'push'**と**'pull'**を使えば、Pythonにおける軽量並行プログラミングのビルディングブロックとなるべき**コルーチン**が自然に構築できます。Pythonのコルーチンの解説についても、紙幅の関係でここでは述べることができませんが、上記ビーズリー氏の講演(3つのビデオに分かれています)が素晴らしい解説となっていますので、興味がある方は閲覧ください。

関数型プログラミング拡張：fn.py

fn.py[注10]は**functools**を拡張する関数型プログラミングモジュールです。これを使うと、Pythonを使っているのに見た目に関数型プログラミングをしているような錯覚を味わうことができます。たとえば、標準のPythonにはパターンマッチングはありませんが、**fn.py**には_という関数を利用して、それっぽい見せかけができます。

```
>>> from fn import _
>>> print (_ **2)(3)
9
>>> print (_ + _)(1, 2)
3
```

注10) https://github.com/kachayev/fn.py

そのほかにもHaskelやScalaでお馴染みのストリームや無限シーケンスなどが簡単に実現できます。

```
>>> from fn import Stream
>>> s = Stream() << [1, 2, 3, 4, 5]
>>> s
<fn.stream.Stream object at 0x10057b2d8>
>>> list(s)
[1, 2, 3, 4, 5]
>>> s[1]
2
>>> from fn.iters import map, drop
>>> from operator import add
>>> f = Stream()
>>> fib = f << [0, 1] << map(add, f, drop(1,
f))
>>> list(fib[0:20])
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,
144, 233, 377, 610, 987, 1597, 2584, 4181]
```

ただ、このモジュールはピュアPythonで書かれたものなので、これを使って見た目に関数型プログラミングっぽく実装できたからといって、本当に効率がよくなるわけではないことに注意してください。

副作用について

Pythonでは副作用を避けるための手段が著しく限定されてます。副作用を避けるためには、副作用のないイミュータブルなデータ型(タプルやフリーズセット)を利用したり、イミュータブルな属性を持つカスタムクラスとして実装することになります。しかし、Pythonではそのことによって逆にコードが冗長になり読みにくくなるという「副作用」が避けられず、一般的に実行効率も悪いものとなりがちです。Pythonにおいては、副作用を避けるために無理なプログラミングをしてもそれだけの効果は得られません。

単にバグを生まないようにするという観点からは、コーディング規律として、

- グローバル変数を使わない
- クラスの実装では`__init__`に属性を集めて副作用の範囲を限定する
- クラスメソッドに`staticmethod`を使う

くらいに留めておいたほうがいいと思います。

終わりに

本稿で示したように、Pythonはある程度まで関数型プログラミングが可能なことは事実です。しかし、前に引用したPythonクリエイターのグイド・ファン・ロッサムも語るように、Pythonは本質として関数型プログラミング言語ではありませんし、それを指向してもいません。したがって、ただ関数型プログラミングが「カッコイイ」からといって、無理矢理Pythonで関数型プログラミングをするのは、賢い大人のやることではありません。

その一方で、いくつかのPythonの関数型プログラミング風の機能はとても有用です。たとえば、リスト内包表現やジェネレータはPythonの関数型プログラミングの部品としてとてもよくできていますので、積極的に使うべきです。結局のところ、Pythonにおける関数型プログラミングにおいては、手続き型プログラミングやオブジェクト指向プログラミングを組み合わせながら得意な部分だけを小さく使うことが要諦になります。

よくPythonを使う人間の間において、Pythonicかどうかというのが議論されますが、Pythonicというのは得意な部分を手早くシンプルな記述でわかりやすく組み合わせる精神から出ているとも言えます。Pythonicに関数型プログラミングとつきあうことがPythonにおいて関数型プログラミングを使うためのクライテリオンとなるでしょう。

最後になりますが、この稿の骨子は2013年のPyCon USにおけるマイク・ミューラー氏の講演[注11]から多大に影響を受けています。筆者が長年もやもやしていたPythonにおける関数型プログラミングの位置づけを鮮やかに描き出してくれました。氏に感謝します。SD

注11) Mike Muller "Functional Programing with Ptyhon"(http://www.youtube.com/watch?v=Ta1bAMOMFOI)

実践 Erlang
～高可用サーバを作ってみよう～

Writer 篠原 俊一（しのはら しゅんいち） Bashoジャパン株式会社（http://basho.co.jp/）

みなさんは“Erlang/OTP”から何を思い浮かべるでしょうか。大量に起動可能な軽量プロセスや、メッセージパッシングによる並行処理、そしてメッセージパッシングはネットワークを介しても透過性を持つという特徴を考える方もいることでしょう。しかしErlang/OTPの最大の目的は高可用性を持つシステムの構築にあります。高可用で高信頼のシステムをなるべく簡単に構築するため、Erlang/OTPがサポートする、いぶし銀の機能を紹介します。

Erlang/OTPとは

Erlang/OTP[注1]は、高可用性を第一に設計、実装された言語、処理系、およびフレームワークです。Joe Armstrongの論文[注2]冒頭の問題設定では高い信頼性を持つシステムを作るという実用主義的な目的を掲げています（以下に引用）。

“The central problem addressed by this thesis is the problem of constructing reliable systems from programs which may themselves contain errors.”

そこを出発点に、軽量プロセスによる並列処理、メッセージパッシング、そして別ノード上のプロセスも透過的に扱えるといった言語仕様が、結果として導かれたのです。

本稿では、サンプルアプリ作成を通じて、高可用性を実現するためのErlang/OTPのしくみのうち、エラー処理と、稼働中の内部調査およびパッチ適用を紹介します。

一方、Erlangを関数型言語の視点から見た場合、単一代入、パターンマッチ、第一級オブジェクトとしての関数という性質を持っています。これらの性質により、ソースコードが短く書ける、可読性が上がるという利点があります。詳しくは、参考書籍[注3]をご覧ください。

実稼働の例

Erlang/OTPで構築されたシステムは、開発元のエリクソンを含め、さまざまに使われています。

Heroku社[注4]では、HTTPリクエストとDyno[注5]をつなぐRouting MeshにErlang/OTPが使われています[注6]。「賢いプロキシ」として、ルーティング、ロードバランス、カスタムの認証、動的なアプリケーション拡張といった役割を担っているようです。またlogprexというsyslogルータは、GitHub上でOSSとして公開されています。

比較的最近の話題としては、WhatsAppというスマートフォン向けチャットアプリでの利用があります。Erlang Factory 2012でのプレゼンテーション[注7]では、SMP環境でのスケーラビリティやボトルネック解析、Erlangの処理系（BEAM）

注1）Erlang/OTP：http://www.erlang.org/、https://github.com/erlang/otp

注2）Joe Armstrong 論文、インタビュー：http://www.sics.se/~joe/thesis/、http://www.se-radio.net/2008/03/episode-89-joe-armstrong-on-erlang/

注3）Programming Erlang：http://pragprog.com/book/jaerlang2/programming-erlang、この翻訳本は『プログラミングErlang』：http://ssl.ohmsha.co.jp/cgi-bin/menu.cgi?ISBN=978-4-274-06714-3

注4）https://www.heroku.com/

注5）Herokuでのプロセス単位。

注6）Erlang Factory - Blake Mizerany, Works with Ruby and Erlang at Heroku：http://www.erlang-factory.com/conference/London2009/speakers/blakemizerany

注7）Erlang Factory - Rick Reed, Software engineer at WhatsApp：http://www.erlang-factory.com/conference/SFBay2012/speakers/RickReed

のロックコンテンションなど濃い話題が満載です。

decho（Delayed-ECHO）作成

1つ前に受け取ったメッセージをエコーするサンプルアプリを作りましょう。

目的は2つです。

①エラー発生時の挙動を知ること

ソフトウェアのバグやネットワーク、ハードウェアの故障が避けられない以上、エラーの影響を最小限に留めつつ、システムとして稼働し続けることが重要です

②稼働中プロセスの内部調査

システムの状態を詳細に取得できることは、障害への迅速な対応、バグの原因発見を可能にします。これをコードのホットスワップ機能と合わせると、稼働しながら調査と修正ができます

次に進む前に、R15B01以降のErlang/OTPを準備してください。Erlang Solutions[注8]のバイナリパッケージがお手軽です。

OTPに則ったapplicationとreleaseの生成

Erlang/OTPでは、OSプロセスとして動作する単位をreleaseと呼びます。releaseは関連性の強いモジュールのあつまりであるapplicationを複数含みます。これらを構成するには、決まった作法に従う必要があります。OTP Design Principles[注9]やLearn You Some Erlang for Great Good![注10]は、監視ツリーのしくみやリリースについて詳細に説明している必読の資料です。

ここではエッセンスを見るために、rebarを使い複雑な手順をスキップしましょう。

```
$ git clone https://github.com/rebar/rebar
$ (cd rebar && make)
```

次にdechoという名前でapplicationのひな形を作成します（**図1**）。

続いてreleaseを生成します。relディレクトリにreleaseに必要なファイルを生成し、releaseのレシピreltool.configを3ヵ所変更します（**図2**）。

最後にreleaseを生成し、起動します。

```
$ (cd rel; ../rebar generate)
$ rel/decho/bin/decho console
```

さて、見慣れたErlang shellが現れたでしょうか。application:which_applications()でdechoが見えると成功です。

生成されたrelease rel/dechoには、Erlang処理系と必要なライブラリがすべて含まれており、別サーバにそのままコピーして動かすことができます。ほかのソフトウェアとの依存関係衝突やライブラリのバージョン間互換性に悩むことなく、全部入りにする潔さはErlang/OTPの実用主義をよく反映しています。

エコーロジック実装

ロジックの実装は簡単です。まずsrc/decho_sup.erlを変更し、decho_serverを監視ツリーに組み込みます（**図3**）。

注8）https://www.erlang-solutions.com/downloads/download-erlang-otp

注9）http://www.erlang.org/doc/design_principles/des_princ.html

注10）http://learnyousomeerlang.com/

▼図1　applicationのひな形を作成

```
$ mkdir decho && cp rebar/rebar decho/
$ cd decho
$ echo '{deps, [{eper, ".*", {git, "git://github.com/basho/eper.git", master}}]}.' ¥
      > rebar.config
$ ./rebar create-app appid=decho
$ ./rebar create template=simplesrv srvid=decho_server
$ ./rebar get-deps && ./rebar compile
```

次はsrc/decho_server.erlです。1つ前のメッセージを保存する状態レコードstateを定義し、初期化関数initとメッセージ処理関数handle_callで利用します(**リスト1**)。

handle_callは1つ前のメッセージpreviousを応答し、受けたメッセージは次の状態とします。実装はこれで終わりです。先ほど起動中のErlang shellはq()で止めて、新しいコードで起動しなおしましょう。

```
$ ./rebar compile
$ (cd rel; ../rebar generate)
$ rel/decho/bin/decho console
```

サーバプロセスをgen_server:call/2で呼び出すと、直前のメッセージを返すようになりました。

```
> gen_server:call(decho_server, ham).
undefined
> gen_server:call(decho_server, bacon).
ham
```

エラーが発生するとどうなる？

正常系が動作するだけのソースコードと、高可用性を持つシステムの間には一般に大きな溝があります。プロダクションで使うコードには多くの例外系の処理、エラーのハンドリングが入っていることをよくご存じでしょう。

ここまで準備したコードは正常系のみの処理に見えます。わざとバグを埋め込み、エラーを発生させ、挙動を見てみましょう。handle_callに、boomメッセージをパターンマッチする処

▼図2　releaseを生成

```
$ mkdir rel
$ (cd rel; ../rebar create-node nodeid=decho)
```

```
編集前: {lib_dirs, []},
編集後: {lib_dirs, ["../deps"]},

編集後: {rel, "decho", "1",
         [
          kernel,
          stdlib,
          sasl,
          eper,   %% <== この行を追加
          decho
         ]},

編集前: {app, decho, [{mod_cond, app}, {incl_cond, include}]}
編集後: {app, decho, [{mod_cond, app}, {incl_cond, include}, {lib_dir, ".."}]}
```

※注：--disable-hipeを指定した処理系を使っている場合は、{app, hipe, [{incl_cond, exclude}]},も追加する必要があります。

▼図3　src/decho_sup.erlの変更

```
編集前: {ok, { {one_for_one, 5, 10}, []} }.
編集後: {ok, { {one_for_one, 5, 10}, [?CHILD(decho_server, worker)]} }.
```

▼リスト1　src/decho_server.erl

```
%% -export の後にレコード定義を追加
-record(state, {previous}).

%% init, handle_call 変更
init(_Args) ->
    {ok, #state{}}.

handle_call(Request, _From, #state{previous = Previous} = State) ->
    {reply, {ok, Previous}, State#state{previous = Request}}.
```

理を追加し、存在しないモジュールboom:boom()呼び出してエラーを発生させます(**リスト2**)。compile、generateして再起動しましょう。boom以外のメッセージには以前と同じ動作ですが、boomを渡した途端にエラーが発生します(**図4**)。

もう一度メッセージを投げるとどうなるでしょう。エラー後の最初の応答はundefinedとなり、新しいプロセスが取って代わっていることがわかります。そして通常の処理が動作します。

この裏側ではErlang/OTPがエラーログの出力やプロセス再起動などを自動で行っています。エラーログからは最後に来たメッセージ、状態の情報、エラーが起きたファイル位置を含めた停止理由が見て取れます。再起動のしくみには、単に止めて起動しなおすだけではなく、一定期間にエラー回数が閾値(しきい)を超えると監視ツリーの上位へエスカレーションすることや、最終的にはErlangノード自体の停止までを含んでいます。OTPの作法に従うと、詳細なログや安全装置つきのプロセス再起動といった高機能なしくみを簡単に利用できるのです。

なぜエラー発生時に、プロセスの再起動のみで対応できるかについては一考の価値があります。プロセス間でメモリを共有し、ロックを用いてアクセスする場合には、プロセスクラッシュ時の再起動のみで正しい状態に復帰できるとは限りません。メモリを共有せず、メッセージパッシングを用いる設計の選択により、再起動で対応ができ、高可用性を実現しやすくしているのです。

稼働中アプリケーションの内部調査

サーバアプリケーションをメンテナンスしている方は、たまに不可解な挙動をするが原因がわからない、という経験が一度ならずあるのではないでしょうか。Erlang/OTPには稼働中のアプリケーションを停止せずに調査する機能も備わっています。

Erlang組み込みのトレース機能をラップしたredbugを使ってみましょう。実は、先ほど作成したreleaseはすでにredbugを含んでいます。decho_serverモジュールのすべての関数呼び出しをトレースしてみます。

```
> redbug:start
    ("decho_server:'_' -> return").
```

gen_server:call(decho_server, ham)を実行すると実行がトレースされ、呼び出し時は引数、戻り時は戻り値が出力されているのがわかります(**図5**)。

redbugは、しばらくすると自動でトレースを終了します。トレースは多かれ少なかれ稼働中

▼リスト2　handle_callに存在しないモジュールを追加する

```
handle_call(boom, _From, State) ->
    {reply, {ok, boom:boom()}, State};  %% !!!セミコロン!!!
handle_call(Request, _From, #state{previous = Previous} = State) ->
    {reply, {ok, Previous}, State#state{previous = Request}}.
```

▼図4　実行するとエラーが発生する

```
> gen_server:call(decho_server, boom).
=ERROR REPORT==== 13-Dec-2013::16:24:17 ===
** Generic server decho_server terminating
** Last message in was boom
** When Server state == {state,bar}
** Reason for termination ==
** {'module could not be loaded', [{boom,boom,[],[]},
     {decho_server,handle_call,3,
      [{file,"/.../decho/src/decho_server.erl"}, {line,35}]}],
[以下略]
```

のプロセスに性能影響があるため、redbugは時間とトレース回数の両方に上限を付け、プロダクション環境でも安全に使える配慮をしています。

先ほどの呼び出しトレース中にboomメッセージを送ったならば、プロセスが再起動されている様子が明確にわかるでしょう。またメッセージ通信のトレースをするとプロセスがどう通信しているかを調べることもできます。稼働中でも関数呼び出しなどの動的な情報を取得できるため、調査の幅が大きく広がります。

そのほかにもErlang/OTPは内部の情報を取得できる関数を多数もっています。たとえばノードの中でメモリを多く消費しているプロセス一覧や、処理待ちのメッセージが溜まっているプロセス一覧、各プロセスのスタックなど、詳細な情報を取得することも簡単です。

稼働中のパッチ適用

最後に、止められないプロダクション環境を想定して、プロセスを停止せずにこのエラーを修正してみましょう。これも、Erlangならではの簡単さをお見せします。

次の内容でsrc/boom.erlを作成します。boom()という関数を呼ぶと、"BOOOOM!"という文字列を返すだけの簡単な機能です。

```
-module(boom).
-export([boom/0]).
boom() -> "BOOOOM!".
```

コンパイルして、稼働中のreleaseにbeamファイルをコピーしましょう。

```
$ ./rebar compile
  $ cp ebin/boom.beam➤
       rel/decho/lib/decho-1/ebin/
```

起動中のシェルでモジュールをロードすることで、稼働中のままエラーを修正できます(1行目のロードの丸括弧の前は小文字のエルです)。

```
> l(boom).
{module,boom}
> gen_server:call(decho_server, boom).
{ok,"BOOOOM!"}
```

これは新規モジュールだけでなく、ロード済みのモジュールにも適用できますし、何度もロードしなおすこともできます。これで、システムを停止せずに実行ファイルを入れ替えられることが確認できました。実際に、プロダクション環境で、無停止でのバグ修正などを行うことができ、高可用性に寄与するしくみの1つです。

まとめ

Erlang/OTPの高可用性という特徴を軸に、実際のアプリケーション作成を通じて、エラー時の挙動と稼働中の内部調査およびパッチ適用を見てきました。関数型であることの利点や分散システムといった華やかな側面とは違った、少々泥臭い内容に感じられたかもしれません。大規模な分散システムや、状態を多く持つような複雑性の高いシステムでも安心して利用でき、また状態を解析できるErlang/OTPの利点を感じていただけたら幸いです。

本稿では紹介できませんでしたが、高可用性のためには複数サーバ上で分散して動作することも必要です。理由は単純でハードウェアは故障するからです。たとえばErlang/OTPの一部であるMnesiaというデータベースを用いると、複数サーバ上でレプリカを作成し、シャーディングすることもできます。商用RDBMSの機能には劣りますが、ディスクへの永続化やACIDトランザクションもサポートしています。Erlang/OTPにはほかにも多くの機能、利点があります。興味を持たれた方はぜひ本稿で紹介した情報も追いかけてみてください。SD

▼図5　gen_server:call(decho_server, ham)の実行

```
23:45:04 <decho_server> {decho_server,handle_call,
                         [foo, {<0.663.0>,#Ref<0.0.0.56214>},
                          {state,undefined}]}
23:45:04 <decho_server> {decho_server,handle_call,3} ->
                        {reply, {ok,undefined}, {state,foo}}
```

Java SE 8のラムダ式で変わるJavaプログラミングスタイル

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

来年3月にリリース予定のJava SE 8ではλ(Lambda、ラムダ)式が導入され、関数型のプログラミングの記述がやりやすくなりました。本稿はJavaを使って関数型プログラミングスタイルで記述する方法について解説します。

Javaにラムダ式が来たよ

Javaでは、関数を値として扱うことはできません。そのため、今までであれば、「関数型プログラミング再入門」という特集にJavaが加わることはなかったと思います。

しかし、今年2014年3月にリリースが予定されているJava SE 8では、ラムダ式が導入されて、関数を式として記述できるようになります。型として直接関数が扱えるようになるわけではないので、記述がスッキリしない部分もありますが、関数型のプログラミングスタイルの記述が非常にやりやすくなります。そのおかげで、今回の特集にもJavaを混ぜてもらえるようになったわけです。

本稿では、Java SE 8のラムダ構文の簡単な概要と、そこから変わるJavaでのプログラミングスタイルについて解説しようと思います。

Java SE 8のラムダ式

それでは、簡単にラムダ式の構文を解説します[注1]。

今まで、たとえばSwingでのイベントハンドラを設定するときに、**リスト1**のような匿名クラスを使っていました。

これが、Java SE 8のラムダ式を使うと**リスト2**のようになります。

匿名クラスから、インターフェース名とメソッド名を省略して、引数部と処理部を「->」で結んだ形です。引数の型は省略できます。また、引数が1つの場合は括弧()も省略できます。さらに、処理部が1行のときには中括弧も省略できます。

最終的に省略できるものをすべて省略すると、**リスト3**のようになります。

最初の、匿名クラスを使った書き方に比べると非常に簡潔になりました。このように、ラムダ式は匿名クラスの簡易記法ということができます。実際には、より効率がよくなるようにコンパイルされるので単なる簡易記法ではありません。

▼リスト1 今までのSwingでのイベントハンドラを設定

```
ActionListener al = new ActionListener() {
    @Override
    public void actionPerformed
            (ActionEvent e) {
        showMessageDialog(f, "こんちは");
    }
};
```

▼リスト2 Java SE 8でラムダ式を使ったSwingのイベントハンドラ

```
ActionListener al = (ActionEvent e) -> {
    showMessageDialog(f, "こんちは");
};
```

▼リスト3 リスト2の記述を省略したもの

```
ActionListener al = e ->
    showMessageDialog(frame, "こんちは");
```

注1) 詳しい説明はこちらのエントリをご覧ください。http://d.hatena.ne.jp/nowokay/20130824#1377300917

関数型インターフェース

ここで、ラムダ式の対象になる型が、関数を表すような型ではなく通常のインターフェースになっていることに注意してください。ラムダ式は、「実装すべきメソッドが1つだけのインターフェース」のオブジェクトとして割り当てることができます。

このような、「実装すべきメソッドが1つだけのインターフェース」のことを、関数型インターフェースといいます。例に挙げたActionListenerインターフェースでは、実装すべきメソッドはactionPerformedメソッドだけなので、関数型インターフェースとして扱えます。ほかには、Runnableインターフェースも実装すべきメソッドがrunメソッドだけなので関数型インターフェースとして扱えます。

既存のインターフェースのほかにも、引数1つで戻り値をとる関数を表すFunctionや、引数をとらず戻り値をとる関数を表すSupplierなどのインターフェースが追加されています。

メソッド参照

ラムダ式の代わりに、定義済のメソッドを指定することもできます。たとえば、**リスト4**のようなメソッドがあるとします。

そうすると**リスト5**のようにメソッドを直接指定できます。

ラムダ式を匿名クラスの簡易記法と説明しましたが、実際のコンパイル時には、ラムダ式の内容がstaticメソッドとして定義されて、メソッド参照が渡されるという形に展開されます。

▼リスト4　通常のメソッド記述

```
public void buttonOnAction(ActionEvent ae){
    showMessageDialog(frame, "こんちは");
}
```

▼リスト5　直接指定したメソッド記述

```
ActionListener al = this::buttonOnAction;
```

Streamでコレクション処理

Java SE 8でラムダ式が導入される大きな目的となったのが、Streamによるコレクション処理です。for文などのループを使った逐次的なコレクション処理では、効率的な並列化ができないということから導入されました。

たとえば、生徒の一覧から福岡の人を抜き出して得点の平均を得る、という処理をこれまでのfor文を使って書くと**リスト6**のようになります。

これをStreamで記述しなおすと**リスト7**のようになります。

「生徒の一覧から福岡の人を抜き出して得点の平均を得る」という、やりたいことに近い記述になっています。

Streamを扱うには、Streamを生成するソース、Streamから別のStreamを生成する中間操作、Streamから結果を取り出す終端操作の3種類の操作を考える必要があります。次にまとめてみます。

Streamのソース

Streamでの処理を行うには、まずStreamオブジェクトを取得する必要があります。このとき、Streamの元になるものをソースと呼びます。

ListなどCollectionインターフェースには、

▼リスト6　通常のメソッド記述

```
int total = 0;
int count = 0;
for(Student s : students){
    if("福岡".equals(s.pref)){
        total += s.score;
        ++count;
    }
}
double ave = (double)total / count;
```

▼リスト7　直接指定したメソッド記述

```
double ave = students.stream()
        .filter(s -> "福岡".equals(s.pref))
        .mapToInt(s -> s.score)
        .average().orElse(0);
```

streamというメソッドが追加され、このstreamメソッドを呼び出すことでStreamを得ることができます。また、配列を処理するStreamは、Arraysクラスのstreamメソッドで得ることができます。

このほかにも、IntStreamインターフェースのrangeメソッドで数値列のStreamを得たり、BufferedReaderクラスのlinesメソッドで各行を処理するStreamを得たりと、Streamがあれば便利というところにStreamを生成するメソッドが用意されています(**表1**)。

中間操作

Streamから別のStreamを生成する操作を中間操作といいます。中間操作には、Streamを条件によってフィルタリングするfilterメソッド、Streamの値を別の値に変換するmapメソッド、要素数を制限するlimitメソッドなどがあります(**表2**)。

このような中間操作をつないでいくことで、Streamの処理を行います。

終端操作と畳込み

Streamは、最終的にそれぞれの値についてなんらかの処理を行うか、値を束ねて1つの値に変換する必要があります。このような処理を終端操作をいいます。

終端操作には、それぞれの値を処理するforEachメソッドのほかに、集計を行って1つの値を取り出すメソッドがあります。集計メソッドには、例でも挙げたようなaverageメソッドや、個数を数えるcountメソッド、そして、さまざまな集計を行うcollectメソッドがあります(**表3**)。

collectメソッドには、おもにCollectorsクラスのメソッドを使ってCollectorオブジェクトを渡します。Collectorsクラスには、文字を連結するjoiningメソッド、Listに変換するtoListメソッドなどがあります(**表4**)。

このように、コレクションを操作して1つのオブジェクトにまとめることを畳込みといいます。最初に挙げた、for文を使う**リスト1**では平均をとるのに合計と個数を集計するための変数が必要になりました。for文を使う畳み込み処理では、中間状態を管理する変数が必要になります。しかし、Streamでの畳み込み処理ではそのような中間状態が隠されて、どのような畳み込みを行うかだけを記述すればよくなります。中間状態の操作でバグを混入させてしまう

▼表1　Streamソースになるメソッド

Streamソースになるメソッド
Collection#stream()
Arrays.stream()
IntStream.range()
BufferedReader#lines()
SplittableRandam#ints()

※#はインスタンスメソッド、.はstaticメソッド

▼表2　中間操作のメソッド

中間操作	説明
filter()	要素を選別する
map()	別の値に変換する
sorted()	整列する
distinct()	重複を省く
skip()	要素をとばす
limit()	要素数を制限する

▼表3　終端操作のメソッド

メソッド	説明
toArray	配列を取得する
sum, min, max, count	集計を行う
anyMatch, allMatch	条件にあてはまるか
forEach	要素ごとの処理を行う
findFirst	最初の要素を得る
reduce	一般的な畳み込み
collect	再利用可能な畳み込み

▼表4　Collectorsクラスのメソッド

メソッド	説明
toList	Listを取り出す
joining	文字列を結合する
groupingBy	グループ化してまとめる
summarizingInt	整数の集計を行う

ことがよくありますが、そのような操作を記述する必要がなくなることで、バグを作りこんでしまう可能性を減らし見通しをよくすることができます。

並列処理

Streamを使った処理は、簡単に並列処理に書き換えることができます。サンプルであれば、streamメソッドでStreamを取得していたところを、parallelStreamメソッドに置き換えるだけで並列処理に変更できます。

for文を使った処理を並列化する際は、畳み込み操作に該当する部分の書き換えに気を遣います。Streamの処理では畳み込みの処理が隠蔽されているために、プログラマがほとんど意識せずに並列化が行えるようになるわけです。

Java SE 8で使えるプログラミングスタイル

ここまでStreamによる処理を見てきました。Streamによるコレクション処理は、Javaでのプログラミングスタイルを最も大きく変えると思いますが、ほかにもJavaのプログラミングスタイルを変えるような記述やライブラリがいくつかあるので、ここで紹介します。

制御構造の自作

ラムダ式を使うと、メソッドに処理を渡すということが記述しやすくなります。そうすると、制御構造を自作して使うことも楽になります。StreamのforEachメソッドもその1つといえます。

たとえば、**リスト9**のようなloopメソッドを用意します。IntConsumerインターフェースはint値を1つ受け取る関数として扱える関数型インターフェースです。

そうすると、**リスト10**のようにして、指定回数繰り返すような制御構造として使えます。

遅延実行

Haskellでは、関数の引数として渡した式は、実際にその式の値が使われるときに評価されます。このように、式の値が実際に使われるまで評価されないようなしくみを遅延評価といいます。

Javaでメソッドの引数として記述した式は、メソッドの呼び出し前に評価されてその値が引数として渡されます。この値がメソッド内で使われなかった場合には、式の評価は無駄な処理だったことになります。

たとえば、次のようにデバッグログでオブジェクトの内容を出力するコードがあるとします。

```
logger.debug(params.toString());
```

そうすると、デバッグ以外のときにはtoString()メソッドで生成された文字列は出力されず、まったく無駄な処理になってしまいます。このような場合、ラムダ式を使って式を間接的に渡すようにすると、値が必要になったときだけ式を呼び出すということが可能になります。

java.util.Loggerクラスのdebugメソッドなどでは、

▼リスト8　Streamを使った並列処理の例

```
double ave = students.parallelStream().
        .filter(s -> "福岡".equals(s.pref))
        .mapToInt(s -> s.score)
        .average().orElse(0);
```

▼リスト9　loopメソッド

```
static void loop(int count, IntConsumer proc){
    for(int i = 0; i < count; ++i) proc.accept(i);
}
```

▼リスト10　loopメソッドを制御構造として使う例

```
loop(5, i -> {
    System.out.println(i + "回目");
});
```

Stringの代わりにSupplier<String>を受け取るものが用意されているので、次のように記述できます。

```
logger.debug(() -> params.toString());
```

このようにすれば、実際にログ出力されるときだけtoString()メソッドが実行されるようになります。ラムダ式の導入で、こういった遅延実行の記述がやりやすくなります。

nullの排除

Javaではオブジェクトを持たないことを表すためにnullが使えますが、このnullはバグの温床にもなっています。また、nullに対応するためのコードも面倒なものになります。たとえば次のようなProductクラスがあるとします。

```
class Product{
  String name;
}
```

そして、キーに対応するProductクラスのオブジェクトを返すメソッドがあるとします。キーに対応するものがなければnullを返すとします。

```
Product findProduct(String key){
  return products.select("key=?", key);
}
```

このとき、Productを検索して、名前を大文字で表示するような処理は、次のようになります。

```
Product p = findProduct(key);
if(p != null && p.name != null){
  System.out.println(p.name.toUpperCase();
}
```

ここでnullのチェックを忘れてしまうと、NullPointerExceptionが発生する可能性が出てしまいます。とくに、p.nameのnullチェックは忘れたりサボったりしてしまいがちです。

関数型の言語によく見られるのは、nullを扱わず、HaskellのMaybeやScalaのOptionのように、値を持たない可能性がある型を用意することです。型として明示することで、予期せず値がないということが起きにくくなります。

Java SE 8からは、値を持たない可能性があることを表すOptionalクラスが導入されました。Optionalクラスを使うとfindProductメソッドは次のようになります。

```
Optional<Product> findProduct(String key){
  return Optional.ofNullable(
    products.select("key=?", key));
}
```

そうすると、Productを検索して名前を大文字で表示するような処理は次のように書くことができます。

```
findProduct("1")
  .map(p -> p.name)
  .map(String::toUpperCase)
  .ifPresent(System.out::println);
```

このOptionalのえらいところは、mapメソッドで値を取得していく限りは、ずっと値を持つことが保証されるということです。つまり、Optionalを使うと、値があったりなかったりするOptionalの外の世界と、値があることが保証されたOptionalの内側の世界とを分離できるわけです。

まとめ

ラムダ式が導入されるJava SE 8では、これまでとは違った関数型のプログラミングスタイルが使いやすくなります。

もちろん、後付けであることからいろいろイケてない点も多くあり、元から関数を扱うことが前提で設計されている言語のようにはスマートに記述できないというのも事実です。しかし、関数型スタイルの記述が気軽に行えるようになることで、Javaのプログラマにも関数型の考え方が広がってくるのではないかと期待しています。Java SE 8以降では、外部ライブラリにも関数型の考え方を取り入れたものが増えてくるでしょう。

Javaでのプログラミングスタイルが、変わっていくように感じています。SD

第7章 Rubyで関数型脳を育てる方法とは？

Writer るびきち（http://www.rubyist.net/~rubikitch/）／Twitter@rubikitch

関数型プログラミング再入門の終章はRubyです。これまで、さまざまなプログラミング言語が生まれてきましたが、第1章で紹介したおなじみのLispからRubyに至るまでコンピュータの発展の歴史を表しているかのようです。その軸になるのは、関数型プログラミングというわけです。本章でそのエッセンスを堪能してください。

RubyはLispだった!?

Rubyはオブジェクト指向言語として知られていますが、実は関数型言語の影響をものすごく強く受けています。しかもスクリプト言語なのでオブジェクト指向と関数プログラミングを身近なものにしてくれます。本記事ではRubyを通して関数型脳をインストールしていただきます。

関数型言語の元祖はLispです。Lispにはさまざまな変種があり、Lispから影響を受けた言語が数多く存在します。Rubyもその1つで、一部の人からは冗談交じりながらも「MatzLisp」と呼ばれています。開発者であるまつもとさん本人が「RubyはLispをベースとしている」と宣言しています。

> Ruby is a language designed in the following steps:
>
> * take a simple lisp language (like one prior to CL).
> * remove macros, s-expression.
> * add simple object system (much simpler than CLOS).
> * add blocks, inspired by higher order functions.
> * add methods found in Smalltalk.
> * add functionality found in Perl (in OO way).
>
> So, Ruby was a Lisp originally, in theory.
> Let's call it MatzLisp from now on.
>
> [ruby-talk:179642]より引用

「Ruby＝Lisp－マクロ－S式＋オブジェクト指向＋ブロック＋Smalltalk＋Perl」といったところでしょうか。「マクロとS式のないLispなんて！」とかLisperから怒られそうですが、Lispの考え方はしっかりと受け継がれています。Lispのとっつきにくさを排除しつつ、Lispの魅力を普通のプログラマに広めた言語とも言えます。

Rubyに見られるLispっぽい特徴

それではRubyに見られるLispっぽさをちょっとずつ挙げていきましょう。

Rubyはすべてのコードが「式」であり、値を持っています。制御構造ですら例外ではありません。たとえば「if」はC言語系列では文ですが、Rubyでは最後に評価した値を返します。なお、この例で使ってるシンボルもLispが由来です。

RubyもLisp同様「最後に評価した値」を返り値にする式が多いです。メソッドの返り値にもあてはまるため、`return`が省けるのもうれしいです（**リスト1**）。

Lispは強力なマクロによるメタプログラミングができます。関数を定義する関数や制御構造が定義できます。そのおかげでLispを問題領域に特化した言語（DSL：Domain Specific Language）に変化させられます。とくにリーダーマクロを使えば好き勝手な言語を構築できてしまいます。

Rubyにはマクロが存在しないものの、メソッドを定義するメソッドや「eval」があるのでメタプログラミング可能です。しかもRubyはもともと可読性の高い構文なのでリーダーマクロなどという難しい道具は不要です。マクロがなくても実用的なDSLが構築できるのはRailsやRakeで実証済です。Ruby そのものがDSLになっているのです。Ruby版MakefileのRakefileを示しますが、初見でも意味は読み取れるでしょう(**リスト2**)。

Lispには関数を呼び出す関数**funcall**・**apply**がありますが、Rubyにも同等の機能があります(**リスト3、4**)。**Object#__send__**はシンボルで指定されたメソッドを呼び出します(動的メソッド呼び出し)。また、配列の前に「*****」を付けてメソッドを呼び出せば、配列の各要素が分解されてメソッドの引数に渡ります(**xmpfilter**を使えば、**# =>**の後に式の値が注釈され、出力結果は**# >>**で表示されます)。

そしてLispの象徴とも言えるラムダ式ですが、Rubyにもちゃんと存在します。ラムダ式は関数プログラミングの中核を担います。そのまま使うこともできますが、絶妙な形で融合されています。

ラムダ(lambda)で遊んでみる

ではRubyのラムダ式を見てみましょう。ラムダ式は次の文法になります。

```
lambda {|args, ...| body ... }
lambda do |args, ...|
  body
  ...
end
```

Rubyをご存じならば、見てのとおり以降で紹介するブロックと同じ構文です。むしろ、ラムダ式を表現するのにブロックを使っています。なお、Ruby 1.9からは次の文法も受け付けます。

```
->args, ... { body ... }
```

「**->**」を回転させればλに見えるとのことですが、そう思ってください。**->**をλとみなせばλ計算のλx**x+1**を**->x {x+1}**と表現できるので、自然にラムダ式として読み下せます。

ラムダ式を呼ぶには、メソッド呼び出しの**()**の代わりに**[]**を使います。Ruby 1.9から「**.()**」でも呼び出せます。

▼リスト1　if式・メソッドの返り値の例

```
def iftest
  if 4%2 == 0
    # return :evenとも書ける
    :even
  else
    :odd
  end
end
iftest # => :even
```

▼リスト2　シンプルなRakefileの例

```
task :default => ["hello"]
file "hello" => ["hello.c"] do
  sh "gcc -o hello hello.c"
end
```

▼リスト3　funcall相当の例

```
Math.sin(0)                              # => 0.0
Math.__send__ :cos, 0                    # => 1.0
[:sin, :cos].map{|m| Math.__send__(m, 0)} # => [0.0, 1.0]
```

▼リスト4　apply相当の例

```
def vsystem(*args)
  puts "$ #{args.join ' '}"
  system(*args)
end
vsystem "ruby", "-v"
# >> $ ruby -v
# >> ruby 2.0.0p247 (2013-06-27) [x86_64-linux]
```

もちろんラムダ式はクロージャーになっています。**リスト5**の例で示すように、外側のローカル変数**z**もラムダ式から参照できます。

ブロック!

そして、Rubyを語るうえでは絶対欠かせないのが独特のブロック構文です！　一見とっつきにくいラムダ式・高階関数を一気に身近なものにしてくれます。ブロックを使いこなしてこそRubyらしいプログラムになり、関数型の恩恵をも受けられるようになります。

Lispは何個でも関数オブジェクトを引数に持てますが、関数引数を持つ多くのLisp関数は1つの関数引数しか持ちません。そこで、1つの関数引数に特化した構文を考案しました。それがブロックです。

ブロック付きメソッドの呼び出しは次のようになります。ラムダ(**lambda**)はその一例に過ぎないことがわかります。ブロック付きメソッドの構文は2種類用意されています。意味は同じですが結合の強さが異なります。**{}**のほうは式で、**do〜end**のほうは制御構造という感じがします。実際、**end**はクラス・モジュール定義、制御構造の終点に使うので、**do〜end**によるブロックは自作制御構造というニュアンスが見て取れます。

```
recv.method(args, ...) {|bargs, ...| body ... }
recv.method(args, ...) do |bargs, ...|
  body
  ...
end
```

おそらく、最初に出会うブロック付きメソッドは**each**でしょう。これは配列などのコレクションの各要素においてブロックを評価するメソッドです。そして、次節で紹介する関数プログラミング的なメソッドは**each**を使って定義されています。**リスト6**に、配列の各要素を表示するコードを示します。

ブロックはラムダ式の化身です。つまり、ブロックの代わりにラムダ式を渡すことも可能です。ブロック付きメソッドの最後の引数に「**&**」をつけてラムダ式を渡せば、そのラムダ式をブロックとして扱ってくれます(**リスト7**)。もちろんこんな書き方はRubyとしてはお勧めできません。ブロックを別のメソッドにそのまま渡すのに**&**引数が使えます(**リスト8**)。

ラムダ式ももちろんオブジェクトなので、Lispのように複数のラムダ式を引数に渡せます。Rubyではめったに登場しません。恣意的な例ですが、関数(Rubyでは厳密な意味で関数は存在しないが、トップレベルのメソッドは関数のように扱えるので便宜的に関数と呼ぶ)**result_formatter**はヘッダと計算の2つのブ

▼リスト5　ラムダ式の例

```
z = 3
f1 = lambda {|x,y|x+y+z}
f2 = ->x,y{x+y+z}
f1[3,4]  # => 10
f2.(3,4) # => 10
```

▼リスト6　eachメソッドの例

```
[1,2,3].each {|x| p x }
[4,5,6].each do |x|
  p x
end
# >> 1
# >> 2
# >> 3
# >> 4
# >> 5
# >> 6
```

▼リスト7　ラムダ式をブロックにする例

```
[1,2,3].each(&->i{p i})
# >> 1
# >> 2
# >> 3
```

▼リスト8　ブロックを別のメソッドに丸投げする例

```
def myeach(obj, &block)
  obj.each(&block)
end
myeach([1,2]){|x| p x }
# >> 1
# >> 2
```

ロックを受け取り、整形した文字列を作成するものと定義します。result_formatterでは、2つのブロックを評価し、改行で結合するだけです(リスト9)。

これではLisp臭がきついので、よりRubyらしくするにはリスト10のようにクラスを定義し、ブロック付きメソッドを立て続けに呼びます。ブロック付きメソッドがあたかもブロックの名前を示しているように見えるので、より読みやすいコードになります。1つのメソッドにつき1つのブロックしか持てませんが、自分自身を返すメソッドをチェーンすることで事実上複数のブロックを渡せることを示しています。

関数型を感じさせるメソッドたち

Rubyで関数プログラミングをするにはラムダ式の化身であるブロックが鍵であることは示したとおりです。eachはブロックを使っているものの、所詮は単なる繰り返し処理メソッドに過ぎないので命令型の考え方です。関数型は抽象度をさらに高めます。

Enumerable#map

まずはコレクションの各要素の値を2倍にした配列を返す処理を考えます。double_aryはeachを使った命令型プログラミングによる関数です。結果を返す空配列resultを用意し、eachで各要素を2倍にした値をresultに追加(Array#<<)し、resultを返しています。

この関数は「要素を2倍にする」処理と「新しい配列に結果を追加する」処理に分かれています。前者は問題によって変化しますが、後者は汎用的に使えると思いませんか？—— そうです。後者を「コレクションの各要素に対して**共通の計算**をしてその結果の新しい配列を返す」処理として抽出したのがEnumerable#mapです。mapは前者の「共通の計算」をブロックとして受け取

▼リスト9　複数のラムダ式を引数に取る例

```
def result_formatter(header_block, result_block)
  format "%s¥n%s", header_block[], result_block[]
end
puts result_formatter(->{"Result:"}, ->{Math.sqrt(2)})
# >> Result:
# >> 1.4142135623730951
```

▼リスト10　メソッドチェーンで複数ブロックを扱う例

```
class ResultFormatter
  def header(&block)
    @header = block    # インスタンス変数にブロックを記憶(まだ評価しない)
    self               # 自分自身を返すのがポイント
  end

  def result(&block)
    @result = block
    self
  end

  def process
    format "%s¥n%s", @header[], @result[] # ここで記憶したブロックを評価する
  end
end

puts ResultFormatter.new.header{ "Result:" }.result{ Math.sqrt(2) }.process
# >> Result:
# >> 1.4142135623730951
```

り、ブロックの評価結果を結果の配列に渡すのです(リスト11)。

Lispのmapcarではlambdaを書く必要がありますが、Rubyのmapはその必要がありません。結果的にとても短く簡潔に表記できます。

Enumerable#select

今度は配列の中から偶数だけを取り出した新しい配列を返す処理を考えます(リスト12)。select_evens_aryは命令型で書いた関数です。同様に、ifの条件式以外は汎用に使えることがわかります。Enumerable#selectは条件式をブロックで受け取り、条件を満たした要素のみを取り出した配列を返します。Lispではremove-if-not相当ですね。

「&引数」を使うことでさらに短く書けます。&引数はラムダ式以外にもメソッド名のシンボルも渡せます。具体的には「{|x| x.METH}」→「&:METH」という感じです。メソッド名を直接渡してるところで、もっともっと関数型な雰囲気が出てきましたよね！

関数型メソッドあれこれ

ここでmapとselectを取り上げましたが重要なことがあります。それは、これらが繰り返しメソッドであることを忘れさせてくれることです。mapは「コレクションに写像(map)を適用する」、selectは「条件を満たす要素を取り出す」と読め、どこにも「繰り返し」に関する言葉は出てきません。行いたい処理そのものがメソッド名になっており、そこに必要な情報を指定するだけで仕事をこなしてくれます。ほかにもEnumerableモジュールには最大値・最小値を求めたりソートをしたり要素数を求めたりなど、eachを使った便利メソッドがたくさん定義されています。Enumerableはeachメソッドの抽象度を上げるものです。

各要素の総和・総積を求めるのはそれぞれ、reduce(:+)・reduce(:*)を使います(リスト13)。Enumerable#reduceはLispのreduce同様に畳み込みを行うメソッドです。eachを使ってこの処理を書くと各要素において累積値を更新していき、最後に累積値を返すことになります。

eachによる命令型プログラミングだと、変数の破壊的変更(map,selectに出てくるArray#<<)や値の更新(reduceに出てくる+=・*=)が出てきます。関数プログラミングはこれらの処理が出てこないうえ短くなるので、プログラムが明確になりバグが少なくなります。

