Special Feature 01
きっかけはオブジェクト指向
Special Feature 02
知っ得UNIXコマンド
2013 June
06
[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

2013年6月18日発行
毎月1回18日発行
通巻338号
(発刊272号)
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
1,280円

Special Feature 01

ちゃんとオブジェクト指向できていますか?

Java、JavaScript、PHP、Perl、オブジェクト指向プログラミングを実践

わかった人だけメキメキ上達

いつ覚えるか?
今でしょ!

Special Feature 02

あなたの知らないUNIXコマンドの使い方

サーバ、ネットワーク、プロ技の基本マスター

Extra Feature 01

リアルタイム分散処理「Storm」

Extra Feature 02

HiveでHadoopを活用してみませんか?

新連載!

結城浩「再発見の発想法」
伊勢幸一「短期集中講座・仮想ネットワークの落とし穴」

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54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。

オープンクラウド

普及が進むオープンクラウド

現在のITシステムを支える要素のひとつとしてクラウドサービスは欠かせない存在となっていますが、その一方で特定ベンダーのクラウド基盤に依存してしまう「ベンダーロックイン」の問題が取り沙汰されています。このベンダーロックインを回避するために注目を集めているのが「オープンクラウド」です。

オープンクラウドというのは、オープンソースの技術をベースとして構成されるクラウド基盤のためのソフトウェアや技術を指す総称です。オープンクラウドの具体的な定義は、これを推進する団体ごとに少しずつ異なりますが、共通する方向性としては次のような要件を挙げることができます。

- オープンなインターフェースによって連携できる
- データやメタデータがオープンなフォーマットによって表わされる
- オープンなプラットフォームの上に構成される
- ユーザは自由にデータの移植やアクセスを行うことができる
- 技術の開発や策定に参加できるオープンなコミュニティを持つ

レイヤ横断で進むクラウドのオープン化

昨今のオープンクラウドを巡る動向として特徴的なのは、IaaSやPaaS、ネットワーク、データセンターなど、レイヤ横断的にその普及が進んでいるという点です。たとえばIaaSレイヤでは、オープンなOSや仮想化技術やストレージ管理技術などを利用して、任意のスペックの仮想マシンやストレージ領域を提供できる基盤ソフトウェア群の開発が進められています。PaaSレイヤでは、業界標準に準拠した複数の開発言語や開発フレームワークをサポートするオープンソースの基盤ソフトウェアが提供されています。ネットワークレイヤで注目されているのは、ネットワーク構成をソフトウェアによって定義・変更できるSDN（Software Defined Networking）技術です。そのほかにも、データセンターの仕様やノウハウをオープンソース化することで高効率なデータセンターの構築を目指すオープンデータセンターと呼ばれる試みなども実施されています。

これらのオープンクラウド基盤は、オープンソースであるために独自のカスタマイズが可能なことや、レイヤをまたいで自由に組み合わせて利用できること、組み合わせのパターンによって他のクラウドシステムとの差別化が図りやすいことなどが大きな強みと言えます。

主要なオープンクラウド基盤

ここでは、オープンクラウドのための主要なIaaS基盤ソフトウェアを2つ紹介します。

● OpenStack

OpenStack Foundationによって開発されているIaaS基盤ソフトウェア。仮想ハイパーバイザ経由で計算ノードを管理するNova、クラウドストレージのSwift、ブロックデバイスストレージのCinder、仮想ネットワーク機能を提供するQuantum、認証管理機構を提供するKeystone、仮想マシンイメージをストレージに格納するGlance、管理UIのHorizon（名称はいずれもコード名）などといったコアプロジェクトから構成される。AWS互換のAPIによって他のパブリッククラウドと組み合わせたハイブリッドクラウド環境を構築することもできる。また、Quantumはさまざまな仮想ネットワークスイッチ製品向けのプラグインが多数用意されている点が大きな特徴となっている。

● Apache CloudStack

現在はApache Software Foundationに寄贈されて開発が進められているIaaS基盤ソフトウェア。仮想マシンのプロビジョニングやリソース管理などの管理機能を提供するManagement Server、CPUやメモリなどのリソースを提供するComputing Node、仮想ディスク領域を提供するPrimary Storage、仮想マシンのイメージやスナップショットを格納するSecondary Storageなどのコンポーネントから構成される。複数ハイパーバイザーを同時稼働させられるほか、仮想ネットワーク技術によってネットワーク構成を柔軟に設定することが可能。SD

OpenStack
http://www.openstack.org/
Apache CloudStack
http://cloudstack.apache.org/

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

Software Design Jun. 2013

Contents

1

第1特集 Special Feature 01

わかった人だけメキメキ上達

ちゃんとオブジェクト指向できていますか?

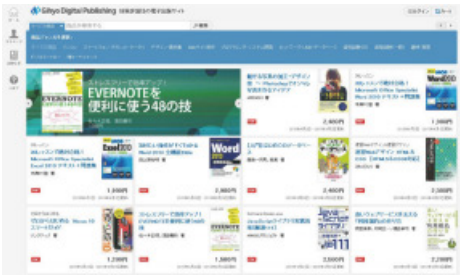
技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

http://gihyo.jp/dp



GIHYO Digital Publishing 未来の"普通"を、今。

Google play

楽天 Rakuten KC　kobo

※販売書店は今後も増える予定です。

法人などまとめてのご購入については
別途お問い合わせください。

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

ソフトウェア・デザイン［目次］

Contents

Software Design Jun. 2013

2

第2特集 Special Feature 02

研修じゃ教えてもらえない!?

あなたの知らないUNIXコマンドの使い方

一般記事 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 Jun. 2013

Column

Development

OS/Network

Inside View

Logo Design ロゴデザイン > デザイン集合ゼブラ+坂井 哲也

Cover Design 表紙デザイン > Re:D

Cover Photo 表紙写真 > Life On White/Getty Images

Illustration イラスト > フクモトミホ、髙野 涼香

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。

なぜ、あなたはJavaでオブジェクト指向開発ができないのか

Javaの壁を克服する実践トレーニング

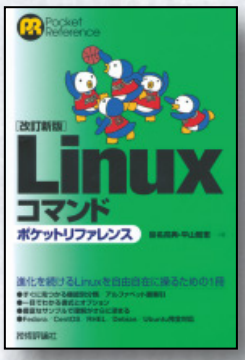
Javaプログラミング言語習得において、新人プログラマーの最初の障害は「オブジェクト指向の壁」です。本書は、単にプログラミングを「覚える」のではなく、その「考え方を理解」させ、さらに「手を動かしソースを書く」ことを基本とし、知らず知らずのうちにオブジェクト指向が身に付くようになっていきます。

こんな方におすすめ

- オブジェクト指向開発の進め方がよくわからない方
- Javaを使えてはいるが、オブジェクト指向での開発に自信がない方

小森裕介　著／A5判／296ページ
定価2,289円（本体2,180円）
ISBN 4-7741-2222-X

[改訂新版] Linuxコマンドポケットリファレンス



Linux操作に必要なコマンドを収録したポケットリファレンスです。機能別に分類し、コマンドの詳細がパッとわかります。また、コマンド名のアルファベット順索引も用意しています。おもなディストリビューションに対応し、初心者から上級者まで決定版の1冊です！

沓名亮典、平山智恵　著／四六判／576ページ
定価2,394円（本体2,280円）
ISBN 978-4-7741-3816-9

[改訂第4版] UNIXコマンドポケットリファレンス

初版より増刷を重ね12年目に突入するポケットリファレンスシリーズの定番本が最新のUNIX環境にあわせ増補改訂となりました。
初心者からベテランまでぱっと引けてすぐわかる機能引きリファレンスです。

石田つばさ　著／四六判／448ページ
定価2,289円（本体2,180円）
ISBN 978-4-7741-4836-6

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

元マイクロソフト社長

「早くも2013年成毛眞のおすすめ本No.1が登場してしまった。」 by 成毛眞(HONZ代表)

鳥類学者 無謀にも 恐竜を語る

鳥は恐竜から進化した。
Tレックスやアパトサウルスを生み出した恐竜は、紆余曲折を経て、今を生きるかわいい鳥なったというわけだ。
だとすると……鳥類学者は恐竜学者とも言えるのではないか？そのへんをパタパタ飛んでいる鳥を観察すれば、太古の恐竜のアレコレがわかってしまうのではないか！？
これは実際に、鳥類学者に恐竜を語って頂くしかない。
そんな難問に挑む、一人の鳥類学者。ユニークな視点と大胆な発想をもとに、恐竜の息吹を蘇らせる。
無謀とも言える挑戦で見えてきた恐竜の姿とは、一体どんなものなのだろうか？

川上和人　著／A5判／272ページ
定価1,974円（本体1,880円）
ISBN 978-4-7741-5565-4

栗本慎一郎が後世に贈る「最後の一冊」

栗本慎一郎の 全世界史

「パンツをはいたサル」以来、幅広く活躍してきた栗本慎一郎氏がライフワークにしてきた世界史の再構築。栗本氏は本書を事実上の遺作として、本気で世に問う最後の作品として位置づけ、出版する。
西欧と中国の偏った史観に依拠してきた従来の日本の歴史学を一掃させることで浮かび上がってくる真実の歴史像を、全ユーラシアの生きた歴史、新しい歴史の教科書として世に問う一冊になる。
前著「ゆがめられた地球文明の歴史」で展開した歴史論をさらに拡大発展させ、読者の要望の多かった日本史についても幅広く著述している。栗本氏が「意味と生命」以来展開させてきた、独自の生命論についても披瀝。文字通り、集大成となる一冊。

栗本慎一郎　著／四六判／256ページ
定価1,659円（本体1,580円）
ISBN 978-4-7741-5639-2

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

ネットワークエンジニア虎の穴

『自宅ラックのススメ』

文／ tomocha（http://tomocha.net/diary/）

第2回 ラックの選び方

イラスト：高野涼香

はじめに

今回は連載2回目。前回は、気づいたら自宅にラックを設置していて、こんなことやってましたというお話でしたが、今回は、ラック選び、設置についてお話しさせていただきます。

ラック選び——EIAとJIS規格の違い

サーバラックにはEIA（米国電子工業会）規格と、JIS（日本工業）規格の2種類がありますが、ラックマウント型のサーバやネットワーク機器はEIA規格となり、EIA規格のものでもたくさんの種類のサーバラックが存在します。では、どういうものを選べば良いのでしょうか。

前回のおさらいになりますが、サーバラックに搭載する機器は、U（ユニット）で表現され、1Uあたり、高さが44.45mm、横幅が19インチで、482.6mmとなります。すなわち、20Uのラックであれば、44.45mm×20Uとなり、高さだけで、889mmとなります。約89cm弱あるわけです。フルサイズ（42U）になると、1866mm（約1.86m）もあるわけですね。これにフレーム分の高さや、土台、足の部分などが入ると、さらに大きくなります。寸法について、ざっくりとイメージがついたら、実際に設置するにあたり、部屋に入るか検討する必要があります。

一般家庭の場合、扉の寸法は、幅600mm～800mm、高さ1800mm程度のことが多く、うっかり選んでしまうと、ラックが扉より大きくなり、部屋に搬入ができなくなってしまいます。また、扉の方向や戸当たり、障害物（下駄箱や通路の曲がり角）の分も考慮する必要が出てきます。もし部屋にラックを設置する前提で、引っ越しを考えているのであれば、搬入ルートや動線を考えておく必要があります。多くの場合、ハーフラックであれば、高さの問題はクリアでき、横幅が、600mm程度のものが多いので、ポイントさえ押さえておれば、比較的容易に設置できるでしょう。

▼写真1　ケージナット対応のラックの穴

▼写真2　ネジ穴式のラック

どんなラックを選ぶべきか ——ネジ選びがポイント

物理的に設置できるサイズがわかったら、どういうラックが良いか、悩む番がやってきます。ネジの取り付け穴方法が重要となり、ケージナット対応の四角い穴が空いたラック（写真1）と、ネジ穴式のラック（写真2）と2種類あります。サーバを搭載する場合は、なるべくケージナット対応がお勧めです。ツールレス（工具不要）のサーバ用のレールなどは、ケージナット用の穴にはめる形（引っかけてはめるタイプ）なので、簡単に装着ができますし、取り外しも容易です。また、ケージナットとネジの2つを準備し、ネジ穴に固定することもできるため、ネジ穴がつぶれても、ケージナットを交換することにより対応ができます。ネジ（化粧ビス）のサイズもいくつか種類がありますが、ケージナットを交換すれば自由自在です。

とくに中古などで探すときは、値段を基準に選ぶと、ネジ穴式のものしか在庫がないというケースもあると思いますが、このような場合は、ネジのサイズに注意が必要です。ネジにも規格、サイズが存在しており、M5やM6と言われ、ネジの径（太さ）が違います。M5よりM6のほうがネジが太く、重量のある機器などは、M6ネジが多かったりします（写真3）。ここはラックしだいですので、ネジを購入する場合は気をつけましょう。

そのほか、ポスト（ケージナットやネジの取り付けの支柱）とフレームの間に隙間があり、ケーブルを通すスペースがあるか、ないかというのも非常に重要なポイントになってきます。スペースが存在する物ほど横幅が大きくなるため、考慮すると、選定が難しくなりますが、可能な限りある物を選んだほうが良いでしょう（写真4）。

ラック設置のポイント —耐荷重とは

次に、重要なことは、床の耐荷重を確認することです。一般的な住宅の場合、1平米あたり180kgまで耐えられる設計になっていますが、1点に荷重がかかりすぎると床などが抜ける場合がありますので、注意が必要です。

また、荷重を分散させるように、コンパネを敷くなど工夫が必要です。ハーフサイズのサーバラックの重量が60kg程度あれば、実際に利用可能な重量は120kg程度となり、1Uあたりで使用可能な機器の重量は、約6kg程度となります。このあたりはラックの重量とユニット数、耐荷重との兼ね合いです。また、耐荷重は部屋全体の平均で考えることが多く、梁（はり）の近くや、壁側であれば、実際にはもう少し耐荷重が高かったりすることもありますし、部屋の真ん中であれば、たわんでしまったりして、実際には低かったりしますので、なるべく壁際や、梁の近くがお勧めです。このあたりはしっかりと不動産屋などに確認することをお勧めします（耐荷重って……？　と言われることも多いですけど）。

耐荷重の次に重要なのは、地震対策。本来はラックはボルトで固定するのが望ましいですが、ご家庭では難しいため、どのようにして倒れないようにするかが大きな課題です。サーバラックは重たいので、転倒防止のグッズではおそらく役に立たないでしょう。天井を突き破る可能性があります（ないよりマシですが）。筆者のように、倒れるスペースを作らないのも1つの手ですが、なかなか難しいと思います。なるべく壁際に設置し、全高の低いラックを使用して、重量物の機器はなるべく下に積むなど工夫をして、転倒しにくい環境を作りましょう。SD

▼写真3　ケージナットとネジ

▼写真4　ポストスペースなし

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 174 >>>>>

究極のウェアラブルは腕時計?

身につける デジタル=ウェアラブル

皆さんが普段、身につけているものといったら何でしょう? お財布や携帯電話は常に持ち歩いているかもしれませんが、"身につける"のとはまた違います。常に身につけるということであれば、指輪、メガネ、腕時計、ブレスレットなどでしょうか。人によっては入れ歯とか、カツラとか、耳に挟んだ赤ペンとかもあるかもしれませんね。身につける——身体の一部となることで、持ち歩くよりも一段と身体との関係性が親密になります。今回は腕時計とウェアラブルコンピューティングに注目してみたいと思います。

腕時計はおもに利き腕と反対側にすると言われていますが人それぞれです。女性であれば、その所作が美しくなるよう、腕の内側に盤面が向くようにつけている人もいます。最近では腕時計を身につけず、携帯電話の時計で済ませている人もいるかもしれません。しかし、腕時計であれば時刻表示を身につけているのと同じですが、時刻表示ができるデバイスを持ち歩いていて、時刻を知りたいときに操作して見ることの間には大きな差があります。時間を知りたいとき、腕時計であればさっと袖をめくって盤面で時間がわかります。Tシャツなど、腕がむき出しの場合であればただ腕時計に視線を向かわせるだけです。携帯電話やスマートフォンでも時刻がわかるとはいっても、動作の数が大きく違うことから明らかでしょう。

また腕時計には「時を知る」使い方のほかにも、ファッションとしての要素、嗜好品としての要素、そしてまた

ディック・トレイシーに出てくる通信機のおもちゃの広告
http://blog.modernmechanix.com/dick-tracy-wrist-radio/

UP

Nike+ FuelBand

Pebble

※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。

身体から取り外されたときには収集物(コレクション)や資産としての意味合いを持ってきます。機能や役目だけではなく、ブランドや装飾品としての価値が重視される場合もあり、その種類の多さ、製造メーカーの多さ、100円ショップで買えるものから家が買えるくらいの価格の幅を考えると、まだまだデジタルデバイスでは到達し得ない深みと広がりを持っていることがわかります。

ウェアラブルの定義

ウェアラブルコンピューティングのウェアラブルは"wear-able"、身につけることが可能という意味です。身につけて(wear)利用するコンピュータ関連機器、つまりヘッドマウントディスプレイや、衣類やアクセサリなど身につけるものと統合されたデジタルデバイスを示しています。

腕時計が単なる時を知る機器から、ウェアラブルデバイスに変化したのはいつごろのことでしょう? 単なるデジタル腕時計から一歩進んだ、スマートウォッチの誕生は1976年、タッチで表示が切り替えられるORIENT Touchtronにさかのぼります。1980年代にはボールペンの先で操作するのかと思えるほど小さなボタンが並んだ電卓付きの時計、1998年には腕時計型のコンピュータ「Ruputer」、1999年の「SWATCH .beat」という時計では全世界で同じ時間帯を利用するインターネットタイムが提唱されました(http://www.swatch.com/jp_ja/internettime.html)。2000年代には、Microsoft SPOT Watchが登場しました(現在は提供終了)。

現在Microsoftの研究所に所属するユーザインターフェースの大家Bill Buxtonによると、1976年以来、今年になるまで37年間、スマートウォッチとしてさまざまな試みや製品が生まれては消えていったとのこと。ものすごく古い例としては、1960年代のコミックヒーロー「ディック・トレイシー」で使われていた双方向通信機も腕時計型でした。そのほかにも、007をはじめスパイの秘密道具として、日本では特撮戦隊ものの通信装置として、腕時計型デバイスは身近で未来的なデバイスとして定番のスタイルでした。

腕時計型デバイスのサービス視点

腕時計型ウェアラブルデバイスには、スマートフォンやその他のデバイスにはない、いくつも利点と課題が含まれています。

» 日常的に肌に触れている。音に限らず振動でも合図を知ることができる

» バッテリーの持ちへの高い要求。充電なしでも1週間程度の動作が求められる

» サブデバイスとしての位置づけ。表示画面が小さく、操作もしづらいため本体では単純な操作のみ。何かと連携したり、ある機能だけ切り離して使えるのが理想

» 生活防水や素材としての耐性が求められる。汗や雨、直射日光、振動など過酷な状況で利用される

機能をつめこんだスイスアーミーナイフのような万能で便利なデバイスから、ある特定の単機能のみを分離してシンプルに使いやすくしたもののほうが、身につけるデバイスとしては向いています。日本では忙しいビジネスマン以外、利用している人を見かけませんが、スマートフォン用のBluetoothヘッドセットもそういう機能分離型で便利に使えるのデバイスの1つです。

Appleからも「iWatch」なるスマートウォッチが出るとか出ないとか、まったくの噂の領域を出ませんが、iPod nanoを腕時計として使えるバンド以上の何かが期待されます。Googleも特許として提出されたアイデアの図柄から、時計的なデバイスをまったく考えていないわけではないことがうかがえます。

もともと身体の一部でないものを、まるで一部であるかのように身につけ

Ruputer

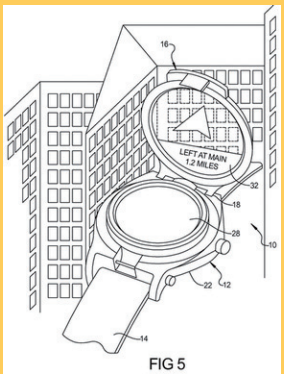

.beat


WRISTOMO

るためには、重さや大きさ、形状、質感や材質、丈夫さや電池の持ち、充電方法も重要な要素です。加えて社会的になじむまでにはいろいろな課題が残っています。腕時計型のデジタルデバイスが数多く出現するのも、そういった課題を乗り越え、受け入れられやすいと考えられているためでしょう。数年のうちに「Google Glass」や「Telepathy」といった頭に装着するウェアラブルデバイスが一般化するかもしれませんが、それまでは腕時計やメガネ以上の特殊な機器を身につけても奇異な視線で見られてしまうでしょう。

続々と登場する腕時計型デバイス

運動する人にも、普段あまり運動しない人にも、体を動かすことを推奨する「Nike+ FuelBand」、さまざまな身体情報を記録するJawboneの「UP」や、歩けば歩くほど企業が寄付してくれる「Striiv」、最適な睡眠サイクルを記録・推奨する「fitbit」、フィットネス機器「MOTOACTV」、Kickstarterで大量の資金を集めた有機EL画面搭載の「Pebble」と、いったい腕が何本あればいいのでしょう？　皆さんもお気に入りのものや、気分によってさまざまなデジタルデバイスを身につけ替えるスタイルを楽しんでください！ SD



Googleが提出した特許「US8279716 B1」番に掲載の図柄
※引用元　http://www.google.com/patents/US8279716

gadget 1

Eco-Drive Proximity

http://www.citizenwatch.com/en-ir/country-gate/

iPhoneと連携するアナログ腕時計

見た目はアナログな針時計ですが、iPhoneと連携してメールや電話が着信するとわずかに腕時計が振動して知らせてくれます。もちろんこの時計自身でメールを読み書きしたり通話はできませんが、光で充電しながら使い続けられる腕時計です。またiPhoneとの距離が離れ、接続が切れたときも振動して知らせてくれるので、置き忘れ防止にも役立ちます。同じようなデジタル時計にはCOOKOO watchやG-SHOCK Bluetoothがあります。

gadget 2

MYO

https://getmyo.com/

ジェスチャコントロール用のアームバンド

MYOは腕に装着し、腕や手の筋肉の動きを読み取るジェスチャーコントロール用のデバイスです。手の握り方や腕のねじり方で、マウスの代わりやゲームコントローラの代わりになります。料理中などで、デジタルデバイスに直接触れないときなどにも活用できます。

gadget 3

Shine

http://www.misfitwearables.com/

超小型デジタル歩数計

一見、囲碁の石のような硬貨大のShineは、スマートフォンと連携する画面のないデジタル歩数計です。Misfit Wearables社から99ドルで発売予定で、衣服や腕時計、ペンダントなど好きなところに装着できるアクセサリのようなデバイスです。バッテリーは半年持ち、ユーザが自分で交換できます。

gadget 4

LinkMe

http://www.kickstarter.com/projects/540550394/linkme-wrist-billboard

スマートメッセージ組み込みリストバンド

LinkMeはスマートフォンと連携して、LEDでメッセージを表示できる腕時計型デバイスです。クロームメッキのリング状のデバイスにはドット表示用LEDが搭載されており、SNSのメッセージ、リマインダ、アラートなどを表示できます。原稿執筆時（2013年4月上旬）Kickstarterで資金募集中です。無骨な電子機器ではなく、アクセサリのような輝く質感を持ったデザインに注目が集まっています。

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

新連載

結城浩の再発見の発想法

Buffer

はじめにひとこと

こんにちは、結城浩です。「再発見の発想法」という連載を始めます。ここでは、技術用語を1つずつ取り上げ、その背後にある発想法を探りましょう。今回は「Buffer（バッファ）」です。

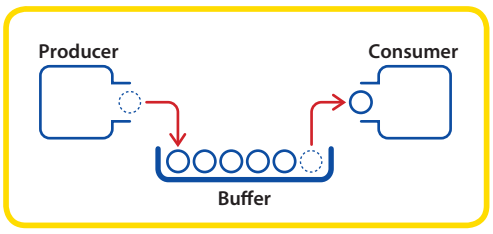
Buffer—バッファ

バッファとは

病院には**待合室**というものがあります。病院に行ってもすぐに診察してもらえるとは限りません。たいていは、待合室で自分の番を待つことになります。混んでいれば長く待たされますし、空いていればすぐに診察されるでしょう……これは誰しも病院で経験していますね。

バッファ（buffer）とは、2つのプロセスの間に置く領域で、2つのプロセスの処理スピードの差をやわらげるためにあるもののことです。たとえば、待合室は、患者の到着スピードと、医者の診察スピードの違いをやわらげるバッファです。2つのプロセスとバッファの関係は、図1の模式図で表せます。

▼図1　プロセスとバッファの関係



左のProducer（プロデューサ、生産者）がデータを生産し、右のConsumer（コンシューマ、消費者）がデータを消費します。プロデューサとコンシューマという2つのプロセスの間にあるのがバッファです。生産スピードが消費スピードよりも速いときにはバッファ中に貯まるデータは多くなります。逆に、生産スピードが消費スピードよりも遅いときには、バッファ中に貯まるデータは少なくなります。

バッファは、2つのプロセスの処理スピードをやわらげる（緩衝する、緩和する）ものです。バッファというと頭痛薬の「バファリン」を思い出しますが、あれは「緩和する（buffer）」をもとにして作られた名前だそうです。

バッファを理解するには、バッファがない場合に起きることを考えればいいでしょう。もしも病院の待合室がなかったら、患者は診察時刻ジャストに来院しなければなりません。早く来すぎたら（待合室がないので）一度帰宅しなければなりませんし、遅く来たら今度は医者のほうが待たされて時間の無駄が生じます。

病院の待合室がバッファの役割を果たすので、患者が到着する時刻がばらついても、診察にかかる時間が変化しても、患者・医者双方の負担や無駄を減らせます。つまり、バッファは2つのプロセスの処理スピードにバラツキがあっても無駄を防ぐ効果があるのです。

バッファのエラー

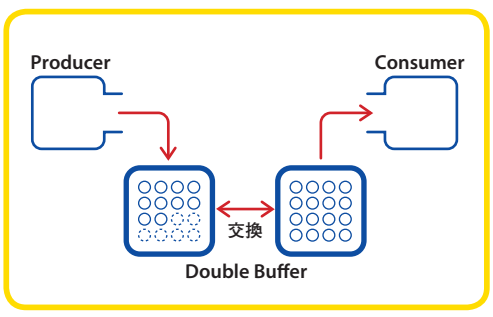
バッファも万能ではありません。医者が診察できる人数を超えて患者数が多くなると、待合室が溢れてしまうでしょう。いわゆるバッファの**オーバーフロー**です。オーバーフローが起きないようにするためには、バッファの大きさを十分大きくしておくか、生産者側を一時停止する措置が必要になりますね。

IT技術におけるバッファ

IT技術ではあらゆる個所にバッファが存在します。最も典型的なのはプリンタのバッファでしょう。プリンタのバッファは、コンピュータの処理スピードとプリンタの印字スピードの差異を緩和するものです。プリンタに限らず、マウスやキーボード、ネットワークなど、多くのデバイスに対してバッファが存在します。

コンピュータグラフィクスの分野では**ダブルバッファリング**という技法が一般的です。これは描画する画像を作り出す描画プロセス(Producer)と、表示プロセス(Consumer)のスピード差を緩和し「ちらつき」を防ぐ方法です。ダブルバッファリングでは、描画するデータを保持するバッファを2つ用意します。1つは描画プロセスの書き込み用で、もう1つは表示プロセスの読み出し用です。描画プロセスが1画面分を描いたタイミングで2つのバッファをカチッと交換します。これで、画面のちらつきを防げるのです(**図2**)。

▼図2　ダブルバッファリングのしくみ



日常生活でのバッファ

「2つのプロセスの処理スピードの差異をやわらげるもの」という視点を手に入れると、日常生活のあちこちでバッファが見つかります。

たとえば、財布もバッファです。お金が必要なときに毎回銀行から預金をおろしていたら手間がかかります。いったん銀行から財布にお金を移し、細かい支払は財布から行うのです。処理スピードの差異をやわらげるとともに、処理コスト(手間)の差異をやわらげています。

蓄積が価値を生む

さて、バッファは2つのプロセスの処理スピードの差異をやわらげますが、途中にある蓄積が価値を生む場合があります。処理スピードの差異が作る時間差が価値を生み出すのです。

プリペイドカードを考えてみましょう。図書カードであれ国際電話カードであれ、プリペイドカードは、利用者が運営者にお金を支払ってから、実際に使用するまでに時間差があります(支払→使用の順)。一人一人の金額は少なくても、多数の利用者がいれば、運営者に多額のお金がバッファされるでしょう。運営者はそのお金を時間差の許す限り運用できます。

クレジットカードでは、使用→支払の順になり、プリペイドカードの支払→使用とは逆です。クレジットカードでは、支払日になったらバッファされていた支払が一気に引き落とされます。クレジットカードは、購買スピードと入金スピードの差異を緩和させるバッファです。

と考えると、クレジットカードの**限度額オーバー**は、バッファのオーバーフローであるとわかります。購買スピードが入金スピードをオーバーしたエラーということです。このエラーを避けるには、信用限度額を増やす(バッファサイズを大きくする)か、購買を抑える(プロデューサ側を一時停止する)ことが必要です。

身の周りにはどんなバッファがあり、何をやわらげていますか。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2回

3人の少年

彼らはどうやって僕たちを魅了(enchant)したか?

enchant.jsというライブラリがどうして誕生したのか。そしてどうしてこれほど短期間で多くの人々の心を捉え、魅了(enchant)したのか、よく聞かれます。今回はそのお話をいたしましょう。

その誕生の背景にはいくつかのドラマチックな出会いがありました。彼らとどのようにして出会っていったのか、そこからお話しします。

不思議な学生

2010年末。僕はとある大学生からメールをいただきました。

「大学でやっている自主ゼミの中で、ARG(代替現実感ゲーム)について研究発表したい。ついては清水さんにインタビュー取材をさせていただきたい」

僕は大学生からメールをいただいた場合、たいていの場合は会ってみることにしています。二つ返事で引き受けると、やってきたのは、伏見遼平。東京大学の1年生でした。

彼が取材したがっていたのは「CRIMSON FOX」という、僕らが経済産業省のバックアップのもと、2008年に実施したA/ARG(Augmented/Alternative Reality Game;拡張/代替現実感ゲーム)についてでした。これは辻と僕が手がけた仕事でした。このゲームは実際の渋谷の街をフィールドとして、街のあちこちに隠された暗号を集め、優勝者には当時発売直前だったiPadが授与されるというダイナミックなゲームでした。

やってきた伏見という少年は、まだあどけなさを残した少年でした。若干19歳です。

しかし驚いたのは、この伏見という少年が、あまりにも僕らの過去の仕事や、僕のブログの内容、過去のインタビューについて詳細な事前調査を重ねて来ていたことでした。彼の質問に対して僕が一言答えると、「それは○○年○○月のブログにあった内容ですね」と即座に出てきます。まるで僕も把握していない僕自身のデータベースが、伏見という少年の内部にあるかのようでした。それどころか、いつのまにか、インタビューされているはずの僕よりも、伏見少年の発言回数のほうが増えていき、僕はいつしか伏見少年の主張に心をすっかり奪われていたのです。

予定していたインタビューの時間が1時間過ぎても、伏見少年はずっと喋り続けていました。僕はずっと黙って話を聞いている辻のほうを見ると、彼女は目を合わせて、ニコリと笑いました。それで、僕は伏見少年の話を遮って、こう言ったのです。

「わかった。君がなにかとても言いたいこと、やりたいことがあるのは解った。だったら来週から、ここで働いてみないか?」

これが伏見少年との出会いでした。

もう一人の天才

彼がただ者ではないことは、あっという間に解りました。どんな仕事を与えても期待以上のアウトプットを想像以上に短い時間で上げてくるのです。

それからしばらくして、伏見少年は僕のもとにやってきて、こう言いました。

「実は、同級生に田中くんという変わった奴がいて、清水さんに会ってほしいのですが」

そもそも伏見少年が変わった奴だったのに、それに輪をかけて変わった奴というのはいかなる人物なのか、僕は興味をそそられました。じゃあ会ってみようか、ということで急遽面接がセッティングされ、僕はその1週間後には田中諒という少年と面談をしていました。

「普段、家でなにやってるの？」僕がそう聞くと、

「家ですか？　まあ、風呂に入ったり、寝たりとかですかね」と所在なさそうに答えました。

「趣味とかは？」さらに聞くと、

「別にこれといって趣味みたいなのはないです」と興味がないように答えました。

僕はなぜ伏見少年が彼を僕に引き合わせたいと思ったのか意図がわからなくなり、続けてこう聞いてみました。

「え、じゃあプログラミングはしないの？」すると田中少年はこう答えたのです。

「風呂に入ってるときと寝てるとき以外はずっとしてますね」

彼にとって、プログラミングは趣味ではなかったのです。そしてまさしく、寝ても覚めてもプログラミングをするという資質こそが、僕が常日頃から掲げる「優れたプログラマ」になるための1つの条件でした。それで伏見少年が僕に田中少年を引き合わせた意図をようやく理解するのです。

こうして伏見、田中コンビが毎日のようにUEIにやってくることになりました。

enchant.js

2人のコンビが会社にやってきては、毎日楽しくプログラミングで遊ぶという日々がはじまりました。彼らはまるで子どもがいたずらを報告するように、毎日僕のところにきて「こんなものを作った」と見せてくれたのです。

これに気を良くした僕は、さらに多くの“才能はあるけれどもその使い方を見つけることができない少年たち”を集めようと、「少年プログラマーよ、秋葉原に集え」というプログを書きました。

するとさらに大勢の才能あふれる少年たちがやってきました。そのうちの1人、高橋諒は、電子回路からC言語までなんでも操れる万能の19歳でした。いまどき電子回路を知っているプログラマは珍しいので、僕は一瞬で彼に秘めた才能があることを悟りました。彼はあまりに突出した才能を持っているため、常に孤独で、仲間や指導者を欲していました。そして伏見少年や田中少年は高橋少年の良きライバルであり、チームメイトとなる素養が充分にあったのです。

あるとき田中少年が、いい加減、自分で思いつくままにプログラミングすることに飽きたのか、「なにか書く（プログラミングする）ものはないか」と僕に聞いてきました。

これこそが、実は僕がずっと待ち望んでいた瞬間でした。

どれだけ優れた人間でも、思いっきり好きなものをプログラミングしたあと、ふと、なにを

enchant.js開発のきっかけとなったMSXとファミリーベーシックは現在、五反田のゲンロンカフェに展示されている

作ったら人に喜ばれるのか気まぐれに考えてみたくなるときがあるのです。

僕はすかさず、「ならばBASICを作ってみないか」と聞いてみました。ところが田中少年はポカンとしています。BASICを知らないのです。そこで僕は、すぐに秋葉原の電気街に出かけて行って、MSXとファミリーベーシックを買って来て、田中少年に見せました。

サンプルコードが書かれたマニュアルを一読すると田中少年は言いました。

「まさか！　こんな短い行数でゲームが作れるの!?」

その声に驚き、伏見少年、高橋少年も駆け寄ってきました。

「噂には聞いたことがあったけれども、BASICというのは、まるで夢のような環境だな」

「ポケコンのBASICしか触ったことない……」

思い思いのことを言いながらも、彼らはBASICの持つ可能性に目を輝かせていました。

「こういうことができるもの――つまり、スプライトとBG、そういうものをHTML5でエレガントに扱うライブラリ。そういうのを作ってみない？」

返事はありませんでした。彼らは一目散にマシンの前に走って行ったからです。田中諒は夢中になってコードを書き、わずか2週間でプロトタイプを完成させました。

「これを見てください」

息せき切ってやってきた田中諒が、スプライトとBGが見事に動くデモを完成させていました。ソースコードを確認しようとフォルダを見ると、「enchant.js」という見慣れないファイルがありました。

enchant.jsの最初のバージョンを2週間で開発した田中諒

「このenchant.jsってのは？」

「それが本体です」

なるほど。面白い。enchant.js。それが誕生した瞬間でした。

ミドルウェアの兵站戦略

enchant.jsはすぐに社内の経験豊富なプログラマ連中にお披露目されました。するとうるさがたの先輩プログラマ連中が「これ、いいんじゃないの？」と口々に言いはじめたのです。実はそのときまで、僕はenchant.jsをファイル名以外はよく知りませんでした。けれども、長年ゲーム開発をしてきたベテラン連中が軒並み「筋がいい」と褒めるので、これはいけるかもしれない、と思うようになります。

時代はまさにHTML5が標準化されようとしていましたし、iOSとAndroidのクロスプラットフォーム開発への関心も高まっていました。この領域には決定版と呼べるようなミドルウェアがまだ存在していなかったことも決め手となりました。

よし、これを真面目に売り出してみよう。僕はそう決意しました。いつしか僕自身もこのライブラリに魅了（enchant）されていたのです。

ただしこの手のライブラリを売り出すというのは一般に言ってかなり難しいことも事実でした。もしこれがどこかの大学の研究室で生まれたり、もしくは大学生個人が発表したとしたら、今のような流行には決してならなかったでしょう。そして本誌の読者が一番気がかりなのは、まさしくこの部分だと思います。数あるオープンソースソフトウェアの中で、なぜenchant.jsだけがこれほどまでに注目を集めたのでしょうか。

成功したものにあとから理由をつけるのは危険なのですが、敢えてその愚を犯すとすれば、成功の原因は僕らが“兵站戦略”と呼ぶものをもってenchant.jsを組織的・計画的に普及させていったことにあるでしょう。

実は単独のライブラリとして見れば、enchant.jsより多機能だったり、高性能だったりするも

のはいくらでもあります。しかし開発者は決して機能や性能だけでミドルウェアを選んではいないのです。たとえばRubyやPythonは、明らかにC++よりも実効効率で劣ります。にもかかわらず、Rubyを好んで使うプログラマは少なくありません。PHPは言語的な美しさでは他の言語に遥かに劣ります。しかし現実的にはPHPを採用する企業が後をたちません。なぜでしょうか。

Webサービスを作るとき、RubyがC++より好まれるのはなぜか。それは明らかに、Ruby on Rails(RoR)があるからです。RoRが登場するまで、WebサービスをRubyで構築する方法がないわけではありませんでしたがあまり現実的とは思われていませんでした。いわばRoRによってRubyは開発者に“再発見”されたのです。RoRは、非常に簡単に高品質なWebサービスを構築できるということで、あっという間に人気になりました。世界中で採用され、本も多数出版され、RoRが使えることが技術者の1つのスキルとして認知されるまでになりました。専門学校や大学でもRoRが教材に使われるようになっています。

こうしたことすべて、つまりRuby以外の環境すべてを、僕らは「兵站(へいたん)」と呼んでいます。兵站とは、戦争において、武器・弾薬の補給、兵員の補充、新兵の確保と教育、通信手段の確保など実際の戦闘以外を構成するすべてのことを指します。ベトナム戦争では5万人の兵士が派遣されましたが、5万人の兵士を支えるために米軍は20万人の兵站担当者を派遣しています。炊事・洗濯、治療、建設、通信、戦闘以外のすべてを支えるためにはそうした兵站が不可欠なのです。

武器の性能だけがいくら高くても、撃つ弾がなければ負けてしまいますし、銃を構える兵士が足りなければやはり負けてしまいます。だから兵站戦略は軍事作戦において最も重要な概念の1つなのです。将棋やチェス、囲碁といったゲームに足りないのはまさしくこの兵站の要素です。

実はミドルウェアも、実際の性能そのものよりももっと重要なのはそのミドルウェアの周辺、つまり、

enchant.jsプロジェクトのリーダーとなる伏見遼平(右)とリードプログラマとなる高橋諒(左)

- ミドルウェアを使うプログラマの教育
- ミドルウェアを使うことが1つのスキルとして認知されること
- ミドルウェアの解説書やWebサイトが多数出ていること
- サンプルコードが手に入りやすいこと
- コミュニティが活発であること

などがより重要なのです。この点を考慮しないと、ミドルウェアは失敗します。誰も使わないミドルウェアには存在価値がないからです。

21歳の頃、僕はまさにこのミドルウェアの世界戦略を目の当たりにする立場に居ました。米Microsoftで、DirectXのエヴァンジャライズ活動に関わっていたのです。enchant.jsもゲームが前提でない環境でゲーム開発を実現するという視点では、実はDirectXと同じです。ならば、DirectXと同じやりかたで兵站戦略を実行すれば成功する確率は高いと確信していました。なによりこういう気軽にゲームを作って発表できる環境を、一番求めているのは僕自身でした。僕には絶対の自信がありました。

けれども、伏見、田中、高橋の3人は、自分たちが作っているものが世の中で受け入れられるとはまったく思っていませんでした。僕にはミドルウェア戦略の経験と人脈があり、彼ら若者には未知のものに対する情熱がありました。enchant.jsプロジェクトはこうしてはじまったのです。SD

コレクターが独断で選ぶ!

偏愛キーボード図鑑

濱野 聖人 HAMANO Kiyoto
khiker.mail@gmail.com
Twitter：@khiker

第2回

一般的なキーの並びで Dvorak配列を使える

写真1　Matias Dvorak Keyboard

Matias Dvorak Keyboard & QIDO

はじめに

今月もDvorak配列にスポットをあてていきます。先月も触れましたが、Dvorak配列とは英文入力に最適化されたキーボード配列です。一般的なQwerty配列と比較して、英文を高速に入力できると言われています。

前回のTypeMatrixはキーの並びが「｜｜」型の格子状で特殊な並びでしたが、今回は「＼＼」型の一般的なキーボードと同じ並びでDvorak配列を使えるキーボードを紹介します。OSの設定でDvorak配列にするのではなく、もともとDvorak配列となっているキーボードを使うことで、BIOSの画面やLive CDでもDvorak配列が使えます。今回も英語キーボードですので、OSのキーボード設定は英語にする必要があります。

写真2　Matias Dvorak Keyboardのキートップ

Matias Dvorak Keyboard

Matias Dvorak Keyboard(**写真1**)は、アメリカのMatias社が販売しています。一般的なキーボードと同じキーの並びでDvorak配列を全面に押し出した、非常に数少ないキーボードです。

入手方法

同社のサイト[注1]で販売しています。クレジットカードで支払いが可能です。値段は、筆者購入当時で$99ほどでした。日本へも発送してくれます。海外からの発送になりますので、注文から到着まで20日ほどかかりました。

写真3　Qwerty配列時のLED点灯

特徴

Matias Dvorak Keyboardは、次の特徴があります。

- Dvorak配列
- Qwerty配列に切り替え可能
- メンブレンキーボード

最大の特徴は、ハードウェアでDvorak配列であることです。キートップにDvorak配列の文字が大きく印字されています(**写真2**)。キーボード上部にボタンがついており、それを押すとQwerty配列にも切り替えられます。Qwerty配列にしている際は、ボタンが青色に点灯します(**写真3**)。これにより練習中や第三者に一時的にキーボードを貸し出す際にも、すぐに切り替えられます。

キースイッチはメンブレンです。ちまたにあるメンブレンキーボードとそう変わらない打鍵感です。そのため、一部の高級なキーボードと比較するとどうしても打鍵感は劣ります。そのほかに、2ポートのUSBハブも内蔵しています。

注1）http://www.matias.ca/dvorak/

QIDO

QIDOはキーボードではなく、keyghost社が販売しているUSBアダプターです(**写真4**)。Qwerty-In Dvorak-Outの略であり、Qwerty配列をDvorak配列に変換します。Qwerty配列のキーボードとPCの間に接続して使います。

入手方法

同社のWebサイト[注2]で販売しています。Matias Dvorak Keyboardと同様にクレジットカードで支払えます。値段は、筆者が購入した当時は1個$89ほど。日本への発送も可能で、送料は$20ほどでした。

特徴と利用方法

QIDOの特徴は、既存のQwerty配列のUSBキーボードをそのままDvorak配列に変換できることです。普通のDvorak配列だけでなくDvorak配列のほかのバリエーションも使えます。

デフォルトでは通常のDvorak配列の設定ですが、QIDOを接続したうえでテキストエディタにフォーカスをあて、「keydvorak」と打つと以下のバリエーションに変更するためのメニューを開けます(**図1**)。

- Dvorak-Qwerty
- 片手Dvorak(左手)
- 片手Dvorak(右手)

メニューで変更した情報は、QIDO自体に保存されるので、次回からも変更後の配列で使えます。

Dvorak-Qwertyとは通常のDvorak配列と同じですが、修飾キーと組み合わせてキーを打ったときQwerty配列となります。具体的には、Ctrl、Alt、Windowsキーと組み合わせて普通のキーを打ったときQwerty配列となります。SHIFTではDvorak配列のままです。たとえば、Windowsでコピー内容を貼り付けるショートカットのCtrl＋Vを押すとき、Dvorak配列ですと、Qwerty配列のCtrl＋.となりますが、Dvorak-QwertyではそのままのCtrl＋Vの位置で入力できます。そのため、キーボードショートカットを違和感なくそのまま使うことができます。

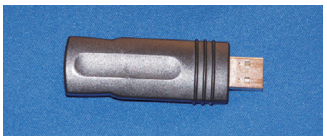


写真4　QIDO

片手Dvorak(左手)と片手Dvorak(右手)は、片手で打つ用のDvorak配列です。詳細は省略します。

NUM LOCKをすばやく2度押すことで、Dvorak配列とQwerty配列の切り替えもできます。ただ、QIDOには次の制約があることに注意が必要です。

①接続するキーボードは、英語キーボードである必要がある
②トラックパッドのようなキーボードの追加機能を無効にする場合がある
③一部のUSBポートを持つキーボードでは動作しない

図1　QIDOのメニュー

①は、QIDOは英語キーボードを前提としているので、日本語キーボードを接続すると一部のキーがおかしな位置に変換されてしまうということです。②は、トラックパッドを内蔵しているようなキーボードとQIDOを接続した場合、キーストロークは通るが、トラックパッドは無効になる場合があるということです。③は、たとえば、Happy Hacking Keyboard Professional 2のようなUSBポートを内蔵するキーボードでは、動作しないということです。③について、**表1**に動作確認した一例を挙げます。

◆ ◆ ◆

これ以外にもDvorak配列を利用できるキーボードは存在します。また、キー配列を自由自在に変更できるプログラマブルキーボードを使うという方法もあります。このプログラマブルキーボードについては、今後紹介する予定です。SD

表1　QIDOの動作状況

キーボード	動作状況
HHKB Lite2(PS/2モデル)	問題なし(PS/2→USB変換コネクタを利用)
HHKB Professional	問題なし
HHKB Professional 2	動作せず
HHKB Professional 2 Type-S	動作せず
Realforce 87UB SE170S	問題なし
Majestouch Ninja Tenkeyless FKBN87MRL/EFB2	問題なし
Lenovo ThinkPad USBトラックポイントキーボード	問題なし(トラックポイントも動作)

注2) http://www.keyghost.com/qido/

はんだづけカフェなう

深圳(しんせん)のMaker Faireに出展してきた

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/

Shenzhen Mini Maker Faire

ちょうど1年前の本連載第20回で紹介した、深圳(しんせん)でのMaker Faireが今年も開催され、出展しないかというオファーをいただいたので参加してきました。今年のMaker Faireは前回とは違い、市内中心部の屋外に設置された大きなテントのもとで開催されました(**写真1**)。

今回のMaker Faireも前回と同様に、中国のホビースト(趣味に熱中している人)よりも、ロボットを作っている会社、基板やキットといったOSHW(オープンソースハードウェア)関連の仕事をしている人々の出展が多くを占めていました(**写真2**)。

日本でも、3Dプリンタが注目を集めており、今までものを作ることを趣味にしていなかった人々も3Dプリンタに関心を持ち始めています。同様に、今回の深圳でのMaker Faireには3Dプリンタメーカーも出展しており、さまざまな3Dプリンタを見かけました(**写真3、4**)。

このMaker Faireはかなりの盛況で、東京よりも規模自体は小さいものの、会場にはたくさんの来場者が訪れていました。午後からは来場者数が多くなり過ぎたためか、入場制限が行われるほどでした(**写真5**)。

筆者の出展

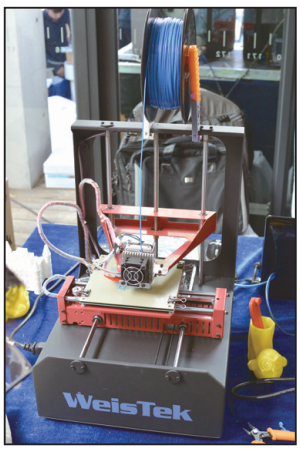
先ほどの会場入り口の看板にもロゴを刷って

▼写真1　会場入り口

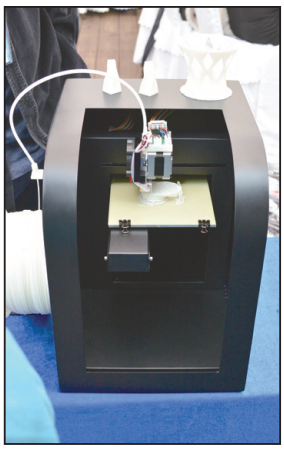
▼写真2　会場内

▼写真5　並んで待つ来場者

▼写真3　3Dプリンタ①



▼写真4　3Dプリンタ②



いただいていたのですが、筆者は最近“Tinker-Pot”という名前で、自分が作った基板やキットを出していきたいと考えています。今回は急にオファーをいただき、とくに何か成果物を用意していたわけではなかったため、ちょうど試作を終えたところだったCMSIS-DAP（しむしずだっぷ）というARMのプロセッサをデバッグするための標準化された規格に対応した、デバッグアダプタ（パソコンをマイコンのデバッグポートに接続するための機器）などを展示してきました。

また、第25、26回でも紹介したLPC1114FN28と、最近リリースされたLPC810というNXP社のブレッドボードで使えるARMマイコン、それから、以前開発をしたmbedの互換機を展示しました（**写真6**）。

余談ですが、筆者は最近、マイコンでの開発をprintfデバッグのみで行うことに疲れ、デバッガが使えることが楽しくてしかたありません。このCMSIS-DAPという規格を使ったデバッグ環境は、MDK-ARMというARM社の比較的高価な開発環境が必要だったのですが、最近はOpenOCDというソフトウェアをCMSIS-DAPに対応させたものが開発されるなど、CMSIS-DAPをGDB（GNU Project debugger）と併用できる環境も整いつつあります。

中国では英語を話せる人が少ないのではないかと考え、パネルも中国語に翻訳してもらい、中国語の通訳の人もお願いしました。もちろん筆者の英語力が大したことないという事情もあるのですが、来場者の半数以上は英語でのコミュニケーションが難しかったのでこのような準備をしておいて正解でした。

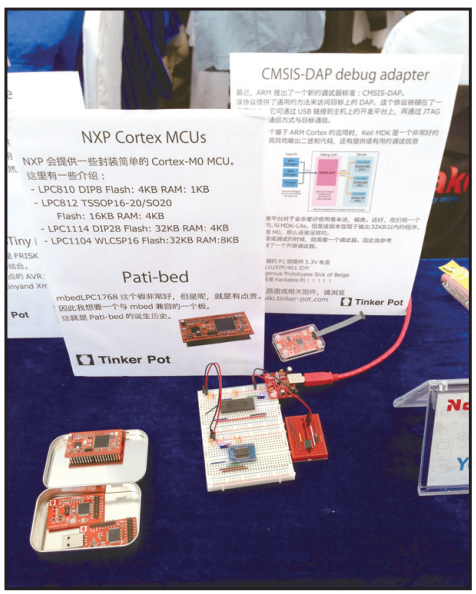
筆者が興味を持った出展

通訳の方が心強い方だったので、ほかの出展者の出展内容も見てまわってきました。興味を引いたものをいくつか紹介します。

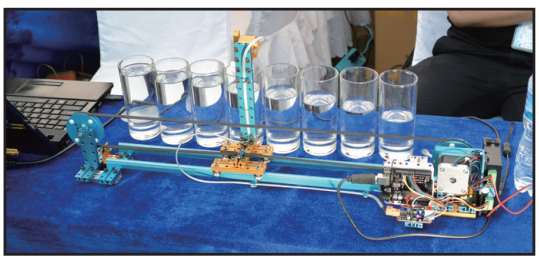
Makeblock（**写真7**）は深圳の企業が開発し販売しているアルミ製のブロックで、ネジ止めをすることで簡単にロボットなどを作ることができるものです。会場では、水が入ったコップを叩いて演奏をするロボットが展示されていました。

iPhoneをオモチャの銃に吸盤で固定するガンコントローラも見かけました（**写真8**）。正確

▼写真6　筆者のブース

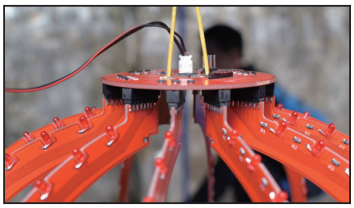


▼写真7　Makeblock



▼写真8　iPhoneガンコン

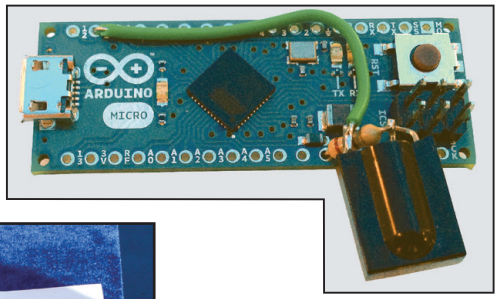
▼写真9　LED灯籠（上部）



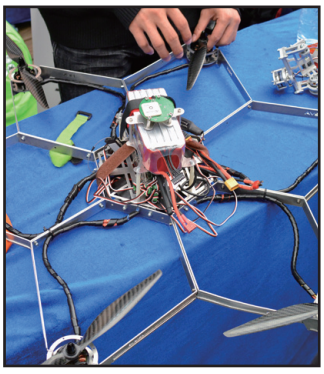
▼写真10　LED灯籠（全体）

▼写真11　BLE

▼写真13　Arduinoリモコン受信機



▼写真12　HEX AIR ROBOT



には出展者ではなく、会場に作ったものを持ってきていた方の作品ですが、銃を動かすとiPhoneに表示されているゲームの画面が動き、また引き金を引くといった操作もできていました。英語が話せない方だったので、詳しくしくみを聞くことができませんでしたが、なかなかの人気を集めていました。