ブロックは繰り返し処理に用いることが多かっ

▼リスト11　Enumerable#mapの例

```
def double_ary(ary)
  result = []
  ary.each{|x| result << x*2 }
  result
end
double_ary [1,2,3]              # => [2, 4, 6]
[1,2,3].map{|x| x*2 }           # => [2, 4, 6]
```

▼リスト12　Enumerable#selectの例

```
def select_evens_ary(ary)
  result = []
  ary.each{|x| result << x if x.even? }
  result
end
select_evens_ary [2,3,4]        # => [2, 4]
[2,3,4].select {|x| x.even? }   # => [2, 4]
[2,3,4].select(&:even?)         # => [2, 4]
```

たため、かつてイテレータと呼ばれていました。しかし、一度しか評価されないブロックも多くなってきたため、イテレータとは呼ばれなくなりました。たとえばopenは開いたファイルに対する処理をブロックで記述できてcloseを省けるなどです。ほかにも条件によって評価したりや文脈を変更するなどの用途があります。関数プログラミングの観点ではやはり「イテレータ」としての用途が多くなります。

命令型から関数型へ

それでは実際に命令型で書かれたプログラムを関数型にしてみましょう。題材は「完全数」です。

完全数とは、真の約数（自分自身を除いた約数）の総和が自分自身に等しい自然数のことです。言い換えると、すべての約数の総和（自分自身含む）が自分自身の2倍になる自然数です。たとえば、28＝1＋2＋4＋7＋14なので28は完全数です。

Nが完全数であるかを判定するメソッドであるInteger#perfect?を定義してみましょう。perfect?それ自体は、約数の総和（sum_of_factors）がNの2倍に等しいと書いてるだけなのですが、sum_of_factorsは命令型で書かれています（リスト14）。

約数は1から√Nまでの各整数においてNが割切れるかどうかをチェックする方法をとっています。もし割切れるのならば、その数（i）とN/iが約数になるので両者を加えています。ただし、平方数（√Nが整数になる数）では両者が等しくなるため、除外しています。

完全数は6 28 496 8128 33550336 8589869056と続いていくのでプログラムは正しく動作しています。

sum_of_factorsを関数型で考えると、次の4ステップとなります。

①1〜√Nで割切れる数iをすべて求め、
②i→[i, N/i]の写像を適用して全約数を得る
③約数から重複を取り除き（平方数対策）、
④総和を求める

①は剰余が0になる条件式をブロックで指定したselectを使います。selectは取り上げたばかりなので問題ありません。a..bは範囲オブジェクトでEnumerableのメソッドが使えます。

②の写像適用は普通にmapを使えばよさそうな気もしますが、実はうまくいきません。整数→配列の写像になっているので、mapでは配列の配列（2次元配列）が返ってきてしまいます。そこで、ブロックの値を返り値の配列に結合するflat_mapという亜種を使います。これは、1段階平滑化したmapなので、flat_map{〜}はmap{〜}.flatten(1)と同じです。

③で複数の同値要素を1つにするにはArray#uniqを使います。

④は出てきたばかりのreduce(:+)です。

▼リスト13　Enumerable#reduceの例

```
[1,2,3,4].reduce(:+)  # => 10
[1,2,3,4].reduce(:*)  # => 24
```

▼リスト14　完全数を求める（命令型）

```
class Integer
  def perfect?
    sum_of_factors == self * 2
  end

  private
  def sum_of_factors
    sum = 0
    (1..Math.sqrt(self)).each do |i|
      if self%i == 0
        sum += i
        sum += self/i unless i == self/i
      end
    end
    sum
  end
end
## テスト
1.perfect?                          # => false
28.perfect?                         # => true
496.perfect?                        # => true
8128.perfect?                       # => true
33550336.perfect?                   # => true
# 1〜10000までの完全数をリストする
(1..10000).select(&:perfect?)
# => [6, 28, 496, 8128]
```

これらをまとめると次のようになります。perfect?は同じです。テストも正しく動作します。

・関数型にしたsum_of_factors

```
def sum_of_factors
  (1..Math.sqrt(self)).
  select{|i| self%i == 0 }.
  flat_map{|i| [i,self/i]}.
  uniq.reduce(:+)
end
```

関数型のコードを見てみると、データを次々に加工していることがわかります。しかも圧倒的に短くなりました。

遅延評価

そしてもうひとつ、関数プログラミングを語るうえで忘れてはならないのが遅延評価です。Ruby 2.0からはEnumerable#lazyの登場により遅延リストがすぐに扱えるようになりました。Rubyの関数型色がますます強まったことを意味します。

遅延評価の何がうれしいのかというと、とてつもなく大きな列、それどころか無限個の列をも扱えるようになることです。Haskellなどの関数型言語では無限個の列を扱うのは普通ですが、Rubyでも可能になったのです。

巨大な列を扱う

まずは、わざとらしいですが遅延評価とは何かを感じ取っていただくために、1〜100,000の二乗から二桁の平方数を取り出してみましょう(**リスト15**)。馬鹿正直な方法(**Mmap**)では1〜100,000のそれぞれの二乗を計算した配列から10以上100未満を取り出すことです。次に遅延評価を使う方法(**Mlazy**)では、必要な部分だけが計算されるようになります。そして最後は命令型による方法(**Meach**)です。

一番簡潔なのは馬鹿正直な**Mmap**ですね。しかし、不要な部分を全部捨ててしまうので無駄な計算があります。かといって、せっかく素晴しい関数型のメソッドがたくさん用意されているのに命令型プログラミングに戻ってしまうのはとてもとても悲しいことです。命令型は確かに無駄を省いてくれますが、関数型の簡潔性は

▼リスト15　3種類の方法によるベンチマーク

```
require 'benchmark'
Benchmark.bm(10) do |b|
  b.report("Mmap") do  # 馬鹿正直に100000までの二乗を求める
    # 10未満の平方数を捨て、100未満の平方数だけを取得する、残りは捨てる
    (1..100000).map{|n| n*n}.drop_while{|x| x < 10}.take_while{|x| x < 100}
    # => [16, 25, 36, 49, 64, 81]
  end
  b.report("Mlazy") do # 遅延評価を使って無駄な計算を省く
    (1..100000).lazy.map{|n| n*n}.drop_while{|x| x < 10}.take_while{|x| x < 100}.to_a
    # => [16, 25, 36, 49, 64, 81]
  end
  b.report("Meach") do # 原始的な命令型プログラミング
    ret = []
    (1..100000).each do |n|
      x = n*n
      next if x < 10
      break unless x < 100
      ret << x
    end
    ret  # => [16, 25, 36, 49, 64, 81]
  end
end
# >>                 user     system      total        real
# >> Mmap        0.000000   0.000000   0.000000 (  0.006329)
# >> Mlazy       0.000000   0.000000   0.000000 (  0.000033)
# >> Meach       0.000000   0.000000   0.000000 (  0.000028)
```

捨てがたいですね。

そこで登場してきたのがMlazyです。Enumerable#lazyはコレクションを遅延オブジェクトに変換し、map、drop_while、take_whileなどの配列を返すメソッドが遅延オブジェクトを返すようになります。そして、to_aで「じゃあ、今計算しろ！」と言うのです。その結果、必要な部分だけmapに渡るようになり、簡潔性と速度を両立できました。遅延オブジェクトのオーバーヘッドがあるので、さすがにMeachよりも多少遅いですが簡潔性を考えれば十分許容範囲です。やることは、コレクションの後にlazyを、最後にto_aをつけるだけです！

無限の列を扱う

次は無限リストを扱ってみましょう。Prime.instanceは素数を無限に生成するEnumerableです。同じ数字が4つ続く素数のうち最初の5つを列挙してみましょう。数字を文字列化したものが正規表現/(.)¥1{3}/にマッチしたものを取り出せばいいのです（リスト16）。

Enumerable#selectは全要素を要求してきます。そのため無限リストに対しては.lazyをつけないと無限ループに陥ってしまいます。現時点ではMlazyはMeachの2倍遅いですが、今後の高速化に期待しましょう。

終わりに

Rubyで関数プログラミング、いかがだったでしょうか？　関数プログラミングにおける関数とは手続きではなくて数学的な意味の関数が起源なので、数学の話題が中心になってしまいました。もちろん応用範囲は数学に限りません。たとえば、巨大なファイルの先頭数行を読む場合だったり、TwitterのStreaming APIを扱うときにも遅延評価が役立ちます。ブロックと遅延評価をうまく活用して、関数型の簡潔性と命令型の速度を両立させてください。

ここでは紹介しきれませんでしたが、lambdaの亜種にProc.newがあります。これはreturnの挙動が異なります。Enumerator.newは任意のコレクションを定義できる面白いメソッドです。

今月Rubyは21歳になります。開発当初からブロックが用意されており、関数型を強く意識した言語設計になっています。そしてマルチコア時代の今、関数プログラミングが人気が出ています。そういう意味においてRubyは先見の明があるのではないでしょうか。今後の発展が楽しみです。SD

▼リスト16　素数列を求める例

```
require 'prime'
require 'benchmark'
Benchmark.bm(10) do |b|
  b.report("Mlazy") do
    Prime.instance.lazy.select{|n| n.to_s =~ /(.)¥1{3}/}.first(5)
    # => [11113, 11117, 11119, 22229, 23333]
  end
  b.report("Meach") do
    ret = []
    Prime.instance.each do |n|
      ret << n if n.to_s =~ /(.)¥1{3}/
      break if ret.length == 5
    end
    ret  # => [11113, 11117, 11119, 22229, 23333]
  end
end
# >>                 user     system      total        real
# >> Mlazy       0.010000   0.000000   0.010000 (  0.011531)
# >> Meach       0.010000   0.000000   0.010000 (  0.004820)
```

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。

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

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

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

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

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

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

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

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

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

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

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

サーバ/インフラエンジニア養成読本 仮想化活用編
Software Design編集部 編
定価 1,980円+税　ISBN 978-4-7741-5038-3

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

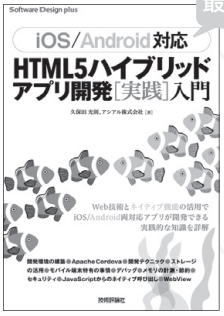
Apache[実践]運用/管理
鶴長 鎮一 著
定価 2,980円+税　ISBN 978-4-7741-5036-9

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

最新刊！

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

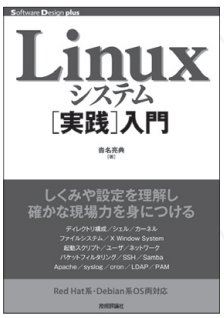


久保田 光則、アシアル㈱ 著
A5判・384ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-6211-9

最新刊！　最新刊！

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

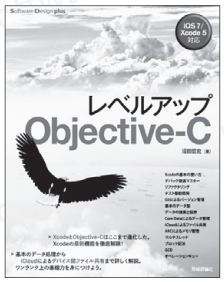
ニコラ・モドリック、安部 重成 著
A5判・336ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-5991-1



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

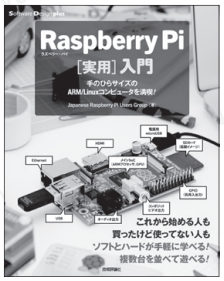
小飼 弾 著
A5判・200ページ
定価 2,080円(本体)+税
ISBN 978-4-7741-5664-4

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



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

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



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

水野 操、平本 知樹、神田 沙織、野村 毅 著
B5判・128ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-5973-7

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

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

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

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

第2特集 目利きによるトレンド予測

2014年 IT業界はどうなるのか?

原点に戻りながら次の一手を考える

さあ、2014年が始まりました!

2013年はどうだったでしょうか。安倍内閣による政策によって景気は上向き傾向にあるそうですが、皆さんの感覚はいかがでしょう。2020年の東京オリンピック開催決定、富士山の世界文化遺産登録、4月からの消費税率8%決定、そしてたくさんの偉人がこの世を去りましたが、ダグラス・エンゲルバート氏が逝去されたのも昨年でした。

本特集では、2014年がエンジニアにとってどんな年になりそうかを、各分野で活躍するエンジニアの方々にそれぞれの視点で予測していただきました。どういったテクノロジに注目し、どんな意識を持って過ごすのか。どうぞ参考にしてください。

※著者プロフィールにある 2013マイヒット とは、2013年に著者個人にヒットしたキーワードを挙げてもらっています。

第1章 ネットワーク／インフラ技術はどうなるのか

ITインフラを支える技術動向について2013年を振り返ると、ネットワークにも押し寄せてきた仮想化の波を現実的に検討するフェーズに入ってきたように思います。SDN(Software Defined Network)は Interop Tokyo 2013 のテーマの1つとして、大いに注目を集めていました。ネットワーク管理を担うエンジニアには、これまで以上にソフトウェアに対するスキルが求められることになるでしょう。

運用に関しては、2013年もさまざまなネットワークサービスが展開され、集客量の多いイベントやソーシャルゲームでのアクセススパイク、スマートフォンの普及によるモバイル機器からのアクセス急増への対策が考えられたことでしょう。ハードウェアではストレージデバイスのボトルネックを解消するためのSSDやioDriveといった製品が実績を上げています。クラウドサービスでは相変わらずAmazon Web Service(AWS)の強さが際立っていたように思います。

本章では2014年のネットワーク／インフラ技術について、4人のエキスパートに展望してもらいました。

インフラエンジニアのあらたな始まりの年

Writer 田中 邦裕

インフラエンジニアのニーズ

インフラエンジニアが注目されている昨今、フロントエンドのプログラマを中心に、オープンソースやクラウドを活用しながら、インフラレイヤーにまで自分の可能性を広げる動きが広まっています。また、アプリケーションの開発サイクルが短くなり、かつオンライン化するなかで、インフラエンジニアに対するニーズも高まってきました。

このような背景を元に、2014年は真のインフラエンジニア元年になり、新たなキャリアパスを見つける人が増えるのではないかと考えています。

インターネットの初期においては大学の研究室などで、サーバ構築からネットワーク設計までやらされる時代がありましたが、90年代後半からはプログラマ、サーバエンジニア、ネットワークエンジニアといったように役割が細分化されました。それだけに、インフラエンジニアへの注目は、ある意味原点回帰なのかもしれません。

ただ、昔と変わってきたのは、プログラマからインフラエンジニアを目指す人が増えているという点です。以前はサーバを準備するというと、購入して、設置して、セットアップして、というプロセスを経なければならず、設置方法や相性の解消など物理的なノウハウが重視される部分もありました。

現在ではクラウドサービスをAPIでたたき、プログラマブルにセットアップできてしまうという時代になり、プログラミングの能力が必要とされるようになりました。また、オープンソースの活用が広がり、障害対応やチューニングのために、ソースコードを改変しなければならないことからもプログラミング能力は重要です。

このようなことから、プログラマはインフラレイヤーで求められている人材であり、hbstudy[注1]やqpstudy[注2]といったインフラエンジニア向けの勉強会への参加などを通じて、あらたなキャリアパスの模索をお勧めしたいと思います。

設備もプログラマブルに

ちなみにインフラレイヤーの話で言うと、DCIM（Data Center Infrastructure Management）というデータセンターをITで効率的に管理するという取り組みに注目が集まっています。今までのデータセンターは、空調やUPS、発電機などのさまざまな設備が独立して稼働していたのですが、IPやIEEE1888などのプロトコルで設備を相互接続し、物理的なレイヤーまでプログラマブルにコントロールさせる時代が来ています。

たとえば、Webアプリケーションの負荷が下がればサーバの電源を落とし、そのラックのファンの回転数を下げ、空調設備の稼働を低減させる、といったような、フロントエンドのアプリケーションからファシリティまでを一括でプログラミングするといった時代が来ると思います。天気が良く太陽光発電量が増えるとサーバを起動してバッチを回すなんてことになるかもしれません。

これからは世界中のモノがインターネットに接続され、ソフトウェアで制御される時代がやってきます。そのような時代に、プログラマがどんどんとインフラ分野へ進出し、ソフトウェアで世の中のしくみを良くしていく動きに期待しています。

そのきっかけの年として2014年が記憶されるようになればと思っています。SD

注1）http://heartbeats.jp/hbstudy
注2）http://sites.google.com/site/qpstudy

クラウド型とベアメタル型の双方をシームレスに扱える技術が必要に

Writer 伊勢 幸一

昨年までのトレンド

昨年までクラウドコンピューティングやSDNのように仮想技術を利用した実装とサービスがIT技術トレンドの主流を占めていました。クラウドコンピューティングは仮想コンピューティングインスタンスを必要なときに必要な数だけ利用し、実際に消費したリソース分の対価を支払うだけで良いという、従来の設備投資概念を根底から覆す斬新なサービスです。またネットワークの仮想化オーバーレイはそういった仮想インスタンスを自由自在に運用するための通信基盤を形成する必要不可欠な技術であり、SDNがクラウドコンピューティングと共に大きく注目されたことは当然でしょう。

このようなオンデマンドコンピューティングリソースサービスは利用者の設備投資負担を軽減したことで、スタートアップ企業や中小企業によるインターネットビジネスへのチャレンジを可能とし、さまざまなサービスとビジネスが生まれ、成長したことは周知の事実です。

安定したサービスシステムはクラウドからベアメタルへ

しかし、クラウドサービスによって乱立したオンラインサービス、とくにソーシャルゲームやソーシャルメディアの多くは2012年頃から次第に淘汰され、現在では爆発的人気を得て成功したサービスやゲームだけが生き残っている状況です。その反面、多くのユーザに支持されなかったサービスであったとしても、すでにいくらかのユーザが付いてしまっているサービスを収支に見合わないからといって乱暴に停止するわけにもいきません。そこで、クラウド上の安価なインスタンスにサービスサーバを集約し、コストを最小限に抑えて維持するだけになっているケースも見受けられます。

成功し、生き残ったサービスはその後もユーザ数を順調に増やし続け、ますます増大する大量のリクエストを処理するため、サービスシステムを仮想オーバーヘッドのない従来型のベアメタル環境へ移行するケースも増えてきました。すでに何年も継続しているサービスであれば処理負荷と収益の見込みがある程度予想できるため、クラウド上での流動的な仮想インスタンスにこだわる必要性がなくなり、逆にサーバのコストと性能が明確に想定できるベアメタル環境のほうがサービスを維持拡大しやすいのです。

実際、筆者の周辺でもある企業が展開する複数のサービスの内、ユーザ数が増えないサービスをまとめてクラウドへ移し、当たったサービスだけを専用物理サーバホスティングへ移動するということが日々行われています。昨年、OpenStackでは仮想環境からベアメタル環境へのマイグレーション機能がサポートされ、またUbuntu Linuxでも物理サーバをクラウド上の仮想インスタンスのようにコントロールするMaaS(Metal as a Service)プロジェクトがリリースされたのも、前述のような仮想サーバからベアメタルサーバへのマイグレーション要求に対応するためでしょう。

今後のトレンドは

今後必要とされる技術とはこのような流れを受け、従来のクラウド環境で慣れ親しんだ優れたユーザインターフェースによってベアメタルサーバ環境上で自由自在にインスタンスを立ち上げ、ミドルウェアやアプリケーションをデプロイするための技術です。たとえばPXEブートやIPMIリモートコンソールとサーバコントロール機能を活用するCobblerなどと、サーバ環境をリモートから一括で整備するPuppetやChefといった管理ツールとのフェデレーションが必要であり、今後さまざまな実装や運用事例が重要視されてくると考えられます。SD

スケーラブルな仮想データセンター構築技術に注目

Writer 小宮 崇博

2013年を振り返って

2013年にIT業界において賑わった技術は、ほとんどすべてが「簡単」「スケーラブル」「コストパフォーマンス」の3つの言葉でまとめられています。ブロケードではシステムをスケーラブルに拡張するためのネットワーク基盤としてデータセンター(以降、DC)のファブリック化を推進しており、クラウドサービスプロバイダを中心に多くのユーザに導入してきました。また、仮想ネットワークファンクションとして、Virtual ADXとVyatta vRouterを提供しました。新技術としては、OpenStackとファブリックの連携のためのNeutronプラグインも提供しましたし、ADXシリーズADCのL3 DSR機能向けのプラグインもYahoo Japan!殿の事例とともに発表しました。

2014年に来るのはこれだ!

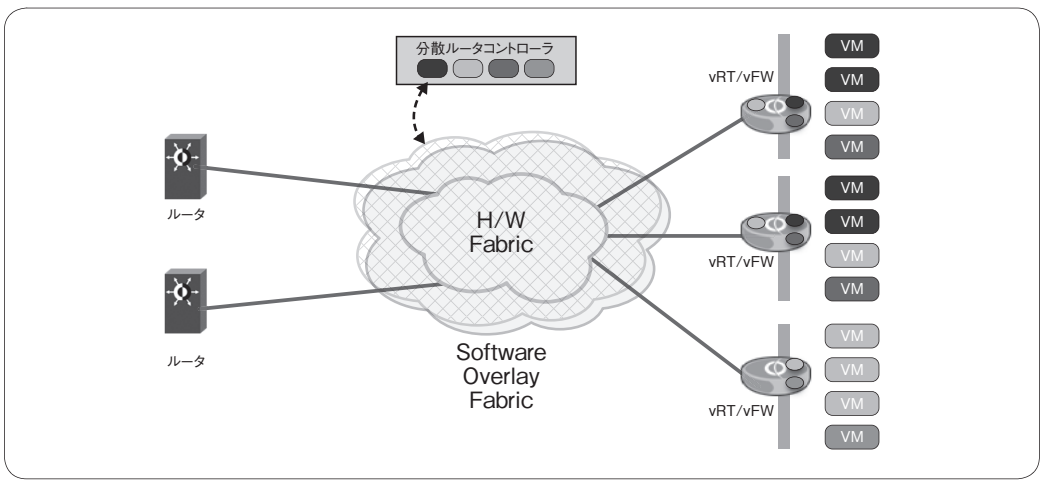
2014年に実現する技術潮流としては、スケーラブルな仮想データセンターの構築があります。前段と同様に「DC基盤」と「DC基盤における機能」の2つに分けて説明しましょう。

2014年のDC基盤はマルチテナント性においてスケーラビリティが向上します。これはファブリックがL2スケーラビリティを向上するTRILL Fine-Grained-Labeling(FGL)技術やVXLANオーバレイに対応することで実現します。

同様に、2014年のDC基盤の各種機能も大きく進化します。仮想スイッチだけではなく、仮想ルータや仮想ロードバランサの実用化の年になるでしょうし、ファイルサービスも仮想化しスケールアウト構成になっていくでしょう。インテルはデータパスの操作を改良し、遅延やパケット処理の改善を行うソフトウェアキットを提供していますが、これに対応する製品群が出てくることになります。また、仮想ルータなどの仮想アプライアンスが乱立するのは管理コストを引き上げるため、イーサネットファブリックのようなソフトウェアファブリック技術も出てくることになるでしょう(**図1**)。

ソフトウェアによる基盤機能のさらなるシンプル化、高性能化そしてスケーラビリティを実現できるようになるのが2014年だと言えるでしょう。SD

▼図1　ソフトウェアファブリック



スマートフォン時代のインフラはレスポンスタイムが生命線

Writer 佐野 裕

今は改めて言うまでもなくスマートフォンの時代と言えます。ネット系企業において、2014年はまさにスマートフォンに最適化したITインフラ環境が広く求められそうです。

スマートフォン時代のITインフラの特徴を簡単に押さえておくと、PCの場合はLANケーブルや無線LAN経由の比較的安定した通信環境の下でリッチなコンテンツが一度に大量にやり取りされるのに対して、スマートフォンでは3G回線、LTE回線、無線LANと品質の異なる多様な通信環境の下で小さな単位のデータが頻繁にやり取りされます。スマートフォンユーザは、手持ち無沙汰なときに頻繁にサーバにアクセスしますが、レスポンスが悪いとイライラして二度とサービスを使ってくれなくなります。

こういったスマートフォンの特性を押さえたITインフラを考えると、レイテンシ(ネットワークの遅延)を最大限抑えることと、サーバ側のディスクI/Oがボトルネックにならない対策を徹底することに尽きると思います。

ここで強調したいキーワードは「インメモリ」「キャッシュ」そして「CDN」です。

「**インメモリ**」は、サーバに搭載されているメモリ上にすべてのデータが載っている状態を指し、ディスクI/Oを発生させずに高速なレスポンスが実現します。幸いなことに近年は1台のサーバに非常に大量のメモリを積むことができます。たとえば、1Uサーバのような高密度サーバでも24枚のメモリが搭載できる機種がありますが、こういったサーバに8GBのメモリを24枚搭載すればメモリ搭載量は192GBにもなります。これだけ大きなメモリ搭載量を満たすようなデータを用意するのは結構難しそうです。

「**キャッシュ**」は、リクエストに対してレスポンスの結果を保存しておいて、繰り返し発生する同等のリクエストに対して保存しておいた結果からレスポンスを返すことで高速な応答を実現するしくみのことです。レスポンスが遅くなりそうなところの前にキャッシュを仕掛けることで高速なレスポンスが実現します。たとえば検索サーバがあったとして、検索サーバの前にキャッシュサーバを置くことで、よくある検索結果は検索結果を生成するオリジンサーバに都度問い合わせなくても、キャッシュサーバが保存したデータから応答を返すことでレスポンスを高速化します。スマートフォンはPCと違って各ページで表示できる情報量が限られるため、必然的にコンテンツ提供者は厳選したデータのみを用意して表示することになります。ですから、PC用サイトと比べると通常キャッシュヒット率が向上します。

そして最後に「**CDN(コンテンツデリバリーネットワーク)**」です。CDNとは世界各国の主要ISP配下にCDN業者のキャッシュサーバを配置することで、世界のどこにいても[注1]低レイテンシでコンテンツを取得できるしくみです。日本国内限定のサービスであればレイテンシが大きな問題になることは考えづらいですが、日本から遠く離れた国ではどうしてもそれなりのレイテンシが発生します。少しでも快適なサービスを提供するためには、世界のどこにいても低レイテンシでユーザがコンテンツを取得できるCDNの利用は必須でしょう。

以上、非常に当たり前のことを記しましたが、それでもあえて今回3つのキーワードを強調したのは、スマートフォンの時代はPCの時代よりもレスポンスタイムに関する要求がシビアなため、このような当たり前なことを確実に実現することが当たり前のように求められるためです。今回の話しでピンとこない方はぜひ一度スマートフォンを持って通信環境の非常に悪い国に旅行してみてください。SD

注1) 特殊な事情がある国向けのCDN配信は別料金な場合があります。

第2章 ソフトウェア開発はどうなるのか

プログラミング言語、フレームワーク、開発手法などソフトウェア開発技術は常に進歩しています。日々の作業の効率化や課題解決のためには、それらの技術動向を押さえておく必要があります。

そこで本章では、ここ数年、注目技術と言われている開発技術／手法について、最新動向や現場での活用事例についてまとめました。具体的には、「仕様が固まりつつあるHTML5などWeb標準技術の影響」「現場で試行錯誤が続けられているアジャイル開発の今後の行方」「2013年リリースされたAWSの新サービスの中から今年とくに使われそうなサービス」「一時注目を集めたプログラミング言語Goの実用性」の4つのテーマを取り上げます。

自分のプロジェクトや仕事に活用できそうな技術かどうか、その見極めの参考にしてください。

Web標準技術で いろんなモノの「幅」が広がっていく

Writer 川田 寛

HTML5は2014年に、W3C勧告化される予定ですが、すでに多くの技術がエコシステムを持ち、ビジネスにも影響を与えています。今回は筆者が注目している技術を中心に紹介します。

RIAの代替技術で、OSSが存在感を強める

FlexやSilverlightのようなプラグインを活用したRIA技術は、Webのエコシステムから外れたことで一気に衰退しました。そして、2012年頃から広がりだしたSPA(Single-page Application)は、その空いた椅子に収まるだけでなく、RIA以上のポテンシャルを得ようとしています。

これまで大規模なSPAは、ベンダ製品の活用が近道でした。しかし、Gruntを中心とした開発ツールが充実化し、AngularJSなどのフレームワークが一般化したことで、寄せ集めのOSSが高い競争力を獲得しました。AppCacheやIndexedDBなどを用いたオフライン処理を大規模な開発で扱う場合、フレームワークはとても有益な手段です。今後、RESTと親和性の高いJava/Ruby系開発者の間で、より広く普及することが予想されます。

性能改善手法が、スマートデバイスを使いやすくする

現在のWeb技術での性能の議論は、依然としてDOMアクセスやJavaScript記述など、既存技術を比較的低いレイヤで議論するのが中心です。しかし、Web標準側でも「Resource Priorities(リソースの優先付け)」や「prefetch/prerender(コンテンツの先読み)」、Page Visibility APIを用いた省電量化などが議論され、安定志向のInternet Explorerでさえもサポートを始めました。

こうした取り組みは、コンテンツの内容にも性能評価の観点で定量的な判断基準を生じさせます。今後の性能最適化はさらに上のレイヤへ広がることが予想されます。とくにこれらのアプローチはスマートデバイスのような貧弱な環境下で高い効果を発揮するため、WebページのUX向上のために必須になると考えています。

デザインから、マウスという前提が姿を消す

デバイスの多様化が進み、Surfaceのようなデバイスの違いをあいまいにするような端末も広がりを見せ、ビューの横幅から入力デバイスを判断するのが怪しくなってきています。ビューのマルチデバイス化は、横幅を検出しサイズに合ったユーザビリティを提供する「レスポンシブWebデザイン」が一般化し、十分に成熟しました。

しかし、タッチパネルなどの入力デバイスについては標準化が難航し、ビュー側より約3年遅れで実装が進んでいます。タッチパネルやマウス、ペンなどの入力デバイスは、Web標準では「ポインター」という抽象的なデバイスとして扱うことが推奨されています。これからは、ナビゲーションバーのhoverを使ったメニュー表示など、マウスを前提としたデザインの定石は姿を消し、ポインターという概念で得られる最高のユーザビリティについての議論が本格化するのではないかと睨んでいます。

通信技術は、上位レイヤの進化を促す

双方向通信を実現するWebSocketは、JavaのProject Avator、.NETのSignalRと、新しいアーキテクチャ作りの基礎技術として存在感を強めています。Ajaxのようなただのデータ通信ではなく、RPC/リソース同期と1つ上のレイヤで技術進化の道が模索され続けるでしょう。

ブラウザ間通信を実現するWebRTCは、これまではネイティブが中心だったOTTを、Webで実現するという新たな道を作りました。今後は、通信サービスというレイヤで、技術進化の方法が検討されるでしょう。SD

アジャイルの浸透とモデリングの一体化、さらに上流工程との融合が進む

Writer 羽生田 栄一

ソフトウェアの開発スタイルが今年大きく変わるということは残念ながらなさそうですが、とはいえ、いくつかの可能性の芽は誰の目にも明らかなくらいには大きくなりつつあります。それが2014年から2015年にかけて花開いてくれることを祈りたいと思います。

まずアジャイルプロセスが一般的になってきたことをひしひしと感じます。1年前まではアジャイルプロセスって何か教えてくれというコンサルや教育が多かったのですが、昨年からは、一般的なユーザ企業・ITベンダーから「アジャイルで開発進めたい」「Dev&Opsを導入・実践したいのでメンバーを教育してほしい／コンサルに入ってほしい」「ある組織全体にアジャイル開発のバーチャルプロジェクトやってほしい」というニーズが広がってきています。併行して、組織的にScrumマスターの認定資格を取られるケースも増えています。なんちゃってアジャイルを回避して、人中心のチーム実践が回せるよう頑張ってほしいと思います。

一方、もうアジャイルなんて昔から当たり前にやっているよというチームにも変化の兆しが見えてきました。昨年の記事でも言及したDDD（ドメイン駆動開発）を前提とするモデリングとアジャイルが一体化した開発への本格的な取り組みです。Scrumは所詮はプロジェクトの枠組みに過ぎず、エンジニアリングに関してきちんとした指針が望まれています。デザインやモデリングをアジャイル開発と一体化する、つまり業務や問題領域について業務関係者と開発者が「対話・会話」をとおして分析・理解する活動と、設計・開発する活動を入れ子にすることで、健全なシステムを目指す。TDD（テスト駆動開発）やリファクタリングといった技術も、業務の理解の深まりとともに更新されるドメインモデルやアーキテクチャの整合を取り続けるために利用する。改めてオブジェクト指向の現代的な意義がここに来てようやく根付きそうです（DDDアジャイルの際、JavaよりもScalaが有用であることはいうまでもありません。3年目の真実）。

さらにこうした枠組みの中で、いままで「上流」という言い方で誤解されていた、“そもそもそのサービスやプロダクトのバリューは何で、それでビジネスをどう行うのか”に関するモデル化も併せて行おうという動きが明確になってきました。たとえば、ビジネスモデルキャンバスとピクト図解を組み合わせる手法やGOAL（WHY?）、ACTORS（WHO?）、IMPACTS（HOW?）、DELIVERABLES（WHAT?）をマインドマップで共有するインパクトマップの手法、アジャイルを意識したBABOK（ビジネスアナリシス知識体系ガイド）などが登場し、アジャイルとモデリングもここに来て「上流から下流まで」というより「ビジネスとソフトウェア」「サービスと運用」をパラレルに実践できる上記のような思考フレームワークと、クラウド関連のツール・環境といった実践プラットフォームが整備されてきたといえます。

また、組込み・製造業の分野ではシステムズエンジニアリングへの注目がじわじわ上がってきていますが、SysMLといったモデリング言語の導入を通してトレーサビリティを追求するのは、あくまでも取っかかりに過ぎません。システム思考を実践しつつ創造的なサービスやプロダクトを作りだす、という可能性に一部の組織は気づいており、そこではデザイン思考とシステムマネジメントをうまく融合するといった、いままで予想もしなかった取り組みが始まっています。その一方で、形式手法やMBD（モデルベース開発）といった技術による信頼性・安全性の最大限の確保への追及も自動車・ロボットなどの分野で続いており、AlloyやSpin、Rodinといった手軽で安価な軽量形式手法ツールの登場とともに一般にも普及していくと予想されます。SD

2013年に続々登場したAWS新プロダクト、その本格利用は2014年から!

Writer 鈴木 宏康

2013年の振り返りと2014年の展望

昨年も本誌で2013年の展望的な記事を書かせていただきました。そのときは、クラウドを扱ううえでの心構え「クラウドアーキテクティング原則」を紹介し、ITエンジニアをとりまく環境は「クラウドを使ってみる」フェーズから「クラウドを使いこなす」フェーズに遷移していくと結びました。

実際に昨年の仕事を振り返ると、EC2(仮想サーバ)だけでなく、いつもは我々から提案していたS3(ストレージサービス)やSQS(キューサービス)の利用も、顧客のほうから先に相談されることが多くなっていました。また、6月の「AWS Summit Tokyo 2013」ではAWS上に構築された多くの事例が発表され、さらに、11月に行われたAWSの世界的なイベント「re:Invent」のセッションでは日本の事例としてNTTドコモの「しゃべってコンシェル」の講演が行われました。まさに、1年かけて「クラウド(AWS)を使いこなす」フェーズに遷移したと言っても過言ではないと思います。

昨年もAWSでは多くのアップデートがありました。その中でもCDPへの影響度が高いと思われるものを簡単に紹介させていただきます。

多機能CDN「CloudFront」

CDNサービスのCloudFrontに「独自SSL証明書対応」「POST、PUTなどのメソッドのサポート」といった機能が追加されました。今後、動的コンテンツに対するプロキシー的な利用例も増え、CDPの動的コンテンツを処理するパターンのいくつかは、CloudFrontを利用する形に進化していくと思います。

クロスリージョンに関するアップデート

リージョンからリージョンへのバックアップ(同期)を容易にする機能も、いくつか追加されています。

- AMIのリージョン間コピー
- RDSのリージョン間スナップショットコピー
- RDS(MySQL)のクロスリージョン・リードレプリカ

これにより世界展開するシステム構築やディザスタリカバリ対策が、今まで以上に容易に実現できます。今後、その手の事例が増えてくると思います。それに伴い、運用保守に関するパターンも、より充実されるはずです。

新プロダクト「Kinesis」「AppStream」

上述の「re:Invent」で、「Kinesis」「AppStream」という新プロダクトが発表されました。Kinesisは大規模なストリーミングデータをリアルタイムで処理する完全マネージド型サービス、AppStreamは大量のリソースを使うアプリケーションやゲームをクラウドからストリーミングできるようにするサービスです。これらは一昨年の「Redshift」の発表と同じくらいインパクトのあるものだと考えています。

今年は、これらのプロダクトを利用するための準備がいたるところで行われるのではないかと思っています(事例が出始めるのは来年あたりでしょうか?)。これらのプロダクトを中心としたCDPもできていくはずです。

2011年3月にAWSの東京リージョンができて、もうすぐ3年が経ちます。そのときから存在するプロダクトは、かなり使いこなされてきたとは思いますが、その間も新機能や新プロダクトは、どんどん発表されています。それらも使いこなすために、AWSに関わるエンジニアにとっては今年も楽しく激しい1年になるのではないでしょうか? SD

Go言語の使いどころ

Writer 山本 泰宇

2009年にGoogleからGo言語が発表されて4年[注1]が経ちました。DockerやHekaのようにGoで開発されたプロダクトも登場し、さまざまなライブラリもそろいつつあります。サイボウズでも、自社クラウドサービスcybozu.comのインフラで利用しています。その経験から、Go言語の適用分野について考察します。

Go導入前の状況

cybozu.comではさまざまなツールを自社で開発して利用しています。バックアップ／リストア機能を含む高可用ストレージシステム、memcached互換でより高速かつ高可用なKVSのyrmcds[注2]、分散P2P技術を利用した自動障害回復システム、などです。

ストレージシステムのコア部分やyrmcdsはC/C++で開発されています。これらは性能が極めて重要であるため、CPUコア、メモリ、ストレージ、それにシステムコールを極限まで効率的に使えるC/C++が合っています。技法としても、ロックフリーなマルチスレッディングや非同期I/O、ストレージの先読みなどを駆使しています。

それ以外のプログラムの大半はPythonで開発しています。便利で開発効率は悪くないのですが、C/C++とPythonでは性能特性が違い過ぎるため、C/C++で書くほどではないけれどPythonでは満足できる速度ではない場面があります。また、pylintという静的解析ツールの遅さや不正確さにも不満がありました。

軽量言語とC/C++の狭間で

Goは強い静的型付けでプラットフォームネイティブな実行コードにコンパイルする言語です。C/C++と特性が似ていますが、Goにはランタイムのオーバーヘッドがあるため、とことんチューニングする目的で使っているC/C++の代替にはしたくありません。

ではPythonをGoで置き換えるかというと、全面的な置き換えはかえって開発効率が下がると判断しました。Goのライブラリは充実してきましたが、コンパイルが必要なため手軽さの面で劣るのです。具体的にはREPLがないのと、XMLやJSONをカジュアルに取り扱えないといった点が不便です。

以上を鑑みると、Goは軽量言語とC/C++の間を埋める使い方が良いでしょう。具体的には、ある程度の性能が求められ、手軽さよりもコンパイル時の静的解析が有用になる規模のプログラムです。GoはNode.js同様に非同期処理が得意ですので、ネットワークサーバにはとくに適しています。

Goの適用分野はどこまで広がるか

そんなわけで、Pythonへのわずかな不満からGoを新規のネットワークサーバや、既存のPython製のサーバを書き直す際に使うことにしました。卒直なところ、Go製のプログラムはまだ数えるほどで、適用範囲は限定的です。

ただし、最初からGoを使っていれば、もっと多くのネットワークサーバをGoで書いていただろうと思います。また、もし我々がC/C++に熟達していなければ、C/C++の代替物として学習コストが低いGoを利用していたかもしれません。

Goが得意とする分野ではNode.jsが先行している状況ですが、まだ勝負はこれからだと思います。SD

注1）http://blog.golang.org/4years
注2）http://cybozu.github.io/yrmcds/

第3章 OSとその周辺技術はどうなるのか

スマートフォン、タブレット、クラウドなど新しいデバイスやサービスが登場するたびに、その環境へ最適化することを求められるオペレーティングシステム。その変化は2014年も止まることがありません。

今回はLinux(RHEL、Debian、Ubuntu)とWindowsの識者4名にそれぞれのOS/ディストリビューションの動向や注目すべきOS周りの技術について聞きました。

ストレージ、クラウド、ARMなど共通するトピックはいくつかあるものの、コンシューマ向け、エンタープライズ向けなどそれぞれの立場により、注目している技術は違います。伝統的な技術に課題解決の糸口を見いだしている人、最新技術に大きな変化を感じている人などさまざまです。また、企業のバックアップを受けているもの、コミュニティ主導で開発が進められるもの、その開発の運用形態によっても、目指すところや開発スピードに違いがありそうです。

今のうちに、自分が利用しているOSについて、変化の方向性や課題を把握し、準備を進めておきましょう。

時代の要求に応え進化し、再び注目を集めるテープメディア

Writer 小島 克俊

2014年に来そうな技術として、インフラ設計のプロでも見落としがちなテープ技術に注目しています。テープ製品の生産量が過去3年間、伸びの傾向にあることをご存じでしょうか。コストと長期保存の観点からも、テープによるデータ記録が再び重要になると予想しています。

コストと保存期間ではテープが有利

ストレージは高速かつ大量にデータを記録し、同時に低コストを要求されることが増えています。普段使いの容量だけではなく、中間計算結果やアーカイブもあるので実際に必要な容量は増え続けます。Red Hat製品ではRed Hat Storage Serverがその解決策として利用されています。

数PBクラスの保存を意識したストレージであっても必ず物理的な記憶メディアがあります。SSD、HDD、テープの記録のコストを比較してみます(**表1**)。秋葉原価格でみるとテープはダントツに低コストです。

SSDやHDDはメディアそのものの部品点数が多く、故障のリスクが高いためRAIDによる冗長化が必要です。RAID装置に電源が入っていないとメディアの故障に気づくことすらできません。機器の償却が終わる5年で電源を止めるとデータは保証されません。長期間に渡って記録を保管することを考えると素朴な作りのテープはHDDよりも有利です(**表2**)。

▼表1　記録メディアの価格

媒体	1TB単価
Tape	1,500円
HDD	5,000円
SSD	80,000円

▼表2　記録メディアの寿命(5年で通電停止を想定)

媒体	寿命年数
Tape	30年
HDD	5年
SSD	5年

テープも進化している

テープは小さな磁石の粒(磁性体)を記録に利用しています。最近ではバリウムフェライトを磁性体に使った製品が出荷されています。バリウムフェライトの結晶は六方晶という六角形の鉛筆をとても短くしたような形をしています。そして表と裏がS極とN極になるようなはっきりした性質を持っています。従来の球状や針状の酸化鉄や鉄の合金よりも、均質かつ密度の高いテープを作れる材料です。

テープが緻密に記録できるようになった結果、テープドライブも進化しました。ハイエンド機だと1秒間に数百MBのデータ転送をします。また、ライブラリ内でテープドライブは急加速、急減速で移動します。テープカートリッジのリール軸は1つだけで、ドライブがテープを高速に巻き取っていきます。動作音も「うぃーん(ドライブ移動)、がちゃがちゃ(テープ格納)、きゅいーん(テープ読み書き)」のような感じではありません。「しゅぱ(ドライブ移動)、がこ(テープ格納)、……(テープ読み書き)」です。

テープを記録メディアとして利用する場合、tarなどによる一時処理が必要でした。しかし最近では、LTFS(Liner Tape File System)というしくみが使われるようになってきました。USBメモリと同様にテープメディアをファイルシステムとして直接マウントできます。理論的には数PBのファイルシステムが使えます。

テープに関わる技術は地味ですが、ハイエンドのシステムにはすでに不可欠です。「最近のテープってバリウムフェライトですよね」とか「LTFSはそのままでも便利だけど管理方法も最適化したい」といった会話をしている方がいると、「さすが」と感じざるを得ません。SD

参考資料：JEITAテープストレージ専門委員会の資料
http://home.jeita.or.jp/cgi-bin/about/detail.cgi?ca=1&ca2=292

2014年はDebian 8への準備期間、より効率的な開発をめざして

Writer やまねひでき

2013年、予測ほど進展せず……

2014年を占う前に、2013年の予測を振り返ってみましょう。昨年は予想として「autopkgtest/DEP-8利用の進展」を挙げましたが、残念ながら芳しくありません。一応、リリースチーム側はautopkgtestを利用しているパッケージについて「unstableからtestingの移行日数を2日に減らす」というボーナスを提示しましたが、手元で更新されていくパッケージのchangelogを眺めている限りではautopkgtestを利用するパッケージが増加している感じはありません(そういえば筆者もまだ使っていません……)。そして、ライセンス記述の改善である「Machine-readable copyright information」についても作業量がある割にはその活用方法が見えず、新規パッケージについては対応が行われるものの、既存パッケージについては対応が遅々としている印象です。クラウド関連の対応はそれなりに行われています(OpenStack関連パッケージのアップロードがunstable/experimentalへ随時行われています)が、これといって目を引く進捗はありません(残念ながら筆者はあまり腕の良い占い師ではないようです ;-)

あぁ、とくに問題なくDebian 7"Wheezy"がリリースされたというのがありました。いったんリリースが終わると安定版リリースはほぼ手を加えないので、パッケージメンテナからすると存在を忘れてしまいがちですね[注1]。

2014年、Debian 8に向けて

さて、あらためて2014年を展望してみますと、2013年と違って確実に当たる出来事、つまりDebianのリリースは今年は残念ながらありません。代わりに「年末にかけてフリーズが始まる」ことは断言します。明るい話題としては、次のリリースに向けての障害となるリリースクリティカルバグ(RC bug)の数がある程度ハンドリングできるレベルまで「現在の時点で」軽減されていることです[注2]。これまではリリースに向けて開発バージョンのフリーズが始まってから大量のRCつぶしで大きく時間が経過して[注3]、リリースしたときには周回遅れのバージョン……ということがありましたが、RCが減ることでフリーズ期間が短縮され、よりタイムリーなリリース(おそらく2015年春)ができるようになるでしょう。

Debian 8で公式サポートアーキテクチャとして64bit ARMが入ることは、期待されていますが、実際そうなるかどうか微妙なところです。開発ターゲットには確実に入っているものの、ビルドサーバ群などの実機リソースを確保できるかどうかが見えていません[注4]。Debian 3.1 "Sarge"のときのamd64のように「公式リリースからは外れるが準公式扱い」という可能性もあります。

「開発者の興味の赴くまま」にアーキテクチャ/パッケージ数ともに拡大の一途をたどっていたDebianですが、それに見合うだけの人的/機械的リソースの増加が行われているかというと正直そうでもありません。既存の枠組みでの運用だけではなく、いかに自動化/省力化のしくみを改善/構築できるか、そして状況の整理と協力者を集められるかが今後の行方を占いそうです。SD

注1) その代わり、セキュリティパッチの作成などで難儀するのですが。

注2) http://bugs.debian.org/release-critical/ 参照。「Number concerning the next release」の数を確認してください(本稿執筆時で600程度でした)。Debianの安定版リリースはこの数を0にするまで行われません。

注3) Debian 7の場合は10ヵ月の間フリーズとなっていましたが、8では半分程度に抑えられるのではないかと期待しています。

注4) リリース対象となったとしても、ビルドエラーの山と戦わなければならないので正直気が重いところです。ほかのディストリビューションでも64bit ARM対応が行われますので、その成果が利用できるとは思うのですが、対応が必要となるサポート対象パッケージ数が段違いに多いので、Debian側で対応をがんばらねばならないものも多数出るでしょう。

2014年のUbuntuは多様な広がりを見せつつも、進化は堅実に

Writer あわしろいくや

2013年のおさらい

2013年のUbuntuも、良くも悪くもさまざまな注目を集めました。X.Orgに代わるMir[注1]の発表、Ubuntu Edgeの挑戦[注2]、通常リリースのサポート期間短縮[注3]、オーストリアのビッグブラザー賞受賞[注4]、13.10でのインプットメソッドの混乱[注5]などです。

国内では日本HPがCanonical Ubuntu Serverの取り扱いを開始した[注6]のが大きなトピックでしょうか。また、Canonicalの日本人社員が増えたのも印象深いです。

2014年はどんな年?