とても中国的でおもしろいと思ったものが、基板で作られた灯籠です（**写真9、10**）。上手に基板を組み合わせて立体的な灯籠を組み立てていました。上部にはマイコンが搭載され、LEDの点灯をコントロールしていました。

そういえば、この連載でBLE（Bluetooth Low Energy）を扱ったことがありませんでした。BLEはBluetooth v4.0規格の一部で、iPhoneと自作のハードウェアをつなぐ最も手っ取り早い方法です。従来はBluetoothを使ったアクセサリを開発するにはAppleのMFiライセンスプログラムに参加する必要がありましたが、BLEを使った通信にこのような手続きは必要ありません。**写真11**のシールドやアダプタは香港のRedBearLabが開発したもので、会場では少しお得なプロモーション価格で販売されていました。

HEX AIR ROBOT（**写真12**）は、HAXLR8R（ハクセラレーター）というものづくり支援プログラムに参加している人達が作っている回転翼機です。けっこうな大きさで、混み合っている会場では全体をカメラのフレームに納めることができませんでした。興味を持った方は検索してWebを見てみてください。

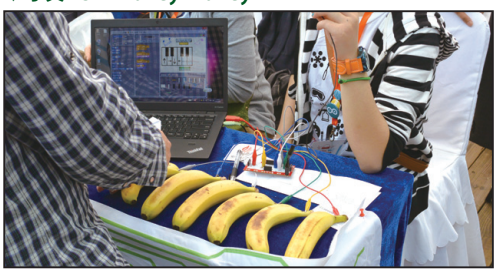
写真13は展示物ではないのですが、目についたので撮らせてもらいました。Maker FaireにはArduino Teamも出展していたのですが、スタッフの1人がプレゼンをするために自作したものということです。比較的最近発売されたArduino Microに赤外線リモコンの受信機を接続し、赤外線リモコンでプレゼンのスライドを切り替えることができるようにしたものです。

Seeed Studioの出展

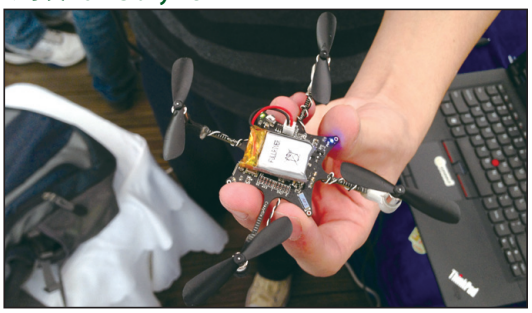
Seeed Studioはかなりの量のシールドを開発

▼写真14　Arduinoのシールド

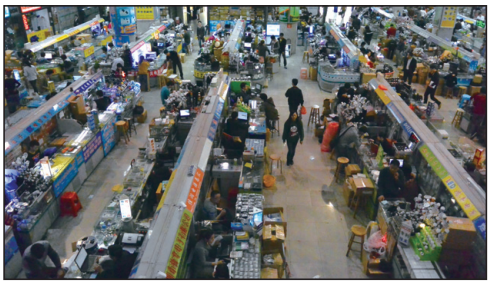
▼写真15　Makey Makey



▼写真16　Crazyflie



▼写真17　華強北のビル内



し、販売しています。写真14に展示されているものは一部なのですが、Maker Faireの彼らのブースに行くと、いつも写真のように多くのシールドが展示されています。とくに、彼らが出しているGroveというシリーズの製品は、Arduinoにさまざまなセンサやデバイスを、はんだづけすることなくケーブルで簡単に接続できますので、ちょっとArduinoをはじめてみたいけれども面倒は嫌だという方にはお勧めです。

Makey Makey（写真15）は、ミノムシクリップでつなぐだけでいろんなものをパソコンのキー入力のキーにできるオモチャです。ここではバナナを鍵盤にした展示がなされていました。恥ずかしながら筆者はこのキットを知りませんでしたが、すでにスイッチサイエンスのWebサイトでも販売されている比較的有名なものだそうです。

筆者が個人的にほしいと思ったオモチャが、このちっちゃいクワッドコプターです（写真16）。裏側にマイコンが搭載されており、無線でコントロールすることができるようです。当日はゲーム機のコントローラで操縦していましたが、スマートフォンやパソコンでの操縦も可能とのことでした。現在、量産をはじめているところのようですので、近いうちにSeeed StudioやスイッチサイエンスからSeeed購入できるようになるでしょう。価格などもわかりませんが、楽しみにしています。

華強北（Huaqiangbei）

昨年深圳に訪問した際には行けなかったのですが、華強北にも行ってきました。華強北は、深圳市内にある秋葉原のような電子街です。秋葉原には電子部品のお店が大分少なくなってしまいましたが、製造業の盛んな深圳ですので、華強北には多くの電子部品店が存在します。大きなビルが建ち並んでいる街並みは、ぱっと見は電子部品が売っているようには思えないのですが、中に入ると写真17のようにところ狭しと小さなお店が並んでいます。

◆

深圳でのMaker FaireはHAXLR8Rに参加している人々が出展していたこともあり、中国人以外の出展者も多く、とても楽しめるものでした。筆者は海外のMakerとの交流が楽しいので、Maker Faire Tokyoにももっと多くの外国人出展者が増えるとおもしろくなるだろうに、と感じています。SD

PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは2013年6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 AtermWG1800HP

1名

無線LANの新規格「Draft IEEE802.11ac」に対応したWi-Fiホームルータ。5GHz帯で最大1,300Mbpsの高速Wi-Fi通信が可能。また世界最小クラスのアンテナ「μSRアンテナ」を採用することで、Draft 11ac対応でありながら従来機と同等のコンパクトデザインを実現しています。

提供元 NECアクセステクニカ
URL http://www.necat.co.jp

02 リュックの中身 Ver.3.0

3名

※製品には写真内の収納品は付属しません。

リュックの中に入れて使うことで収納力をアップできるポケットボード。ノートPCやタブレット端末を持ち運びすることも考慮し、衝撃吸収のためのスポンジパットや固定ベルトが付いています。

提供元 ビー・ナチュラル
URL http://www.b-natural.co.jp

03 スマートフォン防水・防塵ケース dicapac W-P2

1名

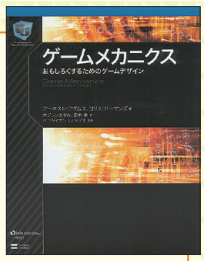
※製品にスマートフォンは付属しません。

お手持ちのiPhoneやスマートフォンを手軽に防水化できます。濡れた手でもタッチパネル操作できるうえ、完全に水に入っても大丈夫。背面には撮影用ウィンドウ付きで鮮明な写真撮影が可能です。

提供元 大作商事
URL http://www.daisaku-shoji.co.jp

04 ゲームメカニクス

3名



アーネスト・アダムス、ヨリス・ドーマンズ　著／
B5変型判、400ページ／
ISBN = 978-4-7973-7172-7

RPGでも携帯電話のゲームでも、おもしろいゲームには優れたしくみがある。本書はそんなゲームの核となるメカニクスについて設計、テスト、チューニングする方法を解説する。

提供元 ソフトバンク クリエイティブ　**URL** http://www.sbcr.jp

05 SSD 完全攻略読本

2名

北川 達也、加藤 勝明、鈴木 雅暢、竹内 亮介　著／
A5判、192ページ／
ISBN = 978-4-8443-3366-1

I/O効率に優れ、HDDの代替手段として注目を浴びているSSD。そんなSSDの上手な選び方、使いこなし方、そして高速性を長持ちさせるテクニックなどを紹介します。

提供元 インプレスジャパン　**URL** http://www.impressjapan.jp

06 これも数学だった!?

2名

河原林 健一、田井中 麻都佳　著／
新書判、208ページ／
ISBN = 978-4-621-05382-9

携帯電話、カーナビ、電力需給、物流、SNSなど、コンピュータを通して私達の生活に役立っている離散数学について解説します。プログラマとしてIT企業で活躍したいと思っている人、必見です。

提供元 丸善出版　**URL** http://pub.maruzen.co.jp

07 小飼弾のコードなエッセイ

2名

小飼 弾　著／
A5判、200ページ／
ISBN = 978-4-7741-5664-4

ブログ「404 Blog Not Found」でおなじみ小飼弾氏によるエッセイ集。2013年3月までの本誌連載の記事を中心に、コード、アルゴリズム、ソフトウェアについて語った内容を1冊にまとめました。

提供元 技術評論社　**URL** http://gihyo.jp

第1特集

わかった人だけメキメキ上達

ちゃんとオブジェクト指向できていますか？

今、多くのソフトウェア製品やWebサービスは、オブジェクト指向をベースにしたプログラミング言語で開発されています。C++やJavaはもちろんオブジェクト指向を習得していることが必要ですが、いまやスクリプト言語もオブジェクト指向でプログラミングするのがごく当然になってきています。しかし、オブジェクト指向は、わかっているようでわからなかったり、いつまで経ってもマスターできなかったりすることもよくあります。

本特集は新年度を迎えるにあたり、そんな苦手意識を克服するには、どうしたらよいのか、その手がかりを紹介します。Part1ではJavaで基本を押さえ、Part2では全体象を振り返り、Part3では現場でどのように使われているのか確認し、Part4、Part5、Part6で実践的なコーディング方法を示します。初心者プログラマさんだけでなく、ベテランの皆さんにもお勧めします！

1 part

オブジェクト指向の基本を学ぶ

オブジェクト指向の良さとは、見通しのよいプログラムを書けるようになることです。ソースコードを部品化し、それらを組み立て、動かす。実行してエラーが出たら、その部分だけを修正する。もしくは動的に組み立てプログラムを進化させることができます。本稿では、その基本を学ぶことで、皆さんの力を2倍にも3倍にもすることを目的にしています。

有限会社システム設計　増田亨（ますだとおる）

はじめに

Javaは、さまざまな分野で使われているオブジェクト指向のプログラミング言語です。

この記事では「基本から学ぶオブジェクト指向」ということで、

・オブジェクト指向の基本的な発想
・Javaでのオブジェクト指向実践のコツ
・オブジェクト指向をもっと勉強するヒント

を書いてみます。

皆さんがオブジェクト指向の考え方を活かしたプログラミングをすることのお役に立てれば幸いです。

オブジェクト指向の発想

「オブジェクト指向」は良いソフトウェアをできるだけ楽に開発するための工夫の1つです。

特定のプログラミング言語や、特定の書き方ではなく、プログラミングの「発想」、プログラミングの「スタイル」と考えるのが良いでしょう。

オブジェクト指向の良さ

オブジェクト指向でうまくプログラミングすると、全体の見通しが良くなり、機能追加や修正が簡単で安全になります。

プログラムの規模が大きく、機能追加や修正が継続的に発生するソフトウェアを開発するとき、オブジェクト指向のありがたさが実感できます。

Javaは規模が大きく、継続的にコードを変更するソフトウェア開発を意図した言語です。Javaを使ってオブジェクト指向プログラミングする動機は「複雑さ」への対応や「継続的な変更（拡張や修正）」が楽になることです。

オブジェクト指向は部品指向

「複雑さ」への対応や「継続的な変更」が楽になるのは、ソフトウェア全体をオブジェクトという「部品」の集まりとして組み立てるからです。

オブジェクト指向プログラミングとは「部品」をいろいろ用意して組み立てる「部品指向」の作り方なんです。

ひとつひとつの部品（オブジェクト）を、小さく単機能にすることがオブジェクト指向プログラミングのコツです。

既存のクラスやメソッドにどんどんコードを追加していくのは失敗パターンです。

・機能追加は、新しい部品（オブジェクト）の追加を考える
・修正は、部品の差し替えを考える

こういう発想でソフトウェアを育てていくのがオブジェクト指向の基本の発想です。

動的に組み立てる

現実の機械は部品間の関係ががっちり固定さ

れています。

オブジェクト指向プログラミングはもっと動的です。プログラムをメモリ上で実行するときにオブジェクト(部品)の組み立て方や結びつき方を動的にコントロールします。

この「実行時に動的に組み立てる」という感覚がオブジェクト指向プログラミングのもう1つのポイントになります。

名前をつける／名前を変える

ソフトウェアの規模が膨らみ、複雑になってくると、だんだん見通しが悪くなってきます。ちょっとした変更も、難しく危険な作業になります。

この問題を軽減するコツが「名前」の工夫です。

パッケージ名、クラス名、メソッド名を意図が明確で区別がしやすい名前にしましょう。名前によって全体の見通しが良くなり、変更すべき個所の特定も簡単になります。

オブジェクト指向プログラミングの成功と失敗の分岐点は良い名前を見つけることができるかどうかです。

最初はぎこちない名前から出発して、プログラミングしながら、何度も名前を見直し、よりわかりやすい名前に変えることを地道に続けることが、オブジェクト指向プログラミングなんです。

Hello, World!

実際のコードで考えてみましょう。**リスト1**は、おなじみのHelloWorld.javaです。プログラミングの第一歩としては悪くはありません。でも、オブジェクト指向プログラミングの第一歩としては不適切なサンプルです。

・どこにオブジェクトがあるの？
・そのオブジェクトをどうやって用意したの？
・部品を組み立てる感じがでている？
・良い名前をつけているの？

……疑問だらけです。

オブジェクトを用意する

部品を用意する感覚を身につける第一歩が、**リスト2**と**リスト3**です。

コンソール画面に Hello, world! を表示する動きは同じですが、リスト1とは違う書き方になっています。

・Launcher と Service にクラスを分けた
・パッケージhelloを宣言した
・クラス宣言はpublicではなくデフォルト(パッケージスコープ)
・hello()メソッドもpublicではなく、デフォルト(パッケージスコープ)

同じことをやるのに、パッケージを導入したり、クラスを分けたりしています。行数も増え

▼リスト1　HelloWorld.java

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello, world!");
  }
}
```

▼リスト2　Launcher.java

```
package hello;

class Launcher
{
  public static void main(String[] args)
  {
    Service service = new Service();
    service.hello();
  }
}
```

▼リスト3　Service.java

```
package hello;

class Service
{
  void hello()
  {
    System.out.println("Hello, world!");
  }
}
```

ました。ステートメント(文)の数で、リスト1は3行なのに、リスト2と3の合計で9行。3倍に膨らんでいます。

リスト2で注目してほしいのは、

```
Service service = new Service();
```

の行です。

new 演算子で、クラスのインスタンス(＝オブジェクト)を作成していますね。Service クラスは、リスト3のService.javaで宣言しています。

リスト1に比べ、部品作って、実行時に動的に組み立てる、という感じがでてきました。

serviceオブジェクトに仕事をしてもらうために、hello()というメッセージ[注1]を送っている行も、オブジェクト指向らしい個所です。

もう少しまともな挨拶を

ちょっぴりオブジェクト指向らしくなったところで、Hello, world! を発展させてみましょう。

・午前と午後で別の挨拶をする
・相手の名前を呼ぶ

午前であれば「増田さん、おはようございます」、午後なら「増田さん、こんにちは」という具合です。

手続き型とオブジェクト指向の発想の違い

リスト4は手続型で書いてみました。10行ほどのプログラムです。

リスト5は、リスト4のプログラムをメソッドに分割して、構造化プログラミングのスタイルで書いたものです(下請けのメソッドの内容は省略)。

構造化プログラミングは、手続きが複雑になってきたら、小さな手続きに分割して組み立てるという発想です。

オブジェクト指向では、分割の単位が「手続き」ではなく「オブジェクト」になります。

オブジェクトに分割して、それぞれのオブジェクトに仕事を分担してやってもらう、という発想です。

▼リスト4　GreetingProcedural.java

```
package hello;

import java.util.Calendar;

class GreetingServiceProcedural
{
  void greet()
  {
    String user = System.getProperty("user.name");

    Calendar calendar = Calendar.getInstance();
    int hour = calendar.get(Calendar.HOUR_OF_DAY);

    String message = "";

    if( hour < 12)
    {
      message = "おはようございます";
    }
    else
    {
      message = "こんにちは";
    }

    System.out.println( user + "さん、" + message );
  }
}
```

▼リスト5　GreetingServiceStructured.java

```
package hello;

import java.util.Calendar;

class GreetingStructured
{
  void greet()
  {
    String user = username();
    int hour = hour();
    String message = message(hour);
    print(user, message);
  }

  // 下請けメソッド群
  private String username() { ... }
  private int hour() { ... }
  private String message(int hour) { ... }
  private void print(String user, String ➡
message) { ... }
}
```

注1)「オブジェクトにメッセージを送る」という言い方は、Java ではあまり一般的ではありません。オブジェクトという部品に仕事を依頼する、という感じを大切にしたいので、この記事では、メッセージを送るという表現を使います。

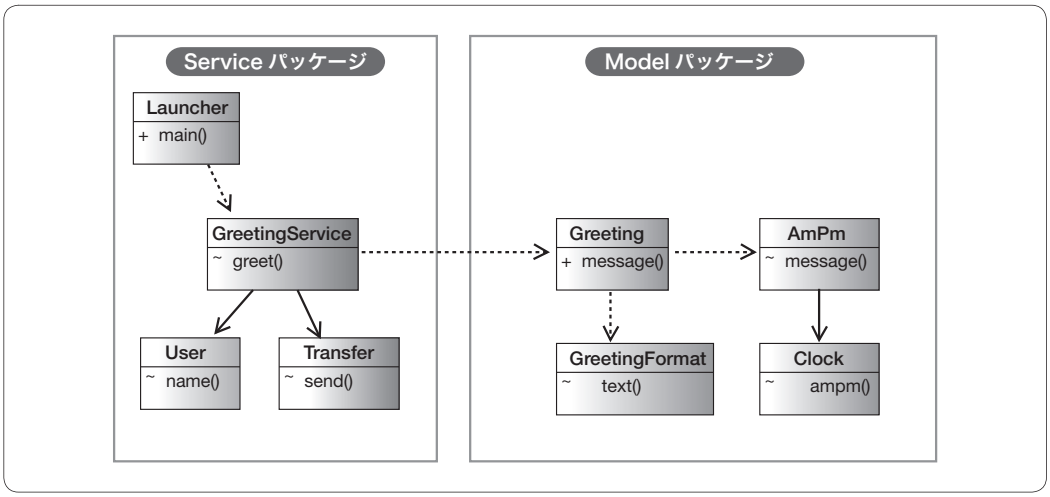
オブジェクトで仕事を分担する

リスト4の手続き型プログラムを、オブジェクト指向の発想で設計しなおしたのが、**図1**のクラス図です。

8つのオブジェクトで仕事を分担する設計にしました。なぜこう設計したのか理由を説明します。

図1のクラスを**リスト6**から**リスト13**にコーディングしました。これらを参考にしてください。

▼図1　クラス図



▼リスト6　Launcher.java

```
package greetingservice;

class Launcher
{
  public static void main(String[] args)
  {
    GreetingService service = new GreetingService();
    service.greet();
  }
}
```

▼リスト8　Transfer.java

```
package greetingservice;

import java.io.PrintStream;

class Transfer
{
  private PrintStream out;

  Transfer()
  {
    out = System.out;
  }

  void send(String message)
  {
    out.println(message);
  }
}
```

▼リスト7　GreetingService.java

```
package greetingservice;

import greetingmodel.Greeting;

class GreetingService
{
  User user;
  Transfer transfer;

  GreetingService()
  {
    this.user = new User();
    this.transfer = new Transfer();
  }

  void greet()
  {
    Greeting greeting = new Greeting(user.name());
    String message = greeting.message();
    transfer.send(message);
  }
}
```

▼リスト9　User.java

```
package greetingservice;

class User
{
 a private String name;

  User()
  {
    name = System.getProperty("user.name");
  }

  String name()
  {
    return name ;
  }
}
```

▼リスト10　Greeting.java

```
package greetingmodel;

public class Greeting
{
  private String username;

  public Greeting(String username)
  {
    this.username = username;
  }

  public String message()
  {
    AmPm ampm = new AmPm();
    GreetingFormat format = new GreetingFormat(username, ampm);
    return format.text();
  }
}
```

▼リスト12　Clock.java

```
package greetingmodel;

import java.util.Calendar;

class Clock
{
  private Calendar calendar;

  Clock()
  {
    this.calendar =  Calendar.getInstance();
  }

  String ampm()
  {
    int hour = calendar.get(Calendar.HOUR_OF_DAY);
    if( hour <= 12 ) return "am";
     return "pm";
  }
}
```

▼リスト11　AmPm.java

```
package greetingmodel;

class AmPm
{
  private Clock clock ;

  AmPm()
  {
    this.clock = new Clock();
  }

  String message()
  {
    String ampm = clock.ampm();

    String message = "";

    if( ampm.equals("am"))
    {
      message = "おはようございます";
    }
    else if( ampm.equals("pm"))
    {
      message = "こんにちは";
    }

    return message ;
  }
}
```

▼リスト13　GreetingFormat.java

```
package greetingmodel;

class GreetingFormat
{
  private String username;
  private String message;

  private static final String template  = "%sさん、%s。";

  GreetingFormat(String username, AmPm ampm)
  {
    this.username = username;
    this.message = ampm.message();
  }

  String text()
  {
    return String.format(template,  username,message);
  }
}
```

LauncherとGreetingServiceの分離

GreetingServiceクラスは、挨拶する機能を部品化したものです。プログラムの起動と、サービスの実行を別のオブジェクトに分けるのは定

石ですね。プログラムを起動するmainメソッドやmainメソッドを持つクラスに、それ以外の仕事を詰め込んではいけません。

Userオブジェクト

挨拶の相手ですね。「相手の名前」という要求を素直にオブジェクトにしました。Javaプログラムを実行しているユーザの名前を使っています。

Transferオブジェクト

コンソールに出力する仕事を分離しました。出力先を変更するときにはこのクラスが変更対象です。Transfer#send()というメソッド名で表したように将来は画面出力よりも、ネットワーク上に送信することを視野に入れました。

パッケージの分割

Launcher、GreetingService、User、Transferを「サービスパッケージ」にまとめました。Greeting、AmPm、Clock、GreetingFormatは、「モデルパッケージ」にまとめました。サービスパッケージは、アプリケーションの実行方式や実行環境に依存したコードがいろいろ含まれています(Systemクラスとか)。モデルパッケージは「挨拶する」という概念モデルをそのまま実装したパッケージです。プログラムの実行環境や実行方式に関係した変更は、サービスパッケージで行います。どんなとき、どんな挨拶をするか、という挨拶モデルの変更はモデルパッケージで行います。モデルパッケージで公開しているのは、Greetingクラスだけです。挨拶モデルのロジックをモデルパッケージ内で変更しても、サービスパッケージは、それに影響されません。パッケージを分離して、できるだけpublicスコープのクラスやメソッドを作らないのが、結びつきを弱くする、変更に強くなる良い設計です。

Greetingオブジェクト

「挨拶」を代表するオブジェクトです。モデルパッケージの外部(使う側)に公開された唯一のオブジェクトです。実際の仕事は、パッケージ内のほかのオブジェクトに依頼します。外部とやりとりするオブジェクトは、やりとりする役割だけに専念させて、実際の仕事はほかのオブジェクトに任せるのが、オブジェクトの役割分担の定石の1つです。

AmPmオブジェクトとClockオブジェクト

午前と午後で別の挨拶をするロジックを2つのクラスで役割分担しました。Clockオブジェクトは、今が午前か午後であるかの判断を担当します。AmPmオブジェクトは、Clockオブジェクトの午前／午後の判断を使って、挨拶の文言を決めます。Clockオブジェクトは、たとえば、朝の挨拶は12時ではなく11時まで、という判断基準の変更を吸収します。将来は、夕方や深夜も判断するように発展するかもしれません。AmPmオブジェクトは、午前／午後の枠組みを固定で想定したオブジェクト名になっています。将来、いろいろな挨拶パターンを追加するときは、オブジェクト名やメソッド名はまったく別のものになるはずです。今は「午前と午後で挨拶を変える」という要求に合わせてこういうクラス名にしています。

GreetingFormatオブジェクト

名前に「さん」を付けたり、挨拶を文として組み立てる仕事を担当します。言い方を変更するときには、このクラスで宣言している、テンプレートを変更します。

小さなオブジェクトで仕事を組み立てる

リスト4の手続き型の書き方で、10行ちょっとのプログラムを、オブジェクト指向的に書くと、図1とリスト6〜13のようになりました。

いかがですか？

ファイルが増え、コードの総行数が膨らみ、どこで何をしているか、直観的にわかりにくくなってしまいましたか？

筆者は、こういうスタイルを身につけることで、ソフトウェア開発の仕事がずいぶん楽になったし、良い仕事ができるようになったと思っています。

経験的には、リスト4やリスト5のスタイルで、追加・修正を繰り返すと、書いた自分でも、どこを変更すればよいのかわからなくなり、わけのわからない副作用に悩まされるハメに陥ります。

リスト6〜13の、小さなオブジェクトに単純な仕事を役割分担させるスタイルだと、変更すべき個所は明確だし、変更が局所的になり、副作用の心配が激減します。

オブジェクトの役割を小分けにして、良い名前を探すのは、たいへんな作業です。

その代わりうまく役割分担ができて良い名前が見つかったときには、びっくりするくらい全体の見通しが良くなり、拡張や修正が簡単に安全にできるようなります。

オブジェクトの作成

オブジェクト指向プログラミングでは、オブジェクトを「誰」が「いつ」作成するかは、重要な設計課題です。new演算子をどこに書くか、という問題ですね。

クラス図の表現、コードの実装

図1のクラス図で矢印の線が実線と破線を使い分けていることに注目してください。

リスト6〜13のコードと突き合わせてもらえばわかりますが、

・実線は、コンストラクタでnewしている
・破線は、メソッドでnewしている

という違いがあります。

結びつきの強さ

実線矢印の場合は、コードではコンストラクタで、仕事を頼むオブジェクトを作成しています。自分自身がオブジェクトとして誕生するときに、仕事のパートナーを、自分で用意するわけです。

破線は、メソッドが呼ばれたときに、一時的にオブジェクトを準備します。メソッドが終了すれば仕事の依頼先のそのオブジェクトは不要になります。

強く固定的な関係は、コンストラクタで(実線で)、弱く一時的な関係は、メソッドで(破線で)、という結びつき方の違いを意識しましょう。

「午前午後」と「時計」とは、強く結び付いた関係です。「挨拶サービス」と「挨拶」は弱い一時的な関係です。

強い／弱いをどう使い分けるかは単純なルールはありません。またプログラムの目的や重視するポイントが変われば、「午前午後」と「時計」は、弱いつながりに設計しなおすべきかもしれません。経験を重ねるとある程度は、使いわけのコツがわかってきますが、それでも悩むケースはでてきます。

結びつき方の強さ・弱さの選択、オブジェクトを準備するタイミング、準備する責任の割り当ては、部品を動的に組み立てるという、オブジェクト指向の考え方の中核の設計課題の1つです。

最初は、なかなか使い分けが難しいと思いますが、違いを意識しながら経験を積んでいくことがオブジェクト指向設計のスキルアップの勘所の1つです。

getter/setterを安易に使わない

Javaでは、getName()、setName()というように、getとsetをペアにしたメソッドを用意することが半ば習慣化しています。

これは悪い癖です。本当に必要になるまでは、できるだけget/setは書かないようにするのが、オブジェクト指向らしいプログラムをするコツの1つです。

リスト6〜13を見てください。getter/setterは1つもありません。

setを使わない理由は簡単です。必要なデー

タは、すべて、コンストラクタで準備してしまうからです。オブジェクトを作ってから、あとからsetでオブジェクトの内部を変更するのは、悪い習慣です。やらないことが基本。

getを使わないのは、意図が明確なメソッド名にしたいからです。getXxxx()にすれば、あまり考えずにメソッド名が見つかります。

それがだめなんです。名前を考えることがオブジェクト指向プログラミングです。名前のつけ方をルール化／パターン化すれば、いちいち考えなくて良くなるから楽、という発想を捨てることが、オブジェクト指向プログラミングのスキルアップのポイントの1つなんです。

安易にgetter/setterを使わないことをぜひ覚えてください。

if文とfor文を減らす

わけのわからないバグで苦しむのは、だいたいが、if文やfor文が入り組んだ場所です。

プログラミングを覚えたての頃は、ある程度複雑なif文やfor文を読んだり書いたりできるスキルを磨くことが必要です。

しかし、if文やfor文をごりごり書くことが、レベルの高いプログラミングではありません。むしろ逆です。

条件分岐をif文を使わずに書くスキル、一見ループが必要な処理もfor文を使わない書き方を覚えること。それがプログラミングの力をつけることなんです。

列挙型（enum）

Javaの列挙型（enum）は、if文の強力な代替手段です。**リスト14**は、列挙型を使って、午前と午後の挨拶文言を宣言しています。

このMessageType.javaを使うと、リスト11のAmPm.javaは、**リスト15**のif文のないAmPm.javaになります。

手品のタネ（?）は、MessageType#valueOf()メソッドですね。

まず、if文以外に、こういう条件分岐の書き方もあることを覚えましょう。Javaの列挙型は入門書ではほとんど取り上げられませんが、Javaでは、if文と同等かそれ以上に重要な条件分岐のプログラミング手段です。

インターフェース宣言と実装クラス

列挙型（enum）を使った例では、MessageTypeオブジェクトが午前の挨拶も、午後の挨拶も、自分で宣言していました。

▼リスト14　MessageType.java

```
package greetingmodel;

enum MessageType
{
  am("おはようございます"),
  pm("こんにちは");

  private String message;

  MessageType(String message)
  {
    this.message = message;
  }

  String message()
  {
    return message;
  }
}
```

▼リスト15　AmPm.java（if文をなくした版）

```
package greetingmodel;

class AmPm
{
  private Clock clock ;

  AmPm()
  {
    this.clock = new Clock();
  }

  String message()
  {
    String ampm = clock.ampm();
    MessageType type = MessageType.valueOf(ampm);
    return type.message();
  }
}
```

もっとオブジェクトを分割して役割分担をしてみたのが、**リスト16**のMessageType(インターフェース宣言と実装クラス版)です。

リスト14では、am()、pm()のパラメータに、Stringオブジェクトを書きました。

リスト16では、Stringオブジェクトの代わりに、それぞれ、MessageAmオブジェクトとMessagePmオブジェクトを作成しています。

異なるオブジェクトを作成していますが、代入先のmessage変数の型は、Messageという別の型になっています。

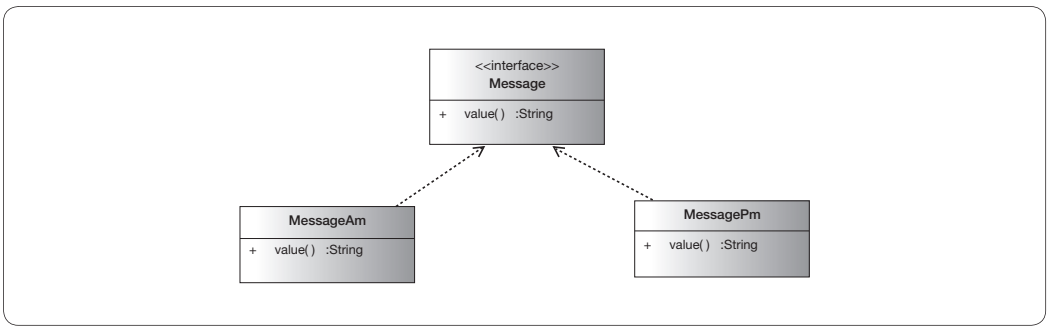
Messageはインターフェース宣言です。MessageAmとMessagePmは、インターフェースMessageを実装したクラスです。

クラス図(**図2**)と**リスト17、18、19**を参考にしてください。

このように、別々の実装クラスのオブジェクトを、使う側(ここではMessageType)からは、同じ型のオブジェクトとして扱えるようにするしくみがインターフェース宣言です。

仕事を頼む側から見ると、同じ型に見えるが、実体は異なるオブジェクトというのが、オブジェ

▼図2　インターフェース宣言と実装クラス



▼リスト16　MessageType.java(インターフェース宣言と実装クラス版)

```
package greetingmodel;

enum MessageType
{
  am(new MessageAm()),
  pm(new MessagePm());

  private Message message;
  MessageType(Message message)
  {
    this.message = message;
  }

  String message()
  {
    return message.value();
  }
}
```

▼リスト17　Message.jav

```
package greetingmodel;

interface Message
{
  String value();
}
```

▼リスト18　MessageAm.java

```
package greetingmodel;

class MessageAm implements Message
{
  @Override
  public String value()
  {
    return "おはようございます";
  }
}
```

▼リスト19　MessagePm.java

```
package greetingmodel;

class MessagPm implements Message
{
  @Override
  public String value()
  {
    return "こんにちは";
  }
}
```

クト指向プログラミングの根幹のしくみの1つです。

オブジェクトとオブジェクトの関係を固定ではなく動的に切り替える変えるしくみとして、異なる実装クラスのオブジェクトを同じ型として使うことができるのは、実に強力なしくみです。

このしくみをきちんと理解し、自信を持って使えるようになることが、オブジェクト指向プログラミングのスキルアップの当面の目標といって良いかもしれません。

デザインパターン

インターフェース宣言と実装クラスのしくみは、GoFの「デザインパターン」を勉強すると良いでしょう。とくにStrategy/Stateパターンは、使う場面も多く必修です。ここで取り上げた列挙型(enum)とインターフェース宣言・実装クラスの例は、Strategy/Stateパターンの簡単な例です。デザインパターンを理解し、使える力を身に着けるのは良いことです。しかし、デザインパターンを積極的に使うことが良いプログラミングとは限りません。上級者は「知っていても使わない」こともあるし「使いどころを適切に判断する」ものです。

コレクションフレームワークAPIに精通する

Javaのコレクションフレームワークは、言語仕様のバージョンアップのたびに、地道に改良されて(まだまだ不満はありますが)、それなりに便利なAPIはそろっています。

コレクションを扱うときに、List<String>＋for文という短絡的な選択ではなく、

・SetやMapを使う
・Comparatorを実装する
・Collectionsユーティリティクラスを使う
・TreeSet、LinkedHashSetを使う

など、設計の選択肢、スキルの幅を広げることが大切です。コレクションAPIをよく勉強してfor文を使わなくてもできることをいろいろ覚えましょう。

ファーストクラスコレクション

コレクションフレームワークのAPIに精通しても、結局for文を使うしかない場面が多いのが現実です。そのときに、List<String>とかSet<String>とかを、操作するfor文があちこちのクラスに散在するのは失敗パターンです。コレクションをfor文で操作することは、バグの入り込みやすい場所です。また、思わぬ場所で副作用を起こしやすいしくみです。

バグや副作用を減らすコツは、List<Order>などのコレクション変数を1つだけ持った専任オブジェクトに、コレクションの管理をさせることです。「ファーストクラスコレクション」という設計パターンの1つです。クラス名は、Ordersなどの複数形か、OrderListなどコレクションを明示した名前です。

for文をファーストクラスコレクションクラスに集めると、思わぬ場所の副作用が防げます。また、似たようなfor文を整理することで、バグの少ないコードに改良できる機会が増えます。

どうしても、コレクションそのものを公開したい場合は、Collections#ummodifiableList()などのメソッドを使って、変更不可のコレクションオブジェクトを渡すのが良いプログラミングの習慣です。

オブジェクト指向の基本スキル

Javaを使ったオブジェクト指向プログラミングをコードサンプルで説明してみました。

列挙型、インターフェース宣言、ファーストクラスコレクションは、初心者には、ちょっと難しい内容かもしれません。

しかし、オブジェクト指向の基本という意味で、列挙型、インターフェース宣言、ファーストクラスコレクションは、必修科目だと思います。

複雑なif文やfor文を書くことが、オブジェクト指向プログラミングの基本スキルではないんです。

オブジェクト指向の学び方

オブジェクト指向プログラミングとは、モデリング(分析)、設計、そしてリファクタリング(設計の改良作業)が、混然一体となったものです。分析は分析、設計は設計、プログラミングはプログラミング、という発想・やり方だと、オブジェクト指向の良さは出てきません。

良い名前やオブジェクトの役割分担を考えるのは、分析やモデリングの作業です。

動くものを作ってからがプログラミングの本番です。地道にリファクタリングを繰り返し、より良い設計を模索することが、オブジェクト指向プログラミングの習慣であるべきです。

オブジェクト指向プログラミングを学ぶためには、分析やモデリング、リファクタリングを積極的に学ぶべきなんです。

本を読む

オブジェクト指向のスキルアップには、本を読むことです。プログラミングの技術書だけではなく、小説、新書、各分野の入門書や専門書。本であればなんでも良いです。英語など外国語まで手を出せればいうことはありません。本は、企画から始まって、情報の収集と整理、全体構成の検討、読み直しと書き直し、など、たいへんな知的な活動の成果物です。そういう知的活動の成果に触れることで、語彙が増え、部品から全体を組み立てる構成力がアップしていきます。

オブジェクト指向プログラミングは、数式を組み立てるよりも、文章を組み立てる活動と似ています。頭の中にある漠然としたイメージを適切な言葉を選びながら、だんだん論理的な文章に組み上げていく。それがオブジェクト指向プログラミングです。

オブジェクト指向プログラミングのスキルアップには、本を読んで、文章力、国語力をアップすることが一番なんです。

クラス図のラフスケッチ

プログラミングは、実際にコードを書くことが一番の学習方法です。オブジェクト指向プログラミングは、さらに、オブジェクトの役割分担を考えたり、名前を考えるスキルアップが重要な学習テーマです。オブジェクトの役割分担は、コードだけで考えるよりも、図1や図2のようなクラス図をいろいろ描いてみるほうが効果的です。

ツールを使ってきっちりとしたUML形式で描く必要はありません。手書きでラフスケッチするほうが簡単だし、いつでもどこでも練習ができます。プログラミング課題を考えるとき、どんな部品をどうやって組み立てるかを考えるのは、クラス図のラフスケッチから始めるのが良いでしょう。コードを書きながら、ぐちゃぐちゃしてきたら、クラス図をラフスケッチしてみることも習慣にしたいものです。

参考書

オブジェクト指向プログラミングのスキルアップの参考書として、手元に置いて活用してほしいのが、『リファクタリング』(マーチンファウラー著、ピアソンエデュケーション)です。

本書のリファクタリングパターンを丁寧に実践すると、オブジェクトは小さく分割され、各オブジェクトの役割が単純になる傾向があります。

『実装パターン』(ケント・ベック著、ピアソンエデュケーション)も良い本です。クラスやメソッドに、なぜ、どういう名前を付けるべきかのガイドラインがたくさん書かれています。

Java言語のしくみを活用するには『Effective Java　第2版』(Joshua Bloch著、ピアソンエデュケーション)は必読でしょう。

オブジェクト指向設計を学ぶには『ドメイン駆動設計』(エリック・エバンス著、翔泳社)がすばらしい名著です。難解と言われるこの本を読み解くには『オブジェクトデザイン』(レベッカ・ワー

フスブラック著、翔泳社)と『実践UML　第3版』(クレーグ・ラーマン著、ピアソンエデュケーション)を読んでおくと良いでしょう。

ここで挙げた本は、いずれも古典的な名著で、読むのがたいへんな本ばかりです。

最初はざっと読んでみて、必要に応じて何度も読み返し、末永くお付き合いすると良い本ばかりです。

「創造的な学び」をデザインする

オブジェクト指向プログラミングに限らず、ソフトウェア開発は学ぶべきことがやまほどあります。

しかも、誰かが丁寧に教えてくれるわけではありません。自分自身で目標を設定しながら、自分の力で学び続けることがソフトウェア技術者の一番大切な習慣かもしれません。

技術を学ぶときに、ぜひ参考にしてほしいのが、慶応大学SFCの井庭研究室が作成した、

学習パターン(http://learningpatterns.sfc.keio.ac.jp/)です。

同じ学習をするなら、楽しく、効果的に学びたいものです。

学習パターンは、技術者の創造的な学びのヒントやコツが満載されています。

プロトタイピング

プログラミングのテクニックやノウハウは作ってみてはじめてわかることばかりです(**図3**)。

技術書を読むことは大切ですが、ただ読むだけでなく、実際にコードに書いてみて初めてわかることがたくさんあります。

本を読みながら、実際に作ってみるを繰り返すのがプログラミングの学び方のコツです。

「まねぶ」ことから

まずは真似てみましょう。たいせつなのは「真」に似せることです(**図4**)。

表面的に同じにするだけでなく、考え方、そこに至った背景を思い浮かべながら、真似てみることが大切です。

思考を停止した「写経」や、ネット上からなにも考えずにコピー&ペーストするのは「まねぶ」ことではありません。

広がりと掘り下げの「T字」

学習パターンの中で、筆者が一番お気に入りのパターンです。

▼図3　プロトタイピング(Copyright©2009、慶應義塾大学湘南藤沢キャンパス学習パターンプロジェクト)

▼図4　「まねぶ」ことから(Copyright©2009、慶應義塾大学湘南藤沢キャンパス学習パターンプロジェクト)

▼図5　広がりと掘り下げの「T字」（Copyright©2009、慶應義塾大学湘南藤沢キャンパス学習パターンプロジェクト）

・何かを掘り下げようと思ったら、幅広くいろいろなことに目を向ける
・いろいろなことを知りたかったら、1つのことをじっくり掘り下げる

これが技術を学ぶ奥義なんだと思います。

特定の技術だけを追いかけて視野が狭くなると、その技術をほんとうに理解することはできません。

さまざまな技術に手を出すだけでは、ほんとうの技術は身につきません。

「広がり」と「掘り下げ」を並行して追いかけ続けることが、結局は、両方とも手に入れることになるわけです（**図5**）。

鳥の眼、虫の眼

技術書を読むときや、大きなソフトウェアを設計するときに、このパターンはほんとうに大切です。

全体を眺める鳥の眼と、部分を詳細に見る虫の眼を両方持つこと（**図6**）。たいせつなのは、両方の見方を、しょっちゅう行ったり来たりすること。

▼図6　鳥の眼、虫の眼（Copyright©2009、慶應義塾大学湘南藤沢キャンパス学習パターンプロジェクト）

オブジェクトの集まりを全体として眺める図1のクラス図の見方が鳥の眼。リスト1のように、実際のコードの1行1行を追いかけるのが虫の眼。

どちらの眼も必要です。そして、両方の眼で、いったりきたりすることが、一番大切なことです。

バグ修正でコードを追いかけはじめると虫の眼だけになって、鳥の眼を忘れがちです。意識的に鳥の眼と虫の眼を切り替えながら仕事をすることを習慣にしたいものです。

終わりに

オブジェクト指向の発想、プログラミング例、技術の学び方を書いてみました。

技術を覚えて、何かを作り上げることは、それ自体、とても楽しいことです。仕事ですから、つらいこともありますが、モノづくりの楽しさ、技術がわかることの面白さが動機になって、筆者はこの仕事を続けているんだと思います。

みなさんが、楽しみながら、技術を覚えモノづくりに取り組むことに、筆者の書いたことが少しでもお役にたてれば幸いです。SD

part 2

オブジェクト指向の学び方、教え方

オブジェクト指向は、ほんの少しのきっかけで、大きく理解が進むことがあります。Part2では、少し高い視点から見直してみます。おさえるべき情報・知識をまとめて概観し、学ぶべきこと、もしくは先輩として教えるべきポイントを確認してみませんか。

青山幹雄（あおやまみきお）　南山大学 情報理工学部ソフトウェア工学科 教授

オブジェクト指向はなぜ「難しい」か？

オブジェクト指向を学ぶこと、教えることを「難しい」と感じている人は少なくないと思います。

筆者がオブジェクト指向という言葉に出会ったのは、約30年も前の1980年代初めでした。その表紙の色からブルーブックと呼ばれるSmalltalk-80の本を買って、毎月1回土曜日の午後に会社の仲間で輪講をしました。筆者の担当はメタクラス階層でした。今でもよく覚えているのは、日本語に訳しても何のことか意味がわからなかったことです。ただ、新しい技術への皆の好奇心と熱意が推進力となり、1年間輪講を続け、読み終えることができました。

その後、オブジェクト指向の普及のために社内の講習などにも関わってきました。大学に移ってからは、オブジェクト指向プログラミングなどの科目で学生にオブジェクト指向を教えたり、企業の技術者向け講習を長年続けてきました。その場合、たとえば、C言語で開発していた人が「難しい」と反応されることが少なくありません。

では、なぜ、オブジェクト指向を学ぶ、あるいは、教えるのは、「難しい」と思われているのでしょうか？　その理由は、次の3点にあると思います。

①オブジェクト指向技術の障壁が高いこと

オブジェクト指向は、プログラミング技術やソフトウェア工学の発展の上に築かれた技術です。したがって、オブジェクト指向を理解するためには、これらの基礎技術を学ぶ必要があります。そのため、難しく感じると思います。

②オブジェクト指向の広がり

一口にオブジェクト指向と言っても、分析・設計とプログラミングでは、学ぶべき内容が違います。ソフトウェア開発にオブジェクト指向を適用するには、上流工程の分析からオブジェクト指向を適用することが望ましいのです。しかし、分析・設計とプログラミングでは行う内容が異なるので、学ぶべき内容も広範囲となります。

③手続き指向からの発想の転換

多くの技術者は、最初にC言語などの手続き指向のプログラミング言語を学び、構造化分析・設計などの機能中心の分析・設計をまず学び、経験を積んでいることが、オブジェクト指向を学ぶうえで、逆に、障壁になっているように思います。筆者も同じでした。学生時代や企業に就職して最初に学んだ言語はFORTRANやCOBOL、アセンブラ言語でした。そのため、オブジェクト指向の考えにすぐには馴染めなかったのです。

このような、オブジェクト指向を学ぶうえで、障壁となる点をまず知って、学習のポイントを押さえておくことは、重要です。戦いに勝つには、まず、敵を知ることです。ただ、ここで強調し

ておきたいのは、オブジェクト指向は強力な技術であり、その技術を使いこなすことが、技術者としての実力を高めることになることです。

さらに、オブジェクト指向を学ぶためには、学び方、教え方に技術が必要だということです。高い山に登るには、近所の低山を登るのとは違う準備や装備が必要です。オブジェクト指向を学ぶことも同じです。良く準備をし、学び方を工夫する必要があります。そうすれば、迷わず山頂にたどり着け、絶景を楽しめることでしょう。

オブジェクト指向への地図を持とう

オブジェクト指向技術へ至る簡略な地図を**図1**に示します。この地図は、技術の進化を表しており、技術マップと呼ばれます。この地図には、オブジェクト指向の考え方、C言語などの手続き型プログラミング言語との違い、オブジェクト指向が基礎とする技術を示しています。オブジェクト指向に限りませんが、技術を学ぶ、あるいは、教える上で重要なことは、その根本原理や考え方を理解することです。表面的な知識だけでは、本当の意味での理解に至らないのです。

原理

オブジェクト指向に限らず、すべてのプログラムはアルゴリズムとデータ構造からなっています。

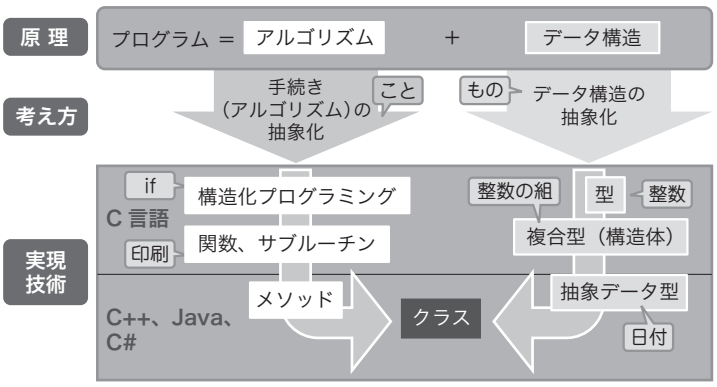
考え方

CにしろJavaにしろ、現代のプログラミング言語は、アルゴリズムとデータ構造の表現を高度化することにより表現能力を高め、ソフトウェアの生産性や品質の向上を目指しています。表現の高度化とは抽象化することです。コンピュータの処理の詳細を知らなくても、やりたいことを書けば実行してくれることです。たとえば、アルゴリズムとデータ構造は、日常生活での物事の「こと」と「もの」にそれぞれ対応します。「もの」はプログラムではデータで表され、それを用いて「こと」を成すのです。

実現技術

実現技術はC言語などの手続き型プログラミング言語とオブジェクト指向プログラミング言語との違い、言語を実現する技術を表します。

▼図1　オブジェクト指向の考え方



C言語では、次の2つの技術を用いています。

①関数による手続きの抽象化

関数によって、一連の手続きを1つの関数に抽象化できます。関数を利用する人は、関数の内部がどのようになっているか知らなくても使えます。

②型によるデータの抽象化

整数や文字列などのデータの型は、データの分類を表します。同じ型のデータは同じ性質をもちます。

まず、扱うデータの型を定義することが重要です。あらかじめ言語で定義された、ビルトインデータ型では複雑なデータ構造の表現が困難です。そのため型を組み合わせてより高度な型を定義する必要があります。たとえば、整数型のYear、Month、Dayを組み合わせて日付を表す場合です。C言語では構造体を用いますが、外部から中のデータを変更できてしまいますので、データが破壊される恐れがあります。

オブジェクト指向では、データの型の抽象化を図りました。これを抽象データ型(Abstract Data Type)と呼びます。オブジェクト指向プログラミング言語は抽象データ型を基礎としている言語と言えます。では、なぜ、抽象データ型でしょうか？　それは、現実のものがデータで表現されるからです。しかも、データこそが継続して保存されるべきであるからです。

オブジェクト指向の意義

抽象データ型には、互いに関係する、次の5つの意義があります。

①現実世界のものに対応した高度な、すなわち、抽象的なデータ型を定義できる

日付や社員、学生など、複合型を一つの型として定義できます。

②ソフトウェア設計者がデータ型を定義できる

抽象的なデータ型を定義できることは、あらかじめ言語で定義した型を組み合わせて、ソフトウェアの設計者、プログラマが型を定義できることを意味します。

③データ型を処理するメソッドを定義できる

プログラムに限らず、計算内容はデータ型によって決まります。たとえば、1＋1＝2の＋は「整数の加算」の意味です。それは、整数という型がまずあって、それに従って、「整数の加算」が決まります。同じように、抽象データ型が決まると、その計算方法が決まります。たとえば、日付型を考えてみます。日付型の2つの変数、Day1とDay2の差Day1－Day2は整数型の日数として定義できます。このように、データ型こそがプログラムの中心となることを理解することが重要です。

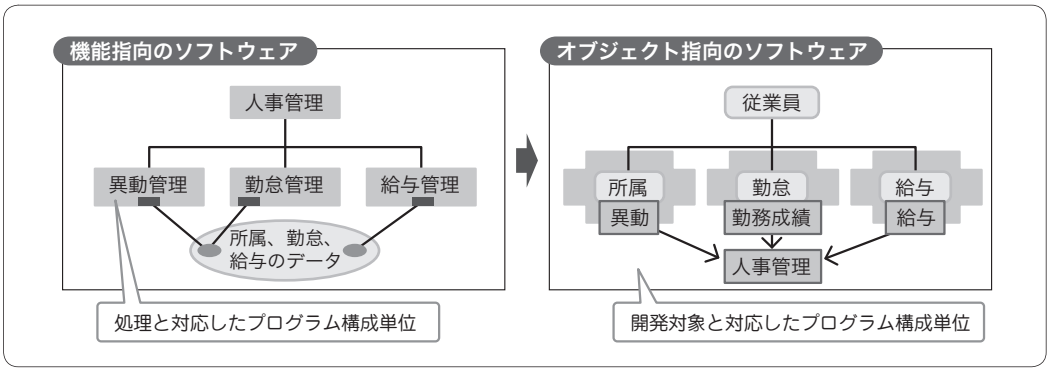
④データ型とその処理をまとめてプログラムの単位とできる

これまでの説明から、データ型を中心に、その処理をまとめることがプログラムの合理的な単位となります。オブジェクト指向では、これをクラス(Class)と呼びます。データ型が実世界のものを分類(classify)した結果を表していますので、それによってできたプログラムの単位をクラスと呼ぶことにしたのです。処理はデータ型へクラスの外部からアクセスする方法となりますのでメソッド(Method)と呼びます。クラスのデータ型は、このクラスの性質を表すので、属性(Attribute)と呼びます。このように、クラス、メソッド、属性などの名称には意味があります。

⑤クラスは安全で独立性の高いプログラム単位となる

データ型はメソッド、すなわち、インター

▼図2　機能指向からオブジェクト指向へ



フェースのみを介して利用できるので、C言語の構造体のように直接外から操作できません。さらに、データ構造を変えてもインターフェースを変えなければ、その変更はほかのクラスに影響しません。この結果、クラスは安全で独立性が高くなります。さらに、ソフトウェアが良い構造となり、変更などが容易になります。

オブジェクト指向プログラミングの発想「アーキテクチャが鍵」

JavaやC++、C#といったオブジェクト指向プログラミング言語はC言語から進化した、C言語の仲間です。プログラム内の処理はC言語の処理から受け継いでいるので、よく似ています。とくに、C言語など手続き型プログラミング言語を学んだ後にオブジェクト指向プログラミング言語を学ぶことが多いので、その違いを理解することはオブジェクト指向プログラミング言語を理解するうえで有効です。

C言語との違いで理解してほしい点は、文法の細部ではなく、オブジェクト指向プログラミング言語の背後にある考え方です。この考え方を理解しないと、Javaで記述したC言語のプログラムのようなオブジェクト指向とは言えない、プログラムになってしまうことになりがちです。

図2に、企業の人事管理システムのソフトウェアを手続き型、すなわち、機能指向とオブジェクト指向で実現する構造の例を示します。これをアーキテクチャと呼びます。

機能指向では、機能である異動管理や給与管理などを中心にソフトウェアを構成します。従って、社員のデータはデータベースなどで管理され、共用されます。一方、オブジェクト指向では、所属、給与などのデータを中心に、各データを処理するメソッドをデータと一緒にしてソフトウェアを構成します。ソフトウェアアーキテクチャの考え方がまったく逆になっています。まず、アーキテクチャの違いを理解する必要があります。

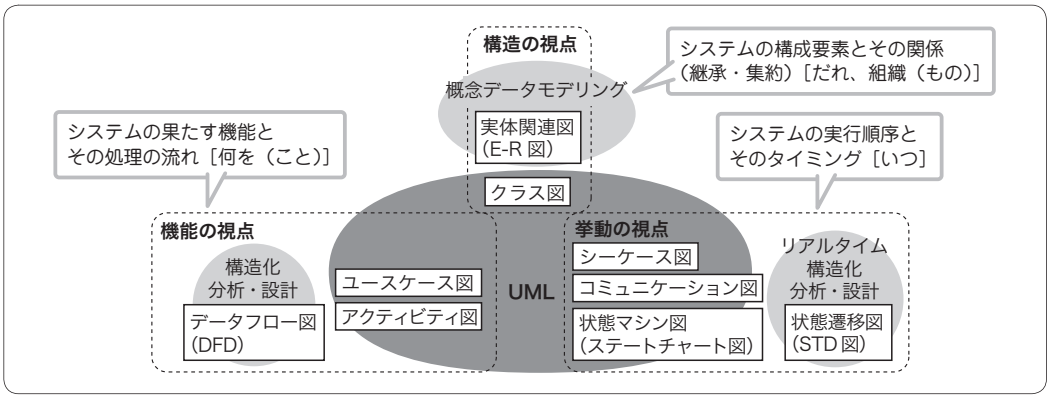
オブジェクト指向では、クラス間の関係を継承や集約によって関係づけできます。また、抽象クラスによって、上位の階層の構造を規定できます。これらの技術は、アーキテクチャを構成する技術です。したがって、オブジェクト指向により、良いアーキテクチャを実現できます。

ただし、オブジェクト指向がすべて良いわけではありません。データを中心とすることは処理の流れをわかり難くします。これを機能の断片化と呼びます。したがって、組込みソフトウェアのように処理に時間制約があるソフトウェアでは、処理を制御できる必要があります。そのため、制御クラスと呼ぶ、ある範囲のクラスを制御するクラスを置くことがあります。

オブジェクト指向分析・設計を学ぶ

これまでの説明はオブジェクト指向プログラミングが中心でした。しかし、ソフトウェア開

▼図3　複数視点のモデル化とUML



発のプロセスを考えれば、上流工程からオブジェクト指向により開発すべきです。そのために、オブジェクト指向分析・設計の技術があります。ただ、多くの場合、オブジェクト指向によるモデル化の言語であるUML（Unified Modeling Language）の習得が中心となっているように思います。UMLは図で表現しますが、日本語と同様、言語ですので、UMLを用いて良いモデルを構築することが重要です。良いモデルは良いアーキテクチャへつながります。

モデル化の手段は抽象化です。しかし、実世界は極めて複雑ですので、抽象化することは難しい。そのため、複数の視点に分けることが基本です。建築や機械などは3次元の物体ですので、2次元の図面を3つ用いて表します。これと同じです。情報システムの場合、**図3**に示す次の3つの視点が基本です。

① 構造の視点
② 機能の視点
③ 振舞の視点

この3つの異なる性質を表現する視点に分けることによって複雑な現実世界を正しく表現し、理解できるようになるのです。UMLはこの複数視点の考えに基づき設計された図形言語です。

オブジェクト指向を使いこなす

オブジェクト指向を使いこなすためには、教える人、学ぶ人、ともに、その基本となっている考え方を理解することが重要です。そのような考え方が参考文献[1]にまとめられています。

さらに、実際に分析・設計やプログラミングができるためには、自ら考え、手を動かして課題に適用してみる必要があります。図書館やレンタルショップなどは、よく用いられる例題です。ぜひ、自分で試してみてください。

さらに、オブジェクト指向の技術は、これまでのソフトウェア工学の技術をその中に取り込んでいることも理解しておく必要があります。たとえば、UMLのユースケース図、シーケンス図、状態マシン図はオブジェクト指向とは異なる分野で開発されたものです。ソフトウェア工学を学ぶことも理解の助けになります。

また、デザインパターンやリファクタリングなど、オブジェクト指向と関連する技術も実際に役立つと思います。

読者の皆さまがオブジェクト指向を使いこなして、ソフトウェア開発が楽しくなることを期待しています。SD

●参考文献
[1]青山幹雄、中谷多哉子（編著）、オブジェクト指向に強くなる、技術評論社、2003.

組み込みからクラウドまで、オブジェクト指向は隅々と!

part 3

Part3では、現在では当たり前過ぎて話題にも上がらないオブジェクト指向が、組み込み系や業務系のソフトウェア業界中で現在はどんな位置付けになっているのか見ていきましょう。

㈱豆蔵　井上樹（いのうえたつき）

はじめに

先日本誌の総集編が手元に届きました。懐かしく昔の記事を見ていたのですが、思い返してみれば、筆者が初めて本誌にオブジェクト指向の記事を書いたのは、もう10年、いや、20年近く前になります。その間、オブジェクト指向は知る人ぞ知るような存在から、当たり前過ぎて話題にも上がらない状態まで、ソフトウェア業界に広まって行きました。ではそんなオブジェクト指向が、ソフトウェア業界の中で現在はどんな位置付けになっているのか、ここで見ていってみましょう。

オブジェクト指向の現在 ～組み込み系の世界

まずは組み込み系の世界でのオブジェクト指向から行きましょう。組み込み系の世界でオブジェクト指向が開発の選択肢として現実的になってきたのは、ここ10年のことです[注1]。すべての開発がオブジェクト指向という状態ではないですが、今では、新規開発の際や、作り直しの際に、オブジェクト指向を導入するという選択肢が出るようになっています。ではこの10年でどんな変化があったのでしょうか。

もともと、オブジェクト指向は組み込み系と相性の良い技術です。たとえば、モータというデバイスで考えてみましょう。モータには、ステッピングモータやサーボモータのようにいろいろな種類のものがあります。ですが、どのモータもソフトウェアから見たときにやりたいことは、回転と停止といった共通なことです。これを何も考えずC言語で設計すると、**図1**のような設計になってしまいます。ですが、オブジェクト指向をうまく活用すると**図2**のように設計できます。

図1のようなC言語の設計は、利用者はデバイスがサーボモータなのか、ステッピングモータなのかを理解して関数を使い分けなければなりません。また、新しい方式のモータが出てきた場合、それ用の関数一式を新たに追加しなければならず、利用者はそれを理解していないと、新しいモータを使うことはできません。

一方、図2のようなオブジェクト指向設計であれば、モータを利用するためのインターフェースがモータクラスによって統一されているので、利用者はモータの種類が違っても同じ操作名で制御できますし、今後新しい種類のモータが出てきたとしても、モータを利用するソフトウェア側に大きな変更は必要ありません。このような「デバイスの抽象化」を行うことで、より自然な考え方でソフトウェアを開発できるようになります(**図3**)。

派生開発

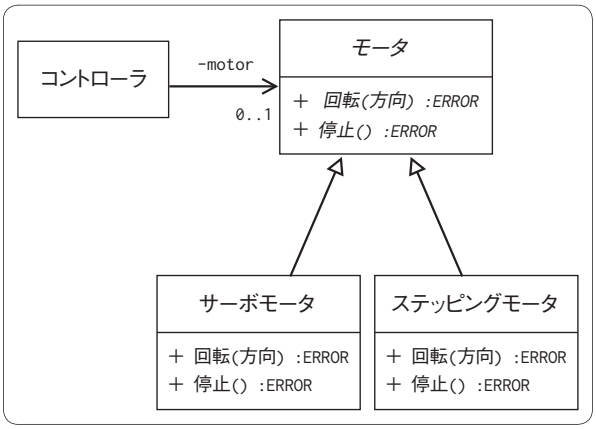
また、組み込み系のソフトウェアは「派生開発」

注1) 組み込み系でのオブジェクト指向の取り組み自体は90年代からありますが、広まり始めたのはここ10年くらいです。

▼図1　C言語での設計