Ubuntuにとって2014年は重要な年です。正確にはLTS(Long Term Support)[注7]が出る偶数年はすべて等しく重要ではあるのですが。14.10でタブレットやスマートフォンなどのモバイルとデスクトップでユーザエクスペリエンスを同一にする意欲的な試みがなされるはずですが、当初の計画よりも進行が遅れ、現段階でかなりトーンダウンしています。

Ubuntu 14.04/14.10

13.10ではMirのX互換レイヤーであるXMirをデフォルトに採用すべく開発が進んでいましたが、マルチディスプレイ対応が不完全ということで採用は延期となりました。今の段階でも14.04では見送られる見通しです。ただし、14.04.1といったポイントリリースで、HWE(Hardware Enablement)スタック[注8]の一環でXMirに移行する可能性はあります。Unityもこれまでどおり7系列で、ベースとなるGNOMEのバージョンも3.8と変わらず、13.10とあまり代わり映えのしないものになりそうです。ただ、インプットメソッドに関しては修正が入る見込みですが、どのような変更になるのかは現段階では決まっていません。いずれにせよ13.10よりはずっと使えるものになるはずです。

本来の予定では14.10でMirとUnity 8に移行することになっています。しかし、XMirのデフォルト化が早くてここになるので、XMirデフォルトをやめるのか、Mirデフォルト化を15.04に先送りするのかが注目です。

Ubuntu Touch

スマートフォンやタブレット端末で動作するUbuntuをUbuntu Touchと呼んでいます[注9]。開発は進んでいますが、Ubuntu Touch搭載ハードウェアが発売されるほどのクオリティに達しているかはやや疑問です。いずれにせよ、今年もNexusシリーズにイメージをインストールするという使い方に大きな変化はなさそうです。

ARM ARM ARM

ARMサポートも引き続き強化される見込みです。13.10では滑りこみセーフ(あるいはアウト)なタイミングでARMv8サポートが入りました。ARMv8は平たくいえば64bit版のARMアーキテクチャです。そもそも現時点では実機がほとんどないものの、本年中にはいろいろ発売されるでしょうし、それを見越して14.04でのサポートが強化されることは確実と見ていいでしょう。

SD

注1) http://unity.ubuntu.com/mir/
注2) http://gihyo.jp/admin/clip/01/ubuntu-topics/201308/23
注3) http://gihyo.jp/admin/clip/01/ubuntu-topics/201303/22
注4) http://www.omgubuntu.co.uk/2013/10/ubuntu-wins-big-brother-austria-privacy-award
注5) 本誌に断続的に掲載してきたため、バックナンバーをご覧ください。
注6) http://h50146.www5.hp.com/products/servers/proliant/announcement/20131004/
注7) 通常のリリースでは9ヵ月サポートですが、LTSは5年サポートです。
注8) https://wiki.ubuntu.com/Kernel/LTSEnablementStack
注9) https://wiki.ubuntu.com/Touch

2014年のWindowsは「サーバ運用」「仮想ネットワーク」「仮想ストレージ」に注目

Writer 横山 哲也

2014年の変化は小さく、技術的には暫定的な変化にとどまると予想しています。ただし、技術が変わらないということは、安心してビジネスに使えることを意味するため、利用シーンが大きく変わる可能性はあります。Windowsに関して言えば、Windows 8.1がブレークするはずです。評判の悪かった8に比べ、8.1はおおむね好意的に受け取られているようで、まるでVistaから7への流れを見ているようです。

昨年は同様のテーマで、「仮想マシンの拡大」「仮想スイッチ(ネットワーク)の強化」「クラウドの本格利用」という3つのキーワードについて触れました。いずれも、ほぼ予想どおりに進行しています。クラウド利用の拡大や、サーバエンジニアとネットワークエンジニアのスキルシフトについても予想どおり進行し、サーバエンジニアとネットワークエンジニアの融合(またはスキルシフト)が始まりました。「何でもこなせるエンジニア」という意味で「フルスタックエンジニア」という言葉も生まれました。

ただし、仮想プライベートクラウドの本格導入は来年に持ち越しそうです。また、ストレージ機能についてはほぼノーマークでした。Windows Server 2012の記憶域プールの可能性には触れていましたが、Windows Server 2012 R2で既存のNASやSANと競争するほど進化するとは予想していませんでした。

2014年のテーマは引き続き仮想化とクラウドですが、サーバそのものよりも、サーバ運用、仮想ネットワーク、仮想ストレージが重要になると考えています。

Windows Server 2012でVMwareにほぼ追いついたHyper-Vは、Windows Server 2012 R2でVMwareを上回る機能も搭載してきました。しかしVMwareも負けていません。vSphere 5.5では多くの新機能が搭載され、機能的にはどちらを選んでもほとんど変わらない状態になりました。Hyper-Vが弱かったLinuxサポートも格段に強化され、LinuxのためにHyper-Vを使うという選択肢も不自然ではありません。Windows AzureでもLinux仮想マシンが積極的にサポートされています。

Hyper-VとVMwareの機能拡張の結果、2014年のサーバ仮想化市場は「ITシステム全体の統合運用管理」が差別化要因となります。Microsoft製品では、オンプレミスとクラウドの両方を統合管理できる「System Center 2012 R2」の存在感が高まるでしょう。

仮想マシンが一般的になると、仮想マシンを接続するためのネットワークも重要になります。ネットワーク管理者は仮想マシン管理者を兼任し、データセンターを仮想化するNVGRE(Microsoftなどが推進)と、VXLAN(VMwareなどが推進)の検討に入ります。

Windows Server 2012 R2ではストレージプールが拡張され、SSDを使った記憶域階層や、重複排除機能の強化など、便利な機能が多く追加されました。以前のソフトウェアRAIDよりもはるかに高度なRAID機能も搭載されており、高価なハードウェアRAIDを必要としないのも利点です。

iSCSIターゲットサーバ機能も充実しており、ローエンドSAN製品の代替としての実力を持っています。

また、Windowsのファイル共有プロトコルSMB 3.0はエンタープライズレベルでも十分実用的な機能と性能、そして信頼性があります。今後はHyper-V仮想マシンやSQL Serverデータベースファイルの配置場所として広く利用されるようになるでしょう。従来のSMBにあった「遅い」イメージは一新され、SMB 3.0ベースのNASの位置付けは変わります。2014年は、NetApp社などの「高機能エンタープライズNAS」が普及すると予想しています。SD

第4章 エンジニアの仕事のしかたを考える

数年前に活況を呈していたソーシャルゲームは、2013年は以前ほどの勢いはありませんでした。SIビジネスも常にインターネット上やイベントでいろいろと問題提起されながらも、相変わらず昔ながらの地味で気苦労の多い受託開発を続けています。新しいプログラミング言語、フレームワーク、サービス、ガジェットなど技術的には明るい話題に事欠かないIT業界ですが、エンジニアの働き方に目を向けると、先が見えない混沌とした状況です。

とはいえ、ITエンジニア(とくに最新の技術動向をウォッチしている弊誌読者)ならば、「日々研鑽した己の技術力を活かせる仕事で飯を食っていきたい」「変化の多いIT業界でもなんとか自分の居場所を見つけ、活路を見いだしたい」と願っているのではないでしょうか?

本章では、IT業界の将来について言及することの多い4名の方に、2014年にエンジニアが進むべき方向性について持論を述べてもらいました。自分なりの働き方や今後の道筋について考える際の参考にしてみてください。

ITのコモディティ化! そのときエンジニアがとるべき道

Writer 湯本 堅隆

2013年(昨年)に今後の未来像について記載したとき、次のような展望を述べました。

・人月を積み上げて価格を算出する方式では価格面でSaaSなどのビジネスには勝てない。しかし、大手に限って基幹系/公共系がクラウドに全部載ることは考えにくいので、影響は限定的。影響を受けるのは中小規模案件
・SIで生き残るには、どんな形にせよ元請で顧客と直接SIをやることでしか活路が開けない

1年が過ぎましたが、おおむねこの大局観は間違っていないと考えています。リーマン・ショックとクラウド台頭のダブルパンチで中小規模案件が減ったことで、システム/ソフトウェア開発業者の倒産件数は2012年に過去最悪になりました。2013年の数字はこれから帝国データバンクなどから発表されますが、好況の兆しが見られることはないと考えています。

SIと言えば、受託開発です。受託開発は「システム構築ビジネスのコメ」とも言える大きな存在ですが、人員をプロジェクトごとに割く必要があることや一括請負契約が多いことから、開発業者に請けるだけの体力が求められます。しかしながら、大きな流れとして案件単価は下落しており、人月単価も下落し続けています。

友人のSI勤務者曰く「どこもデフレだよ」とつぶやいていました。かつての価格水準は通用しなくなっている中で、その「人月デフレ」の象徴的事件が昨年ありました。クロノス(株)の取締役である山本大さんが昨年ブログに書かれていた「人月0円セール」です。0円で送り込まなければならない理由はさまざまでしょうが、1つの時代が終わるのだなと感じました。システム構築はクラウド技術によってコモディティ化したということです。

ITリソースが湯水のごとく使える時代が、すでにやってきています。技術で差別化するのが難しいからこそ、固有技術をさらに極めて頂点に立つという正統派なキャリアもあるでしょう。ただ、頂点に立てる人間はとても少ない。それを考えると、筆者のように技術的に凡庸なエンジニアは多能化することで自分の技術を「活かす技術」を身につけるべきだと考えます。

筆者は大企業から年商数億の中小企業に転職しました。この規模の企業のIT予算は年商の1%、もしくはそれ以下であることも少なくありません。情報システム部門など存在しておらず、請求書だけは出せないとマズイので伝票発行/集計だけを行う既成のシステムを入れているケースがほとんど。ITで会社の業務が変わることがまったく想像できない人だらけの世界に足を踏み入れました。

そんな世界で生き残るために、「身の回りのことをITを使って改善して、会社の経営に貢献する」こととは何かを探して、少しずつ内製の範囲を広くしていきました。そうしないと、自分の居場所を作ることはできませんでした。

こんな時代に求められるのは、ビジネスを俯瞰できて、新しいシゴトをデザインできる人です。IT技術を持っている人は、事業を変革させる/成長させる大きな武器をすでに持っています。その武器の使い道を、ぜひとも探してください。営業や経営がプログラムやシステムのことをわかるようになるより、プログラマが業務や経営戦略を理解できるようになるほうが、ずっと簡単。

専門分野でスキルを持っているだけでなく、異なる分野の高いスキルを持っている人をつなげて、その会社なりが抱えている問題に対して新しい何かを提案できる人が圧倒的に頼りになります。しかし、数は少ない。だからこそそんな存在をめざすというのが、日本の企業の99.9%を占める非IT系中小企業の海で揉まれて得た、生き残りの術です。SD

2014年は「さらに先に行く技術」と「足下のSIビジネス」が乖離する年

Writer 神林 飛志

まずはビジネスの概況から言うと、確実に景気は上向いています。結果としてSIビジネスの受注は好調のまま2014年に突入し、受注増の形でSIビジネスは進みます。その一方で人手不足がさらに深刻化するでしょう。現場では、プロジェクト推進力の維持が精一杯で、技術的なトレンドの採用はますます困難になっていくでしょう……なので読者の皆様は、現場にどっぷりつかるのではなく、距離をとりつつ「自己研鑽が正しい2014年」というのが結論ですね。

SIビジネスの延命措置の拡大

人繰りですべてを回すSIは受注さえできていれば、ビジネス自体は回ってしまいます。ユーザサイドの体制は、今までのリストラによるスタッフ部門の弱体化のボディーブローが効き過ぎて崩壊気味です。よって、お金は払いますのでSI屋さんへお任せスタンスがさらに拡大します。若干景気も良くなっているので、システム投資は少なくとも昨年よりは行われるでしょう。

受けるほうのSI屋としては、営業も頑張るはずです。案件は確実に増えるでしょう。そもそも若年層の人口減少の日本労働市場では、できるだけ生産性を向上させていかないと企業活動自体が窒息気味になるはずなのですが、SI屋さん的には、「とにかく回せ。システム作ってたたき込め」が優先モードで、生産性向上どころか力業SIに拍車がかかります。結果、人数不足のデスマが増えるでしょう。しかし残念ながらSIビジネス自体は回ってしまうわけです。本来では次のスタイルを目指すべきSIビジネスは景気の上向きとともに、「無理でもそのまま進め」の文字どおりの延命措置のデスマーチならぬゾンビマーチが主軸になる。そんな2014年でしょう。

技術の進歩は分散処理に一直線

その一方で、技術動向は完全に分散処理的なものに一直線です。現状の新技術はほぼすべてクラウドからしか出てきていません。今のクラウドはほぼ分散処理がベース・テクノロジーになりつつあり、理論と実践の試行錯誤が重ねられています。そして、その成果がオープンソース・研究論文・プロダクツとして公開されてきています。こういった分散技術はクラウド上では開発は進んでいますが、いまだSIで使うには十分な道具立てとして成熟してはいません。

しかし、Hadoopが帳(とばり)を上げてしまったことに加えて、ビッグデータというバズワードもこれらの分散技術の一般化を後押ししている状態です。新技術の採用動向としては無視するわけにはいきません。分散クラスタでのR&Dや実装・フレームワークはさらに雲霞のごとく出てくるでしょう。日本の技術者も近い将来、これらの技術を使っていくことが迫られるでしょう。

SIの延命とクラウド技術の進歩が乖離し始める

こういったSIビジネスの需要過多と安定供給不足のマーケットの状況と、クラウド技術の乖離は広がる一方なのが2014年です。回すことに精一杯の状態では「お願いだから安定してない新技術は使わないでね」が通常の現場になります。技術的な試行錯誤ができない状況は、新しい技術への追従のコストをさらに高めます。

さて、では業務系エンジニアはどうしたら良いか？ということになります。携帯ゲーム屋が雲行き怪しいどころか土砂降り状態では、SI業界からの転職脱出もまた難しくなりました。とはいえ仕事は降ってきます。仕事を「選びながら」被弾を少なくして、自己研鑽に時間を割くという方法を選択していく以外にちょっと道はなさそうです。**景気が良くなるからといって、早々都合良く物事が運ぶわけではありません。**今まで以上に、自分の選択肢を増やす努力が必要なのが2014年なのかもしれません。SD

2014年押さえておくべき技術

Writer 村上 福之

2014年に押さえておくべきモバイル技術について、Tizenとウェアラブルコンピューティングについて書いておきます。

Tizenは押さえておくべき

Tizenはぜひとも押さえておくべき、モバイルOSです。絶対に押さえておくべきです。なぜなら、**これほど出る前から期待されていないのに、大手がかかわっているOSもないからです**。Tizen Associationが頑張っているらしいのですが、コミュニティはさほど活発でもなく、開発環境も未だに整ってないですし、一応、それでも国内最大手のNTTドコモが2014年には出すとアナウンスはしています。

ソフトウェアエンジニアにとって、**これほど、伝説を作りやすいOSはありません**。このように「海外で開発されて、国内の大手が乗っかって、早くに失敗したプラットフォーム」といえば、「3DO」というゲーム機を思い出します。

3DOといえば、同様に当時としては珍しい映画的な手法で、ムービーをたれ流すだけのゲームなのに伝説的名作と呼ばれた「Dの食卓」という有名なゲームがありました。これも3DOで出したためかメディア露出が多く、いろいろな賞を取りました。その後、いろんなプラットフォームに移植され、それなりに話題を呼びました。ゲームの製作者であった飯野賢治さんも時代の寵児と呼ばれました。彼は2013年に亡くなった最も有名なゲームクリエイターでもあります。もし、Dの食卓が3DOで出ていなかったら、このような伝説を生まなかったでしょう。

おそらく、Tizenでアプリを作る人は非常に少ないですし、多少おかしなものでも審査が通るでしょうし、ドコモのマーケットでトップを取るのも容易でしょう。さらに、一応、サポートしていないAPIがいろいろありますが、HTML5ベースでアプリを開発できますから、非常に学習コストが低いです。あなたのソフトウェアエンジニア人生で名前を残し、伝説となるなら、Tizenしかないのではないでしょうか?

あなたの人生の伝説のために、押さえておくべきOS、それはTizenです。

ウェアラブルコンピューティング

ウェアラブルコンピューティングもまた、押さえておくべきテクノロジだと思います。2013年中に発売されたメジャーなウェアラブルデバイスはGoogle Glass(一般販売ではありませんが)、Galaxy Gear、Sony SmartWatchです。これらはすべてOSがAndroidです。AppleからiWatchと呼ばれているものが出るとか出ないとか言われていますが、今のところ、よくわかりません。

Google Glass以外は今のところ、アプリケーション開発環境が載っていません。Google Glassのほうは、Google Mirror APIを使ったGlasswareアプリとGDK(Glass Developers Kit)を使ったアプリがあります。前者はJSONを使ったRESTful APIですし、後者は通常のAndroidのアプリケーションにAPIを拡張したものです。つまり、ウェアラブルコンピューティングの多くの実装技術は今後も、既存のテクノロジを流用したものになる可能性が高いです。

しかし、Googla Glassは表示領域が640×360ピクセルしかありません。Galaxy Gearも320×320ピクセルしかありません。通信が多いとすぐ電池がなくなります。そんなわけで、ウェアラブルコンピューティングにおいて今大事なのは、限られたリソースと画面サイズで実装するテクノロジとノウハウになってくると思います。

そんなわけで、2014年もおもしろいテクノロジがでてくるのではないかと思います。SD

次が見えない今だからこそチャンスはある

Writer 清水 亮

本当の意味での「Ubiquitous Computing」への理解

パーソナルコンピュータの次にどんな時代が訪れるのか、実はすでに名前がついています。それが「Ubiquitous Computing(ユビキタスコンピューティング)」です。このアイデアもパーソナルコンピュータと同じく、XEROXのパロアルト研究所で生まれました。半導体技術の躍進によるダウンサイジングの結果生まれたパーソナルコンピュータからさらに進み、これまで使われることのなかったものにまでコンピュータが入り込むことで、あらゆる事物がシームレスに接続されて1つの統一体のようにコントロールされる。これがユビキタスコンピューティングの時代です。

ところがこの10年というもの、日本ではこの言葉の意味が誤解され、単に携帯電話にコンピュータが入った、という現象のみを指すようになってしまったのは非常に残念なことです。

ユビキタス世代のコンピュータは、そのプロトタイプと呼べるものすら、未だ誕生していません。しかし、Arduinoの台頭やさまざまな周辺技術が出そろい、その実現は目前にまで迫って来ています。こうした時代の流れを読み取るには、やはり「どうして我々がここまで来たのか」という歴史を深く知る必要があります。

トランジスタの発明はほとんど無視されていた

昨年、量子コンピュータの実現に王手をかけた東大の古澤明教授にお会いした際、大変興味深い話をうかがいました。それは世界で最初のトランジスタ製品とはなにか、ということです。

ほとんどの人は「ラジオ」だと答えると思います。ところが実際は、ヘルメット大の巨大な補聴器なのです。しかしそれまで、トランジスタがどのような産業的インパクトを持っているのかわかっている人は世界のどこにもいませんでした。ニューヨークタイムズは1948年7月1日に、見出しなしのわずか40行の記事としてその発明を紹介するだけでした。

ところがヘルメット大の補聴器が作れる、ということがわかった途端にトランジスタの可能性に初めて注目が集まったのです。正確にはここで発見されたのはトランジスタというよりも半導体の可能性そのものです。半永久的に動作し、真空管に比べてコンパクトで壊れにくく低消費電力の半導体があれば、今まで物理的に不可能だった補聴器を実現できるという事実は、多くの科学者や発明家の心を捉えたのだそうです。

日本の小さなベンチャー企業がその可能性に注目し、膨大な特許使用料を支払って高周波への応用を研究し始めます。その会社が高周波トランジスタの生産に成功し、その後「SONY」と名前を変えて世界規模の会社に成長しました。イノベーションのタネは誰もが見落としているところにこそ、落ちているのです。

2014年は「新たなる希望」の年

2013年にはMicrosoftのスティーブ・バルマーが引退を宣言し、パーソナルコンピュータ革命に重要な貢献を果たした主要な起業家(スティーブ・ジョブズ、ビル・ゲイツ、スティーブ・バルマー)がすべて退場したかたちになりました。

一方で、ウェアラブル端末として期待されたGalaxy Gearの不調、先の見えないGoogle Glass、ここに並べるのは僭越ですが筆者らが開発したenchantMOONなど、新しいことへの果敢な挑戦は行われたものの、次世代はこれだ!という決定打にはまだ欠けているという印象でした。

逆に言えば、2014年はまだまだ新しいアイデアを試せる時期、停滞しはじめたコンピューティングの世界に新風を吹き込むチャンスがまだまだあるということではないかと思います。SD

第5章 エンジニアとしての幅を広げよう

本章では2014年に限った話ではなく、これから数年先をも見据えてエンジニアとして取り組んでおきたい事柄を展望してみたいと思います。

エンタープライズでもコンシューマでも、CPUのマルチコア／メニーコアの流れはしばらく続くでしょう。このCPUを効率よく使うために必要な並列処理の技術は、どのようなエンジニアであっても学んでおいて損はありません。並列処理には関数型言語が適しているとも言われていますので、本誌第1特集にも関係する話です。

また、プログラマであれば、数学的な思考能力や視点は大きな武器になります。増え続ける一方のデータをビジネスに活かすため、データ分析の需要が急激に高まっています。このデータ分析を行うためには、たとえツールが簡単にやってくれるとしても、自分自身に統計解析の知識がなければ、その分析の妥当性を判断することはできません。これはほんの一例に過ぎず、数学は至るところで力になってくれるはずです。

次ページからの展望が、つねに自身の技術力を高めているエンジニアの皆さんにとってなんらかのヒントとなっていれば幸いです。

具体化と抽象化の狭間で

Writer 結城 浩

次の式を見て、何か気づくことはありますか。

$$1 + 8 + 27 + 64 = 100$$

多くの人が1、8、27、64に何か**規則性**がありそうだと思うでしょう。そして注意深い人はこの4個の数がすべて3乗数になっていることに気づきます。つまり、上の式は次のように書けるということです。

$$1^3 + 2^3 + 3^3 + 4^3 = 100$$

さらに、左辺で気づいたことを右辺に**適用**するとどうなりますか。右辺の100は3乗数ではありませんが2乗数です。ですから冒頭の式は、

$$1^3 + 2^3 + 3^3 + 4^3 = 10^2$$

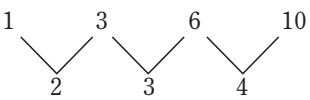
と書けることになります。

もしかしたら、**一般化**して、こんな仮説が成り立つのではないでしょうか。

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = N^2 \quad \textbf{（仮説）}$$

ここには発想の飛躍がありますが、小さなnで**実験**すると確かに成り立つことがわかります。n = 1、2、3、4で実験すると、それぞれN = 1、3、6、10で仮説が成り立ちます。

ここで出てきたN = 1、3、6、10はどんな数列でしょう。数列の謎を解くときの基本的な**道具**は、隣り合った数同士の差をとることです。



これで、隣り合った数同士の差は1ずつ増えているらしいことが予想されます。仮説の式でnが1増えるときNも1増えるので、実は次のようなきれいな式が成り立つのです。これは簡単に証明することもできます。

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = (1 + 2 + 3 + \cdots + n)^2$$

以上、長々と考えてきましたが、実はここにはデバッグのエッセンスがあります。

デバッグを行うとき、今までお話しした要素に出会います。目の前に現れる現象の**規則性**に気づき、**一般化**し、**仮説**を立てること。仮説を立てるためには**道具**も知らなければいけません。

現象を把握するためには、意識して規則性を見つけようとすることが大切です。そのためには細かいところまで注意深く見ることが必要になります。

規則性を見抜くためには構成要素に対する知識も必要です。ちょうど27を3^3だと見抜くように、あなたが扱っているプログラムの基本的な構成要素の理解が大切です。

規則性からさらに一般化する大胆な飛躍も大切です。その際には具体的な数から離れ、変数や文字を使った別の表現が必要になるかもしれません。それは仮説となって表されます。

一般化して得られた仮説は、検証が必要です。小さな数で仮説をためしたように、自分のプログラムが自分の仮説どおりになっているかどうかを検証する必要があります。この検証がなければ単なる推測で終わってしまうでしょう。

デバッグをうまく行うというのは、問題解決をうまく行うということです。あなたは毎日、多くの問題解決を行っているでしょう。いきあたりばったりの活動にしないために、規則性を見つける、仮説を立てる、検証する……といった個々の活動を意識することが大切です。

問題解決には、一見矛盾する2つの態度が必要です。1つは具体的な問題にどっぷりつかり、細かいところまで注意を払うこと。もう1つは細かいところに惑わされず、抽象化した本質を見抜くこと。具体化と抽象化の狭間を行き来して問題解決を行うのです。

あなたの今年の問題解決が実り豊かなものになりますように。SD

ビッグデータに対する新たなニーズに応える「リアルタイムクエリエンジン」

Writer 古橋 貞之

Hadoop／OLAPキューブの課題

データ分析の目的は、データをさまざまな観点から可視化することで、問題や傾向を発見し、ビジネスや運用を最適化することにあります。そんな中でHadoopは、月や日に1回といった頻度でバッチ処理を行うことで、どんなに巨大なデータでも何らかの分析を可能にするという点で強みを発揮しています。しかし一方で、「インタラクティブに条件を変えながら、データをいじり回して分析&可視化したい」というニーズに応えられないという弱みを持っていました。

これに対する代表的な解決策は、部分的に集計した中間集計データをバッチ処理で事前に作っておくという方法です。中間集計データのサイズは、各集計軸が取り得る種類数の掛け合わせで決まります。たとえば「広告ID」が50種類、「ページID」が100種類、「時刻」が1時間単位で24時間分の中間データを作成した場合、50 × 100 × 24 = 12万行のデータ量になります。この程度ならストレスなく分析ができますね。しかし、「アクセス元デバイス」30種類を追加すると、データ量は360万行になります。さらに「ユニークユーザID」3,000種類も加えると、108億行に膨れあがります。

このように、中間集計してインタラクティブな分析をするには、どんな方法で分析をしたいか予想し、条件に入れる項目を必要最小限に絞らなければなりません。さもなければ、巨大な中間集計データを高速に扱えるインメモリデータベースを別途構築する必要があります。そのどちらも非現実的であるケースも多々あります。

リアルタイムクエリエンジンの登場

近年登場してきたリアルタイムクエリエンジンは、以上のようなジレンマを解決します。MapReduceとは異なるアーキテクチャを持ち、「巨大なデータを直接、いかにストレスなくインタラクティブに分析するか?」という点に重点が置かれています。2014年はさまざまな実装が群雄割拠する年になるでしょう。

多くの実装では、巨大なデータを保存する外部ストレージ(多くの場合HDFS)、クエリを分散処理するワーカー、そしてクエリの進行を統括するコーディネータという3つの構成要素からなるアーキテクチャを持っています。中でもストレージとワーカーが分離されている点は伝統的なMPPデータベースとは異なり、すでに集約的に保存されている巨大なデータを移動させることなくクエリを実行でき、またステートレスなワーカーを必要に応じて柔軟に追加／縮退する運用が可能になります。

最後に実際に開発が進むOSSプロジェクトをいくつか紹介します。

- Impala:C++で書かれており、単純なクエリを極めて高速に処理できる
- TezとStinger:TezはMapReduceをより一般化し、単純な有向グラフに基づいた分散処理モデルの上に載せ替える。StigerはPigやHiveをTez上に直接再実装することで、既存資産を活かしたまま大幅な最適化を目指す
- Drill:実績あるコストに基づいたクエリ最適化機構を採用し、高い性能とSQL標準互換性(SQL 2003)の実現を目指している
- Presto:高い拡張性とSQL標準互換性が特徴。HDFSにも依存しないように疎結合化され、非常にメンテナンス性の高い設計を持つ
- Tajo:ほかのプロジェクトでは低遅延性のために可用性を犠牲にしがちである一方、Tajoは数時間におよぶ長大なクエリの実行にも耐える耐障害性を兼ね備える

2014年の年始、この流れを先取りして試してみるのもおもしろいのではないでしょうか。SD

データサイエンティストという職業について

Writer 佐藤 洋行

データサイエンティストはどこからやって来るのか

ビッグデータ時代である現在、データサイエンティストと呼ばれる人材が強く求められています。しかし、必要とされるスキルの領域が多岐にわたることもあり、専門的に育成されたデータサイエンティストが社会に排出されるようになるまでには、もう少し時間が必要だと考えられます。では、直近の未来においてデータサイエンティストはどこからやって来るのでしょう。

後述のとおり、データサイエンティストに求められるスキルは、大きく「IT系スキル」「分析系スキル」「ビジネス系スキル」に分けられると考えられます。そこで当然、これからデータサイエンティストになろうとする、あるいはそれを育成する場合、これらのうちいずれかをあらかじめ持つ人材が適していると言えるでしょう。

中でも、現在の学習環境とビジネス事情を考えますと、ビジネス系スキル以外の2系統のスキルのいずれかを持つ人材のほうが、データサイエンティストになろうとする、あるいはそれを育成する場合に少なからず優位かと思います。世に存在する数からしても、少なくともしばらくは、SEまたはプログラマの中から生まれて来る、というのが主流になりそうです。

データサイエンティストに求められるスキル

データサイエンティストには、データの分析はもちろん、その結果をビジネスに活用することが強く求められます。そのため、まずデータをハンドリングするのに必要なRDBMSとSQLの知識・実務経験は必須と言えます。もし、データがより巨大である場合には、Hadoopとそれに関連する知識(Java、HDFS、MapReduce、HiveやPigなど)が必要になるかもしれません。また、Linuxコマンドによるデータ処理もしばしば行われます。

次に分析という観点では、上記に加え、統計解析や機械学習に関する知識が必要となります。それらを実行するための言語としてのR、PythonやPerlは、アメリカにおけるデータサイエンティストの募集要項に必ず記載されるようになってきています。ここでもデータが巨大であれば、分散コンピューティング、あるいはIn-Database処理を行うために、MahoutやMADlibのような言語が必要となる場合もあるでしょう。

一方、自らその知識や技術を身に付け、処理する以外の選択肢もでてきています。「kaggle[注1]」のようなクラウドソーシングを利用する、という選択肢です。ビッグデータ分析のような非常に幅広い知識を必要とする分野には、このようなクラウドソーシングは適合性が高いと考えられますので、これからの発展が期待されます。

最後に、分析結果を活用するにあたって必要とされるのは、分析対象となるビジネスの業界や業務に対する知識です。実際にはすべての業界や業務に対して深い知識を持つことは不可能ですので、関係者へのヒアリングを行うのですが、そこでは質問力や理解力が必要となってきます。また、最終的に分析結果を活用した施策を実行するにあたっては、担当者に対して分析結果を適切に説明するだけでなく、理解して行動してもらわなければなりませんので、情報の伝達力に加え、説得力、あるいはプロジェクトの推進能力のようなものまで必要とされます。

それらすべてを身に付けることは非常に困難かもしれませんが、分析スキルを持つ人材が、アメリカだけでも2018年までに14～19万人不足するといわれている現在[注2]、そのような人材になることを目指してみる価値はあるのではないでしょうか。SD

注1) http://www.kaggle.com/
注2) "Big data: The next frontier for innovation, competition, and productivity"; McKinsey Global Institute, May 2011

関数型言語のカーネルへの適用

Writer 後藤 大地

マルチコア／メニーコアとC言語の不仲?

カーネルはCやC++/Objective-Cで開発されていることが多く、おおざっぱに言うと実装上の工夫でもってマルチコア／メニーコアを活用しています。この構成にはいくつかの課題が見えています。C言語はもともとマルチコアやメニーコアを意識して設計されたプログラミング言語ではありませんので、並列処理を実装するのが大変です。この方法ではいよいよ人間が負担しなければならない部分が大きくなってきました。

こうした理由もあって、カーネルをC言語系以外の言語で開発するという取り組みは以前からありました。2014年はこのあたりがもっと進展するかもね、というのが筆者の予想です。

C言語系以外でカーネルの記述用言語として開発者が採用できるプログラミング言語は、次の特徴を備えている必要があります。

- CやC++/Objective-Cで開発したのと同レベルに高速である
- プログラミング言語レベルで並列処理を想定した設計が行われている
- 強い型チェック機能を持っている
- コーディングが楽できる

プログラミング言語が言語レベルで並列処理を想定して設計されたものであるかどうかがとくにポイントです。楽をして並列処理を記述でき、効果的に論理コアを使い切れる言語である必要があります。カーネルのビルドでは強い型チェックを実施しますので、型のチェックもちゃんと行える言語である必要もあります。

代替候補を検討しなければならない状況が到来

2013年8月に英国のケンブリッジで開催されたFreeBSD開発者会議では、ネットワークの処理にカーネル内で動作するOCaml実行環境を使う「Mirage/kFreeBSD」という取り組みが紹介されました。FreeBSDカーネルに限らずカーネル内部の処理に関数型言語を使おうという取り組みは以前からあり、試験的な取り組みも進められています。

予想という割には逆のことを言いますが、結局のところ漸進的な改善を繰り返して、どこまでいってもC言語系のプログラミング言語を使い続ける可能性もあります。

これまでと状況が違っているのは、マルチコア／メニーコアのマシンがコンシューマ市場に広がり、エンタープライズ市場ではさらに多くの論理コアを搭載したマシンが登場したことで、現場の技術者がリアルに「解決策」を求めだしている、という点にあります。なんらかの方策を見つけなければならない時期が来てしまった、というわけです。

当面はハイブリッドな利用で技術の模索

カーネルの実装が一気に別のプログラミング言語に置き換わるということはあり得ないと思います。Mirage/kFreeBSDのように特定の機能を特定のプログラミング言語で実装するといった取り組みの数が増えていく中で優れた方法を模索する、というのが実際に起こる行動になると想定されます。

ひとつ覚えておきたいのは、カーネルの機能を実装する1つの手段として、従来のC言語系以外の採用を現実的と考える技術者が増えてきている、ということです。そんなわけで、2014年は並列処理の実装に向けた新しい年と位置づけ、並列処理を言語レベルで取り込んだプログラミング言語を学んでみるというのはいかがでしょうか :) SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2014年1月号

第1特集
シェルがわかればシステムがわかる
あなたの好きなシェルは何ですか?

第2特集
未来を作るITインフラ
10ギガビットを実現する ケーブリング技術

特別付録＆一般記事
・法輪寺鎮守社電電宮 情報安全護符シール Ver.2
・ソーシャルゲームのDevOpsを支える技術（後編）
1,380円

2013年12月号

第1特集
SDN/OpenFlowの流れを総まとめ!
SDN/OpenFlowで 幸せになれますか?

第2特集
下手でも好印象で効果絶大
エンジニアの伝わる図解術

一般記事
・LinuxとFreeBSDのファイルシステムの良い・悪いところをご存じですか?
・「Mirama」a.k.a. VIKING ほか　1,280円

2013年11月号

第1特集
思考をコード化する道具
我が友 Emacs

第2特集
コンピュータの加速装置
サーバサイドフラッシュ Fusion-io全方位解説

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（5）
・ソーシャルゲームのDevOpsを支える技術（前編）
・Ubuntu 13.10 "Saucy Salamander"　1,280円

2013年10月号

第1特集
Vimを使いこなしてますか?
Vim至上主義

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

一般記事
・Key Value Storeをゼロから創る
・小規模プロジェクト現場から学ぶJenkins活用（4）
1,280円

2013年9月号

第1特集
今からはじめる
sed/AWK再入門

第2特集
開発するなら やっぱりMacですよね?
9人9色のデスクトップ拝見＋新OS傾向と対策

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（3）
・最終段階に入ったFedora 19
1,280円

2013年8月号

第1特集
理論から実践まで
3Dプリンタが未来を拓く理由

第2特集
バグを狙い撃つ技術
システムを見通す力でソフトウェア開発を楽にしませんか!

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（2）
1,280円

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。

たまには膝詰め・車座で語り合ってみませんか!

会社組織を活性化するスパイス「コンパ」
―― 脈々と継承される稲盛イズム

若い社員の「宴会離れ」は最近よく言われる話です。会社の飲み会に参加することは、個人の意思を妨げているとか、そうしたイベントに出ないと表明することが意識の高い若者であるような風潮すらあります。確かに個人の意思は大事です。しかし、問題はそこでしょうか。話し合いの場としての社内宴会「コンパ」を運用し、京セラコミュニケーションシステム株式会社(KCCS)は効果を上げてきました。組織活性化の秘訣を同社常務取締役 松木憲一氏に伺いました。

取材/文:Software Design編集部

なぜコンパ(Not 宴会)なのか?

――最近の若い社員は、いわゆる会社の宴会を好まないという風潮があります。そもそも、なぜ社内宴会を重視されているのでしょうか。

松木:KCCSは京セラの情報システム部門が分離独立し設立された会社です。京セラ文化を継承しており、コンパもその1つです。京セラをはじめとして当社は「宴会」という言葉は使わずに「コンパ」と呼んでいます。コンパの始まりは、京セラ創業当時に遡ります。京セラはそもそも創業してから半世紀以上経ちますが、当時は今でいうベンチャー企業でした。知名度が低かったので人材採用に非常に苦労したそうです。最初から優秀な学生を採れませんので、それならばチームワークで市場に切り込んでいこうと考えたのです。

▼写真1　京セラコミュニケーションシステム株式会社　常務取締役 ICT事業統括本部長 松木 憲一

そのチームワークは中途半端なものではなく、強い組織にしていかねばならなかったのです。人間関係をすごく緊密にすることを前提に、他人に対して斜に構えずに正面から付き合っていこうと……それがスタートです。人が力を発揮するときは、トップから一気通貫に意識のベクトルを合わせることが重要なんです。横のつながりだけではなく、縦のつながりを作るために、コンパが必要だったのです。コンパを重要視するエピソードは稲盛名誉会長のさまざまな話の中に出てきます。

――現在、飲み会・宴会はどれくらいの頻度で行われているのでしょうか?

松木:公式なものは月1回程度ではないかと思います。この宴会場は、比較的に公式なイベント、たとえばプロジェクトのキックオフや、カットオーバーの打ち上げ、歓送迎会、期が終わるときの決起集会などに使われていますが、予約でいつも一杯です。

――この畳の宴会場はかなり広いですね?

松木:ここ東京支社は60畳以上で、80人は入れます。京都のKCCS本社、烏丸事業所など、ある程度規模が大きいところでは、コンパ用の宴会場を備えています。烏丸事業所では、祇園祭のときは長刀鉾が見える最高のロケーションの部屋が使われています。

コンパの基本とは何か?

――コンパがいわゆる宴会と違うところは何でしょうか?

松木: コンパの趣旨は、社員同士が本音で語り合うことにあります。皆で膝と膝を突き合わせ車座で座り、「俺はこう思っているんだ」と本音で語る、お互いにふれあうほどの密度が重要です。なので「畳」が大事なのです。何年か前の話です。某ホテルで新入社員の入社式の後にコンパをしたのですが、畳敷きの部屋がなかったので、わざわざ畳を借りて持ち込んだこともありました。そのくらい社員同士で話し合うことに重きを置いています。

――アルコールが苦手な方はどうしたらよいのでしょうか?

松木: 酒嫌いの社員にソフトドリンクを必ず用意しています。酒は無理にすすめません。カラオケもありますが, 最近はあまり使いません。

――運用ルールのようなものはありますか?

松木: コンパは飲み会ではないので基本全員参加です。決まりごととしては、プロジェクトの場合、たとえ上位の管理職がいてもプロジェクトマネージャーが最初の挨拶を行います。そして決意表明をよくやりますね。挨拶と乾杯をしてから、決意表明をして、最後は一本締めです。決意表明とは有言実行のスピーチみたいなものです。仕事上の目標や経営の数字に対しての自分の思いなどのアピールをします。

▼写真2　コンパ風景

仕事に効くコンパとは?

――コンパのメリットは何でしょうか?

松木: 仕事場や会議などでは話せない本音を話せたり、普段会えない方や直接関わりがない人と話ができる、といったところでしょうか。いつもは厳しい事業部長が家族の話をしたり、人間の多面性を理解するきっかけになることです。私は稲盛名誉会長にお会いする機会は年に1～2回程度でほとんどないのですが、コンパの場で仕事の話をお聞きできたりすると相当モチベーションが上がりますね。

――KCCS流コンパの極意とは何でしょうか?

松木: プロジェクトをうまく実践していくためにはチームワークが欠かせません。OSI参照モデルにたとえると、チームワークを発揮しようというようなプレゼンテーション層やアプリケーション層レベルの話をいきなりしても心に響かせるのは大変です。まずは物理層で線をつなぎます。それをするのがコンパです。そこから信頼関係を作りあげていけば一致団結できるはずです。また、コンパは日常のオフィスでの職制上の立場を切り替えるスイッチとも言えます。少しお酒が入ることで、自制心の壁がやや低くなって本音で語れるようになったりします。もちろん職場でも本音でぶつかりあいますが、どうしても立場があります。私には入社以来30年一緒に飲んでいる仲間がいます。30年もです。本音で語り合うことで強い組織を作り最高のパフォーマンスを生み出すその原動力がコンパなのです。SD

より速く、より莫大に、より高みへ！

サーバマシンの測り方

ベンチマークを極める実践テクニック

第3回　データベースベンチマークからioDriveを測る

第2回では、データベースのベンチマークツールを紹介しました。今回はディスクI/Oが多くなるケースのデータベースのベンチマークを実行して、Fusion-io社のioDriveによる性能向上を見ていきます。さらにioDriveのためのチューニングをして、ベンチマークのスコアを上げていきます。

Writer ㈱IDCフロンティア　ソリューションアーキテクト　藤城拓哉（ふじしろたくや）／Twitter@tafujish

I/O負荷が大きいDBベンチマークのやり方とは

データベースのベンチマークでは、ディスクI/Oへの負荷がかからず、スコアがCPU性能に依存するというケースがあります。しかし実際のシステムでは、データベースの性能はディスクI/Oがボトルネックとなるケースが多々あります。SysBenchではCPUに負荷がかかりやすく、ディスクI/Oまで負荷をかけるのが難しいです。今回は、tpcc-mysqlを用いてディスクI/Oに負荷がかかるデータベースのベンチマークをしていきます。

tpcc-mysqlのパラメータ

データベースのベンチマークでディスクI/Oに負荷をかける場合、データベースのサイズを大きくすることと広範囲にアクセスさせる必要があります。こうすることでメモリ上のキャシュにヒットせずディスクまでアクセスされることが多くなります。tpcc-mysqlでは、warehouseの値を大きくするとデータベースのサイズも大きくなります。500〜1000くらいの値にするとハイスペックなサーバでも十分なディスクI/Oが生じます。warehouseを500としたとき各テーブル合わせて約43GBのデータが生成されます。

MySQLのコンフィグ

MySQLのコンフィグではおもにバッファプールのサイズがディスクI/Oに大きく影響してきます。今回はデータベースエンジンとしてInnoDBを利用します。このとき、`innodb_buffer_pool_size`の値が小さいとメモリが使われずディスクへのアクセスとなり、値が大きいとメモリへのアクセスとなります。ディスクI/Oの性能を測りたい場合は、わざと値を小さくするということもあります。しかし、メモリの容量も含めてそのサーバの性能とするときにデータベースのベンチマークをする場合、バッファプールには十分な値を設定し、先述のデータベースのサイズを大きくすることでディスクI/Oに負荷をかけます。

HDDとioDriveを比較

では、実際に計測していきます。今回の測定対象のサーバ環境は**表1**です。MySQLのコンフィグ（`/usr/my.cnf`）は**リスト1**で計測しました。`innodb_buffer_pool_size`は、控え目に搭載メモリの50%としました。tpcc-mysqlのwarehouseは500です。次のコマンドでtpcc-mysqlを実行しました。

```
$ ./tpcc_start -d tpcc -u root -w 500 ➘
-c 30 -l 3600 -i 60 | tee -a result.txt
```

teeコマンドは、標準出力を画面に出しつつ指定したファイルにも出力するツールです。tpcc-mysqlのように実行中にスコアが表示されるようなベンチマークツールの場合便利なのでぜひ利用してみてください。

実行オプションとしては、コネクション数(-c)30、計測時間(-l)3600秒、途中のスコア表示の間隔(-i)60秒としました。ウォームアップ時間(-r)はデフォルトの10秒としメモリに頼り過ぎないようにしました。

また、ベンチマークをかける(tpcc_startを実行する)マシンと、ベンチマークをかけられる(データベースサーバ)マシンは同じマシン上での実行となります。ベンチマークをかけるときにCPU負荷もかかるため、CPU負荷が高いケースでは、ベンチマークをかけるマシンとかけられるマシンは分けたほうが良いです。

実行結果は**図1**です。-iで指定した60秒間隔でのNew-order処理のトランザクション数です。最終的なスコアは、SASのHDDが1本では、1831.100 TpmC、SASのHDDを4本で構成したRAID10(キャッシュ有)の場合では12150.417 TpmCでした。RAIDキャッシュの効果でディスクの本数以上の性能が出ました。ioDriveは33412.184 TpmCと高速で、ioDriveを入れるだけで速くなるという話が垣間見えました(いずれもファイルシステムはEXT4で計測しています)。

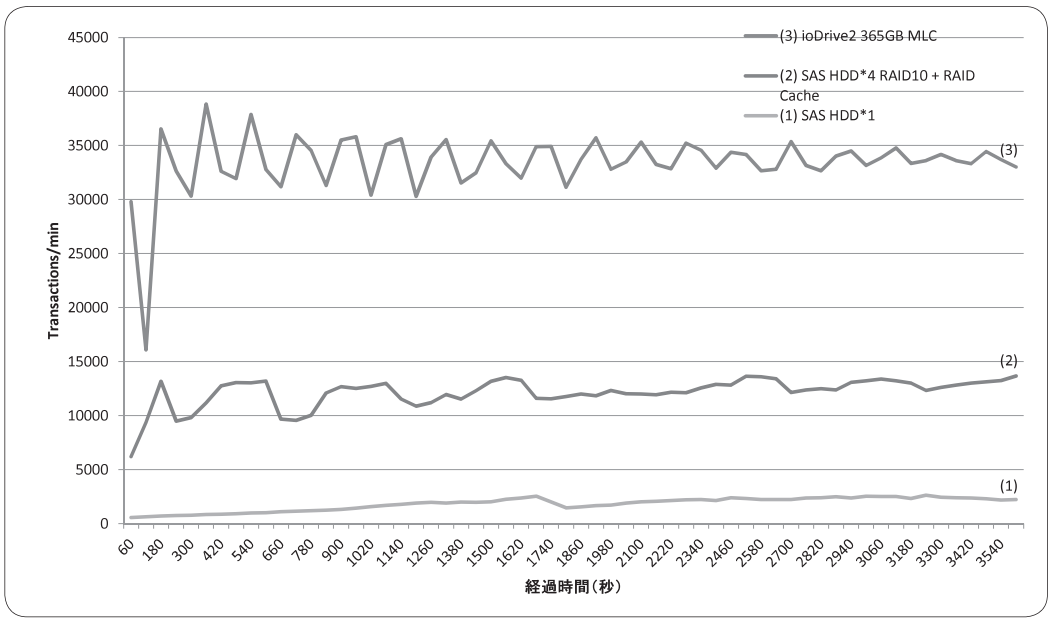
▼表1　測定サーバ環境

機種	HP ProLiant DL360p Gen8
CPU	Intel Xeon E5-2670 (2.6GHz/8コア) 2個
Memory	64GB (PC3-10600R 8GB 8本)
Disk	SAS 15000rpm 4本 の RAID10、RAIDキャッシュ1GB
ioDrive	ioDrive2 MLC 365GB
OS	CentOS 6.4 x86_64、GCCはOS標準パッケージ利用
MySQL	MySQL Community Server 5.6.14 GA Release

▼リスト1　計測に利用したMySQLコンフィグ

```
[mysqld]
datadir=<対象パス指定>
user=mysql
max_connections = 1000
thread_cache_size = 1000
innodb_buffer_pool_size = 32G
innodb_log_file_size = 512M
skip-name-resolve
```

▼図1　HDDとioDriveの比較



ioDriveをチューニングするには

せっかくのioDriveなのにもったいないということで、次にioDriveのためのチューニングをして、tpcc-mysqlのスコアを上げていきます。ioDriveに対するチューニングのポイントとして、ファイルシステム、MySQL Serverを紹介します。tpcc-mysqlの実行オプションは先と同じで、その結果は**図2**にまとめました。

ファイルシステムをチューニング

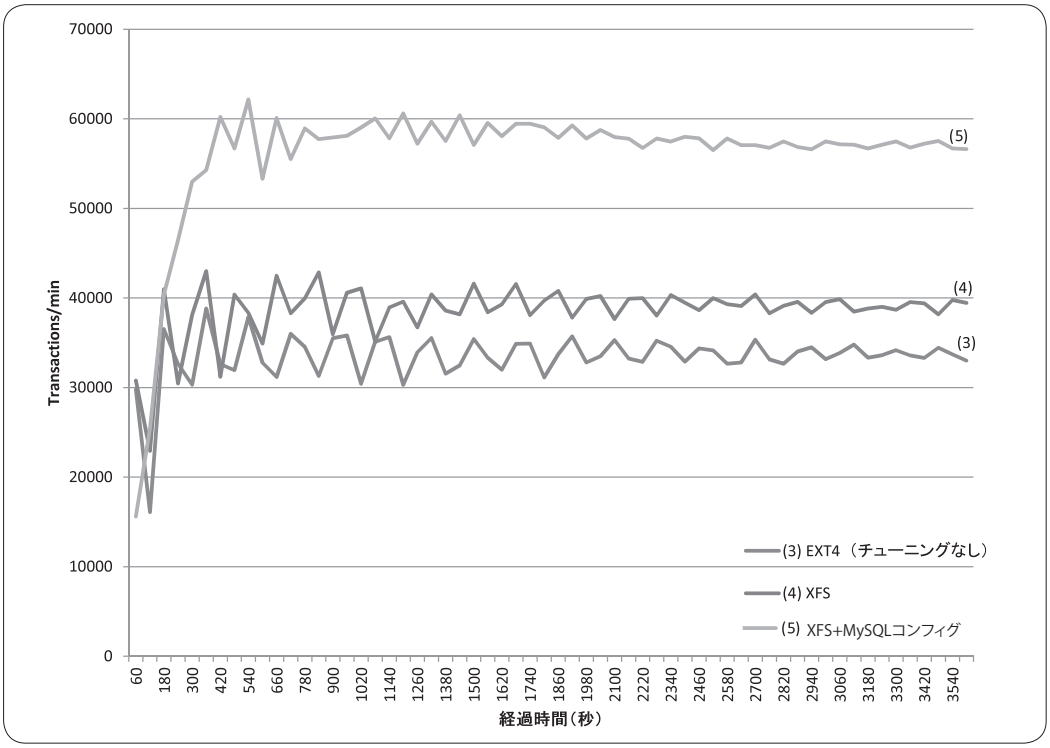
ioDriveを利用したときのファイルシステムとしてXFSが適しているという話はよく知られているかと思います。XFSは並列処理に適しているため、ioDriveとの相性が良いです。インストールと設定は**図3**のとおりに行います。ポイントとしては、I/Oスケジューラのuse_workqueueの設定を忘れないようにしてください。そうしないと十分な性能を出せないケースがあります。

図2の結果のとおり、XFS(38601.434 TpmC)のほうがEXT4(33412.184 TpmC)より約15%スコアが向上しています。ファイルシステムをXFSにするのは有効でした。

MySQLコンフィグをチューニング

MySQL Serverをチューニングすると言え

▼図2　ioDriveチューニングの結果



▼図3　XFSの利用設定方法

```
# yum -y install xfsprogs ←XFSをインストール
# mkfs.xfs -s size=4096 -b size=4096 /dev/fioa ←XFSでioDriveをフォーマット
# mount -t xfs /dev/fioa <マウントポイント> -o nobarrier,noatime ←ioDriveをマウント
# vi  /etc/modprobe.d/iomemory-vsl.conf ←に「echo options iomemory-vsl use_workqueue=3」を追記
```

ばコンフィグをいじってということになりますが、ioDriveのI/O性能に合わせた設定があります。今回は、MySQL 5.6向けにリスト1にリスト2の設定を追加しました。innodb_flush_methodでダイレクトI/Oを利用します。innodb_write_io_threadsとinnodb_read_io_threadsでI/Oスレッド数を増やします。innodb_io_capacityでI/Oオペレーション数も増やします。ここでは、innodb_log_file_sizeは変更しませんでしたが2Gくらいに増やしても良かったです。

MySQLのコンフィグをチューニングした結果55994.266 TpmCと劇的な性能向上が見られました。MySQLのI/O周りの設定をチューニングすることは大きな効果があることがわかりました。

▼リスト2　追加したMySQLコンフィグ

```
innodb_flush_method=O_DIRECT
innodb_write_io_threads=16
innodb_read_io_threads=8
innodb_io_capacity=10000
```

次回予告

前回紹介したtpcc-mysqlを使いディスクI/Oに負荷がかかるデータベースのベンチマークを実施しました。MySQLのコンフィグチューニングで性能は大きく変わりますし、稼働させるサーバのスペックによっても変わってきます。そのため、ベンチマークによりその設定が有効かどうか確認してみてはいかがでしょうか。

次回はがらりと変えてCPUのベンチマークとその評価を行っていく予定です。SD

分散データベース「未来工房」

第8回 Hadoop on Riak CS

Writer 上西 康太（うえにし こうた） Basho ジャパン株式会社 kota@basho.com

ストレージは、そのデータを扱う処理装置と一緒になってこそ力を発揮する。Riak CSはまだ発展中のソフトウェアだが、Riak CS上のデータをプログラムで処理するには、プログラムが動作するCPUのあるところに移す必要があった。そこで本稿では、Riak CS上のデータ処理の方法の1つとして、Hadoopを使ってデータ処理を行う方法を解説する。

注）本稿は、筆者の意向により常体で表記している。

はじめに

Hadoopは分散ファイルシステムと分散タスク管理が巧妙に組み合わさってI/Oから計算、集約などの分散処理ができるという優れた製品だ。それまで多数の分散処理のためのライブラリや製品が登場していたが、分散処理の本質的な難しさから、専門家でない人にはなかなか使い辛いものが多かった。Hadoopが優れているのは、分散処理のいくつかの本質的な難しさをシステムとライブラリで解決しつつ、ある程度の自由度をユーザに提供できるフレームワークだったためだ[注1]。

Hadoopの登場によって、それまでは一部の人たちだけのものであった分散処理、並列処理は一気にポピュラーなものになり、これによって多くの問題が解決された。こと分析やバッチ処理については、他に選択肢がないといっても過言ではない。

Hadoopは大きくわけて2つのシステムに分かれる。分散処理に最適化した形でデータを保持するHDFS[注2]と、その上で動作するMapReduceのクラスタだ。2.0からはYARNというジョブ管理システムも同梱されるようになったが、ここでは、より構成がシンプルな1.2系のHadoopを例に説明する。

MapReduceとHDFSはきれいに分離されており、MapReduce自身がさまざまなファイルシステムからデータを読み出せるようにインターフェースが抽象化されている。典型的にはHDFSだが、ほかにもさまざまなストレージから入力データを取り出せるようになっている。通常のOSにマウントされたローカルのファイルシステムだけでなく、FTPによるアクセスやAmazon S3のインターフェースにも対応している。このように、さまざまなシステムからデータを取り込めるようにエコシステムが発達しているのもHadoopの魅力の1つだ。

そこで今月号では、Riak CS上のデータを、HadoopのMapReduceの入力データとして取り込むことが実用的なレベルで可能なことを解説し、最後に簡単に試す手順を紹介したい。

Hadoop on Riak CSの使いどころ

Riak CSについては、本誌9月号の連載でも解説した。そのときは自宅用ストレージへの応用として紹介したが、当然、プロダクション向けのシステムを組むことも可能だ。実際にRiak CSを利用してストレージサービスを内部向けまたは外部向けに運用している企業はいくつかある。その特徴は、コモディティサーバを複数台並べることで高可用性を担保しつつ、単純なスケールアップも可能なところだ。また、

注1）具体的には、I/Oの並列化、失敗したタスクの再実行の扱い、結果の集約のしくみなど難しいポイントがいくつもある。

注2）Hadoop Distributed File System

下側のデータ保存レイヤには分散データベースとして実績のあるRiakを採用しているので、運用が容易なRiakの特徴をそのまま引き継いでいる。とくにサーバの追加時および障害時の対応は非常に簡単で、その簡単さは本誌の連載で何度か解説してきたところである。

運用の簡単さの一例として、Riak CSのプロセス構成を説明しよう。Riak CSは**図1**のようなプロセス構成を持つ。このとき、データを保持し、かつ状態を持つプロセスはRiakのプロセスだけなので、Riak CSとStanchionは起動、停止などをカジュアルに実行することができる。

また、Riak CSにはマルチテナントの機能があり、ユーザごとに分離された名前空間を提供することができる。これによって、オペレーションミスなどで他のユーザのデータを見たり、消したりすることができないことを保証しつつ、設備に相乗りしてコストを下げることができる。

HDFSとRiak CSの比較

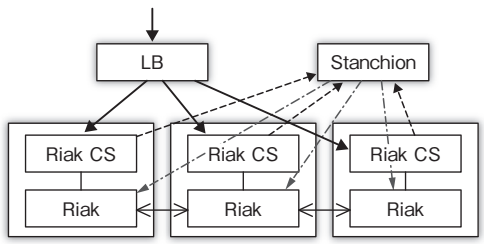
MapReduceが登場する以前は、プログラムがデータをロードし、プログラムが動作するCPU上でデータの処理を実行してもとの場所に戻すというコンピュータの基本的な概念にもとづいたシステムがおもに使われていた。MapReduceはこの既成概念を実用的なレベルでひっくり返し、HDFS上のチャンクがあるマシン上にタスクを配置し、そこでロード、処理、ストアの基本的な部分を分散処理させるというモデルを選択した。つまり、プログラムのある場所にデータを移動するのではなく、**データのある場所にプログラムを移動した**のである。これによって、いままでのシステムの処理のネックであったディスクI/Oを並列化することに成功した。つまり、HDFSが最も威力を発揮するのはノード間でデータを移動するコストが相対的に高い場合である。

一方で、Riak CSはコンシステントハッシュベースのデータ分散で、ブロックサイズが1MBであるため、HDFSのようにある程度まとまったサイズでデータが局在していない。また、機能面でも、ファイルのオフセットからデータを持つサーバのネットワーク上の位置を知ることはできない。このことから、HDFSのようにデータのある場所にプログラムを移動してI/Oを並列化することが難しい設計になっている。

ところが、近年では10Gbitのネットワーク設備の価格が急速に下がってきたこと、MapReduceで分散処理したいジョブはI/Oインテンシブなものだけでなくインテンシブなものもかなりの割合で存在することから、HDFS以外のデータソースからもデータをロードして、よりカジュアルにMapReduceを行うケースが増えてきている。

つまり、I/Oインテンシブでない処理や、ネットワーク帯域がある程度確保されている環境では、MapReduceの処理対象になるデータをRiak CS上に置いても十分に実用的な分散処理の性能が得られる。組み合わせて使うことで、Riak CSのよいところと、Hadoopの強力な分散処理基盤の両方のメリットを享受する方法をこれから紹介したい。

▼図1　Riak CSプロセス構成図



構成例

最も素直な構成例としては、同一クラスタにRiak、Riak CS、MapReduceのTaskTrackerが同居するものだ。ここは別にしてもよいところだが、台数が限られている場合もあるだろうから同じクラスタに同居するものとした。

図2(a)は典型的なHadoopの構成である。HDFSのDataNodeとMapReduceのTaskTrackerが同じマシンに同居しており、JobTrackerは入力ファイルのデータの物理配置から、処理対象になっているデータのあるマ

シンに同居しているTaskTrackerにデータ処理をさせる。TaskTrackerは同じマシンに同居しているDataNodeからデータを読み出す。このときDataNodeのローカルにあるものが優先して指定されるため、ネットワークを経由せずに入力データを読み出すことができる。これによって、ネットワークを経由せずにすべてのデータをロードすることができ、ボトルネックになりやすいディスクI/Oの並列化をミドルウェアのレイヤーで実現できる。ネットワークコストが問題になる状況では、この構成が最も有利である。

Riak CSのセットアップ

ここでは、Riak CS 1.4.2をパッケージからインストールする。主要な内容は本誌9月号にも執筆したし、カーネルパラメータの設定まで含めてドキュメント[注3]にもなっているからそちらを参照いただきたい。ここではDebian、Ubuntuなどの APTが利用できるLinuxについて例を掲載しておく。

Riak CS の設定や起動

RiakとRiak CSそれぞれ、`app.config`と`vm.args`という設定ファイルがある。それぞれ`/etc/riak`と`/etc/riak-cs`にあるので、必要な設定を行う。こちらも詳しくは公式ドキュメント[注4]に掲載されているのでそちらを参照いただきたいが、Riakに必要なことは、

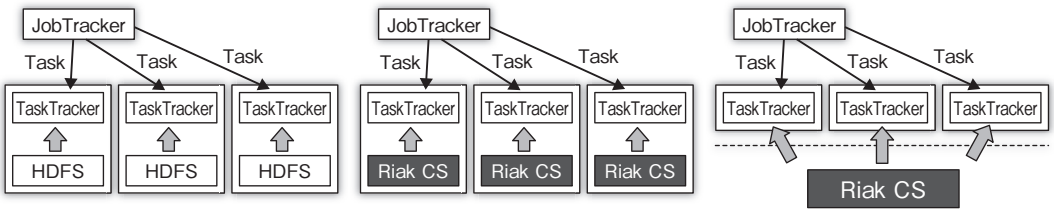
- システムレベルで、プロセスあたりの最大ファイルディスクリプタ数を5万以上にしておく
- 各種パラメータ変更を行う。とくにI/Oスケジューラを`cfq`から`deadline`に変更しておく
- riak coreのセクションに`allow_mult`を設定する
- riak kvのセクションで`storage_backend`をマルチバックエンドに変更する
- `vm.args`に適切なIPアドレスを設定する

だけである。これをもとに、Riakを設定すればアクセスできるようになる。あとは、Stanchion

注3） http://docs.basho.com/riakcs/latest/

注4） http://docs.basho.com/riakcs/latest/

▼図2　Hadoopの典型的な構成と、Riak CSを使った構成例の比較



(a)典型的なHadoopの構成である。HDFSのDataNodeとMapReduceのTaskTrackerが同居していることにより、ディスクI/Oの単純並列化を実現し、かなりの並列処理性能を実現する。

(b)HDFSを単純に置換した例。Riak CSはHDFSのようにデータの位置を取得するAPIはないため、ローカルのRiak CSにアクセスしたからといって、その先で必ずローカルのディスクにアクセスできるとは限らない。しかし、ネットワークコストが安い環境や、CPUインテンシブな種類のジョブを実行する状況では、あまり問題にならない。

(c)完全にサービスとしてMapReduceの計算ノードと、データを保持するRiak CSのクラスタを分離する構成例。ある程度クラスタを分離することによって構成管理を容易にしたりと、システムの疎結合化を進めることができる。Riak CSのマルチテナント機能を活かして、リソースが隔離された複数の計算クラスタから利用する場合にはこの構成がよいだろう。

▼図3　Riak CSのインストール手順
（RiakとRiak CSはすべてのマシンにインストールする。Stanchionはどこか1台で動作していれば十分である。マシンに対する負荷は小さいので、運用を簡単にするためJobTrackerと同じノードに入れてもよいだろう）

```
$ 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
```

と Riak CS に anonymous_user_creation を true で設定すればよい(図3)。

この状態で、Riakをすべて起動し、クラスタ化する。ここでも細かい手順を省略するが、基本的には図4のような流れになる。全マシンを同時に追加する場合にはcluster joinの部分を、追加したいノードの分だけ実行することになる。

Stanchionは、1台だけ設定して起動すればよい。HTTPのアドレスとポート番号を設定しておく必要があるので、それはあらかじめほかのRiak CSプロセスがアクセスできるように設定しておく。Stanchionの起動はstanchion startとすればよい。Riak CSは全マシンでriak-cs startを繰り返すとよい。

Riak CSが起動したら、まず管理者アカウントを作成する(図5)。

本来は、管理者アカウントの作成に成功したらそれをRiak CS、Stanchionに登録し、anonymous_user_creationをfalseに戻したうえで、作成した管理者アカウントから一般ユーザを作成して(設定を読み込むために再起動が必要)、一般ユーザからRiak CSにアクセスを行う。しかし、今回は簡単のため手順を省略する。

最後に、s3cmdというツールで簡単な動作確認を行う(図6)。ここでは、Riak CSをローカルで動作させているため、s3cmdの設定には、proxy_base、proxy_portの設定が必要だ。

ここですべての設定に成功していたら、コンソールには何も表示されない。何も作成してないので当然だ。

Hadoop(MapReduce)のセットアップ

まず、簡単のためにHadoop MapReduceローカルファイルシステム上で動かす。筆者の実験では、Apache版のHadoop 1.2.1を使用した[注5]。

Hadoop 1.2.1は理研のミラーサイト[注6]などからダウンロードできる。本来はJREで十分だが、ここではJDK1.7.0_25を用いた。

conf用のディレクトリのうち、masters、slavesを設定する。それに合わせてmapred-site.shを実行する。

これでJobTrackerとTaskTrackerが起動する。これで、通常のローカルファイルをインプット・アウトプットとする設定は完了した。試しにExamplesの中から、wordcountを適当なファイルに対して実行すると、実際に結果を確認できる。

最初にサンプルを実行する。Hadoopのサンプルといえばwordcountが有名なので、まずは動作確認する。catでWordcountの結果が内部にあれば、無事に成功したことになる(図7)。

jets3tの設定

jets3t[注7]は、Hadoopの配布バイナリに含まれている、Amazon S3へアクセスするためのライブラリだ。これを変更して、S3の代わりにローカルで起動しているRiak CSへアクセスできるように設定を行う。

▼図4　Riakのクラスタを構築するための主要なコマンド

```
$ sudo riak start
$ sudo riak-admin cluster join riak@10.0.0.1
$ sudo riak-admin cluster plan
$ sudo riak-admin cluster commit
```

▼図6　s3cmdによるRiak CSへの最初のアクセス

```
# pip install s3cmd ## s3cmd インストール
$ s3cmd --configure ## s3cmd 設定
... ## 作成したアカウント情報を入力する
$ s3cmd mb s3://test ## bucket 作成
$ s3cmd ls ## 表示
$ s3cmd put README.txt s3://test
```

注5) Hadoopには複数のディストリビューションがあり、オープンソースで最も安定しているのはCDHだろう。RPMやdebパッケージなどが豊富に準備されているので、インストールも簡単なので推奨する。

注6) http://ftp.riken.jp/net/apache/hadoop/common/hadoop-1.2.1/

注7) 本家サイトによると、ジェットセットと読むそうだ。

▼図5　Riak CSにおける最初のアカウント作成(-cluster joinは各ノードで繰り返す)

```
$ curl -H 'Content-Type: application/json' -X POST http://localhost:8080/riak-cs/user \
--data '{"email":"foobar@example.com", "name":"foo bar"}'
```

まず、conf/jets3t.propertiesを作成して、Job-Tracker、TaskTrackerがアクセスする対象を設定する。ここでは、TaskTrackerがアクセスするローカルのRiak CSに直接アクセスさせるので、図8のように設定をする。

これは、Hostヘッダには S3と同様のFQDNを記述しておきながら、実際には別のホストとポート番号にアクセスするjets3tの機能で、ファイヤーウォール内にあるユーザ向けの機能である。これを設定するとHadoopがAmazonだと思ってアクセスしに行ったら、jets3tがプロキシとして設定されているホストにアクセスしにいく。これを利用して、Riak CSなどS3以外の別のシステムに接続できるようになる。

また、最後に認証情報をcore-site.xmlに書いておく(図9)。ここまで設定しておくと、ローカルにあるRiak CSに対して、TaskTrackerがデータを読みに行くようになる。実際には、図2(b)のようなシステム構成に近いものになる。

動かしてみる

設定が完了したら、実際に動かしてみよう。図10のように、最終的にMapの出力で179、Reduceの出力で131個のレコード(wordcountの場合は単語の種類)が計算されたことがわかる。

次に、結果の出力先を同じRiak CS上に設定して実行してみる。途中まで正常にジョブが実行され、大量のJavaのエラーが表示され、いったん成功したかのような結果が表示されるが、結果ファイルはRiak CS上には見つからない。これはRiak CSにはまだrenameが実装されていないからと考えられるが、試す場合には注意したほうがよい。

renameは、Riak CS 1.5.0でできるようになる予定で、すでにテスト実装も公開されている。しかし、Riak CSでの巨大なファイルのrenameはあまり推奨しない。当面はローカルファイルとしていったん出力してからコピーするのがよいだろう。HiveなどMapReduce上の動作ツールを使っている場合には適宜変更を加えるしか今のところは方法がない。ローカルに書き出すには結果が大きい場合には、別途起動しておいたHDFSに書き出してからdistcpするのも選択肢の1つだろう。

▼図7　ローカルでのMapReduceの実行例

```
$ bin/hadoop jar hadoop-examples-1.2.1.jar \
wordcount README.txt /tmp/wc
...
$ ls /tmp/wc
_SUCCESS _logs part-r-00000
$ cat /tmp/wc/part-r-00000
...
```

▼図8　ローカルのRiak CSへアクセスするjets3tの設定例

```
s3service.https-only=false
s3service.proxy-host=localhost
s3service.proxy-port=8080
```

本番環境で動かすために

ここまで、ローカルでテスト環境を構築する方法を簡単に解説してきた。これがうまく動作すれば、fluent-plugin-s3などでRiak CSにログを集めつつ、MapReduceで分析のための前処理を行うところまである程度のテストができてきたことになる。

しかし、サーバ環境で、複数台のサーバを用意するためにはまだいくつかやらなければならないことがある。

ネットワーク環境

ネットワークに負荷がかかるようなジョブを実行する場合は、ネットワーク帯域をある程度確保しておく必要がある。そうでない場合(CPUインテンシブなタスクの場合など)でも、冗長

▼図9　Riak CSのアカウント情報をcore-site.xmlに設定する

```
<property>
  <name>fs.s3n.awsAccessKeyId</name>
  <value>J_PJJS7NEHXJSQQV5X..</value>
</property>
<property>
  <name>fs.s3n.awsSecretAccessKey</name>
  <value>rktX1Hy4VX3V0Favo3b...</value>
</property>
```

化も兼ねてたとえば1GbEのケーブルをトランケートまたはボンディングなどをして少し帯域を広げておくとよいだろう。また、Riak 1.x系は認証などの機能がないため、Riak CSのネットワークに意図しないアクセスが行われないように、ネットワークレベルでのセキュリティを確保しなければならない。

DNSの設定など

Riak CSをテスト以外の目的で運用する場合、FQDNを固有のものに変更して運用することが推奨されている。たとえば、riak-cs.example.comというURLでRiak CSのサービスを行う場合には(**図2(c)**のような環境)、ロードバランサやDNSのラウンドロビンを使ってシステムを構成し、Riak CSのサーバに正しくアクセスできるようにしておく必要がある。httpsを利用する場合にはサイト証明書の準備も必要だ。

endpointの変更

固有のFQDNを設定できたら、今度はjets3tの設定を変更してやる必要がある。デフォルトのままではAWSのs3.amazonaws.comに接続してしまうためだ。jets3t.propertiesに**図11**のように設定を行う。ポート番号がもし80や443のデフォルトから変更されている場合はここを併せて変更しておこう。disable-dns-bucketsは、bucketname.riak-cs.example.comのようなアクセスを許すかの設定で、DNSが正しく設定されていれば問題ないだろう。

▼図10　Riak CS上のデータを入力にしたMapReduceの実行例

```
$ bin/hadoop jar hadoop-examples-1.2.1.jar \
wordcount s3n://bucket-name/README.txt /tmp/wc2
...
Total committed heap usage (bytes)=274661376
CPU time spent (ms)=980
Combine input records=179
SPLIT_RAW_BYTES=86
Reduce input records=131
Reduce input groups=131
Combine output records=131
Physical memory (bytes) snapshot=281690112
Reduce output records=131
Virtual memory (bytes) snapshot=1763856384
Map output records=179
```

他システムとの組み合わせ

本稿では、Riak CSのHadoopとの連携について解説してきたが、他のさまざまなシステムとの連携を行うことができる。2013年11月に筆者が主催したRiak Meetup Tokyo#3で発表されたmixiの事例[注8]がよい例になるだろう。各種ログ、静的コンテンツ、MySQLのスナップショットなどさまざまなデータをRiak CSに保存しつつ、HiveでRiak CS上のデータを分析している。

まとめ

本稿では、Riak CS上にあるデータをHadoopのMapReduceの入力として読ませて分散処理するための方法について解説した。サンプルは1台のマシン上で実行するだけのものであるが、分散環境で構築するためのポイントも解説した。

Hadoopは、それだけでも巨大なエコシステムを持っている。そこに多くの製品が参加しており、やりたいことに合うものを見つけ出して組み合わせるだけでもたいへんだ。データベース、ストレージ、分散処理、クエリエンジンなど、それぞれ用途が異なるものがある。

筆者はあくまでもRiakとRiak CSを推薦するが、読者諸賢においては、自分たちの用途に合わせて最適な製品とシステム構成を選択して、スケールアウトするインフラを構築してビジネスを拡大してもらいたい。SD

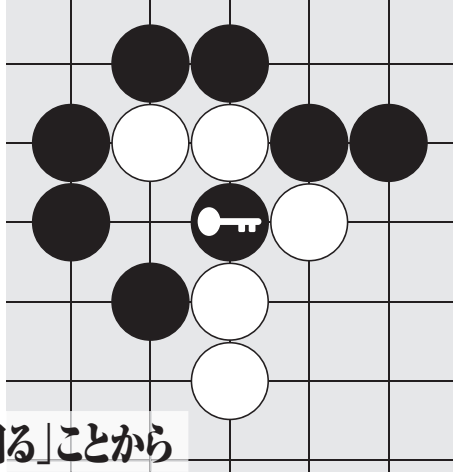
注8)「Riak CSとmixiのプライベートクラウド環境」http://www.slideshare.net/hidetakakojo/riak-meetup3-upload

▼図11　FQDNを設定したRiak CSにアクセスするjets3tの設定例

```
s3service.s3-endpoint=riak-cs.example.com
s3service.s3-endpoint-http-port=8080
s3service.s3-endpoint-https-port=8443
s3service.disable-dns-buckets=true
```

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

【第八回】真のフィッシング対策は「敵を知り、己を知る」ことから

前回に続き、フィッシングをテーマに取り上げます。個人ができるフィッシング対策として「怪しいリンクはクリックしない」ということが盛んに語られますが、本当にそれで良いのでしょうか？　今回はそもそもフィッシングとはどんな行為なのかを整理し、個人で対策できることは何か、国家や世界レベルではどんな対策がなされているのかについて考えてみます。

フィッシングとは

「フィッシング(Phishing)」は、電話回線を不正使用する「フリーキング(Phreaking)」と人を釣る「フィッシング(Fishing)」の合成語だと言われていますが、この説はどこまで正しいかわかりません。いつの間にか、人をだます電子メールを送り、偽サイトを使ってユーザアカウントとパスワードを盗むことをフィッシングと呼んでいました。極めて単純なモデルにもかかわらず、昔も今も非常に有効な手法です。

またフィッシングは、これまでのセキュリティの問題とは異なり、愉快犯とかバンダリズム[注1]といった類のものではなく、最終的には窃盗を目的とするものです。また、法の改正により、今ではフィッシングを企てた段階ですでに犯罪として成立する時代になっています。

すべては電子メールから始まる

フィッシングの始まりは、まず電子メールがやってきます。その電子メールは、どこかの有名な電子商取引サイトや、銀行のオンラインサイト、あるいはクレジットカード会社のサイトへのリンクが張られています。そこには、「セキュリティの問題が発生しました。パスワードを変更してください」「重要なメッセージが届きました。確認してください」「支払いの確認があります。ログインしてください」といったようなアカウントとパスワードを利用させるようなメッセージが書かれているはずです。

そして、そのリンクをクリックすると、サイトが開き、あなたはアカウントとパスワードを入力します。すると、入力エラーが発生します。再度入力画面になりますが、入力エラーは日常茶飯事ですし、あなたは何も気にせずにもう一度入力するはずです。今度は、無事にサイトに入ることができます。何かおかしいような気がするかもしれませんが、「このサイトはいつもよくわからない」と自分に言いきかせて終了させることでしょう。

この時点で、あなたはアカウントとパスワードが抜き取られていることになりますが、それに気がつく人はごくごく僅かでしょう。ほかのバリエーションとしては、クレジットカードの情報を入力させるなどがあります。

フィッシングは意外と敷居が低い

同じく電子メールをきっかけに発生する問題としては、電子メールでマルウェアを送ったり、あるい

注1)　破壊をして喜ぶ行為。

は、マルウェアを用意しているサイトを閲覧させてマルウェアをシステムに侵入させる標的型攻撃があります。標的型攻撃の場合、攻撃側はマルウェアを用意するという技術的スキルを必要としますが、フィッシングの場合はWebサイトのデザインさえできれば良く、技術的な敷居は極めて低くなっています。

もちろん、標的型攻撃の場合、成功したならばシステムに対し大きなダメージを与えられます。「システムの支配権を奪う」「個人の権限でアクセスできるすべてのファイルに対して何かをする」など大きな脅威となりますので、インパクトは標的型攻撃のほうが大きいのは言うまでもありません。しかし、それを行うには技術的に一定のスキルを必要とします。

フィッシングは「メールを送ること」、そして「見た目が本物によく似たWebサイトを作れること」といった要件を満たしていればできるのです。

2013年10月に起こったゲームサイトのフィッシングでは、18歳の少年を逮捕しています。

偽サイト開設容疑、18歳少年を初逮捕 清水署

インターネット上で他人のIDとパスワードを盗み取る「フィッシングサイト」を開設したとして、清水署と県警生活経済課サイバー犯罪対策室は16日、不正アクセス禁止法違反と商標法違反の疑いで沖縄県宜野湾市の店員の少年（18）を逮捕した。

（中略）

昨年5月の改正不正アクセス禁止法施行後、フィッシングサイト開設による逮捕者は全国で初めて。

（静岡新聞 2013年10月17日7時35分
http://www.at-s.com/news/detail/775173052.html）

この18歳店員がフィッシングを試すことができたというだけで、フィッシングの敷居が低いことは理解できると思います。ですが一方で、インターネット上で証拠を残さないようにフィッシングサイトを用意するのは難しいということもわかります。

◆表1　フィッシング対策協議会 緊急情報一覧（2013年12月10日確認分）

掲載日時	概要
2013年12月10日	ハンゲームをかたるフィッシング
2013年11月19日	コミュファ光 Webメールをかたるフィッシング
2013年11月18日	三菱東京UFJ銀行をかたるフィッシング
2013年11月15日	[11/15更新] eoWEBメールをかたるフィッシング
2013年10月10日	スクウェア・エニックスをかたるフィッシング
2013年10月10日	UCカード（アットユーネット）をかたるフィッシング
2013年10月03日	ODNをかたるフィッシング
2013年10月01日	ポータルサイトgooをかたるフィッシング
2013年08月07日	ハンゲームをかたるフィッシング
2013年06月07日	NCSOFTをかたるフィッシング

フィッシングは目的が明確

フィッシングには1つの特徴があります。狙われるのはたった1つのアカウントとパスワードですが、それを使うサイトは必ずお金を扱うか、あるいは何か価値のある利用方法を持つサイトだということです。そして、そこから推察できることはフィッシングはその目的が明確であるということです。逆の言い方をすると、どこかのサイトを真似て人をだますわけですから、そのターゲットとなるサイトを選んだ時点で、目的は明らかというわけです。

表1はフィッシング対策協議会（後に説明します）のサイトで2013年12月10日に確認した緊急情報一覧です。

銀行のアカウントとパスワードを盗もうとしているならば、それは銀行口座にあるお金をすべてどこかに送金するためでしょう。クレジットカードも同様です。金銭目的だというのは簡単に想像つきます。

表1にはありませんが、オークションのアカウントであれば、登録しているクレジットカードを使ってオークションをして決算する、あるいは、そのアカウントの情報を書き換え、自分がオークション出展者になるなどに使われるでしょう。

Webメールのアカウントを盗めば、これまた足

がつかないアカウントを手に入れたことになるので、直接的な金銭の被害はなくとも、そのメールアカウントは不正なことに使われることでしょう。

ゲームの場合は、いろいろなアイテムを盗んでトレーディングすることにより金銭的利益を得るといったことが十分に考えられます。ただし、中にはキャラクターを使って本来の自分の経験値を上げるといった子どもじみた動機もあるような気がします。

フィッシングの目的は昔も今も、愉快犯でも腕試しでもなく明確に犯罪目的です。法律の面では、現在では国内法は改正されて、かなり厳しい扱いになっています。改正された「不正アクセス行為の禁止等に関する法律」では、フィッシングそのものもが、「識別符号の入力を不正に要求する行為の禁止」ということで違法化されています。フィッシングによりアカウントとパスワードを取るのはもちろんのこと、それ以前にフィッシングサイトのURLを送りつけたところで違法になります。

フィッシングする立場で考える

フィッシングを行う側、つまり犯罪者側から考えてみます。フィッシングサイトを作るのに特殊なスキルはいりません。ターゲットとしたWebサイトの見た目を真似するだけなら、ホームページビルダーのようなツールを使ってでもできます。ちょっとぐらい変でも、前回の本連載で紹介した「100万円札贅沢(ぜいたく)ふせんが1万円札として使われる」という例からもわかるように、人間の観察力はそれほど鋭くありません。かといって、すべての人間がだまされるわけでもありませんので、この発見する／したら、という議論はちょっとの間だけおいておきましょう。

サイトのデザインはわりと技術を理解していなくても簡単に作れますが、実際に偽サイトを作るとなると、格段に難しくなります。少なくとも日本国内で法に従ってインターネットにアクセスする限りは利用者を探し当てることができます。携帯電話もインターネットも契約がなければ使えません。フリーアクセスのアクセスポイントを使えばどうにかなると思うでしょうが、通信のデータリンク層で使われるMAC(Media Access Control)アドレスがハードウェアに組み込まれているので、それが残ります。(WiFiの)通信チップの「指紋」を残すようなものです。その指紋がどこかで見つかれば、簡単に特定できます。ましてや合法的にWebサーバを使うとなると隠れようがありません。

そこでは(非合法に)Webサーバを乗っ取るなどの行為が必要になるでしょう。フィッシングは、フィッシング単独には止まらず、そのようなほかの問題も同時に発生している場合がほとんどです。その際には、外部からサイトを乗っ取るようなスキルが必要になるはずです。

ただし、高度なスキルが必要かといえば、そうでもありません。世の中にはインターネットに接続されているにもかかわらず、まともに整備されていないサーバがいくらでもありますから、簡単に手に入るでしょう。しかも、ほんのわずかな期間、たぶん1日も経たないうちに閉鎖されるので、サーバの安定性などは考える必要はありません。理屈の上ではサーバである必要はなく、インターネット側からアクセスできれば十分ですので、インターネット側からダイレクトにアクセスできるパソコンにマルウェアを感染させ、HTTPをサポートするサーバアプリを立ち上げることで代用することも可能です。

フィッシングする側の立場で考えると、犯罪捜査をかわすために、中国やロシア、あるいは東欧諸国や東南アジアといった、捜査のしづらい地域や、インターネットは使えるものの環境整備が遅れている地域を選び、そこのサーバ、パソコンを使う(乗っ取る)という戦略を立てるはずです。つまり「海外を経由するならば、わりと簡単」という結論に結びつきます。

フィッシングの電子メール対策を考える

URLを電子メールに添付するだけでは、その先のサーバを検査しない限り、そこがフィッシングサイトかどうかはわかりません。

パソコンの電子メールのユーザアプリケーションにおけるユーザインターフェースは、いい意味で

も、悪い意味でも良くできており、ユーザがその内部の複雑なしくみを意識することなく使えるようにできています。HTMLフォーマットで送付されるとき、ユーザには「このボタンをクリック」とか「銀行サイトへ」といった形で表示され、その実際のURLはわかりません。人間が理解する情報と、実際の中身の情報が違うのですから、専門家でもない人にわかれというほうが無理です。

また、そもそも論ですが、フィッシングサイトというサイトはありません。既存の、あるいは新規のサーバがフィッシングに用いられるサイトになるだけです。つまり、いったん、誰かがフィッシングサイトであると認定し、その情報を広く配布しない限り、どこがフィッシングサイトなのかという情報を共有できません。

いまだに「電子メールにある怪しいサイトなどには～」という類の注意書きを目にしますが、人間に怪しいかどうか判断することを求めているならば、それは効果がないどころか逆効果です。なぜならば、その言葉を裏返して読んでしまい「怪しくないならばアクセスして良いという錯覚」を起こしてしまうからです。そもそもフィッシングのメールを送る側は、「怪しいサイトだと思わせないように工夫」をこらしたものを送ってくるのですから、その結果は「怪しくないならばアクセスして良い」ということに結びつくはずです。

フィッシングの対策としては、「電子メールで送られてくるURLを使ってアクセスしたサイトにはいっさい入力はしない」、あるいは「自動的にフィッシングの検知をするブラウザを使う」のどちらかです。動的にフィッシングの検知をするブラウザを使うというのは、完璧ではありませんし、標的型攻撃には効果は薄いです。それであっても無防備よりはまだ十分に意味があります。

ちなみに筆者はEmacsのメールリーダーを使っているのですが、つい最近までHTMLのメールにアンカータグ（<a href="URL">であるタグ）がある場合には、それをクリックしても勝手にブラウザで開くことがないような設定にしていました。FirefoxがVer 3になってフィッシング検知機能がつき、その機能が信頼できるレベルだと判断したので、今では、Firefox 3を呼び出す形でアクセスできるようにしています。

フィッシング対策は情報共有から

最初に答えをいうと個人の心がけや判断ではフィッシングに対応できません。もちろん筆者のように「全部のメールのURLを無効にする」「URLを開く場合は、わざわざURLをコピーしてブラウザに貼り付ける」といった極端な方法をとるならば可能ですが、さすがにこれは現実的ではありません。個人のユーザを効果的に守るには、システムがアシストするしかありません。しかし、それは裏方がいて情報共有するしくみがあって初めて可能になります。