```
ERROR サーボモータ_回転(方向 dir) {
  /* サーボモータの回転処理 */
}

ERROR サーボモータ_停止() {
  /* サーボモータの停止処理 */
}

ERROR ステッピングモータ_回転(方向 dir) {
  /* サーボモータの回転処理 */
}

ERROR サーボモータ_停止() {
  /* サーボモータの停止処理 */
}
```

▼図2　オブジェクト指向を使った設計



▼図3　モータ制御時のプログラムの違い

・Cの場合

```
ERROR result;

/* サーボモータ制御 */
result = サーボモータ_回転(時計回り);
result = サーボモータ_停止();

/* ステッピングモータ制御 */
result = ステッピングモータ_回転(時計回り);
result = ステッピングモータ_停止();
```

プログラマは使っているデバイスに応じて、どの関数を使うか意識しないといけない

・C++の場合

```
ERROR result;
モータ* motor;

/* サーボモータ制御 */
motor = new サーボモータ;
result = motor->回転(時計回り);
result = motor->停止();

/* ステッピングモータ制御 */
motor = new ステッピングモータ;
result = motor->回転(時計回り);
result = motor->停止();
```

デバイスが違っても同じように利用できる

といって、ベースになるソフトウェアを拡張・改造しながら次の製品を作っていく方法が主流です。派生開発をきちんと行うためには、開発時に既存機能を壊すことなく新しい機能を追加できたり、ほかの機能に影響を出さずに既存機能の修正ができるような設計になっていることがソフトウェアに求められます。そうした設計は、オブジェクト指向の得意なところです。

このように、オブジェクト指向にピッタリの組み込み系なのですが、組み込み系といえば、メモリも処理能力も潤沢にある一般的なコンピュータの世界と異なり、ギリギリまで下げた処理能力、メモリ量で開発を行うのが当たり前の業界です。2013年の今でも8ビットのCPUが現役の業界です。そんな世界で、C言語[注2]で開発した場合と比べて、処理速度が遅かったり、メモリを多く必要とするオブジェクト指向は、性能が出ない／メモリに収まらないという物理的な理由で、なかなか導入が進みませんでした。

組み込み系ソフトの大規模化

しかし、そんな組み込み系のソフトウェアの世界も、一般的なソフトウェアの世界と同じように年々ソフトウェアの大規模化が進み、数百万行規模のソフトウェアで派生開発をやっていかなきゃいけないという状況になってきました。ここで、C言語で開発を行うときに採用される構造化手法(機能分割でソフトウェアを考えて

注2）ハードウェア制御を行う組み込み系ソフトウェアの世界では、プログラムからメモリ操作が可能なC/C++が開発の中心です。AndroidはJavaだという声も聞こえてきそうですが、Androidでもハードウェア制御まで行おうと思ったら、C/C++が必要になります。

いく方法)は継続的な変更に弱い手法のため、大規模な派生開発にも耐えられる、オブジェクト指向を導入したらどうかという流れが生まれました。そして幸いなことに、ときを同じくして、高性能のチップやメモリが安くなり、C++コンパイラの性能もよくなってきたので、処理速度やメモリに余裕が生まれ、オブジェクト指向を導入するための物理的な問題も解決され始めました。そうして、組み込み系へのオブジェクト指向導入が進み始めました。これがここ10年の話です。

そして、さらにハードウェア性能が向上してきた昨今では、ハードウェアをOSやプラットフォームやインタプリタで抽象化して、そのうえでソフトウェアを走らせても、問題ない速度でアプリケーションが動くようになってきました(**図4**)。それにより、組み込み機器向けのアプリケーションをJavaやJavaScriptといった、より扱いやすい(開発しやすい)プログラミング言語で開発できるようになりました(ちなみにFirefoxOSではJavaScriptでハードウェア制御まで書けます)。こうした変化を受け、開発の現場では、業務系のソフトウェア開発者が組み込み系にやってくるという現象も起きています。

組み込み系での今後のオブジェクト指向

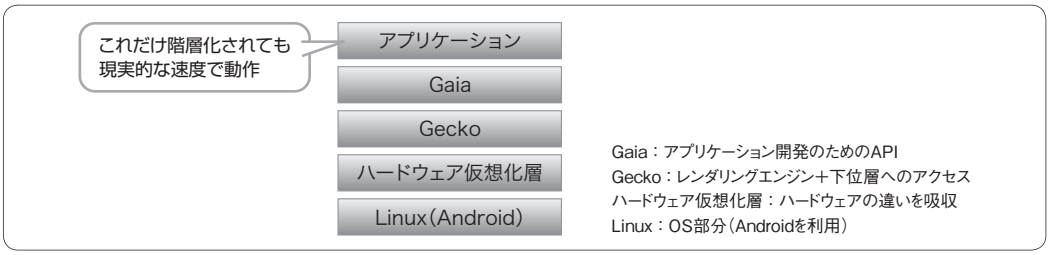
では今後、組み込み系の世界でオブジェクト指向はどのような位置付けになっていくのでしょうか。筆者の個人的な見解ですが、組み込み系の特徴である派生開発をやっていくうえで、オブジェクト指向は外せない技術です。さらに、組み込み系は後述の業務系の世界のような汎用的なフレームワークの組み合わせでソフトウェアを作るのではなく、問題領域に合わせて、自身でプラットフォームやフレームワークを作っていくことが求められる世界です。ですので、組み込み系の世界でのオブジェクト指向は、今後ますますすべての開発者が身に付けておかなければならないソフトウェア設計のための中心技術になっていくと考えられます。

オブジェクト指向の現在 ～業務系の世界

次は業務系の世界(乱暴ですが、基幹系やWeb系やクラウドもみんな合わせて、組み込み以外のソフトウェアをまとめて、ここではそう呼びます)でのオブジェクト指向の現在を見てみましょう。こちらの世界では、基幹系の業務システムの一部を除けば、もうオブジェクト指向は当たり前過ぎて、開発者の間で、ことさらオブジェクト指向という言葉も出てこないような状況になっています。業務系の開発で使われている主要なプログラミング言語(C#、Java、JavaScript、Ruby、Python、PHPなどなど)を見ても、当たり前のようにオブジェクト指向が取り入れられています[注3]。

また、Gitなどのオープンソースプロジェクトのアーキテクチャを解説した、"The Architecture of Open Source Applications"[注4]を見ると、いくつかのプロジェクトでは、何の説明もなく、いき

▼図4　FirefoxOSの構造



注3) ScalaやErlangといった関数型言語の流れもありますが、現時点では主要なプログラミング言語とは言えないので、ここでは割愛します。
注4) http://www.aosabook.org/en/

なりUMLで構造や振る舞いが説明されており、説明しなくても大丈夫なぐらいオブジェクト指向やUMLが使われているんだなというのがわかります(**図5**)。

また、こうした業務系の世界のソフトウェアを開発するときは、今ではたいてい、フレームワークを利用して(ときには複数組み合わせて)、開発が行われていると思いますが、そもそも、この「フレームワーク」や「アプリケーションフレームワーク」という考え方自体が、オブジェクト指向がなければ成立しません。

といったように業務系の世界はオブジェクト指向が当たり前であり、また、その状態で発展が続いているので、オブジェクト指向前提で新しい技術が出てきます。そのため、業務系の世界で最新の技術動向を押さえていくためには、オブジェクト指向をきちんと理解していることが必要になります。

プログラミングパラダイム

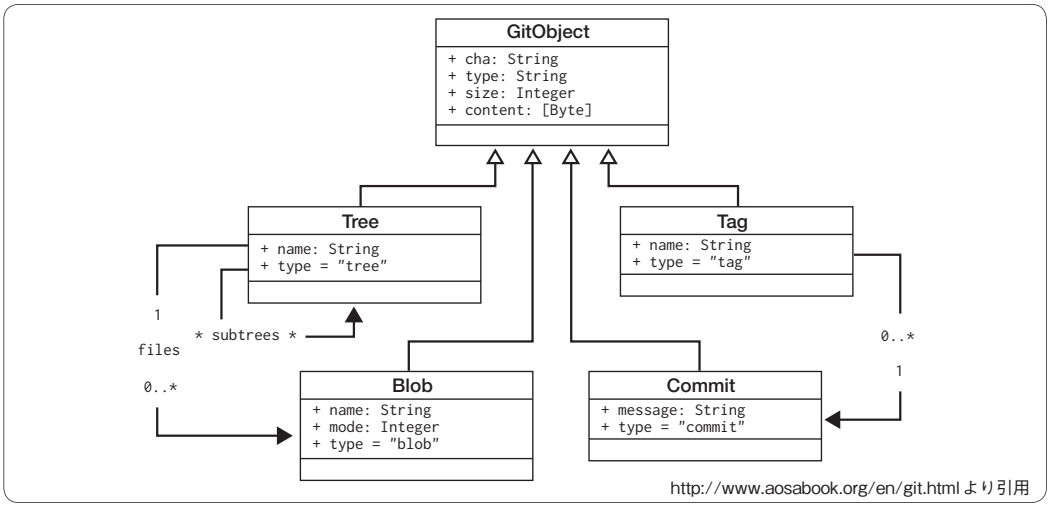
たとえば、アスペクト指向というプログラミングパラダイム[注5]がありますが、アスペクト指向はオブジェクト指向の弱点を補うために出てきた技術ですので、アスペクト指向を理解するためにはオブジェクト指向の弱点を知らないと正しい理解や活用につながりません。

ほかにも、プログラミングパラダイムといえば、ScalaやErlangといった関数型プログラミング言語がここのところ注目されていますが、オブジェクト指向をきちんと理解していれば、関数型プログラミング言語のメリット/デメリットがわかり、オブジェクト指向と関数型の使い分けができるようになります。とくにScalaはオブジェクト指向と関数型のハイブリッドな言語なので、Scalaを使いこなしたいと思っているのであれば、関数型だけでなく、オブジェクト指向の知識も持っていたほうが良いです。

また、ゲーム開発等で注目されているUnityという環境がありますが、この環境では、画面に出ているものすべてがオブジェクトという考え方で、ゲーム自体もオブジェクト指向で設計されたアプリケーションフレームワーク上に構築していきます。こうした環境では、オブジェクト指向を知っているのと知らないのとでは、開発効率に大きな差が出てきます。

さらに、最近注目されているSysMLというモデリング言語があります。SysMLは名前からわかるようにUMLから派生したモデリング言語で、シス

▼図5　Gitのオブジェクト構造を表したモデル(UMLに似た表記)



http://www.aosabook.org/en/git.html より引用

注5) プログラミングパラダイムとは、プログラムの考え方です。たとえば、C言語は構造化、C#やJavaはオブジェクト指向、Erlangは関数型、Prologは論理型というプログラミングパラダイムに属します。

テムズエンジニアリング（複数のシステムを組み合わせて新しい「超システム（SoS：System-of-Systems）」を構築していく世界）の領域で使われる技術です。データセンターを構築するような仕事では、役割の異なる多数のサーバを組み合わせて、1つの大きなシステムを組むことになりますが、SysMLはそういった領域で今後使われるようになります。このSysMLもモデリングの基本的な考え方はオブジェクト指向です。

このように業務系の世界は、表立ってオブジェクト指向が出てくることはほとんどないですが、いたるところの背後にオブジェクト指向が潜んでいます。

業務系での今後のオブジェクト指向

では今後、業務系の世界でオブジェクト指向はどのような位置付けになっていくのでしょうか。組み込み系の世界と同じく筆者の個人的な見解ですが、前述のように業務系の世界はフレームワークなどで、開発者にとってめんどくさいところは隠蔽され、より問題領域に注力してアプリケーションが作れるようになっていっています。ですので、標準的なアプリケーション開発をするだけであれば、オブジェクト指向は知らなくても問題ない世界になっていくと思います[注6]。

しかし、最新の技術を追いかけたい、新しい技術（新しい“サービス”ではないですよ）を作り出していきたい、ほかの開発者と差別化したい、問題領域をうまくとらえたいと考えているなら、オブジェクト指向は必須です。とくに、新しい技術を追いかけて行くのであれば、オブジェクト指向は現在のソフトウェア技術の根幹をなす考え方ですので、一度オブジェクト指向をきちんと勉強しておくことをお勧めします。

さいごに

組み込み系と業務系のオブジェクト指向の位置付けについて見て来ましたが、いかがだったでしょうか。筆者がコンサルティングの現場で見ている範囲のことですので、相当に主観的かもしれませんがご容赦ください。主観ついでに最後にもう1つ話をすると、ここまで、オブジェクト指向は普及している、当たり前になっているという切り口で書いて来ましたが、現在の開発の現場でオブジェクト指向が活用されているかというと、疑問が残ります。

確かに、オブジェクト指向を実践する環境はそろって来ましたし、オブジェクト指向プログラミング言語を使った開発も一般的になってきました。しかしながら、現場で作られているソフトウェアの設計やコードを見ていると、オブジェクト指向を使ってはいるけど、オブジェクト指向らしくないものを多く見ます。率直な言い方をすれば、オブジェクト指向プログラミング言語を使って機能分割中心の設計を行っていることが多いです。つまり、オブジェクト指向を活用できている人はまだまだ少数派といえます。

たとえば、クラスとインスタンスの使い分け、適切なカプセル化、情報隠蔽による疎結合な構造の実現、ポリモルフィズムを使ったフレームワーク化、といったことができている人は本当に少ないです。

また、組み込み系／業務系共通ですが、オブジェクト指向を使った要求分析ができる人も少数派です。オブジェクト指向を活用するためには要求～設計～実装と一貫してオブジェクト指向を利用するのが理想なのですが、各工程の目的に合わせてオブジェクト指向を使い分けられている人は、オブジェクト指向設計ができる人以上に少ないです。

ただこれは、裏を返せば、それだけ一味違うエンジニアになる余地がまだまだあるということですので、ぜひ、オブジェクト指向をきちんと学び、オブジェクト指向を活用できるエンジニアを目指して下さい。SD

注6）高級化、抽象化能力の向上というプログラミング技術の進化の方向性からすれば、正しいと思います。

4 part JavaScriptでオブジェクト指向

現在主流となっている多くのオブジェクト指向言語では、クラス定義を積み重ねてシステムを記述するようになっています。よって、設計や実装といった開発の大半がクラスで行われることになり、その名前に反してオブジェクトを意識する場面は少ないのではないでしょうか。JavaScriptへようこそ。オブジェクトがオブジェクトの振る舞いを決める、異色のオブジェクト指向の世界にご案内します。

㈱日立ソリューションズ　川尻剛（かわじりたけし）

オブジェクトとアイデンティティ

まずは「オブジェクトとは何か」というところから始めていきましょう。オブジェクト指向において、オブジェクトとはアイデンティティを持つ値であり、JavaScriptにはオブジェクトと、オブジェクトではない値があります。後者のオブジェクトではない値を「基本値」と呼び、これには数値や文字列、真偽値などの種類があります。

ここで「アイデンティティを持つ」とは、同一性を確認できることや、独自性を持つことなどを意味します。たとえば、1を1で割った結果の1が、式中の1と同じ存在かどうかは、プログラム上は判定できません。これらの値は基本値ですので、等価演算子は内容で比較するのです。一方で、Numberクラスのインスタンスはオブジェクトですので、**リスト1**に示すように区別できます。

なお、基本値では代入時に値が複製されるのに対して、オブジェクトでは参照が複製されます。リスト1の例では、変数p1とp2はそれぞれ独立した値を保持しますが、変数o1とo2は同一オブジェクトを参照します。よって、o1の内容を更新すると、o2の参照先にも影響を与えます。このあたりの動作の違いも、識別性から連想するとわかりやすいのではないでしょうか。JavaScriptのオブジェクトは、内容よりも存在こそが重要となるのです。

また、オブジェクトには内部状態を保持できるという特徴があります。オブジェクトは独自性のある値ですので、自身で値を管理できると考えると良いでしょう。JavaScriptではこれらの状態をプロパティと呼び、**リスト2**のようにドットで名前を指定することで操作できます。

Objectインスタンスは、波括弧でプロパティ群を囲むことでも生成でき、このような式を「オブジェクト初期化子」と呼びます。先の例の内容をオブジェクト初期化子で表現したサンプルを**リスト3**に示します。ここではオブジェクト初期化子がObjectインスタンスの生成と同じ

▼リスト1　サンプルコード：基本値とオブジェクト

```
//基本値(同値性判定となる)
var p1 = p2 = 1;
var p3 = p1 / p2;
p1 == p2; //=> true
p2 == p3; //=> true

//オブジェクト(同一性判定となる)
var o1 = o2 = new Number(1);
var o3 = new Number(o1 / o2);
o1 == o2; //=> true
o2 == o3; //=> false
o2.valueOf() == o3.valueOf(); //=> true
```

▼リスト2　サンプルコード：オブジェクトとプロパティ

```
var obj = new Object();
obj.a = 1;
obj.b = 2;
obj.a; //=> 1
```

▼リスト3　サンプルコード：オブジェクト初期化子

```
var obj = {
  a: 1,
  b: 2
};
obj.a; //=> 1
```

▼リスト4　サンプルコード：関数宣言と関数呼び出し

```
var a = 1;
function func(b) {
  return a + b;
}
func(10); //=> 11
```

▼リスト5　サンプルコード：関数式とメソッド

```
var obj = {
  a: 1,
  func: function(b) {
    return this.a + b;
  }
};
obj.func(10); //=> 11
```

▼リスト6　サンプルコード：this参照の束縛

```
var func = obj.func.bind({a: 2});
func(10); //=> 12
```

意味を持つことを確認しましょう。

関数とメソッド

オブジェクトが理解できたところで、次はクラスへと進むために関数を確認していきましょう。関数は**リスト4**のように「functionキーワード」で宣言でき、宣言した関数は括弧を指定することで呼び出すことができます。なお、呼び出し結果からわかるように、関数内部からは外側の変数であるaを参照できます。逆に外側から内側は参照できません。

このようなJavaScriptの関数ですが、実は内部ではオブジェクトとして表現されています。先のサンプルコードでは、変数funcが実行可能なオブジェクトで初期化されていると考えてください。本オブジェクトはFunctionインスタンスであり、通常の値と同じように変数や引数などに指定できます。なお、宣言してから代入するのではなく、直接代入したいような場合は関数式が利用できます。プロパティfuncに関数式を指定した例を**リスト5**に示します。このようにオブジェクトに属する関数をとくにメソッドと呼びます。

上記のメソッドの内部では「this」というキーワードを指定してaプロパティを参照しました。「this」キーワードは常に何らかのオブジェクトを参照し、関数をメソッドとして呼び出した場合は、自身を呼び出したオブジェクトを指します。この例では、変数objから呼び出しているので、そのaプロパティは1を返します。

なお、明示的に違うオブジェクトを参照したい場合は、Functionインスタンスのbindメソッドが利用できます。bindメソッドは指定された引数でthisの参照先を束縛したFunctionインスタンスを生成します。**リスト6**の例で確認しましょう。

コンストラクタとプロトタイプ

ここまでの例では、オブジェクトに直接メソッドを設定してきました。この方法は直感的ですが、オブジェクトごとにFunctionインスタンスを生成することになるので、メモリ上に無駄が生じます。次は、オブジェクト間でプロパティを共有する方法を確認していきましょう。

まず、Functionインスタンスには、「new」キーワードを指定して呼び出すことで、オブジェクトを生成できるものがあります。このようなFunctionインスタンスをとくに「コンストラクタ」と呼び、ユーザが定義した関数はすべてコンストラクタとなります。

コンストラクタはprototypeという特別なプロパティを持ち、本プロパティが指すオブジェクトをとくに「プロトタイプ」と呼びます。プロトタイプはインスタンスのひな形のように動作し、インスタンスが未設定プロパティを参照すると、代わりに参照されます。その結果、プロ

トタイプのプロパティがインスタンス間で共有されることになります。

実際のコードで確認していきましょう。**リスト7**の例ではMyコンストラクタからインスタンスを生成しています。インスタンスのうち、my1オブジェクトではmethodメソッドを呼び出していますが、本オブジェクトはmethodプロパティを保持していません。よって、プロトタイプに委譲されて、そこで指定されているFunctionインスタンスが呼び出されます。

なお、リスト7のコンストラクタではthisキーワードを使用しました。これは、コンストラクタでは生成オブジェクトを参照するためです。また、メソッド中でもthisを指定していますが、本メソッドは生成オブジェクトから呼び出しています。結果的に、コンストラクタとメソッド中のthisは同一オブジェクトを参照することになります。

クラスとインスタンス

ここまでの例では、オブジェクト初期化でメソッドを指定してきました。これらはObjectインスタンスとなりますが、よく考えるとオブジェクトが独自のメソッドを持っていることになります。

一般的に、クラスは同じ振る舞いを持つオブジェクトの集合ですので、インスタンスが勝手に振る舞いを持つことに違和感を覚えるかもしれません。実は、JavaScriptではインスタンスの意味がほかの言語と少し異なるのです。最後にこれらを確認してオブジェクト指向の実践へと入りましょう。

まず、何をもって同じクラスとみなすかは技術によって異なります。たとえば、Java言語などでは同じインターフェースを備えるオブジェクトの集まりがクラスですが、HTMLでは同じスタイルを持つ要素の集まりがクラスです(class属性、擬似クラス)。また、正規表現では文字の区分がクラスであり(数値クラス)、IPアドレスでは数値の範囲です(クラスAアドレス)。そして、JavaScriptでは同じプロトタイプに委譲するオブジェクトの集合がクラスであり、Myインスタンスを生成するということは、Myプロトタイプに委譲するオブジェクトを生成することを意味します(**図1**)。

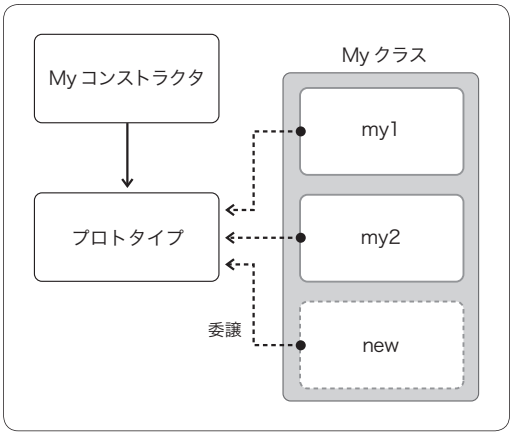
ここで、MyプロトタイプはMyインスタンスではない点に注意してください。クラス担任がクラスメイトではないように、プロトタイプはそのクラスのインスタンスではないのです。同様に、Objectコンストラクタのプロトタイプはオブジェクトですが、**リスト8**に示すようにObjectインスタンスではありません。

復習しましょう。オブジェクトとは何か、アイデンティティを持つ値でした。Objectインスタンスとは何か、Objectプロトタイプに委譲する具体例(instance)です。

▼リスト7　サンプルコード：独自インスタンスの生成

```
function My(a) {
  this.a = a;
}

My.prototype = {
  method: function(){ return this.a; }
};

var my1 = new My(1), my2 = new My(2);
my1.method(); //=> 1
my1.method == my2.method; //=> true
```

▼図1　プロトタイプとインスタンス



なお、継承は性質を引き継ぐことですので、JavaScriptではインスタンスを親と子の両方のプロトタイプに委譲させることを意味します。たとえばMyクラスのインスタンスは、**図2**に示すようにObjectインスタンスにも処理を委譲するので、MyクラスはObjectクラスを継承していると言えます。ここで、数あるObjectインスタンスのうち、独自の振る舞いを持ったものがMyクラスのプロトタイプとなっている点に注目しましょう。まるでJavaScriptにおいてインスタンスを生成するとは、自身に教えを請う弟子を生み出すようであり、クラスを継承するとは、優れた弟子が新たな伝道師となるかのようです。

▼リスト8　サンプルコード：Objectプロトタイプ

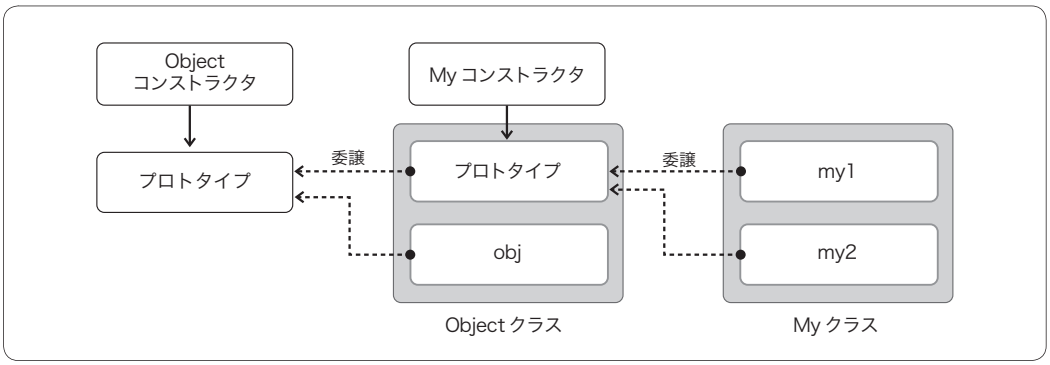
```
var obj = {};
obj instanceof Object; //=> true
obj.prototype instanceof Object; //=> false
```

オブジェクト指向の実践

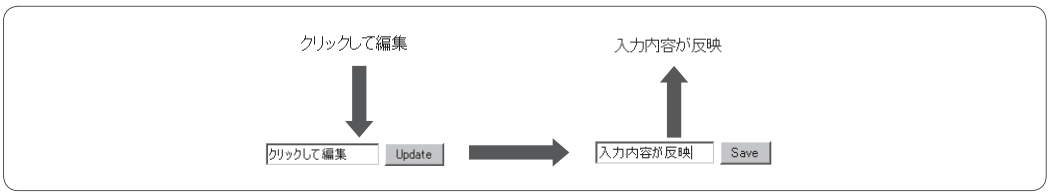
以上で、JavaScriptでオブジェクト指向プログラミングをするための知識が身につきました。次はこれをWebアプリケーション開発で活かす方法を確認していきましょう。ここでは一例として、表示内容をその場で編集できるUI部品を考えます。

本部品は**図3**のように表示内容をクリックすることで入力欄に切り替わり、ボタンを押して確定すると、入力内容で元の表示欄に戻るとします。つまり、本部品は入力値という内部状態を持つことになります。

▼図2　クラスの継承



▼図3　InPlaceEditor部品



コラム　JavaScriptにクラスはあるの？

JavaScriptにクラスはあるのでしょうか。厳密には、ObjectインスタンスはObjectオブジェクトなどと呼び、仕様ではクラスという用語を使用していません。歴史的経緯からこのような形となっていますが、instanceof演算子が言語仕様に入った以上、クラスやインスタンスと呼ぶほうが自然と著者は考えます。

開発者間のコミュニケーションでも、「XXのプロトタイプを親とするオブジェクト」などと呼ぶより、XXクラスのインスタンスと呼ぶほうが解かりやすいのではないかという意見があります。多くのライブラリでも、これらの概念をクラスと呼んでいますので、原理主義に陥らず積極的に使っていくことをお勧めします。

このような部品の実装でよくやってしまうのが、**リスト9**のように外部で内部状態を保持してしまうことです。この例ではボタンがクリックされたタイミングで入力値を外部の変数valueに退避し、その内容で表示欄を構築しています。一見、問題なさそうですが、同じ画面で本部品の利用個所が増えるとどうでしょうか。その数だけ、状態の入れものが必要となり、不慣れな後輩はコードごと複製して変数をvalue2、value3と増やしていくかもしれません。

リスト9の実装例を順に詳しく見ておきましょう。本サンプルではjQueryとHandlebars[注1]というライブラリを使用しました(A)。このうちHandlebarsはテンプレートエンジンで、テンプレート定義とパラメータから文字列を生成できます。ここでは表示欄と編集欄の2つのテンプレート定義を用意して(H、I)、テンプレートを生成しています(B)。また、入力値を格納するための変数valueを宣言し(C)、表示内容の構築を行っています。

この構築処理では、まず初期状態として表示欄を表示し(D)、クリック操作で切り替えるようにしています(E、G)。なお、編集欄から表示欄に戻るときは入力内容を変数valueに退避するようにしています(F)。

▼リスト9　サンプルコード：1人ぼっちの内部状態

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="jquery.min.js"></script>
    <script src="handlebars.js"></script>  ... (A)
    <script>
      $(function() {
        var templates = {
          viewer: Handlebars.compile($("#viewer_template").html()),
          editor: Handlebars.compile($("#editor_template").html())
        };  ... (B)
        var value = "クリックして編集";  ... (C)
        var target = $("#target")
          .html(templates.viewer({value: value})) ... (D)
          .on("click", "p", function(ev) {
            $(ev.delegateTarget).html(templates.editor({value: value}));
          })   ... (E)
          .on("click", "button", function(ev) {
            value = $(ev.delegateTarget).find("input").val(); ... (F)
            $(ev.delegateTarget).html(templates.viewer({value: value}));
          });  ... (G)
      });
    </script>
  </head>
  <body>
    <script id="viewer_template" type="text/x-handlebars-template">
      <p>{{value}}</p>
    </script> ... (H)
    <script id="editor_template" type="text/x-handlebars-template">
      <input type="text" value="{{value}}"></input>
      <button>Save</button>
    </script> ... (I)
    <div id="target">
    </div>
  </body>
</html>
```

注1）http://handlebarsjs.com/

▼リスト10　サンプルコード：オブジェクト指向の導入結果

```
function InPlaceEditor($root, templates) {
  this.$root = $root;
  this.templates = templates;

  this.update_value("クリックして編集")
      .on("click", "button", function(ev) {
         this.update_value($(ev.delegateTarget).find("input").val());
      }.bind(this))
      .on("click", "p", function(ev) {  ... (A)
         this.open_editor();  ... (B)
      }.bind(this));
}

InPlaceEditor.prototype = {
  update_value: function(value) {  ... (C)
    this.value = value;   ... (D)
    return this.$root.html(this.templates.viewer(this));
  },
  open_editor: function() {  ... (E)
    return this.$root.html(this.templates.editor(this));
  }
};

$(function() {
  var templates = {
    viewer: Handlebars.compile($("#viewer_template").html()),
    editor: Handlebars.compile($("#editor_template").html())
  };
  var target = new InPlaceEditor( $("#target"), templates);
});
```

インスタンスによる状態管理

リスト9のコード部分をクラスで書き換えると**リスト10**のようになります。修正前との最大の違いは、内部状態をインスタンスで管理するようにしたことです(D)。これによってインスタンスごとに状態の入れものが用意されるので、その管理の手間から解放されることになります。同じ部品を画面に増やしたい場合は、新たにnewすると良いでしょう。

また、副次的な効果として、部品や処理に命名できているのがわかります。とくに表示内容の切り替え処理がそれぞれメソッドとなり、update_valueやopen_editorメソッドと名付けています(C、E)。これによってたとえば「表示内容のクリック時(A)に、編集欄を開く(B)」といったことがコードから読み取りやすくなっています。これは開発者間のコミュニケーションが円滑になりますし、イベント処理に依存せず実行できるのでテストもしやすくなるでしょう。もしかすると状態管理よりも、こちらのメリットがうれしい方が多いかもしれません。

まとめ

本記事ではJavaScriptのオブジェクト指向を通して、オブジェクトやクラスを少し変わった角度から解説しました。これらの用語に慣れた方にも何か新しい発見があれば幸いです。また、オブジェクト指向の実践例として、フロントサイドでの活用方法を紹介しました。近年のフレームワークにはこれらのベストプラクティスを自然に実践できるようにしたものもあります。次のステップとしてBackbone.jsなどに挑戦してみるのも良いでしょう。

本記事がより楽しいJavaScriptプログラミングのきっかけとなることを祈っています。SD

PHPでオブジェクト指向

5 part

Part5ではPHPでのオブジェクト指向的なコードの書き方を紹介します。オブジェクト指向でコードを書くと量が増えてしまいますが、オブジェクト指向でないコードと比較するとその特徴やメリットが見えてきます。

㈲テクノランド　星野香保子（ほしのかほこ）

PHPでオブジェクト指向プログラミング

みなさんはPHPでオブジェクト指向的なコードを書いていますか？　PHPにはオブジェクト指向プログラミングを行うための機能が備わっています。PHPマニュアルの「クラスとオブジェクト」のページ（**図1**）を見ると、関連する文法についての説明が数多くあるので、ざっと目を通しておくことをお勧めします。

オブジェクト指向プログラムにメリットはあるのか

PHP言語に限りませんが、オブジェクト指向の文法を使ってコードを書いたからと言って、必ずしもオブジェクト指向プログラムが出き上がるわけではありません。オブジェクト指向ではコードの量が増えるとも聞きますが、オブジェクト指向で作るとどんなメリットがあるのでしょうか？

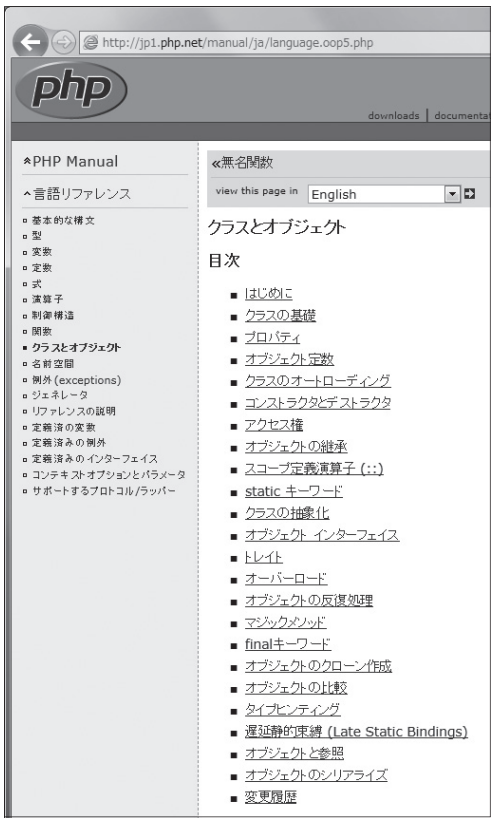
例題でオブジェクト指向を考えてみる

簡単な例題をもとに、まずはオブジェクト指向でないコードを作成します。その後、オブジェクト指向のコードに変更して、その特徴やメリットについて見てみましょう。

作成するスクリプトの仕様

図2のようなCSV形式の購入履歴ファイルを読み込み、請求金額を計算して**図3**のように結果を表示します。1行が1人分の顧客データで、各行のデータは左から、名前、会員種別（SまたはA）、購入金額（円）です。各顧客への請求金額は**図4**のように計算します。会員種別に応じて、割引率と送料は異なります。

▼図1　PHPマニュアル（http://www.php.net/manual/ja/language.oop5.php）



▼図2　購入履歴ファイル（data.csv）

```
Hana,A,3200
Yumi,S,13000
Taku,S,34800
Hiro,A,1500
```

▼図3　実行結果

```
Hanaさんへの請求額は、3340円です。
Yumiさんへの請求額は、11700円です。
Takuさんへの請求額は、31320円です。
Hiroさんへの請求額は、1725円です。
```

▼図4　各顧客への請求金額の計算方法

請求金額 ＝（購入金額 × 会員種別に応じた割引率）＋ 送料

会員種別Sの顧客は、割引率10%で送料は無料
会員種別Aの顧客は、割引率5%で送料は300円

オブジェクト指向的でないコード

PHPのオブジェクト指向プログラミングではクラスを使うことが基本となりますが、まずはクラスを使わずに書いたコード例を**リスト1**に示します。

▼リスト1　オブジェクト指向的でないPHPスクリプト（list1.php）

```
<?php
//! @param  $kaiinType  会員種別
//! @param  $kingaku    購入金額
//! @return             請求金額  -1: エラー
function procKeisan($kaiinType, $kingaku) {
  $kei = -1;
  if($kaiinType === 'S'){
    $kei = (int)($kingaku * 0.9);
  }
  else if($kaiinType === 'A'){
    $kei = (int)($kingaku * 0.95) + 300;
  }
  return $kei;
}

if(($fd = @fopen('data.csv', 'r')) !== FALSE){
  while(($dt = fgetcsv($fd, 32, ",")) !== FALSE){
    $kaiinType = $dt[1];
    $kingaku = $dt[2];
    $kei = procKeisan($kaiinType, $kingaku);
    if($kei >= 0){
      echo $dt[0], 'さんへの請求額は、',
             $kei, '円です。<br>';
    }
  }
  fclose($fd);
}
?>
```

コードの説明

リスト1では、procKeisan()という関数に会員種別と購入金額を引き渡して、請求金額の計算処理を行っています。実装する機能の規模が小さいので、クラスを使わなくてもとくに問題ないと言えますが、機能が増えたときには見通しが悪いコードになりそうです。もし会員種別が増えたり計算方法や計算用パラメータ（割引率、送料）が変更になったりした場合は、ロジック（制御構文のif文）に直接手を加えなければならず簡単には修正しづらい気もします。

オブジェクト指向的に書き換えたコード

クラスを使ってオブジェクト指向的に書き換えたコード例を**リスト2**に示します。オブジェクト指向でないコードに比べてライン数が倍以上になっていますが、オブジェクト指向によるメリットもちゃんとあるのです。

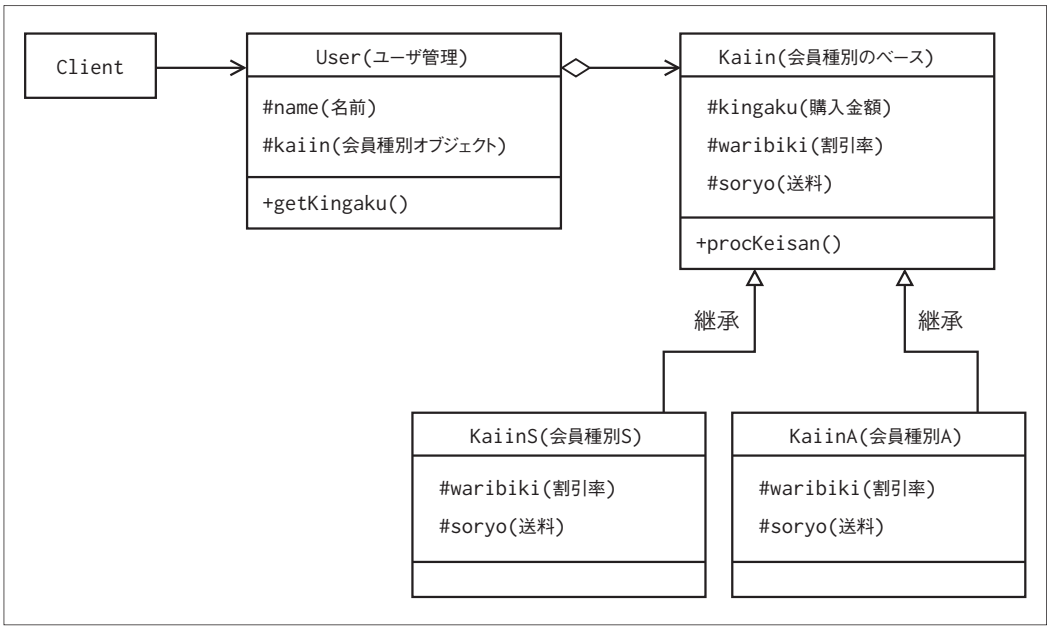
各クラスの説明

リスト2で作成したクラスを**図5**に示します。

▼リスト2　オブジェクト指向的なPHPスクリプト（list2.php）

```
<?php
class Kaiin { // 会員種別クラス ←①
  protected $kingaku = 0;
  protected $waribiki = 0;
  protected $soryo = 0;

  //! @param  $kingaku    購入金額
  function __construct($kingaku) {
    $this->kingaku = $kingaku;
  }

  public function procKeisan() {
    $kei = (int)($this->kingaku * $this->waribiki)
              + $this->soryo;
    return $kei;
  }
}

class KaiinS extends Kaiin { // 会員種別Sクラス←②
  protected $waribiki = 0.9;
  protected $soryo = 0;
}

class KaiinA extends Kaiin { // 会員種別Aクラス←③
  protected $waribiki = 0.95;
  protected $soryo = 300;
}

class User { // ユーザ管理クラス ←④
  protected  $name = '';
  protected  $kaiin = NULL;

  //! @param  $name       会員の名前
  //! @param  $kaiin      会員種別
  //! @param  $kingaku    購入金額
  function __construct($name, $kaiin, $kingaku) {
    $class = 'Kaiin' . $kaiin;
    if(class_exists($class)){
      $this->kaiin = new $class($kingaku); ←⑤
    }
    $this->name = $name;
  }

  function getKingaku() {
    if(!isset($this->kaiin)){
      return;
    }
    $kingaku = $this->kaiin->procKeisan();
    echo $this->name, 'さんへの請求額は、',
            $kingaku, '円です。<br>';
  }
}

// 以下はクラスを使う側の処理
if(($fd = @fopen('data.csv', 'r')) !== FALSE){
  while(($dt = fgetcsv($fd, 32, ",")) !== FALSE){
    $user = new User($dt[0], $dt[1], $dt[2]); ←⑥
    $user->getKingaku();
  }
  fclose($fd);
}
?>
```

▼図5　クラス図



Userクラス(リスト2の④)は、ユーザに関する機能を管理するためのクラスです。Kaiinクラス(リスト2の①)は、会員種別のベースとなるクラスです。このクラスをもとにして作ったのが会員種別ごとのKaiinSクラスとKaiinAクラス(リスト2の②と③)です。「extends Kaiin」の記述によってKaiinクラスを継承しています。請求金額の計算は会員種別によらず共通ですので、親クラスのKaiinクラスにprocKeisanメソッドとして記述しました。計算用パラメータ(割引率、送料)は会員種別ごとに異なるので、サブクラス(KaiinSクラス、KaiinAクラス)で設定しています。

会員種別ごとに異なるクラスを利用

Userオブジェクトの生成時(リスト2の⑥)に、ファイルから読んだ顧客情報(名前、会員種別、購入金額)を引き渡し、会員種別に対応するオブジェクトを生成します(リスト2の⑤)。たとえば、会員種別が'S'のときは'KaiinS'というクラス名を作り、newキーワードを使ってKaiinSオブジェクトを生成します。

オブジェクト指向プログラムのメリット

オブジェクト指向的なコードに変更したことで、どのようなメリットがあるのでしょうか？

設計上の役割分担を明確にできる

オブジェクト指向的なコード(リスト2)はライン数がずいぶん増えていますが、次の2点を分離するように設計したためです。

- どのような機能を持たせるか？
- どのように処理を実装するか？

オブジェクト指向設計に限らずこれらの2点を分けて考えることは大切です。リスト2では次のようにクラスに対して分担を割り当て、「機能」と「実装」を分離しています。

- Userクラス……どのような機能を持たせるかを決める
- Kaiinクラス……機能をどのように実装するかを決める

このようにすると、機能に影響を与えず実装のロジックだけを差し替えるなどということが可能になります。

独立性と拡張性を高めやすい

もし会員種別の種類が増えた場合には、Kaiinクラスを継承したサブクラスを新たに作ることで対応できます。計算方法が変更になった場合にも、変更の影響範囲をKaiinクラスに留めることができます。また、機能が増えた場合は、Userクラスにその機能を追加して対応できそうです。Userクラスを利用する側も、内部の具体的な処理については意識せずに実行したい機能をメソッドで呼び出せます。

おわりに

例示したサンプルコードは規模が小さいので、オブジェクト指向のメリットをなかなか実感しづらかったかもしれません。オブジェクト指向に関連する技術要素は種類が多く、本稿で説明できなかったメリットもたくさんあります。以下に参考文献を紹介しておきます。とくに「パーフェクトPHP」では、オブジェクト指向的に作られた実践的なアプリケーションのサンプルコードが載っていますので参考にしてみてください。SD

●参考文献
小川雄大、柄沢聡太郎、橋口誠著、「パーフェクトPHP」、ISBN978-4-7741-4437-5(技術評論社)
星野香保子著、「プロになるためのPHPプログラミング入門」、ISBN978-4-7741-4972-1(技術評論社)

6 part

Perlによるオブジェクト指向入門

『すぐわかるオブジェクト指向Perl』
深沢千尋(著)、2008年、3,780円

本誌ではじめまして。深沢千尋と申します。技術評論社からは過去に異常にゆるふわなPerl入門書『すぐわかるPerl』、『すぐわかるオブジェクト指向Perl』を出版しました。今回は伝統あるSD誌に登場できて光栄です。Perlを使ったOOPというお題をもらいましたので、概説的な話をしようと思います。

深沢千尋(ふかざわちひろ)　Twitter@query1000

カンタンなプログラミング言語Perl

Perlと言えば、テキトーに書け、ちょっとした用事を済ますのに便利なプログラミング言語という印象ですね。とりあえず、手を抜けるところは全力で手を抜こうとします。ファイルを辞書順にソートするプログラムは、

```
#! /bin/perl
# sortStdinStdout.pl -- 標準入力をソートして標準出力に出力する

print sort <>;
```

です。このように、ちょっとした用事にはPerlが最高です。

Perlには型がない

カンタン言語Perlの1つの特徴として「型」がないことが挙げられます。今、**$date**というスカラー変数を考え、日付を格納するとします。Perlのスカラーには型がありませんので、

```
$date = 20130629;
```

のように数値を代入しても、

```
$date = "20130629";
```

のように文字列を代入しても、同じものとして扱えます。

```
++$date;
```

で**1**が加算されますので**20130630**になりますし、

```
$date =~ s/^2013/2014/;
```

では正規表現で年が検索置換され、**20140630**に変更されます。

型が欲しいときもある

しかし、型が欲しいときもあります。「日付型」のようなデータを持っていて、日付らしい操作をしたい。たとえば、スカラー変数**$date**に**20140630**が入っているとき、

```
++$date;
```

を実行すると、**$date**の中身は**20140631**になります。

20140630という数字を見れば、我々人間は2014年6月30日という日付のことだろうなーと判断します。**$date**に**1**を加算するということは、当然次の日の2014年7月1日になってほしい。しかしさすがのPerlもそこまで気を利かせられず、2014年6月31日というあり得ない日付になってしまいました。

こういうとき、われわれはオブジェクト指向(OOP = Object Oriented Programming)の機能を使います。

たとえば「日付型」の変数を作って、1足せば翌日に、1引けば前日になり、月末／年末も的確に処理したい。その日付が何年か、何月か、何曜日かもすぐわかる。こういう機能があった

らいいと思いませんか。

このとき我々はOOPの「日付クラス」を導入し、$dateを日付クラスのオブジェクトにします。

CPANはプログラムの宝庫

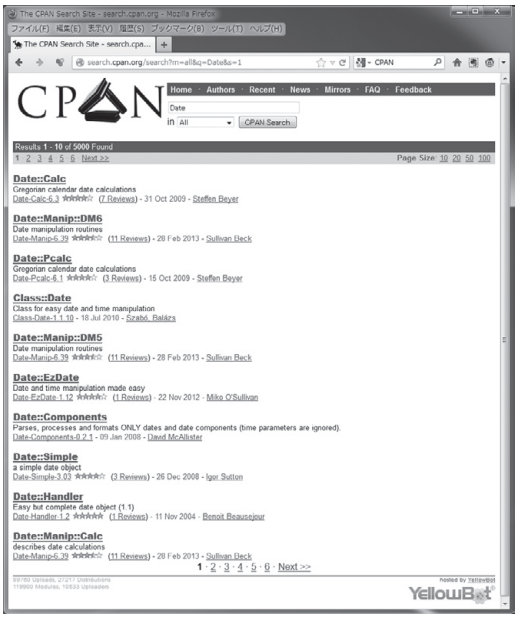
では、「日付クラス」を作りたい／使いたいと思ったら、どうプログラムを書けばいいでしょうか。こういうとき、普通は「車輪の再発明」を行わず、インターネット上に世界中のPerl腕自慢が部分プログラムをアップロードしたアーカイブ、CPANをサーチします。下図はCPANを「Date」という言葉で検索してみたところです(図1)。

Date::Simpleというのが使いやすくて良さそうです(後で使ってみましょう)。

車輪の再発明も時にはいいね!