日本国内では、2005年にフィッシング対策協議会[注2]（**図1**）が作られ、事例情報、技術情報を収集し、提供する活動が行われています。総務省、経済産業省、警察庁、金融庁などの省庁やISP、銀行、電子

◆図1　フィッシング対策協議会のサイト

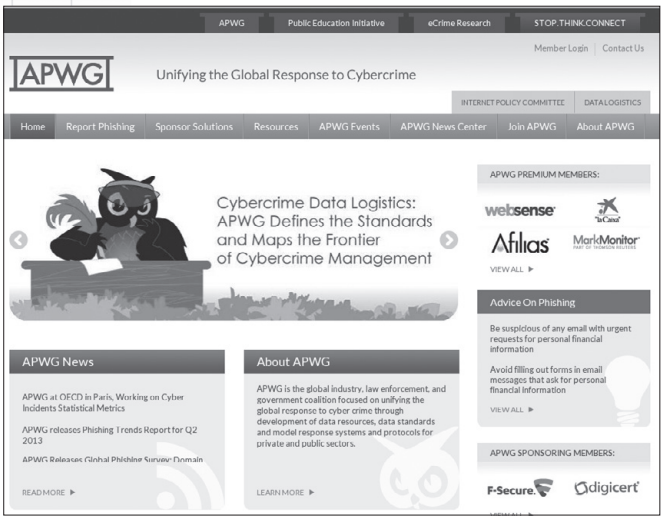
注2）http://www.antiphishing.jp/　Twitterアカウントは　@antiphishing_jp

商取引サイト、セキュリティベンダがメンバーになっています。

世界的にはAnti-Phising Working Group[注3]（**図2**）という組織があって、世界中にあるフィッシング対策のグループが情報交換をするしくみがあります。

フィッシング対策協議会ではフィッシングサイトが報告／発見された場合、関係団体に働きかけ、そのサイトを停止する役割も果たしています。また、フィッシングサイトのURLをサービス事業者へ提供するしくみがあり、セキュリティベンダや通信事業者などは、それらの情報を共有し、システムでフィッシングサイトへのアクセスを警告、あるいは遮断するようになっています。

◆図2　APWG（Anti-Phising Working Group）のサイト



問題はタイムラグと情報の拡散

フィッシングに対応する組織が活動していても、もちろん限界があります。たとえば、フィッシングが発生してから、そのフィッシング対応をするまでのタイムラグです。

Step 0：フィッシングを計画／フィッシングサイトの入手および構築／フィッシングメールの送付
Step 1：ユーザがフィッシングを認知し対応組織に通報
Step 2：フィッシング情報を確認して対応に入る
Step 3：フィッシングの警告告知
Step 4：フィッシングのURLを提供
Step 5：フィッシングサイトの停止（テイクダウン）

Step 1

Step 0はフィッシング犯側ですので我々は次のステップであるStep 1からの話になります。

たとえばeBayを使っていないユーザに「eBayのパスワードを変更しろ」というメールが送られてきたら、すぐに「これはフィッシングのメールだな」と気がつくと思います。ところが、これが実際に自分が使っている商取引サイトからのメールを偽っていたらどうでしょうか。この連載でも過去に取り上げていますが、商取引サイトや会員サイトから大量のユーザのメールアドレスとパスワードが漏れています。ただし、パスワードの保護のしくみがしっかりしていれば、計算してパスワードを探すにも骨が折れます。

そこで、このメールアドレスを使い、次の文面が書かれたうえで送られてきたらどうでしょうか。

「報道のとおりシステムからパスワードが流出しました。ユーザの安全のために速やかにパスワードを変更してください。現在、サイトは復旧中のため仮サイトから登録をお願いしております」

わざわざHTMLメールではなく、URLをかくすことなく堂々と、しかもSSLサーバ証明書まで取得しているこんなURLが張ってあったなら、どうでしょうか。

https://secure-password.example.com[注4]

かなりの数のユーザがだまされることが想像でき

注3）**http://antiphishing.org/**
注4）もちろん、こんなサイトは存在しません。SSLサーバ証明書にも盲点がある点に関しては、前回の本連載をご覧ください。

ます。とくにテレビや新聞で、「大量のパスワードが流出」などと大きな報道があって、それをすでに見聞していた場合、「何かしなければいけない」という心理になっているはずです。

一方で、だまされる人がいれば、少数とはいえ必ず見破る人も出てくるわけで、そこから対応組織に通報するという善意の行動も生まれるでしょう。

Step 2～4

組織に通報してからの行動ですが、すべての組織が24時間体制ということはできず、また、関連している企業や団体も営業時間や組織の体制との兼ね合いがありますから一晩越してしまうこともあるでしょう。ですが、基本的にはStep2からStep4までは短い時間で対応されます。

ただし、一般に情報を告知しても、その告知が必要な人に必要なタイミングで届くかどうかはわかりません。Twitter上にフィッシング対策協議会のアカウント「@antiphishing_jp」があるのですが、フォロワーも少なく、あまり注目されていないようです。また、フィッシング対策協議会のアカウントも2013年12月10日現在、公式アカウントとはなっていないのもちょっと気になるところです。

Step 5

いくら告知しようとも、あるいは、システムがフィッシング情報を持っていて排除しようとしても、そこから漏れてしまう人たちは大勢いるはずです。フィッシングサイトを停止させないことには、根本的な解決にはなりません。ちなみに停止というのを「テイクダウン」という言葉で表現するときがありますが、これはアメリカンフットボールのボールを持ったプレイヤーを止めるときの言葉「テイクダウン」から来ています。

国内ですと、Step2～4と同じレベルで迅速に対応される場合がほとんどだと思います。一方、サイトが日本国外の場合、まず日本国内から海外の対応窓口へコンタクトを取ります。多くの国も同じくフィッシングの問題を抱えていますので、国際的な連携は取れており、国際間の連絡はスムーズにいくと思われます。しかし、日本国内と同様のスピードで解決されるとは限りません。その点は十分に理解すべきでしょう。

国内外に限らずサイトが乗っ取られている場合などは、単純にフィッシングサイトの停止という対応だけでは済まない場合も考えられます。

このようにStep 5に関してのタイムラグは想定しておく必要があり、またその期間のリスクを理解する必要があるでしょう。

最後に

繰り返しますが、フィッシングの対応は、個人で何かを判断するというアプローチでは限界があり、システムのアシストがない限り難しいということを重ねて理解してほしいと思います。組織であればアンチフィッシングのシステムを入れる、個人でパソコンを使っていれば、やはりアンチフィッシングのセキュリティソフトを入れる、あるいは電子メールから使うWebブラウザはフィッシング警告の機能のあるものを使うといった方法を選ぶのが属人性にとらわれず、かつ最も効果的な対策です。

逆にこれまでよく言われていた「怪しいサイトだからクリックしない」というアプローチは問題であり、怪しいサイトかどうかは深い洞察力がない限り判断は難しく、むしろ安易に「怪しいサイトではない」という判断をしてしまい、結果的に「怪しいサイトと思わないからクリックする」ということにつながることを強く指摘したいと思います。

日本国内ではフィッシング対策協議会、世界ではAPWGのような組織が活動していることにより、ユーザは安全なインターネット環境を利用できていることを理解してほしいと思います。また、電子商取引やフィッシングの対象になる企業も、フィッシング対策の組織の存在を理解してほしいです。みんながそれらの組織と連帯し、より一層の効果的なフィッシング対策が進むことを願って、今回は終わりたいと思います。SD

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10回

アプリ内のデータを保存する

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。今回はこれまで作ってきたブックアプリに、アプリを終了しても覚えておきたいデータを保存する機能を追加する方法を解説します。

1回だけのアプリで終わらせないために

これまでの連載でブックアプリだけでなくクイズアプリやノベルゲーム、時計アプリなどいろいろなアプリを作ることができる知識を得たかと思います。今回はこれらのアプリに必要となるであろうアプリ内のデータの保存方法を学びましょう。ブックアプリをどこまで読んだかやゲームの得点、各種アプリの設定などを、アプリを終了しても覚えているようにするために必要な機能です。

これができるようになればアプリが1回こっきりのものではなく、前回の続きから本を読んだり、ゲームを遊んだ結果が蓄積されていくようなことが実現でき、アプリの楽しみ方に奥行きを持たせられるようになります。それでは、さっそく学んでいきましょう！

各ページの表示回数を記録する

応用が利きやすいように、今回は次のようなシンプルな形で機能を追加します。

- 各ページが閲覧された回数を数える
- 各ページにその閲覧回数を表示
- 閲覧回数を保存し次回起動時にも正しく続きから数えられるようにする

完成イメージは**図1**になります。

▼図1　今回の完成イメージ

ステップ1

まずは1ページ目のViewControllerに必要な変数を追加しましょう。ヘッダファイル「Page1ViewController.h」を開いてください。

`@interface Page1ViewController : UIViewController <UIActionSheetDelegate> { …… }`の中に次の2行を追加してください。

```
NSInteger pageReadNumber;
UILabel *readNumberLabel;
```

これは、1ページ目が読まれた回数を収納しておくための変数(pageReadNumber)とその回数を表示するためのラベル(readNumberLabel)を定義しているコードになります。

追加位置はどこでもかまいませんが、作例では先月号で追加したコードの下に追加しました(**図step1**)。

step1

```
@interface Page1ViewController : UIViewController <UIActionSheetDelegate>{

    IBOutlet UIImageView *page1Image;

    IBOutlet UILabel *titleLabel;
    IBOutlet UILabel *bodyLabel;

    IBOutlet UIImageView *carImage;

    IBOutlet UIButton *goldBtn;
    IBOutlet UIButton *mosqueBtn;

    AVAudioPlayer *sound;

    UILabel *clockLabel;
    NSTimer *clockTimer;

    NSInteger pageReadNumber;
    UILabel *readNumberLabel;
}
```

ステップ2

続いて「Page1ViewController.m」に実際のコードを記述していきましょう。

まずは**- (void)viewDidLoad{ …… }**の中に次のコードを追加します。

```
//ページ毎の閲覧回数の初期値を設定
NSMutableDictionary *defaults = [NSMutableDictionary dictionary];
[defaults setObject:@"0" forKey:@"PAGE1_NUMBER"];
[[NSUserDefaults standardUserDefaults] registerDefaults:defaults];
```

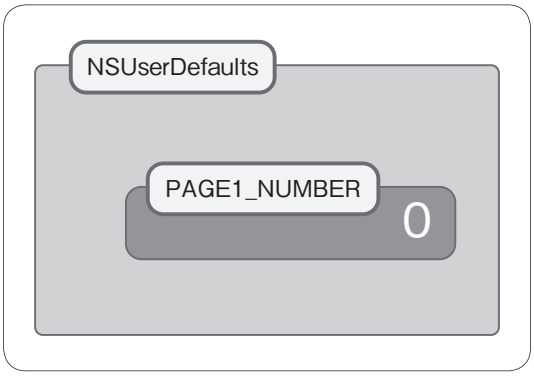
アプリ停止後も保存しておきたいデータを記録するには、**NSUserDefaults**というしくみを使用します。

イメージとしてはアプリごとにNSUserDefaultsという箱があり、その中にさまざまなデータを「名前をつけて」保存しておき、その値を取り出すときはその「名前」を使って呼び出すと思ってください。この名前のことを「キー」と呼びます。

そして先ほどのコードは、その保存するデータの一番最初の値をNSUserDefaultsに保存するためのコードです。1ページ目の閲覧回数に「PAGE1_NUMBER」というキー(名前)をつけ、また初期値として「0」を設定してNSUserDefaultsに保存しています。

この保存のイメージを図で表すと**図step2**のようになります。

step2



ステップ3

次に、この閲覧回数をページに表示するための、ラベルを配置するコードを追加します。先ほどのコードに続いて次のコードを追加してください。

```
    //ページ毎の閲覧回数を表示
    readNumberLabel = [[UILabel alloc] initWithFrame:CGRectMake(270, 10, 40, 20)];
    readNumberLabel.textAlignment = NSTextAlignmentCenter;
    readNumberLabel.textColor = [UIColor grayColor];
    readNumberLabel.backgroundColor = [UIColor whiteColor];
    [readNumberLabel setFont:[UIFont boldSystemFontOfSize:14]];
    [self.view addSubview:readNumberLabel];
```

先月行ったように、コードのみでラベルを配置しています。ラベルの位置を右上(270, 10, 40, 20)に、文字色をグレー(grayColor)、ラベルの背景を白(whiteColor)、文字の大きさを14ポイント(boldSystemFontOfSize:14)に設定しています。

ステップ4

そして、ページが表示されるたびに閲覧回数を増やして、表示を更新するコードを記述します。- (void)viewDidLoad{ …… }の後に次のコードを追記してください。

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    //ページ毎の閲覧回数を更新
    pageReadNumber = [[NSUserDefaults standardUserDefaults]integerForKey:@"PAGE1_NUMBER"];
    pageReadNumber++;
    readNumberLabel.text = [NSString stringWithFormat:@"%d",pageReadNumber];
    [[NSUserDefaults standardUserDefaults] setInteger:pageReadNumber forKey:@"PAGE1_NUMBER"];
    [[NSUserDefaults standardUserDefaults] synchronize];
}
```

この- (void)viewWillAppear:(BOOL)animated{ …… }というのはUIViewControllerの主要メソッドの1つで、ViewControllerが画面に表示される直前に呼び出されるものです。

この中にページごとの閲覧回数をカウントし、ラベルの表示を更新する処理を書けば、このページが表示されるたびにページ上の数字が1つずつ増えることになります。

それではこのコードの解説をしていきましょう。

pageReadNumber = [[NSUserDefaults standardUserDefaults]integerForKey:@"PAGE1_NUMBER"];の部分は、pageReadNumberにNSUserDefaultsからデータを取り出しています。このときに、初期値を設定したときと同じキー「PAGE1_NUMBER」を使用しています。

そして取り出した値をカウントアップしているのがpageReadNumber++;の部分です。この「変数＋＋」という書き方はインクリメントと言い、変数の値を1増やす動作をします。つまり変数が0のときにインクリメントすると変数は1になり、さらにインクリメントすれば2になるということです。

次の行ではラベルに表示された数字を更新しています。readNumberLabel.textに表示したい文字列を代入するのですが、pageReadNumberは文字ではなく数字ですので、stringWithFormatを使用して文字に整形しています。

最後にインクリメントされたpageReadNumberをNSUserDefaultsに書き戻す必要があります。[[NSUserDefaults standardUserDefaults] setInteger:pageReadNumber forKey:@"PAGE1_NUMBER"];の部分でpageReadNumberの内容をPAGE1_NUMBERのキーをつけ、NSUserDefaultsに

格納しています。その後、[[NSUserDefaults standardUserDefaults] synchronize]; でNSUserDefaultsへの変更を即時反映しています。

このステップで追加したコードで、viewDidLoad～viewWillAppearは**図step4**のようになっているはずです。

step4

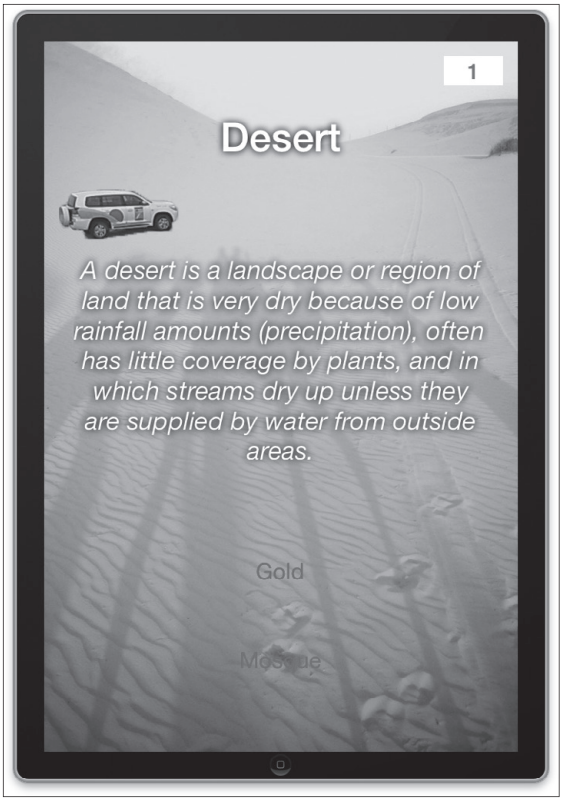
```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    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;



    // 長押しジェスチャー
    UILongPressGestureRecognizer *longPressGesture =
    [[UILongPressGestureRecognizer alloc] initWithTarget:self action:@selector(longPress:)];

    // 指定したビュー（画面全体）にジェスチャーを追加
    [self.view addGestureRecognizer:longPressGesture];


    clockLabel = [[UILabel alloc] initWithFrame:CGRectMake(0, 100, 320, 100)];
    clockLabel.textAlignment = NSTextAlignmentCenter;
    clockLabel.textColor = [UIColor whiteColor];
    [clockLabel setFont:[UIFont fontWithName:@"HelveticaNeue-UltraLight" size:100]];
    [self.view addSubview:clockLabel];
    clockLabel.hidden = YES;
    [self runTimer];

    //ページ毎の閲覧回数の初期値を設定
    NSMutableDictionary *defaults = [NSMutableDictionary dictionary];
    [defaults setObject:@"0" forKey:@"PAGE1_NUMBER"];
    [[NSUserDefaults standardUserDefaults] registerDefaults:defaults];

    //ページ毎の閲覧回数を表示
    readNumberLabel = [[UILabel alloc] initWithFrame:CGRectMake(270, 10, 40, 20)];
    readNumberLabel.textAlignment = NSTextAlignmentCenter;
    readNumberLabel.textColor = [UIColor grayColor];
    readNumberLabel.backgroundColor = [UIColor whiteColor];
    [readNumberLabel setFont:[UIFont boldSystemFontOfSize:14]];
    [self.view addSubview:readNumberLabel];
}

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    //ページ毎の閲覧回数を更新
    pageReadNumber = [[NSUserDefaults standardUserDefaults]integerForKey:@"PAGE1_NUMBER"];
    pageReadNumber++;
    readNumberLabel.text = [NSString stringWithFormat:@"%d",pageReadNumber];
    [[NSUserDefaults standardUserDefaults] setInteger:pageReadNumber forKey:@"PAGE1_NUMBER"];
    [[NSUserDefaults standardUserDefaults] synchronize];
}
```

ステップ5

それでは実行してみましょう。いかがでしょう、**図step5-1**のように1ページ目の右上にカウンターが表示されたでしょうか？　他のページに移動して、再び1ページ目に戻ってくるとちゃんとカウンターが増えているはずです(**図step5-2**)。

また、一度アプリを落として再起動した際も、ちゃんと続きからカウントアップされているか確認しましょう。

step5-1



step5-2

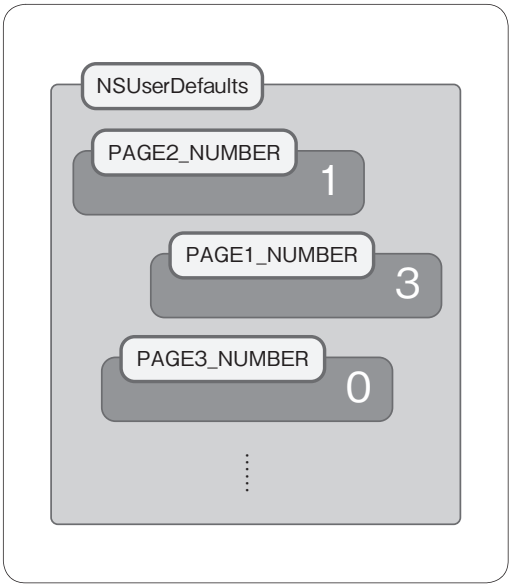
ステップ6

きちんと動作することが確認できましたら、同様な追加を各ページに行いましょう。その際に注意しなければいけないのは、NSUserDefaultsに格納する際につけるキーが重複しないようにすることです。

これまで見てきたようにNSUserDefaultsに格納、また取り出しを行う際にはキーを手がかりに行います。ですので、キーが重複してしまっているとほかのページの値を上書きしてしまったり、予期せぬ不具合の原因となりますので注意しましょう。

今回の作例ではPAGE1_NUMBER、PAGE2_NUMBER、PAGE2B_NUMBER、PAGE3_NUMBER……のようにページ番号に合わせて名付けました(**図step6**)。

step6



初期値について

さて、ここで1つ疑問に思うことはないでしょうか？　ステップ2で記述した初期値をNSUserDefaultsに保存しているところですが、viewDidLoadはViewControllerがメモリにロードされる際に実行されます。つまりアプリを起動するたびなどに、初期値である「0」がNSUserDefaultsに書き込まれてしまうのではないかと心配になりませんか？

実行してみるとわかりますが、実際にはそのようなことにはなりません。それはなぜかと言いますと、初期値を登録する**registerDefaults:**というメソッドの動きが鍵になっています。

registerDefaults:は指定されたキーに対応した値がNSUserDefaultsに存在するときは何もせず、存在しなかった場合にのみ初期値をNSUserDefaultsに収納してくれるからです。とても便利な機能ですので積極的に使っていきましょう。

データの種別について

今回は保存する数値が「ページ数」でしたので整数形式に対応したメソッドintegerForKey:とsetInteger:を使用しました。NSUserDefaultsに値を保存するためには、扱うデータ種別にあわせたメソッドを使用する必要があります。主なメソッドを**表1**に記載いたしますので参考にしてください。

アプリの寿命を延ばそう

いかがだったでしょうか？　これで、アプリ内のデータを次回起動時にも引き継ぐことができるようになりました。アプリを使うたびになにかが蓄積されたり、成長していくような要素を入れるとアプリ自体も長く使ってもらえるようになります。

NSUserDefaultsを使って何ができるか、いろいろ考えてみるのも楽しいと思います。SD

▼表1　データ種別に対応する主なメソッド

データ種別	メソッド	メソッド
整数型	integerForKey:	setInteger:
浮動小数点型	floatForKey:	setFloat:
倍精度浮動小数点型	doubleForKey:	setDouble:
論理型	boolForKey:	setBool:
文字型	stringForKey:	setObject:

本連載のサポートページで記事の補足説明をしています。あわせてご活用ください！
› http://www.gimmiq.net/p/sd.html

いたのくまんぼう／Itano Kumanbow
Twitter @Kumanbow

神奈川工科大学非常勤講師。リオさんとはGimmiQ名義で「MagicReader」（手を使わずにページがめくれる電子書籍ビューワ）をリリース。個人ではNinebonz名義で「江頭ジャマだカメラ」（無料総合1位獲得）、「i列車の車窓から-そうだ! 京都行こう!-」（バーチャル旅行アプリ）などをリリース。アプリ紹介サイト「あぷまがどっとねっと（http://appmaga.net/）」の技術サポータ。

リオ・リーバス／Leo Rivas
Twitter @StudioLoupe

iOSアプリ開発を中心に電子絵本作家・漫画家として活動中。代表作は、KDDI株式会社に社内導入され、世界で40万ダウンロードを記録する革新的な電卓アプリ「FusionCalc」と、国連主催のWSA Mobile 2013を受賞した、顔の動きで電子書籍が読める「MagicReader」。電子絵本はiBookstore/Kindleストア共に児童書カテゴリ総合1位を獲得。

第45回 [アプリ開発2013] 6 アプリの成長のための運用

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！

重村 浩二　SHIGEMURA Koji
日本Androidの会 中国支部長
Mail k-shigemura@android-group.jp

アプリを育てよう

前々回、前回と、LikeSiriというアプリを通してアプリの発想から実装、さらにアップデートを行う流れを見てきました。アプリをGoogle Playストアなどに公開すれば、ユーザからの意見や、皆さんが思いついたアイデアを追加で実装することによって、ユーザの増減が発生します。

今回は、アプリを育てるために必要となる運用面に注目してみたいと思います。

Developer Consoleでアプリの状況確認

完成したアプリは、まずはGoogle Playストアに公開することを考えるでしょう。現在のAndroidアプリの大半はGoogle Playストアから配信され、ユーザがダウンロードしているためです。開発者の皆さんは「Developer Console」でアプリを公開することになります。

Developer Console
URL https://play.google.com/apps/publish/

公開すると、Developer Console側で利用者のダウンロード数やアンインストール数が日々記録されるようになります。このとき、ユーザと開発者とでは蓄積される情報を見る視点が違います。ユーザが気になるのは、「みんなが使っているのはどのアプリ？」ということなのではないかと筆者は考えています。そこでユーザには、「先日公開のアプリ、XXXのダウンロード数が○○○○を突破しました！」などとTwitterやFacebookといったソーシャルメディアやブログを通して展開していくことになります。

しかし、開発者が気にするべきはダウンロード数だけではありません。Developer Consoleから得られる情報にはもっと重要な情報が含まれています。その1つは、「現在どれぐらいのユーザが端末にインストールしてくれているのか」という情報です。インストールされている端末の数が、現在のユーザ数とほぼ一致すると考えられます。Developer Consoleのトップ画面では、各アプリの「現在のインストール数／総インストール数」という表示が行われています。

Developer Consoleでは、それ以外にも次のような情報を確認することができます。

- 1日のインストール数（端末数）
- 1日のアンインストール数（端末数）
- 1日のアップグレード数（端末数）
- 現在のインストール数（ユーザ数）
- 総インストール数（ユーザ数）
- 1日のインストール数（ユーザ数）
- 1日のアンインストール数（ユーザ数）
- 昨日の平均評価
- 累計平均評価

アプリの成長状況を確認する

筆者の場合は前述に加えて次の情報を定期的に確認するようにしています。

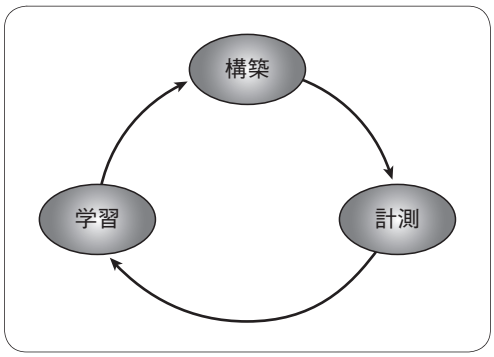
> 1日のインストール数(端末数) －1日のアンインストール数(端末数)＝1日のインストール増減数(端末数)

1日のインストール増減数を日々比較しながら確認することで、現在そのアプリが「成長フェーズ」にあるのか、「衰退フェーズ」にあるのかということを簡単に確認できます。増加数が低いようであれば、プロモーションなどに力を入れてユーザに知ってもらうことに注力します。逆に減少しているようであれば、アプリの操作性や見た目、機能への不満などが考えられると思いますので、そういったポイントから重点的に改善方法を模索します。高い増加数を出していても安心はできません。数日間の動向を確認し、増加しているダウンロードがどこから流入してきているのかを可能な限り調査しましょう。レビューサイトで掲載されたことがきっかけとなることもありますし、口コミだけで伸びていることも考えられます。大事なのは、高い増加数が一時的なものなのか、継続して得られているものなのかを理解しておくことです。このアプローチは、とくにアプリのアップデートを行った後のタイミングに既存のユーザがどう反応したのかを見るのによい指標となります。

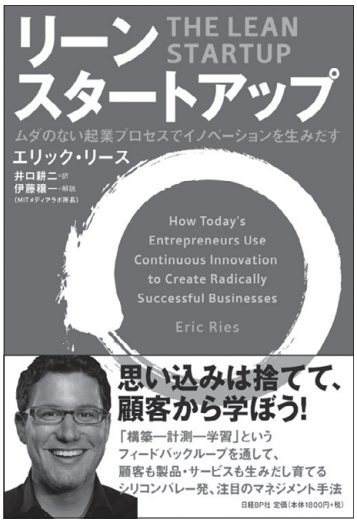
アプリを成長させていくには、**図1**のようなアプローチをとるのがよいでしょう。この図は、『リーン・スタートアップ[注1]』(**写真1**)という書籍で取り上げられている手法で、起業や新規事業を立ち上げるプロセスで実践するとよいとされています。この手法は、アプリの開発でも応用することが可能です。

まず、課題などからどのようにアプリを改善すれば利用ユーザ数が増加するのか「仮説」を立て、それをもとに改善点をアプリに実装して「構築」します。ここでの「仮説」には、「目標」が含まれています。前述でしたら「利用ユーザ数の増加」が目標ですし、ショッピング系のアプリであれば「指定の画面を呼び出した回数が○○回に到達すること」が目標となります。次に、構築したアプリをユーザに使ってもらうことで、仮説のなかで立てた目標に到達できそうかを「計測」します。仮説を立てるときの目標は、数値化して分析できるものにしておくことが計測をするうえでの大事な点です。そして、「計測」することで導き出された成果をもとに、改善した効果が出たのか、出なかったのかを「学習」します。立

▼図1　リーン・スタートアップのモデル



▼写真1　『リーン・スタートアップ』



注1) エリック・リース 著、井口 耕二 訳、伊藤 穣一(MITメディアラボ所長)解説／日経BP社 刊／ISBN：978-4-8222-4897-0／発行日：2012年4月16日

てた目標によって、得られる成果も、学べる内容も変わってくるでしょう。大事なのは、ここから学んだことをもとにして、次のアクションにつなげることです。そのために再びアプリの改善点を導き出す次の「仮説」を立て、これをもとにアプリを改善し、「構築」を行っていくことです。このようなサイクルを回していくことで、アプリをより良いものへと進化させていくことができるのです。

アプリの利用状況を分析する

前述したDeveloper Consoleから得られる情報も、仮説をもとに「構築」したアプリを「計測」した情報として見ることができると思います。しかし、それだけでは本質的な情報は得られません。Developer Consoleから得られる情報は、アプリを「インストールしてくれている人の数」であって、「アプリを利用してくれている人の数」ではないのです。

「このアプリ、使いやすそう」と感じてインストールしてくれたユーザが増えたとしても、日々アプリを利用をしてくれているアクティブユーザとはイコールではないことを見誤ってはいけません。この両者の違いこそが改善の仮説を立てるうえで重要なポイントです。

その情報を補完するためにGoogleから提供されているのが、「Google Analytics」です。

Google Analytics
URL http://www.google.com/intl/ja/analytics/

Google Analyticsを導入することで、現在利用しているユーザの状況をほぼリアルタイムに分析することが可能となります。

取得できる情報には以下のようなものが含まれています。

- 新規ユーザ数
- アクティブユーザ数
- セッション数
- 国／地域
- 端末
- 閲覧したスクリーン数
- 平均セッション時間
- セッション別スクリーン数

これら以外にも、さまざまな視点から情報を分析するための情報が詰まっているのがGoogle Analyticsなのです。

Google Analyticsの導入

Google Analyticsを導入するには、まず前述のGoogle Analyticsのサイトでアプリ用のアカウントを作成する必要があります[注2]。ユーザアカウントを作成後、アプリ用のアカウントを作成することで利用が可能になります。

アプリ用のアカウントを作成すると、トラッキングID（UA-XXXXXXXX-X。Xは英数字）が発行されるので、こちらをメモしておいてください。このアカウントをアプリ側に設定して、アプリの操作情報をGoogle Analyticsに蓄積します。

アプリ側にGoogle Analyticsを導入するには、Google Developersより「Google Analytics SDK for Android v3（Beta）」をダウンロードします。「Beta」とありますが、サイト上ではGoogleからこちらを推奨（Recommended）されています。Android SDKからもGoogle Analyticsがダウンロードできますが、あちらは「v2」と、1つ古いバージョンとなっていますのでインストール時には注意してください。

Google Analytics SDK for Android v3(Beta)のダウンロード先
URL https://developers.google.com/analytics/devguides/collection/android/resources?hl=ja

zipファイルで提供されているので解凍すると

注2）本記事ではGoogle Analyticsのアカウント作成に関する詳細な手順は省略します。アカウント作成の手順については公式サイトなどを参考にしてください。

▼リスト1　パーミッションの許可

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

「libGoogleAnalyticsServices.jar」というライブラリが格納されています。こちらを導入したいプロジェクトの/libsディレクトリにコピーし、Build PathでExportされるように設定を行ってください。これでGoogle Analytics利用のためのメソッドが利用可能となりました。

■実装方法

Google Analyticsで分析を行うアプリには、**リスト1**にあるパーミッションの許可を行います。そして、トラッキングを行いたいActivityに対して、**リスト2**のような実装を追加します。EasyTrackerというクラスを利用して、Activityの開始と終了をactivityStart/activityStopメソッドで記録していきます。

最後にres/valuesディレクトリ配下に、**リスト3**にあるようなanalytics.xmlというリソースファイルを用意すれば、基本的な情報の収集が可能となります。もしもlintチェッカーでトラッキングIDに含まれている「-」でワーニングが出る場合には、**リスト4**のように、<resources>タグに対するlintチェックを無視するよう設定してください。

この状態にしてアプリをアップデートすることで、日々のアクティブユーザ数、そのうちの新規ユーザ数、さらにそのユーザがどの国／地域からアクセスしているのか、端末はどんな端末(モバイルなのか、タブレットなのか、メーカーの詳細)などなど、さまざまな利用状況の詳細情報を追加で取得することが可能となります。

▼リスト2　トラッキングの開始と終了

```
@Override
public void onStart() {
    super.onStart();

    // トラッキングの開始
    EasyTracker.getInstance(this).activityStart(this);
}

@Override
public void onStop() {
    super.onStop();

    // トラッキングの終了
    EasyTracker.getInstance(this).activityStop(this);
}
```

▼リスト3　analytics.xmlの準備

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
  <!--トラッキングIDの指定。取得した値に差し替えてください-->
  <string name="ga_trackingId">UA-XXXXXXXX-X</string>

  <!--Activityでのトラッキングを許可します-->
  <bool name="ga_autoActivityTracking">true</bool>

  <!--例外処理(異常終了)のトラッキングを許可します-->
  <bool name="ga_reportUncaughtExceptions">true</bool>
</resources>
```

▼リスト4　lintチェックの無視

```
<resources
  xmlns:tools="http://schemas.android.com/tools"
  tools:ignore="TypographyDashes">
```

Google Analytics利用時の注意点

このように利便性の高いGoogle Analyticsですが、導入を行うときにGoogle Analytics利用規約とSDK Policyに同意する必要があります。この中でもとくに重要なのが、SDK Policyに含まれている「Google Analyticsによる情報収集に関してはユーザからの同意を得る」という内容です。

Google Analyticsは使い方によっては個人情報に当たる情報まで収集することが可能なため、そのような情報は取らないようにと、事前に

▼リスト5　利用許諾の確認

```
// プリファレンスから使用許諾の取得
SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences(this);
if (!prefs.getBoolean(KEY_AGREEMENT, false)) {
    // アプリ使用許諾用画面の呼び出し
    Intent intent = new Intent(this, AgreementActivity.class);
    startActivityForResult(intent, REQ_AGREEMENT);
    GoogleAnalytics.getInstance(getApplicationContext()).setAppOptOut(true);
} else {
    // KEY_AGREEMENTのboolean値がtrueであれば、情報の収集開始
    GoogleAnalytics.getInstance(getApplicationContext()).setAppOptOut(false);
}

// Activityからの応答処理
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQ_AGREEMENT) {
        if (resultCode == RESULT_CANCELED) {
            // アプリ使用許諾用画面でキャンセルされたら、アプリを終了する例
            finish();
        }
    }

    super.onActivityResult(requestCode, resultCode, data);
}
```

Google Analytics側で規約やポリシーが作られています。

SDK Policy

URL https://developers.google.com/analytics/devguides/collection/protocol/policy

ユーザから許可が得られない情報を収集しないしくみをオプトアウトと言います。Google Analyticsを利用する1つの実装方法として、**リスト5**にあるように使用許諾の画面を呼び出し、そちらで許可が得られたらアプリが利用できるようなしくみにするのがよいでしょう。setAppOptOutメソッドで、Google Analyticsのオプトアウトが有効化されているのか、されていないのかを設定することができます。このしくみの実現方法は、前回の記事で取り上げたShared Preferencesを用いたデータの保存を用いるのがとても簡単ですので、ぜひ取り入れてみてください。

アクティブユーザを増やすための施策

前項までに示したGoogle Analyticsを利用することで、利用状況まで取得できるようになりました。最後に、アクティブユーザを増やすための施策について考えてみたいと思います。Android上で実現できる手法には、以下のようなものが考えられるでしょう。

❶広告やソーシャルメディアを利用した宣伝
❷ステータスバー上にアプリ利用促進のメッセージ表示
❸アプリ評価依頼のメッセージ表示
❹定期的なアプリの更新

❶の広告やソーシャルメディア、レビューサイトを利用した宣伝は、新規ユーザを獲得するために手軽に使える手段の1つかと思います。アプリレビューサイトのAndroiderへの登録や、TwitterやFacebookでの宣伝はぜひ漏らさずしておくべきでしょう。

❷❸のステータスバー上へのアプリ利用促進のメッセージ表示とアプリ評価依頼のメッセージ表示は、少し敷居が上がる手法です。

定期的にアプリを利用してもらうため、たとえば家計簿系のアプリの中には、1日アプリの利用がなければ、入力を促すためにステータスバー上にメッセージを表示するものがあります。

アプリの評価依頼は、Google Playストアでのレビュー評価が多いほうがそれだけ新規ユーザに注目してもらえることから、アプリ内で一定回数の利用があったことを確認してから、評価依頼を行うような仕掛けを組み込むというものです。

筆者の場合、**図2**のような評価ダイアログを一定回数使っていただいたユーザに向けて表示し、評価を促すような試みを行っています。「評価する」というボタンを押せば、Google Playストア上のアプリ画面を呼び出すような仕掛けを実現しています。

これらの手法はうまくいくと効果が高いですが、あまりやりすぎるとユーザに煩わしさを感じさせてしまうため、逆にユーザが離れてしまう結果につながってしまうリスクがあります。使うアプリの特性に合わせて、無理のない範囲で導入することが望ましいでしょう。

❹の定期的なアプリの更新は、大変ですが一番効果のある手段です。とくにすでにアプリをインストールしてくれているユーザに対して利用を促す効果が出ますので、まずはぜひアプリの更新を実施していってください。

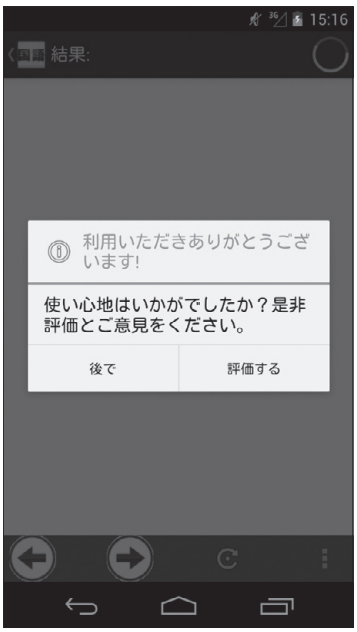
まとめ

今回は「運用」という点に注目して、開発したアプリをいかにして育て上げていくのか、その手法の一部を筆者の実践している手法を交えて紹介しました。

なかでもGoogle Analyticsは大変高機能で、紹介できなかった機能がまだまだたくさんあります。たとえばアプリ内でのボタン押下といったイベント情報をトラッキングすることで、実装した新機能が使われているかどうかを確認する手法などは、UIのA/Bテストを行うときにも有効でしょう。広告に仕掛ければ、より詳細な広告のクリック状況が分析できます。

アプリの改善点はいたるところに転がっているものですので、本稿で紹介した手法を用いながら、より良いアプリへと進化させてください。SD

▼図2　評価ダイアログの表示例



重村 浩二（しげむら こうじ）　日本Androidの会 運営委員 中国支部長

日本Androidの会にて運営委員、中国支部長として毎月山口県・広島県を中心に自作アプリの発表やハッカソン、ハンズオンなどを中心とした勉強会を精力的に開催。個人としても、雑誌への寄稿などを中心に活動中。
URL http://buildbox.net/　Twitter @shige0501

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17回 仮想マシンの初期化とBHyVeのゲストOSローダ

浅田 拓也（ASADA Takuya）Twitter @syuu1228

はじめに

前回は、VT-dの詳細について解説しました。今回は、仮想マシンの初期化とBHyVeのゲストOSローダについて解説していきます。

物理マシンの初期化と仮想マシンの初期化の違い

まず、物理マシンの初期化手順について考えていきましょう。物理マシンの電源投入時には、次のような手順で初期化処理が実行されます。

①物理的な初期化

コンピュータの電源が投入されると、CPUは決められたアドレスから命令の実行を開始します。このアドレスにはROMがメモリマップされており、電源投入時にCPUはROM上のファームウェア[注1]の初期化ルーチンから実行を開始します。電源を上げたときのレジスタ値やメモリ内容、周辺デバイスの状態は初期化されていないため不定ですが、ファームウェアが最低限の初期化を行います。

②ファームウェアのロード

初期化が終わると、ファームウェアはブートローダまたはOSをロードします。

③OSのロード

ブートローダを実行した場合、ブートローダがOSをロードします。この初期化手順はPCに限ったものではなく、スマートフォンのようなデバイスやマイコンのようなより単純なデバイスでもおおむね同じです。

一般的に、CPUのレジスタやメモリ上のデータは電源を遮断すると状態を維持できません。このため、電源投入ごとに上述のような初期化処理を行う必要があります。

しかしながら、仮想マシンははにはこのような物理デバイスの制約がないので、必ずしも物理マシンと同じ初期化手順を踏む必要がありません。

ゲストOSのダイレクトブート

いくつかのハイパーバイザでは、ブートローダーを実行することなく、直接OSをブートするダイレクトブートを実装しています。これは、上述のような仮想マシンの特性を活かしたものです。

たとえばQEMU（KVM）では、次のようなコマンドでホストOS上に置かれたLinuxカーネルをロード・ブートできます。

```
$ qemu -kernel vmlinuz -append "ro root=LABEL=/ " -initrd initrd.img
```

BHyVeでは、このダイレクトブートを利用してゲストOSを起動します。これは、現在BHyVeにBIOSが実装されていないためです。

FreeBSDゲストのロードでは、FreeBSDブートローダをホストOSのユーザランドへ移植しレジスタアクセスやメモリアクセスをゲストマシンに対するアクセスで置き換えています。これによって、通

注1）PCでは通常BIOSまたはUEFIがファームウェアとして用いられています。同じx86 CPUが搭載されていてもファームウェアの仕様が異なれば別々のブートローダを用意しなければならなくなるため、BIOSもUEFIもベンダ間で差異が生じないよう仕様が定められています。

常のブートローダと同じインターフェースを持つOSローダを実装しています。

LinuxゲストやOpenBSDゲストのロードでは、同様の手法でGRUB2をホストOSへ移植することにより[注2]、通常のGRUB2と同じインターフェースを持つOSローダを実装しています。**このとき、ゲストOSローダはゲストOSをメモリ上にロードし、レジスタの初期値を設定し、CPUのモードをプロテクトモードに切り替えてから仮想マシンを始動します。**これは従来ブートローダが行っていたことであり、この方法ではBIOSへ依存するブートローダをホストOS側で動作するプログラムで置き換えています。

ただし、ゲストOSが実行中にBIOSを呼ぼうとするとBIOSコールハンドラが存在しないため、エラーが発生してしまってOSが正常に動作しません。これに対処するには部分的にせよBIOSサポートを導入するしかなく、そのためBHyVeでは動作しないゲストOSが存在します[注3]。

具体的な実装方法

BHyVeでは、VMインスタンスの操作は**/dev/vmm/<VM名>**に対してioctl・mmapなど発行することによって行います。ただし、これらの処理を書きやすくするためにvmmapiというライブラリが用意されているので、通常こちらを利用します。

以降に行いたい操作の種類によってどのような実装を行えばよいかを示します。

VMインスタンスの作成

新しくVMインスタンスを作成するときは、まずsysctlを通して**/dev/vmm/<VM名>**を作ります。このデバイスファイルはBHyVeが保持するVMインスタンスのステート(注:CPU数などの情報、レジスタ値、メモリの内容などを含む)をユーザランドから操作するためのインタフェースになっています。vmmapiではこれを**vm_create()**という名前で定義しています(**リスト1**)。以降のVMインスタンスの操作は**vm_open()**して取得したctxポインタを使用します。

メモリへの書き込み

ページテーブルの作成など、ゲストメモリ空間への書き込みはvmmapiを用いてゲストメモリ空間をmmapすることによって行います(**リスト2**)。

まず、**vm_setup_memory()**でゲストマシンのメモリサイズを指定し、**/dev/vmm/<VM名>**をmmapしてプロセスのメモリ空間へゲストメモリ空間をマップします。

次に、**vm_map_gpa()**へオフセットを渡すことによりポインタを取得できます。

レジスタへの書き込み

ゲストマシンのレジスタへの書き込みはvmmapiを通じてioctlを発行することによって行います(**リスト2**)。セグメントレジスタとそれ以外のレジスタではVMCSのフォーマットが異なるため、APIが異なります。セグメントレジスタでは**vm_set_desc()**で**base**、**limit**、**access**を設定し、**vm_set_register()**で**selector**を設定します。その他のレジスタでは、**vm_set_register()**で値を設定します。