しかし、勉強のためにあえて，頑張ってスクラッチから書いてみるのもオツなものです。これは絵画の「模写」に似ています。ということで、PerlのOOPの勉強として、日付クラスを作ってみます。とりあえず

▼図1　CPANをDateで検索してみた



・日付はY+MMDD形式の5ケタ以上の数字である(紀元前は考慮しない)
・数値nを足すと(年末、月末を考慮して)n日後になる

という仕様にしましょう。

実験環境とファイル名

今回テスト環境として、Windows 7でActivePerl 5.16を使いました。

クラス定義は、メインプログラムと同じファイルに入れることもできますが、普通は再使用のことを考えてモジュールというファイルに切り分けます。今回は日付クラスのモジュールをmyDate.pmとします。このようにモジュールのファイルの拡張子は.pmとします。

一方、myDate.pmを使うテスト用のプログラムをmyDateTest.plとします。この拡張子はどうでもよくて、UNIX系では拡張子なしが好まれます。WindowsのActivePerlは拡張子.plでPerlエンジンと関連付けていますので、ここでは.plを付けます。

myDate.pmと、myDateTest.plは、カンタンのため、単一のディレクトリ、C:¥myperlに入れました。

メインプログラムとその動作

では、すでにmyDateモジュールを作ったという前提で、それを駆動するメインプログラムを作ります(リスト1)。

▼リスト1　myDateTest.pl

```
#! /bin/perl
#
# myDateTest.pl -- 日付のテスト

use strict;
use warnings;
use myDate;

my $date = myDate->new(shift);
print "Date is $date¥n";

$date = $date->add(10);
print "10 days after is $date¥n";
print "30 days after is ", $date + 30, "¥n";
```

もくろみどおり、スッキリしてますね。
では実行します。

```
C:¥Perl>myDateTest.pl 20140630
Date is 20140630
10 days after is 20140710
30 days after is 20140819
```

バッチリですね。きちんと月末処理が行われています。

このように、モジュールを書いたら、なるべく小さなテストプログラムを書いてテストするといいです。

メインプログラムのミソ

では、メインプログラムのOOP部分を研究します。

```
use myDate;
```

という文では、作成するモジュールmyDate.pmを読み込んでいます。

```
my $date = myDate->new(shift);
```

は、myDateクラスのnewメソッドを呼び出しています。このメソッドはmyDateオブジェクトのコンストラクタと呼ばれるメソッドで、新しいオブジェクトを生成してそのリファレンスを返します。

```
$date = $date->add(10);
```

は、$dateオブジェクトにaddメソッドを引数10を渡して作用させ、10日後を得ています。

```
print "30 days after is ", $date + 30, "¥n";
```

は、add同様の操作を、+演算子を使って行っています。

myDate.pmの中身

では、いよいよモジュールを作成します(リスト2)。

OOP的なミソ部分を紹介します。

```
package myDate;
```

これはプログラムがmyDateというパッケージ(名前空間)に突入したことを示します。OOPでは、オブジェクトが属するものをクラスと言いますが、Perlではクラスはパッケージのことです。ここでmyDateパッケージは、ファイルmyDate.pmが終了するまで続きます。

```
sub new {
  my ($class, $date) = @_;
```

このnewはコンストラクタです。メインプログラムmyTestDate.plからの

```
myDate->new("20140630")
```

という呼び出しは、実はPerlの内部的に

```
myDate::new('myDate', "20140630")
```

に展開され、第1引数にクラス名myDateが挿入されます。だから引数の受け側は

```
  my ($class, $date) = @_;
```

というふうになり、newメソッドは$classに格納されたクラス名myDateを使うことができます。

```
    bless { date => $date, y => $1, m => $2, ↵
d => $3 } => $class;
```

blessされたリファレンスです。

bless関数によって日付データを格納する無名ハッシュリファレンスにクラス名を関連付けます。このように、Perlにおけるクラスはパッケージで、オブジェクトはパッケージ名を

オブジェクトメソッドadd

次に日付の加算を行うメソッドaddを研究します。まず呼び出し側を見てみましょう。

```
$date->add(10)
```

これによって、$dateオブジェクトのメソッドaddを呼び出しています。これも実は

```
myDate::add($date, 10)
```

▼リスト2　myDate.pm

```
# myDate.pm -- 日付のモジュール
use strict;
use warnings;

package myDate;

use overload
  '""' => 'toString',
  '+' => 'add',
;

sub new {
  my ($class, $date) = @_;
  unless (defined $date and $date =~ /(¥d+)(¥d¥d)(¥d¥d)/) { # 5桁以上の数字かチェック
    die "myDate: date $date should be 5 or more digits numeric!";
  } else {
    bless { date => $date, y => $1, m => $2, d => $3 } => $class;
  }
}

sub add {
  my ($self, $add) = @_;
  my ($date, $y, $m, $d) = ($self->{date}, $self->{y}, $self->{m}, $self->{d});

  for (1..$add) { # 加算する日付のぶんループする
    if ($m == 12 and $d == 31) { # 大晦日
      ++$y; $m = 1; $d = 1;
    } elsif ((&leap($y) and $m == 2 and $d == 29) # 月末
    or ($m == 2 and $d == 28)
    or (($m == 4 or $m == 6 or $m == 9 or $m == 11) and $d == 30)
    or ($d == 31)) {
      ++$m; $d = 1;
    } else { # それ以外
      ++$d;
    }
  }
  $date = $y. sprintf("%02d", $m). sprintf("%02d", $d); # 1桁の場合はゼロを付加
  bless { date => $date, y => $y, m => $m, d => $d } => ref $self;
}

sub toString { # 文字列スカラーに変換する
  return shift->{date}; # shiftは第1引数=オブジェクトそのもの
}

sub leap { # うるう年なら真を、それ以外なら偽を返す
  my $y = shift;
  return 1 if $y % 400 == 0;
  return 0 if $y % 100 == 0;
  return 1 if $y % 4 == 0;
  return 0;
}

1;
```

に展開されます。これをmyDate.pmのaddメソッドでは

```
sub add {
  my ($self, $add) = @_;
```

と受けます。変数$selfには、矢印記法->によって挿入されるオブジェクト（リファレンス）が入ります。

```
  my ($date, $y, $m, $d) = ($self->{date}, ➡
$self->{y}, $self->{m}, $self->{d});
```

この文では日付オブジェクト$selfを日付、年、月、日という各要素に分解しています。$selfはハッシュリファレンスですので、$self->{y}では無名ハッシュのyというキーの値が取り出されます。ここでは2014のような年が取り出されます。

addの最後は次の文です。

```
  bless { date => $date, y => $y, m => $m, d => $d } => ref $self;
```

addもコンストラクタ(オブジェクトを作るメソッド)です。というのは、addは日付に日数を加算した日付(myDateオブジェクト)を返す手続きだからです。ということで、加算によって更新されたデータ構造に、今度は「ref $self」という式をblessしています。refは$selfオブジェクトのパッケージ名を抜き出す関数です。

演算子のオーバーロード

なお、メインプログラムでは、

```
$date = $date + 30;
```

のように、プラス+演算子を使って日付の計算もしています。これは、演算子+のオーバーロード(多重定義)を行って、+演算子にaddメソッドを関連付けています。これには、overloadモジュールを使います。

```
use overload
  '""' => 'toString',
  '+' => 'add', # 余計なカンマを付けておくとエントリーを追加するとき便利
;
```

toStringメソッド

'""'というのは、オブジェクトを文字列化する演算子で、toStringメソッドを割り当てています。これで

```
print "10 days after is $date¥n";
```

のように$dateオブジェクトを二重引用符""で囲むと20140715のような文字列に置き換わります。

toStringメソッドの中身はこれだけです。

```
sub toString { # 文字列スカラーに変換する
  return shift->{date}; # shiftは第1引数=オブジェクトそのもの
}
```

shiftは第1引数を返す関数で、ここではmyDateオブジェクトになります。

myDateオブジェクトはハッシュへのリファレンスですが、このリファレンスのdateキーに対応する値(つまり20130630のような日付)を得て、メソッドの戻り値として返しています。

「1;」って何?

最後に

```
1;
```

という唐突な文を書いています。これはメインプログラムでモジュールを挿入するuse文が正常終了するために書いているものです。まあ、1;ぐらい気にしないで書いてください。

以上で自作モジュールmyDateの解説を終わります。

今度はCPANを使ってみよう

ではCPANモジュールDate::Simpleを使って同様の操作をします。

その前に、Date::Simpleは標準モジュール(Perlにデフォルトでインストールされるモジュール)ではありませんので、別途インストールする必要があります。もしインストールしていないモジュールをuseすると実行時に

```
Can't locate Date/Simple.pm in @INC (@INC contains: C:/Perl/site/lib C:/Perl/lib .)...
```

というメッセージで怒られます。エラーで@INCが出たらモジュールを入れてない、と覚えてください。

CPANモジュールのインストールは、最近はcpanm(CPAN minus)というプログラムを使うのが流行っています。

また、本誌読者で一番多いと思うUbuntuやMintなどのDebian系Linuxのユーザのみなさんは、apt-get installでモジュールをインストールするのも簡単です。ここではActivePerl付属のppmでインストールしました。

テスト

インストールができたらテストします。ここでは、Date::Simpleの説明ページに使いやすそうなサンプルプログラムがあるので(図2)、この先頭部分にちょっと手を加えたものを試してみます。

```
#! /bin/perl
# dateSimpleTest.pl --- Date::Simpleのテスト

use strict;
use warnings;

use Date::Simple ('date', 'today');

# Difference in days between two dates:
my $diff = date('2001-08-27') - date('2000-08-27');
print "$diff days passed from 2000-08-27 to 2001-08-27¥n";

# Offset $n days from now:
my $date = today() + 10;
print "10 days from now is $date in ISO 8601 format (YYYY-MM-DD)¥n";
```

サンプルプログラムの実行

では、サンプルプログラムを実行してみましょう。

```
C:¥Perl>dateSimpleTest.pl
365 days passed from 2000-08-27 to 2001-08-27
10 days from now is 2013-04-21 in ISO 8601 format (YYYY-MM-DD)
```

いい感じですね。

ではサンプルコードと実行結果を比較して解説します。

```
use Date::Simple ('date', 'today');
```

ここではDate::Simpleモジュールのメソッド、dateとtodayをインポートしています。

```
my $diff = date('2001-08-27') - date('2000-08-27');
print "$diff days passed from 2000-08-27 to 2001-08-27¥n";
```

これは、dateメソッドに'2001-08-27'と'2000-08-27'を渡して引き算しているところです。このように、dateメソッドだけで簡単に日付オブジェクトを作ることができます。

▼図2　CPANのサンプルコード

```
365 days passed from 2000-08-27 to 2001-08-27
```

実行結果はこうです。1年離れているので、ちゃんと365が返っています。

```
my $date = today() + 10;
print "10 days from now is $date in ISO 8601 format (YYYY-MM-DD)¥n";
```

これは、$dateに今日の10日後を入れています。$dateを""の中に入れるとおなじみのYYYY-MM-DDフォーマットで表示します。

以下説明は割愛しますので、各自研究してください。ほかにも年を返すyearメソッド、月を返すmonthメソッド、曜日を返すday_of_weekメソッドなど、いっぱい機能があります。日付関係はほぼこれで用が足りるのではないでしょうか。

継承する

このように偉大なDate::Simpleモジュールですが、日本人にとっては和暦の日付が使えないのが残念です。このような場合、OOPではクラスの継承を行って自分好みの機能を追加します。次はJapanDate.pmというモジュールでDate::Simpleを継承しています。紙数の関係で平成しかサポートしていません。コンストラ

クタnewの引数に"H25-06-30"のように書くと、平成25年6月30日、つまり2013-06-30でnewしたのと同じことになります(リスト3)。

use base文は基底クラスDate::Simpleを指定しています。

ここではnewメソッドを書き直しています。これをオーバーライド(override、上書き)と言います。

$class->SUPER::new($date)は基底クラスDate::Simpleのnewを呼び直しています。引数$dateを、H25-06-30のような平成和暦のときは西暦化して、それ以外のときはそのまま、基底クラスのnewメソッドを呼んでいるわけです。

このように、CPANモジュールにちょっとだけ自分ならではの機能を追加できます。また、Perlの標準モジュールも継承して改造することができます。

▼リスト3　JapanDate.pm

```
# JapanDate.pm -- 和暦が使える日付モジュール

use strict;
use warnings;

package JapanDate;
use base 'Date::Simple';

sub new {
  my ($class, $date) = @_;
  if ($date =~ /^H(¥d¥d)-(¥d¥d)-(¥d¥d)/) { # Hがついていたら平成
    my ($year, $month, $day) = ($1, $2, $3);
    $year = sprintf "%04d", $year + 1988; # 平成元年は西暦1989年
    $date = "$year-$month-$day"; # 西暦に直した
  }
  $class->SUPER::new($date); # Date::Simpleの$dateを普通に呼ぶ
}

1;
```

PerlとOOP

あるプログラミング言語がOOLたりうる条件は「カプセル化」、「多態性」、「継承」の3つで、これを「オブジェクト指向の3本柱(three pillars of object oriented programming)」と言います。

カプセル化(encapsulation)はデータ構造や機能の実装をユーザに意識させないこと(隠蔽)ですが、これをPerlではパッケージ、リファレンス、bless、矢印記法で実装します。myDate.pmでは年、月、日、月末、年末、うるう年チェックなどを隠蔽して、リファレンス変数$dateが日付型変数のように振る舞っています。

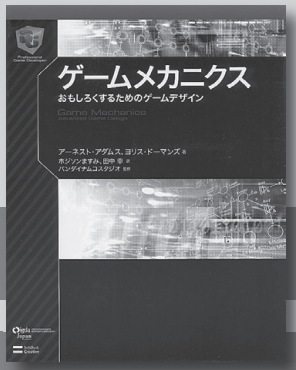
多態性(polymorphism)は1つの機能が対象によって異なる振る舞いをすることですが、Perlでは演算子のオーバーロードで実装します。myDateでは同じプラス+という演算子に「数字+数字」の場合は普通の足し算をし、「日付+数字」の場合は日付を進めるという複数の機能を与えています。

継承(inheritance)はあるクラスの機能を自分好みに作り替えることですが、Perlではuse base、SUPER、オーバーライドの機能を使って実装します。上記のJapanDateではCPANモジュールDate::Simpleを継承して和暦機能を追加しました。

いかがでしょうか。PerlのOOPはバージョン5になって後付けされたもので、ある程度ユーザが手作りで実装しなければいけない独特なところがあります。しかし筆者は、この「普通の言語にこれとこれを加えるとOOPになっちゃったー」という感覚によって、かえってOOPの本質が理解できました。これが「生まれつきOOL」、「OOLで当然」、「逆に言うとOOPしかできない」という言語では得られない面白い感覚だと思います。

何よりOOPはプログラミングをラクチンにする技法です。「おぶじぇくとしこう……」などと言うと、構えてお勉強しないといけないようなイメージがありますが、ざっと言うと本稿で述べたこれだけのことです。いろんなCPANモジュールをインストールして、ちょいちょいとメソッドを呼び出すだけで、自分のプログラムがどんどん高機能になっていく快感を、ぜひ楽しんでください。SD

BOOK no.1 ゲームメカニクス
おもしろくするためのゲームデザイン



アーネスト・アダムス、ヨリス・ドーマンズ【著】／ホジソンますみ、田中 幸【訳】／バンダイナムコスタジオ【監修】
B5変形判、 400ページ／価格＝3,990円 （税込） ／発行＝ソフトバンク クリエイティブ
ISBN＝978-4-7973-7172-7

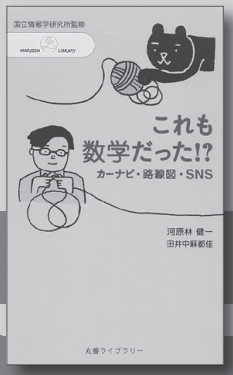
RPGの場合、ゲームをおもしろくするのは、RPGそのもののルール、ストーリー、敵キャラクターとの戦い、武器の入手による主人公のレベルアップなどさまざまな要素がある。これらの要素や組み合わせをゲームメカニクスという。RPGに限らずメカニクスがゲームのおもしろさを決める。本書前半は典型的なメカニクスのパターンを説明。後半はマキネーションという著者が開発したメカニクスを視覚化する記法を使ってメカニクス設計の解説を行う。ゲーム開発者にとって実践的な内容ではあるのだが、ゼルダの伝説、スーパーマリオなどを例に、そのおもしろさの理由を分析している部分などはゲーム開発者でなくともおもしろく読める。

BOOK no.2 SSD完全攻略読本

北川 達也、 加藤 勝明、 鈴木 雅暢、 竹内 亮介 【著】
A5判、 192ページ／価格＝1,974円 （税込） ／発行＝インプレスジャパン
ISBN＝978-4-8443-3366-1

出始めはノートパソコン1台くらい買えそうな価格だったSSDも、今や256GBのもので2万円を切るお手頃な価格のものが出てきている。高速化のためにHDDから乗り換えようと思っている人も多いのではないだろうか。本書は、SSDの基礎知識から活用方法まで幅広く解説しており、各社SSDの速度の比較や売れ筋の製品の詳細なカタログもあるので、導入方法がわからず躊躇している人にとって役立つだろう。HDDの載せ替えのテクニックや、RAID構築、コントローラ、インターフェース、ファームウェアの解説や実験、マニアックな記事もあり、すでにSSDを利用している人も一見の価値がありそうだ。

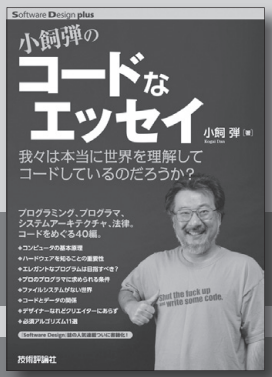
BOOK no.3 これも数学だった!?
カーナビ・路線図・SNS



河原林 健一、 田井中 麻都佳 【著】
新書判、 208ページ／価格＝798円 （税込） ／発行＝丸善出版
ISBN＝978-4-621-05382-9

本書は日常生活の中に潜んでいる数学、とくに離散数学について解説する。野球の試合スケジュールの計算、カーナビにおける最短経路の計算、データマイニングなどさまざまな場面でも数学が使われている。何の共通点もなさそうなものばかりだが、いずれも単純に計算したら膨大な計算量が必要なものばかり。これらをいかに速く計算するかというところに離散数学は使われている。速く計算するための効率的な計算方法も多数紹介されている。いくら高性能なCPUやメモリがある時代とはいえ、単純計算を繰り返すだけでは無駄に時間がかかるだけ。効率的なプログラムを書かけなければいけない。本書はそんな教訓を与えてくれる。

BOOK no.4 小飼弾のコードなエッセイ
我々は本当に世界を理解してコードしているのだろうか?



小飼 弾 【著】
A5判、 200ページ／価格＝2,184円 （税込） ／発行＝技術評論社
ISBN＝978-4-7741-5664-4

本誌2010年5月号から2013年3月号まで掲載されてきた同名連載がこのほど書籍化。本書における「コード」とは「電脳（コンピュータ）」に適用されるもの、「人脳」に適用されるもの、そしてこの世界に適用されている「法則」のこと。関数型言語のありがたみ、カーネルとプログラミング言語のつながり、ハードウェアを理解する重要性、数学的思考、ファイルシステムがない世界……など、「コード」をテーマに縦横無尽に語り尽くす。技術者、そして科学者たる読者に向けた「我々は本当に世界を理解してコードしているのだろうか？」という著者の問いかけを、ぜひ味わい、そしてご自身の行動のヒントにしていただきたい。

第2特集

研修じゃ教えてもらえない!?

あなたの知らないUNIXコマンドの使い方

入社から1ヵ月が経ち、そろそろITエンジニアとしての仕事が見えてきたころでしょうか。本特集はこれまでUNIX/Linux系のOSに触れたことがあまりない新人ITエンジニアさんに向け、今後の仕事で力強い味方となるUNIX/Linux系コマンドを紹介します。与えられたGUIツールできっちり仕事をこなすことはもちろん大切です。しかし、コマンド操作にはGUI操作にはないメリットがたくさんあります。サーバやネットワークの管理・監視、そしてデスクトップ環境で、使い込むほど作業効率があがるコマンドを習得しましょう！　かたわらにコマンドリファレンスの本を置き、本特集を読んでいただければ学習効果抜群です。

※本特集ではUNIXシステムライクなOSであるLinuxのコマンドも便宜上まとめて「UNIXコマンド」としておりますことをご了承ください。

RHEL編

UNIX/Linuxで必須のファイルシステムの基礎

UNIX/Linux OSにはじめて触れる方にとってファイルシステムの理解は必須です。本稿では、ファイルシステムの作成と操作で重要なコマンドを交えながら、その基礎を説明します。

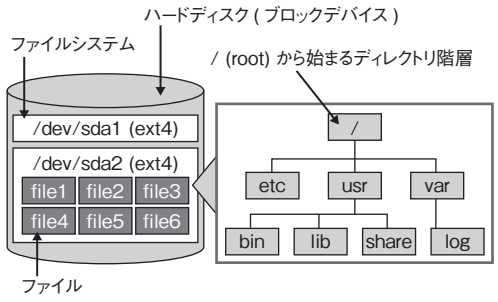
平 初 *TAIRA Hajime* レッドハット(株) ソリューションアーキテクト Twitter @htaira

ファイルシステムとは

システムに搭載されているハードディスクの容量やセクタ数はシステムによって異なります。もしOSがハードディスクの中に記憶領域の番地を決め打ちでデータを格納していった場合、システムの管理者は番地をすべて把握しておかなければならず、ハードディスクの管理は複雑性の高い作業になることでしょう。

そこでOSの中では**ファイルシステム**というしくみを提供しています(**図1**)。Windowsを使ったことがある人に一番簡単な表現をすれば、「C:ドライブ」がそうだと言えばイメージできるでしょうか。今までAndroidやiOSのスマートフォンしか触ったことがないニュージェネレーションの人には申し訳ないですが、ファイルシステムというものがコンピュータにあると覚えていただきたいです。

▼図1 ブロックデバイスとファイルシステムについて(イメージ)



ファイルシステムは、ハードディスクの中身を人間がわかりやすいように表現したデータ管理手法です。ユーザはファイルシステムの中にディレクトリ(フォルダ)やファイルを作成し、そして階層型のデータ管理を行います。つまりファイルシステムがなければコンピュータ上のデータはファイルとして管理することができません。RAWデバイスなどファイルシステムがなくてもハードディスク上にデータを格納する方法もあるが今回は割愛します。

ファイルシステムの種類(タイプ)

Linuxでは実にさまざまなファイルシステムを扱うことができます。主なものは**表1**にまとめましたが、これ以外に数えきれないほどのファイルシステムが存在します。

なぜ、ファイルシステムにこれほど多くの種類があるかと言えば、ファイルシステムがコンピュータやディスクの性能、また時代ごとのさまざまな要求によって進化したからです。どれが良いのか悩ま

▼表1 Linuxが扱える主なファイルシステム

Minix Filesystem
Linux Extended Filesystem(ext1、ext2、ext3、ext4)
FAT12/16/32
ISO9660(CD-ROM用のファイルシステム)
XFS
ReiserFS
JFS
GFS(Global File System)

しいところですが、どれも長所や短所があり、それぞれのファイルシステムの使い道は適材適所です。

しかしながら、デフォルトで採用されるファイルシステムが一番無難で汎用的であることは間違いありません。最近ではext4ファイルシステムが、一般的なLinuxディストリビューションでデフォルトとして選ばれる傾向にあります。本稿ではこのext4ファイルシステムを管理するためのコマンドについて取り上げていきます。

ext4ファイルシステムについて

Red Hat Enterprise Linux（RHEL）6では最新のext4ファイルシステムがデフォルトになりました。RHEL5までデフォルトだったext3ファイルシステムでは、対応できるディスク容量の限界が懸念されていました。そこでext4が生まれたわけです。ext4の一番の特徴は、大容量ファイルシステムのサポートです。ext4はファイルシステムの物理ブロック番号が48bitで管理されるようになったため、仕様上1EB（TBの100万倍）までのファイルシステム容量を扱えます。RHEL6のリリースでは上限16TBまでの容量がサポートされます。また、ext4はext3の後継ファイルシステムであり後方互換性もあります。ちなみに、RHEL6ではext4以外の大容量ファイルシステムとして「XFS」や「GFS2」も利用可能です。

ジャーナリング機能

エンタープライズのサーバ環境ではファイルシステムにジャーリング機能が必要となります。ext4はジャーナルをサポートするジャーナリングファイルシステムです。

ジャーナリングとは、ファイルシステムに変更がある場合に、その更新内容をジャーナルログに記録します（**図2**）。電源喪失やハングアップなどのシステム障害が生じた際、次回起動時にファイルシステムのチェックが行われますが、その際にジャーナルログに記録された内容を確認するだけでよく、修復作業が早くなります。したがって、サービス再開までの時間が短縮できるメリットがあり、また問題があった際にはジャーナルログを使った復旧が可能となる優れた管理機能です。

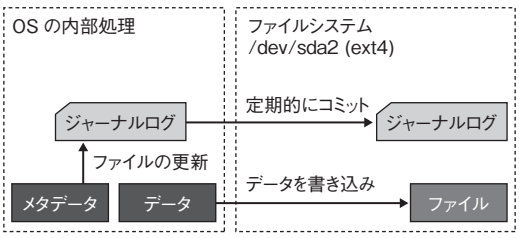
とはいえジャーナリングでさえも、おかしなデータがジャーナルに入り込んでしまうとファイルシステムを破損する可能性はあります。ext4からはジャーナリングに対する整合性を確認するためのしくみとして、ジャーナルのチェックサムを実装し、ファイルシステムに対して確実な変更が適用できるようにしました。

ファイルシステムの作り方

ファイルシステムを作るには、まず、他のOSと同じくパーティショニングを行う必要があります。この点については日ごろパソコンを使っていると経験する作業なので割愛します。Linuxの環境では、ディスクは**ブロックデバイスファイル**という/dev内にある特別なファイルとして扱われます。とくに最近のLinuxでは、SATAやSAS、FC（Fibre Channel）経由で接続されているディスクは「/dev/sda」や「/dev/sdb」といった命名規則でOSから認識されています。

ブロックデバイスファイル/dev/sdbをパーティショニングした場合、/dev/sdb1や/dev/sdb2などの数字がブロックデバイスファイルの後にふられます。このブロックデバイスファイルに対して、**mkfs**コマンドを実行することでファイルシステムが作られます（いわゆるフォーマット）。ファイルシステムを作るといっても何か派手なことが起きるわけではなく、パーティションの中身にメタデータが書き込まれるだけのことです。

▼図2　ジャーナリングファイルシステムの動き（イメージ）



Linuxではext3やext4、XFSなど有名なものから、あまり馴染みがないファイルシステムまでさまざまなタイプを扱えることは前述しました。そのため、mkfsというコマンドも各ファイルシステムごとに用意されています。ext4ファイルシステムを作るmkfsコマンドには、**mkfs.ext4**という名前のコマンドが用意されています[注1]。

```
# mkfs.ext4 /dev/sdb1
```

TRIMのサポートの可否

お使いのディスクがTRIM（Block Discard）に対応しているか否かは**hdparm**コマンドを実行することで確認できます。たとえば、/dev/sdaに認識されているディスクが「SAMSUNG SSD 840 Series」というモデルだった場合、**図3**のように表示されます。なお、**grep**[注2]コマンドで必要な部分だけ取り出すとよいでしょう。

inodeテーブルの遅延初期化

ファイルシステムの作成を早く終わらせたい場合、inodeテーブルの遅延初期化をオプションで指定することで、mkfs.ext4コマンドの処理で時間がかかるinodeテーブルの初期化作業をバックグラウンド処理で行わせることができます。これを行うためには、mkfs.ext4コマンドに「-E」オプションで「lazy_itable_init=1」という値を渡して実行します（**図4**）。

ちなみに筆者の手元の環境で1TBのハードディスクの中にファイルシステムを作成してみたところ、デフォルトでは約3分かかっていた処理が、inodeテーブルの遅延初期化をしたところたったの5秒で完了しました。

ファイルシステムのマウント

ファイルシステムをマウントするには、**mount**コマンドを使います。通常はファイルシステムタイプとブロックデバイスファイルとマウントポイント（ディレクトリ）の3つを指定します。必要に応じて「-o」オプションでマウントオプション[注3]を指定できます。何も指定しなければデフォルトで読み書き可能モードでマウントされます。

```
# mount -t ext4 /dev/sdb1 /srv
```

すでにマウント中のファイルシステムをオプションを変更して再度マウントしたいときには-oオプションで「remount」を指定する必要があります。次の例はマウント中の/srvを、読み込み専用を意味する「ro」を指定して再度マウントするという指示です。この操作はレスキュー作業でファイルを救うときに行う場合があります。

```
# mount -o remount,ro /srv
```

なお、ファイルシステムをアンマウントする場合には、**umount**コマンドを実行します。ちなみにunmountコマンドは存在しませんのでスペルミスにご注意ください。

注1）最近の実装では、ストレージ側にTRIMやSCSI UNMAPなどBlock Discardと総称される機能が実装されているものがある（SSDやハイエンドストレージ）。この場合には未使用ブロックの回収作業が行われるトリガーの役割も兼ねている。

注2）指定されたパターンを含む行を抽出し、結果を標準出力に出力。

注3）書き込み権限など指定が可能。

▼図3　TRIMに対応したディスクを見つける

```
# hdparm -I /dev/sda | grep 'Model\|TRIM'
↑詳細表示をする「-I」オプションで出力をgrepに渡し、その中から「Model」または「TRIM」が含まれる行を画面に表示
    Model Number:       SAMSUNG SSD 840 Series
       *    Data Set Management TRIM supported (limit 8 blocks)
```

▼図4　inodeテーブルの遅延初期化を指定してファイルシステムを作成

```
# mkfs.ext4 -E lazy_itable_init=1 /dev/sdb1
```

```
# umount /home
```

また、リムーバブルハードディスクやUSBフラッシュメモリなどの場合には、ブロックデバイスファイルに対して**eject**コマンドを実行すると自動的にアンマウント処理も行われます。

```
# eject /dev/sdb
```

ファイルシステムの修復方法

システムの起動時や、ファイルシステムのマウント時にファイルシステムの整合性チェックが定期的に実行されます。ext4ファイルシステムではファイルシステムの整合性チェックと修復を行うコマンドとして**fsck.ext4**というコマンドが用意されています。

もし起動時にファイルシステムのエラーで起動できなくなった場合には、fsck.ext4コマンドで修復作業を行わなければなりません。また、fsck.ext4を実行する場合にはファイルシステムを一度アンマウントする必要があります。/(root)のファイルシステム以外であればシステムが起動している状態でもアンマウントできるため、ファイルシステムのチェックを行うことができますが、/(root)のファイルシステムに対しては、インストーラDVDにてレスキューモードで起動し、マウントされていない状態でfsck.ext4コマンドを実行する必要があります。

オプションで「-fpcv」を指定して実行すると、ファイルシステムの自動修復とバッドセクタの修復を行います。

-f ……ファイルシステムの状態がcleanでもファイルシステムチェックを行う
-p ……ファイルシステムの自動修復を行う
-c ……バッドセクタの修復を行う
-v ……ファイルシステムのチェック状況を詳細に出力する

```
# fsck.ext4 -fpcv /dev/sdb1
```

root予約領域

ext4ファイルシステムには、従来のext2、ext3ファイルシステムでも存在していた、rootユーザに対して用意されるroot予約領域(Reserved block count)があります。これはファイルシステムの容量を100%使い果たした万が一の場合に、必要最低限のオペレーションを管理者であるrootユーザが行えるようにと確保されている領域です。しかしながら、デフォルトでファイルシステムの全体容量の5%が確保されるため、最近の4TBハードディスクだと約200GBも使えない領域ができてしまいます。

/(root)ファイルシステムに対して、予約領域を0%にしてしまうと緊急時に問題になってしまいますが、おもにファイルサーバのデータ領域や、仮想化環境の仮想マシンイメージの保存先として使う場合において5%はもったいないですね。

Reserved block countを確認するには**tune2fs**[注4]コマンドの結果をgrepコマンドにて「Reserved block count」をキーワードに検索すれば見つかります(**図5**)。

Reserved block countを0%にするには同じくtune2fsコマンドで「-m」オプションを指定して**図6**のように実行します。

注4) ファイルシステムのパラメータを調整する。

▼図5 Reserved block countの確認

```
# tune2fs -l /dev/sdb1 | grep 'Reserved block count'
```

▼図6 Reserved block countを0%にする

```
# tune2fs -m 0 /dev/sdb1
```

ファイルシステムの探し方（UUIDやラベル）

ファイルシステムには、コンピュータがユニークに管理できるように16進数での表記のUUIDや、人間がわかりやすいようにラベルという属性値がメタデータに含まれています。これらの値がわかれば、ファイルシステムのマウント時にmountコマンドでブロックデバイスファイルを指定する代わりに、UUIDやラベルとマウントポイントを指定するだけで構いません。また、システムの起動時にマウントする際に、そのブロックデバイスとマウントポイントのマッピングを行っている/etc/fstabにも、UUIDやラベルで指定することができます。

なぜこのようにUUIDやラベルで指定したいかというと、ブロックデバイスファイルの識別名はハードディスクの認識順によって決まるため、たとえばUSB接続の外付けハードディスクなどで/dev/sdc1だったものが、内蔵ハードディスクを増設して再起動を行うと次回起動時に/dev/sdd1といった違った識別名で認識されてしまうことが運用上起きるからです。よってよほど特別な理由がない限り、ファイルシステムをUUIDやラベルで指定したほうが賢い選択と言えます。

説明が長くなってしまいましたが、UUIDやラベルから、そのファイルシステムのメタデータに値が含まれているブロックデバイスファイルを探すのが**findfs**コマンドです。**図7**のように「UUID=ファイルシステムのUUID」か、**図8**のように「LABEL=ラベル名」をfindfsコマンドのオプションとして指定して実行することで、（見つかった場合は）そのブロックデバイスファイルが結果として返ってきます。

なお、UUIDを調べるには**tune2fs**コマンドの結果から「UUID」をキーワードにして検索すれば見つかります（**図9**）。

マウント中のブロックデバイスの探し方

シェルスクリプトなどでディレクトリの名前からマウント中のブロックデバイスファイルの名前を知りたいときなど、**df**[注5]コマンドの結果を見てテキスト処理をするという方法もありますが、最近では、もっと便利なコマンドが存在します。それが**findmnt**コマンドです。オプションなしで実行するとfindmntコマンドで取得できる情報がどのようなものかわかると思います。

たとえば、マウントポイント/srv/backupsの情報を取得する場合は**図10**のようにします。また、

注5）ファイルシステムごとに空き容量や使用率などの情報を表示する。

▼図7　UUIDからブロックデバイスファイルを調べる

```
# findfs UUID=79abb798-6664-44f1-929e-03919b1f3594
/dev/sda1
```

▼図8　LABELからブロックデバイスファイルを調べる

```
# findfs LABEL=/boot
/dev/sda1
```

▼図9　ブロックデバイスファイルのUUIDを調べる

```
# tune2fs -l /dev/sda1 | grep UUID
Filesystem UUID:          79abb798-6664-44f1-929e-03919b1f3594
```

▼図10　/srv/backupsの情報を取得する

```
# findmnt /srv/backups
TARGET       SOURCE                    FSTYPE OPTIONS
/srv/backups /dev/sdb3                 ext4   rw,relatime,seclabel,data=ordered
```

マウントポイント/srv/backupsのブロックデバイスファイルだけを表示する場合には、次のオプションをつけて**図11**のように実行します。

--noheadings ……ヘッダ情報を表示しない
--output ……指定した情報だけ表示する

ファイルシステムのリサイズ

システムが運用を開始した後にデータ量の増加により、ファイルシステムのサイズを変更(リサイズ)したい要求が発生する場合があります。最近では各ファイルシステムごとにresize2fsや、xfs_growfsといった専用のリサイズコマンドが用意されています。ext4ファイルシステムには**resize2fs**コマンドがあります。resize2fsコマンドにはリサイズしたい対象のブロックデバイスファイル名とサイズをオプションで指定します。ファイルシステムを拡張する場合はオンライン(マウント中の状態)で実行可能です。たとえば、/dev/sdb1のファイルシステムを100GBに拡張したい場合には、次のように実行します。

```
# resize2fs /dev/sdb1 100G
```

ファイルシステムサイズを変更したい場合には、事前にLVMなどでブロックデバイス側のリサイズ、それもサイズ拡張を行った後に行われることがほとんどです。よってファイルシステムのサイズ指定を省略することができます。

```
# resize2fs /dev/vg01/lv02
```

これで指定したブロックデバイスのOSで認識している容量までファイルシステムサイズを拡張します。

なお、ファイルシステムを縮小する場合にはマウント中のファイルシステムを一度アンマウントしてから、fsck.ext4コマンドにてファイルシステムの整合性を確認したうえで行う必要があります(**図12**)。

ちなみに/(root)にマウントされているファイルシステムはアンマウントできないため、インストーラDVDにてレスキューモードで起動し、マウントされていない状態にしてからresize2fsを実行する必要があります。この前処理を怠ると**図13**のようにコマンドの実行が拒否されます。

おわりに

ファイルシステムの操作はLinuxシステムを運用していくうえでとくに重要な操作の1つです。どのコマンドも実行してから返答があるまで、しばらく時間がかかります。実行時間が肌間隔でわかるぐらいまで、テスト環境で何度も実行してみてください。SD

▼図11　/srv/backupsのブロックデバイスファイルだけを表示させる

```
# findmnt --noheadings --output SOURCE /srv/backups
/dev/sdb3
```

▼図12　ファイルシステムのサイズを縮小する場合には、あらかじめ整合性をとるように指示される

```
# resize2fs /dev/vg01/lv01 20G
resize2fs 1.41.12 (17-May-2010)
Please run 'e2fsck -f /dev/vg01/lv01' first.
```

▼図13　/(root)に接続したままファイルシステムのサイズを縮小しようとした場合のメッセージ

```
# resize2fs /dev/vg01/lv01 18G
resize2fs 1.41.12 (17-May-2010)
Filesystem at /dev/vg01/lv01 is mounted on /; on-line resizing required
resize2fs: On-line shrinking not supported
```

Column 1 超入門者に捧げるコマンド&シェルスクリプト

上田 隆一 *UEDA Ryuichi* （有）ユニバーサル・シェル・プログラミング研究所 Twitter @uecinfo

未経験者のあなたへ

こんにちは。本誌の後ろのほうで『開眼シェルスクリプト』を連載しているUSP研究所／USP友の会の上田です。こちらの特集では、新人さん向けに「あなたの知らないUNIXコマンドの使い方」というお題を仰せつかりました。しかし、本誌の読者で「新人」と言うと、まったく油断できません。「んなもん知っとるわい」と言われかねません。それをかわしているとやたらマニアックな記述になり……。悩ましい。

そこで、「あなた」は、未経験でこの世界に放り込まれた人だと勝手に仮定して話を進めます。玄人の人には、未経験者に最初何を教えるべきか考えつつ読んでいただければと。

未経験者はどれくらい未経験か

筆者が大学生のころは、UNIX（今思うとBSD的な何か）を使った情報処理の演習がありました。筆者はすでにWindowsでC++を書いていたので楽勝……と思ったのですが、計算機センターの安いモニタのちらつきが気持ち悪く、夜通し麻雀ばかり打っていたというやむを得ない理由のため、講義中はほとんど寝てました。

ある日、どうしてもメールを読まなければならず計算機センターに行きました。で、ログインしました。画面に、

```
[hoge:~] ueda%
```

と出てきたのでなんとかメールを読もうとガチャガチャしてたら、ホームから叩き出されて、

```
[hoge:/] ueda%
```

となってしまいました。大パニックです。ホームに帰るべく、マニュアルを調べるけど「cdをオプションなしで打つ」という正解にたどり着きません。

そして、「ログアウトしてログインする」という高度な準最適解にたどり着いたのが、16年前の筆者です。

こんな野郎に、何から教えるべきか。筆者は仕事で未経験者と接することが多いので理解できていますが、未経験者というのは、このような人たちです。

tree(1)

かつての筆者のケースを考慮すると、最初にちゃんとわかっておかなくてはならぬのは、1にも2にもディレクトリということになります。この際、別に「フォルダ」と言ってもよいでしょう。いつまでもフォルダじゃいけませんが。

ディレクトリを調べるのはpwd(1)やls(1)（(1)の意味は後述）が王道のコマンドですが、こいつらはディレクトリの中の一部しか照らしてくれません。

ということで、最初に紹介するのはtree(1)コマンドです。たとえば、筆者のUNIXマシン（＝Mac）のホームで、

```
uedamac:~ ueda$ tree | less
```

と打つと、画面が切り替わり、

```
.
├── 201306.html
├── 201306.rst.txt
├── ARCHIVE
│   ├── DEL010224
│   │   ├── Picture
│   │   │   ├── Picture4
│   │   │   │   ├── 2.txt
│   │   │   │   ├── 3.txt
│   │   │   │   └── 4.txt
```

というように、ファイルやらディレクトリやらが木になって表示されます（treeが入ってない場合は、人に頼んでインストールしてもらってください）。**less(1)**コマンドの機能で、上矢印キーや下矢印キー（本当はJキーとKキーがお勧め）で舐め回して見ることができます。閉じるときはQキーを押します。

|はパイプと言って、本来画面に表示される文字を、右側のコマンドに渡すための記号です。実はこの「|」（パイプ）は、UNIX系OSをUNIXたらしめているものです。

treeに話を戻しましょう。これはCUIを使っていても非常に直観的です。ディレクトリの中に、他のディレクトリやファイルがぶら下がっていることがわかります。この図を見たら、UNIXを触ったことがなくても、なんとかHDDの中身[注1]がどうなっているかぼんやりわかるでしょう。

man(1)

次は**man(1)**コマンドです。

このコマンドは、システムにインストールされているコマンドのマニュアルを表示するためのものですが、実質、「すでに知っているコマンドのオプションを調べるため」に使うものです。

manで見られるマニュアルには章立てがあり、さっきからコマンドの後ろに「(1)」と書いているのは、「1章」ということを示しています。コマンドのマニュアルは1章にあるので、**printf(1)**と書いてあればそれはコマンドのprintfということになります。C言語のprintfは**printf(3)**です。

manを使ってみましょう。たとえば、後から出てくる**find(1)**について調べるにはコマンドを次のように打ちます。「man 1 find」のように1章と明示する方法もあります。

```
uedamac:~ ueda$ man find
```

マニュアルが開いたら、less(1)と同じ操作で閲覧できます。

```
FIND(1)                   BSD General Commands ↵
Manual                  FIND(1)

NAME
     find -- walk a file hierarchy

SYNOPSIS
     find [-H | -L | -P] [-EXdsx] [-f path] ↵
path ... [expression]
     find [-H | -L | -P] [-EXdsx] -f path ↵
[path ...] [expression]
... (略) ...
```

UNIXも英語も両方覚えるのは大変だ！　という人には、日本語のマニュアルもありますので、無理をしないでください。rootになって、たとえば「intstall man 日本語 <お使いのOSなど>」などで検索をかけてインストールの方法を調べましょう。

シェルスクリプトにメモ書き

パイプでコマンドをつなげると、コマンドの出す文字を次のコマンドに次々と渡していくことができます。たとえば、次のように書くと、今いるディレクトリの下で、「rstという拡張子で、かつ中にawkという文字列があるファイル」という2つの条件でファイルを検索することができます。

```
uedamac:SD_GENKOU ueda$ find . | grep "\. ↵
rst$" | xargs grep awk | sed 's/:.*//' | ↵
sort -u
./201202.rst
./201203.rst
... (略) ...
```

最初のうちは、頑張って書いたら取っておきたいと思うでしょう。上矢印を押すと前に打ったコマンドが再表示されますので、コピーしてファイルに保存します。ここではfindgrep.shというファイル名で保存したとします。

注1）正確な表現ではない。

cat(1)コマンドで保存したファイルを見て、次のような状態にします。

```
uedamac:SD_GENKOU ueda$ cat findgrep.sh
find . | grep "\.rst$" | xargs grep awk | ↵
sed 's/:.*//' | sort -u
```

そうしたらchmod(1)で次のようにすることで、findgrep.shを実行可能ファイルにします。

```
uedamac:SD_GENKOU$ chmod +x findgrep.sh
```

これでOK。次のように打つと、さっきと同じファイルのリストが画面に表示されます。

```
uedamac:SD_GENKOU ueda$ ./findgrep.sh
./201202.rst
./201203.rst
…(略)…
```

素朴ですが、これがシェルスクリプトです。もう少し垢抜けた(?)感じにすると次のようになります。

```
uedamac:SD_GENKOU ueda$ cat findgrep.sh
#!/bin/bash
# $1ディレクトリの下のrstファイルから、
# $2の語句を検索する                    ↗続く
find "$1" |
grep "\.rst$" |
xargs grep "$2" |
sed 's/:.*//' |
sort -u
```

実行してみましょう。

```
uedamac:SD_GENKOU ueda$ ./findgrep.sh . ↵
awk | head -n 2
./201202.rst
./201203.rst
…(略)…
```

このように、シェルスクリプトを書くと、コマンドの複合技を使えるようになります。「単純な数個のコマンドで、どれだけ仕事ができるか」ということを考えるのは、シェルスクリプトを書く楽しみであり、古き良きUNIXプログラミングの技法でもあります。

個人的には、この程度のものは端末にすぐに書けるようにして、シェルスクリプトにすることは極力控えます。ただ、慣れるまでは、こういうふうに「動くメモ書き」をたくさん残しておくこともアリかと思います。SD

CentOS編

マネジメントサービスプロバイダ業務を支えてきた、いざというときに備えるコマンド

VPS1台構成のシステムから100インスタンスを越えるシステムまで、多数のサーバを監視・管理する筆者が遭遇した数々の危機。それらを乗り越えるために活躍したコマンド・シェルを紹介します。

馬場 俊彰 *BABA Toshiaki* （株）ハートビーツ Twitter @netmarkjp

どんな環境や状況でも変わらず使えるコマンド・シェルは管理者必須

マネージメントサービスプロバイダ（MSP）業務において、コマンド・シェルは必須ツールです。動作がシンプルなコマンドの出力と入力をつないで、情報を思いどおりに引き出し、加工し、状況を把握して対応するのはMSP業務の醍醐味です。

MSP業務は大きく分けると“監視・障害対応”と“サーバ管理”の2つに分かれます。サーバ管理というと最近はPuppet[注1]やChef[注2]などのツールを使ってサーバの設定を一元的にバージョン管理するのが常識ではありますが、筆者の勤務先は少数多種のサーバを顧客と共同管理しているため、ツールありきの運用は厳しいという事情があります。そこでコマンドやシェルが大活躍しているのです。

また障害対応のときはタイムリーに情報を収集し対応する必要があるため、コマンドやシェルを活用して情報をすばやく収集することが迅速な対応の肝になります。

そんなわけで本章では筆者の経験をもとに、現場でよくあるシチュエーションでのコマンドの使い方を紹介します。いざというときに使えるよう頭の片隅に置いておいていただくと、そのうち——たぶん割とすぐに——役に立つと思います。

注1） https://puppetlabs.com/
注2） http://www.opscode.com/chef/

CASE 1 ● 刻一刻と減るディスク残容量

ディスク残容量はサーバ監視の超定番項目です。ディスク残容量が0になるのは非常にマズいです。ログの書き出しができなくなり、プログラムによっては動作が停止する深刻な事態です。

実際にサーバを運用していると、残容量20％を切ったあたりでwarningアラートが発報し、それをきっかけにログインして詳しい状況を確認……**図1**のように、**df**[注3]コマンドを連打していると、みるみる残容量が減っていくことがよくあります。

こんなときも慌てずに原因を探りましょう。ディスク残容量が減っているということは、ファイルが新たに生成されているか、すでにあるファイルのサイズが大きくなっているかのどちらかです。

ファイルを探すといえば**find**コマンドです。findには作成日や更新日を条件にファイルを探すオプションがあるので、それを利用しましょう。

今回使うオプションは「-mmin」です。このオプションの値に「-15」とすると、15分以内に更新されたファイルを探すことができます（**図2**）。

最近更新されたファイルが抽出できたら、あとは**ls**[注4]コマンドや**du**[注5]コマンドでファイルのサイズを

注3） ファイルシステムごとに空き容量や使用率などの情報を表示する。
注4） 指定されたパスにあるファイルやディレクトリを一覧表示する。
注5） ディスクの使用量を確認する。

確認します。1ファイルずつ手作業であたるのはナンセンスなので、xargs[注6]コマンドを使って楽をしましょう（図3）。なお、xargsをうまく活用するとさまざまな作業がグンと楽になります。たとえばssh[注7]コマンドと組み合わせて、多数のサーバに並列で処理を実行させることもできます（本稿では割愛）。

ディスク使用量上位を確認する場合は、さらにsortコマンドを組み合わせましょう（図4）。これで原因が特定できます。

たいていはユーザ操作による大容量ファイルのアップロードや、アプリケーション設定の誤りによるデバッグログ出力が原因です。変わったところでは、ユーザがMovableTypeの出力ディレクトリ設定を誤り、今までとは違うディレクトリに大量の小さいファイルが生成され、結果としてディスク残容量がみるみる減少する事態に立ち会ったこともあります。みなさまディスクの使いすぎにはご注意ください。

注6）標準入力からコマンドラインを作成して実行する。
注7）通信経路を暗号化してリモートホストにログインする。

CASE 2● 突然の死

さてはて、サーバの構築はしてみたけれど、なぜだかデーモンが起動しない。そんなことって、ありますよね。あるんですよ。

そんなとき、どうしましょう？

まずはログを見ます。ログを見てください。しかしログすら出力されていないとき、ありますよね。そんなときはstraceコマンドを使いましょう（図5）。

トレースログが大量に出力されるので、ファイ

▼図1　ファイルシステムの使用状況をdfで細かくチェック

```
$ df -m      ←-mオプションは容量をメガバイト単位で表示する指定
Filesystem           1M-ブロック     使用    使用可 使用% マウント位置
/dev/mapper/vg_root-lv_root
                         50397     18802     29036  40%
tmpfs                     3967         0      3967   0% /dev/shm
/dev/sda1                  485       108       352  24% /boot
/dev/mapper/vg_root-lv_home
                         398766     90166    288344  24% /home
$ df -m
Filesystem           1M-ブロック     使用    使用可 使用% マウント位置
/dev/mapper/vg_root-lv_root
                         50397     19829     28008  42%
tmpfs                     3967         0      3967   0% /dev/shm
/dev/sda1                  485       108       352  24% /boot
/dev/mapper/vg_root-lv_home
                         398766     90166    288344  24% /home
```

▼図2　15分以内に更新されたファイルを探す

```
$ sudo find / -mmin -15 -type f > /tmp/newfiles.txt
↑「-type f」で検索対象をファイルに指定。「>」でその出力結果を/tmp/newfiles.txtに上書き保存
```

▼図3　図2の結果から不要なものを除いてファイルサイズを表示する

```
$ cat /tmp/newfiles.txt | grep -vE "^/(proc|sys)" | xargs sudo ls -al
↑catコマンドでnewfiles.txtの中身を読み出してgrepコマンドに渡す。grepは拡張正規表現を使って行頭に/procや/sysを
 含まない行を出力しxargsに渡す。xargsの内容はlsの引数として渡されてファイルサイズ付きで一覧表示される
```

▼図4　図3での表示をファイルサイズの大きい順に並べ替えて表示

```
$ cat /tmp/newfiles.txt | grep -vE "^/(proc|sys)" | xargs sudo ls -al | sort -n -r -k 5 | head
↑sortのオプションは「-n」で数値順、「-r」で逆順、「-k 5」でls -alで表示される5番目のフィールド、つまりファイルサイズでソートすることを指定している。
 最後のheadコマンドで受け取った出力の最初から10行目までを表示
```

ルに保存するのがいいでしょう。標準エラー出力をファイルにリダイレクトしてください（**図6**）。

トレースを見ていると、ファイルパスと共に「No such file or directory」が大量に出力されていることがありますが、これは問題ではないことが多いです。プログラム側で「ファイルがあれば読み込む」というコーディングがされているため、そのようにファイルを探しまわるのです。たとえばMySQLにおける「${HOME}/.my.cnf」のような、あれば使う・なければ使わないファイルがそれにあたります。

トレース出力を読んでいると、見たことがない・知らないシステムコールがたくさん出てきます。知らないシステムコールは**man**コマンドで確認しましょう。manはコマンドだけでなくシステムコールにも用意されています。トレース出力を確認するのは骨がおれますが、原因を突き止められる可能性はかなり高いです。最後の手段として覚えておきましょう。いざというときは「strace」です。

CASE 3● 消えたディスクイメージ

世はまさに、大仮想化時代！！！……というわけで猫も杓子も仮想化ですね。VMware、VirtualBox、LXCなど実装はさまざまありますが、筆者の勤務先ではKVMを多数利用しています。ディスクイメージをファイルとして作成して利用しているのですが、たまに100台に1台くらいの確率で……消えるんですよ、ディスクイメージのファイルが……仮想マシン稼働中に。今までで3回ほど遭遇しました……いや本当に……困る。

と思いきや大丈夫です。落ち着きましょう。VMが起動している間はファイルは消えません。VMが不慮の事故で停止する前にディスクイメージを復活させましょう。手順はこうです。

1. ディスクイメージへのリンクを突き止める
2. ファイルを復活させる

まずはディスクイメージのi-node番号を突き止めます。このために**lsof**[注8]コマンドを使います。rootユーザで実行しましょう。大量に出力されるので、grepでうまくフィルタしてください（**図7**）。

すると図7のようにコマンド、PID、ユーザ名などが一覧表示されます。「NODE」がi-node番号、「NAME」がファイルパスです。

プロセスがオープンしているファイルは「/proc」からたどることができます。具体的には「/proc/<PID>/fd/<FD>」にシンボリックリンクがあります。図7だと「/proc/1837/fd/9」になります。

この場所を**図8**のように確認しましょう。ありました。この「/proc/1837/fd/9」を**cp**コマンドでコピーすると、その時点のデータが複製できます。

重要 cp -aではなくcpでコピーしてください！

さて、これでcp時点のデータが確保できました。このあとVMを停止してからディスクイメージを再配置し、VMを起動しなおせば万事解決なのですが、コピーから停止までのデータはなくなってしまいます。そこでなんとかファイルを復活させてみましょう。具体的にはファイルシステムをいじってi-nodeに対するリンクを手動で作成します。危険な方法なので、実施するなら気をつけてくださいね。

まずは先ほどと同じくlsofで目的のファイルのi-node番号を確認します。今回は「9961507」ですね。i-node番号が判明したら、このi-node番号をもとに**debugfs**[注9]コマンドを使ってファイルシステムを書き込み可能で開いて、リンクを作成します。リンクの作成先は目的のファイルと同じパーティションである必要があります。

▼図5　straceでデーモンの起動を確認

```
$ sudo strace -f /etc/rc.d/init.d/httpd start
```

▼図6　標準エラーをファイルに保存

```
$ sudo strace -f /etc/rc.d/init.d/httpd start 2 > /tmp/trace
  ↑2は標準エラー出力を指す。tmpディレクトリのtraceというファイルに保存
```

注8）開いているファイルを表示する。
注9）システムファイルのデバッガ。

rootユーザでdebugfsを実行するとプロンプト(debugfs:)が表示されます。リンクの作成は「link <i-node番号> パス」です(**図9**)。i-node番号を「< >」で囲う必要があるので気をつけてくださいね。この操作により、目的のi-node番号を指すファイル「/tmp/newfile」が作成できます。あとはVMを停止し元々のファイルパスにcpしてください。

なお、この操作では/tmp/newfileのリンク数は0と表示されます。debugfs実行中に「modify_inode」をするとリンク数が変更できるのですが、筆者の理解不足か、うまく反映されませんでした。うまくいったらぜひblogなどを書いて教えてください。

なお、lsofの利用方法応用編として、不審なプロセスが起動しているときに、そのプロセスがどのファイルを開いているか確認することができます。ただし不審なプロセスがあるということはlsofの結果が操作されている可能性があるということなので、そのあたりも勘案して出力を確認しましょう。

CASE 4● 大丈夫か?

MSP業務では、作業1つ1つの確実性と合理性がとても重要です。自分の知見が不足していることを前提に「NG要素が見当たらなければOK」ではなく「OKであることを検証すればOK」「OKでなければNG」という考え方をする世界です。そのため誤りがありそうな個所は事前に誤りを取り除いておく、前提がある場合はその前提を確認する必要があります。そんなときに活躍するのが**diff**コマンドです。

失敗のあるあるとして、SSL証明書更新の際の証明書とキーの不一致によるSSL証明書更新失敗があります。これを回避するためにSSL証明書とキーが正しいかを事前に確認しておきましょう。

diffの入力はファイル名ですが、**<(COMMAND)**という構文を使うことでコマンドの出力を直接diffに渡すことができます(**図10**)。

コピペで使いまわせるように変数を利用しています。「${変数名:?}」の「:?」は「変数が定義されていなければ終了」という意味です。うっかりコマンド部分だけ貼り付けてしまってもコマンドが実行されないので安心です。とくに書き換えや削除の対象を指定する使い方をする場合には必ずつけてください(例:**rm -rf ${OLD_BACKUP:?}**)。

diffは差がないと終了コードが「0」になるので、それを利用して「&&」よる結果出力をするとわかりやすいですね。図10ではdiffを1回しか実行していませんが、たとえばダウンロードしたrpmのmd5チェックサムをぱっと確認したいときに、**図11**の方法で確認することができます。

「確実性」を求めるときにdiffは重宝します。つまりMSP業務の場合は手放せません。手動でのサーバ同士の移行で、構築ミスや設定漏れがないことを確認するために両サーバの「/etc」配下をまるごと比

▼図7 ディスクイメージのi-node番号を調べた例

```
sudo lsof | grep -E '(^COMMAND| .img)'
COMMAND  PID  USER FD TYPE DEVICE SIZE/OFF    NODE    NAME
qemu-kvm 1837 qemu 9u REG  253,2  21474836480 9961507 /var/lib/libvirt/images/guest.img (deleted)
```

▼図8 シンボリックリンクを確認

```
$ sudo ls -al /proc/1837/fd/9
lrwx------ 1 qemu qemu 64 Apr  4 23:03 /proc/1837/fd/9 -> /var/lib/libvirt/images/guest.img (deleted)
```

▼図9 リンクの作成

```
$ sudo debugfs -w /dev/mapper/vg_localhost-lv_root
debugfs 1.41.12 (17-May-2010)
debugfs: link <9961507>  /tmp/newfile
debugfs: quit  ←debugfsから抜ける
```

較して、想定外の差分がないか確認する、「rpm -qa」の出力を比較する、「php -i」の出力を比較するなどの操作は日常茶飯事です。

ChefやPuppetを使わない昔ながらの素朴な方法で構築されたサーバはまだ山のようにあります。これらを安全確実に運用するためにはdiffは必要不可欠なのです。

CASE 5● 終わらないリストア

最近は本当にディスクが大きくなってきました。伴って扱うデータ量が増え、そしてバックアップの容量も大きくなってきました。データが大きいので圧縮しますが、そうすると今度はリストアするときに時間がかかります。バックアップの容量は小さく圧縮できるに越したことはありませんが、リストアに時間がかかるのは困りものです。

そこで救世主登場。**pbzip2**コマンドを使いましょう。pbzip2を使えば、bzip2圧縮で容量節約、マルチコア利用で圧縮・展開が高速といいことづくめです。CentOSであればEPELリポジトリにパッケージがあるので、yumでインストールできます。

tar[注10]コマンドと組み合わせて使う場合は、**図12**のようにオプションでpbzip2を使うよう指定します。

解凍するときも同じ要領です。/tmp配下に解凍するときは**図13**のようにします。pbzip2を活用してリストア中の待ち時間を減らしましょう。

おわりに

本章ではMSP業務でよくあるコマンドの使い方をいくつか紹介しました。ころばぬ先の杖。日頃からコマンドに慣れておき、いざというときにオドオドせずスムーズに対応できるよう、普段から素振りをしておきましょう。SD

注10) ファイルやディレクトリをアーカイブする。

▼図10　diffを使った確認

```
$ KEYFILE=ssl.key
$ CRTFILE=ssl.crt
$ diff <(openssl rsa -in ${KEYFILE:?} -modulus -noout) \   長いコマンドの途中で改行を入れたい場合は「\」を入れてから改行をする
       <(openssl x509 -in ${CRTFILE:?} -modulus -noout) \
    && echo 'OK'
```

▼図11　md5チェックサムの確認方法

```
$ diff <(md5sum MySQL-client-5.5.30-1.el6.x86_64.rpm             | awk '{print $1}') \
       <(echo '6abd3f0ef88adb254c334ca9a7de37d9') \
    && diff <(md5sum MySQL-devel-5.5.30-1.el6.x86_64.rpm         | awk '{print $1}') \
       <(echo '8cdafd98919fb85961f586341e039a0a') \
    && diff <(md5sum MySQL-server-5.5.30-1.el6.x86_64.rpm        | awk '{print $1}') \
       <(echo 'fc81f2ce8f8c429a120bcacd6a45dc38') \
    && diff <(md5sum MySQL-shared-5.5.30-1.el6.x86_64.rpm        | awk '{print $1}') \
       <(echo '355bb32c6591289l2b42e6a2aa78d089') \
    && diff <(md5sum MySQL-shared-compat-5.5.30-1.el6.x86_64.rpm | awk '{print $1}') \
       <(echo 'e4173e65e030f80f30b0f2fd5389e2ea') \
    && echo 'md5 checksum ok'
```

▼図12　tarとpbzip2を組み合わせた圧縮

```
$ sudo tar cf backup.tar.bz2 --use-compress-prog=pbzip2 /etc /var/lib/mysql
```

▼図13　tarとpbzip2を組み合わせた解凍

```
$ sudo tar xf backup.tar.bz2 --use-compress-prog=pbzip2 -C /tmp
```

Column 2 サーバを管理するコマンド講座の最初の最初

桑野 章弘 KUWANO Akihiro （株）サイバーエージェント Twitter @kuwa_tw

はじめに

渋谷のサイバーエージェントという会社でサーバサイドエンジニアとしてサーバの構築・運用などをしております。UNIXコマンドについてということですが、筆者たちの仕事では日々いろいろな調査やファイル操作などにコマンドを使って処理しています。今回はそれらの中から非常にシンプルだけれどもちょっと役立つ、そんな使い方にしぼって実例を混ぜながら紹介します。

ファイル操作

ファイルの操作をするコマンドです。サーバでの作業をする際には必ずファイル操作が含まれるものですが、コマンドの組み合わせによって省力化することができます。

■ 特定のファイルの操作

たとえばディレクトリ/testdir以下にある、ファイル名に「del」が含まれるファイルの削除を行うにはどのようにするでしょうか。このような場合にはfindコマンドとxargsコマンドを使います。xargsはパイプ（|）で渡された出力をrmコマンドへと渡しており、この例ではfindで出力したファイルがrmで削除されます（**図1**）。

削除以外にも、特定のファイルだけ権限や所有者を変更したりと一括処理に便利です。ただし削除に関しては、目的のファイル以外を削除しないようによく確認してから実行しましょう。

▼図1 「del」を含むファイルの削除

```
$ find ./testdir -name "*del*" -print | xargs rm
```

[使われているコマンド]
find……検索コマンド
xargs……渡された出力を別のコマンドへと渡す
rm……ファイルを削除する

■ 他のサーバへのファイル書き出し

次に紹介するのは、ローカルディスクの容量がないサーバでの作業時に、tarコマンドでバックアップアーカイブを作りたい場合です。ローカルサーバに置けないので別にサーバを用意する必要があります。

たとえば**図2**のようなコマンドを実行します。tarで作成したアーカイブを標準出力として出力するために「-」を使用します。tarコマンドでの「-」は、標準出力にファイルの内容を出力し、後半の「-」にそのファイルの内容が渡されます。結果として./backupdir/以下をtarアーカイブとしてバックアップしたものが、{別のサーバのIPアドレス}の/tmp/send.tgzとして保存されることになります。

▼図2 別のサーバでバックアップアーカイブを作る

```
$ tar zcvf - ./backupdir/ | ssh ${sshでログインできるユーザ}@${別のサーバのIPアドレス} "cat - > /tmp/send.tgz"
```

[使われているコマンド]
tar……tar形式のアーカイブを作成する
ssh……sshで他のサーバへ接続する
cat……ファイルの内容を表示する

ログファイルの集計

次に、あなたはApacheなどのHTTPサーバのログファイル「access_log」を集計することになったとします。担当サービスの4月10日の12時台のログを調べて、どのURLへのアクセスが多いかを集計して見るためにはどのようにすればいいでしょう。

ログは**図3**のようなものが1リクエスト1行形式になります。実際に出力するためのコマンドは**図4**ような形になります。

catでaccess_logの内容を出力し、**grep**コマンドで、「10 Apr 2013 12」が含まれた行のみを抽出します。次にある、「awk '{ print $11 }'」は、スペースで区切られた11個目のフィールド（つまり、http://example.jp/foo/bar.html）を出力します。この出力を集計したいので、**sort**コマンドで同じものに並べ替えた後に、**uniq -c**コマンドで集計します。uniq -cの出力は行数の多いものが下にきてしまい見栄えが悪いため、最後の**sort -rn**で、数字の多いものを逆順にしてもう一度並べ替えています。

集計が簡単にできましたね。難しい集計を始めるとパイプをつなげ続けたり、awkの中が複雑になったりと、プログラムを書いたほうが良いのですが、簡単な集計であればこれで十分です。

[使われているコマンド]
grep……特定の文字列のみを選択して出力する
awk……テキスト処理用のプログラム言語
sort……文字列の並べ替えをする
uniq……重複文字列を一つにまとめる。-cオプションを付けると重複件数を合計する

まとめ

このように、さまざまなコマンドを使用することでサーバの管理を簡単にすることができます。ある程度慣れてきたらスクリプトに組み込むことで、手作業では面倒臭い作業をコンピュータに行わせることができます。

『UNIXという考え方』という本に9つの定理というものが紹介されていますが、その中には「一つのプログラムには一つのことをうまくやらせる」「スモール・イズ・ビューティフル」「すべてのプログラムをフィルタとして設計する」といったものがあります。これはUNIXコマンドの根底にある考え方です。今回はほんの一例を紹介しましたが、これらの考え方について少しでもわかっていただけたら幸いです。

コマンドにはほかにもいろいろなものがあります。調べてみると仕事などの役に立つことも多いですので、お暇なときに/bin/の中を調べてみてはいかがでしょうか。SD

▼図3　access_logの内容（一部）

```
XXX.XXX.XXX.XXX [Wed, 10 Apr 2013 12:00:03 GMT] - - "GET http://example.jp/foo/bar.html HTTP/1.1" ↵
304 undefined "" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ja; rv:1.9.2.16) Gecko/20110319 ↵
Firefox/3.6.16 ( .NET CLR 3.5.30729)" 32
… (略) …
```

▼図4　4月10日12時台のアクセス数を集計

```
$ cat access_log  | grep "10 Apr 2013 12" | awk '{ print $11 }' | sort | uniq -c | sort -nr
   1234 http://example.jp/foo/bar.html
    123 http://example.jp/foo/url2.html
     33 http://example.jp/foo/url3.html
     22 http://example.jp/foo/url4.html
     11 http://example.jp/foo/url5.html
      1 http://example.jp/foo/url6.html
```

FreeBSD編

サーバ運用と自動化に役立つ厳選コマンドリファレンス

その高い安定性が評価され、各種サーバ用途に多く採用されているFreeBSDでは、とくにCLIによる操作が重要視されます。本章で厳選したFreeBSDコマンドを足がかりに、デキるサーバ管理者を目指してください。

後藤 大地　*GOTO Daichi*　BSDコンサルティング（株）　Twitter @daichigoto、@BSDc_tweet

はじめに

FreeBSDは大規模Webサービスのプラットフォームやホスティングサービスのプラットフォーム、高性能アプライアンス、家電機器、組み込み機器など、さまざまな場所で活用されています。なかでも企業や教育機関などで多くの方が扱うことになるのは、Webサーバやメールサーバ、DNSサーバ、ファイルサーバなどエッジサーバとしてのFreeBSDでしょう。本章では、そうしたサーバ運用を前提として、FreeBSDコマンドの使い方を紹介します。

本章ではUNIXを利用するにあたっての基本的なコマンド——cat、cd、chmod、chown、chroot、cp、date、dd、diff、find、fsck、hostname、kill、less、ln、locate、ls、man、mkdir、more、mv、newfs、passwd、patch、ps、pwd、reboot、rm、shutdown、su、tarなど——については紙幅の都合上説明していません。ですが、これらのコマンドはもちろん使えるようになっておきましょう。

ネットワーク管理

データセンターに格納されたラックマウントサーバにインストールされたFreeBSDを管理したり、ホスティングサービスで提供されているFreeBSDを管理するなど、物理的にアクセスすることができ

NOTE　ソフトウェアのインストールには「Ports Collection」

FreeBSDのアプリケーション管理にはPorts Collectionを使います。Ports Collectionはソフトウェアのビルド方法やパッチなどをまとめたアプリケーションカタログのようなもので、2万4千を超えるソフトウェア情報が登録されています。

Ports Collectionは常に更新されていますので、使う前に、次のように**portsnap**コマンドを使って最新の状態に更新します。

```
% portsnap fetch extract update
↑本当に最初の1回だけ
% portsnap fetch update
↑以後はこちらでアップデート
```

たとえばvimをインストールするのであれば、次のように作業します。

```
% cd /usr/ports/editors/vim-lite/
% make install clean
…(略)…
```

Ports Collectionからパッケージをビルドすることもできます。数十台や数百台のサーバに同じソフトウェアをインストールする場合、必要になるパッケージをビルドして利用します。

ないFreeBSDサーバを使うケースが多く見られます。ネットワーク関連のコマンドはFreeBSDサーバを管理するための重要なコマンドです。

ifconfig

ネットワークインターフェースに関する設定は**ifconfig**コマンドで行います。IPアドレスの設定、ネットマスクの設定、ネットワークインターフェースカードの状況確認、機能設定、有効無効の切り替え、ブリッジの設定など、さまざまな操作が可能です。ネットワークの設定が固定化している場合には使うことが少ないコマンドですが、システム構築時などネットワークの設定を頻繁に変更したり、ノートPCなど接続場所を頻繁に変えるような場合にはよく使うコマンドです。

ifconfigは引数に何も指定しなければ、**図1**のように現在設定されているネットワークインターフェースの情報が表示されます。この例であればem0というネットワークインターフェースと、lo0というソフトウェア的に作成されたネットワークループバックインターフェースが表示されています。

通常、ネットワークインターフェースの設定は/etc/rc.confなどに書き込みます。実は、やっていることはシステム起動時にシェルスクリプトで/etc/rc.confの内容を読み込んだ後に、ifconfigコマンドを実行してネットワークの設定を実施するというものです。典型的には**図2**のような使い方をします。図2では、em0というネットワークインターフェースにIPv4アドレス192.168.1.10を設定し、ネットマスクとして255.255.255.0を指定しています。

なお、読んでいてわからない部分が出てきたらマニュアルを引いてみてください。この節の説明であれば、「man ifconfig」「man em」「man lo」などでそれぞれの詳しい説明が表示されます。

dhclient

DHCPでIPアドレスを取得したい場合には**dhclient**コマンドを使います(**図3**)。一時的にIPアドレスを変更して作業するためにifconfigで設定を変更して、そのあとでdhclientを実行して元の状態に戻す(ここではDHCPでの集中管理が実施されていると想定して)といった場合に使います。作業用のノートPCなどをネットワークに接続するといった用途でも使われます。

▼図1　ifconfigの実行

```
% ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
     options=4219b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,TSO4,WOL_MAGIC,VLAN_HWTSO>
     ether e0:69:95:f5:42:84
     inet 192.168.1.101 netmask 0xffffff00 broadcast 192.168.1.255
     nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
     media: Ethernet autoselect (1000baseT <full-duplex>)
     status: active
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
     options=600003<RXCSUM,TXCSUM,RXCSUM_IPV6,TXCSUM_IPV6>
     inet6 ::1 prefixlen 128
     inet6 fe80::1%lo0 prefixlen 64 scopeid 0x5
     inet 127.0.0.1 netmask 0xff000000
     nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
```

▼図2　ifconfigの使用例

```
% ifconfig em0 inet 192.168.1.10 netmask 255.255.255.0 up
```

▼図3　dhclientの使用例

```
% dhclient em0
DHCPREQUEST on em0 to 255.255.255.255 port 67
DHCPACK from 192.168.1.10
bound to 192.168.1.101 -- renewal in 300 seconds.
```

ping / traceroute

特定のホストへの到達性を調べるコマンドがping、どの経路を通って到達しているのかを調べるコマンドがtracerouteです。pingは簡単なハートビート(死活確認)としても利用できます(図4)。

tracerouteは、たとえばメールが送れなくなったとか、sshでログインできなくなったといった場合に、そのホストにたどり着くためのネットワークのどこに問題があるのかを調べるといった用途で使われます(図5)。

たとえばクライアントからサーバに対してtracerouteを実行し、LANとWANの接続部分にあたるルータまで到達していることがわかれば、たぶんルータで実施したファイアウォールの設定変更などが影響してアクセスできなくなったのではないか、といったことが推測できます。

dig / resolv.conf(5)

DNSの正引きおよび逆引きの確認にはdigを使います。図6の実行でDNSサーバに対してgihyo.jpというドメイン名のAレコードを問い合わせています。ここではDNSサーバは「192.168.1.1」のIPv4アドレスのサーバに存在しており、「gihyo.jp」というドメイン名のIPv4アドレスは「49.212.34.191」であることがわかります。いわゆる正引きと呼ばれる問い合わせです。

図7の使い方で逆引きになります。DNSサーバ(ここでは192.168.1.1)に対して「49.212.34.191」というIPv4アドレスに対応するドメイン名を問い合わせています。49.212.34.191に対応するドメイン名として「gihyo.jp」が返ってきていることがわかります。

▼図4　pingの使用例

```
% ping -c 3 192.168.1.105
PING 192.168.1.105 (192.168.1.105): 56 data bytes
64 bytes from 192.168.1.105: icmp_seq=0 ttl=64 time=16.064 ms
64 bytes from 192.168.1.105: icmp_seq=1 ttl=64 time=292.466 ms
64 bytes from 192.168.1.105: icmp_seq=2 ttl=64 time=1.318 ms

--- 192.168.1.105 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.318/103.283/292.466/133.908 ms
```

▼図5　tracerouteの使用例

```
% traceroute 192.168.1.105
traceroute to 192.168.1.105 (192.168.1.105), 64 hops max, 52 byte packets
 1  192.168.1.105 (192.168.1.105)  101.740 ms  0.982 ms  1.143 ms
```

▼図6　DNSの正引き

```
% dig gihyo.jp a

; <<>> DiG 9.8.4-P1 <<>> gihyo.jp a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24617
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;gihyo.jp.     IN  A

;; ANSWER SECTION:
gihyo.jp.  27650 IN  A 49.212.34.191

;; Query time: 16 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Mon Apr  8 19:51:50 2013
;; MSG SIZE  rcvd: 42
```

DNSでは、IPv4アドレスはPTRレコードという形式で保存されます。49.212.34.191というIPv4はアドレスは「191.34.212.49.in-addr.arpa.」という名前で保存されます。アドレスを1バイトごとに区切って、逆順に並び替えたものに.in-addr.arpa.という文字列を付加したものです。図7のコマンドは「dig 191.34.212.49.in-addr.arpa. ptr」のようにコマンドを実行して191.34.212.49.in-addr.arpa.のPTRレコードを問い合わせるという内容と同じです。ただし、これでは人間が理解しにくいので、「-x」オプションをつけて自動的にPTRレコードに格納されている形式に変換してから問い合わせるようにしています。

DNSサーバのIPアドレスは/etc/resolv.confファイルに書き込みます(**リスト1**)。dhclientを実行するとresolv.confファイルは自動的に差し替わるしくみになっています。

OSはDNSサーバに問い合わせを実施するためのライブラリを提供しています。ライブラリは問い合わせがあると/etc/resolv.confを読み、そこに記載されているDNSサーバへ問い合わせを実施します。リスト1のように「nameserver 192.168.1.1」のように記述があれば、192.168.1.1というIPv4アドレスのサーバに対して名前解決の問い合わせをします。

netstat

ネットワークに関係するさまざまな情報を出力するコマンドが**netstat**です。もっとも使われるのは、デフォルトルータのIPアドレスを取得するためではないかと思います(**図8**)。

fetch

HTTPやFTP経由のファイルを取得するには**fetch**コマンドを使います(**図9**)。同じようなコマンドにMac OS XやLinuxディストリビューションなどでよく使われる**curl**があります。

▼図7　DNSの逆引き

```
% dig -x 49.212.34.191 ptr

; <<>> DiG 9.8.4-P1 <<>> -x 49.212.34.191 ptr
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17769
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;191.34.212.49.in-addr.arpa. IN  PTR

;; ANSWER SECTION:
191.34.212.49.in-addr.arpa. 3566 IN  PTR gihyo.jp.

;; AUTHORITY SECTION:
34.212.49.in-addr.arpa. 3566  IN  NS  ns2.dns.ne.jp.
34.212.49.in-addr.arpa. 3566  IN  NS  ns1.dns.ne.jp.

;; ADDITIONAL SECTION:
ns1.dns.ne.jp.   98  IN  A 210.188.224.9
ns2.dns.ne.jp.   98  IN  A 210.224.172.13

;; Query time: 8 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Mon Apr  8 19:52:12 2013
;; MSG SIZE  rcvd: 141
```

▼リスト1　/etc/resolv.conf

```
search example.co.jp
nameserver 192.168.1.1
```

newaliases / aliases(5)

FreeBSDサーバを構築したあとで管理者がすぐに作業することの1つに、/etc/aliasesファイルを編集してrootに飛ばされるメールを自分のメールアドレスに転送する、というものがあります。こうするとFreeBSDからのメール、いわゆる「Charlie Rootからのメール」が届くようになります。FreeBSDではおなじみの名前であり、出社してから最初にチェックするのは、このCharlie Rootから届く報告メールです。

図10のような設定で、root宛てのメールがすべてdaichi@gihyo.jpおよびdaichi@gihyo.co.jpに転送されるようになります。転送先に複数のアドレスを登録する場合、カンマ区切りで連続してアドレスを記述します。/etc/aliasesファイルは編集したあとで必ず**newaliases**コマンドを実行してデータベースを更新する必要があります。忘れないようにしましょう。

ipmitool

データセンターに設置されたラックマウントサーバでは、物理的にアクセスできるサーバと違ってBIOSの設定変更や、強制的な電源リセットなどが困難です。この場合、IPMIを使って制御します。

図11のように**ipmitool**コマンドで192.168.1.103

▼図8　netstatの使用例

```
% netstat -nr -f inet
Routing tables

Internet:
Destination        Gateway            Flags    Refs      Use  Netif Expire
default            192.168.1.1        UGS         0   415939    em0
127.0.0.1          link#5             UH          0     3255    lo0
192.168.1.0/24     link#1             U           0    77678    em0
192.168.1.101      link#1             UHS         0        0    lo0
```

▼図9　fetchの使用例

```
% fetch ftp://ftp.freebsd.org/pub/FreeBSD/releases/amd64/amd64/ISO-IMAGES/9.1/FreeBSD-9.1-RELEASE-
amd64-bootonly.iso
FreeBSD-9.1-RELEASE-amd64-bootonly.iso        100% of  146 MB  324 kBps 00m00s
```

▼図10　aliasesの記述例（一部）と更新

```
% cat /etc/mail/aliases
…(略)…
root: daichi
daichi: daichi@gihyo.jp,daichi@gihyo.co.jp
…(略)…
% newaliases
/etc/mail/aliases: 31 aliases, longest 17 bytes, 330 bytes total
```

NOTE　仮想FreeBSD環境を構築する「jail」

より高いセキュリティが求められるサービスを提供する場合や、1台のサーバに何十といったホスト環境を作って提供する場合には**jail**コマンドが使われます。chrootの発想を推し進めた機能で、ファイルシステム名前空間以外にもさまざまな空間を隔離します。

jail環境の構築方法はここでは説明しませんので、FreeBSD HandbookのJailの章（http://www.freebsd.org/doc/handbook/jails.html）を参考にするか、PC-BSDをインストールしてWardenを使ってみてください。WardenはGUIでjailの構築や利用ができるツールで、はじめて使ってみるにはよいとっかかりです。

のIPv4が設定されているBMC（Baseboard Management Controller）に対して「ユーザ：admin、パスワード：admin」というアカウント権限で電源リセットをかけています。物理的にリセットボタンを押すのと同じ処理を実施していると考えてください。BMCはM/B（Motherboard）とは独立した回路になっており、PCとは独立して動作するためこのような管理が可能になっています。

とくにサーバが反応しなくなった場合にハードリセットをかけるためにIPMIが使われます。制御用のコマンドはPorts Collectionのsysutils/ipmitoolからインストールできます。

ファイル管理

tail

ログファイルを目視でモニタリングする場合に**tail**コマンドが使われます。tailは「-f」オプションを指定すると対象ファイルをモニタリングしつづけ、新しい書き込みがあるとそれを読み込んで表示します。設定の変更などを実施し、サービスが出力するログをリアルタイムにチェックしたい場合などに使います。

図12のコマンドで、/var/log/httpd-error.logというファイルのモニタリングに入ります。ターミナルはこの状態でいったん出力が停止し、何か新しいメッセージが/var/log/httpd-error.logに書き込まれると、その内容が順次このターミナルに出力されるようになります。

mount / swapinfo

ファイルシステムがどのようなオプション指定で使われているかは、**mount**コマンドに引数を与えないで実行することで表示できます。**dumpfs**コマンドでより詳細なデータを得ることができますが、通常の使い方であればmountで十分です。

たとえば**図13**の出力で、システムには/（root）、/dev、/procという3つのマウントポイントがあり、/はUFS＋Softupdatesでフォーマットされた物理ディスク（SATAの1つ目に接続されたディスクのGPTテーブル2番目のパーティションにある）であることがわかります。/devは動的にデバイスファイルを生成するデバイスファイルシステムになってい

▼図11　ipmitoolによるハードリセットの例

```
% ipmitool -I lanplus -H 192.168.1.103 -U admin -P admin chassis power reset
```

▼図12　tailの使用例

```
% tail -f /var/log/httpd-error.log
[Fri Apr 05 21:51:56 2013] [notice] caught SIGTERM, shutting down
[Fri Apr 05 21:54:11 2013] [warn] Init: Session Cache is not configured [hint: SSLSessionCache]
[Fri Apr 05 21:54:11 2013] [notice] Digest: generating secret for digest authentication ...
[Fri Apr 05 21:54:11 2013] [notice] Digest: done
[Fri Apr 05 21:54:11 2013] [notice] Apache/2.2.24 (FreeBSD) PHP/5.4.13 mod_ssl/2.2.24 OpenSSL/ ➲
0.9.8y DAV/2 configured -- resuming normal operations
[Fri Apr 05 21:54:34 2013] [notice] caught SIGTERM, shutting down
[Sat Apr 06 12:06:30 2013] [warn] Init: Session Cache is not configured [hint: SSLSessionCache]
[Sat Apr 06 12:06:30 2013] [notice] Digest: generating secret for digest authentication ...
[Sat Apr 06 12:06:30 2013] [notice] Digest: done
[Sat Apr 06 12:06:29 2013] [notice] Apache/2.2.24 (FreeBSD) PHP/5.4.13 mod_ssl/2.2.24 OpenSSL/ ➲
0.9.8y DAV/2 configured -- resuming normal operations
```

▼図13　mountの出力例

```
% mount
/dev/ada0p2 on / (ufs, local, soft-updates)
devfs on /dev (devfs, local, multilabel)
procfs on /proc (procfs, local)
```

ること、/procfsはプロセス情報をファイルシステムに写像するプロセスファイルシステムであることがわかります。

システムの故障やヒューマンエラーによるシステム破壊を避けるために、特定のパーティションをリードオンリーにして運用することがあります。リードオンリーにしたパーティションは**図14**や**図15**のようにmountに「-o マウントオプション」で動的に設定を変更できます。

mountではスワップの使用状況は表示されません。スワップ情報の確認には**swapinfo**コマンドを使います。**図16**の例ではちょっとだけスワップアウトが発生していることがわかります。

現在のシステムではメモリを十分に搭載することができるので、基本的にはスワップアウトを発生させないようにシステムを組みあげるのが、高速なシステムを構築する場合のキーポイントです。swapinfoを使ってスワップアウトが発生しているかどうかをチェックし、スワップアウトしているようであれば重いアプリケーションを終了させたり、システムを再起動したり、メモリの追加を検討します。

データベースソフトウェアなどにバグがある場合、使い続けると異様にメモリを消費することがあります。データベースに限らず、サーバのように動作し続けるタイプのソフトウェアはこの問題を発生させることがあります。swapinfoでスワップの状態を監視し、スワップアウトが異常なレベルに達していると判断した場合には、自動的にシステムを再起動するといった処理を仕込むことがあります。安易な方法ではありますが、実効性が高く効果的な方法です。

chflags

保護ドメインやアクセス制御リストで削除できないファイルを作ることはできますが、管理者権限では削除できます。MAC（Mandatory Access Control；強制アクセス制御）を使って管理者も削除できない状態にもできますが、これは規制が強すぎて不便だったりします。この中間的な機能として使えるのが、ファイルシステムの拡張機能を使う**chflags**コマンドです。

どうしても削除してはまずいファイルについてはchflagsで変更不可能のフラグを指定します（**図17、18**）。管理者権限で誤ってrmを実行するといったエラーからファイルを保護できます。

ジョブ、タスク管理

cron / crontab

サーバ管理の基本は、スクリプトで記述できることなど、人間が作業する必要がないものについては、すべて自動化してサーバに自動的に実施させることにあります。その場合、**cron**または**crontab**で自動的に実施するスクリプトやプログラムを指定

▼図14　書き込みを許可

```
% mount -u -o rw /
```

▼図15　書き込みを禁止

```
% mount -u -o ro /
```

▼図16　スワップ情報の確認例

```
% swapinfo
Device          1K-blocks     Used    Avail Capacity
/dev/ada0p3       4194304    35448  4158856     1%
```

▼図17　削除不可設定

```
% chflags schg DATA.20130408
% rm DATA.20130408
chflags: invalid flag: unschg
```

▼図18　削除可能設定

```
% chflags 0 DATA.20130408
% rm DATA.20130408
```

▼図19　crontab（設定ファイル）の記述例

```
*/4 * * * * root /MONITOR/MONITOR.001.SH > /dev/null 2>&1
*/3 * * * * root /MONITOR/MONITOR.002.SH > /dev/null 2>&1
*/5 * * * * root /MONITOR/MONITOR.003.SH > /dev/null 2>&1
```

して利用します(crontabはコマンド名と設定ファイル名が同じなので注意してください)。

図19の設定で、15分ごとに/MONITOR/MONITOR.001.SHが、20分ごとに/MONITOR/MONITOR.002.SHが、12分ごとに/MONITOR/MONITOR.003.SHが実行されるようになります。

このコマンドはさまざまな使い方ができます。たとえば、“アクセスログをチェックし、sshポートに対して不正アクセスを試みてくるIPアドレスをドロップの対象として、ファイアウォールの設定をアップデートする”といった処理をするスクリプトを10分おきに実行できたりします。

/etc/crontabにはシステムワイドなジョブを追加します。書き変えたあとは「service cron restart」としてcrondを再起動します。ユーザ権限で動作させる場合、対象となるユーザで「crontab -e」としてエディタを起動し、ジョブスケジュールを書き込みます。なんらかの出力がある場合、実行されるごとにメールが送られてきますので、それを避けたい場合には上記のようにリダイレクトします。

ユーザ管理

adduser / passwd / pw

ユーザの追加には**adduser**コマンド、パスワードの変更には**passwd**コマンドを使います。adduserとpasswdはインタラクティブ型のコマンドで、対話的に入力しながら使います。管理するユーザ数が少ない場合にはこれでもよいのですが、数百といった単位で管理するとなると、この方法には限界があります。また、繰り返し同じ環境をセットアップするスクリプトを制作する場合にも、インタラクティブ型のコマンドは使えません。この場合、**pw**コマンドを使います(**図20、21、22**)。

作成したユーザにデフォルトの設定ファイルがほしい場合、/usr/share/skel/にテンプレートがありますので、これをコピーします。ランダムなパスワードは**apg**コマンドを使えば簡単に作成できます。apgはPorts Collectionのsecurity/apgからインストールします。

ツール・ユーティリティ

bash / zsh

サーバ管理業務はターミナルで行います。作業効率の引き上げのために高機能シェルを使います。これもなんでもよいのですが、シェルスクリプトと同じほうが混乱が少なくてよいでしょうから、sh系(POSIX.1/POSIX:2008)のシンタックスを採用しているbashかzshあたりがよいでしょう。bashはshells/bashから、zshはshells/zshからインストールします。

tmux

安定したネットワークと大きなスクリーンのデュアルヘッドなど十分なスクリーンサイズが確保されている場合、複数のターミナルソフトウェアを起動してssh経由で作業できますが、ネットワーク

▼図20　グループ追加

```
% pw groupadd -n daichi -g 1001
```

▼図21　ユーザ追加

```
% pw useradd -n daichi -u 1001 -m -d /home/daichi -g 1001 -s /bin/sh
```

▼図22　パスワード設定

```
% printf "パスワード" | pw usermod -n daichi -h 0
```

が不安定だったりノートPCなどディスプレイサイズに限りがある場合、tmuxなどを併用します。

tmuxを使うことでコマンドの実行を中断することなく接続・切断を繰り返したり、複数のターミナルを単一のウィンドウの中に保持できます。tmuxはsysutils/tmuxからインストールします。

alias

サーバ管理業務はパターン化した操作が多く、作業の自動化が簡単です。多用する処理はスクリプトを記述したり、シェルのaliasに登録するなどして処理のショートカットを用意します(**図23**)。

kbdmap

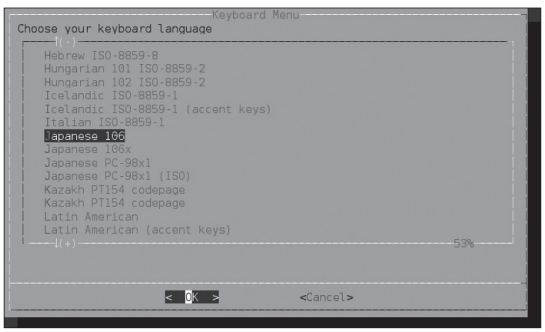
FreeBSDは英語キーボードをデフォルトのキーボードとしています。日本語キーボードをサーバルームに持って行って作業した場合、日本語キーボードを英語キーボードと解釈するため、キーに印字されているものとは別のキーが入力されることがあります。

サーバ管理者はこうした状況に遭遇することが多いため、日本語キーボードのまま英語キーボードの配列で入力するという技能を身につけてしまっていることがありますが、キーマップを変更すれば済む話ですので、変更します。kbdmapコマンドを引数なしで実行すれば選択画面が表示されますので、そこから選択します(**図24**)。

おわりに

サーバ管理業務の基本は、自動化できることは自動化し、人間が繰り返しの操作を行うようなことを避けることです。作業しながら人力で繰り返し作業をしているなと感じたら、一度作業を止め、その作業を自動化することを考えてみてください。ここで取り上げたコマンドはそうした自動化に貢献するツールになるはずです。SD

▼図24　キーボード選択画面



▼図23　aliasの記述例

```
% alias
backup='/TOOLS/BACKUP.SH'
df='df -h'
du='du -h'
edit_ie6='vim /CONTENTS/IE6HACK.CSS'
edit_ie67='vim /CONTENTS/IE67HACK.CSS'
edit_ie7='vim /CONTENTS/IE7HACK.CSS'
edit_js='vim /CONTENTS/MENU.JS'
edit_main='vim /CONTENTS/MAIN.CSS'
edit_menu='vim /CONTENTS/MENU.CSS'
edit_temp='vim /CONTENTS/TEMPLATE.HTML; update_errors'
la='ls -a'
ll='ls -al'
ls='ls -G -w'
su='su -l'
update_errors='/TOOLS/ERROR404MAKE.SH; /TOOLS/ERROR400MAKE.SH'
update_event='/TOOLS/EVENTMAKE.SH'
update_news='/TOOLS/NEWSMAKE.SH'
update_top='/TOOLS/TOPMAKE.SH'
update_whatsnewrss='/TOOLS/RSSMAKEWHATSNEW.SH'
where='command -v'
which-command=whence
```

Column 3 Linuxのパフォーマンスモニタのおさらい

大久保 修一　*OHKUBO Shuichi*　さくらインターネット（株）　Twitter @jq6xze_1

はじめに

普段、サーバの運用をしていると、ユーザからのアクセスが増え、負荷が増大して遅くなったり、サーバの操作が重くなったりすることがあります。その際、ただ闇雲にチューニングしていては効果が発揮されないばかりか逆効果になることもあります。まずはきちんとボトルネックを把握し、有効な対策をとれるようにしましょう。本稿では、筆者が普段使用しているLinuxのパフォーマンスモニタを紹介します。

CPUの負荷を把握する

サーバの動作が重いと感じたら、まず始めに**top**コマンドでCPU負荷の高いプロセスを調べてみましょう。**図1**のように表示されるはずです。

さまざまな情報が表示されますが、8番目のフィールド（プロセス状態）に着目します。「R」と表示されるプロセスは、実行（可能）状態となっており、CPUリソースを消費しています。併せて9番目のフィールド（CPU利用率）を確認し、継続してパーセンテージが高くなっている場合は、CPUがボトルネックになっている可能性があります。

状態が「D」になっているプロセスは、ストレージのIO完了待ち状態が頻発しており、ストレージがボトルネックになっている可能性があります。後述するストレージの利用状況を併せて把握すると良いでしょう。

状態が「S」になっているプロセスは、単純にスリープしているか、ネットワークを介したデータの送受信などのイベント待ちの状態が多く発生しています。CPUには余裕があるが、通信が遅くなっている場合は、ネットワークがボトルネックになっている可能性があります。

CPU、もしくはストレージがボトルネックになっている場合、**vmstat**コマンドを用いて切り分けを行います。実行例を**図2**に示します。

右から3番目のフィールド（cpuのid）がCPUの空きぐあい（%）を、右から2番目のフィールド（cpuのwa）がストレージのIO完了待ちの割合（%）を示して

▼図1　topコマンドの実行例

```
[root@test ~]# top
top - 10:23:00 up 136 days, 16:00,  2 users,  load average: 0.51, 0.95, 0.67
Tasks: 378 total,   2 running, 376 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  1.0%sy,  0.0%ni, 98.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  16325776k total, 16155300k used,   170476k free,   139088k buffers
Swap: 18563064k total,  1115420k used, 17447644k free,  5219060k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
   99 root      25   5     0    0    0 R  5.0  0.0   9579:25 ksmd
  419 root      20   0  511m 244m  876 S  0.3  1.5 205:50.17 qemu-kvm
27159 root      20   0 15264 1540 1000 R  0.3  0.0   0:00.07 top
…（略）…
```

います。この例ではCPUリソースは消費していないもののIO完了待ちが発生しており、ストレージがボトルネックになっている様子がうかがえます。

なお、マルチコアのCPUを使用している場合、vmstatは全コアの利用率を平均した値を示します。併せて**mpstat**コマンドを用いて、コアごとの利用率を把握すると良いでしょう。**図3**の例では、1つのコア(#1)に処理が偏って頭打ちし、マルチコアを活かせていないデーモンが存在することが確認できます。

特定コアに処理が偏っている場合、デーモンのプロセス数やスレッド数を増やすなどの対策が有効です。もし増やせない場合やすでにすべてのコアを使い切っている場合には、CPUのスペックアップやサーバの負荷分散といった対策が必要になるでしょう。

メモリの利用率を把握する

現在のメモリの利用率を把握するには、**free**コマンドを使うのが手軽です。**図4**のように、「-m」オプションをつけるとMB単位で表示されます。

この例では、トータル16GBの搭載メモリのうち、空きメモリは169MBで、ほぼすべて使用されていることがわかります。ただし、ストレージのキャッシュに使用されている空間(buffersとcached欄)が5GB程度存在するので、プロセスに割り当てられている実際の消費メモリは、差し引き10GB程度であることがわかります。3行目の値は、補正した値が表示されるため、この空きメモリが少なくなっている場合は、メモリの増設といった対策が必要になるでしょう。

4行目はSwap領域の利用量です。メインメモリから1GB程度のページが追い出されていることがわかります。休眠プロセスのページが多少追い出される分には問題ありませんが、アクティブなページが追い出されると、パフォーマンス劣化につながります。Swapの利用量が増えた場合は注意しましょう。

ストレージの負荷を把握する

ストレージへのアクセス状況を把握するには**iostat**コマンドを用います。**図5**のように実行すると、1秒間隔でブロックデバイス毎の負荷状況が表示

▼図2　vmstatコマンドの実行例

```
[root@test ~]# vmstat 1
procs -----------memory----------   ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache     si   so    bi    bo   in   cs us sy id wa st
 0 34 1115420 157724 138616 5224504    0    0     4    12    0    0  1  1 96  2  0
 1 20 1115420 157700 138620 5224764    0    0     0   564 2040 3579  0  1 62 37  0
 0  0 1115420 157584 138624 5224992    0    0     0   304 1988 3292  0  1 71 28  0
…(略)…
```

▼図3　mpstatコマンドの実行例

```
[root@test ~]# mpstat -P ALL 1
11:06:59     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest   %idle
11:07:00     all   12.55    0.00    0.12    0.00    0.00    0.00    0.00    0.00   87.33
11:07:00       0    0.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00   99.00
11:07:00       1  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
11:07:00       2    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
…(略)…
```

▼図4　freeコマンドの実行例

```
[root@test ~]# free -m
             total       used       free     shared    buffers     cached
Mem:         15943      15773        169          0        136       5098
-/+ buffers/cache:      10538       5404
Swap:        18127       1089      17038
```

されます。

r/sは秒間あたりの読み込み、w/sは秒間あたりの書き込みIO数（IOPS）を示しています。データベースのようにランダムアクセスが発生するワークロードの場合、一般的なHDD単体では200～300IOPSが限界となります。ストレージの性能はドライブの種類（SSDやHDD）やRAID構成によって大きく変わりますので、事前に計測し、限界に近づいていないか確認しましょう。もし、プロセス状態がIO完了待ち（D）になるようであれば、ストレージの高速化が必要になります。

ネットワークの流量を把握する

ネットワークの流量をリアルタイムに把握するには、**vnstat**コマンドが便利です。eth0の流量をリアルタイムに表示する例を**図6**に示します。

もし帯域が不足している場合は、NICやスイッチの広帯域化が必要になるでしょう。一方、帯域は不足していないのに思ったようにスループットが出ない場合は、途中の経路で輻輳していたり、パケットロスが発生している可能性もあります。

ボトルネックの調査は難しい

本稿では、Linuxにおける主要なパフォーマンスモニタの使い方を紹介してきました。ただし、実際の現場では目に見えないボトルネックがたくさんあり、追跡が非常に難しいケースもあります。とくにストレージはワークロードのパターン（ランダム、シーケンシャル）で10倍以上も性能が変化しますし、インターネットを経由した通信では、経路の途中で何が起きているかまったくわかりません。

しかしながら、「調査の入り口」として今回紹介したコマンドをマスターしておけば、その後の切り分けがスムーズに進むはずです。SD

▼図5　iostatコマンドの実行例

```
[root@test ~]# iostat -dmxt 1
04/15/13 21:29:52
Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.00   178.00    0.00  161.00     0.00     1.32    16.84     3.13   19.52   6.21 100.00
dm-0              0.00     0.00    0.00  339.00     0.00     1.32     8.00     9.24   27.30   2.95 100.00
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
dm-2              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
…（略）…
```

▼図6　vnstatコマンドの実行例

```
[root@test ~]# vnstat -i eth0 -l
Monitoring eth0...    (press CTRL-C to stop)

   rx:   256.63 Mbit/s 22665 p/s          tx:     4.38 Mbit/s  8423 p/s
```

NOTE　熟練者向けのコマンド

どうしてもボトルネックがわからない場合、最終手段として流れているデータをダンプすることがあります。ストレージの場合、**blktrace**コマンドを使います。以前、IO発行に遅延が生じる事象をこのコマンドで調べた結果、IOスケジューラに起因していることが判明したケースがありました。

ネットワークの場合、**tcpdump**コマンドやWiresharkを用いてパケットの中身や送受信のタイミングを確認します。ドライバのバグでチェックサムが正しく生成されていなかったり、回線やルータの故障でビット化けが発生していたり、中身を見てみなければわからないさまざまな事象を発見できました。

Ubuntu編

GUIが苦手とする作業を効率よく解決するために、デスクトップでもコマンドが活躍する

WindowsやMacのGUI操作に慣れてしまうと、仕事でUNIX系サーバでも扱わない限り、コマンド操作なんて縁遠いものだと思っていませんか? とくにデスクトップ環境ではほとんど考えたことがないと思いますが、実はコマンドラインのほうが楽できることがあるんです!

水野 源 *MIZUNO Hajime* Ubuntu Japanese Team Mail mizuno-as@ubuntu.com

デスクトップOSでもコマンドラインを使うワケ

本誌読者の中には、この春から仕事で、あるいは研究で、UNIX/Linuxを使いはじめたという方も多いと思います。1ヵ月が経ち、少しはUNIXの世界に慣れてきたころでしょうか?

さて、本特集でも取り上げたRed Hat Enterprise LinuxやCentOS、FreeBSDといったOSは、どちらかと言えばサーバ用途として使われることが多いOSです。通常サーバにはGUIをインストールせず、コマンドラインから操作するのが普通です。ですのでそのようなOSを業務で使う場合、WindowsやMacのデスクトップからSSHで接続して操作することになるでしょう。ですが場合によってはデスクトップでもUNIX系のツールを使うために(あるいは新人が早くUNIX環境に慣れるために)、デスクトップにもLinuxを使うというケースも、決して珍しくはありません。現在、そのような用途でまず最初に候補として挙げられるのがUbuntuではないかと思います。

Ubuntuは「誰でも使える」ことを目指して開発されているLinuxディストリビューションです。デスクトップとしても使いやすいUNIX系OSという特徴が注目され、最近では大学などで利用されるケースも増えてきました。

UbuntuのデスクトップはGUIだけで操作が可能なように設計されています。各種設定や運用に必要なツールにはGUIが用意されていますので、実際の運用でコマンドラインが必要になる場面はほとんどありません。またそのインターフェースは、WindowsやMacの利用経験があるユーザであれば、「なんとなく触っている」うちに使い方を身に付けることができる程度には直感的です。そんなGUIだけで事足りるUbuntuで、あえてコマンドラインを使う意味はあるのでしょうか?