▼リスト1 VMインスタンスの作成

```
vm_create(VM_NAME);
```

▼リスト2 メモリへの書き込み

```
ctx = vm_open(VM_NAME);
vm_setup_memory(ctx, M_MEM_SIZE, VM_MMAP_ALL);
ptr = vm_map_gpa(ctx, addr, len);
```

▼リスト3 レジスタへの書き込み

```
ctx = vm_open(VM_NAME);
vm_set_register(ctx, cpuno, VM_REG_GUEST_RFLAGS, val);
vm_set_desc(ctx,cpuno, VM_REG_GUEST_CS, base, limit, access);
vm_set_register(ctx, cpuno, VM_REG_GUEST_CS, selector);
```

注2) https://github.com/grehan-freebsd/grub2-bhyve
注3) FreeBSD/i386など。

コンソールへの文字列表示

コンソールへの文字列表示に関しては、printf()やputs()を用いればよいため、vmmapiは使用しません。

ディスクの読み込み

ディスクイメージは通常のファイルであるため、通常のファイルAPIを使用できます。

このため、vmmapiは使用しません。

簡易ローダの実装例

BHyVeにおけるOSローダの実装方法を例示するため、シリアルコンソールにアスタリスクを表示し続けるだけの簡単なプログラムをゲストマシンへ

▼リスト4　簡易ローダのソースコード

```
#include <sys/cdefs.h>
__FBSDID("$FreeBSD$");
#include <sys/param.h>
#include <sys/stat.h>
#include <machine/specialreg.h>
#include <machine/vmm.h>
#include <x86/segments.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <vmmapi.h>

#define VM_NAME    "test1"
#define VM_MEM_SIZE    128 * 1024 * 1024UL

#define MSR_EFER  0xc0000080
#define CR4_PAE   0x00000020
#define CR4_PSE   0x00000010
#define CR0_PG    0x80000000
#define  CR0_PE   0x00000001
#define CR0_NE    0x00000020

#define PG_V  0x001
#define PG_RW   0x002
#define PG_U  0x004
#define PG_PS   0x080

#define ADDR_PT4    0x2000
#define ADDR_PT3    0x3000
#define ADDR_PT2    0x4000
#define ADDR_GDT    0x5000
#define ADDR_STACK    0x6000
#define ADDR_ENTRY    0x10000

#define DESC_UNUSABLE    0x00010000

#define GUEST_NULL_SEL  0
#define GUEST_CODE_SEL  1
#define GUEST_DATA_SEL  2
#define GUEST_GDTR_LIMIT(3 * 8  - 1)

int
main(int argc, char** argv)
{
  struct vmctx *ctx;
  uint64_t *gdt, *pt4, *pt3, *pt2;
  int i;
  unsigned char *entry;
  unsigned char program[] = {
```

①ゲストマシンで実行するプログラムです。
ALレジスタに*(ASCIIコードで0x2a)をロード、DXレジスタにシリアルポートのデータレジスタのアドレスをロード、outbでALレジスタの内容をDXレジスタで指定したポートへ書き込み、を繰り返し行っています。ここではプログラムのロード処理を省略するため、配列上にプログラムのHEXダンプを持ってきています。

```
  0xb0, 0x2a,          /* mov $0x2a,%al */
  0xba, 0xf8, 0x03, 0x00, 0x00,/* mov $0x3f8,%edx */
  0xee,           /* out %al,(%dx) */
  0xeb,0xfd           /* jmp 7 <loop> */
};

vm_create(VM_NAME);
ctx = vm_open(VM_NAME);
vm_setup_memory(ctx, VM_MEM_SIZE, VM_MMAP_ALL);

pt4 = vm_map_gpa(ctx, ADDR_PT4, sizeof(uint64_t) * 512);
pt3 = vm_map_gpa(ctx, ADDR_PT3, sizeof(uint64_t) * 512);
pt2 = vm_map_gpa(ctx, ADDR_PT2, sizeof(uint64_t) * 512);
gdt = vm_map_gpa(ctx, ADDR_GDT, sizeof(uint64_t) * 3);
entry = vm_map_gpa(ctx, ADDR_ENTRY, sizeof(program));

bzero(pt4, PAGE_SIZE);
bzero(pt3, PAGE_SIZE);
bzero(pt2,PAGE_SIZE);
for (i = 0; i < 512; i++) {
    pt4[i] = (uint64_t)ADDR_PT3;
    pt4[i] |= PG_V | PG_RW | PG_U;
    pt3[i] = (uint64_t)ADDR_PT2;
    pt3[i] |= PG_V | PG_RW | PG_U;
    pt2[i] = i * (2 * 1024 * 1024);
    pt2[i] |= PG_V | PG_RW | PG_PS | PG_U;
}

gdt[GUEST_NULL_SEL] = 0;
gdt[GUEST_CODE_SEL] = 0x0020980000000000;
gdt[GUEST_DATA_SEL] = 0x0000900000000000;

memcpy(entry, program, sizeof(program));

vm_set_desc(ctx, 0, VM_REG_GUEST_CS, 0, 0, 0x0000209B);
vm_set_desc(ctx, 0, VM_REG_GUEST_DS, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_ES, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_FS, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_GS, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_SS, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_TR, 0, 0, 0x0000008b);
vm_set_desc(ctx, 0, VM_REG_GUEST_LDTR, 0, 0, DESC_UNUSABLE);
vm_set_desc(ctx, 0, VM_REG_GUEST_GDTR, ADDR_GDT, GUEST_GDTR_LIMIT, 0);

vm_set_register(ctx, 0, VM_REG_GUEST_CS, GSEL(GUEST_CODE_SEL, SEL_KPL));
vm_set_register(ctx, 0, VM_REG_GUEST_DS, GSEL(GUEST_DATA_SEL, SEL_KPL));
vm_set_register(ctx, 0, VM_REG_GUEST_ES, GSEL(GUEST_DATA_SEL, SEL_KPL));
vm_set_register(ctx, 0, VM_REG_GUEST_FS, GSEL(GUEST_DATA_SEL, SEL_KPL));
vm_set_register(ctx, 0, VM_REG_GUEST_GS, GSEL(GUEST_DATA_SEL, SEL_KPL));
vm_set_register(ctx, 0, VM_REG_GUEST_SS, GSEL(GUEST_DATA_SEL, SEL_KPL));
vm_set_register(ctx, 0, VM_REG_GUEST_TR, 0);
vm_set_register(ctx, 0, VM_REG_GUEST_LDTR, 0);

vm_set_register(ctx, 0, VM_REG_GUEST_CR0, CR0_PG | CR0_PE | CR0_NE);
vm_set_register(ctx, 0, VM_REG_GUEST_CR3, ADDR_PT4);
vm_set_register(ctx, 0, VM_REG_GUEST_CR4, CR4_PAE | CR4_VMXE);
vm_set_register(ctx, 0, VM_REG_GUEST_EFER, EFER_LMA | EFER_LME);
vm_set_register(ctx, 0, VM_REG_GUEST_RFLAGS, 0x2);
vm_set_register(ctx, 0, VM_REG_GUEST_RSP, ADDR_STACK);
vm_set_register(ctx, 0, VM_REG_GUEST_RIP, ADDR_ENTRY);

  return (0);
}
```

②vm_create()でvmm.koへsysctlを発行し、VMインスタンスを作成します。作成したインスタンスは/dev/vmm/<VM名>で表され、ioctlで制御できます。

③vm_setup_memory()でゲストマシンのメモリサイズを指定し、/dev/vmm/<VM名>をmmapしてプロセスのメモリ空間へゲストメモリ空間をマップしています。

④vm_map_gpa()でゲストメモリ空間へのポインタを取得しています。ゲストメモリ空間上のアドレスは引数で指定しています(ここではADDR_PT4 = 2000h)。vm_map_gpa()では渡されたアドレスをオフセットとしてポインタを計算します。ここでは、ページテーブル(pt4,pt3,pt2)、GDT、プログラム領域(entry)のアドレスを指定してそれぞれのポインタを取得しています。

⑤ゲストメモリ空間上のページテーブルを初期化しています。

⑥ゲストメモリ空間上を初期化しています。

⑦ゲストメモリ空間へ①で記述したprogramをコピーしています。

⑧各セグメントレジスタを初期化しています。

⑨GDTRに作成したGDTのアドレスをセットしています。

⑩CR0レジスタにプロテクテッドモード有効、ページング有効などのビットを設定しています。

⑪CR3レジスタに作成したページテーブルのアドレスを設定しています。

⑫CR4レジスタにPAE有効化などのビットを設定しています。

⑬EFERレジスタに64bit有効化などのビットを設定しています。

⑭RSPレジスタにスタックアドレスの初期値を設定しています。

⑮RIPレジスタにロードしたプログラムのアドレスを設定しています。

ロードする簡易ローダを実装しました[注4]。

このローダを使うと、ゲストマシンが起動した最初の瞬間から、64bitモード・ページングが有効な状態で実行されます。**リスト4**にソースコードを示します。

実行イメージ

ビルドしたローダは次のようなコマンドで実行できます(**リスト5**)。

まとめ

以上、物理マシンのような制約のない仮想マシンの解説をしました。制約の少ない仮想マシンでは、仮想マシンの状態を設定して仮想マシンを動作させることができます。BHyVeが採用しているダイレクトブートは、この特徴を活かしてBIOSを用いずにゲストOSの実行を実現しています。SD

注4) https://github.com/syuu1228/bhyve-embedded-guest

▼リスト5　画面出力

```
sudo ./load
sudo bhyve  - c 1  - m 128  - s 0:0,hostbridge  - S 31,uart,stdio test1
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
```

第23回

Linux 3.13の新機能
PowerCapとSquashfsのマルチキュー対応

Text：青田 直大　AOTA Naohiro

今月もLinux 3.13に追加された機能について解説していきます。今月は電力消費を制御するPowerCapという機能、そしてSquashfsのマルチキュー対応について見ていきます。

PowerCapサブシステム

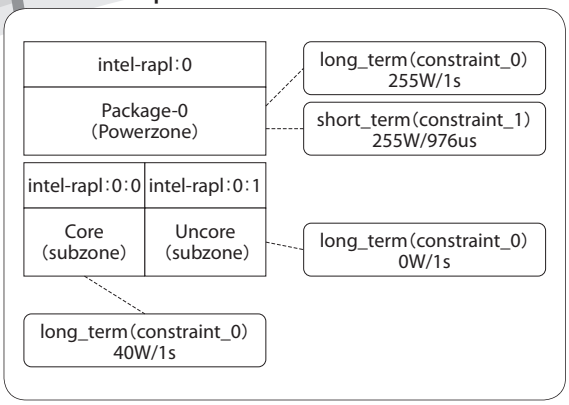
Linux 3.13では新しくPower Capというサブシステムが実装されました。これは電力制限が可能なデバイスをユーザ空間から操作するためのインターフェースになります。もともとIntelのRunning Average Power Limit（RAPL）専用のドライバが提案されていたものが、汎用的なサブシステムに書き換えられたのもあってIntel RAPLのみが現在はサポートされていますが、やがてはほかのデバイスのドライバも実装されていくかと思います。

では、PowerCapの使い方を見てみましょう。PowerCapのインターフェースはsyfsの/sys/class/powercapからアクセスできます。**図1**のように、ドライバのトップレベルおよび電力制限をかけることができる各部分へのアクセスを提供するディレクトリへのsymlinkが作られています。symlink先を見るとわかりますが、**図2**のようにintel-raplの下に“intel-rapl:0”が、“intel-rapl:0”の下に“intel-rapl:0:0”と“intel-rapl:0:1”が配置されています。後述するように、CPUの構造がこのディレクトリ構造に反映されています。

“intel-rapl:0”は“power zone”と呼ばれ、その下の“intel-rapl:0:0”、“intel-rapl:0:1”は“subzone”となっています。“inte-rapl:0”、“intel-rapl:0:0”、“intel-rapl:0:1”の下にはそれぞれ“name”というファイルがあるので、これを読むとそれぞれ“package-0”、“core”、“uncore”という名前になっていることがわかります（**図3上部**）。“package-0”が1つのプロセッサパッケージ内の電力について、“core”がその中のCPUコアの電力について、そして“uncore”がグラフィックスなどコア以外の部分の電力についてのディレクトリとなります。

nameと同様にenergy_ujというファイルも各区分のディレクトリ下にあります。これを読むことで累積消費電力量をマイクロジュール単位で見ることができます（**図3下部**）。この値は累積ですので段々と増えていきmax_energy_range_ujまでいくと0から折り返します。この値を読むだけでも興味深いデータを得ることができます。たとえば、muninのプラグイン（**リスト1**）を書いてプロットすると**図4**のような結果を得ることができます。

▼図2　intel-raplの構造



▼図3　名前と電力量の表示

```
# grep . */name
intel-rapl:0/name:package-0
intel-rapl:0:0/name:core
intel-rapl:0:1/name:uncore
# grep . */energy_uj */max_energy_range_uj
intel-rapl:0/energy_uj:34046807281
intel-rapl:0:0/energy_uj:31149954772
intel-rapl:0:1/energy_uj:30208578598
intel-rapl:0/max_energy_range_uj:65535999984
intel-rapl:0:0/max_energy_range_uj:65535999984
intel-rapl:0:1/max_energy_range_uj:65535999984
```

電力の制限

次に電力の制限について見てみましょう(**図5**)。電力制限に関する値は“constraint_”から始まるファイルに記載されています。packageにはconstraint_0とconstraint_1という2つの制限が、coreとuncoreにはconstraint_0の1つだけの制限をかけることができます。time_window_usで指定したマイクロ秒数の間の平均消費電力をpower_limit_uwにするようにCPUに設定します。ここで設定できる値はmax_power_uw以下になります。例の部分で“intel-rapl:0:0”と“intel-rapl:0:1”の constraint_0_max_power_uwが「利用可能なデータがありません」となっているように、CPUにとっては取得できない値もあります。これらの値を設定したときは、enabledを1にして設定が適用されるようにしてください。

▼図1　PowerCapのディレクトリ構造

```
# cd /sys/class/powercap/
# ls -l
合計 0
lrwxrwxrwx 1 root root 0 12月 10 19:40 intel-rapl -> ../../devices/virtual/powercap/intel-rapl
lrwxrwxrwx 1 root root 0 12月 10 19:40 intel-rapl:0 -> ../../devices/virtual/powercap/intel-rapl/intel-rapl:0
lrwxrwxrwx 1 root root 0 12月 10 19:40 intel-rapl:0:0 -> ../../devices/virtual/powercap/intel-rapl/intel-rapl:0/intel-rapl:0:0
lrwxrwxrwx 1 root root 0 12月 10 19:40 intel-rapl:0:1 -> ../../devices/virtual/powercap/intel-rapl/intel-rapl:0/intel-rapl:0:1
# ls -l intel-rapl/
合計 0
-rw-r--r-- 1 root root 4096 12月 10 19:42 enabled
drwxr-xr-x 5 root root    0 12月 10 16:54 intel-rapl:0
drwxr-xr-x 2 root root    0 12月 10 19:42 power
lrwxrwxrwx 1 root root    0 12月 10 16:54 subsystem -> ../../../../class/powercap
-rw-r--r-- 1 root root 4096 12月 10 16:54 uevent
# ls -l intel-rapl:0/
合計 0
-r--r--r-- 1 root root 4096 12月 10 19:40 constraint_0_max_power_uw
-r--r--r-- 1 root root 4096 12月 10 19:40 constraint_0_name
-rw-r--r-- 1 root root 4096 12月 12 20:19 constraint_0_power_limit_uw
-rw-r--r-- 1 root root 4096 12月 10 19:40 constraint_0_time_window_us
-r--r--r-- 1 root root 4096 12月 10 19:40 constraint_1_max_power_uw
-r--r--r-- 1 root root 4096 12月 10 19:40 constraint_1_name
-rw-r--r-- 1 root root 4096 12月 10 19:40 constraint_1_power_limit_uw
-rw-r--r-- 1 root root 4096 12月 12 20:26 constraint_1_time_window_us
lrwxrwxrwx 1 root root    0 12月 10 19:40 device -> ../../intel-rapl
-rw-r--r-- 1 root root 4096 12月 10 19:40 enabled
-rw-r--r-- 1 root root 4096 12月 12 14:45 energy_uj
drwxr-xr-x 3 root root    0 12月 10 16:54 intel-rapl:0:0
drwxr-xr-x 3 root root    0 12月 10 16:54 intel-rapl:0:1
-r--r--r-- 1 root root 4096 12月 10 19:40 max_energy_range_uj
-r--r--r-- 1 root root 4096 12月 10 19:40 name
drwxr-xr-x 2 root root    0 12月 10 19:40 power
lrwxrwxrwx 1 root root    0 12月 10 16:54 subsystem -> ../../../../../class/powercap
-rw-r--r-- 1 root root 4096 12月 10 16:54 uevent
```

RAPLの実装

そもそもIntel RAPLとはSandy Bridge以降のCPUに実装されている消費電力を監視し、消費電力を制限するための機能です。実際にモデル固有レジスタ(MSR)への読み書きをしてみて、RAPLの中身について見ていきましょう。

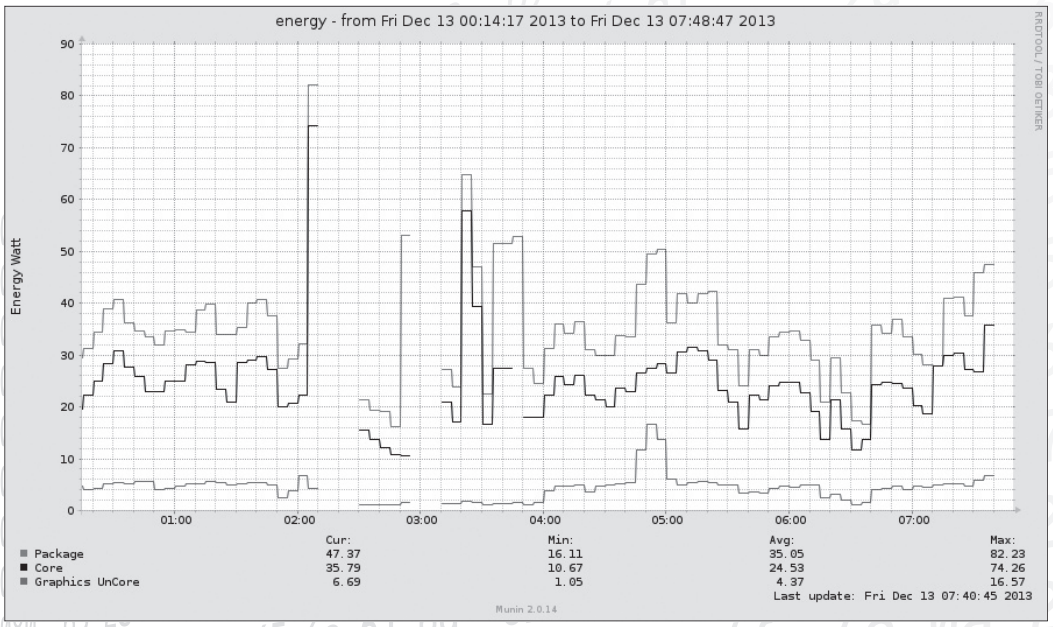
まずはRAPLで使われる電力・電力量・時間の単位を決定します。単位についての情報はMSR_RAPL_POWER_UNIT(0x606)で管理されています(**図6**)。3から0bitは電力単位に、12から8bitが電力量単位に、19から16bitが時間単位の設定に使われます。それぞれこの値をXとすると"$1/2^X$"が単位量となります。すなわち下の例(CPUのデフォルト値のまま)では、1/8ワット、15.26マイクロジュール、976.56マイクロ秒が単位となります。後述のMSRから読み取られる値にこの単位量をかけることで求めたい値がわかります。

次にenergy_ujに対応する値を読んでみましょう。"intel-rapl:0"のenergy_ujに対応する値はMSR_PKG_ENERGY_STATUS(0x611)を読

▼リスト1　munin plugin

```
#!/bin/bash

case $1 in
  config)
    cat <<'EOM'
graph_category system
graph_title energy
graph_vlabel Energy Watt
graph_args --base 1000 -l 0
package.label Package
package.type DERIVE
package.min 0
core.label Core
core.type DERIVE
core.min 0
uncore.label Graphics UnCore
uncore.type DERIVE
uncore.min 0
EOM
    exit 0
    ;;
esac

cd /sys/class/powercap
printf "package.value "
cat "intel-rapl:0"/energy_uj | perl -ne
'print int($_/1000000),"¥n"'
printf "core.value "
cat "intel-rapl:0:0"/energy_uj | perl -ne
'print int($_/1000000),"¥n"'
printf "uncore.value "
cat "intel-rapl:0:1"/energy_uj | perl -ne
'print int($_/1000000),"¥n"'
```

▼図4　muninによるプロット



むことで取得できます(**図7**)。MSRからの出力をマイクロ単位に変換(10^6を掛ける)し、上で取得したとおりに2^{16}で割ることでenergy_ujとMSRの値が確かに一致していると確認できます。このMSRはおよそ1ミリ秒に一度更新されるのでぴったり同じにはならないこともあります。

最後にconstraintに関係する値を読みとってみましょう(**図8**)。MSR_PP0_POWER_LIMIT(0x638)の14から0bitが"intel-rapl:0:0"のconstraint_0_power_limit_uwに対応し、23から17bitがconstraint_0_time_window_usに対応しています。たとえば14から0bitから得られる値が320で、1/8ワット単位なので40,000,000マイクロワットとなっています。時間のほうは読み方が少し特殊になっています。21から17bitの値をX、23から22bitの値をYとおくと"$2^X \times (1 + Y/4) \times$ <単位時間>"という読み方をします。

▼図5　電力制限に関する値の表示

```
# grep . */constraint_*
intel-rapl:0/constraint_0_max_power_uw:95000000
intel-rapl:0/constraint_0_name:long_term
intel-rapl:0/constraint_0_power_limit_uw:255000000
intel-rapl:0/constraint_0_time_window_us:1000000
intel-rapl:0/constraint_1_max_power_uw:0
intel-rapl:0/constraint_1_name:short_term
intel-rapl:0/constraint_1_power_limit_uw:255000000
intel-rapl:0/constraint_1_time_window_us:976
grep: intel-rapl:0:0/constraint_0_max_power_uw: 利用可能なデータがありません
intel-rapl:0:0/constraint_0_name:long_term
intel-rapl:0:0/constraint_0_power_limit_uw:40000000
intel-rapl:0:0/constraint_0_time_window_us:1000000
grep: intel-rapl:0:1/constraint_0_max_power_uw: 利用可能なデータがありません
intel-rapl:0:1/constraint_0_name:long_term
intel-rapl:0:1/constraint_0_power_limit_uw:0
intel-rapl:0:1/constraint_0_time_window_us:1000000
# grep . */enabled
intel-rapl/enabled:1
intel-rapl:0/enabled:1
intel-rapl:0:0/enabled:1
intel-rapl:0:1/enabled:1
```

▼図6　RAPLでの使用単位の読み取り

```
# rdmsr -u --bitfield 3:0 0x606
3
# rdmsr -u --bitfield 12:8 0x606
16
# rdmsr -u --bitfield 19:16 0x606
10
```

▼図7　消費電力量の読み取り

```
# cat "intel-rapl:0"/energy_uj & rdmsr -u 0x611
[1] 2174
2364038996
36072372375
[1]+  終了                  cat energy_u
# python
>>> 2364038996 * (10 ** 6) / (2 ** 16)
36072372375
```

PowerClampとの比較

Intel RAPLはCPUの消費電力を制限するという点でLinux 3.9に入った機能であり、以前に紹介したIntel PowerClampとも似ています。この2つはどのように違っているのでしょうか。1つの違いは対応CPU世代です。Intel RAPLはSandyBridge以後のCPUでしか動きませんが、Intel PowerClampはNehalemでも動作します。もう1つの違いはRAPLではCPUのモデル固有レジスタ(MSR)に目標とする電力値を設定し、CPUに任せるというある意味でシンプルなカーネル実装になっているのに対して、PowerClampでは各CPUに優先度が最高のカーネルスレッドを割り付けて、そのスレッドによってCPUを強制的に一番消費電力が低くなる動作状態に移行するというOSの実装側でがんばっているしくみになっています。また、Intel RAPLは最終的にPowerCapサブシステムの一部とされたようにほかのCPU・アーキテクチャを想定した作りをしていますが、Intel PowerClampにはそのような様子はありません。設定項目も含めてまとめると**表1**のようになります。

Squashfsのマルチキュー対応

Squashfsは、1つのファイルに圧縮されたファイルツリーを読み込み専用でマウントできるファイルシステムです。LiveCDなどのディスク容量が限られているようなデバイスで、読み込み専用のファイルシステムを提供するのによく使

われています。Linux 3.13では、このSquashfsにパフォーマンスを改善する変更が2つ追加されました。

Squashfsでは圧縮されているファイルを適宜展開して、ファイルシステム中のファイルへのアクセスを提供しています。今まではこの展開はシングルスレッドで行われてきました。今ではマルチコアのCPUが増え、VMの中でも複数のCPUが与えられていることもある中で、この実装はパフォーマンスの大きな足かせとなります。そこでLinux 3.13ではSquashfsが複数のI/Oを行っているときに、展開も複数のスレッドで行えるような変更が入りました。具体的には次の2つの展開方法が使えるようになりました。

- 各CPUにつき最大2個までの展開スレッドを作る
- 各CPUにつき1つずつロードバランスされるように展開スレッドを作る

もう1つのパフォーマンス改善は、展開のための中間バッファを排除したことで実現されました。今までは中間バッファに展開し、それからその内容をページキャッシュにコピーしていたものを、直接ページキャッシュへと展開できるようにしました。これはメモリコピーの手間を減らすだけではなく、複数のI/Oが行われているときの中間バッファのロック競合も取り除きパフォーマンスを劇的に改善しています。

どちらの変更にも、コミットログにパッチ作者のパフォーマンステストの結果が報告されていますので、紹介しておきましょう。1つめの複数スレッドでの展開では、4つの1GBのファイルを2コア・メモリ4GBのKVM上でそれぞれ1つのddプロセスで読み込みます。以前には1分39秒かかっていたものが、複数スレッドにすることで9秒になると驚くほどの改善が報告されています。もう一方の中間バッファを取り除く変更では、ファイルを同時に4つ読み出すのに以前は13.7MB/sだったものが、変更後には67.7MB/sと5倍ほどに向上しています。どちらのパッチも大きなパフォーマンス改善が実現できておりたいへん期待できる変更ですね。

まとめ

今回は電力監視・制限を行うIntel RAPL、そしてSquashfsのパフォーマンス改善について解説しました。電力消費が問題となることも多くなっているのではないでしょうか。ぜひまずは監視からでも始めてみてください。SD

▼図8　constraint関係の値の読み取り

```
# grep . "intel-rapl:0:0"/constraint_0_{power_limit_uw,time_window_us}
intel-rapl:0:0/constraint_0_power_limit_uw:40000000
intel-rapl:0:0/constraint_0_time_window_us:1750000
# rdmsr -u --bitfield 14:0 0x638
320
# rdmsr -u --bitfield 21:17 0x638
10
# rdmsr -u --bitfield 23:22 0x638
3
# python
>>> (2 ** 10) * (1 + 0.25 * 3) * (10 ** 6) / (2 ** 10)
1750000.0
```

▼表1　PowerClampとRAPLの違い

	対応世代	実装方法	設定項目
PowerClamp	Nehalem〜	カーネルスレッドで低電力の状態にする	アイドルのCPU時間の割合
RAPL	Sandy Bridgeから	CPUのMSRに制限を設定	時間あたりの平均消費電力

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第4回 ボトルネックはHDDか？　交換の前にシステム情報から推測しよう

◉著者プロフィール

後藤 大地（ごとう だいち）

BSDコンサルティング（株）　取締役／（有）オングス　代表取締役／FreeBSD committer

エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

ディスクの性能を使い切れているか調べる方法、知ってますか？

コンシューマ向けに出荷されているSSDはだいぶ値段がこなれてきました。TB（テラバイト）クラスの商品も登場しはじめています。SSDとHDDを組み合わせたハイブリッドなディスク（SSHD）も普及をはじめています。だいたい容量単価あたりの価格が高いほうが高速に動作するようです。

HDDをSSDに換装するとシステムが高速に動作するようだ、ということは実感としては理解できるのですが、実はどの程度速くなったのか、そもそもディスク部分が原因で動作速度が遅いと感じているのか、そのあたりを調べる方法を知らないまま直感で判断していることがあります。

ディスクの性能を調べる方法がわかれば、システムがどういった状態にあるのか、どういったディスクやストレージへ交換すればシステムの性能を上げることができるのか、こういったことがわかるようになります。今回はそうした場合に利用するコマンドを紹介します。

ディスクの限界性能を調べる（UFSの場合）

まず一番最初にするのは、そのディスクとファイルシステムのもっとも高い性能を知ることです。ここでは/dev/ada1として認識されているディスクを調べてみます。**図1**のように**gpart**(8)でパーティショニング情報をチェックすると、/dev/ada1 p1がUFSファイルシステムになっていることがわかります。

このファイルシステムがどのようなオプション指定でフォーマットされたものかは**dumpfs**(8)コマンドに**-m**オプションを指定すると調べることができます。たとえば**図2**で調べたパーティションはUFS2でフォーマットされており（-O 2）、Soft updates（-U）とSoft updates journaling（-j）が有効になったファイルシステムだということがわかります。

ファイルシステムの使用状況などの詳細情報もdumpfs(8)コマンドで確認できます。dumpfs(8)の出力データの読み方については、連載の間においおい解説するとしましょう。**図3**の出力を見る限りでは、まだまだ余裕がある状態だとわかります。

▼図1　gpart(8)でディスクのパーティショニング情報を確認

```
% gpart show ada1
=>        34  488397101  ada1  GPT  (233G)
          34  482344960     1  freebsd-ufs  (230G)
   482344994    6052141        - free -  (2.9G)

%
```

▼図2　newfs(8)でフォーマットしたときのオプション

```
% dumpfs -m /s
# newfs command for /s (/dev/gpt/s)
newfs -O 2 -U -a 4 -b 32768 -d 32768 -e 4096 -f 4096 -g 16384 -h 64 -i 8192 -j -k 6408 -m 8 -o time -s
482344960 /dev/gpt/s
%
```

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

このディスクへの書き込みを実施して限界性能を調べます。**図4**のように**dd**(1)コマンドでゼロデータを10GB分書き込みます。この作業を実行している間に、別の疑似端末で**図5**のように**iostat**(8)コマンドを実行します。

iostat(8)コマンドによる1つめの出力はシステム稼働時間全体に渡る入出力の統計情報です。ここでは2つめ以降の値、とくにada1の「tps」と、cpuの「id」に注目してください。このディスクでは1秒間に2,000回のトランザクションを実行するというのが書き込み性能の上限値のようだということが、tpsの値から推測できます。その間、idの値は98前後ですので、ほとんど「アイドル状態」にあることがわかります。プロセッサをほとんど使っていませんので、ディスクの書き込み処理性能をほぼほぼ計測できていることになります。

▼図3　ファイルシステムの詳細情報

```
% dumpfs /s | head -18
magic   19540119 (UFS2) time    Fri Dec  6 18:33:03 2013
superblock location     65536   id      [ 527eca07 3775b6b3 ]
ncg     377     size    60293120        blocks  58396030
bsize   32768   shift   15      mask    0xffff8000
fsize   4096    shift   12      mask    0xfffff000
frag    8       shift   3       fsbtodb 3
minfree 8%      optim   time    symlinklen 120
maxbsize 32768  maxbpg  4096    maxcontig 4     contigsumsize 4
nbfree  6348236 ndir    91419   nifree  29210281        nffree  244335
bpg     20035   fpg     160280  ipg     80256   unrefs  0
nindir  4096    inopb   128     maxfilesize     2252349704110079
sbsize  4096    cgsize  32768   csaddr  5056    cssize  8192
sblkno  24      cblkno  32      iblkno  40      dblkno  5056
cgrotor 99      fmod    0       ronly   0       clean   0
metaspace 6408  avgfpdir 64     avgfilesize 16384
flags   soft-updates+journal
fsmnt   /s
volname         swuid   0       providersize    60293120
%
```

▼図4　/s/の下でゼロデータで埋められたファイルを作成

```
% dd if=/dev/zero of=out bs=1024x1024x100 count=100
100+0 records in
100+0 records out
10485760000 bytes transferred in 40.857527 secs (256642064 bytes/sec)
%
```

▼図5　別の疑似端末でiostat(8)コマンドを実行して入出力状況を確認

```
% iostat -d -C -w 10 ada1
             ada1                cpu
   KB/t  tps   MB/s  us ni sy in  id
  50.05    2   0.09   0  0  0  0 100
 127.53 1795 223.57   0  0  2  0  98
 127.18 1956 242.91   0  0  3  0  97
 127.63 1970 245.52   0  0  2  0  98
 127.49 1973 245.60   0  0  3  0  97
^C
%
```

ディスクの限界性能を調べる（ZFSの場合）

入出力の限界性能はファイルシステムによって大きく変わります。先ほどと同じモニタリングをZFSでやってみましょう。**図6**によると、ここで使用したZFSはディスクを2台ミラーリングしたZFSプール上のデータセットで実行されていることがわかります。

図7のようにdd(1)で書き込みを実施します。**図4**では40秒ほどで終わっていましたが、こちらは110秒ほどか

▼図6　/z1/は2台のディスクをミラーリングで構成したZFSプール

```
% zpool status z1
  pool: z1
 state: ONLINE
  scan: resilvered 479G in 3h1m with 0 errors on Thu Dec 13 15:33:30 2012
config:

        NAME        STATE     READ WRITE CKSUM
        z1          ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada2    ONLINE       0     0     0
            ada3    ONLINE       0     0     0

errors: No known data errors
%
```

▼図7　/z1/の下でゼロデータで埋められたファイルを作成

```
% dd if=/dev/zero of=out bs=1024x1024x100 count=100
100+0 records in
100+0 records out
10485760000 bytes transferred in 108.842429 secs (96338901 bytes/sec)
%
```

かっています。だいぶ遅いことがわかります。dd(1)で書き込み実行中に、iostat(8)コマンドを使ってモニタリングを実施します(**図8**)。こちらもほとんどプロセッサを使っておらず、出力の限界性能を計測していることになります。

ZFSのほうでは1秒間の最大トランザクション回数が800回くらいということがわかります。たとえばZFSのデータセットを使っているシステムを構築しているのであれば、システム動作時の単位時間当たりのトランザクション回数をどれだけこの値に近づくようにできるかが、そのシステムで書き込み性能を使い切れているかということになります。

ZFSはUFSと比較するとかなり複雑な作りをしています。ここまで機能を詰め込んでいるにもかかわらずそれほど性能が劣化しないのは、かなり大量のキャッシュを活用しているからです。ZFSを使っているシステムで`top`(1)コマンドを実行すると、**図9**のように固定化されたキャッシュ(Mem:のWiredの項目)が13GBあることがわかります。Wiredにはスワップの対象とならないメモリの領域が表示されます。たとえばカーネルそのものがここに該当します。カーネルがスワップアウトしてしまってはシステムが動作しません。

▼図8　別の疑似端末でiostat(8)コマンドを実行して入出力状況を確認

```
% iostat -d -C -w 10 ada2
             ada2              cpu
   KB/t tps  MB/s  us ni sy in  id
  10.39   1  0.01   0  0  0  0 100
 119.40 781 91.08   0  0  1  0  99
 119.36 675 78.65   0  0  1  0  98
 120.70 790 93.14   0  0  1  0  98
 121.63 797 94.67   0  0  1  0  99
 119.16 775 90.19   0  0  2  0  98
 118.77 658 76.26   0  0  1  0  98
 120.46 681 80.13   0  0  1  0  98
 123.70 703 84.96   0  0  1  0  99
 121.86 691 82.24   0  0  1  0  99
 122.78 785 94.15   0  0  1  0  99
 124.04 808 97.88   0  0  2  0  98
^C
%
```

しかしカーネルはいくら大きくても13GBもありません。これはほとんどZFSのキャッシュに使われています。FreeBSDのZFSではZFSキャッシュがスワップアウトしない領域として扱われます。ZFSのキャッシュを表現するARC:の項目が11Gになっていることからもわかります。

プロセッサを使い切ってディスク性能がまったく使い切れていないケース

では先のUFS環境で、模擬的に入出力性能を使い切れていない状況を作り出してモニタリングしてみましょう。まず、システムのプロセッサの論理コア(スレッド)の数を調べます。**図10**のように`sysctl`(8)コマンドでhw.ncpuの値を調べます。このシステムの論理コア(スレッド)の数は8個であることがわかります。

データの圧縮や復号処理を、指定した数で並列して実施するコマンドに`pigz`(/usr/ports/archivers/pigz/)があります。これを**図11**のように実行して10GBのファイルの圧縮を試みます。このpigz(1)の実行中にiostat(8)を実行すると、**図12**のような出力が得られます。プロセッサをほぼ使い切っていて、かわりにディスクの性能はほとんど発揮できていません。最大で2,000トランザクション/秒くらいのところが、50トランザクション/秒くらいになっています。

こういった使い方をしているシステムでは、ディスクを高速なものに換えても処理性能の向上は望め

▼図10　プロセッサの論理コア(スレッド)の個数を確認

```
% sysctl hw.ncpu
hw.ncpu: 8
%
```

▼図11　16並列でファイルの圧縮処理を実行

```
% pigz -p 16 -11 out
```

▼図9　ZFSは高速に動作するために大量の領域をスワップ不可能なキャッシュとして使用する

```
last pid: 43376;  load averages:  0.14,  0.20,  0.14 up 1+01:46:03  18:39:04
58 processes:  1 running, 57 sleeping
CPU:  1.5% user,  0.0% nice,  1.1% system,  0.0% interrupt, 97.4% idle
Mem: 21M Active, 941M Inact, 13G Wired, 3288M Buf, 2127M Free
ARC: 11G Total, 723M MFU, 10G MRU, 213K Anon, 39M Header, 399M Other
Swap: 2047M Total, 2047M Free
```

▼図12　CPUを使い切り、逆にディスクの入出力性能は使い切れていない

```
% iostat -d -C -w 10 ada1
             ada1                cpu
   KB/t tps  MB/s  us ni sy in  id
  89.42   4  0.35   0  0  0  0 100
 125.75  51  6.25  99  0  1  0   0
 125.35  51  6.28  99  0  1  0   0
 124.50  52  6.27  99  0  1  0   0
 125.32  51  6.26  99  0  1  0   0
 125.50  51  6.24  99  0  1  0   0
 125.18  51  6.29  99  0  1  0   0
 127.31  51  6.31  99  0  1  0   0
 127.49  50  6.25  99  0  1  0   0
^C
%
```

ません。この場合、もっとコアの多いプロセッサに換えるとか、もっと処理速度が速いプロセッサに交換することが、システムの性能を引き上げることにつながります。

ディスクもプロセッサも性能を使い切れていないケース

ちょっとだけ先ほどのコマンドのオプションを変更してみましょう。-pオプションで指定する値を16から4へ変更してみます（図13）。

今度は図14のような結果が得られます。プロセッサが50％くらいアイドルした状態で、かつ、ディスクもほとんど性能を発揮できていない状態であることがわかります。この場合には処理の並列度をあげることで全体としての処理性能を向上させることができる可能性があります。

このような感じで動作しているシステムの状態を入出力（単位時間当たりのトランザクション回数）とプロセッサの使用率という側面で捕らえると、どこが性能のボトルネックになっているかがわかるようになります。

おまけ：読み込みキャッシュも意識してみよう

ディスク入出力系のベンチマークを実施する場合にはキャッシュ効果に注意してください。たとえば今回の記事ですと、モニタリングを実施する前にデータの読み込みにおいてキャッシュが効かないように、いったんキャッシュをすべて削除しています。

たとえば図15のようにdd(1)コマンドを実行すれば読み込みキャッシュをほぼ無効にできます。top(1)のBufに表示された値が3.2GBほどでしたので、図15のようにランダムデータを4GBほど書き出してあげると、キャッシュされていた分がすべて消えることになります。もちろんここで消しているのはUFS2のキャッシュです。

キャッシュ効果はよく効きます。メモリをたくさん積んだマシンでは、実のところSSDとHDDの違いを体感できないことがあります。これはキャッシュにデータが載って、SSDでもHDDでもメモリからデータが読み込まれている場合などに起こります。

おわりに

今回はiostat(8)の読み方を中心に単位時間当たりのトランザクション回数を読むということ、プロセッサの使用率との関係からシステムの性能ボトルネックを判断する方法などを紹介しました。今までよくわからなかったコマンドの出力の意味がわかるようになってくると、システムの動きが目に浮かぶようになりますよね（ならない？）SD

▼図13　4並列でファイルの圧縮処理を実行

```
% pigz -p 4 -11 out
```

▼図14　CPUも使い切れていないし、ディスクの入出力性能も使い切れていない

```
% iostat -d -C -w 10 ada1
             ada1                cpu
   KB/t tps  MB/s  us ni sy in id
  93.00   4  0.40   0  0  0  0 99
 108.70  40  4.26  50  0  0  0 50
  96.23  16  1.54  50  0  1  0 50
  93.28   7  0.63  50  0  0  0 50
 101.29  11  1.09  50  0  1  0 50
 102.38   9  0.91  50  0  0  0 50
 100.26  16  1.57  50  0  1  0 50
 107.93  22  2.36  50  0  0  0 50
 115.47  36  4.03  50  0  1  0 50
 123.35  38  4.64  50  0  0  0 50
^C
%
```

▼図15　dd(1)で読み込みキャッシュを無効化

```
% dd if=/dev/random of=hoge bs=1024x1024x100 count=40
```

12

Debian Developer　やまねひでき　henrich@debian.org

前途多難なInitシステム／64bit ARM対応、動き出したRCバグ対応

Debian Hot Topics

はじめに

読者のみなさん、2014年が始まりましたが、年末年始の長期休暇からの復帰に遅滞はありませんか？　筆者はこの原稿を書いているときは年末進行まっただ中で、例のごとく遅延して編集の方をお待たせしています……毎度すみません。2014年はこのへんが改められればいいのですが。

Initシステムと技術委員会の動向

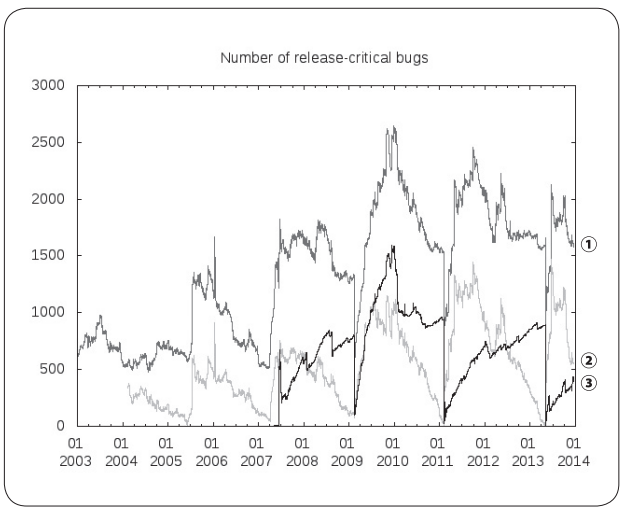
2014年1月号の本連載でお伝えしたDebianでの標準Initシステムを巡る騒ぎですが、いまだに結論は見えないままです。ホリデーシーズンへ突入したためか開発メーリングリスト(debian-devel)での議論は停滞ぎみになっており、技術委員会(ctte)からもとくに意見表明が出ていません。cttеメーリングリストでは議論が続いており、2013年12月時点ではsystemd/upstartのセキュリティ問題などへ議論は発散しています。興味のある方はBug#727708を引き続きウォッチしてください。

そして技術委員会についてですが、Keith Packardさんが空席を埋める新たな委員として任命されました[注1]。Keithさんは Intel社に所属してX.orgのupstreamとして活動をしているのでDebian以外のところで名前を見かけている人も多いかもしれません[注2]。

「不具合パッケージ自動削除」の効果測定

また、2013年12月号の本連載で取り上げた「不具合パッケージの自動削除(auto-rm)」ですが、大きな効果が出ているようです[注3]。バグの数の減り方のグラフ(**図1-②**)を見てみると明

▼図1　RCバグの状況[注4]



縦軸：件数　横軸：月・年
凡例：①現在(unstable)のRCバグ数　②次のリリース(testing)に対するRCバグ数　③安定版(stable)のRCバグ数

注1）URL https://lists.debian.org/debian-devel-announce/2013/11/msg00009.html
注2）個人的にはBdale Garbeeさんと一緒にロケット打ち上げて楽しんでいる人、という印象が強いです。
注3）パッケージメンテナとしては最終的にはバグ修正する必要があるので、直すのを先回しにしただけとも言えるのですが……。
注4）URL http://bugs.debian.org/release-critical/

らかに減少していることがわかります(stableでのRCバグ(Release Critical bug)に近い数になってきています)。testingに大きな機能の欠損がなければDebian 8“Jessie”のリリース作業は以前よりスムースなものとなるでしょう。

64bit ARMサポートへの道程

2013年11月14～17日にMiniDebConf in UKがケンブリッジのARM社で開催されました。100名程度の参加者を集めてさまざまな発表が行われました。開催地からもわかるようにおもにARM関連の作業が「ARM sprint[注5]」として実施され、その結果をまとめた「Bits from ARM porters」というメール[注6]が流れています。内容をかいつまむと、

- 性能のいいARMサーバが手に入ったらそこで集中してビルドしようと考えていた。しかし、ARMv5tのマシンでARMv4tのコードをビルドした場合などに潜在的な問題が出てくる可能性があるので、これは止める
- armelはしばらくビルドを続ける[注7]
- armhfマシンはいくつかのハードウェアベンダから寄付のオファーをもらっている
- Jessieではarmhfでarmmpとlpaeのサポートを追加する
- GRUBでのARMサポートを実験的に追加
- Jessieのフリーズ前に64bit ARMマシンが手に入らなければ、Jessieでのサポートは見送る

というものでした。

そして、64bit ARMのサポートの話題については、

「入手できるarm64[注8]マシンそのものが市場にない(唯一、iPhone 5sがあるがこれはビルドマシンという目的に合わないので対象外)。また、Debianは寄付で成り立っており、ARM(32bit)サーバに7,000ドルもかけるという選択肢はない。Canonical社がアプライド・マイクロ・サーキット社とX-Geneサーバ[注9]をUbuntu arm64版のために提供してくれるよう交渉しているようだが、我々はあと半年は使えそうになく、できたとしてもDebian 8“Jessie”のフリーズ直前になる。とりあえず、クロスビルドかqemuでのビルドを正式版になる前の足がかりとして作成するのが良いのではないか」

「リリースに間に合わないのであれば、以前に行った『etch-m68k』のようにunstableのスナップショットを『jessie-arm64』という形で提供すれば良いのでは？[注10]」

などの意見が出されています。

Debianプロジェクト内にはARM社に雇用されている人や、Linaro[注11]で働いている人が何人かいるなど、それなりにつながりはあるようです。しかし、ARM社はプロセッサを直接生産しておらずCPUのライセンスを各社に提供するだけです。ライセンス供給を受けた会社が各チップを設計／生産するというIntel/AMD社のx86とは違うスキームのため、arm64のマシンを入手できそうなコネがないのが辛いところです。

また、通常アーキテクチャを追加する際には

注5) sprintとは短期間にある事柄について集中して作業するイベント。

注6) URL https://lists.debian.org/debian-devel-announce/2013/12/msg00001.html

注7) armel自体はDebianの公式サポートアーキテクチャからは外れています。

注8) Debianで利用する正式なアーキテクチャ名はAARCH64ですが、arm64のほうが通りが良いので皆そちらを使っているようです。

注9) URL http://www.apm.com/products/data-center/x-gene-family/x-gene/ 「世界初の64bit ARMサーバ」だそうです。

注10) 以前にDebian 3.1“Sarge”の際、公式リリースにはamd64は含まれず、あとから「sarge-amd64」としてビルドが提供されたことがあったので選択肢としてはありえそうです。

注11) ARM用のLinuxカーネル開発などを推進している非営利組織。URL http://www.linaro.org/jp ARM社やCanonical社から出向して作業している人も結構います。

前段階としてdebian-ports[注12]で作業が行われるのですが、これについてのやりとりも

「debian-portsへarm64移植版を追加するように依頼を受けて準備はしておいたよ。ずっとbuildd[注13]のセットアップに必要な情報を待っているんだけど」
「その返信は見てなかった……まだ実機がないので作業できない。linaroのパートナーが持っているarm64マシンに一時的にアクセスできるようにはなったけど、NDAのせいで私は触れない(けれどアクセスできる別のDebian開発者がいるよ)」

というような感じで「準備は進めているものの利用できるハードウェア待ち」といった様子です。

なお、debian-portsは、Debian本体で利用しているインフラではなく独自のインフラを持ってパッケージの管理を実施しています。これをDebian本体と統合する話も出ていますが、

「debian-portsをDSA[注14]管理にするのはいいけど、機能は落とさないでね。それから、移行するにしてもフリーズ期間中であっても更新パッケージをかまわずにアップロードできるようにしてくれないと移植作業が滞る。この機能(unreleased suite)をdak[注15]に追加をしてほしいと言ったのだけど、誰も作業してくれていない(からまだ無理なんじゃないの?)」

とこちらも検討はしているものの滞っている様子。

debian-portsをDebian本体のインフラと統合することで、将来的なインフラ拡張への柔軟性を高めることと、debian-ports側の作業を減らして新規アーキテクチャのサポート追加/公式サポートから外れたアーキテクチャをdebian-portsへ移管する作業がスムースに進むことが期待されますが、現在運用している公式インフラの統合を行うのにはある程度の時間集中して関係者の人的リソースを提供する必要があります。フルタイムで雇用されている人員がいないDebianではこの辺のハードルが高そうです。

MiniDebConf in Paris/FOSS ASIA 2014

台北、ケンブリッジと続けて開催されているMiniDebConfですが、ちょうどこの号が出るころの2014年1月18〜19日にパリでも開催すること[注16]をSylvestre Ledruさん[注17]が発表しています。詳細は(本稿を書いている時点では)まだこれからのようですが、すでに20名ほどの登録者がいるので、今回も盛況となりそうです。

また、Debianも参加するイベントとして「FOSS ASIA 2014」[注18]の情報が入ってきました。2014年2月28日〜3月2日にカンボジアのプノンペンで行われ、さまざまなFLOSSコミュニティが参加します。興味のある方はTwitterで@fossasiaをフォローしてみてください。

AWSからの寄付とアーカイブリビルド

Debianではパッケージがビルドできないバグを重大な問題として位置づけています(これ

注12) URL http://www.debian-ports.org/ Debianの公式移植版の前段階としてインフラを提供するプロジェクト。
注13) build daemon、パッケージビルドサーバのこと。
注14) Debian System Adminの略で、debian.orgドメインのサーバ管理を行う人たちを指します。ちなみに同じ略語でDebian Security Advisoryというのもあります。ややこしい。
注15) Debian Archive Kitの略。Debianのリポジトリ管理を行っているツール。
注16) URL http://lists.debian.org/debian-devel-announce/2013/11/msg00004.html
注17) LLVM周りのパッケージをメンテナンスしている方。clangのバージョンアップごとにDebianの全パッケージをビルドしなおしてステータスをまとめている URL http://clang.debian.netなども彼の手によるものです。
注18) URL http://fossasia.org/

をFTBFS(Fails To Build From Source)、つまりソースからビルドできないという略語で示します)。FTBFSはリリースの障害となるRCバグとして扱われるので、リリース前にはすべて解決しなければなりません。この問題への対処策の1つとして、定期的に全パッケージをビルドする「アーカイブリビルド」という作業が、数年前から現プロジェクトリーダーのLucas Nussbaumさんによって開始されました。この作業によって、ビルドに問題の発生しているパッケージを早期に洗い出して修正ができるようになり、リリースの遅延が軽減されています。

すべてのパッケージのリビルド作業ですから、膨大なマシンパワーが必要になります。最初のうちはLucasさんが利用可能なフランスの研究機関所有のグリッドクラスタマシンによって行われていました。しかし、これだとLucasさん以外の人が触れないことから、現在はグリッドクラスタからAWS上に移行して作業が継続されています(現在はDavid Suarezさんがメインで実施)。そして、この作業にかかる費用はすべてAWSからの寄付で賄われており、2014年はその寄付額が8,000ドルとなることが報告されました。AWSのソリューションアーキテクトであるJames Bromberger さんがDebian開発者であることから一連の寄付が実現したとのことで、ありがたい限りです。

ただ、定期的とはいえアーカイブリビルドの実施とパッケージのアップデートにはタイムラグが生じているので、個人的には「より早期の対処を目的としてCI(継続的インテグレーション)が実現できればいいな」と考えています。

ゲームOSとDebian

コアなゲーマーの方であれば、「Steam」の名前を聞いたことがあるのではないでしょうか。SteamはValve社が提供するゲームのダウンロード販売サービスで、このSteamで購入／ダウンロードしたゲームをプレイすることに特化したOS「SteamOS」(現在開発中)はLinuxを採用していることが知られています。以前ささやかれていたようにUbuntuをベースとしているのではなく、Debianに手を入れたものであることが明らかにされています。

SteamOS FAQ[注19]によると、

- Debian 7“Wheezy”ベース
- glibcをtestingからbackport
- kernelも3.10ベースに変更
- グラフィックドライバを更新、その他グラフィックコンポーネントを独自で実装
- ValveのSteamOSリポジトリから自動更新するようにしている

などが記載されています。また新たなDebian派生ディストリビューションが登場するわけですね。

Debian 8“Jessie”用アートワーク募集中

最後は軽めの話題で。次のリリース用デスクトップテーマの公募が始まっています[注20]。Debian 6ではポップな「Space Fun」、Debian 7では打って変わってシックな「Joy」が採用されました。Debian 8ではどのようなものになるのか、すでにいくつかのテーマがDebian Wikiに掲載されているようですので[注21]、ちょっと覗いてみてJessieのデスクトップを想像するのも楽しいのではないでしょうか。デザインに自信のある方はぜひご応募ください。

◆ ◆ ◆

読者のみなさんの興味を引くような話題はありましたか？　ぜひご意見を@gihyojpまたは@debianjpまでお寄せください。可能な限り反映させていただきます。末尾になりましたが、本連載を今年もよろしくお願い致します。SD

注19) URL http://steamcommunity.com/groups/steamuniverse/discussions/1/648814395741989999/

注20) URL http://lists.debian.org/debian-devel-announce/2013/11/msg00002.html

注21) URL https://wiki.debian.org/DebianArt/Themes

2014年はBRMSが来る?

梅野 昌彦
UMENO Masahiko
レッドハット㈱
シニアソリューションアーキテクト

はじめに

こんにちは。レッドハットでミドルウェアのソリューションアーキテクト(プリセールス)を担当している梅野です。とくにビジネスルールマネージメントシステム(BRMS)を中心に担当しています。BRMSはここ2年くらいで急速に注目を浴びてきています。

BRMSって何?

2007年からこのBRMSに関する仕事をしています。当時は大手検索エンジンで検索しても、出てくるのはベンダーのサイトが数件ある程度でした。それが今では21万件以上のヒットがあります。かなりメジャーになってきましたが、今一度ここで説明します。

皆さんが会社で業務を行うにあたり、会社の決まりに従って行動していると思います。モノを作っている会社ではレシピに従ってモノを作っていく、サービスの販売ではオプションの組み合わせで、これとコレの組み合わせはOKだけど、アレとコレの組み合わせにはソレも必要と説明したりしているでしょう。

これらすべてが人の頭の中に入っていて、誰でも同じ品質で作業ができていれば問題はありませんが、競争相手が新しいことを考えてくると、業務はだんだん複雑になっていきます。そうなると、デキる人にばかり仕事が回されたり、確認や承認に時間がかかってくるようになります。おそらく皆さんの会社・組織でもそのようなことがあるのではないでしょうか。

このために、「アプリケーション」というものを作って、仕事を楽にしようとしています。ITが業務を支えているところです。

ところが、その「アプリケーション」の作りが、とりあえず動けばいいや……という観点で作られていると、業務の変更要求が来た場合に応えられなくなります。アプリケーションの改造に多額のお金と手間がかかってくることになります。

そこで活躍するのがルールエンジンです。アプリケーションの主要な部分を占める、データ、プロセス、ルールのうち、プロセスとルールの部分をアプリケーションから切り出し、外で管理・実行するようにします。さらに、その「ルール」を、スプレッドシートで記述できて、書いたものがそのまま動くとなると、メンテナンス性が非常に高くなります(**図1**)。

BRMSの2つの顔

今読んでいただいた部分を、筆者はBRMSの表の顔と言っています。わかりやすい表現でロジックを作るというところです。もう1つに裏の顔というのがあります。それは、パターンマッチングをベースとした人工知能のアルゴリズムを使うことで、難しいロジックを簡単に記述することです。難しくは推論という分野になります。40年ほど前に発明されたRETEアルゴリズムというのを脈々と受け継いでいます。その昔は、エキスパートシステムと呼ばれていたこともあります。

今までRETEアルゴリズムが受け入れられて

▼図1　ルールの説明

ビジネスルール

ポイント100倍キャンペーン！
2014/1/1 0:00〜2014/3/31 23:59までに下記の組み合わせでご購入の場合、ポイントを100倍差し上げます！
ViViDカラーのツバサと妖精の鏡の組み合わせ
白鳥のドルチェとダイヤモンドの組み合わせ
…

どちらがわかりやすいでしょう？

通常のプログラミング

```
while (basket[i] == null){
    if (basket[i] == ViViD) flag = true; i++;}
while (basket[i] == null) {
    if (combi == true && basket[i] == Yousei) point = point * 100; i++;}
…
```

BRMS

RuleTableキャンペーン				
CONDITON	CONDITON	CONDITON	CONDITON	ACTION
Item	Item	b:Basket		
Itemname	Itemname	Date >= $1	Date < $1	b.setPoint(point *$1)
商品名1	商品名2	キャンペーン開始	キャンペーン終了	Point倍率
ViVidカラーのツバサ	妖精の鏡	2014/1/1 0:00	2014/3/31 23:59	100
白鳥のドルチェ	ダイヤモンドJewels	2014/1/1 0:00	2014/3/31 23:59	100

いなかったのは、そのままでは扱いが難しかったのと、オブジェクト指向でメモリを多く消費する傾向にあるからです。ビジネスルールという表の顔を得たことと、メモリの価格が下がり64bit OSが当たり前になって利用可能なインフラが整ったお陰で、ようやく陽の目を見るようになってきたように思えます。

BRMSの効果

ロジックが可視化されることは意外と多くのメリットがあります。業務ユーザが書いたスプレッドシートをそのまま取り込めるので、少しですが工数の削減になります。また、仕様の変更時に、スプレッドシートで直接変更指示ができますので、システム開発側との言葉の齟齬がなくなります。

開発スタイルも、設計時からプロトタイプを繰り返し作って、少しずつ大きくしていく形になります。デスクワークだけで作られた設計書を元に作られたプログラムと、技術的に裏を取りながら作られた設計書で作られたプログラムを比較すると、テスト工程まで行ったときの品質が全然違います。その結果、開発工数、テスト工数が少しずつ削減され、ちりも積もれば山となり、結果的には大きく工数が削減されることが多くのプロジェクトで実証されています。

残念なことにSIは今や精神的におかしくなってしまう人も生んでしまう産業になってしまっていますが、残業がない・品質が悪くならないとなると、不幸になる人も減ります。経営者からするとこの効果は絶大なはずです。

ITとしての直接的な効果だけでなく、ビジネスへの間接的な効果、経営への副次的な効果がジワジワ出てくるのが特徴です。

恵比寿といえば

この見出しにも鯛が描かれているのに、今までの恵比寿通信の中で触れられていないので筆者が紹介します。恵比寿といえば、「ひいらぎ」という鯛焼き屋さんがあります。パリっとした薄皮の中に甘さ控えめのアンコがギッシリ詰まっています。筆者が心を動かされて、人生で初めてファンクラブに入り、仕事のPCにもサインを頂いた歌手のMay'n（メイン）さんもここの鯛焼きが大好きで、店内にはサインも飾ってあります。恵比寿に来られたときはぜひ訪れてみてください！　May'nさんのライブは、時には楽しく、時には誰にも触られたことのないココロの隙間をつつかれるようで、歌がとても熱くて素敵です。今年はデビュー10周年で、47都道府県＋ワールドツアーを行うようなので、ぜひ遊びに行ってみてください！ SD

第46回 Ubuntu Monthly Report

ReVIEWで電子書籍を作成してみよう

Ubuntu Japanese Team
あわしろいくや　AWASHIROI Ikuya
Mail ikuya@fruitsbasket.info

今回はReVIEWを使用して1つのソースからPDFとEPUB形式の電子書籍を作成してみます。

電子書籍とReVIEWとフリーと

読み手としては、電子書籍は十分に普及したと言っていいと思います。筆者もAmazonのKindleや角川のBook Walkerでライトノベルやマンガをたくさん読んでいます。では書き手としてはどうでしょうか。PDFで出力するのであれば、おおよそどのようなアプリケーションでもいいですが[注1]、やはりEPUBでも出力したいです。PDFもEPUBも互いに得手不得手があり、読者に選択する権利があるのが望ましいと考えるからです。

筆者が普段執筆に使用しているLibreOfficeでも、拡張機能をインストールすればEPUBで出力することもできます。しかし、当然ですがLibreOfficeが持つ機能のごく一部分しか反映できず、それを意識すると執筆がおろそかになるという本末転倒な未来が見えます。やはり専用のツールを使用するのがいいと考えました。

EPUBにもPDFにも出力できる電子書籍作成システムはいくつかあるようですが、筆者は迷いもなくReVIEW[注2]にしました。記法が比較的容易であり、かつ普通のテキストエディタで執筆できることと、Ubuntuで容易に環境構築できそうなことが決め手でした。仮想環境などを使わず、最短ならたった2行のコマンドで準備完了というのは、それだけで極めて魅力的です。PDFはバックエンドでTex Liveを使用しているうえに、ReVIEWの開発は日本人によって行われているので、組版ルールを最大限に考慮した美しい仕上がりが期待できたという点も重要です。実際に高品位で満足しています。

ReVIEW自体のライセンスはLGPLであり、その他使用するアプリケーションはすべてオープンなライセンスのものです。そればかりかEPUBもPDFもフリーなフォーマットです。これもフリーにこだわりたい筆者にとってはとても魅力的でした。

ReVIEWの構造

ReVIEWは1章につき1ファイルにしてください。ファイルにはreという拡張子を付けます[注3]。CHAPSには、各章のファイル名を記入します。1行目が1章、2行目が2章となります。1章しかなくてもこのファイルは用意してください。

画像はimagesファイルにまとめます。注意すべきはファイル名で、対応するreファイルの拡張子を除いたものと画像のユニークな名前をハイフンでつなぎます。具体例はのちほど紹介します。画像の形式はJPEGやPNGが使えるようですが、今回はPNGを使用しました。

注1）とくにUbuntu上だとPDFでの出力は印刷するように簡単に行えますし。

注2）https://github.com/kmuto/review

注3）仕様上はこれでなくてもいいようではありますが、これ以外のものにする積極的な理由が思い当たりません。

ReVIEWの環境準備

今回はVirtualBoxを用意し、そこにXubuntu 13.10をインストールしました。もちろんUbuntuなどでもいいですし、ReVIEWを使用するだけであればGUIすらも必要ありません。しかし、制作した環境で手軽にプレビューしたかったので、仮想環境でも軽快に動作するXubuntuを選択しました。Ubuntuのバージョンは12.10以降であればなんでもいいです。サポート期間を考えると13.10が無難でしょう。VirtualBoxのバージョンは4.2.20や4.3.4以降にしてください。これより前のバージョンでは、共有フォルダの扱いに不具合があります。どうしてもこのバージョンのVirtualBoxが用意できない場合は、Ubuntuのバージョンを下げてください。

共有フォルダはゲストでGuest Additionsをインストールし、vboxsfグループにユーザを追加します。コマンドで追加する場合は、

```
$ sudo adduser $USER vboxsf
```

を実行してください。ホストでは、[設定]-[共有フォルダ]-[共有フォルダを追加]で共有フォルダを指定してください。その際、[自動マウント]にチェックを入れてください(**図1**)。このようにPublicフォルダを共有フォルダに指定した場合は、ゲストOSからは"/media/sf_Public"でアクセスできるようになります。

いよいよ必要なパッケージをインストールします。今回はrubygemsとtexlive-lang-cjkとtexlive-fonts-recommendedが必須です。必須ではありませんが、EPUBの確認用にCalibreをインストールしておくと便利です。コマンドラインからインストールする場合は、**図2**のコマンドを実行してください。

実のところReVIEWはSubversionやGitからもインストールできますが、今回はgemにしました。インストールするだけでパスが通っているので楽というのが主な理由です。開発版ではなくリリース版を手っ取り早く使うにも、gemはいい選択肢だと思います。ReVIEWのgemは次のようにインストールしてください。

```
$ sudo gem install review
```

サンプルでビルド確認

ReVIEWのgemは/var/lib/gems/1.9.1/gems/review-1.1.0にインストールされています[注4]。ここにはサンプルの書籍も収録されています。まずはこれでビルドができるかどうか確認してみましょう。

testフォルダの下にsample-bookフォルダがあるので、これをまるごとHOMEフォルダにコピーします。今回はHOMEフォルダ直下にしますが、もちろん別のところでもかまいません。コマンドラインからコピーする場合は、次を実行してください。

```
$ cp -r /var/lib/gems/1.9.1/gems/review-1.1.0/test/sample-book/ ~/
```

では、実際にビルドしてみます。まずはHTMLに変換して確認します。ここから先はコマンドラインで行うのがいいでしょう。

```
$ cd ~/sample-book/src
$ review-compile --target html -a
```

これで ch01.htmlとch02.htmlとpreface.htmlの3つのファイルができました。targetオプションで変換する形式を選択し、aオプションでそのフォルダにあるすべてのreファイルを変換します。もしch01.htmlだけを変換する場合は、

```
$ review-compile --target html ch01.re > ch01.html
```

注4) 執筆の段階でです。バージョンによって読み替えてください。

図1 VirtualBoxの共有フォルダの設定

図2 必要なパッケージのインストール

```
$ sudo apt-get install rubygems texlive-lang-cjk texlive-fonts-recommended calibre
```

としてください。見てのとおり、リダイレクトしないと標準出力に表示されるだけで終わってしまいます[注5]。サンプルですので当然エラーは出ませんが、重大なエラーがあったら標準出力に表示されることがありますので、そのエラーは見逃さないでください。

HTMLを確認したら、次はEPUBを出力してみます。EPUBの出力にはzipパッケージが必要ですが、Xubuntuにはデフォルトでインストールされているのでとくにインストールはしていません。次のコマンドを実行してください。

```
$ review-epubmaker config.yml
```

config.ymlファイルはその名のとおり各種設定が書かれています。ここにbooknameというパラメータがあり、出力に成功するとこれに拡張子（今回はepub）がついたファイルができます。すなわち、今回はbook.epubが出力されます。そのほかにbook-epubというフォルダも出力されています。review-epubmakerを再実行する場合は、このフォルダとファイルを事前に削除してください。

作成されたEPUBファイルを確認する場合は、先ほどインストールしたCalibreを使用します。book.epubをダブルクリックしてみてください（**図3**）。

次はPDFを作成してみます。次のコマンドを実行してください。

```
$ review-pdfmaker config.yml
```

ここでもconfig.ymlファイルを使用しています。作成されるファイル名も同じルールで、book.pdf（**図4**）とbook-pdfフォルダができました。再作成する場合も、やはり事前にこれらのファイルとフォルダを削除してください。確認した限りでも、PDFの出力はエラーが出やすいです。ただ、このサンプルではtexlive-fonts-recommendedさえインストールされていればエラーは出ないはずです。

注5）もちろんページャーに渡して脳内でHTMLをレンダリングしてもいいのですが、普通はファイルとして出力してWebブラウザで確認するのがいいと思います。

図3　CalibreのEPUB表示アプリケーションでサンプルのEPUBファイルを表示させた

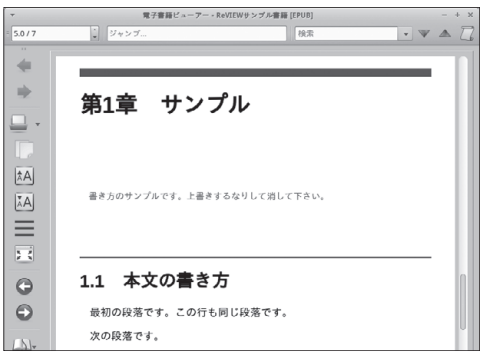


図4　book.pdfを表示させた

ReVIEWの記法

ビルド環境を整えるのと、実際に書くのとはもちろん順番が前後してもかまわないので、まずはとにかくReVIEW形式の文章を書き始めるのもいいと思います。なお、/var/lib/gems/1.9.1/gems/review-1.1.0/doc/format.rdocにより詳細な解説がありますので、こちらを参照してほしいのですが、ここには書かれていないハマりどころを解説していくことにします。

■箇条書き

箇条書きを有効にするためには、前後にも改行が必要です。すなわち、箇条書き全体が1つの段落になっている必要があります。

■脚注

記法に関する注意ではなく運用上の注意ですが、脚注には識別子[注6]が必要で、最初のうちは数字を使用していましたが数が多くなると破綻するので、こ

注6）という表現が正確なのかはわかりませんが。

れはやめたほうがいいです。

■図

図の記法は、

```
//image[ファイル名][キャプション]
```

ですが、ファイル名には拡張子は含めません。また、実際の画像のファイル名とそっくりそのまま使用すればいいわけでもないです。たとえばubuntu.reというReVIEWファイルからsaucy.pngを呼び出したい場合、

```
//image[saucy][これがSaucyのデスクトップ]
```

と記述します。画像のファイル名はimages/ubuntu-saucy.pngとすると呼び出せます。

■表

表自体はシンプルな記法ですが、PDFにした場合は改行しないので、右端が切れてしまいます。1行をあまり長くしないほうがよさそうです。また、表中での脚注も正しく使用できなかったので、頭を抱えるということがありました。

実際に書いてみる

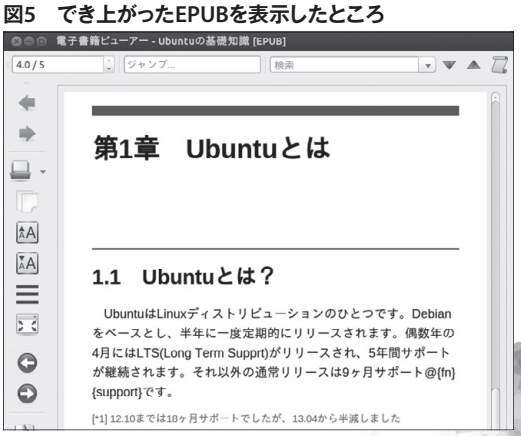
というわけで、今回は試しに『Ubuntuの基礎知識』という本を書いてみます。内容は**リスト1**(introduction.re)のとおりです。これをサンプルフォルダ(~/sample-book/src)に保存します。CHAPSファイルを開き、"introduction.re"(引用符は不要)の1行だけにします。今回はPREDEFファイルとpfeface.reは使用しないので削除します。最低限これは必須です。表示画像がimages/cover.jpgにありますが、今回はcover.pngを新たに作成しました。その場合、_cover.htmlを開き、忘れずにファイル名を変更してください。

最後にconfig.ymlを編集します。おおよそコメントのとおりに書き換えればいいので、とくに躓くところはないはずです。あとは先にも出た、"review-epubmaker config.yml"や"review-pdfmaker config.yml"を実行し、EPUBやPDFを生成してください(**図5**)。SD

リスト1　ReVIEWで書いたサンプル(introduction.re)

```
= Ubuntuとは

== Ubuntuとは？
UbuntuはLinuxディストリビューションのひとつです。Debianをベースとし、半年に一度定期的にリリースされます。偶数年の4月にはLTS(Long Term Supprt)がリリースされ、5年間サポートが継続されます。それ以外の通常リリースは9ヶ月サポート@{fn}{support}です。
//footnote[support][12.10までは18ヶ月サポートでしたが、13.04から半減しました]

バージョンはリリースされた年と月から取り、コードネームも割り当てています。

ほかにも、

 * 主としてデスクトップ環境を変更したフレーバーを多数用意
 * 企業によるサポートあり
 * サーバー版も用意

といった特徴があります。

==[column] Ubuntuの意味は？
Ubuntuは、ズールー語やコサ語で「他者へのおもいやり」という意味です。@<br>{}
発音は「ウブントゥ」が近いです。しかし「ウブンツ」と発音する人も多いです。

==現在サポートが継続しているUbuntuのバージョン
//table[version][Ubuntuのバージョン]{
バージョン　リリース日　サポート終了月
----------
10.04  2010/04/29    2015/04
12.04  2012/04/26    2017/04
12.10  2012/10/18    2014/04
13.04  2013/04/25    2014/01
13.10  2013/10/17    2014/07
//}
ただし10.04ではサーバー版のみで、デスクトップ版はすでにサポートが終了しています

==ダウンロード
Ubuntuでは日本語Remixという日本語向けのバージョンがあり、特段の理由がない限りはこれを使用してください。

@<href>{http://ubuntulinux.jp/, Ubuntu Japanese TeamのWebサイト}からダウンロードしてください。
```

図5　でき上がったEPUBを表示したところ



February 2014

NO.28

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

関西ITコミュニティの熱気に包まれた2日間 KOF2013

今回は2013年11月に大阪で行われた関西オープンフォーラムの模様をお届けします。

関西オープンフォーラム2013

■関西オープンフォーラム2013
【日時】2013年11月8日（金）～11月9日（土）
【会場】大阪南港ATC ITM棟

関西圏では屈指のITコミュニティイベントである関西オープンフォーラム（KOF2013）が2013年も開催され、2日間合計で約1,500人の参加者を集めました。実施されたプログラムの中から一部を紹介します。詳細はKOF2013のWebサイト[注1]を参照してください。

■基調講演

実行委員会が講師を招待して行う講演会です。今回の講演は次の4本でした。

- **「パーソナルデータ保護を考える」**
  高木浩光［産業技術総合研究所］
- **「地場産業としてのオープンデータの可能性」**
  大向一輝［国立情報学研究所］
- **「国内最大級の省エネ型データセンター『石狩データセンター』のご紹介」**
  田中邦裕［さくらインターネット］
- **「『知らなかった』から『聞いたことがある』へ ～『子供とネットを考える会』で考えていること」**
  山口あゆみ（はなずきん）［子供とネットを考える会］

注1）URL http://k-of.jp/2013/

とくに高木さんの登壇は開催前から話題になり、多数の参加がありました（**写真1**）。山口さんの講演も、大人たちの間では意外に知られていない子供のネット利用の実態が詳細に報告される濃い内容で、反響が大きかったです。

■ユーザ企画

参加団体が自主的に企画し、実施するセミナーなどのセッションです。10階の多目的ホールとサロン、9階のセミナールーム2部屋、6階PCルームを利用し、東邦之さん（Cubicroot）による「実践的bouncehammer」、新原雅司さん／原田康生さん（Kansai PHP Users Group）による「Composerを活用しよう」など、2日間で31本を実施しました。jusもjus研究会大阪大会を行いました。

■展示企画

10階のデザインギャラリーとデザインショーケースを利用しての展示会です（**写真2**）。ファーエンドテクノロジー、Joe'sクラウドコンピューティング、KDDIウェブコミュニケーションズなど47団体が出展しま

写真1　基調講演の様子

した。例年の光景ですが企業もコミュニティも机1つでところ狭しと出展する様子は、このイベントの熱さと手作り感を象徴しているように思います。

■ショーケース

新たな試みの1つとして、デザインショーケースの2部屋を利用して、従来の枠には収まらないような形態のプログラムを実施しました。1つはMozilla JapanによるFirefox OS搭載スマートフォンの展示とFirefox OSアプリ開発ワークショップです。もう1つはCTF(Capture The Flag)[注2]のデモと解説で、実際にセキュリティ系の問題を解く様子を画面で見せつつ解説がなされました。

■ステージ企画

展示会場の片隅に設けた特設ステージで繰り広げられるショートプレゼンテーションです。下野智美さん(GMOクラウドWEST)による「ゥピコ誕生物語」、鈴木敏郎さん(神戸ITフェスティバル実行委員会)による「『自慢したくなるスゴいIT』神戸ITフェスティバルの自慢話」をはじめ、後述する書籍関連のセッションも含めて合計27本を実施しました。例年実施している15分間の対談形式のほかに、初の試みとして5分間のライトニングトーク形式の発表も行いました。

■ジュンク堂書店KOF店

展示会場の一角にジュンク堂の出張書店が設けられました。参加団体から推薦された書籍を取りそろえ、充実したラインナップでの出店となりました。また、書店ブースの向かいにある特設ステージを利用しての書籍紹介やサイン会も行われました。とくに、横田真俊さん(さくらインターネット)による『ブログ+ツイッター+フェイスブック使い分け・連携テクニック』[注3]の書籍紹介は観客も多く、盛り上がりました。

写真2　展示会の様子

■ウォーキングツアー

実行委員の引率で会場内を見て回るツアーです。金曜日2回、土曜日3回の計5回実施しました。10階の展示会場からスタートし、ユーザ企画が行われるサロンや多目的ホールを経由して、9階のセミナールームまで足を伸ばしてから展示会場に戻る、所要時間約30分のコースです。参加者は毎回数名ですが、1人でも多くの人にこのイベントの開催意義を理解してもらうことは重要と考えています。

■懇親会

KOFにおける最重要イベントの1つが懇親会です。例年どおり金曜日の夜に行いましたが、前回まで利用していた店が閉店したため、今回はITM棟6階にあるPIER6にて開催しました。152人という大勢の参加があり、ビアスポンサー提供によるベルギービールをはじめとするお酒や料理とともに、コミュニティ活動に携わる者同士の熱い交流が展開されました。

■終わりに

jusはKOF創立時から共催という形で参加していましたが、KOFの財政もようやく安定してきたので、2013年からは人的支援を中心とする協賛という名義に変わりました。しかし、名義は変わっても幹事数名が実行委員会に名前を連ね、当日も現場で手伝うといった協力体制はこれからも続けていきます。2014年のKOFも11月7日(金)～8日(土)に大阪南港ATCにて開催の予定です。SD

注2) DEF CONなどで実施されているコンピュータセキュリティ技術の競技。リバースエンジニアリング、パケット分析、プロトコル解析、暗号解読などの知識技能が試される。

注3) 横田真俊、秀和システム、2012年9月、978-4-7980-3513-0

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26回 「ITx災害」会議（前編）

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。今回と次回は2013年10月に開催された「IT災害」会議の模様を前後編に分けて紹介します。

●Hack For Japanスタッフ
及川 卓也　OIKAWA Takuya
Twitter @takoratta
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka

Hack For Japanは2011年3月11日の東日本大震災発生直後に活動を開始し、まもなく2年9ヵ月（執筆時点）になろうとしています。今までの活動の中には、成果を出せたものもあれば、かけ声だけに終わったものや、あまり効果的ではなかったものなどもあります。

また、私たちHack For Japanの活動はITを活用した復旧・復興支援の中でも「開発」に重きを置いたものですが、私たち以外にも多くのITを用いた復旧・復興支援を行っている団体やコミュニティ、そして個人がいます。

思いを同じとする人々が集まり、今までの活動を振り返るとともに、今後の活動を検討し、協力関係の可能性を検討するためにと企画されたのが、「ITx災害」会議です。

2013年10月6日に開催されたこの「ITx災害」会議の模様を今回はご紹介します。

きっかけと準備

2012年、Hack For Japanでは、復興支援などに対してITの力を活かして貢献したいという思いを持つ方と支援を必要としている方とをつなぐための取り組みである、スキルマッチングを開始しました[注1]。しかし、今に至るまであまり活用されていません。Hack For Japanの活動に賛同してくださっている人は多いものの、やはり開発者が参加者の多数を占めるHack For Japanのコミュニティだけでは協力関係構築が難しかったのではないかと考えられます。この課題はスタッフの中では認識されており、コミュニティの拡大やほか団体との連携などが以前から検討されていました。

そのような中、Hack For Japanに当初から参加いただいている方から、今までの活動を振り返り、ITによる復興支援のあり方を考える必要があるのではないかという声もあがってきました。

これらがきっかけになり、「ITによる復興支援のあり方会議」（当初、「ITx災害」はこう呼ばれていました）の準備会が発足しました。2013年6月末に行われた最初の準備会では30名近くの方が集まり、震災以降の取り組みなどが共有されました。お互いにそれぞれの存在は知ってはいたものの、直接話すのは初めてという参加者も多く、「つながる」という意義を再確認し、本会議への期待がさらに高まりました。

その後の準備会で、将来に起きうる災害に対しての防災や減災も話し合うことから、会議名を「ITx災害」会議（「x」は「かける」と読みます）とすることに決定しました。また、参加者全員に積極的に参加してもらうため、アンカンファレンスという形式をとることとしました。

アンカンファレンスとは、Software Designの読者であればすでにお馴染みかもしれませんが、あえて事前にセッション内容などを決めずに、当日に参加者自身に提案してもらう形式のカンファレンスです。通常のカンファレンスの場合、セッションで話す人と聞く人というのが明確に別れてしまいがちですが、アンカンファレンススタイルの場合はセッションのトピックを提案できますし、たとえほかの人が提案したトピックであったとしても、事前にス

注1　http://blog.hack4.jp/2012/12/blog-post.html

ライドなどが準備されていることはまれなので、全員が同じ立場でセッションに参加できます。

準備会は合計4度行われ、総勢20名以上のスタッフが手分けして準備を進める、文字どおりの手作りの会議として用意され、当日を迎えました。

午前の部：ショートスピーチ

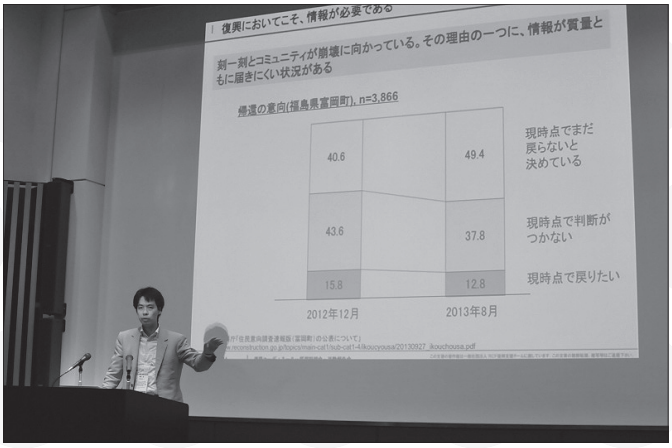
「ITx災害」会議は10月6日に東京大学駒場リサーチキャンパスにて行われました（**写真1**）。参加者は112名となりました。この会議は招待制としたのですが、招待した方の9割以上が参加されるという大変高い参加率の会議となりました。

午前の部はイベントの主旨の説明と、これまで復興支援活動をされてきた10名の方からのショートスピーチが行われました（**写真2**）。

◆◆◆

◆**写真1　会場風景**

◆**写真2　ショートスピーチの様子**



◎RCF復興支援チーム：藤沢烈さん

情報を復興に活用する3つのキーワードを紹介したいと思います。

1. ラストワンマイル～地域コミュニティ形成

「復興においてこそ、情報が必要である」

ラストワンマイルをつなぐために現地コーディネーターが必要

2. One to One～産業復興

被災者、被災事業者と支援者がOne to Oneで結ばれることが意味があり、持続的なかかわりが求められる

3. テーマ別のプラットフォーム～人材支援

今回の震災は被災地も多く、支援も多い。被災地と支援側のニーズを集約し、見える化するテーマ別のプラットフォームが必要

復興には10年以上かかり、まだ入り口の段階。引き続き東北を支えるためにも、情報の整理がたいへん大事だと思っております。

◆

◎岩手震災IT支援プロジェクト：村山優子さん

災害時のネット接続支援の課題として、IT支援の重要性が必ずしも認識されないというものがありました。まず支援ありきのスタンスではダメで、相手が必要とするものを理解し、人や車が先、そのうえで情報交換が必要です。また、組織プロトコルも重要で管轄する部署への説得、なぜインターネットが必要か、今起こっていることをどうやって意思疎通するかということが大切です。できることから1つ1つ解決していくと信頼が出てきて、最終的には情報処理が必要だということがわかってもらえます。

◎さくらインターネット研究所：松本直人さん

写真からはたくさんのことがわかります。言葉は曖昧さを持っているため、より多くの視覚情報の共有、正確な位置情報、データ加工の容易さが大切です。とくにGPS情報付き写真は災害直後の復旧のときに有用となります。必要な情報がリアルタイムで手に入る環境づくりを見据えて、GPS情報付き写真を発信しましょう。常に自分でも情報発信することを心がけていると役に立つことがあります。

◎うらと海の子再生プロジェクト：小泉勝志郎さん

私自身が宮城県塩竈市で被災し、弟が携わっている浦戸諸島という離島の養殖業も壊滅状態となってしまいました。それを救うために始まったのが「うらと海の子再生プロジェクト」で、クラウドファンディングで1億8千万円を集め、現在はオーナーさんにほぼ返した状態となっています。

また、よくボランティアが集まらないという話を聞きますが、つらいとか大変という言葉ばかりでは誰も来たいと思いません。楽しい要素を伝える必要があります。たとえば浦戸は海の物が豊富でそれを採ったり食べたりすることが楽しめるといった地域の魅力を伝えていけば、ボランティアも集まってくるのではないかと思っています。

◎岩手医科大学：秋冨慎司さん

スーパーコンピュータによる試算では揺れだけを計算して、津波を考慮していませんでした。それが岩手は大丈夫という当初の誤認識につながりました。情報の持つ怖さというものがあると思います。

また、災害後のフェーズは10のn乗の時間で考えることが大切です。

1時間：自分たちの安全確保
10時間：救助、救援の準備、情報収集
100時間：人命救助
1000時間：復旧活動
10000時間：復興活動

このように受ける側、送る側の双方がきちんと時間軸で考えていく必要があります。

情報のマネジメントの仕方によっては、嘘の情報でも本当の情報になってしまうことがあります。テレビで「かわいそうですねぇ、ここの避難所」といったような取り上げられ方をした場所に支援が集中し、斑（まだら）が発生します。また道路が復旧したという情報も、民間の利用により、自衛隊などが使えないことがあってはいけません。情報は魔物にも神にもなると思っております。

◎株式会社Eyes, JAPAN：山寺純さん

アメリカで盛り上がっているヘルスケアITの日本版を推進しており、Health 2.0 Fukushimaチャプターを務めています。昨年は医療ハッカソンも行いました。

◎情報支援プロボノプラットフォーム（iSPP）：岸原孝昌さん

被災地の情報サービスにおいてさまざまな取り組みが行われたと思うのですが、ちゃんと記録にとっておかないと次に進められません。そこで、地域と時系列、情報ツールが生きていたのか、提供されたものと必要とされた情報の差などについて情報行動調査を行いました。

災害発生直後はラジオが有効で、徐々に携帯電話、インターネットなどのメディアに移行していき、必要とされた情報も時系列とともに変わってくることがわかっています。まだ構想段階ですが、情報支援レスキュー隊のコンセプトを検討しています。

◎Code for Japan：高木祐介さん

Hack For Japanなどで行ってきたハッカソンはコミュニティ形成や問題への理解の促進には効果を発揮するが、継続したサービスに結びつけていくのは難しいと考えて始めたのがCode for Japanです。オープンデータを活用することで地域防災を推進していき、緊急時にスムーズな対応ができるように地域でのITコミュニティ作りを支援します。

◎イトナブ石巻：古山隆幸さん

震災後に街づくりの活動をしていく中で、ITを活用して若者たちが動けるフィールドを作っていこうと始

めたのがイトナブです。支援を受け続けるだけではなく、自活できる人作り＝自立した産業の開拓サイクルを回していくという活動を目指しています。

◇

◎IPA/WASForum：岡田良太郎さん

震災後、個人として社会継続のために自発的に行動した人たちによるWeb上の支援サイトを調査したところ、地理情報を扱ったものが44%、人命を取り扱ったものが41%、さらに起案からリリースまで72時間以内のものが52%、さらに半数が3名以下のプロジェクトだったという統計が得られました。利用可能なデータがなくて苦労したという話もありますが、プライバシーの問題など、当時はスピードのことを考えて意識していなかったこと、なりゆきでやってはいけないことが、改めて考えると多々あるはずです。

◇◇◇

参加証にひと工夫して交流のきっかけに

会議では参加者間の交流を促すために、いくつかのしくみを用意しました。その1つが参加証です。首から下げた参加証で、名前と所属がわかるというのはどのイベントでも良くある風景なのですが、今回の会議では、その参加証を自作してもらうようにしました。

実は、参加証をスタッフ側で用意し、受付でそれを受け取ってもらう方法にするか、事前に参加者に作ってもらうようにするかは、準備会でかなりの議論を重ねました。というのも、ただでさえ準備に時間がかかり、当日の受付が混乱する可能性もある中で、さらにスタッフにも参加者にも負荷のかかる作業を追加する意味があるかが争点となったからです。