答えはYesです。コマンドラインには、GUIにはないさまざまなメリットがあります。たとえば大量のデータの一括処理や処理の自動化は、マウスを使った手作業の苦手とする分野ですが、逆にコマンドラインが最もその力を発揮する分野でもあります。ネット上では「コマンドラインは旧世代の遺物」などといった意見も見られますが、これは大きな勘違いだと言えるでしょう。GUIとコマンドラインにはそれぞれ得手不得手があります。コマンドラインは前時代的な道具などではなく、GUIが苦手とする作業を効率よく解決するために、システムから与えられた強力な武器なのです。

この章ではUbuntuをより便利に使うための、デスクトップユーザにも役立つコマンドラインの活用法を紹介します。

コマンドを使って面倒な作業を簡単に

デスクトップでメーラー、ブラウザ、オフィススイートなどを利用して仕事をしている場合で、コマ

ンドラインが役立つシチュエーションはどこでしょうか。それは前述のとおり、データの一括処理や処理の自動化です。ここでは先日、筆者の職場で実際にあった例をもとに、コマンドを使って大量のテキストデータを処理、整形する方法を紹介します。

大量のデータを表にまとめたい!

筆者の職場では、多くのLinuxサーバを運用しています。先日、その中のメールサーバを利用しているお客さんから、次のような要求がありました。

「メーリングリストをたくさん運用させてもらってるんだけど、メーリングリストの一覧と人数、それと参加者のリストをもらえないかな。Excelで[注1]」

メーリングリストの設定は、サーバのmlディレクトリの中に格納されています。ディレクトリの中には、メーリングリストごとにサブディレクトリが切られており、その名前は「メーリングリストのアドレスのローカルパート部分」となっています(**図1**)。メーリングリストの参加者は、そのサブディレクトリの中のmembersというテキストファイルに、1行1ユーザの形式でリストアップされています(**図2**)。

ここで各メーリングリストのmembersファイルをテキストエディタで開き、Excelにコピー&ペーストしてもかまいません。メーリングリストの数が2個や3個でしたら、それで対応できるでしょう。しかし筆者がディレクトリの数を数えてみたところ、その数は700を越えていました。とても手作業で処理できる量ではありません。この時点で、コマンドを利用した一括処理を検討します。

テキストの整形を繰り返し実行する

最終的に、**表1**のようなシートを起こすことを考えてみます。ディレクトリ名にアドレスのドメインパートを追加すれば、メーリングリスト名が得られます。参加人数は、membersファイルからコメント行を除いたものの行数を数えればよいでしょう。メンバー一覧は、membersファイルからコメントを除いたものについて、改行をカンマ区切りに変換すれば1行にまとめられそうです。これらの情報を1行ずつ書き出してCSVファイルにできれば、目的は達成できそうです。

そこで筆者は、**図3**のようなコマンドを考えてみました。

まず複数のディレクトリを順番に処理する必要

注1) 最終的なデータをExcelやWordで要求されるのは、よくあることなのです……。

▼図1 メーリングリストの設定の構成

```
/var/spool/ml
├── admin   ← 各MLごとのサブディレクトリ
│   └── members   ← そのMLの参加者のリスト
├── manager
│   └── members
├── market
│   └── members
… (略) …
```

▼図2 メーリングリストの参加者のリストの例(membersファイル)

```
ikeuchi@example.com
#BYE horiuchi@example.com   ← 先頭が#ではじまる行はコメントとして扱われ無視される。MLから抜けた人をコメントとして残してある場合などがある
#BYE kida@example.com
amano@example.com
murakami@example.com
takizawa@example.com
```

▼表1 メーリングリストのリストを表にする

メーリングリスト名	人数	メンバー1	メンバー2	メンバー3	……
admin@ml.example.com	5	ikeuchi@example.com	amano@example.com	murakami@example.com	……
manager@ml.example.com	6	ooki@example.com	takaoka@example.com	akiyama@example.com	……
market@ml.example.com	9	oshita@example.com	miura@example.com	uchimura@example.com	……
… (略) …					

があるため、繰り返し構文のforを利用します。forはinの後に指定されたリストの内容を、1つずつ変数に代入しながら、doとdoneで囲まれた内容を繰り返し実行する構文です。ここではリストに「*（アスタリスク）」を指定しています。アスタリスクはワイルドカードなどとも呼ばれ、あらゆるパターンの文字列にマッチします。つまり、カレントディレクトリにある全ファイル（各メーリングリストのディレクトリ）をリストに指定したことを意味します[注2]。これでループが繰り返されるたびに、リストの要素（カレントディレクトリ内にあるファイル／ディレクトリ名）が順番に変数dirに代入され、doとdoneの間にあるechoコマンドが実行されます。

それでは、echoコマンドが実際に出力するデータを見ていきましょう。echoコマンドは、引数で与えられた二重引用符の中身を標準出力（ディスプレイ）へ出力しています。二重引用符で囲われた中には変数や他のコマンドがカラムごとに埋め込まれ、ここで各メーリングリストの設定を1行に整形しています。

1カラム目の処理

まず最初は、メーリングリスト名です。変数dirは、for文によってディレクトリ名が代入されていますので、その後にメーリングリストのドメインパート（@ml.example.com）と、区切り用のカンマを出力しています。これで表の1カラム目が出力できました。

2カラム目の処理

次は2カラム目、メーリングリストの参加人数です。これはサブディレクトリ内のmembersファイルから、コメントを除外した行数をwcコマンドで数えることで実現しています。grepコマンドは、ファイルや標準入力からパターンにマッチする行を探すコマンドですが、「-v」オプションをつけると動作が逆転し、パターンにマッチしない行を探すことができます。これを利用し、行頭が「#」ではじまらない行、つまりコメントではない行を探しています。wcコマンドは通常、行数、単語数、バイト数を数えますが、「-l」オプションをつけると行数のみを数えることができます。これをパイプでつないでgrepの結果の行数を数えています。

また、このコマンド全体を「$()」で囲んでいます。これは**コマンド置換**と呼ばれ、この部分をコマンドの実行結果と置き換える機能です。つまりechoが実行される際には、$()で囲った部分全体は、$()内部に書かれているコマンドの実行結果、つまりwcコマンドの出力に置き換えられます。これで2カラム目、メーリングリストの参加人数を出力できました。

3カラム目以降の処理

3カラム目以降には、メーリングリストに登録されているメンバーのメールアドレスが並びます。これも2カラム目と同様に、コマンド置換を用いて実現しています。実行するコマンドもgrepでコメン

注2）このコマンドはmlディレクトリで実行することを前提にしており、またmlディレクトリ直下には、各メーリングリストのサブディレクトリ以外のファイルは存在しないことを想定しているため、アスタリスクを指定しました。もしも確実にディレクトリのみを抽出したいのならば、「ls -F | grep '/$'」などを併用してもよいでしょう。

▼図3　メーリングリストごとの情報を1行で出力するコマンド例

```
for dir in *
do
echo "${dir}@ml.example.com,$(grep -v '^#' $dir/members | wc -l),$(grep -v '^#' $dir/members | tr ➥
'\n' ',')" >> ~/list.csv
done
```

▼図4　コマンドの実行例

```
admin@ml.example.com,4,ikeuchi@example.com,amano@example.com,murakami@example.com, ……
manager@ml.example.com,5,ooki@example.com,takaoka@example.com,akiyama@example.com, ……
market@ml.example.com,9,oshita@example.com,miura@example.com,uchimura@example.com, ……
nagoya@ml.example.com,8,tominaga@example.com,kawahara@example.com, ……
…（略）…
```

トを除外するところまでは同じで、パイプでつなぐコマンドだけが異なっています。

trコマンドは特定の文字を変換、あるいは削除するコマンドで、1番目の引数に変換対象の文字を、2番目の引数に変換後の文字を指定して使います。ここでは改行(\n)をカンマに変換することで、1行ごとに1ユーザが記述されたリストを1行にまとめています。これで3カラム目以降の、カンマ区切りのユーザ一覧が出力できました。

ファイルへの出力処理

最後の「>>」は**リダイレクト**と呼ばれる、出力先を切り替える機能です。出力リダイレクトには「>」と「>>」の2種類があり、「>」は都度ファイルを上書きし、「>>」はファイル末尾に追記していくという違いがあります。今回はループの度に上書きされては困るので、「>>」を利用しています。これでコマンドの実行結果を、ディスプレイからホームディレクトリの「list.csv」というファイルに切り替えています。

水平方向にファイルを結合する

前述の例では、各メーリングリストの情報を行(横方向)にまとめました。これは行単位でデータを読み書きする、UNIXコマンドのしくみから考えて自然な動作だからです。しかし場合によっては**表2**のように、情報を列(縦方向)にまとめたいという要望もあるかもしれません。もしも表2のような表をループで出力しようとしたら、どうなるでしょう?

まず1行目に各メーリングリストのアドレスを表記するために、すべてのサブディレクトリ名を調査する必要があります。また2行目を書き出すためには、また全メーリングリストのmembersファイルを横断して検索し、行数を数えなくてはなりません。3行目以降はmembersファイルの内容を1行ずつ順番に出力していかなければなりませんが、メーリングリストごとに行数は違いますから、そこも考慮しなくてはなりません。このように、少々処理が複雑になってしまうでしょう。

そこで、あらかじめ各カラム単位の情報をまとめておき、最後に結合するというアプローチを考えてみます。

まず、各カラムの処理を**図5**のように考えました。処理は先ほどと同じように、forでディレクトリごとにループして行います。今回は縦方向に情報を並べればよいので、trコマンドによるテキスト整形などは不要です。変数と各コマンドの実行結果を、**cat**コマンドとヒアドキュメントを使って、「ML名.tmp」という一時ファイルに書き出しています。

「<<EOF」という記述がヒアドキュメントです。これは「<<」の直後に指定した文字列が表れるまでの内容を、コマンドへの標準入力として扱うという機能で、複数行の入力を扱う際に便利です。catコマ

▼表2　行列を逆にした表

メーリングリスト名	admin@ml.example.com	manager@ml.example.com	market@ml.example.com	……
人数	5	6	9	……
メンバー1	ikeuchi@example.com	ooki@example.com	oshita@example.com	……
メンバー2	amano@example.com	takaoka@example.com	miura@example.com	……
メンバー3	murakami@example.com	akiyama@example.com	uchimura@example.com	……
… (略) …				

▼図5　メーリングリストごとの情報をカラムにまとめるコマンド例

```
for dir in *
do
cat > ~/${dir}.tmp <<EOL
${dir}@ml.example.com
$(grep -v '^#' $dir/members | wc -l)
$(grep -v '^#' $dir/members)
EOL
done
```

ンドは標準入力から受け取った内容を、標準出力へ出力するコマンドです。ヒアドキュメントと前述のリダイレクトを組み合わせることで、複数行からなる文字列をファイルに出力しています(**図6**)。

最後に、出力した一時ファイルを**paste**コマンドで結合します(**図7**)。pasteコマンドは引数に指定されたファイルの内容を、デリミタとして指定された区切り文字をはさんで水平方向に結合するコマンドです。CSVファイルとして出力したいので、ここではデリミタ(-dオプション)にカンマを指定しました。pasteコマンドは各ファイルの行数の違いも考慮し、末尾に到達してしまった場合はデリミタのみを出力してくれるため、表が崩れるようなことはありません。

LibreOfficeでファイルを変換する

できあがったCSVファイルをLibreOffice CalcやExcelにインポートすれば、期待していた表を得ることができるでしょう。ですがせっかくコマンドラインを使っているのですから、ファイルの変換も自動でやってしまいたいところです。Ubuntuに標準搭載されているLibreOfficeは、実はコマンドラインからファイルのコンバータとして利用することもできます。それではLibreOfficeをコマンドラインから実行して、ファイルを変換してみましょう(**図8**)。

LibreOfficeの実行コマンドは、「/usr/bin/soffice」です。「--help」を見るといくつものオプションがあることがわかりますが、今回はその中にある「--convert-to」を利用します。これは文字どおり、ファイルを変換するオプションです。オプション引数として変換先のファイル形式を指定します。今回のお客さんはExcel形式をご所望でしたので、xlsを指定しました[注3]。また「--headless」は、LibreOfficeをヘッドレスモード(ユーザインターフェースを使用しないモード)で起動するオプションです。今回はコマンドラインからファイルの変換をするだけなので、不要な画面を表示しないために利用しています。**図9**が出力結果を開いたものです。

テキスト一括処理のまとめ

今回はループを使ったテキストの一括処理のみを駆け足で紹介し、あまり凝ったテクニックやコマンドは紹介していません。ですが実際のところ、ループ、パイプ、コマンド置換という基本パターンさえ覚えれば、大抵の処理は記述することができます[注4]。あとはUNIX/Linux環境に用意されている無数のコマンドを組み合わせて、バリエーションを展

▼図6　一時ファイルの例

```
yokohama@ml.example.com
4
fukasawa@example.com
kai@example.com
nagano@example.com
tamaki@example.com
↑これが表2の1カラムに相当する
```

▼図7　pasteコマンドでファイルを水平方向に結合する

```
$ paste -d ',' ~/*.tmp
admin@ml.example.com,manager@ml.example.com,market@ml.example.com, ……
4,5,9, ……
ikeuchi@example.com,ooki@example.com,oshita@example.com, ……
amano@example.com,takaoka@example.com,miura@example.com, ……
murakami@example.com,akiyama@example.com,uchimura@example.com, ……
takizawa@example.com,watabe@example.com,arita@example.com, ……
,fukazawa@example.com,hashino@example.com, ……
,,kiyota@example.com, ……
…(略)…
```

▼図8　LibreOfficeでCSVをxlsに変換する

```
$ soffice --headless --convert-to xls list.csv
convert /home/mizuno/list.csv -> /home/mizuno/list.xls using
```

注3) 同様の方法で、PDFを作成することも可能です。

注4) もちろん少々効率の悪いスクリプトになってしまうこともありますが、それが問題になるケースもあまり多くはないでしょう。

開していくだけです。それができれば、デスクトップ環境で必要になるようなデータ処理のほとんどは、より便利に、効率よくこなせるようになるはずです。

デスクトップを使ったドキュメント作業であっても、コマンドを使って少し工夫する余地はないか。考えてみるとよいかもしれませんね。

CLI環境をカイゼンしよう

Ubuntuにはコマンドラインをより便利に使うための、Ubuntuならではのしくみがいくつも用意さています。ページの都合ですべてを紹介することはできませんが、これは便利、と筆者が思うものをピックアップして紹介します。

Byobu

皆さんはGNU screenやtmuxを使っているでしょうか？　screenやtmuxはターミナルマルチプレクサなどとも呼ばれる、複数の端末の同時利用、端末の内容のコピー&ペースト、端末のログの取得などを可能にしてくれる、コマンドラインを使いこなすうえで必須とも言えるツールです。そしてscreenやtmuxをより便利に使うためのフロントエンドが、Ubuntu発の「Byobu」です[注5]。

Byobuを起動するには、ターミナルから**byobu**コマンドを叩くか、UnityのDashから「Byobu Terminal」を検索して実行します（**図10**）。ターミナルの最下行にウィンドウの一覧や、システムの各種情報がグラフィカルに表示されるのがわかるでしょうか。screenやtmuxは、標準では最低限の設定しかなく[注6]、“使いやすい”環境に育てるのはそれなりの調整が必要です。Byobuは最初から必要な（やや過剰かもしれませんが）情報が表示されるよう設定がなされているため、インストールするだけで作業を開始することができます。また設定用のメニューが用意されており、チェックをオン／オフするだけで表示する項目のカスタマイズができるようになっています。

screenやtmuxは「不意の回線切断からセッションを保護する機能」を提供してくれるため、リモートでサーバを操作する場合には必須です。ですがせっかくですので、Ubuntuをリモートから操作する場合には生のscreenやtmuxではなく、Byobuの利用を検討してみるとよいでしょう。

run-one

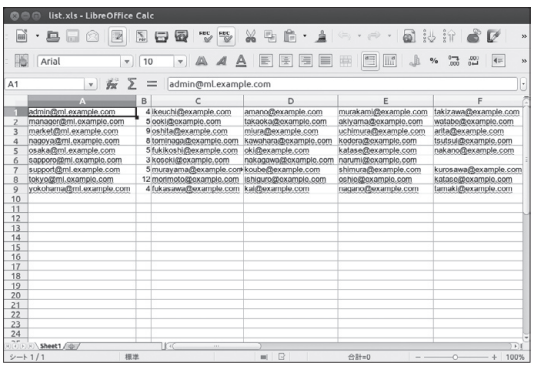
あるコマンドを実行する際に、すでに同じコマンドが実行中であれば何もしないで終了したい……。つまり同じコマンドを重複して起動したくない、というシチュエーションはよくあると思います。こういう場合は、/run[注7]以下に<プロセス名>.pidという

注5）「Byobu」パッケージをインストールしておく必要があります。なおUbuntu Serverには標準でインストールされています。

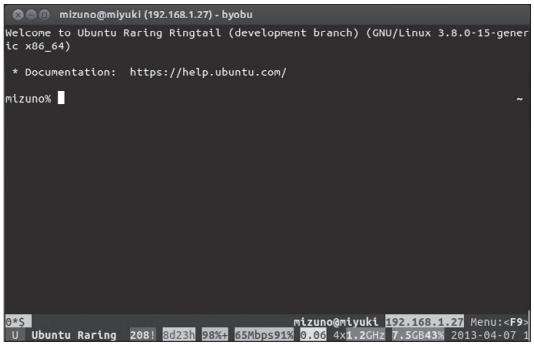
注6）screenに至っては、標準の状態ではウィンドウの一覧すら表示されません。

注7）少し古いシステムの場合は、/var/run以下が使われます。Ubuntu 13.04では/var/runが/runへのシンボリックリンクになっています。

▼図9　CSVからコマンドで変換したExcelのシート



▼図10　Byobuのスクリーンショット



ようなロックファイルを作成し、ロックファイルが存在する(=コマンドが実行中)の場合は何もしない、というような例外処理を設けるのが普通です[注8]。ですがこういったしくみを設けるにはそれなりの分量のシェルスクリプトを書く必要があり、単発でコマンドを実行したい場合には不向きです。

Ubuntuには「run-one」というラッパーコマンドが用意されています[注9]。run-oneは引数に実行したいコマンドを指定して使います。run-oneは内部で**flock**コマンドを呼び出してロックファイルの排他ロックを取得し、指定されたコマンドを実行するというしくみになっています。自動的に作成されるロックファイル名には「実行されるコマンドと引数を合わせた文字列のMD5ハッシュ値」が利用されるため、同一のコマンドの重複起動を防止することができるようになっています。Ubuntuならではの便利機能として、覚えておくと役に立つでしょう。

alert

Ubuntuのデスクトップを利用していると、システムからのさまざまな通知情報が、半透明なポップアップで表示されることがあります(**図11**)。これはNotify OSD(On Screen Display)と呼ばれる機能で、notify-sendというプログラムをキックすることで、コマンドラインから任意のメッセージをOSDで表示させることが可能です。つまりCLIの出力をGUIで表示できるわけです。

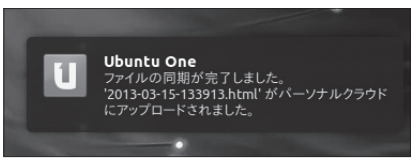
Ubuntuでは、最初からログインシェル(bash)にさまざまな設定が用意されているのですが、その中に「alert」というエイリアスがあります(**リスト1**)。これはnotify-sendのラッパーで、historyコマンドの最後の1行からsedコマンドで実際に実行されたコマンド名を抜き出して、notify-sendコマンドでデスクトップ上に表示させています。また変数「$?(直前のコマンドの成否)」をもとに、表示するアイコンを変化させています。つまり直前に実行したコマンドの内容と、その成否を表示するという機能を持っています。

図12の例のように、実行したいコマンドの後にセミコロンとalertを指定して実行します。するとコマンドの終了時にポップアップで成否が通知されます(**図13**)。とくに**dd**や**cp**コマンドで大きなファイルをコピーしたり、**wget**コマンドでISOイメージをダウンロードしたりといった、「時間がかかるので実行したら放置しておく」タイプのコマンドを実行する際に利用すると便利です。SD

注8) 典型的な/etc/init.d以下の起動スクリプトによく見られるしくみです。

注9) 「run-one」パッケージをインストールしておく必要があります。

▼図11 Notify OSDによる通知



▼図13 alertの表示例

▼リスト1 エイリアス「alert」の設定

```
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)"
"$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
```

▼図12 alertの使用例

```
$ wget http://ubuntutym2.u-toyama.ac.jp/ubuntu/quantal/ubuntu-12.10-desktop-amd64.iso; alert
```

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2013年5月号

第1特集
IT業界ビギナーのための お勧め書籍55+α
新しい季節に君へ

第2特集
覚えておきたい、ちゃんと使いたい!
正規表現をマスターしていますか?

一般記事
・バーチャルネットワークコントローラ2.0開発の実際

1,280円

2013年4月号

第1特集
僕（私）の言語の学び方
裏口からのプログラミング入門

第2特集
オブジェクト指向再入門
ソフトウェア開発に効くSmall Objectをご存じですか?

特別企画
・スクウェア・エニックス＋Skeed
「ゲーム開発の舞台裏」

1,280円

2013年3月号

第1特集
オープン環境でスキルアップ!
もっとクラウドを活用してみませんか?

第2特集
光、ギガビット、高速ネットワークを体験!
実践! ワイヤリングの教科書

一般記事
・「SSDストレージ」爆発的普及の理由

1,280円

2013年2月号

第1特集
UNIXコマンド、fork、pipeを復習し、高度なスクリプティングへ
シェルスクリプティング道場

第2特集
忙しいITエンジニアのための
超効率的勉強法

一般記事
・Samba 4.0.0ファーストインプレッション

1,280円

2013年1月号

第1特集
いざというときに備える
システムバックアップ

第2特集
IT業界のキーパーソンに聞く
2013年に来そうな「技術」・「ビジョン」はこれだ!

特別付録
法輪寺電電宮情報安全護符シール

1,380円

2012年12月号

第1特集
判断をあおぐ／経緯を説明する／手順の理解を得る文章を書くためのアタマの整理術
なぜエンジニアは文章が下手なのか?

第2特集
高速・高機能HTTPサーバ
Nginx構築・設定マニュアル

一般記事
・エブリデープログラマの発想と実践

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。

春の嵐吹く
リアルタイム分散処理
Storm

日々発生する大量なデータをリアルタイムに処理し続ける「ストリームデータ処理」に対するニーズが高まっています。同じビッグデータでもバッチ処理のHadoopとはまた違った解決方法が求められる分野です。本記事ではそのストリームデータ処理を実現するプロダクトとして、今、注目を集めている「Storm」について解説します。

Acroquest Technology㈱
鈴木 貴典 SUZUKI Takanori

Twitter社が公開したオープンソース

「Storm」は、Twitter社が公開しているオープンソースプロダクトであり、耐障害性に優れたリアルタイムの並列分散処理を簡単に実現するためのフレームワークです。もともとは、Twitterのつぶやきを解析するシステムを開発していたBackType社のメンバが、ビッグデータをリアルタイムに処理するためのプラットフォームとして、Stormの開発を進めていました。そのBackType社を、Twitter社が2011年7月に買収し、その後、オープンソースとして公開されることになりました。

今回は、Stormとはどのようなものなのか、Stormを利用してどのようなことができるのか、その概要を紹介します。

ビッグデータ×リアルタイム＝ストリームデータ処理

Stormで実現するリアルタイム処理は、より正確に言うと、「ストリームデータ処理」というものに該当します。「ストリームデータ処理」とは、連続的に発生するデータ（これを、「ストリームデータ」と言います）をリアルタイムに処理をし続けることです（**図1**）。Twitterのつぶやきがそれに該当しますが、ほかにもネットワークで発生するトラフィックデータ、工場などで利用されるセンサーが生み出すデータ、気候や株価の変動情報なども、ストリームデータに該当します。

近年、ネットワークの普及やクラウド化が進んできたことにより、社会全体で扱うデータ量が爆発的に増加しています。そのため、その大量データをいかにリアルタイムに処理するか、ということで、ストリームデータ処理に対するニーズも高まっています。

Stormで何ができるのか？

Stormが生まれた背景

ビッグデータに対する処理と言えば、Hadoopが有名でしょう。Hadoopの登場によりビッグデータという言葉が広まった、といっても過言ではありません。しかしながら、Hadoopが対象とする分野は、基本的にはバッチ処理です。そのため、リアルタイムというニーズにはマッチングしにくい状況でした。

一方、従来のシステムでは、リアルタイムで分散処理を行う場合、MOM（Message Oriented Middleware）のようなメッセージング技術を利用して、メッセージのキューイングと非同期の処理を組み合わせて実現することが一般的でした。しかしながら、従来の技術では信頼性や拡張性を満たすためには、苦労す

る部分も多くありました。

Stormはそれらの課題を解決するために登場しました。Stormを利用することで、多大な苦労なく、大規模なリアルタイム分散システムを構築することが可能となります。

6つの特徴

Stormは次に示す6つの特徴があります（**図2**）。

①シンプルなAPI

StormのAPIは非常にシンプルです。複雑な分散処理などをとくに意識することなく、システムを開発可能です

②拡張性

Stormは複数のマシンで構成されるクラスタ上で並列分散的に動作します。これにより、膨大な数のメッセージに対しても低レイテンシを維持しつつ、スケールします。スケールするために必要なことは、マシンを増設して処理の並列数を増やすだけであり、プログラムを改修する必要はありません

③耐障害性

障害が発生し、データ処理中のノードがダウンした場合でも、Stormは必要に応じてタスクの再割り当てやノードの再起動を行います。そのため、処理が完全に停止してしまうようなことがありません[注1]

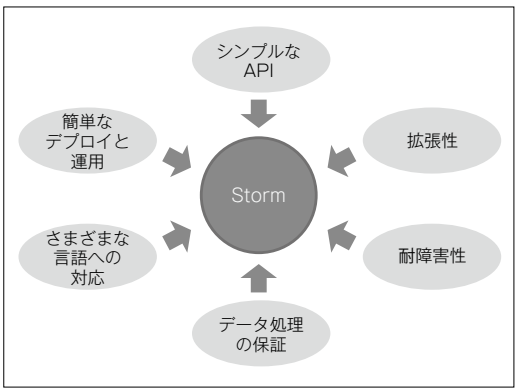
④データ処理の保証

何らかの理由により、データの処理に失敗したり、タイムアウトが発生したりした場合でも、Stormはそれを検知し、再処理するしくみを有しています。この機構により、すべてのメッセージが処理されることを担保できます

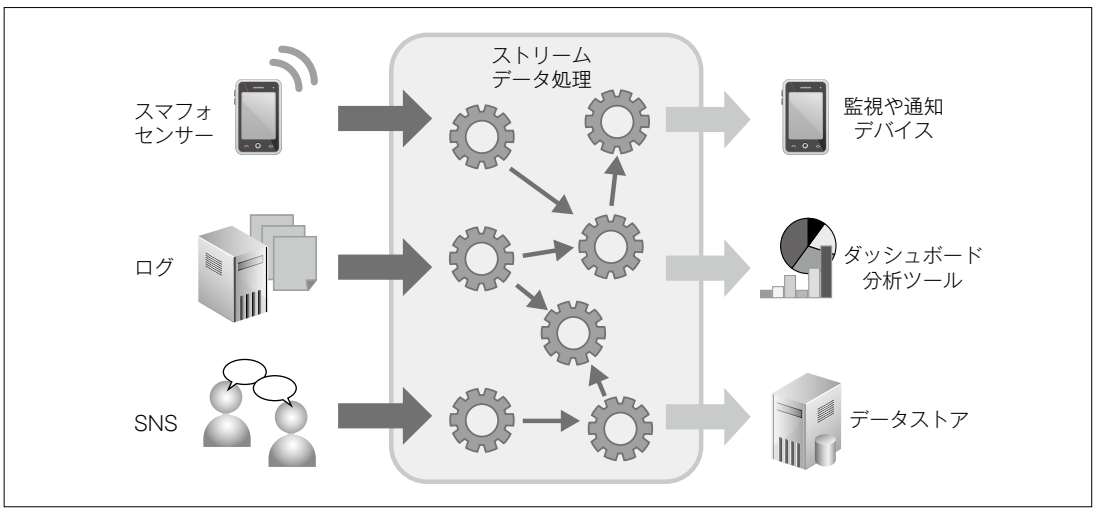
⑤さまざまな言語への対応

Storm自体はClojure[注2]で実装されていますが、ユーザが開発するアプリケーション部分は、いろいろな言語で開発できます。Java、Scala、Ruby、

■図2　Stormの6つの特徴



■図1　ストリームデータ処理



注1）処理のタイミング次第では、同一メッセージが重複して処理されることがあります。
注2）LISP系の言語の方言の1つ。関数型プログラミングのスタイルでのインタラクティブな開発を支援し、マルチスレッドプログラムの開発を容易化する汎用言語です。Java仮想マシン上で動作します。http://clojure.org/

Python、Perl、JavaScript、および、PHPなどの多くの言語をサポートしています

⑥簡単なデプロイと運用

Stormは簡単にデプロイし、動作させることが可能です。システム構成もわずかな設定で変更できます。また、Amazon EC2などのクラウド環境でも動作させられます

Stormでとくに興味深いのは、耐障害性やデータが完全に処理されることをサポートしている点にあるでしょう。このようなしくみが標準で備わっているため、ミッションクリティカルな分野にも適用しやすくなっています。

Stormでできること

ここではStormで実現できる、主な機能を紹介します。

①継続的な並列処理

Stormの基本的な機能です。通常はストリームデータであるメッセージを一時的にキューに保存し、そのメッセージを継続的、かつ、並列的に処理し続けます

②メッセージのグルーピング

メッセージは特定のルールに従って、グルーピングして処理できます。グルーピングの方法としては、ランダム(Shuffle grouping)、指定されたフィールドの値が一致するもの(Fields grouping)、全メッセージを処理するもの(All grouping)などがあります。Stormが標準で提供する7種類のグルーピングのほか、独自のグルーピングを実装することも可能です

③トランザクション

前述のとおり、Stormはメッセージが必ず処理されることを保証します。しかしながら、障害が発生し、メッセージが再送された場合、重複して処理される可能性があります。たった一度だけ処理をしたい場合は、トランザクション機能(Transactional topologies)を利用します。トランザクション機能を利用した場合、Stormで処理されるメッセージには、トランザクションIDが自動で付与されます。このトランザクションIDを利用することで、重複する処理を判別することができます

Stormの基本アーキテクチャ

Stormのプロセス構成、および、アプリケーション構成について説明します。

プロセス構成

Stormは複数マシンにまたがって動作させられます。それら全体を「クラスタ」と呼びます。Stormのクラスタは基本的にはMaster-Slave構成をとり、具体的には**図3**のような構成となります。

◎Nimbus

クラスタ内におけるMasterノードであり、SupervisorやWorkerプロセスの管理を行います

◎Supervisor

クラスタ内におけるSlaveノードであり、タスク(後述のSpoutやBoltが該当します)のアサイン待ち受けや、Workerプロセスの起動/停止を行います

◎Worker

タスクを実行するプロセスになります

◎Zookeeper

ZookeeperはApache Foundationのプロダクトの1つであり、Hadoopのサブプロジェクトの1つとして開発されました。StormではNimbusとSupervisor間の協調管理に用いられています

アプリケーション構成

Storm上で動作するアプリケーションは、Topology(トポロジ)と呼ばれ、**図4**のようなイメージになります。Topologyは次の要素から構成されます。

◎Tuple

Stormで処理されるメッセージのことです。デフォルトではinteger、long、short、byte、string、double、float、boolean、byte配列などをサポートします

◎Stream

途切れずに連続するTupleを意味します。どのよう

なTupleが流れるかはグルーピングより決定されます

◎Spout

Streamのソースとなるもので、外部からデータを取得したり、受け付けたりして、Tupleを生成／送出します。Stormの処理の起点となるものです

◎Bolt

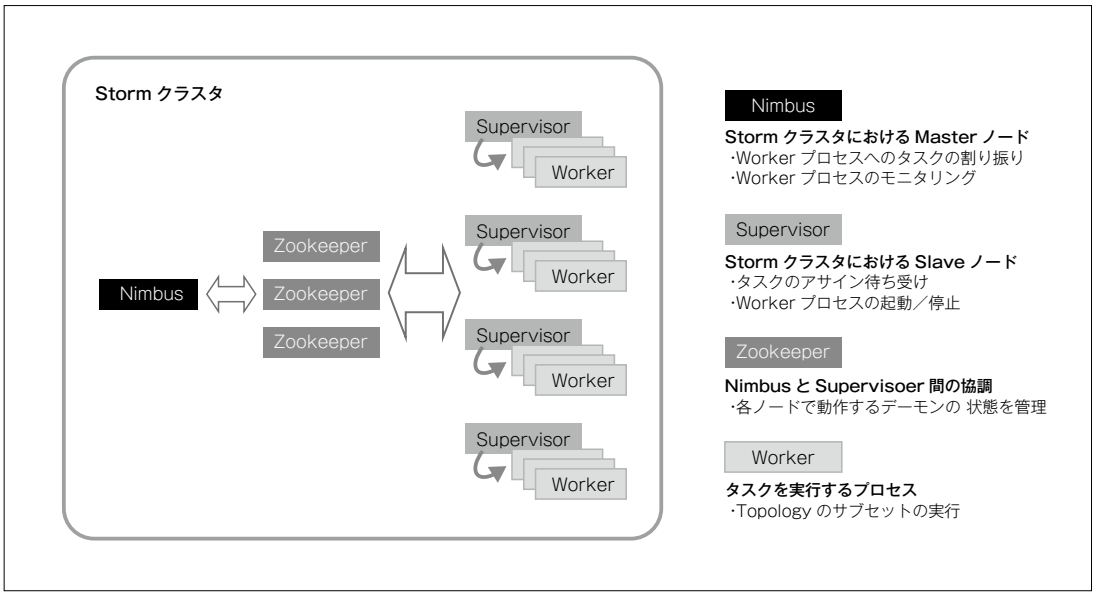
Streamの変換処理を行います。単一または複数のStreamからTupleを受信し、加工したうえで、新たなStreamにメッセージを送信します

Topologyはある一連の業務フローに該当するものです。Topology同士はStormクラスタ内で複数存在することが可能であり、それぞれのTopologyは独立して動作します。

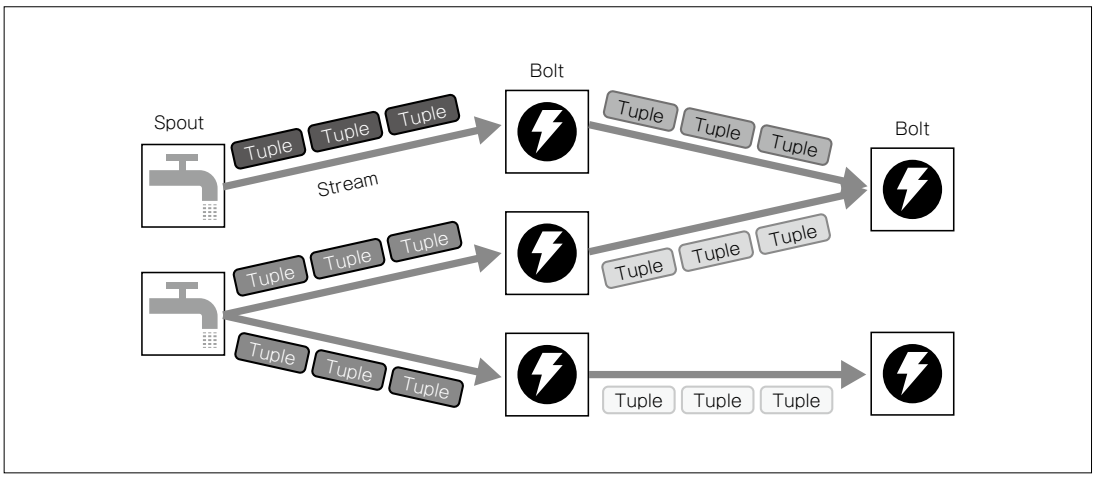
基本アーキテクチャ

Stormを利用したストリームデータ処理を行うシステムは、基本的には**図5**のようなアーキテクチャとなります。

■ **図3　Stormクラスタの構成**



■ **図4　Topology**



①イベント受付

処理対象のイベントは一時的にメッセージキューに保存するのが一般的です。これは大量のイベントを受信した場合でも、イベントを消失することなく処理を継続するためです

②イベント処理

次にメッセージキューに一時的に保存されたイベントをStormが取得し(Spout)、業務に応じた処理(Bolt)を分散して行います

③結果の処理

Storm自体は処理結果を表示する画面などは提供していません。そのため、Stormで処理をした結果はユーザへ通知されたり、RDBやKVS(NoSQL)などに保存されたりします

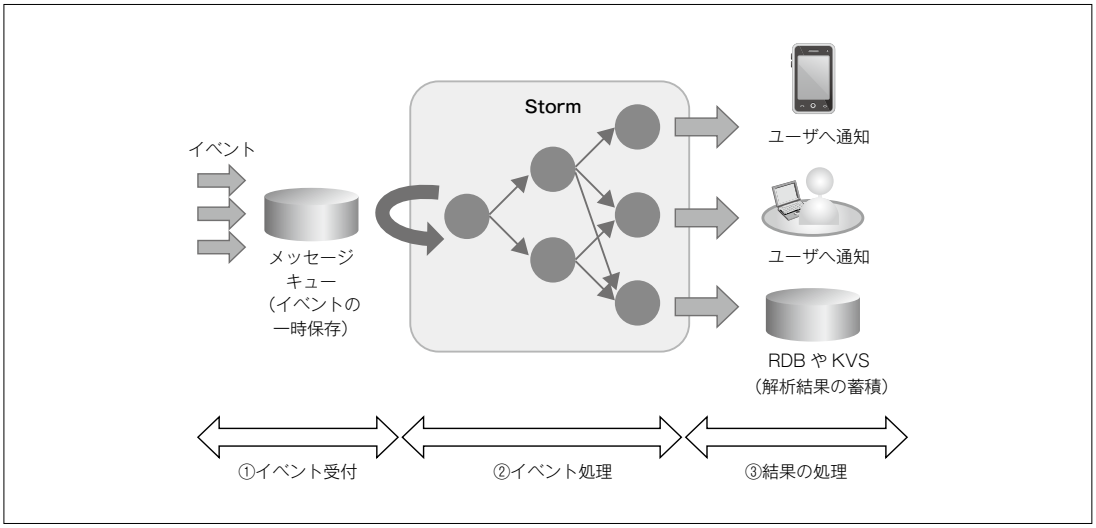
Storm自体はデータを永続化する機能や、ユーザに処理結果を通知するためのUIなどは、提供していません。それらが必要な場合は、システムに合わせて開発することが必要となります。

Hadoopとの比較

冒頭で、StormはHadoopでは困難なリアルタイム処理を実現するために開発された、と書きましたが、HadoopとStormを比較した場合のそれぞれの特徴を**表1**に示します。

アーキテクチャとしてはHadoopに似た構成となっている部分もあり、どちらも分散処理を行うフレームワークですが、最大の違いはバッチ処理なのか、リアルタイム処理なのか、というところにあります。

■ **図5　Stormを利用したシステムのアーキテクチャ**



■ **表1　HadoopとStormの比較**

	Hadoop	Storm
対象	バッチ処理	リアルタイム処理
処理技術	MapReduce	ストリームデータ処理
特性	・巨大で有限のジョブ ・たくさんのデータを一度だけ処理する ・長い待ち時間	・小さい無限のジョブ ・無限のストリームデータを連続して処理する ・短い待ち時間
ノード	マスター：JobTracker スレーブ：TaskTracker	マスター：Nimbus スレーブ：Supervisor
処理単位	Job	Topology

ただし、HadoopとStormは競合するものではありません。システムによっては、両者を組み合わせて処理を行うケースもあります。重要なことは、導入するシステムの特性を見極め、その特性に応じて使い分けることと言えるでしょう。

サンプルプログラム

次に、サンプルプログラムを用いてStormの実装内容を解説します。今回のサンプルプログラムは、文章を解析してそこに登場する単語数をカウントするものです。Stormのサンプルプログラムとして公開されているものがベースになっていますが、もともとSpout内部で文章を生成していた部分を、実際のシステムに近いイメージで処理するように、メッセージキューのOSSであるKestrelと連携して処理する構成に変更しています(**図6**)。

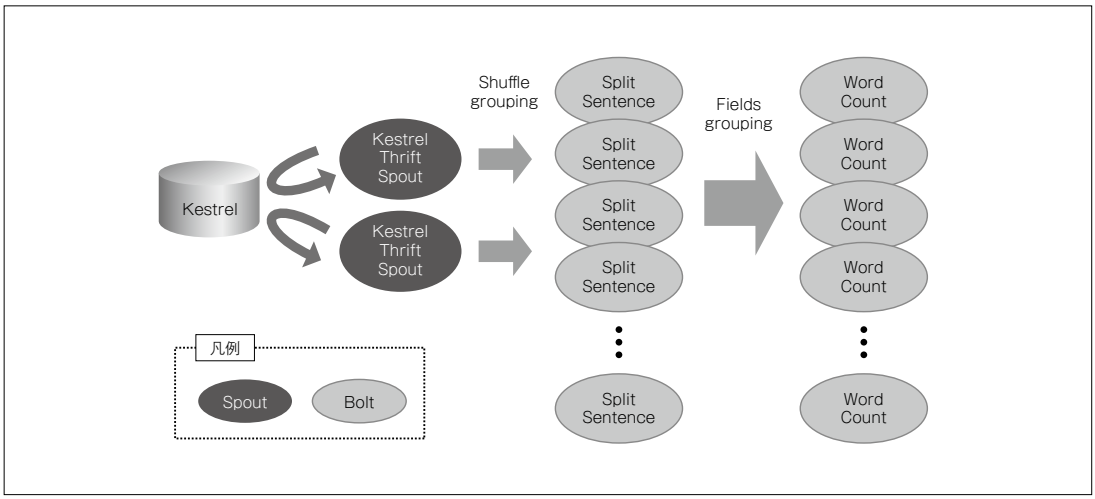
それぞれのコンポーネントの処理内容は、**表2**のようになります。以降では、各コンポーネントの実装について説明します。

データを取得する －KestrelThriftSpout－

Stormが提供するBaseRichSpoutを継承してSpoutクラスを実装します(**リスト1**)。KestrelThriftSpoutではおもに次の流れで処理を行います。

①Kestrelへ接続する
②Kestrelからデータを取得する
③Boltへデータを送信する

■ **図6　サンプルプログラムの処理イメージ**



■ **表2　コンポーネントの処理内容**

コンポーネント	処理内容
KestrelThriftSpout (Spout)	メッセージキューのKestrelから文章データ(今回は1行あたり約100バイトの英文データ)を取得する 取得した文章データを1行ずつBoltへ送信する
SplitSentenceBolt (Bolt)	1行の英文の文章を単語単位に分割する
WordCountBolt (Bolt)	分割された単語をカウントする

●openメソッド

最初に、Kestrelへ接続を行うため、Spoutのopenメソッドをオーバーライドします(**リスト2**)。このメソッドはSpoutが起動時に呼び出されます。ここでは、Kestrelとの接続を行うためのクラスであるKestrelClientInfoクラスをnewしています。

●nextTupleメソッド

次に、Kestrelからデータを取得するために、nextTupleメソッドをオーバーライドします(**リス**

■ **リスト1　KestrelThriftSpoutクラス**

```
public class KestrelThriftSpout extends BaseRichSpout {

  @Override
  public void open(Map conf,
                   TopologyContext context,
                   SpoutOutputCollector collector) {
       (中略)
  }

  @Override
  public void nextTuple() {
       (中略)
  }

  @Override
  public void ack(Object msgId) {
       (中略)
  }

  @Override
  public void fail(Object msgId) {
       (中略)
  }

}
```

■ **リスト2　openメソッド**

```
@Override
public void open(Map conf,
                 TopologyContext context,
                 SpoutOutputCollector collector) {
  this.collector = collector;

  (中略)

  int numTasks = context.getComponentTasks(context.getThisComponentId()).size();
  int myIndex = context.getThisTaskIndex();
  int numHosts = this.hosts.size();

  if (numTasks < numHosts) {
    for (HostInfo host: this.hosts) {
      this.kestrels.add(new KestrelClientInfo(host.host, host.port));
    }
  } else {
    HostInfo host = this.hosts.get(myIndex % numHosts);
    this.kestrels.add(new KestrelClientInfo(host.host, host.port));
  }
}
```

ト3)。tryEachKestrelUntilBufferFilledメソッド内(本稿では具体的な処理は割愛します)で、Kestrelからデータを取得し、一時的にemitBufferに英文データを格納しています。

その後で、collectorのemitメソッドを呼び出すことでデータをBoltに送出しています。collectorはStormのSpoutやBolt間でTupleを受け渡すためのクラスで、openメソッド中で初期化されています。

注意点としては、emitするオブジェクト(Tuple)はシリアライズ可能であること、また、一意のIDを指定して送出する必要がある点があります。

Spoutでは、上記の他に、Boltでの処理が成功/失敗した際に呼び出される、ack/failメソッドを必要に応じて実装します。

単語単位に分割する ー SplitSentenceBolt ー

今回はSplitSentenceBolt、および、WordCount Boltの2種類のBoltを実装していますが、最初にSplitSentenceBoltについて説明します。SplitSentenceBoltはStormが提供するBaseRichBoltを継承して、Boltクラスを実装します(**リスト4**)。SplitSentenceBoltでは、おもに次の流れで処理を行います。

①Boltを初期化する
②Tuple(1行の英文)を受信する
③1行の英文を単語に分割し、次のBoltへ送出する

●prepareメソッド

Boltを初期化するためのprepareメソッドをオーバーライドします(**リスト5**)。ここではcollectorをインスタンス変数に設定するだけです。

●executeメソッド

executeメソッドの引数であるTupleは、英文の1行のデータです。そのデータを単語単位に分割します。その分割された単語は次のBoltへ送出するために、Spoutのときと同様に、collectorのemitメ

■ **リスト3　nextTupleメソッド**

```
@Override
public void nextTuple() {
  if (this.emitBuffer.isEmpty()) {
    tryEachKestrelUntilBufferFilled();
  }

  EmitItem item = this.emitBuffer.poll();
  if (item != null) {
    if (this.immediateAck) {
      this.collector.emit(item.tuple);
    } else {
      this.collector.emit(item.tuple,
                          item.sourceId);
    }
  }

  // Sleep Interval
  Utils.sleep(this.tupleEmitInterval);
}
```

■ **リスト4　SplitSentenceBoltクラス**

```
public class SplitSentenceBolt extends BaseRichBolt implements IRichBolt {

  @Override
  public void prepare(Map stormConf,
                      TopologyContext context,
                      OutputCollector collector) {
    (中略)
  }

  @Override
  public void execute(Tuple input) {
    (中略)
  }

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    (中略)
  }
}
```

ソッドを呼び出しています(**リスト6**)。

ここでTupleが処理されることを保証するための機構について、簡単に説明しておきます。TupleがStormのTopology内を移動していく途中で、処理が失敗した場合は、Spoutに通知されるようにする必要があります。Tupleは、並列分散で処理されるため、Tupleの生成元の参照を含めておくことで、Spoutへの通知を行えるようにします。この方法を「アンカリング」と呼びます。

実際にこの「アンカリング」を行っているのは、collector.emit()のステートメントになります。元のTupleを引数に含めていますが、そうすることでTupleの発生元をトレースすることが可能となります。さらに、collector.ack()のステートメントが実行され、各Tupleの処理結果が通知されます。もし、Bolt内での処理に失敗した場合は、ackメソッドではなく、failメソッドを呼び出すことが必要となります(処理されるTupleは、必ずackかfailが実行される必要があります)。

●declareOutputFieldsメソッド

declareOutputFieldsメソッドでは、Boltで送出するTupleの内容にフィールド名を付与します(**リスト7**)。今回の例では、「new Values(targetWord)」と、フィールドはtargetWordの1つだけであるため、「new Fields("word")」と1つのフィールド名を指定していますが、複数指定することも可能です。フィールド名はBoltがメッセージを受信する際のグルーピングで利用されたり、次で処理を行うBoltでフィールド名を指定して値を取得するのに利用されたりするのに必要となります。

■リスト5　prepareメソッド

```
@Override
public void prepare(Map stormConf,
                    TopologyContext context,
                    OutputCollector collector) {
  this.collector = collector;
}
```

■リスト6　executeメソッド

```
@Override
public void execute(Tuple input) {
  // 文章を単語単位に分割する
  String sentence = input.getStringByField("str");
  String[] words = StringUtils.split(sentence);

  // 単語単位にTupleに分割し、次のBoltに送信する
  for (String targetWord : words) {
    this.collector.emit(input, new Values(targetWord));
  }

  this.collector.ack(input);
}
```

■リスト7　declareOutputFieldsメソッド

```
@Override
public void declareOutputFields(
                OutputFieldsDeclarer declarer) {
  declarer.declare(new Fields("word"));
}
```

単語をカウントする －WordCountBolt－

WordCountBoltはBaseBasicBoltを継承しています(**リスト8**)。SplitSentenceBoltが継承していたBaseRichBoltクラスとの違いは、prepareメソッドを必要としない点です。WordCountBoltでは、おもに次の流れで処理を行います。

①Tuple(1単語)を受信する
②単語の出現回数をカウントする

●executeメソッド

SplitSentenceBoltでは､"word"というフィールド名を付与してTupleを送出していたので、ここではそのフィールド名を使ってTupleから単語データを取得し、カウントしています(**リスト9**)。同じ単語であれば、カウントアップされていきます。

●declareOutputFieldsメソッド

WordCountBoltでは、「new Values(word, count)」と2つのフィールドを指定しているため、"word"、"count"という2つのフィールド名を指定しています(**リスト10**)。

全体をつなげる －WordCountTopology－

最後に、これまで作成したSpoutやBoltを連携させるためのTopologyクラスを実装します(**リスト11**)。Topologyを実装する際に意識するのは、並列

■ **リスト8　WordCountBoltクラス**

```
public class WordCountBolt extends BaseBasicBolt {

  /** 単語出現回数カウンタ */
  Map<String, Integer> counts = new HashMap<String, Integer>();

  @Override
  public void execute(Tuple input, BasicOutputCollector collector) {
    (中略)
  }

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    (中略)
  }
}
```

■ **リスト9　executeメソッド**

```
@Override
public void execute(Tuple input, BasicOutputCollector collector) {
  // 単語出現回数カウンタからカウンタを取得
  String word = input.getStringByField("word");
  Integer count = this.counts.get(word);

  if (count == null) {
    count = 0;
  }
  count++;

  // 結果を単語出現回数カウンタに反映
  this.counts.put(word, count);

  collector.emit(new Values(word, count));
}
```

数とグルーピングとの指定です。

並列数はSpout／BoltをTopologyに登録する際に指定するだけです。パフォーマンスを向上させたければ、この数値を変更するだけで可能になります。グルーピングについて、SplitSentenceBoltはランダムにデータを受け取るshuffleGrouping、WordCountBoltは単語ごとにデータを受け取る必要があるためfieldsGroupingを指定しています。

◆ ◆ ◆

■リスト10　declareOutputFieldsメソッド

```
@Override
public void declareOutputFields(
                OutputFieldsDeclarer declarer) {
  declarer.declare(new Fields("word", "count"));
}
```

■リスト11　WordCountTopologyクラス

```
public class WordCountTopology {
  public static void main(String[] args)
      throws Exception {
  （中略）

  // Topology を作成する
  TopologyBuilder builder = new TopologyBuilder();

  // Add Spout(KestrelThriftSpout)
  KestrelThriftSpout kestrelSpout
    = new KestrelThriftSpout(kestrelHosts,
                             kestrelQueueName,
                             new StringScheme());
  builder.setSpout("KestrelSpout",
                   kestrelSpout, 2);

  // Add Bolt(KestrelThriftSpout -> SplitSentence)
  builder.setBolt("SplitSentence",
                  new SplitSentenceBolt(), 20)
    .shuffleGrouping("KestrelSpout");

  // Add Bolt(SplitSentence -> WordCount)
  builder.setBolt("WordCount",
                  new WordCountBolt(), 10)
    .fieldsGrouping("SplitSentence",
                    new Fields("word"));

  if (isLocal) {
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("WordCount",
        conf, builder.createTopology());
  } else {
    （中略）
  }
}
```

今回のTopologyの処理でサーバ（CPU：Xeon E3-1230 3.2GHz 4Core、メモリ：32GB）2台で分散するようにして性能を測定したところ、次のような結果になりました。

◎スループット（Tuple／秒）

- KestrelThriftSpout：3,302
- SplitSentenceBolt：3,302
- WordCountBolt：545,223

今回の実装で、並列分散処理を意識した内容はほとんどありません。そのような簡単な実装で、数千～数十万のイベントを処理するようなシステムを簡単に実現できるようになるのは、Stormの効果であると言えます。

今後の動向

Stormはリアルタイムの並列分散処理を実現するには非常に強力なフレームワークですが、一方、それ単体では実現できる処理はシンプルなものです。そのため、StormとほかのOSSを連携するSpoutやBoltが提供されつつあります。

- Spout：Kestrel、RabbitMQ、Kafka、JMS、Redis、Scribe
- Bolt：HBase、Cassandra、Mongo

Hadoopもそれ単体ではMapReduceというシンプルな処理しか提供していませんでしたが、エコシステムという周辺プロダクトが整備されたことにより、非常に利便性が向上しました。SpoutやBoltは疎結合であるため、同様に周辺プロダクトが整うことで、より効率的にシステムを開発できるようになっていくことが期待されます。

また、その他にも、CEP（複合イベン

ト処理）や機械学習といった、より高度な機能の実現が検討されています。このような機能が提供されれば、適用範囲もより広がるでしょう。海外では、すでにStormを利用した商用システムもいくつか公開されています。今後、日本でも、そのようなシステムが増えることが期待されます。SD

Stormの日本語情報は、まだまだ少ないですが、筆者が所属するAcroquest Technologyでは、若手エンジニアが、ブログでStormの情報を公開しています。

◎Taste of Tech Topics
http://acro-engineer.hatenablog.com/

本家のサイトでも公開されていないような、独自に調査や評価した内容も掲載しています。ぜひ、参考にしてみてください。

●参考情報

◎Storm本家サイト
http://storm-project.net/

◎Storm-Installer
https://github.com/acromusashi/storm-installer
Stormの環境を簡単に構築できるように、インストーラを公開しています。

◎本記事のサンプル
https://github.com/acromusashi/storm-example-wordcount
本書で利用したStormのサンプルプログラムです。

FBで生まれたビッグデータ分析ツール

HiveでHadoopを活用してみませんか!

上司より「ビッグデータを分析せよ!」と命じられていきなりHadoop環境を構築して、さて次に何をすべきか。また新たに勉強すべき知識が山ほどありそうです。そんなあなたに強い味方がいます。SQLライクにHadoopを扱うことができる「Hive(ハイブ)」です。本稿はその導入方法と使い方のエッセンスを紹介します。Hiveでビッグデータ分析を始めましょう!

株式会社リクルートテクノロジーズ
ITソリューション1部 ビッグデータグループ　**石川信行**　ISHIKAWA Nobuyuki

ハードルが高すぎるHadoop?

今でこそ機械学習や大量データ集計のツールとして、Coolなイメージがあるですが、筆者の所属するリクルートテクノロジーズ(以降、当社)が5年前に検証を始めたころ、その導入目的は既存のバッチ処理の高速化と大量データの保管という、たいへんシンプルなものでした。このゴールのためにリクルートグループの各事業会社へ導入提案を積極的に推進したのですが、そこで大きな障壁となったのが、MapReduceで演算処理を書くコストの高さでした。つまり「Hadoopを導入すれば大量データを蓄積し、高速に処理できるようになります。でも、処理はJavaで記述しなくてはならないのですけど」といった具合で提案をしても、Javaを書くハードルの高さとCoolとはかけ離れた「処理の高速化と大量データ蓄積」という地味な目的とのトレードオフから、ユーザの皆さんの反応はいまひとつだったことを覚えています。

ハードルを一気に下げるHive!

そのような状況を打開するために注目したのが、米Facebookが開発を行っていたHiveでした。RDBMSで用いられてきたSQLと非常に似たHiveQLと呼ばれる操作言語でHadoop(HDFS)上のデータを処理できるエコシステムです。

各事業会社では、RDBMSとJavaのフレームワークを用いたアプリケーション開発が主流のため、SQLの知見を持った方々が多くいます。そのため簡単なSQLを書くことができる企画担当者(ユーザ)が多かったのです。

Hiveを導入することで、利用初期に発生する単純な帳票出力のような「ちょっとやってみる作業」を事業のユーザの方々が自らHiveQLを書いて実装しはじめました。そんな小さな一歩から、Hadoopの活用が始まっていったのです。誤解を恐れずにいえば、ユーザから見ればHadoopが大量のデータを高速に処理できるDBぐらいの距離感になったといえます。もちろんユーザの方々にも「新しいものにも積極的

に取り組んでみよう」という心意気があったのはいうまでもありません。

また、先述のようにRDBMS + SQLで時間がかかっていたバッチ処理を高速化するという目的を達成するために、このHiveの導入が、プログラミング工数の削減とHadoop上にデータがあれば既存システムに影響を与えず検証ができるという観点で、Hadoop導入のハードルを下げてきたとも言えます。

Hadoop Conference Japan 2013 Winterのエントリ時のアンケートによればHadoopエコシステムの中でHiveがもっとも使われていることが示されており（Hadoop利用者624名のうち245人）、同様な操作言語レイヤにあるPigとくらべて3倍近く差もあることがわかっています。このことからも当社だけではなくHiveが日本でのHadoop利用のハードルをいかに下げてきたかがわかるかと思います。

Hive環境の構築は簡単!

HadoopおよびHiveのインストールはとても簡単で、インストール手順の情報もWeb上に溢れています。またHadoopのディストリビューションを提供しているCloudera社のページ[注1]やMapR社のページ[注2]でHadoopやHiveが包括されたディストリビューションがダウンロードできます。これらを利用すれば簡単に開発環境を構築できます。ここではインストールの詳細な手順は省かせていただき、Hiveの特徴にフォーカスを当てて説明していきたいと思います。

Hiveの構造は**図1**のように概略できます。簡単にいえばクライアントで作成したHiveQLをドライバ経由でパース（parse）し、MapReduce処理に変え、メタストアにある定義で構造化されたHDFSのデータを操作するというロジックになっています。この構造をふまえてSQLとHiveの違いを理解するために、まず簡単にHiveとSQLの相違点を**表1**にまとめてみます。

▼図1　Hiveの内部構造概略

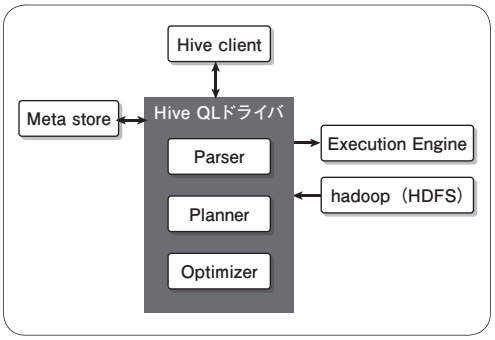


表1を参照すると、UPDATEやDELETEが使えない、データ型が限定されるなどがわかります。ただ、これをそのまま覚えるというよりは、次に示すような本質的な特徴を理解したほうが良いでしょう。

筆者の経験からHiveの特徴として少なくとも押さえておきたいのは、次の2点です。

・裏でMapReduceに変換されること
・操作しているのはHDFS上のデータであること

この特徴をふまえてSQLとの違いを理解しなければ、HiveはRDBMSをリプレイスできるなどの都市伝説を広めてしまうことになります。

Hiveをいかに導入すべきか

まず1つめですが、これはどんな小さな処理でも必ずMapReduceに変換する準備時間がかかります。つまり10件のテーブルから1件のレコードを選択するだけでも、その準備で10秒程度を要します。よってこのオーバーヘッドが無視できるほど膨大な量のデータを扱う処理がHiveには適しています。

当社の各事業部では、過去さまざまな指標を集計・モニタリングしてきましたが、扱うデータ量が膨大になると、RDBMSや既存のGUIベースの集計ツールでは、性能やリソースの問

注1）https://ccp.cloudera.com/display/SUPPORT/Downloads
注2）http://www.mapr.com/products/download?utm_expid=44001406-0&utm_referrer=http%3A%2F%2Fwww.mapr.com%2F

▼表1　HiveとSQLの比較（※ hiveはversion 0.7.1を想定）

機能	SQL（Oracleを想定）	HiveQL
INSERT	可能	基本的にはOverWriteで上書き。パーティションを駆使すれば部分insertも可能（0.8でinsert intoをサポート）
UPDATE	可能	不可
DELETE	可能	不可
SELECT	可能	可能
テーブル結合	可能	可能（内部結合、外部結合、部分結合、マップ結合）
トランザクション	可能	不可
オンライン処理	可能	不可（MapReduce処理オーバーヘッドあり）
CREATE/DROP TABLE文	可能	可能（EXTERNAL表も利用可能）
インデックス作成	可能	可能（キーだけの中間テーブルを作るイメージ。キーへのアクセスはインデックス領域の全件検索） ※Hive0.8からはBitmap Indexesが対応している
ビュー	ビュー、マテリアライズドビューで更新可能	読み出しのみで、マテリアライズドビューはサポートされない
サブクエリ	任意の節中に書け、相関サブクエリであってもなくてもよい	FROM節中にしか書けなく、相関サブクエリはサポートされない 別名必須
複数テーブルへのインサート	非サポート	サポート
レイテンシ	1秒以下	分単位
拡張可能部分	ユーザ定義関数・ストアドプロシージャ	ユーザ定義関数・MapReduceスクリプト
NULL	NULLと空文字は区別される	NULLと空文字は区別される（NULLは¥Nで表現）
ROWNUM・OFFSET	利用可能	ROWNUM・OFFSETともに存在しない。LIMIT句で表示データ数を制限可能
パーティション	可能	可能（SELECT時にパーティションを指定可能）
クライアント	SQL*Plus(Sqlplusコマンド)	HiveCLI（Hiveコマンド）
データ型	VARCHAR,CHAR,LONG,CLOB,NUMBER,BINARY_FLOAT,BLOB,DATE,TIMESTAMPなど、さまざま	TINYINT,SMALLINT,INT,BIGINT,BOOLEAN,FLOAT,DOUBLE,STRING,TIMESTAMP(Hive0.8),BINARY(Hive0.8)

題で処理できないという問題が起きていました。

当然のことながら処理の高速化という観点でHadoopへ移行するという案件も複数ありました。その例の1つに、PV（Page View）や閲覧数などの指標をカウントするSELECT文を1行ずつシーケンシャルに何本も回している処理をリプレイスするものがありました。これに関しては、そのままHiveへ移行すると、繰り返しシーケンシャルに流す分だけMapReduceのオーバーヘッド分の秒数が加算され、結果的に既存の処理より遅くなってしまったのです。このような例もふまえ、Hiveの処理で得たい結果とその形式を押さえたうえで、HiveのMapReduce変換のオーバーヘッドをなるべく出さないようにするように、単一のHiveQLで大きなデータを処理する方法を考えることが重要だとわかりました。つまり、クエリプラン作りの徹底です（場合によっては帳票フォーマットの変更も必要かもしれません）。

前述のようにHive積極導入を推進している筆者も、この性質をユーザにうまく伝えられなかったことがあります。SQLライク処理ができるという認識のもとでHiveを使用し、実際に単一レコード取得などをすると、「遅い」という印象を与えてしまったのです。

既存のSQLをHiveにリプレイスする際には単なるリプレイスではなく、新しい方針を打ち出すことも大切です。たとえば、従来のSQLでは実行できそうもない、重たくて複雑な処理、単純に対象の集計期間を増やすことから始まり、

エリアや性別などで区切ってクロス集計していたものに、職種などの新たな条件軸を複数加えて細分化するなどの新しい集計方針の提案です。

裏側のしくみを見極める

そして2つめは「裏で行っているのはファイル操作にすぎない」ことです。HiveではCREATE文を使ってテーブルを作成しますが、これはただ単純にファイルのデータを特定のデリミタで区切り、指定したテーブル定義に合わせて構造化しているだけにすぎません。よって、「カンマ」をデリミタに指定した際に不自然な位置にカンマが混入してしまえば予期せぬ区切られ方をすることもあります。しかも、データのロード時にカラムの数が合わなくてもエラーが発生しないため注意が必要です(**図2**)。またHiveがUPDATE文に対応していないこと、基本的にINSERTはINSERT　OVERWRITEであり、全件更新であるという特徴も裏がファイル操作といえば理解しやすいと思います。

Hiveサンプル実行

これらの特徴をふまえたうえで、実際にテーブル作成からサンプル実行までの過程を説明したいと思います。今回使用した環境は次のとおりです。

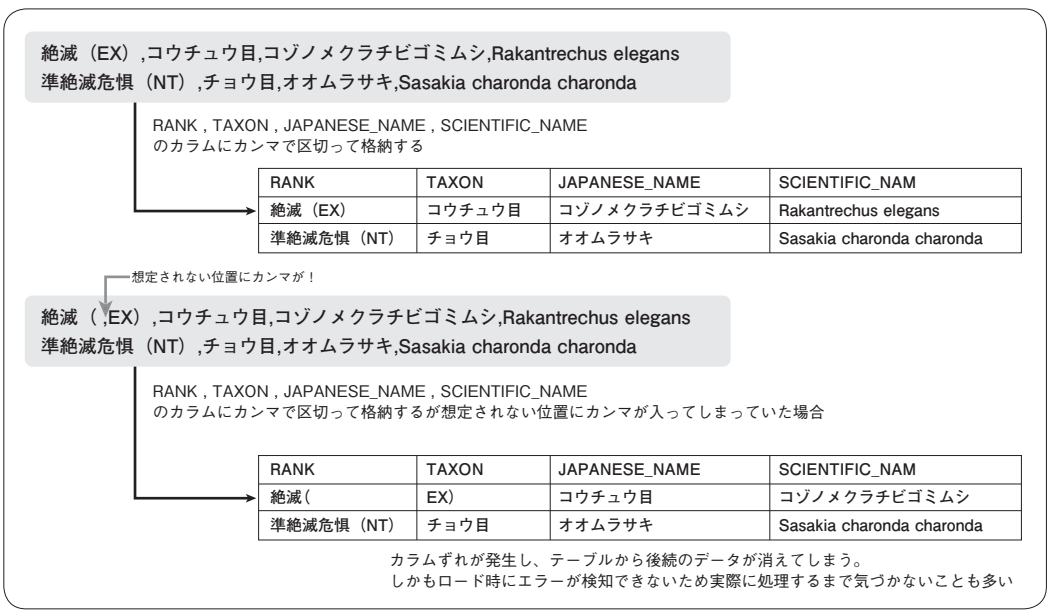
- OS：Red Hat Enterprise Linux Server Release 6.2
- Hadoop：0.20.2(MapR1.2.3)
- Hive：0.7.1

①Hadoopの起動

まずは、「$HIVE_HOME/bin/hive」をコマンドライン上で実行し、Hiveを起動させます。手始めにRDBMSと同様な感覚でスキーマを作成します。

```
hive> CREATE DATABASE IF NOT EXISTS ↵
RED_DATA;
OK
Time taken: 0.32 seconds
```

▼図2　Hiveのしくみと特徴



②テーブルの作成

次に、作成したスキーマ内にテーブルを作成します。今回作成するテーブルには、次のURL[注3]からダウンロードできる次のようなCSV形式のデータを格納します。

```
絶滅(EX)  コウチュウ目  カドタメクラチビゴミムシ  Ishikawatrechus intermedius
絶滅(EX)  コウチュウ目  コゾノメクラチビゴミムシ  Rakantrechus elegans
絶滅(EX)  コウチュウ目  キイロネクイハムシ  Macroplea japana
絶滅危惧(特)類(CR+EN)  トンボ目  オオセスジイトトンボ  Cercion plagiosum
絶滅危惧(特)類(CR+EN)  トンボ目  ヒヌマイトトンボ  Mortonagrion hirosei
絶滅危惧(特)類(CR+EN)  トンボ目  オオモノサシトンボ  Copera tokyoensis
絶滅危惧(特)類(CR+EN)  トンボ目  オガサワラアオイトトンボ  Indolestes boninensis
```

このデータに見合うカラム名と型を指定し、テーブルを作成します。カラムはカンマで区切り(FIELDS TERMINATED)、改行で1レコードとみなします(LINES TERMINATED)。

```
hive> CREATE TABLE IF NOT EXISTS RED_DATA.NAME_LIST
    > (RANK STRING, TAXON STRING, JAPANESE_NAME STRING, SCIENTIFIC_NAME STRING)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > LINES TERMINATED BY '¥n';
OK
Time taken: 0.242 seconds
```

ここでHadoopのコマンドでHDFSの中身を参照してみると作成したスキーマとテーブルはHDFS上ではディレクトリのように表示され、裏がファイルシステムだということがわかると思います。

```
[hadoopuser@***** ~]$ hadoop fs -ls /user/hive/warehouse/red_data.db
Found 1 items
drwxrwxrwx - hadoopuser hadoop 0 2013-04-08 19:46 /user/hive/warehouse/red_data.db/name_list
```

注3) http://www.biodic.go.jp/rdb/rdb_f.html (ちなみに、なぜこのデータなのかという問いには、筆者の大学時代の専攻は昆虫機能学ですと答えておくことにします)

③データの投入とロード

上記で作成したテーブルにデータを投入します。今回はHDFS上にあるファイルではなく、ローカル上(Linux)にWebからダウンロードしたテキストファイルを配置し、これをロードしています。

```
hive> LOAD DATA LOCAL INPATH '/tmp/reddata_sample.txt'
    > OVERWRITE INTO TABLE name_list;
OK
Time taken: 0.286 seconds
```

④データの確認

実際にHDFSにどのようにデータが格納されているかを確認すると次のようにテーブル名と同じ名前のディレクトリの下にファイルが配置されていることがわかります。

```
[hadoopuser@***** ~]$ hadoop fs -ls /user/hive/warehouse/red_data.db
Found 1 items
drwxrwxrwx   - hadoopuser hadoop 0 2013-04-08 19:46 /user/hive/warehouse/red_data.db/name_list
```

ちなみに、このHDFSのディレクトリ配下にputコマンドなどで直接CSVファイルを配置するだけでもHive上でデータを取り扱うことが可能となります。

⑤テーブル操作

ここから実際に作成したテーブルに対していくつかの操作を行ってみたいと思います。

まずは手始めにLIKE句を使って絞り込んだあとにカウントをとってみます。簡単なSELECT文でもMapReduceが起動し、オーバーヘッドがかかることがわかります。

```
hive> SELECT COUNT(1) FROM RED_DATA.NAME_LIST WHERE TAXON LIKE 'チョウ%';
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
…… (中略) ……
2013-04-10 21:41:00,793 Stage-1 map = 100%,  reduce = 100%Ended Job =
```

```
Ended Job = job_201209031838_74061
OK
102
Time taken: 14.52 seconds
```

SELECT句でカラムを指定せず＊を使うと、テーブル定義に合わせた解釈を行うためMapReduceが走らず、高速に結果が返ってきます。

```
hive> SELECT * FROM RED_DATA.NAME_LIST ➥
LIMIT 5;
OK
絶滅(EX)　コウチュウ目　カドタメクラチビゴミ➥
ムシ　Ishikawatrechus intermedius
絶滅(EX)　コウチュウ目　コゾノメクラチビゴミ➥
ムシ　Rakantrechus elegans
絶滅(EX)　コウチュウ目　キイロネクイハムシ　➥
Macroplea japana
絶滅危惧㊎類(CR+EN)　トンボ目　オオセスジイ➥
トトンボ　Cercion plagiosum
絶滅危惧㊎類(CR+EN)　トンボ目　ヒヌマイトト➥
ンボ　Mortonagrion hirosei
Time taken: 0.097 seconds
```

⑥HiveQLのシーケンシャル実行

先ほどHiveの特徴で述べたとおり、複数の簡単なカウント処理をシーケンシャルに流してみます。Hiveでは-fオプションを付けることで、ローカル上に置いたテキストファイルからHiveQLを読み込み実行できます。まずは、次のようなHiveQLを入力し、sample.hqlという名で保存します。

```
SELECT COUNT(1) FROM RED_DATA.NAME_LIST
WHERE TAXON LIKE 'チョウ%';
SELECT COUNT(1) FROM RED_DATA.NAME_LIST
WHERE TAXON LIKE 'カゲロウ%';
SELECT COUNT(1) FROM RED_DATA.NAME_LIST
WHERE TAXON LIKE 'カメムシ%';
SELECT COUNT(1) FROM RED_DATA.NAME_LIST
WHERE TAXON LIKE 'ハチ%';
```

これを実行してみます。timeコマンドを付けて時間も計測しておきます。

```
[hadoopuser@******* tmp]$ time hive -f ➥
/tmp/sample.hql
Hive history file=/opt/mapr/hive/hive-➥
0.7.1/logs/hive_job_log__201304102213➥
```

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at ➥
compile time: 1
……(中略)……
Kill Command = /opt/mapr/hadoop/➥
hadoop-0.20.2/bin/../bin/hadoop job  ➥
-Dmapred.job.tracker=maprfs:/// ➥
-kill job_201209031838_74086
2013-04-10 22:13:48,353 Stage-1 map = ➥
0%,  reduce = 0%
2013-04-10 22:13:52,371 Stage-1 map = ➥
100%,  reduce = 0%
2013-04-10 22:13:58,397 Stage-1 map = ➥
100%,  reduce = 100%
Ended Job = job_201209031838_74086
OK
55
Time taken: 11.43 seconds

real    0m50.470s
user    0m10.177s
sys     0m0.437s
```

このように1文ごとにMapReduceが起動し、オーバーヘッドの時間が積み重なっていることがわかります。これを次のようなHiveQLに書き換えて一度で集計できるように変えてみましょう。

```
SELECT
  COUNT(IF(TAXON LIKE 'チョウ%', 1, ➥
NULL)),
  COUNT(IF(TAXON LIKE 'カゲロウ%', 1, ➥
NULL)),
  COUNT(IF(TAXON LIKE 'カメムシ%', 1, ➥
NULL)),
  COUNT(IF(TAXON LIKE 'ハチ%', 1, NULL)),➥
FROM RED_DATA.NAME_LIST;
```

同様にconvert_sample.hqlという名で保存し実行してみます。

```
[hadoopuser@*******  tmp]$ time hive ➥
-f convert_sample.hql
Hive history file=/opt/mapr/hive/hive-➥
0.7.1/logs/hive_job_
log_201304102241_642693019.txt
Total MapReduce jobs = 1
Kill Command = /opt/mapr/hadoop/➥
hadoop-0.20.2/bin/../bin/hadoop job  ➥
-Dmapred.job.tracker=maprfs:/// -kill ➥
job_201209031838_74088
2013-04-10 22:41:51,057 Stage-1 map = ➥
0%,  reduce = 0%
2013-04-10 22:41:56,088 Stage-1 map = ➥
100%,  reduce = 0%

……(中略、次ページへ続く)……
```

```
2013-04-10 22:42:03,129 Stage-1 map = ➥
100%,  reduce = 100%
Ended Job = job_201209031838_74088
OK
102     4       88      55
Time taken: 16.537 seconds

real    0m18.139s
user    0m8.473s
sys     0m0.298s
```

このようにMapReduceが起動のオーバーヘッドが少なくなる分、処理時間が短縮されることがわかります。ただし、カウントの結果を横並びに出力することになるので最終帳票のフォーマットなどは最初から考慮しておく必要があります。

また、はじめてHiveを扱う際に地味に役立つのがexplain句です。これを使用することにより自分が描いたクエリが成功するかどうかをチェックすることができます。

●成功例

```
hive> explain SELECT
    >   COUNT(IF(TAXON LIKE 'チョウ%', ➥
1, NULL)),
    >   COUNT(IF(TAXON LIKE 'カゲロウ%', ➥
1, NULL)),
    >   COUNT(IF(TAXON LIKE 'カメムシ%', ➥
1, NULL)),
    >   COUNT(IF(TAXON LIKE 'ハチ%', 1, ➥
NULL))
    > FROM RED_DATA.NAME_LIST;
OK
……(中略)……
    Fetch Operator
      limit: -1
Time taken: 2.661 seconds
```

●失敗例

```
hive> explain SELECT
    >   COUNT(IF(TAXON LIKE 'チョウ%', ➥
1, NULL)),
    >   COUNT(IF(TAXON LIKE 'カゲロウ%', ➥
1, NULL)),
    >   COUNT(IF(TAXON LIKE 'カメムシ%', ➥
1, NULL)),
    >   COUNT(IF(TAXON LIKE 'ハチ%', 1, ➥
NULL)),
    > FROM RED_DATA.NAME_LIST;
FAILED: Parse Error: line 6:0 cannot ➥
recognize input near 'FROM' 'RED_DATA' ➥
'.' in select expression
```

⑦サブクエリの実行例

もちろんSQLと同様にサブクエリやJOINも使用できます。サブクエリを実装した例を示します。

```
hive> SELECT  count(1)
    > FROM(
    >   SELECT  TAXON
    >   FROM  RED_DATA.NAME_LIST
    >   GROUP BY  TAXON
    > ) A;
……(中略)……
Time taken: 23.973 seconds
```

JOINやサブクエリを使うと処理によってはMapReduceが複数段走ることが特徴です。何度も繰り返しますがその分だけオーバーヘッドがかかることを忘れてはなりません。

GUIツールの活用

前述のようにSQLとの違いは、よく理解しなければならないものの、MapReduceを直接書くよりも導入コストが下がることは一目瞭然です。このメリットはほかのデメリットを相殺するほどです。MapReduceに変換されるゆえにUDFを使って自在に独自関数を作成可能なことも見逃せません。

当社ではこのHiveをさらに活用するために、WebのGUIからHiveQLを入力し、HDFS上のデータを操作・ダウンロードできるツールであるWebHiveをオープンソースとして公開しています[注4]。

コマンドラインのインターフェースに苦手意識がある人に対してGUIツールを通して気軽にHiveQLを書く機会を提供することで、Hadoopを用いたデータの利活用が促進され、施策接続へとつながっていっています。こうしたBI(Business Intelligence)ツールなどをはじめとするGUIのツールと相性が良いのもHiveの特徴の1つと言えるでしょう。

注4) https://github.com/recruitcojp/WebHive

Hadoopのエコシステムと長所

RDBMSの処理を代替することでHadoopとHiveの有用さを確かめるという観点で、Hiveと相性がよく重要なツールとして挙げておきたいのが、Sqoopというエコシステムです。SqoopはおもにRDBMSからHDFSへデータのインポート、HDFSからRDBMSへエクスポートを行うために利用されます(**図4**)。このSqoop最大のメリットは、Hadoopクラスタ側からJDBCでRDBMSへ接続を行うコマンドを発行できることにあります。これにより、いちいちRDBMSからHadoopへのファイル出力するためのバッチを書くなど、既存のアプリケーション開発者に余計な手間をかける必要がなくなります。つまりHadoopによるデータ集計のもっとも重要で手間のかかる部分であるデータをHDFSへ格納するというフェーズを簡略化できます。

このSqoopにはHiveへデータ格納を行うオプションが存在し、これを併用すればRDBMSからデータをHiveに直接格納し、HiveQLで操作するといったシームレスな連携も可能となります。また、既存のRDBMSのシステムに大きな負荷をかけず、RDBMSと同等以上のデータ量が入った大規模データ処理用システムをクローンとして作れることで、導入難易度が著しく下がりHadoopとHiveの活躍の場が増えたことも、すでに当社内で実績として残っています。このようなエコシステムが豊富にそろっており、連携な可能な点もHadoopの長所だと言えます。

Hadoopを Hiveで活用しよう!

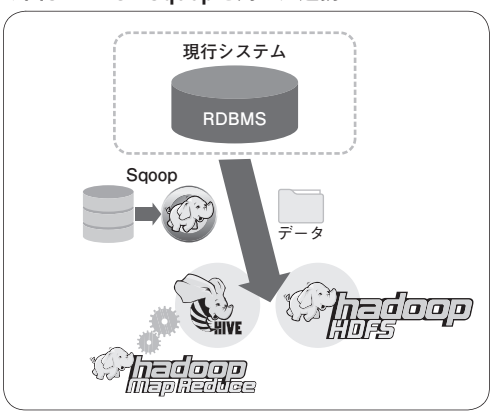
HadoopのCoolな利用シーンといえば冒頭にも述べたとおり、計算量の膨大な機械学習であったり、テキストや画像といった非構造データ解析を想像しがちです。これはまったく間違いではなく、むしろ正しい利用シーンです。その証拠して、当社ではデータの一元管理から始まり、大量データ集計、パーソナライズ、機械学習、自然語処理などのさまざまな処理をHadoopで行っており、その結果を基にレコメンドや帳票出力、内部システムのロジック改善などの施策に結び付けその利活用を進めています。ですが、その裏にはいわゆる集計・データ整形と呼ばれるような地味な作業がいくつも積み重なっており、ここで扱うデータも紛れもなく大量であるケースがほとんどです。

こういった作業に対して、手軽にHDFSのデータ操作ができるHiveが活躍することは間違いなく、また、Hadoop全体の利用を考えるうえで、ユーザの方に実際にHadoopに触れてもらいイメージをつかんでもらうための、事始めとして非常に有用なツールとなります。

これまで述べてきたようなHiveの特徴を把握し、利用シーンが想像できれば、まさにこれからHadoop導入するみなさんにHiveは必須のツールの1つとなると言えるでしょう。

まずは、身近にあるRDBMSで大量のデータを集計しているSELECT文を見つけ、その1つをHiveQLに書き換えることから、Hadoop利用のCoolな第一歩を踏み出しましょう！SD

▼図3　HiveとSqoopを用いた連携



Ubuntu 13.04 “Raring Ringtail”

～新世代のUbuntuへの最初のマイルストーン～

Ubuntu Japanese Team
㈱創夢　吉田 史　YOSHIDA Fumihito　hito@ubuntu.com

本稿では、2013年4月25日にリリースされたUbuntu 13.04 “Raring Ringtail”の新機能と、今後のUbuntuの方向性について説明します。

Ubuntu 13.04 “Raring Ringtail”

Ubuntu 13.04[注1]は、Ubuntuにとって18回目のリリースにあたります。リリースマスコットは「Ringtail」[注2]で、アライグマに似た、すばしこく体が引き締まった小動物です。

13.04は、12.04 LTSと14.04 LTSの間に行われる3回の「standard」リリースのうち、2度目のリリースとなります[注3]。LTSリリースからちょうど1年、大きな機能追加が完了するタイミングです。結果、このタイミングのリリースでは、直前のリリースで大幅な変化が起きていなければ、非常に大きな変化が生じる傾向があります。たとえば9.04では直前の8.10が「ブラッシュアップ」的なリリースだったこともあり、「起動時間の大幅な短縮」という大きな変化が生じています。13.04も、12.10が比較的「おとなしい」ものだったため、大きな変化があることを期待されるリリースです。

ところが、デスクトップやサーバ向けOSとしてのUbuntuにとっては、13.04は強烈な変化を伴うリリースではありません。リリース時点の変更は「おとなしい」もので、Unityのデザイン修正やUIの修正、新しいソフトウェアの搭載、といった程度で、「LTSとLTSの間に起きる大きな変化」と言うには限定的です[注4]。

これは、デスクトップやサーバに大きな変化を起こすのではなく、タッチベースのデバイスに向けたUbuntuである「Ubuntu Touch」、すなわちUbuntu PhoneやUbuntu Tabletにリソースが向けられた結果です。Ubuntuがデスクトップやサーバ、組み込み（Ubuntu Core）などだけでなく、スマートフォンやタブレットに広がる未来がやってこようとしています。Ubuntu Touchのリリース予定時期は2013年後半で、13.04のリリース時点では「プレビュー」的な立ち位置となります。

13.04の新機能

前述のとおり13.04はやや「おとなしい」ものではありますが、変更点が少ないわけでもありません。13.04で変化する点を見て行きましょう。

注1） Ubuntuでは「リリース月.日」という形式でバージョン番号を付与することになっています。2013年4月リリースですので「13.04」となります。

注2） 和名カコミスル。Ubuntuではリリースごとにシンボルとなる動物を選び、さらに頭韻を踏む形容詞句をリポジトリ名や相性として利用します。13.04では「raring」です。

注3） Ubuntuでは、「6ヵ月に一度リリースを行う」「2年に一度は通常版（standard）ではなく、サポート期間の長い特別版（LTS：Long Term Support）としてリリースする」というリリース方針を取っています。「standardを3回リリースしたら、次はLTS」という周期です。

注4） メモリ容量の削減などは行われています。

「Nexus 7用リリース」の提供

13.04における目玉の1つは、「Nexus 7用のUbuntu」が提供されることです。これをAndroidのカスタムROMと同じ方法でインストールすることで、Nexus 7上でUbuntuを利用できます。導入されるのは「Ubuntu Core」で、最小限の環境だけが導入され、あとは利用者にまかせる形です。OpenGL ESベースでUnityも動作し、デスクトップ版と同じUIが利用できます。

Ubuntu上で動作する専用のインストールツール（ラッパー）「ubuntu-nexus7-installer」（**図1**）が準備されるため、デスクトップ／ノートPCなどにNexus 7を接続し、ツール（**図2**）を用いてインストールすることもできます[注5]。

注意すべきは、Nexus 7用リリースは実際には「実用的なものではない」、ということです。このリリースは、ARM SoCベースの環境、すなわちx86に比べるとCPU性能やメモリ容量、ストレージ性能などが限定的で、かつバッテリ駆動時間がシビアに要求される環境で、「使いものになる」状態を作ることができるか、というチャレンジに用いられる、開発者向けのリリースとなっています。とはいえ、7インチ、300g強のデバイスでUbuntuが「そこそこ」動作するため、モバイルなどに用いることもできます。

インターフェースの改良

Ubuntu 13.04のデスクトップ環境の変更は、おもにユーザインターフェースの更新となっています。変更点を見て行きましょう。

■Unityの更新

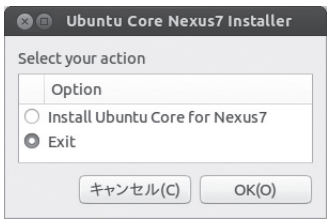
Unityそのものはそれほど大きく変化していませんが、UnityのDash呼び出しアイコン（通称「大きなヘンなボタン」；Big Funny Button。画面左のLauncherの一番上に登録されたアイコン）のリデザインが行われています（**図3**）。

Unityそのものへの変更は、「Scope」と呼ばれる検索カスタマイズや、検索結果からU1MS（Ubuntu One Music Store）を利用して「その場で音楽を購入できる」といった新機能が品質上の問題から見送られたため、それほど多くありません。

なおRaringでは、これまでに引き続きCompizベースで実装されたUnityが利用されます。12.04まで提供されていたQtベースのフォールバック実装（Unity 2D）は存在しないため、GPUドライバがOpenGLないしOpenGL ESをサポートしていない場合はLLVMpipeによるエミュレーション動作を用いることになります。

▼図1　Ubuntu Core Nexus7 Installer

▼図2　インストール画面



▼図3　リデザインされた大きなヘンなボタン

注5）興味がある場合は、https://wiki.ubuntu.com/Nexus7/Installationを参照してください。

■シャットダウンダイアログとアップデートマネージャ

Unity組み込みの機能の中では、シャットダウンダイアログが新しいものに更新されています(**図4**)。UnityのDash(Windowsキーで展開されるアイコンベースのランチャー画面)とデザインがそろえられ、これまでのGTKベースの確認ダイアログよりもわかりやすいものになりました。また、このダイアログはステートに応じて変化し、たとえばカーネルの更新など、再起動が必要なタイプの処理が行われたあとでは「再起動」が優先される、あるいはまだファイルを開いているアプリケーションが存在する場合はそれに応じた確認を行うようになっています。

同じように、アップデートマネージャのデザインも変更され、現代的なデスクトップ環境に合わせたものになっています(**図5**)。

▼図4　シャットダウンダイアログ

▼図5　アップデートマネージャ

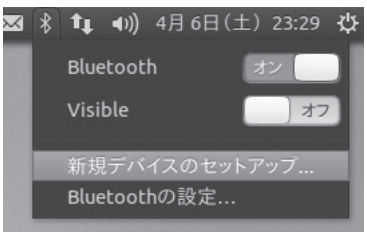
■新しいIndicator

画面上部／右端に登録される各種アイコン(Indicator)にもいくつかの変更が加えられています。

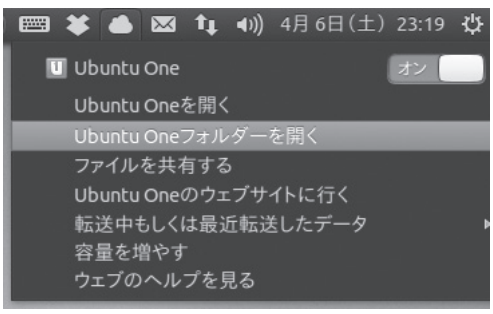
まず、Bluetooth Indicator(indicator-bluetooth)がブラッシュアップされ、非常にわかりやすいものになりました(**図6**)。Bluetoothデバイスの操作の中でもっとも頻繁に行われるであろう、電波のOn/Offと、「Visible」(サーチ可能)の切り替えをワンタッチで行えるようになっています。

また、Ubuntu One(Dropboxに似たオンラインストレージサービス)の動作を制御する、「Sync」Indicator(indicator-sync)が追加され、Ubuntu Oneの同期設定をワンタッチで制御できるようになりました(**図7**)。これらはとくに、タッチベースのインターフェースを採用したデバイス(タブレットやスマートフォン、そしてWindows 8対応のPC)でも利用できるようにするためと考えられます。

▼図6　Bluetooth Indicator



▼図7　Ubuntu Oneの同期設定



コア部分の変更

Ubuntuにとって、「目に見えない部分」の更新も重要です。13.04で更新された、「目に見えない場所」の更新点を見て行きましょう。

■カーネル

Ubuntu 13.04 では、Linux Kernel 3.8.5 をベースにしたカーネルが利用されます。Ubuntuでは、いわゆる“Mainline”、kernel.orgで開発されているツリーに、次の機能を追加[注6]してビルドしたもの[注7]が用いられます。

- Ubuntu独自のドライバ(aufs、dm-raid45、overlayfsサポートと、EFI環境向けのefivarfsなど)
- Ubuntu独自の拡張機能、パッチ(各種入力デバイス用ドライバや、OEMで採用されているノートPC向けのパッチ、AppArmorの調整など)
- 一部のARM SoC(OMAP3、OMAP4、Evnergy Core)向けBSP(Board Support Package)由来のドライバの追加

特筆すべきは、これまでSoCごとに管理されていた(場合によってはSoCごとにベースバージョンすら違っていた)カーネルソースツリーが統合されたことです。統合されたツリーからx86、x64、OMAP3、OMAP4、Calxeda EnergyCore EMX-1000向け(i386/generic、amd64/generic、armel/omap3、armhf/omap3、armhf/omap4、armhf/highbank)がビルドされることになります。ただし、前述の「Nexus 7向けリリース」ではTegra 3への対応の都合から[注8]、3.1.10ベースの別ツリーが使われます。

■Upstart 1.8

Ubuntuが採用するinitデーモンであるUpstartは、13.04では1.8が採用されます。

Upstart 1.8では、ファイルの変化をトリガーとしてプロセスを起動できる、「upstart-file-bridge」と、テクノロジプレビューとしてUser session機能が搭載されています。また、Upstartイベントを追跡するための、upstart-monitiorツールが追加されました。

■upstart-file-bridge

upstart-file-bridgeは、「ファイルが変化した」ときにemit(Upstartにおけるイベント発呼)が行われる機能です。たとえば、『「/var/run/http-alert」というファイルが変化するたびにアラートを発呼するプログラムを起動する』という設定ができます。この設定を行うには、/etc/init/以下に、**リスト1**のような設定を記載した定義ファイルを設置します。Upstartはこの設定ファイルをもとに自動的にファイルシステムの監視を行い、/var/run/http-alertが作成／変化／削除されたタイミングで自動的に/usr/local/sbin/alert-by-mailを実行するようになります。監視する対象はファイルだけでなく、ディレクトリやglobパターン(アスタリスクなどを使った指定)にも対応します。

▼リスト1　定義ファイル

```
example : alert when trigger file created
description "http-alert catcher"

start on file FILE=/var/run/http-alert
stop on runlevel [06]

expect fork
exec /usr/local/sbin/alert-by-mail
```

注6) https://wiki.ubuntu.com/KernelTeam/Specs/RaringKernelDeltaReview 参照。

注7) 13.04で変更されたオプションを把握したい場合、https://wiki.ubuntu.com/KernelTeam/Specs/RKernelConfigReviewを参照してください。極端に大きな変化はありませんが、NFS_V4_1(NFS_V4ではなく、長らくexperimental扱いだったV4_1)が有効になっており、pNFSなどのNFS 4.1ベースの機能が利用できるようになっています。

注8) これはTegra 3用のBSP(Board Support Package。SoCベンダが各OS用に提供する「オリジナルのカーネルからの差分」をひとまとめにしたもの。Linuxの場合はとくにLSP/Linux Support Packageと呼ばわれることもあります)のサポートがLinux Kernel 3.1系をターゲットにしたもので、3.8などの新しいカーネル用に移植が完了していないことによります。

また、「created」「modify」「delete」トリガーを指定することで、検知する変化を指定することもできます。たとえば、**図8**のように「EVENT=create」を指定することで、ファイルが新規作成された場合だけを検知対象にします。

■upstart-monitor

upstart-monitiorは、Upstartの各種イベントが発生したこと、そしてそこから各種プロセスが正常に起動されたかどうかを追跡するためのプログラムです(**図9**)。この機能と、以前から搭載されているinitctl2dot(Upstartのイベント間の依存関係をdot形式で出力する)プログラムを組み合わせることで、イベントのデバッグが可能になります。複雑なサービス定義ファイルを記述する場合、この機能を用いて動作を確認することが必須になるでしょう。

デフォルトではインストールされていないので、明示的に「upstart-monitor」パッケージのインストールが必要です。GUIモード／コンソールモードを切り替えて利用することができます。デフォルトではuser sessionへ接続しようとするので、「--destination=」を用いて接続先を適切に指定する必要があります。より詳しい使い方は、man upstart-monitorを参照してください。

■user session

Upstartのuser session機能は、これまでの「user job」、「ユーザ個別のサービス管理デーモン」としての機能を応用し、デスクトップセッションで必要なサービスの管理を行うための機能です。

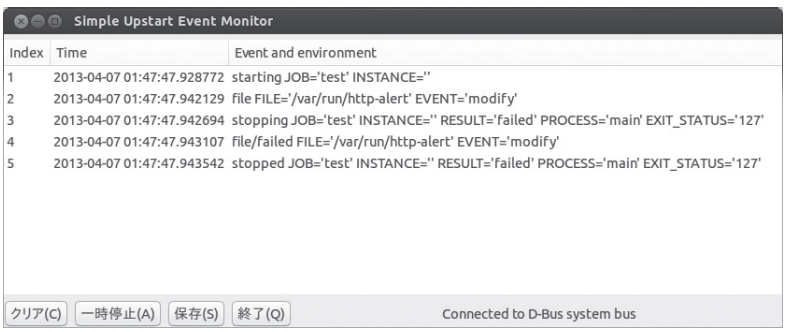
user jobは、通常のシステムジョブ管理と同じく、システム上特定のイベント(たとえば「ファイルシステムがマウントされた」「ネットワークが有効になった」「Xが起動された」)をトリガーにして、ユーザ用のプロセスを起動／終了できます。たとえば、「なんらかのUSB接続ストレージがシステムに接続されたタイミングで中身を確認する」といったことに用います。

user sessionはこの機能をさらに拡張し、デスクトップセッションで必要となるデーモンすべてをUpstart経由で管理する機能です。13.04ではあくまでプレビューとして搭載されており、有効にできるのは13.04のみ、かつ、有効化には/etc/upstart-sessionを編集し、user sessionを有効にするデスクトップ環境を指定する必要があります。この変更を行ってからログインすると、デスクトップで必要なデーモンが/usr/share/upstart/sessions/以下にあるupstartのnative job設定ファイルに基づいて起動されるようになります。

▼図8　「EVENT=create」を指定