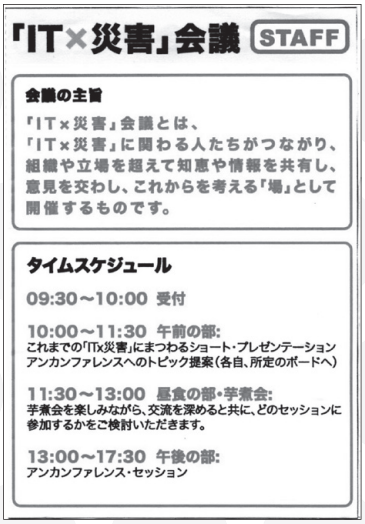
結果から言うと大成功でした。PDFのテンプレートを用意し、参加者に項目を埋めてもらうようにした[注2]のですが、それぞれ工夫を凝らしてさまざまな参加証を用意していただきました。**図1、2**は筆者（及川）の参加証をスキャンしたものです。

今回はPDFでテンプレートを用意したのですが、「開発」「運用」「インフラ」などの自分の専門分野に関しては複数選択可能としていたので、Acrobat Readerのレイヤー機能を利用して、該当する分野のレイヤーを有効にしてもらうようにしました。残念ながら、ITの専門家ではない参加者には難易度が高い操作だったようで、ふと気づくと、そこかしこにすべての分野の専門家というスーパーエクスパートがいるという状態になっていました。次回があるならば、この部分はWebのフォームに該当項目を入力することで、PDFが自動生成されるようなシステムを用意できればと考えています。

次号では後編として、お昼に振る舞われた芋煮の話と午後の部のアンカンファレンスについてレポートします。SD

◆図1　参加章の例（表）

◆図2　参加章の例（裏）



注2　http://www.itxsaigai.org/20131006/namecard.html

温故知新
IT むかしばなし

第30回

ネットアクセスとバイナリファイルの転送

北山 貴広　KITAYAMA Takahiro　kitayamat@gmail.com

はじめに

今回は電話回線を使ったモデムから現在に至るまで筆者が経験したネットへのアクセスとバイナリファイルの転送について振り返ります。

モデムの時代

現在のように携帯電話などの無線機器が一般化していないころは、通信に利用できるのはアナログ回線と呼ばれる加入者電話のみでした。

そのころに通信に使用されていたのが、パソコンと電話回線をつなぐモデム(Modulator-Demodulator：変復調装置)です。パソコンからRS-232C経由で渡されたデータを音声に変換し、電話回線を使って遠隔の目的地に送信し、先方で逆の経過でパソコンでデータを受け取るものです。

筆者が一番最初に使ったモデムは、秋月通商の手作りキットで友人が作成したものでした。蓋が半透明のタッパウェアに入っていて「弁当箱」と呼んでいました。通信速度はたしか110bpsだったように思います。当時はモジュラージャックではなく電話機と並列に電線をつないで、アクセスする際は電話機で電話番号をまわして相手側の音が聞こえてからおもむろにモデムのスイッチを入れて受話器を置いてました(ちなみに受話器をはめ込むタイプの音響カプラというものもありました)。

その後、300bps→1,200bpsと速くなり、2,400bpsの時代に筆者が使っていたのは当時普及していたオムロンのモデムでした。1988年頃の広告を見ると、OMRON MD2400F MNP Class 5は59,800円となっています。MNPといえばすぐに思い浮かぶのが、携帯電話会社を変更する際の番号ポータビリティですが、当時はモデムで使われてる通信規格の呼称で、誤り訂正や圧縮機能でMNP4とかMNP5とかがありました。

モデムを使っていても音声通話と同じく電話料金がかかりますが、当時はまだ市外通話が高く、アクセスポイントがどちらにあるかで通信料金が大きく違ってきていました。当時、テレホーダイと呼ばれる23時～翌日8時までの固定料金サービスも始まり夜中のネット人口が増大した時期でした。

筆者が通っていた大学や会社では、JUNETに接続して電子メールやNetNewsを使うことができました。その当時モデムとして使われていたのは、アメリカのTelebitが製造していたTrailBrazer T2000 や T2500です。接続する先でこの機種を使っているのでこちらもこれ一択でした。

ISDN (MN128-SOHO)

アナログ回線を使ったモデムでは最大速度で56kbpsほどで回線の状態によっては速度低下もあるため、安定した速度で最大128kbps接続可能なISDN回線は1990年代後半のアクセス方式として使われていました。

とくにNTT-MEとBUGが共同開発したMN128-SOHOはTAとルータ機能が一体化していて、価格も当時としては破格的に安く[注1]、設定方法も現在では一般的となっているWebインターフェースを採用していました。

注1) といっても69,800円しました^^;

当時のモデムやルータの設定は、シリアル接続で直接コマンドやパラメータを渡す方式だったので、一般の人には使いにくいものでした。MN128-SOHOでは外部からPHSのPIAFSプロトコルを使った接続も可能だったため、筆者は自宅にPHSで接続し、そこからインターネットに接続して自宅のルータ経由でインターネットにアクセスしていました。

CATV（子羊ルータ）

その後2000年ごろに筆者の住む西宮市の地域ケーブルテレビのケーブルビジョン西宮でCATVインターネットが開始されました。月額固定料金で下りが256kbpsと、今から見ると低速ですが、固定料金で常時接続できるようになったので喜んで加入しました。

CATV会社から貸与されたケーブルモデムは、DHCPでIPアドレスを1つ割り当てる方式だったので、同時に接続できるパソコンは1台だけでした。そんなわけで、当時はルータを接続して複数台のパソコンから接続できるようにするのが流行っていました。

このころワイルドラボからものすごくコンパクトな子羊ルータ[注2]が発売されました。49,800円でしたが筆者は即買いしました。Linuxがコンパクトな箱で動くのがとてもうれしいのは、現在のRasberry Piがかわいいのとまったく同じ感覚です。

バイナリ→テキストファイル変換

バイナリファイルを転送する方法として、バイナリファイルをアスキー文字列の範囲内に置き換えてテキストとして転送し、受信側は転送されたテキストファイルをバイナリに変換することが行われていました。当時はDOSだとish[注3]、Unixだとuuencode/uudecodeなどが使われていました。また同時に、複数のファイルを1つにまとめるアーカイブ機能と、ファイルサイズを圧縮する圧縮機能を備えた、DOSではZIPやLZH、Unixだとtar.gz（tar ball）が利用されていました。

バイナリ転送プロトコル

バイナリファイルをテキスト化して送信する方法以外にも、直接転送する方法もありました。ただし、シリアル通信ではホストがバイナリ転送に対応していなければならなかったために普及が遅れたのです。

たとえば現在でも利用することができるシリアル転送ソフトにTera Termがあります。筆者が現在使っているTera Term Version 4.71では、メニューバーでファイル（F）-転送（T）を選択すると、Kermit、XMODEM、YMODEM、ZMODEM、B-Plus、Quick-VANを選択することができます。これらはバイナリ転送プロトコルです。当時は、インターコムのまいと〜くや、技術評論社のCCT-98[注4]などが使われていました。

実際にファイルを転送するときは、ホストにログインし、ホストのメニューからバイナリ転送プロトコルを選択し、その後Tera Termなどの端末エミュレータ側で該当するバイナリ転送プロトコルを実行することで、バイナリファイルをアップロードやダウンロードするしくみです。

その間は端末はバイナリ転送に使われていますので、ファイル転送が終わるまでは端末が使えませんでした。

終わりに

携帯電話の新たな通信規格で現在主流となっているLTE（Long Term Evolution）の下り通信速度は、最大37.5Mbps〜326Mbpsとなります。アナログ回線の限界64kbpsからすると1,000倍以上の速度進化ですね。

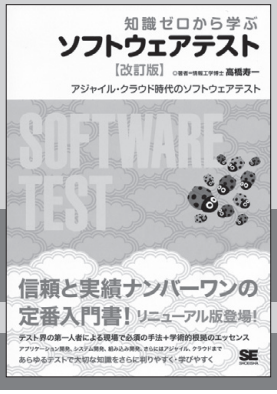
昔は音声の電話回線を使って一生懸命デジタルデータを送っていたのに、今はデジタルの回線を使って一生懸命音声を送っているってなかなかおもしろいもんだと思いますし、土台が変わったのを実感します。SD

注2） 型番はLAMB-RT-01。AMDの486SX互換CPUを搭載した、小型ルータの先駆けで、3.5インチのフロッピーディスク7枚を重ねた大きさ。今だとOpenBlockSのイメージか。

注3） 開発者の石塚氏の名前からishと命名された。当時は通信環境が悪くテキストの通信でも文字化けが頻発していた。ishは誤り訂正機能もあり、当時のデファクトスタンダードだった。

注4） CCTはConcurrent Communication Terminalの略。バイナリ転送中にエディタが使えるために命名された。

BOOK no.1 知識ゼロから学ぶソフトウェアテスト【改訂版】



高橋 寿一 【著】
A5判、 240ページ／価格＝2,400円 （税別） ／発行＝翔泳社
ISBN＝978-4-7981-3060-6

カバレッジテスト、同値分割法、境界値分析法などの伝統的なテスト技法を解説しつつ、最新の技法も学べる。たとえば、ホワイトボックステストの章では、TDD（テスト駆動開発）の考え方や実践例が示される。SI業界ではよく仕様書をもとに作成したテストケースに従って網羅的なテストを行うが、それでもリリース後に単純なバグが見つかることが多い。そんな問題に悩んでいる人は、探索テストの章を読んでみてほしい。テストケースベースのテストでは発見できないバグを見つけるためのヒントが得られるだろう。テストに重要なのは網羅性だけではなく「適材適所でいろいろなテスト技法を試みること」。そんな考え方が身につく1冊だ。

BOOK no.2 人生って、大人になってからがやたら長い

きたみりゅうじ 【著】
A5判、 176ページ／価格＝1,200円 （税別） ／発行＝幻冬舎
ISBN＝978-4-344-02472-4

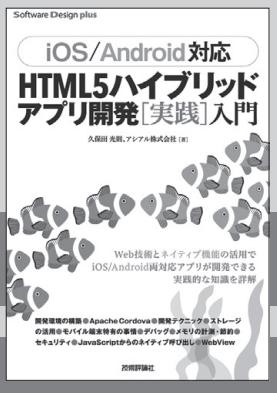
仕事、出産、子育て、家購入など、大人になったら多くのイベントがあり、その時々で悩みがある。そんな悩みに対して著者がどのように考え、乗り越えてきたかを、おもしろおかしく描いたコミックエッセイ。仕事の話題については「ああ、自分も同じ悩みを持ってた！」とうなずけることが多かった。著者は一時期、転職セミナーで講演していたこともあるというから、転職希望者の生の声をたくさん聞いた経験が活かされているのかもしれない。著者の考えも、気負い過ぎておらず、庶民目線なのがいい。日常生活や仕事に疲れたら、本書をパラパラと見てみると良い。頑張るだけでなく、立ち止まって一息つく時間も大切だと感じられるはずだ。

BOOK no.3 Software Design plus シリーズ 過負荷に耐えるWebの作り方 国民的アイドルグループ選抜総選挙の舞台裏

パイプドビッツ株式会社 【著】
A5判、 224ページ／2,480円 （税別） ／発行＝技術評論社
ISBN＝978-4-7741-6205-8

Webシステムにおいて急激なアクセス、すなわち過負荷な状況に耐えることは企業にとってとてもメリットがある。Webページから情報を途切れることなく配信したり、データの入力を取りこぼしなく行ったりといった、普段できていることを過負荷な状況でも遂行できれば運営側の信頼性が著しく向上するのは言うまでもない。本書は、オンプレミス環境下で秒間10,000アクセスに耐えるWeb投票システムの舞台裏を解説したものだ。顧客から示されたシビアな条件をクリアすべく、開発・運用の各面で技術者が工夫をしながらシステムを完成し、実運用に耐えるシステムを作り上げる様子は、Webに関わる人なら誰しも興味が湧くのではなかろうか。

BOOK no.4 Software Design plus シリーズ【iOS/Android 対応】 HTML5ハイブリッドアプリ開発［実践］入門



久保田 光則、 アシアル株式会社 【著】
A5版、 384ページ／価格＝2,880円 （税別） ／発行＝技術評論社
ISBN＝978-4-7741-6211-9

いまやスマートフォンアプリケーション開発では、iOS、Androidの双方に対応することは必須とも言える要件になっている。さらに、短納期、低予算ということも多い。こうした制約に対応しつつ、ユーザのニーズに応えるアプリ開発を実現する方法として、HTML5ハイブリッドアプリが注目されている。本書は、このHTML5ハイブリッドアプリ開発の基礎知識と、現場で役立つ実践的なノウハウ、知識を1冊に集約した書籍。実際の開発で役立つ、Apache Cordova（PhoneGapのOSS版）による開発、ストレージの使い分け、タッチ・ジェスチャなどの活用、デバッグノウハウなどをわかりやすく解説している。

Event

日本OpenStackユーザ会、2月13日、14日に「OpenStack Days Tokyo 2014」を開催

日本OpenStackユーザ会は2014年2月13日、14日の2日間、OpenStack専門カンファレンス「OpenStack Days Tokyo 2014」を、御茶ノ水ソラシティカンファレンスセンターにて開催する。

「OpenStack Days Tokyo」は2013年3月12日に第1回が開催され、国内外22社が講演や出展を行い、700名以上が来場。第2回となる今回は会期を2日間に拡大して開催し、1,500名の来場を見込んでいる。

基調講演は、OpenStackプロジェクト共同創始者の1人であり、OpenStack FoundationのCOOとして、OpenStack開発コミュニティとビジネスエコシステムを成長させるためのマーケティングやビジネスディベロップメントを主導するMark Collier氏が登壇。グローバルのOpenStackエコシステムや最新の導入事例などについて講演する。

また、2013年5月から商用環境にOpenStackを導入し、運用／改善を続けているグリー㈱の渡辺光一氏、松橋洋平氏が登壇する。OpenStackを商用環境にリリースするに至った経緯、自社インフラへOpenStackを導入するにあたり、実際に行った技術面の工夫や悩んだ点について実例を交えた紹介が行われる。また、導入フェーズが一段落した今、OpenStackについて感じていることや、これからの展望についても語られるとのこと。

▼開催概要

名　称	OpenStack Days Tokyo 2014
テーマ	広がるオープンクラウド エコシステム
会　期	2014年2月13日（木）、14日（金）
会　場	御茶ノ水ソラシティカンファレンスセンター 2F（御茶ノ水駅徒歩1分） http://solacity.jp/cc/access/
主　催	OpenStack Days Tokyo 2014 実行委員会
内　容	基調講演、協賛企業セッション、協賛企業ブース展示
対　象	クラウドを導入したいCTO/CIO層、クラウドビジネスの企画者、クラウド業界のビジネスユーザ、クラウド業界内開発者、パートナ企業、データセンター／テレコム業界のビジネスユーザ、ICTへの関心と利用率の高いビジネスユーザ
定　員	1,500名（予定）
参　加	無料
申し込み方法	同イベント公式サイトより申し込みのこと
〆　切	2014年2月7日（金）17:00まで（予定） ※定員になりしだい締め切る

CONTACT　OpenStack Days Tokyo 2014公式サイト
URL http://openstackdays.com

Topic

サイボウズ、脆弱性発見コンテスト「cybozu.com Security Challenge」中間報告

サイボウズ㈱は、2013年11月11日～25日の2週間に渡り開催していた脆弱性発見コンテスト「cybozu.com Security Challenge」の途中結果を発表した。

cybozu.com Security Challengeとは、同社サービスの品質向上を目的にSECCON（特定非営利活動法人日本ネットワークセキュリティ協会が主催する日本最大級のセキュリティコンテスト）の協力のもと実施したオンラインによる脆弱性発見コンテスト。

コンテスト結果

- 開催期間：2013年11月11日～11月25日
- 申込人数：95名
- 参加人数：75名
- 脆弱性の報告者：14名
- 発見された脆弱性：19件

コンテスト実施期間の15日間で、アクティブに検証に参加した人の延べ人数は75名。報告された脆弱性19件を同社で評価した結果、緊急対応が必要な深刻な脆弱性は検出されなかった。

コンテスト上位入賞者は2014年3月に東京で実施される「SECCON全国大会」にて発表し、どんな脆弱性を、どのように発見したのかを本人に紹介してもらうという。

同社は今回のコンテストにより、具体的な脆弱性の発見のみならず、「リアルタイムのモニタリング（画像）によって、攻撃の手法に関するノウハウを得られた」「報告された脆弱性と不具合の切り分け方に関して社内の認識を統一できた」といった成果を得た。報告された脆弱性情報は、同社の脆弱性対応ポリシーに従って、改修していく。

今後は、cybozu.com Security Challengeに続き、セキュリティ検査を実施したい人たちに向けたcybozu.comの開発環境の公開などを検討している。

同社は、引き続き外部の専門家の協力を得ながら、より高度なセキュリティを兼ね備えた製品を提供していくという。

CONTACT　サイボウズ㈱
URL http://cybozu.co.jp

Topic

日本データ復旧協会、2012年のデータ復旧市場規模を発表

データ復旧の業界団体である日本データ復旧協会は、2011年に引き続き、2012年統計のデータ復旧業界の市場規模を発表した。

2012年1月～12月における業界全体のHDDの復旧依頼件数は、前年と比べて4,000台少ない74,000台（2011年：78,000台）。復旧件数は前年の東日本大震災など自然災害が減少した分上昇し、700台多い59,200台（2011年度：58,500台）。

今後のマーケットの展望と課題について

同団体は、今後のマーケットについて、スマホやタブレット端末の普及、クラウドサービスの増加で、クライアントPCは減少するとみている。とはいえ、2013年に起こったデータセンターの大規模な事故をきっかけに、クラウドサービスを利用した社外でのデータ保持が必ずしも安全ではないという認識が広がり、社内でのデータ保持の必要性が見直されている。そのため、サーバやNASなどの案件は現状維持、または微増するだろうと予測している。

また、2014年はWindows XPサポート期間終了に伴うPCリプレイス時のデータ事故も多く予想されることから、業界全体として準備態勢を整えているという。

さらに、同団体はタブレット端末、スマートフォン端末のデータ復旧も課題として挙げている。これらのデータ復旧技術はいまだ確立した状態にないとのこと。「タブレット端末に搭載されているSSDは、HDDとはまったく異なったデータ記録方式を採用しているために、現段階では、まだ復旧技術が確立していない部分がある」「スマートフォンの場合はデータの暗号化の問題など、技術的な困難さが存在しており、いまだ研究開発が必要」というのがその理由だ。タブレット端末やスマートフォンのデータ復旧を行っている企業はあるが、業界のスタンダードサービスとして確立されているわけではない。今後、スタンダードサービスとして確立されたら、発表の機会を持ちたいという。

本発表の詳細な数字、レポート内容は同団体のWebサイトで確認できる。

CONTACT 日本データ復旧協会
URL http://www.draj.jp

Software

キヤノンITソリューションズ、「ESETセキュリティソフトウェアシリーズ」の新バージョンを販売開始

キヤノンITソリューションズ㈱は、「ESETセキュリティソフトウェアシリーズ」の新バージョンを、2013年12月12日より販売開始した。

Windows用プログラムの新バージョン「ESET Smart Security V7.0」「ESET NOD32アンチウイルス V7.0」では、次の3機能が追加された。

- **アドバンスドメモリスキャナー**
  高度に難読化、暗号化されたウイルスによる不審なプロセスの振る舞いを監視し、メモリ内でウイルスを解析する。これにより難読化や巧妙な手法で偽装されたウイルスの検出力が向上した
- **エクスプロイトブロッカー**
  アプリケーションの脆弱性を悪用する動作を監視し、疑わしい振る舞いを検出したら、動作をブロック。脆弱性を悪用して個人情報やFTPアカウントなどを盗もうとするウイルスに対して検出力が向上した
- **バルナラビリティシールド**（※「ESET Smart Security V7.0」のみ対応）
  脆弱性に対してネットワークレベルで機能する。これにより既知の脆弱性やセキュリティホールを悪用したウイルスによる攻撃、ハッキングによる外部からのネットワーク攻撃に対する防御力が向上した

製品のラインナップ概要は次のとおり。パッケージ版、ダウンロード版の有無や価格など詳細については同社Webサイトを参照のこと。

＜個人向け総合セキュリティソフト＞
- ESETファミリーセキュリティ
- ESETパーソナルセキュリティ

＜法人向け総合セキュリティソフト＞
- ESETオフィスセキュリティ 5PC+5モバイル
- ESETオフィスセキュリティ 1PC+1モバイル

＜法人向けウイルス・スパイウェア対策ソフト＞
- ESET NOD32アンチウイルス Windows/Mac対応
- ESET NOD32アンチウイルス Windows/Mac対応 5PC

CONTACT キヤノンITソリューションズ㈱
URL http://www.canon-its.co.jp

Software

エンバカデロ・テクノロジーズ、「C++Builder XE5」で新たにiOS向け開発をサポート

エンバカデロ･テクノロジーズは2013年12月10日、「C++Builder XE5」のアップデートのリリースを発表した。

2013年9月に販売開始した同製品は、Windows、Macのネイティブクロス開発を実現したビジュアル開発ツール。今回のアップデートで新たにiOS向け開発もサポートし、デスクトップPCからスマートフォン、タブレットに至る幅広いデバイスに対応できるようになった。具体的には、次のiOS開発機能が提供される。

- **iOSアプリのネイティブ開発**
  C++Builder ARM最適化コンパイラを搭載し、最高レベルのパフォーマンスを発揮するiOS向けネイティブアプリを構築できる
- **FireMonkeyアプリケーションプラットフォーム**
  iOS、Windows、Mac OS Xのマルチデバイスに対応した高性能アプリケーションを単一のコードベースで開発でき、大幅な工数削減が可能
- **モバイルフォームデザイナ**
  画面サイズの異なるiPhone、iPod touch、iPad向けのUIをドラッグ＆ドロップ操作で容易に設計できる
- **企業のバックエンドシステムに接続**
  企業のシステムとモバイルアプリをつなぐ技術を提供。DataSnapと呼ばれる多層技術をモバイルアプリから簡単に使える
- **高度な機能も柔軟に利用可能**
  ジャイロ、GPS、カメラ、加速度センサーといったデバイスサービスやセンサーを利用可能。高度なプログラミングを必要とする場合には、iOS APIへのアクセスも可能で、高い拡張性と柔軟性を提供する

登録ユーザは、アップデート版を2013年12月11日よりダウンロード可能となっている。C++Builder XE5 Enterprise以上、またはC++Builderを含むスイート製品RAD Studio XE5 Professional以上の購入者は無料でアップデートできる。

CONTACT エンバカデロ・テクノロジーズ
URL www.embarcadero.com/jp

Hardware

日本ヒューレット・パッカード、バックアップ／アーカイブのストレージラインアップを刷新

日本ヒューレット・パッカード㈱は、バックアップストレージ製品「HP StoreOnce」とアーカイブストレージ製品「HP StoreAll」のラインアップを刷新した。

HP StoreOnceは、同社独自の連携型重複排除エンジンを搭載し、シングルアーキテクチャで、アプリケーションサーバからバックアップサーバ、バックアップアプライアンスまで、適材適所でデータの重複排除を行える次世代バックアップストレージ製品。今回リリースされるのは、小規模環境やリモート拠点向け「HP StoreOnce 2700 Backup」、中規模環境向け「HP StoreOnce 4500 Backup」「HP StoreOnce 4700 Backup」、大規模環境およびデータセンター向け「HP StoreOnce 4900 Backup」「HP StoreOnce 6500 Backup」の全5モデル。2013年12月19日より販売開始している。

HP StoreAllは、オブジェクトファイルに対応したデータアーカイブ向けのストレージ製品。性能、機能を強化したゲートウェイタイプ「HP StoreAll 8200 Gateway」とアプライアンスタイプ「HP StoreAll 8800 Storage」の2モデルを提供する。2014年1月23日より販売開始される。

▼新製品のラインアップと価格

製品名	参考価格
HP StoreOnce 2700 Backup	1,260,000 円～
HP StoreOnce 4500 Backup	4,200,000 円
HP StoreOnce 4700 Backup	11,550,000 円
HP StoreOnce 4900 Backup	個別見積もり
HP StoreOnce 6500 Backup	個別見積もり
HP StoreAll 8200 Gateway	個別見積もり
HP StoreAll 8800 Storage	個別見積もり

▲HP StoreOnce 4500 Backup

CONTACT 日本ヒューレット・パッカード
URL http://www.hp.com/jp

Service

トレジャーデータ、新サービス「トレジャービューワー」と「トレジャークエリーアクセラレーター」を提供開始

米Treasure Data社、およびトレジャーデータ㈱は、「トレジャーデータサービス」の新サービスとして「トレジャービューワー」と「トレジャークエリーアクセラレーター」を追加した。

トレジャービューワーは、ブラウザ内でデータを簡単に可視化できるサービス。基本的なグラフなどであれば、他社から販売されている高度なBIツールを使わなくても、同サービスだけで可視化が行える。

また、トレジャークエリーアクセラレーターは、アドホックなクエリーの実行速度が従来の10～50倍に高速化されるオプションサービス。データ集計／分析をこれまで以上にすばやく行える。

さらに、これらの新機能の提供に合わせて、価格体系を一新した。これまでは、月額3,000ドル(約31万円)の「スタンダード」価格と、ユーザ企業ごとに価格を設定する「カスタム」価格の2種類だったが、月額7,500ドル(約77万円)の「プレミアム」価格が新たに加わった。

「プレミアム」価格でのサービス内容には、今回追加された新サービス「トレジャークエリーアクセラレーター」が含まれているほか、CPUは16コアと高性能で、データの保存量は年間あたり500億件となっている。

▼「トレジャーデータサービス」新価格体系

分類	価格／月	保存データ件数／年	その他
スターター	無料	20億件	・トレジャーデータのサービスの概要や簡単なプロトタイプ、基本的な利用方法を理解できる ・オンラインサポート付き
スタンダード	3,000ドル	150億件	・クエリーの実行数は無制限でCPUは8コア ・サポートが含まれる ・小中規模の企業向け。投資対効果を短期間で得られる
プレミアム	7,500ドル	500億件	・クエリーの実行数は無制限でCPUは16コア ・サポートが含まれる ・トレジャークエリーアクセラレーターが含まれる ・構築開始から14日間で導入可能
カスタム	個別対応		

CONTACT トレジャーデータ㈱
URL http://www.treasure-data.com

Service

IDCフロンティア、「ネイティブアプリケーションプラットフォーム for Gaming」を提供開始

㈱IDCフロンティアは2013年12月3日、クラウドサービスのラインアップに、新たにスマートフォンなどのモバイルアプリケーション向けプラットフォームサービスを追加し、提供することを発表した。

第1弾として、グルーヴノーツ社と提携し、ゲーム開発者向けに「ネイティブアプリケーションプラットフォーム for Gaming」を提供する。同社は今後、ゲーム業界を皮切りに、順次他分野のアプリケーションプラットフォームにも対応していく。

ネイティブアプリケーションプラットフォーム for Gaming

「ネイティブアプリケーションプラットフォーム for Gaming」は、グルーヴノーツ社が持つゲーム業界向けクラウドサービス「GSS Fairy」と「IDCフロンティアクラウドサービスセルフタイプ」「コンテンツデリバリーネットワーク(CDN)」「ネイティブアプリケーションに必要となるUIやAPI」「運用管理基盤」をプラットフォーム化したサービス。

iOSやAndroidのほか各種ゲーム機などにも対応可能で、スマートフォンと専用ゲーム機およびモバイルゲーム機とのゲーム相互利用を見据えたクロスデバイス開発の効率を高めることができる。

これにより高度なインフラ構築や運用の知識は不要になり、開発も容易になる。運用工数の大幅な削減も実現できるという。おもな機能は次のとおり。

- クロスデバイス認証機能(複数種類の端末で共通の認証が可能で独自実装不要)
- マルチタイトル管理機能(複数ゲーム間でのユーザ情報の共有)
- レポート機能(ユーザやゲームタイトル単位での利用状況把握)
- UI/APIとCDNおよびアプリケーション実行環境用仮想マシン

CONTACT ㈱IDCフロンティア
URL www.idcf.jp

Hardware

アライドテレシス、
マルチレイヤー・モジュール・スイッチ「AT-SwitchBlade x8106」をリリース

アライドテレシス㈱は、マルチレイヤー・モジュール・スイッチ「AT-SwitchBlade x8106」と予備用ファントレイ「AT-SBxFAN06」を12月16日より出荷開始した。

AT-SBx8106は、高さ4Uサイズ、コントロールファブリックカード用に2スロット、ラインカード用に4スロット、システム用電源を2スロット、PoE用電源を2スロット搭載した計6スロットタイプのシャーシ型スイッチ。販売中のAT-SBx8112と共通のラインカードを利用可能。ギガ、PoE、10ギガインターフェースなどさまざまな種類のラインカードがラインアップされているため、用途やネットワーク構成に合わせて必要なラインカードを選択できる。

価格はAT-SBx8106が828,000円。AT-SBxFAN06が98,000円。

▲AT-SwitchBlade x8106

CONTACT アライドテレシス㈱
URL http://www.allied-telesis.co.jp

Service

アシスト、
「Postgres Plus Advanced Server」新バージョン9.3の各種支援サービスを提供開始

㈱アシストは、「Postgres Plus Advanced Server」（開発元：EnterpriseDB Corporation／エンタープライズDB㈱）の新バージョン9.3に対応した各種支援サービスの提供を開始した。

Postgres Plus Advanced Serverは、PostgreSQLをベースにし、可用性向上やOracle Databaseとの互換性などの企業向け機能を付随したデータベースソフトウェア。新バージョンの9.3は、PostgreSQL 9.3をベースに開発されている。同社はこの製品のミッションクリティカルシステムへの導入を支援する2つのサービスを提供する。

- **パーティション設計サービス**
  検索性能、データメンテナンスの効率化を目的にパーティショニングを実装する。各種パーティショニング設計、データ移行、運用に関する支援を行う
- **データベース移行支援サービス**
  商用RDBMSやPostgreSQLから同製品への移行に関する計画策定および移行作業を実施する。移行計画の検討、利用機能の選定、アプリケーション移行、データ移行の支援を行う

CONTACT ㈱アシスト
URL http://www.ashisuto.co.jp

Software

ソースネクスト、
PCのブルーライトを軽減するソフトウェア「超ブルーライト削減」を発売

ソースネクスト㈱は、PCやスマートフォンの画面から出るブルーライトを軽減するソフトウェア「超ブルーライト削減」について、2013年11月のダウンロード版の発売に続き、2013年12月13日よりパッケージ版も販売開始した。

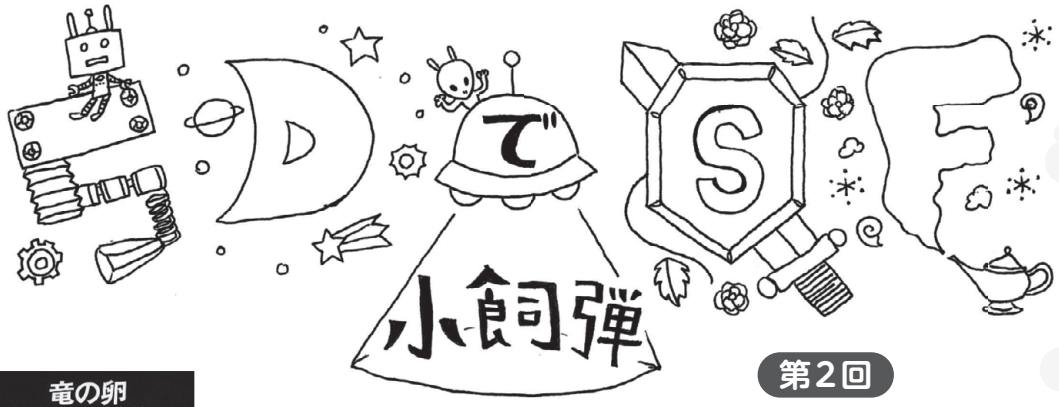
ブルーライトとはPCやスマートフォンから発せられる波長が400～500ナノメートルの青色光で、目の疲れや痛みの原因になると言われている。本製品は画面表示の色合いを調整することで、ブルーライトを抑えるソフトウェア。ISO取得企業が行った検証により、PC画面で約69%の削減効果が実証されている。

「ワンタッチでON/OFFの切り替えが可能」「スライドバーで濃度を調節できる」「起動時に自動でONになるよう設定できる」といった特徴を持つ。

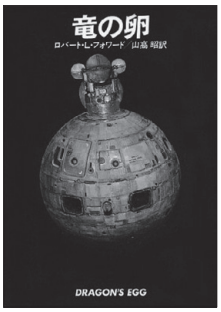
本製品1つで3台までのデバイスで使える。PCはWindows 8.1/8/7/Vista、スマートフォン／タブレットはAndroid 2.2～4.2で動作する。価格は2,980円（税別）。

ダウンロード版については、同社オンラインショップで、すでに3,500本を販売しているとのこと。

CONTACT ソースネクスト㈱
URL http://www.sourcenext.com



第2回



『竜の卵』（ロバートL.フォワード／ハヤカワ文庫）

SFというのは実に懐が深いジャンルで、「すべてのノンフィクションはSFに入る」と豪語している人も少なからずいます。それでも「SFっぽくない」作品と「SFとしかいいようがない作品」は確かにあって、後者の代表が「ハードSF」。SFという言葉が本来Science Fictionを意味するのであれば、徹底的に科学的考証にこだわったのがこのジャンル。

「科学的にありうる」以上、ある意味最も現実的なはずなのに、最も常識離れしているのがこれで、個人的には最も好きな分野です。

その中でも最高傑作なのが、『竜の卵』（ロバートL. フォワード／ハヤカワ文庫）。まず舞台がすごい。中性子星。超新星爆発した星の芯。その爆誕の瞬間をニュートリノで撮った小柴昌俊がノーベル賞を取ったことはみなさんご存じのはずですが、本作はその中性子星「竜の卵」で進化した知的生命体、いや、宇宙人チーラたちの歴史と、竜の卵を探索するべく派遣された人類の科学者たちとの交流の物語なのです。

中性子星のスペックを見れば、これがどれほどパネェか、いやほど伝わってきます。表面温度、8,000度。磁力、1億テスラ。そして表面重力、670億G。太陽より熱く、分子がちぎれるほどの高磁場で、そして体どころか原子がひしゃげるほどの重力。こんなところで「ヒト」以前に「生命」なんてありえるのでしょうか？　竜の卵と比べたら火星なんて地球の双子といえるほど、両者はかけはなれているのに。

生命活動に不可欠な化学反応は、電子に被われた原子同士ではなく原子核同士でやればいい。ただし反応速度は100万倍。世界のさしわたしが600分の1なら、体の大きさだって600分の1にすればいい。だいたい2〜3mmぐらい。これで計算すると、体重は60kgぐらい。なんてこったい。我々と同じぐらいじゃないか！

さすがに重力と磁力が強すぎて、同じ姿とはまではいきません。その艶姿はaicoさんのイラストで見ていただくとして、もうひとつ重要な違いがあります。彼らは我々の100万倍速で生きているのです。彼らの一生は我々の15分。「中国4000年」の歴史も、たった1日。そんな彼らと我々にコミュニケーションの成立する余地はあるのでしょうか……。

ぜひ読者自身の目でご確認を。1980年に書かれた本作は今や古典ですが、いまだ新品で手に入ります。さらに余裕があれば原著も。Kindleで簡単に手に入りますし、科学の言葉で書かれているだけあって技術書を読める皆さんなら読みこなせるはずですヨ。SD

題字／イラスト　by aico

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第2回 さてはコマンド使いだな！

既刊『Linuxシステム【実践】入門』が売れ筋のくつな先生の連載が読めるのはSDだけ!!

to be Continued

Linuxをどうやったら楽しく使えるかを考えると、使い始めた人にとって最初の障壁はやっぱりコマンドラインだと思うんですよ。そのまま考察を深めるとコマンドを覚えられないんじゃないかと。そこで、みんな大好き『ジョジョの奇妙な冒険』なんかで遊んでみるといいんじゃないかと提案します。コマンドも「能力」で覚えれば記憶に残るはず。「お前、新手のコマンド使いか！」って職場で話題になれば「この人はコマンドラインが使える人」って扱いにグレードアップ！——なワケないか……。

Letters from Readers

2014年はどんな年？

2013年はGNUや日本UNIXユーザ会が30周年、FreeBSDが20周年を迎えるなど、本誌でもお馴染みのOSSプロジェクトの節目の年でした。2014年はどんな年なのでしょう？　Wikipediaで「2014年」を調べたところ、大きなイベントとしてソチオリンピックやFIFAワールドカップが挙げられています。IT関連では「Windows XPのサポート期間終了」。重要なイベントですが、地味ですね。

2013年12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　SDN/OpenFlowで幸せになれますか?

2012年はまだ概念的な技術だったSDN/OpenFlow。2013年になってようやく機器やソフトウェアに実装され、ビジネスへの適用も進んできました。そんなSDNについて、現在の状況、利点／欠点、実用例などを整理してみました。

OpenDaylightやMininet、hostapdのコマンドやサンプルプログラムのおかげで、SDNの理解が深まりました。やはり手を動かさないとなかなか頭に入りませんね。

東京都／山下さん

ネットワークの仮想化は今後避けて通ることのできないテーマなので、今回の特集は社内のネットワーク仮想化を検討する際の参考になりました。

茨城県／ルビーさん

個人的にネットワークの分野に強い興味があり、今後のネットワーク構築、管理方法において大きな分岐点になり得るOpenFlowの話は、ぜひ読んでみたいと思っていた。

東京都／HL.Tさん

中小の社内SE向けな視点がほしい。

沖縄県／當眞さん

データセンターなどでの活用が期待されるSDNですが、第7章のOpenFlowによる無線LAN構築などは、一般的なオフィスでも使えそうな例です。以前と比べて、かなり身近な活用例が出てきましたね。

第2特集　エンジニアの伝わる図解術

2012年12月号の特集「なぜエンジニアは文章が下手なのか」に続く、文書作成術特集の第2弾です。今回は、図解のしかたにスポットを当ててお送りしました。

以前の「なぜエンジニアは文章が下手なのか」と組み合わせることで、すごくためになった。

神奈川県／kysさん

表現に関する技術はとても大切。これからも取り上げてください。

長崎県／JavaSplictさん

エンジニア以外にも使える。

長野県／高橋さん

エンジニアが業務で作る資料は、文章よりも図のほうが多いかもしれません。文章で説明するのが難しいときは、効果的に図解できるようになりたいものですね。

特別企画　LinuxとFreeBSDのファイルシステムの良い・悪いところをご存じですか?

OSの基本的な機能でありながら、案外知られていないファイルシステムについて、そのしくみを解説しました。

概要はわかっていても、実際の細かい検証まではたいていしない。それをしてくれるところがありがたい。

東京都／つんさん

最近のファイルシステムについて理解していなかったので、本記事は勉強になりました。自分の知識は古かったために、「FreeBSDに比べてLinuxのファイルシステムは脆弱」と思い込んでいましたが、今はそんなことないのですね。ZFSがLinuxで安定して使えることには期待したいです。

埼玉県／犬棟梁さん

オープンソースのOSを使うなら、用途に合わせてファイルシステムそのものや、設定を変えて、チューニングできるようになりたいですね。その参考になったのではないでしょうか。

一般記事　iOS 7アプリの魅力はどうやって引き出すのか

2013年9月にiOS 7がリリースされました。デザインは一新され、新たなフ

レームワークやAPIが追加されました。記事では、その中から代表的な変更点をピックアップして解説しました。

iOSの新たなおもしろさを見た気がします。また新しい情報があれば紹介してください。

愛知県／榎本さん

いつかはアプリを開発して、お小遣いを稼いでみたいです。

大阪府／山崎さん

多くの人に使われる魅力的なアプリを開発するには、常にiOSの最新バージョンに追随していく必要があります。iOSアプリ開発者の方は、本記事をきっかけにiOS 7に本格的に取り組んでみてはいかがでしょうか?

一般記事 「Mirama」a.k.a VIKING

メガネ型情報機器「Mirama」とその専用OS「Mirama OS」を紹介しました。開発担当者へ直接取材し、Mirama OSに採用されている技術や今後の開発の方向性について聞きました。

便利だとは思うけど、使用環境について熟考が必要だと思う。

東京都／戸澤さん

未来を感じさせる記事でした。

神奈川県／眞　泰志さん

実社会への浸透はまだまだ難しいと思うが、新技術の紹介として興味深かった。

北海道／村橋さん

Miramaも今はゴツゴツした試作機ですが、やがてはスマートなメガネの形状になると思うと、早く完成形が見てみたい気がします。

フリートーク

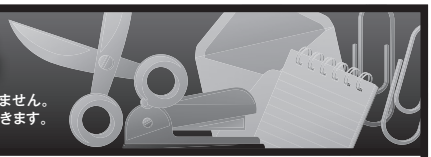
Raspberry Piにはまっています。ほかの人はどんな使い方をしているか興味があります。

富山県／jukさん

Raspberry Pi、人気ですね。弊社から『Raspberry Pi［実用］入門』という書籍も出ていますので参考にしてみてください。また、今号の読者プレゼントは限定版Raspberry Piです。ぜひご応募ください。

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。



ノートPCタブレットデュアルアームDN-10313

6,999円（税込）／上海問屋　http://www.donya.jp/

狭いデスクで複数の端末を使うときは、本製品のようなアームで立体的に端末を設置すると、デスクのスペースを有効活用できます。本アームはノートPCとタブレット端末を同時に取り付けられます。写真1の例では、メインで使用するMacBook Proはデスク上に置き、それとはべつに本アームを使ってiPadとMacBookを設置しました。ノートPCをアームの一番低い位置で取り付けても、それなりの高さがあるので、アームに設置するPCはタイピングよりも、ブラウザやSNSのタイムラインを常時表示させるなど、「見る」ことを主にした使い方をするほうが効果的です。1台のPCでいくつものソフトウェアを起動させるくらいなら、表示させているだけのソフトは空いているタブレットなどに任せましょう。メインPCは動作が軽くなり、快適に作業できますよ。（読者プレゼントあります。P.16参照）

▲写真1 アームにMacBookとiPadを設置

祝 12月号のプレゼント当選者は、次の皆さまです

①My Book 2TB ..... 愛知県　川上　拓真様
②ウイルスバスタークラウド1年版 ..... 富山県　小林久壽雄様

★その他のプレゼントの当選者の発表は、発送をもって代えさせていただきます。ご了承ください。

次号予告

Software Design
March 2014

2014年3月号
定価1,280円　176ページ

2月18日発売

［第1特集］データベースの諸問題

RDBとNoSQLどちらを選びますか?

真っ当に考えるDBの鉄則

「NoSQLが流行っているから、これでいこう」そんな軽くていいのですか?
データベースは企業の実績を支える重要な資産です。リレーショナルデータベースを本当に活用していますか?　あなたに本当に必要なデータベースは何ですか?
本特集では、データベースについてRDBかNoSQLか根底から問いかけつつ、エンジニアの軸となる知識と技術を整理し紹介します。

［第2特集］ネットワークエンジニアのための

プロキシーサーバの教科書

基本機能からリバースプロキシーまで構築+運用マニュアル

単なるプロキシーから、リバースプロキシーまで実際の構築方法と運用の基本を押さえ、クラウド・オンプレミスの混在環境での利用まで事例をもとに解説します。

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2014年1月号　連載「IPv6化の道も一歩から」最終回
●P.125　左段
［誤］金沢星陵大学　　［正］金沢星稜大学

SD Staff Room

●『過負荷に耐えるWebの作り方』は2年越しの企画。気鋭のイラストレータaicoさんと、バイブドビッツのエンジニアとのコラボレーション。先月号を作っているときが山場で、同時にもう1冊作っていて、そのままの流れで年末進行にはいったら風邪を引きました。抗生物質投与もいまだ治らず。（本）

●幕張メッセ近辺で呑んだ後、ぶらぶらしていたら外人に「マンハッタンどこですか?」と英語で聞かれた。そんな名前のホテルがあったなと思ったが簡単に説明できず、iPad mini retinaをポケットから取り出し、Google Mapで見せた。「ありがと」って日本語で言われた。みおぼんONでよかった。（幕）

●昨年秋スタートのアニメでは『ガンビル』（略したほうが恥ずかしい?）を親子で見ています。ガンダム世代の私はセリフやマニアックなMSに過剰に反応してしまうわけですが、そんな予備知識のない子供たちにも好評。お正月はガンプラ作りで、父の技術を子に伝承するつもりです（ろくでもないね☆）。（キ）

●最近、購入した「あったか敷パット」がとても良い。夜寝るときに、敷布団の上にそれを敷いて寝るわけですが、さらに湯たんぽを入れると最高にあたたかい。だめだ……、思い出しただけで眠ってしまう。昨年は電気毛布を使っていましたが、今年は不要ですね。これで原発も止めて大丈夫だな。（よし）

●先日、部屋の壁紙の張り替えをしてもらいました。壁紙を100種類近くから選ぶことができ、キッチンは私の自由にしていいとのことだったので、明るい雰囲気の花柄模様に。と同時にキッチンの掃除にも入ってもらったのでピカピカ。ようやくキッチン雑貨も置けるようになり、どれを買うか悩み中です。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2014年2月号

発行日
2014年2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。