```
start on file FILE=/var/run/http-alert EVENT=created
```

▼図9　Simple Upstart Event Monitor



現時点ではこの機能はあまり役に立ちませんが、将来的にデスクトップ環境との統合が進むと、「特定のイベントが発生した場合、必要なプロセスをすべて再起動する」といった機能が搭載できるようになります。また、「このプロセスのあとにこのデーモンを起動したい」という挙動が実現できます。たとえば、Unityが完全に起動しきってからインプットメソッドを起動したい、といった指定も可能になるため、タイミング的にデーモンの起動が失敗する問題を根絶できるようになります。

その他の変更点

■OpenStack

Ubuntuは、OpenStackの主要な動作プラットフォームでもあります。UbuntuではOpenStack側のリリースだけでなく、Ubuntu Serverで「apt-get install」するだけで簡単に扱えるよう、Ubuntu側で別途パッケージングが行われています。13.04世代のOpenStackはOpenStack “Grizlly”（2013.1）[注9]です。

Grizllyのリリースが4月に入ってからだったこともあり、13.04のリリース時点には間に合わず、リリース後に-updatesや-backportsといったリポジトリ経由で別途提供されることになる可能性もあります。なお、OpenStack関連は「その時点で最新のstandardとLTSリリース」に対して提供されるため、12.04（12.04.2）用にもパッケージングが行われる予定です。

また、HAProxyを利用することで、OpenStackの各ノード（Quantum、Nova、Cinder、Keystone・Glance/Ceph）の擬似冗長構成を採ることができるようになっています。この機能はJuju[注10]を用いて制御できるため、数操作で冗長化されたOpenStack環境を構成することもできます。

■Wubi

Ubuntu 13.04では、「品質上の理由からWubi（Windows環境向けインストーラ）はリリースイメージに含まれない」という決定が行われています。これは、Windows 8上で期待どおりに動作しないことや、多くのバグを抱えたままになっているためです。今後バグが修正された時点で再びリリースイメージに含まれるようになる可能性もありますが、13.04ではWindows環境と併用する場合はDVD/USBからブートするか、あるいは仮想環境を利用することになります[注11]。

■“Ubuntu GNOME”と“Kylin”

13.04では、Ubuntuの公式なFlavourとして、「Ubuntu GNOME」と、「Ubuntu Kylin」が追加されました。前者は12.10で「GNOME Remix」と呼ばれていたGNOME3/GNOME Shellベースのデスクトップ環境、そして後者が中国政府が主導する「Kylin」プロジェクトのUbuntu版[注12]です。

Ubuntuには豊富なデスクトップ環境やアプリケーションが存在するため、インストールイメージに収録するプリセットを変更することで、さまざまなバリエーションを作ることができます。これを「Flavour」と呼びます。たとえばKDEをメインにしたKubuntu、Xfceをメインにした Xubuntuなどです。これら以外にもさまざまなバリエーションが存在し、Ubuntu的に公式に認定されたもののうち、特別な許可を得たものが「Ubuntu ****」あるいは「***buntu」のネーミングを使うことを許されます。これ以外に、公式な認定を受けたものが「Edition」、そうでないもの

注9）OpenStackのバージョン表記は、「年.（リリース回）」方式で、Grizzlyは「2013年の1回目のリリース」ですので2013.1です。
注10）https://blueprints.launchpad.net/ubuntu/+spec/servercloud-r-openstack-ha
注11）基本的にWubiは「NTFS領域にOSイメージを配置し、WindowsのブートローダーからLinuxカーネルをキックしてNTFS上のイメージをマウントする」というしくみで動作するもので、もともと非常にhackyな実装です。基本的に「テスト用」として提供されるものですので、これによる影響はそれほど大きくありません。UbuntuとWindowsをデュアルブートにすることもできますが、各OSでトラブルが起きた際のリカバリーの難易度が上がってしまうため、あまりお勧めしません。
注12）「Kylin」は「中国におけるスタンダードシステム」として開発されているOS環境で、以前はFreeBSDベースで実装されていました。これがUbuntu Chinese Remixベースの Ubuntu Kylinに引き継がれる形です。

を「Remix」と呼びます。この方針に基づいた審査をクリアしたうえで、GNOME Remixは「Ubuntu GNOME」とリネームされています。

13.04のサポート期間は9ヵ月

■ローリングリリースに関する議論

Raringのリリース前のタイミングで、Ubuntuでは「半年ごとのリリースは非効率的なので、ローリングリリースモデルに切り替え、リリース作業は2年に一度、LTSのみとするべきではないか」という議論が行われました。

これは、各リリース時に行われるQA作業、あるいは各リリース単位のセキュリティアップデート作業の負荷が大きなものになっている、という賛成派と、定期的なリリースは品質の担保として重要であり、同時にローリングリリースモデルはユーザが混乱しないように仕上げるのが困難なので切り替えるべきではない、という反対派に分かれ、綿密な議論が行われました。実際には賛成派と反対派も流動的で、「より良いリリース体制とはどのようなものか？　ただしコストは下げたい」という立ち位置のもと、各人がアイデアを出すという状態が続きました。こうして、約3ヵ月にわたる非常に長い議論の末、最終的に、「半年ごとにリリースすべきである」という結論となり、「6ヵ月ごとのリリース」というUbuntuのポリシーは維持されています。

■サポート期間の短縮

しかし、6ヵ月ごとにリリースを行う以上、作業コストはそれほど少なくなりません。Jenkinsなどを用いた継続的インテグレーションのための自動化は進められていますが、根本的な作業コストそのものを軽減する必要がありました[注13]。

そこでUbuntu Teamが下した結論は、「非LTSリリースのサポート期間を半分にすること」です。これにより、「少なくとも5～6系統のUbuntuのメンテナンスをする」という状態から脱却できます[注14]。併せて、「非LTS」リリースについては「standard」と呼ぶことになりました。この変更は13.04から適用され、以前の「standard」リリース、11.10と12.10には影響せず、これらのリリースは18ヵ月のサポートが提供されることになります（ただし、11.10は5月9日に18ヵ月のサポート期間を終了します）。

6ヵ月ごとのリリースと、それぞれ9ヵ月のサポート期間が提供されることから、今後「standard」リリースを使う場合は、「新しいバージョンが出てから3ヵ月以内にアップグレードする」という運用になります。

13.04以外の動き

12.10のリリースから13.04までの間に、Ubuntuには多くの変化が起きました。13.04を語るにはこれらについても把握しておく必要があります。簡単に見て行きましょう。

Steam for ubuntu

現在のUbuntuでは、いわゆる「PC向けゲーム」で遊ぶことができます。これは、大手ゲームパブリッシャーであるValveが、自社のゲーム配信システム『Steam』をUbuntu向けにリリースしたためです。

これは2012年夏から宣言されていたもので、11月にベータ版が、そして2月に正式版がリリースされました。推奨環境はUbuntu 12.04 LTSと12.10です。Valveからリリースされるゲームには「Half-Life」や「Counter Strike」「The

注13）これは、Ubuntuの支援企業であるCanonical社の収支にも大きな影響を与えます。リリースされた各Ubuntuのバージョンのメンテナンスは、おもにCanonicalに雇用された開発者によって行われており、メンテナンスコストの増大はそのままCanonicalの収支に影響を与えます。Ubuntu Touchなどの新しい方向へ舵を切ろうとしているCanonicalにとっては、メンテナンスのためにコストを取られるのはあまり望ましい展開ではありません。

注14）もともと「18ヵ月のサポート期間を提供する」というポリシーは、LTSの影も形もない初期リリース、2004年10月にリリースされた4.10で採用されたものです。この後、エンタープライズ用途などでの利用に適合するように6.06でLTSがリリースされるポリシーに切り替わっています。

Elder Scrolls V: Skyrim」など、非常に強力なタイトルがそろっており、「ゲームプラットフォームの1つ」としてUbuntu搭載PCが選択肢になり得る状態になりました。残念ながらLinux向けにリリースされるタイトルは現状では限定的なものの、今後、Ubuntuがユーザを獲得していくうえで強い味方になる可能性を秘めています[注15]。

Mirへの移行

UNIX環境ではGUI表示時にはX Window Systemをディスプレイサーバとして使う、という常識が変わろうとしています。古典的なX環境はあまりにも複雑過ぎるため、モバイルなどの非力な環境で使ったり、あるいは積極的な機能追加を行う際の実装難度が高い、といった問題を抱えています。こうした問題を解決するため、「Xに代わる実装」が作られようとしています。代表的な実装はWaylandで、以前はUbuntuもWaylandへの移行を宣言していました。

しかし、13.04の開発フェーズにおいて、UbuntuではCanonicalが独自に開発したディスプレイサーバ「Mir」（ロシアの宇宙ステーション、ミールと同じスペルです）を採用することが宣言されました。いわゆるデスクトップ環境、すなわちPC向けだけでなく、モバイルにも展開される予定で、Ubuntu TouchについてはMir上で動作するデモ版が存在します。WaylandではCanonicalが求める機能が実現できないから、ということが理由とされています。

Ubuntu Touch

13.04世代には間に合わないものの、Ubuntuには非常に大きな変化が訪れようとしています。それが「Ubuntu Touch」で、スマートフォンとタブレットに対応したUbuntu[注16]です。アプリケーション開発のためにQMLベースのSDKが提供され、また、HTML5アプリケーションを動作させることができる、新しいスマートフォン環境です。UnityのQML移植版が動作し、デスクトップに触ったことがあれば、直感的に操作できるようになっています。画面左側に表示されるLauncherもデスクトップ同様です。

また、Ubuntu Touchには「標準的なUbuntuアプリケーション」の動作をとくに阻害する要素がないため、実装上の問題が出てこなければ、必要に応じてデスクトップ版と同じアプリケーションを利用できる可能性があり、また、クレードルなどからキーボード、マウス、ディスプレイに接続することで、そのままスマートフォンやタブレットをデスクトップPCライクに利用する機能も提供される予定です。

リリース時点ではAndroidカーネル（しかも通常のものではなく、CyanogenModのもの）をそのまま利用したもので、一部のユーザの間で利用されている「AndroidにchrootしたUbuntu環境を導入する」というものに近似した実装でした。ただし、これはあくまでプレビュー版で、最終的にMirベースのGUIを実現するものになる予定です。

UbuntuにはUbuntu TVや自動車向けのプロダクトも存在するため、このプロジェクトが首尾よく進むと、「コンピュータへのインターフェースはすべてUnity」という未来がやってくる可能性があります。SD

注15） いわゆる「ゲーム向けPC」の中にもUbuntuを搭載したモデルが登場しており、USAのDell（Alienwareブランド）からUbuntuプレインストールモデルが提供される予定です。
http://alienware.com/ubuntu/

注16） 開発コードネーム的なネーミングでは、「Ubuntu for Phablet」という名称もあります。Phabletはスマートフォンとタブレットの中間的なデバイス、とくに5～7インチのモバイル回線に対応したデバイスを指す単語ですが、UbuntuでのPhabletは「PhoneとTablet」というニュアンスを持っています。

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2回

ちょこっとコードを書いて、より本らしく!

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/

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。前回はコードを書かずに最もシンプルな写真集アプリを作ってみました。今回はちょっとコードを入れるだけで、前回より一歩進んだアプリになります。

おさらい

前回の記事で説明した写真集アプリは完成しましたでしょうか? 動くものを完成させる喜びを知っていただくために、前回はプログラムのコードを一切書かないで簡単にアプリを完成させてみました。今回は少しだけコードを書いて機能アップしたアプリにしてみましょう。

アプリの基本部分を作成

今回のアプリはこれから先の連載の基礎となる部分を作成します。一見、前回のものと似てはいますが、メモリ管理などをきちんとできるものになっています。前回はコードを書かないで作る都合上、メモリ管理のことなどは考慮しませんでした。ですので前回の作り方ではページを作るたびにメモリを消費してしまい、何千ページもあるものを作ると問題が出る可能性があります。今回はこの部分をクリアするためにNavigation Controllerを使い、コードも多少書いています。

連載2回目ですので、まずは復習もかねて新プロジェクトを作って前回のステップ11まで進めてください。これがこれから作るアプリの基本部分となります。なお、お手元に前号がない場合は連載のまとめページ[注1]にステップ11までの手順を公開していますのでそちらをご覧ください。

さて、準備できましたか? Xcodeの操作にもだいぶ慣れてきたのではないでしょうか。

注1) http://www.gimmiq.net/p/sd.html

ページが戻れるしくみを入れる

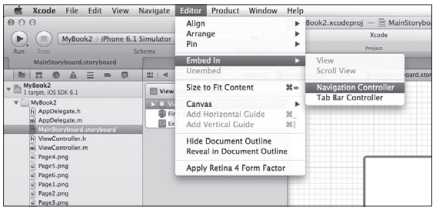
前回の記事で作成したアプリはページを進めるのみで、戻ることはできませんでした。より本らしくするためにページを戻るしくみを入れたアプリにしましょう。

ステップ1

なるべく簡単にページの前後への移動、つまりナビゲーションを実現するためにNavigation Controllerを使います。これはその名のとおり画面の遷移をコントロールするパーツだと考えてください。

StoryBoardにある1ページ目となるView Controllerを

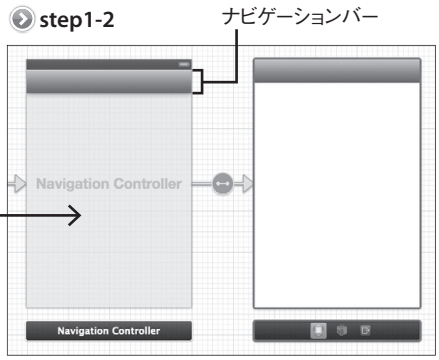
step1-1



クリックし選択状態にしたまま、メニューバーから[Editor]－[Embed In]－[Navigation Controller]を選択します(図step1-1)。

StoryBoardが図step1-2のような状態になったと思います。これは**Navigation Controllerの管理下に1ページ目のView Controllerが配置されている**状態を表しています。

step1-2

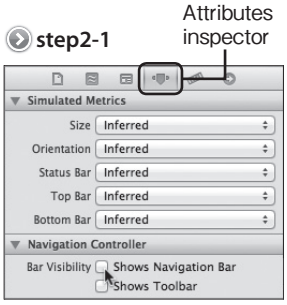


ステップ2

このままですと、実行時に画面上部にナビゲーションバー(図step1-2参照)が表示されてしまいます。写真集アプリなどでは表示の邪魔になってしまいますのでNavigation Controllerの機能は使いつつ、表示自体はしないようにします。

StoryBoardのNavigation Controllerページをクリックし、選択状態にしたままXcodeの右カラムにある「Attributes inspector(アトリビュートインスペクター)」をクリックしてください。「Bar Visibility」の「Show Navigation Bar」のチェックを外します。これでNavigation Controllerは表示されなくなります(図step2-1)。

step2-1

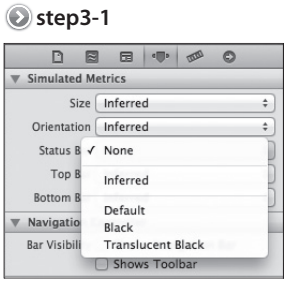


ステップ3

続いてステータスバーも非表示にします。同じくアトリビュートインスペクターの「Status Bar」の項目をクリックし、「None」を選択してください(図step3-1)。

これでページの前後への移動の準備ができました。

step3-1



1～2ページ目を作る

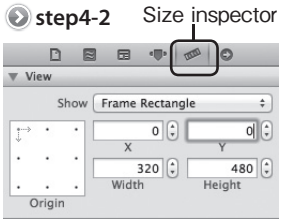
それでは1ページ目と2ページ目を作りましょう。

ステップ4

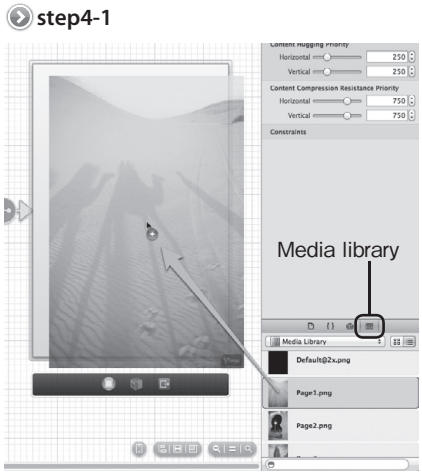
まずは、1ページ目に写真を貼り付けます。Xcodeの右カラムの「Media library」から1ページ目の画像を1ページ目のView Controllerまでドラッグ&ドロップします(図step4-1)。

配置後、画像をクリックし選択状態にしたまま、Xcodeの右カラムの「Size inspector(サイズインスペクター)」のアイコンをクリックすると画像の位置(X, Y座標)と大きさ(幅、高さ)が確認できます。図step4-2と同じ値になって

step4-2



step4-1

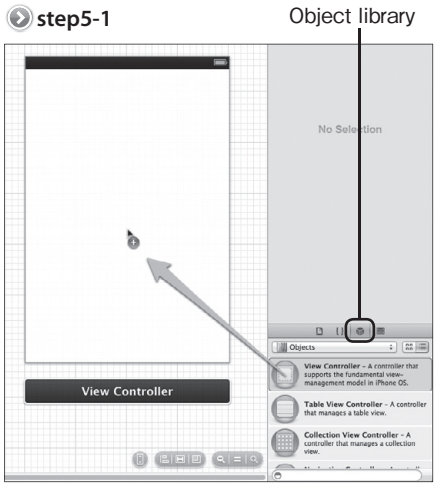


いるか確認してください。なっていなければここの数値を直接編集して修正します。

ステップ5

2ページ目も用意しましょう。Xcode右下の「Object library」から「View Controller」をStoryBoardの1ページ目の横の位置にドラッグ&ドロップします(図step5-1)。このView Controllerが2ページ目になります。ステップ4と同様の手順で、2ページ目のView Controllerにも2ページ目の画像を貼り付けてください。

step5-1



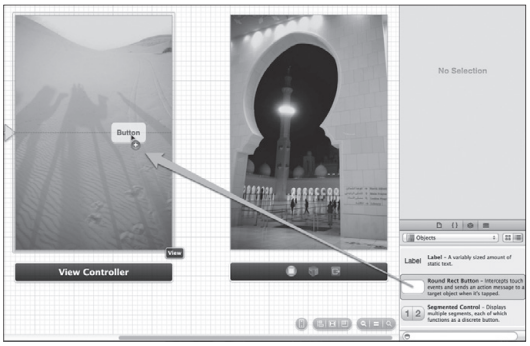
ステップ6

ページ移動用のボタンを配置します。「Object library」から「Round Rect Button」を1ページ目にドラッグ&ドロップし貼り付けます(図step6-1)。

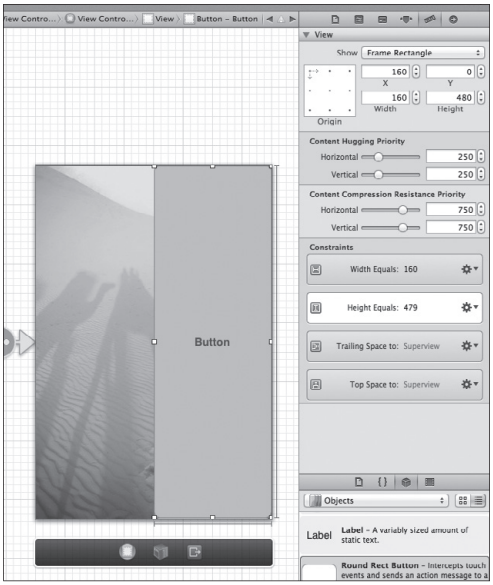
貼り付けた「Round Rect Button」をクリックして選択状態にし、右カラムのサイズインスペクターのアイコンをクリックします。図step6-2と同じ値になるように各値を入力してください。これは1ページ目から2ページ目へ、ページを**進める**ためのボタンになります。

2ページ目にも同様に「Round Rect Button」をドラッグ&ドロップします。位置とサイズは図step6-3にあわせてください。こちらは2ページ目から1ページ目へ、ページを**戻る**ためのボタンになります。

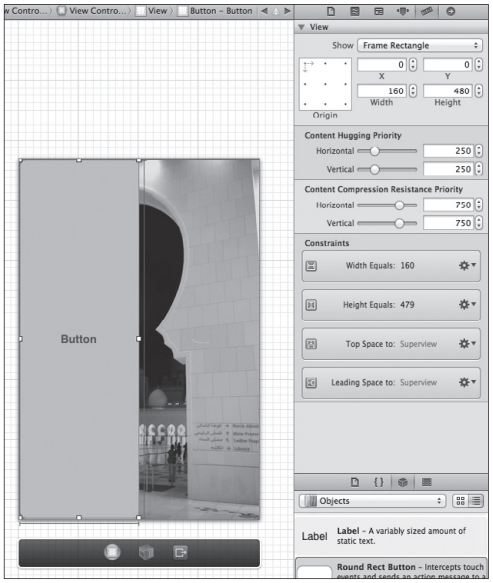
step6-1



step6-2



step6-3



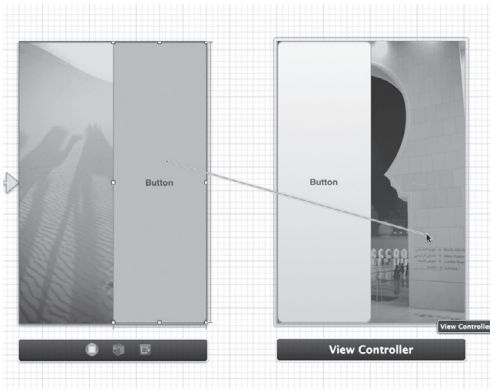
ページをつなげる

1ページ目と2ページ目を行き来できるようにしましょう。

ステップ7

1ページ目のボタンをcontrolキーを押しながらドラッグし、表示された青い線を2ページ目まで引っ張り、2ページ目が青い枠で囲まれたらマウスのボタンを放します(図step7-1)。すると図step7-2のようなメニューが出ますので、「push」をクリックします。これでページが進む方向には移動できるようになりました。

step7-1



ステップ8

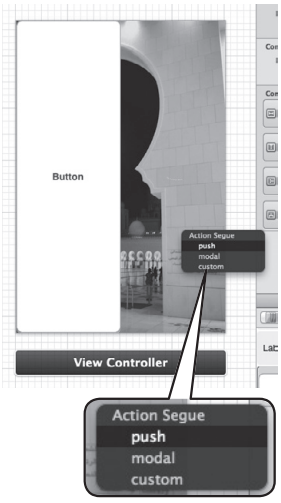
ページを戻る機能を組み込むにはソースコードのほうに少しだけ手を加える必要があります。恐れなくても大丈夫です。今回追加するコードは大変簡単なもので、厳密に言うと関数を追加するだけで、関数の中身は空のままですので難しいことはありません。

まずは各ページに対応したソースコードを用意しましょう。Xcodeのメニューバーから[File]－[New]－[File]を選択します。

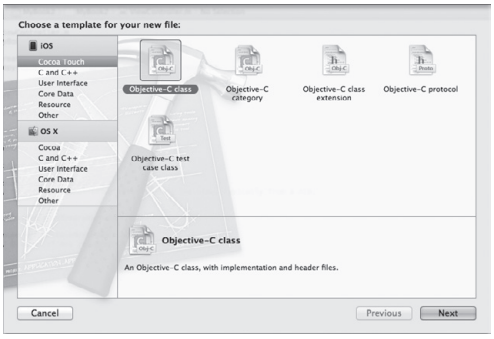
表示されたウィンドウ(図step8-1)左のカテゴリから「Cocoa Touch」を、右のファイル種類は「Objective-C class」をそれぞれ選択し、[Next]をクリックします。

次のウィンドウ(図step8-2)ではソースコードのファイルネームなどを決めます。「Class」の項目にクラス名を入れます。この名前がそのままファイルネームになります。ここでは1ページ目の制御に使うソースであることをわかりやすくするために「Page1ViewController」という名前にしましょう。今回はStoryBoard上の各ページのView Controllerに対応したソースを用意しますので「Subclass of」は「UIViewController」を選択します。[Next]ボタンを押すと保存先を指定するダイアログボックスが表示されるので、保存先にプロジェクトフォルダを指定します。作例ではデスクトップに置いた「MyBook」フォルダです。

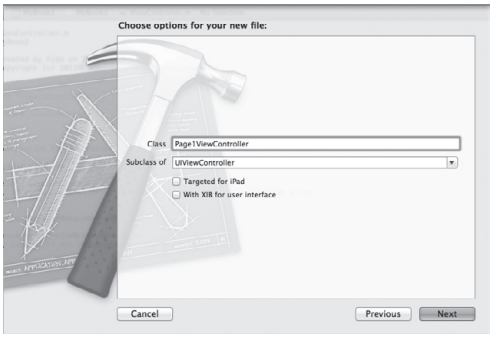
step7-2



step8-1

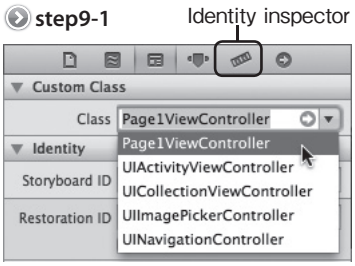


step8-2



ステップ9

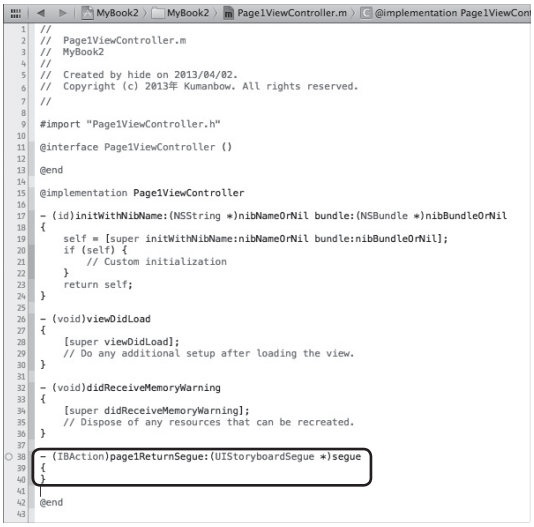
step9-1



1ページ目のView Controllerと1ページ目のソースファイルを関連づけます。StoryBoardの1ページ目のView Controllerを選択し、右カラムの「Identity inspector(アイデンティティインスペクター)」のアイコンをクリックします。「Class」の項目をクリックし「Page1ViewController」を選択します(図step9-1)。

ステップ10

step10-1



いよいよソースコードを修正します。左カラムの「Project Navigator」から「Page1ViewController.m」をクリックし、ソースコードを表示してください。ファイル末尾の「@end」の前に次のコードを追記します(図step10-1)。

```
- (IBAction)page1ReturnSegue:
(UIStoryboardSegue *)segue
{
}
```

これは**関数**と呼ばれるものです。関数とはあるまとまった処理をするものと考えてください。追記したこの関数が他のページから1ページ目に戻ってくるための目印となりますので、それとわかる名前にしましょう。作例では「page1ReturnSegue」という名前にしてあります。

関数では「{」から「}」までの間に処理のコードを記述するのですが、今回はページを戻る際の目印としてだけ使用しますので、関数の中身は空っぽのままでかまいません。

ステップ11

step11-1

2ページ目のボタンを1ページ目につなぎましょう。StoryBoardの2ページ目に置いたボタンをcontrolキーを押しながらドラッグし、View Controller下にある緑色の四角いアイコン「Exit」につなげます(図step11-1)。

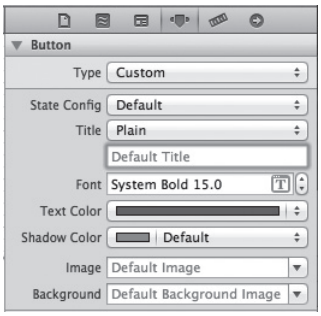
図step11-2のようなメニューが表示されますので、「page1ReturnSegue:」をクリックします。このメニューに表示されたのは先ほどソースコードを追記した、ページを戻るための目印としての関数です。

step11-2

ステップ12

ボタンが写真を隠してしまっていてじゃまですので、ボタンを透明にします。ボタンをクリックして選択状態にし、右カラムのアトリビュートインスペクターアイコンをクリックします。図step12-1のように「Type」を「Custom」に設定し、「Title」に入力されている「Button」という文字列を削除します。これでボタンが透明になりました。

step12-1



実行して確認

ステップ13

Xcodeのウィンドウ左上にある「Run(実行)」のアイコンをクリックしましょう。iPhoneシミュレーターが起動し、1ページ目の画像が表示されていると思います。1ページ目の右半分をタップすると2ページ目へ、2ページ目の左半分をタップすると1ページ目へとページの行き来ができるはずです。

ページを増やして完成

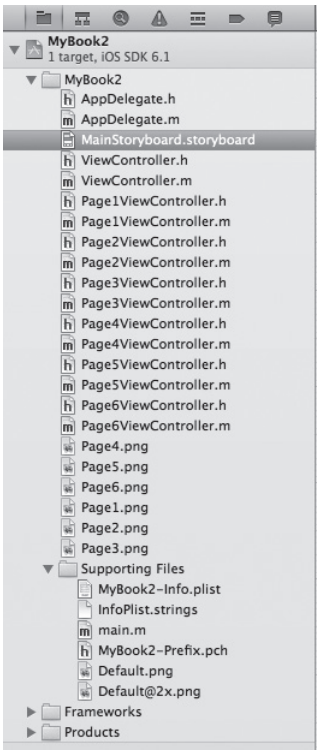
ステップ14

あとはステップ4〜ステップ12を繰り返して、ページを増やしていくだけです。注意点は次の2つです。

- **View Controllerとソースファイルはページごとに用意する**
- **ソースファイルとその中に記述する戻り用の関数の名前は、どのページのものなのかわかりやすくする**

サンプルでは第1回目同様6ページまで作成してみました。このときのファイル一覧は図step14-1のようになりました。

step14-1



今回作ったアプリがこれからの基本形となります。次回からはこれを発展させ、アプリでしかできないさまざまな要素を入れていきたいと思いますのでご期待ください! SD

いたのくまんぼう／Itano Kumanbow Twitter @Kumanbow

神奈川工科大学非常勤講師。リオさんとはGimmiQ名義で「MagicReader」(手を使わずにページがめくれる電子書籍ビューワ)をリリース。個人ではNinebonz名義で「Crop It Cam!」(おしゃれな切り抜き写真カメラ)、「i列車の車窓からーそうだ! 京都行こう!ー」(バーチャル旅行アプリ)など。アプリ紹介サイト「あぷまがどっとねっと(http://appmaga.net/)」の技術サポータ。

リオ・リーバス／Leo Rivas Twitter @StudioLoupe

iOSアプリ開発を中心に電子絵本作家・漫画家として活動中。個人ではスタジオルーペとして数字を指でドラッグ&ドロップ保存できる「フュージョン計算機(FusionCalc)」が代表作。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。現在HPにてWeb漫画「HELL BASEBALL」を連載中。

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9回

Intel VT-xを用いたハイパーバイザの実装 その5「vmm.koへのVMExit」

浅田 拓也（ASADA Takuya）Twitter @syuu1228

はじめに

前回は、vmm.koがVM_RUN ioctlを受け取ってからVMEntryするまでの処理を解説しました。今回はVMX non root modeからvmm.koへVMExitしてきたときの処理を解説します。

解説対象のソースコードについて

本連載では、FreeBSD-CURRENTに実装されているBHyVeのソースコードを解説しています。

このソースコードは、FreeBSDのSubversionリポジトリから取得できます。リビジョンはr245673を用いています。お手持ちのPCにSubversionをインストールし、次のようなコマンドでソースコードを取得してください。

```
svn co -r245673 svn://svn.freebsd.org/base/head src
```

/usr/sbin/bhyveによる仮想CPUの実行処理のおさらい

/usr/sbin/bhyveは仮想CPUの数だけスレッドを起動し、それぞれのスレッドが/dev/vmm/${name}に対してVM_RUN ioctlを発行します（**図1**）。vmm.koはioctlを受けてCPUをVMX non root modeへ切り替えゲストOSを実行します（これがVMEntryです）。

VMX non root modeでハイパーバイザの介入が必要な何らかのイベントが発生すると制御がvmm.koへ戻され、イベントがトラップされます（これがVMExitです）。

イベントの種類が/usr/sbin/bhyveでハンドルされる必要のあるものだった場合、ioctlはリターンされ、制御が/usr/sbin/bhyveへ移ります。イベントの種類が/usr/sbin/bhyveでハンドルされる必要のないものだった場合、ioctlはリターンされないままゲストCPUの実行が再開されます。

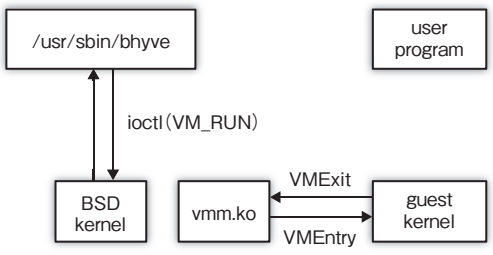
今回は、VMX non root modeからvmm.koへVMExitしてきたときの処理を見ていきます。

vmm.koがVM_RUN ioctlを受け取ってからVMEntryするまで

vmm.koがVM_RUN ioctlを受け取ってからVMEntryするまでの処理について、順を追って見ていきます。今回は、I/O命令でVMExitしたと仮定します。

前回解説のとおり、VMExit時のVMX root modeの

▼図1　VM_RUN ioctlによる仮想CPUの実行イメージ



再開アドレス(RIP[注1])はVMCSのHOST_RIPで指定されたvmx_longjmpに設定されています。vmx_longjmpはvmx_setjmpと対になっている関数で、POSIX APIのsetjmp/longjmpに近い動作を行います。つまり、vmx_longjmpはvmx_setjmpが呼ばれた直後のアドレスへジャンプします。結果として、VMExitされるとvmx_longjmpを経由しvmx_runのwhileループへ戻ってくることになります。

また、vmx_setjmpはどこからreturnしてきたかを示すために戻り値を用いています。ここではvmx_longjmpから戻ってきたことを表すVMX_RETURN_LONGJMPを返します。

では、以上のことをふまえてソースコードの詳細を見ていきましょう。**リスト1**、**リスト2**、**リスト3**に示します。解説キャプションの番号は、注目すべき処理の順番を示します。

注1) すでに前回までの記事でも「RIP」と表記していますが、なんのことだろうと思った方もいらっしゃるかもしれません。これは、x86_64アーキテクチャでの64bit幅のEIPレジスタ(インストラクションポインタ)の名前です。ほかにもEAX、EBXレジスタがRAX、RBXのような名前になっています。

sys/amd64/vmm/intel/vmx.c

intel/ディレクトリにはIntel VT-xに依存したコード群が置かれています。今回はゲストマシン実行ループの中心となるvmx_runと、VMExitのハンドラ関数であるvmx_exit_processを解説します。

▼リスト1 sys/amd64/vmm/intel/vmx.c

```
…… (省略) ……
 1197:  static int
 1198:  vmx_exit_process(struct vmx *vmx, int vcpu, struct vm_exit *vmexit)
 1199:  {
 1200:      int error, handled;
 1201:      struct vmcs *vmcs;
 1202:      struct vmxctx *vmxctx;
 1203:      uint32_t eax, ecx, edx;
 1204:      uint64_t qual, gla, gpa, cr3, intr_info;
 1205:
 1206:      handled = 0;
 1207:      vmcs = &vmx->vmcs[vcpu];
 1208:      vmxctx = &vmx->ctx[vcpu];
 1209:      qual = vmexit->u.vmx.exit_qualification;
 1210:      vmexit->exitcode = VM_EXITCODE_BOGUS;
 1211:
 1212:      switch (vmexit->u.vmx.exit_reason) {
…… (中略) ……
 1238:      case EXIT_REASON_INOUT:
 1239:          vmexit->exitcode = VM_EXITCODE_INOUT;
 1240:          vmexit->u.inout.bytes = (qual & 0x7) + 1;
 1241:          vmexit->u.inout.in = (qual & 0x8) ? 1 : 0;
 1242:          vmexit->u.inout.string = (qual & 0x10) ? 1 : 0;
 1243:          vmexit->u.inout.rep = (qual & 0x20) ? 1 : 0;
 1244:          vmexit->u.inout.port = (uint16_t)(qual >> 16);
 1245:          vmexit->u.inout.eax = (uint32_t)(vmxctx->guest_rax);
 1246:          break;
…… (中略) ……
 1310:      default:
 1311:          break;
 1312:      }
…… (中略) ……
 1351:      return (handled);
 1352:  }
 1353:
 1354:  static int
 1355:  vmx_run(void *arg, int vcpu, register_t rip)
```

⑲Exit Qualificationを取り出す

⑳I/O命令でVMExitした場合、Exit Reason 30(EXIT_REASON_INOUT)となる

㉑Exit Reasonを代入

㉒Exit Qualificationからアクセス幅を代入

㉓Exit Qualificationからアクセス方向を代入

㉔Exit QualificationからString命令かどうかのフラグを代入

㉕Exit Qualificationからrep prefix付きかどうかのフラグを代入

㉖Exit Qualificationからポート番号を代入

㉗raxレジスタの値を代入

㉘EXIT_REASON_INOUTでは、ユーザランドでのエミュレーション処理を要求するためhandled = 0を返す

```
 1356:  {
……(中略)……
 1394:      do {
……(中略)……
 1398:          rc = vmx_setjmp(vmxctx);
……(中略)……
 1402:          switch (rc) {
……(中略)……
 1412:          case VMX_RETURN_LONGJMP:
 1413:              break;          /* vm exit */
……(中略)……
 1437:          }
 1438:
 1439:          /* enable interrupts */
 1440:          enable_intr();
 1441:
 1442:          /* collect some basic information for VM exit processing */
 1443:          vmexit->rip = rip = vmcs_guest_rip();
 1444:          vmexit->inst_length = vmexit_instruction_length();
 1445:          vmexit->u.vmx.exit_reason = exit_reason = vmcs_exit_reason();
 1446:          vmexit->u.vmx.exit_qualification = vmcs_exit_qualification();
……(中略)……
 1455:          handled = vmx_exit_process(vmx, vcpu, vmexit);
 1456:          vmx_exit_trace(vmx, vcpu, rip, exit_reason, handled);
 1457:
 1458:      } while (handled);
……(中略)……
 1481:      return (0);
……(中略)……
 1490:  }
```

⑫vmx_returnからここへリターンされてくる。戻り値としてVMX_RETURN_LONGJMPを返す

⑬rcはVMX_RETURN_LONGJMP

⑭VMCSからゲストOSのRIPを取得してvm_exit構造体にセット

⑮VMCSからRIPが指している命令の命令長を取得してvm_exit構造体にセット

⑯VMCSからExit reasonを取得してvm_exit構造体にセット

⑰VMCSからExit qualificationを取得してvm_exit構造体にセット

⑱vmx_exit_processでExit reasonに応じた処理を実行

㉙handled = 0が返ったため、ループを抜けvmx_runから抜ける

sys/amd64/vmm/intel/vmx_support.S

vmx_support.SはC言語で記述できない、コンテキストの退避／復帰やVT-x拡張命令の発行などのコードを提供しています。今回は、vmx_setjmp・vmx_longjmpを解説します。

▼リスト2　sys/amd64/vmm/intel/vmx_support.S

```
……(中略)……
  100:  /*
  101:   * int vmx_setjmp(ctxp)
  102:   * %rdi = ctxp
  103:   *
  104:   * Return value is '0' when it returns directly from here.
  105:   * Return value is '1' when it returns after a vm exit through vmx_longjmp.
  106:   */
  107:  ENTRY(vmx_setjmp)
  108:      movq    (%rsp),%rax           /* return address */
  109:      movq    %r15,VMXCTX_HOST_R15(%rdi)
  110:      movq    %r14,VMXCTX_HOST_R14(%rdi)
  111:      movq    %r13,VMXCTX_HOST_R13(%rdi)
  112:      movq    %r12,VMXCTX_HOST_R12(%rdi)
  113:      movq    %rbp,VMXCTX_HOST_RBP(%rdi)
  114:      movq    %rsp,VMXCTX_HOST_RSP(%rdi)
  115:      movq    %rbx,VMXCTX_HOST_RBX(%rdi)
  116:      movq    %rax,VMXCTX_HOST_RIP(%rdi)
  117:
  118:      /*
  119:       * XXX save host debug registers
```

①スタック上のリターンアドレスを%raxに取り出す

②次の行では、VMEntry時にVMCSへ自動保存されないホストOSのレジスタをvmxctx構造体へ保存している

③%raxに取り出したリターンアドレスをvmxctx構造体のhost_ripメンバに保存。ここまでがVMExit前に行われている処理

```
120:      */
121:     movl    $VMX_RETURN_DIRECT,%eax
122:     ret
123: END(vmx_setjmp)
124:
125: /*
126:  * void vmx_return(struct vmxctx *ctxp, int retval)
127:  * %rdi = ctxp
128:  * %rsi = retval
129:  * Return to vmm context through vmx_setjmp() with a value of 'retval'.
130:  */
131: ENTRY(vmx_return)
132:     /* Restore host context. */
133:     movq    VMXCTX_HOST_R15(%rdi),%r15
134:     movq    VMXCTX_HOST_R14(%rdi),%r14
135:     movq    VMXCTX_HOST_R13(%rdi),%r13
136:     movq    VMXCTX_HOST_R12(%rdi),%r12
137:     movq    VMXCTX_HOST_RBP(%rdi),%rbp
138:     movq    VMXCTX_HOST_RSP(%rdi),%rsp
139:     movq    VMXCTX_HOST_RBX(%rdi),%rbx
140:     movq    VMXCTX_HOST_RIP(%rdi),%rax
141:     movq    %rax,(%rsp)           /* return address */
142:
143:     /*
144:      * XXX restore host debug registers
145:      */
146:     movl    %esi,%eax
147:     ret
148: END(vmx_return)
149:
150: /*
151:  * void vmx_longjmp(void)
152:  * %rsp points to the struct vmxctx
153:  */
154: ENTRY(vmx_longjmp)
155:     /*
156:      * Save guest state that is not automatically saved in the vmcs.
157:      */
158:     movq    %rdi,VMXCTX_GUEST_RDI(%rsp)
159:     movq    %rsi,VMXCTX_GUEST_RSI(%rsp)
160:     movq    %rdx,VMXCTX_GUEST_RDX(%rsp)
161:     movq    %rcx,VMXCTX_GUEST_RCX(%rsp)
162:     movq    %r8,VMXCTX_GUEST_R8(%rsp)
163:     movq    %r9,VMXCTX_GUEST_R9(%rsp)
164:     movq    %rax,VMXCTX_GUEST_RAX(%rsp)
165:     movq    %rbx,VMXCTX_GUEST_RBX(%rsp)
166:     movq    %rbp,VMXCTX_GUEST_RBP(%rsp)
167:     movq    %r10,VMXCTX_GUEST_R10(%rsp)
168:     movq    %r11,VMXCTX_GUEST_R11(%rsp)
169:     movq    %r12,VMXCTX_GUEST_R12(%rsp)
170:     movq    %r13,VMXCTX_GUEST_R13(%rsp)
171:     movq    %r14,VMXCTX_GUEST_R14(%rsp)
172:     movq    %r15,VMXCTX_GUEST_R15(%rsp)
173:
174:     movq    %cr2,%rdi
175:     movq    %rdi,VMXCTX_GUEST_CR2(%rsp)
176:
177:     movq    %rsp,%rdi
178:     movq    $VMX_RETURN_LONGJMP,%rsi
179:
180:     addq    $VMXCTX_TMPSTKTOP,%rsp
181:     callq   vmx_return
182: END(vmx_longjmp)
```

⑧次の行では、VMExit時にVMCSから自動復帰されなかったホストOSのレジスタを復帰している

⑨vmxctx構造体のhost_ripメンバから%raxへリターンアドレスをコピー

⑩リターンアドレスをスタックにセット

⑪⑩でセットしたアドレスへリターン

④VMExit時にはここから実行が再開される。以降の行で参照されている%rspはVMEntry時に自動保存され、VMExit時に自動復帰されている

⑤次の行では、VMExit時にVMCSへ自動保存されなかったゲストOSのレジスタを保存している

⑥戻り値としてVMX_RETURN_LONGJMPを指定

⑦vmx_returnを呼び出してホストOSのレジスタを復帰する

sys/amd64/vmm/vmm.c

vmm.cは、Intel VT-xとAMD-Vの2つの異なるハードウェア仮想化支援機能のラッパー関数を提供しています。今回はvmx_runのラッパー関数のvm_runを解説します。

▼リスト3　sys/amd64/vmm/vmm.c

```
……(中略)……
 672:  int
 673:  vm_run(struct vm *vm, struct vm_run *vmrun)
 674:  {
……(中略)……
 681:      vcpuid = vmrun->cpuid;
……(中略)……
 686:      vcpu = &vm->vcpu[vcpuid];
 687:      vme = &vmrun->vm_exit;
 688:      rip = vmrun->rip;
……(中略)……
 701:      error = VMRUN(vm->cookie, vcpuid, rip);
……(中略)……
 709:      /* copy the exit information */
 710:      bcopy(&vcpu->exitinfo, vme, sizeof(struct vm_exit));
……(中略)……
 757:  }
```

㉚vmx_runはEXIT_REASON_INOUTをハンドルしてここへ抜けてくる

㉛vm_exit構造体の値はユーザランドへの戻り値としてここでコピーされる

まとめ

VMX non root modeからvmm.koへVMExitしてきたときの処理について、ソースコードを解説しました。次回は、I/O命令によるVMExitを受けて行われるユーザランドでのエミュレーション処理について見ていきます。SD

Software Design 総集編 2001→2012

好評発売中!!

12年分のバックナンバーがこの1冊に！

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

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

さらに……
『Software Design総集編【1990～2000】』2013年5月25日発売予定！
【2001～2012】と合わせて購入すれば、約23年間のITの技術／ノウハウが手元にそろう！

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

技術評論社　全国の書店、またはオンライン書店でお買い求めください。
〒162-0846　東京都新宿区市谷左内町21-13　販売促進部　TEL:03-3513-6150　FAX:03-3513-6151

テキストデータならお手のもの

開眼シェルスクリプト

㈲ユニバーサル・シェル・プログラミング研究所　http://www.usp-lab.com
上田 隆一　UEDA Ryuichi　Twitter @uecinfo

第18回 サーバにデータを渡して処理させる ——nc、ssh、scpを使う

はじめに

みなさん、いい季節いかがお過ごしでしょうか。この原稿を書いているのが3月頭ですので、筆者は現在、杉からの花粉ハラスメントを受けている最中です。

そんな時差を利用した小話はいいとして、今回のテーマは「サーバを股にかける」です。「時をかける少女」じゃなくて、「サーバをかけるオッサン」になろうということで、いろいろネタを準備しました。

字しか書けない端末の良いところは、1つの端末に字を書くだけであっちこっちのコンピュータを気軽に使えることです。vncやリモートデスクトップではそうはいかず、あっちこっちの画面を覗いているうちに疲れてきます。データを移すのにも一苦労です。やりたいことに対して情報量が多いことは、けっして良いことではありません。

そういえば、ガンカーズのUNIX哲学には、主要な9ヵ条のほかに、二軍の10ヵ条がありますが、その中に、

「90パーセントの解決を模索せよ。」

というのがあります。プログラムを書いたり、仕事をしたりすると、本筋でない雑事が気になるものですが、それにはある程度目をつぶれということを言っています。

これは、時短の発想であるとも解釈できます。筆者の仕事の場合は、端末とWebブラウザがあれば、仕事の90%は片付いてしまいます。そのうちの端末を使う数十%の仕事は字だけを見てさっさと終わってしまうので、たとえ残りの10%で困ったとしても、メニューをマウスでクリクリしている人よりは、トータルでも時間を得しているはずです。

環境の準備

実行環境

今回は多種多様です。**図1**で簡単に説明します。なるべくみなさんを混乱させないように注意しながら話を進めます。

USP友の会のサーバは、www.usptomo.comというホスト名でDNSに登録されています。bsdは秘密のサーバですが、bsd.hoge.hogeで登録されているとしておきます。

鍵認証の設定で悩まないために

シェルスクリプトという範囲の話ではありませんが、今回はscpコマンドやsshコマンドを多用しますので、ssh接続の鍵認証の方法について触れておきます[注1]。いや、手順については「ssh 鍵認証」などと検索すれば方法が書いてあるのでここでは説明しません。が、鍵認証はクライアントとサーバ、公開鍵と秘密鍵が登場して、どっちで何をするのか慣れるまで非常に混乱するの

注1）パスワードを入れなくてもログインできるアレのことです。念のため。

で、そんな人のために、次の一文を書いておきます。

「ssh接続されるほう(サーバ)は危険に晒されるので、接続してくる奴をリスト化して管理しなければならない」

このリストに登録されるのは、「接続してくる奴」の公開鍵です。ですから、クライアント側では秘密鍵と公開鍵を準備し、公開鍵をサーバに登録してもらうという手続きを行うことになります。これを頭に入れて、設定をお願いします。

ほかのマシンと通信する

bashの/dev/tcp/

さて本題に入っていきましょう。まずはbashの機能を使ってみます。どのバージョンからかは調べていませんが、少なくとも3以降のbashには、**図2**の方法で、特定のホストの特定のポートにfileの内容を送信する機能があります。リダイレクトの左側は、echoでもgrepでもなんでもかまいません。

さっそく使ってみましょう。と言ってもこの機能単独だと、いたずら程度くらいしか思いつきませんので、**図3**のようにUSP友の会のサーバを餌食にしてみました。みなさんも何かメッセージを残してもらってかまいませんが、あまり連発しないでください。

図4のように調べるとわかりますが、/dev/tcp/はシステム側にあるわけではなく、bashが擬似的にファイルに見せかけているようです。

/dev/udp/も準備されていますので、UDPを

▼図1　登場するマシン、サーバ

```
1. 筆者のMacBook Air(uedamac)
uedamac:~ ueda$ uname -a
Darwin uedamac.local 12.2.1 Darwin Kernel Version 12.2.1: (略)
uedamac:~ ueda$ bash --version
GNU bash, version 3.2.48(1)-release (x86_64-apple-darwin12)
Copyright (C) 2007 Free Software Foundation, Inc.

2. VPS上のFreeBSD(bsd)
bsd /home/ueda$ uname -a
FreeBSD bsd.hoge.hoge 9.0-RELEASE FreeBSD 9.0-RELEASE #0: (略)

3. VPS上のUSP友の会サーバ(tomonokai)
[ueda@tomonokai ~]$ cat /etc/redhat-release
CentOS release 6.3 (Final)

4. ビジネス版Tukubaiが使えるサーバ(usp)
[ueda@usp ~]$ cat /etc/redhat-release
CentOS release 5.9 (Final)
```

▼図2　bashで通信するときの書式

```
$ cat file > /dev/tcp/<ホスト名>/<ポート番号>
```

▼図3　Apacheにいたずらする

```
macからUSP友の会のサーバにちょっかいを出す
uedamac:~ ueda$ echo aho > /dev/tcp/www.usptomo.com/80
USP友の会のサーバのログに記録が残る
[root@tomonokai ~]# tail -n 1 /var/log/httpd/access_log
123.234.aa.bb - - [03/Mar/2013:00:58:21 +0900] "aho" 301 231 "-" "-"
```

▼図4　/dev/tcpは存在しない

```
uedamac:~ ueda$ ls /dev/tcp
ls: /dev/tcp: No such file or directory
```

使うサービスにもちょっかいが出せます。

Netcatを使う

bashの/dev/tcp/を使うと、基本的にデータをポートに投げつけることしかできません。投げつけたデータの受け手として、Netcatを紹介します。

たいていの環境では、ncというコマンドでNetcatが使えます。bashからテキストを投げて、ncで受けてみましょう。もちろん文字は暗号化されずにそのまま送られるので、秘密の情報は送らないようにしましょう。この実験をするには、受信側で使うポートが開いている必要があります(**図5**)。

図6のようにシェルスクリプトにして実行すると、ちょっとしたサービスのように振る舞います。

NetcatはWikipediaに「ネットワークを扱う万能ツールとして知られる。」とあるように、単にポートをリッスンするだけでなく、データの送信側になったり、邪悪な組織のポートスキャナになったりします。

▼図5　10000番ポートで通信する

```
先にncで受信側のポートを開いておく
ncが立ち上がったままになる
[ueda@tomonokai ~]$ nc -l 10000 > hoge

データを投げる
uedamac:~ ueda$ echo ひえええええ > /dev/tcp/www.usptomo.com/10000
ncが終わって、hogeの中に文字列が
[ueda@tomonokai ~]$ cat hoge
ひえええええ
```

▼図6　whileループで何回も受信

```
[ueda@tomonokai ~]$ cat file.sh
#!/bin/bash

mkdir -p ./tmp/

n=1
while nc -l 10000 > ./tmp/$n.txt ; do
        n=$(( n + 1 ))
done

立ち上げる
[ueda@tomonokai ~]$ ./file.sh
送る
uedamac:~ ueda$ echo ひえええええ > /dev/tcp/www.usptomo.com/10000
uedamac:~ ueda$ echo どひぇー > /dev/tcp/www.usptomo.com/10000
uedamac:~ ueda$ echo NOOO! > /dev/tcp/www.usptomo.com/10000
Ctrl+C(シェルスクリプトを終了)してファイルができていることを確認
[ueda@tomonokai ~]$ ./file.sh
^C
[ueda@tomonokai ~]$ head ./tmp/{1,2,3}.txt
==> ./tmp/1.txt <==
ひえええええ

==> ./tmp/2.txt <==
どひぇー

==> ./tmp/3.txt <==
NOOO!
```

ファイルを転送する

さて、いつも大きなデータを扱っている人は、サーバ間で何十GBものファイルをコピーしなければいけないことがあります。このようなときは**図7**のように、普通はscpを使うことでしょう。図中の**-P 11111**は、USP友の会のサーバがデフォルトの22番でなく11111番でssh接続を受け付けているため、必要となります[注2]。

scpには圧縮してデータを送る-Cというオプションがあります。**図8**のように使います。ただ、圧縮はCPUを酷使するので効果のある場合は限られます。1回しか試していないのでかかった時間は参考程度にしかなりませんが、user時間で圧縮にかなり時間を使っていることがわかります。

実は、暗号化しなくてよいなら**図9**のように転送するほうが速いことがあります。user時間はほとんどゼロです。

CPUが速くて通信速度が遅いときは、scpの-Cオプションが有効になりますが、上のncの方法でgzipやbzip2などを挟んで送ったほうが、速いこともあります。速いこともある、というより、本来圧縮はscpの仕事ではないはずですし、圧縮の方式も自由に選べるべきですので、面倒ですがこっちのほうがUNIX的です。ただまあ、そういうチューニングは本当に困ったときだけにしておきましょう。

1つの巨大なファイルを複数のサーバにコピーしたい場合は、**図10**のようなことを試みても良いでしょう。頭がこんがらがるかもしれませんが、ちゃんと書けばちゃんと動きます。

この方法のようにサーバを数珠つなぎにすると、何台ものサーバに同時にコピーができます。ただし、サーバが同じハブにぶらさがっていると、ハブにトラフィックが集中します。

あともう1個だけ紹介します。sshコマンドを

注2）実際には別のポートを使っています。

▼図7　普通にscpでファイルをコピー

```
bsd /home/ueda$ time scp -P 11111 TESTDATA www.usptomo.com:~/
TESTDATA                          100% 4047MB   4.0MB/s   16:48

real    16m49.064s
user    3m2.550s
sys     13m38.727s
```

▼図8　圧縮送信したらかえって遅くなった

```
bsd /home/ueda$ time scp -C -P 11111 TESTDATA www.usptomo.com:~/
TESTDATA                          100% 4047MB   2.6MB/s   26:16

real    26m16.678s
user    20m33.275s
sys     6m55.593s
```

▼図9　ポートをダイレクトに使ってファイル転送

```
受信側で待ち受け
[ueda@tomonokai ~]$ nc -l 10000 > TESTDATA
送信
bsd /home/ueda$ time cat TESTDATA > /dev/tcp/www.usptomo.com/10000

real    12m3.584s
user    0m0.000s
sys     10m22.737s
```

使ってもファイルを転送できます(**図11**)。この例で、sshコマンドが標準入力を受け付けることがわかります。

リモートマシンで計算する

さて、もっと便利に使ってみましょう。このままではコピーだけで今回の記事が終わってしまいます(それはそれでおもしろいかもしれませんが……)。

たとえば、今使っているマシンが遅い場合や使いたいコマンドなどがインストールされていない状況を考えます。筆者の場合は、USP研究所のビジネス用Tukubaiコマンドを使いたい場合や、あるマシンのTeXの環境を使いたいという場合がこれに相当します。

一例として、手元にあるファイルをリモートのサーバでソートして戻してもらうことを考えましょう。

まず**図12**に、普通のシェルスクリプトを示します。これは、あるリモートのサーバにscpでファイルを送り込み、ソートしたあとにファイルを戻すという処理です。Macのsortコマンドで1千万行のソートなんかやってしまったらい

▼図10　一度の転送で2つのサーバにファイルをコピー

```
友の会サーバで10000番ポートからファイルへリダイレクト
[ueda@tomonokai ~]$ nc -l 10000 > TESTDATA
bsdサーバで9999番ポートからの出力をteeでファイルにためながら
友の会サーバにリダイレクト
bsd /home/ueda$ nc -l 9999 | tee TESTDATA > /dev/tcp/www.usptomo.com/10000
手元のMacからbsdサーバにデータを投げる
uedamac:~ ueda$ cat TESTDATA > /dev/tcp/bsd.hoge.hoge/9999
```

▼図11　sshコマンドの標準入力を使う

```
bsd /home/ueda$ time cat TESTDATA | ssh -p 11111 www.usptomo.com 'cat > TESTDATA'

real    16m22.054s
user    2m46.163s
sys     12m44.448s
```

▼図12　「べたな」リモートサーバの使い方

```
このデータ(1千万行)を左端の数字でソートしたい
uedamac:~ ueda$ head -n 2 TESTDATA10M
2377 高知県 -9,987,759 2001年1月5日
2910 鹿児島県 5,689,492 1992年5月6日
uedamac:~ ueda$ cat sort.sh
#!/bin/bash -xv
usp.usp-lab.comは、図1のuspサーバのフルドメイン
scp -P 11111 ./TESTDATA10M usp.usp-lab.com:~/
msortは、マルチスレッドの高速ソートコマンド
ssh -p 11111 usp.usp-lab.com "msort -p 8 key=1 ~/TESTDATA10M > ~/ueda.tmp"
scp -P 11111 usp.usp-lab.com:~/ueda.tmp ./TESTDATA10M.sort

手元のMacで実行
uedamac:~ ueda$ time ./sort.sh

real    4m1.717s
user    0m13.969s
sys     0m10.090s
結果が得られた
uedamac:~ ueda$ head -n 2 TESTDATA10M.sort
0000 岩手県 5,630,892 2006年5月26日
0000 新潟県 1,367,399 1998年8月22日
```

つ終わるのか読めないので、これくらいのことは行う価値はあります。

こういった通信ばっかりのシェルスクリプトを書いた人はそんなにいないと思いますが、シェルスクリプトなどしょせん、人の操作のメモ書きですので、いつもscp、sshを使っていれば理解できるでしょう。

ところでこのシェルスクリプトでは、中間ファイルがリモートのサーバにできてしまっていますが、これを避けるにはどうすればよいでしょうか。こういう中間ファイルは、計算を Ctrl+c などで中断した場合にリモートのサーバにゴミを残すことになります。処理によっては、次に計算したときに悪さをすることもあります。

これを解決するには「シェル芸」です。「開眼シェルスクリプト」という名前で連載をしていますが、

不要なシェルスクリプトと中間ファイルはゴミ

です。こんなもん、ワンライナーで十分です。**図13**に示します。

これで、sshでソートした出力は、**手元のMacの標準出力から出てきます**。sshが(リモートでなく)手元のマシンの標準入出力に字を出し入れしてくれることは、sshコマンドが手元のマシンで動いているので当然と言えば当然ですが、よくよく考えるととても便利なことです。ワンライナーとしては難解かもしれませんが、リモートとローカルがシームレスにつながっています。パイプラインですので、ストレージを使うこともありません。

ちょっとやり過ぎですが、筆者の自宅とサーバの間の通信速度がそんなに速くないので、gzip、gunzipを使って**図14**のようにチューンしたらさらに時短できました。

終わりに

今回も前回に引き続き作り物をさぼって、サーバを股にかけてデータをやりとりし、処理する方法について書きました。bashの通信機能や、ssh、scp、ncなどのコマンドについてちょっとした使い方を紹介しました。

マシンを複数台使うと頭の中が混乱しがちです。その点、sshをパイプにつなぐことを覚えると、あまり頭を悩ませずに複数のマシンを使いこなすことができます。パイプラインは一方通行で順番にサーバをつなげていくだけですので、頭の中でいろんなマシンの絵を同時に思い浮かべる負荷が不要です。マシン間の通信速度はまだ向上していくでしょうから、これからは使う人が増えるかもしれません。

通信を扱ってウォーミングアップできましたので、禁断のお題をやる覚悟ができました。次回からは「シェルスクリプトでCGI」というお題で作り物をしてみます。SD

▼図13　リモートサーバを使うワンライナー

```
uedamac:~ ueda$ time cat TESTDATA10M | ssh -p 10022 usp.usp-lab.com 'cat | ➪
msort -p 8 key=1' > TESTDATA10M.sort3

real    5m0.033s
user    0m14.077s
sys     0m9.415s
```

▼図14　圧縮を挟み込んだワンライナー

```
uedamac:~ ueda$ time gzip < TESTDATA10M | ssh -p 10022 usp.usp-lab.com 'gunzip | ➪
msort -p 8 key=1 | gzip' | gunzip > TESTDATA10M.sort3

real    1m10.669s
user    0m42.874s
sys     0m2.806s
```

短期集中連載

SDNは使えるのか、使うべきなのか？

仮想ネットワークの落とし穴

第1回

ファブリックモデルの検証
——TRILL、SPB、MACアドレス学習問題

株式会社データホテル
伊勢幸一（いせこういち）
Twitter @ ibucho

はじめに

2000年代末に出現したクラウドコンピューティングとOpenFlowによるSDN（Software Defined Networking）に刺激を受け、ネットワークの仮想化技術に対しさまざまなプロトコルが提案され導入が検討されています。ITU-T Y.3011勧告におけるネットワークの仮想化フレームワークでは、さまざまな物理コンピューティング資源を仮想的に統合し、さらにそれらをサービス別に分離分割して運用することを指しています。しかし、この仮想化技術は現在まだ研究開発段階にあり、一般的にとらえられている仮想ネットワークとは、コンピューティングリソースを含まず、物理ネットワークレイヤに依存しないネットワークコネクティビティを形成することを指す場合が多いでしょう。そのコンピューティング資源を柔軟に連結することを可能とする仮想ネットワークですが、その恩恵と引き換えにさまざまなことが犠牲になることはあまり議論されていません。本連載では、それら仮想的なコネクティビティを形成するネットワークの仮想化技術の概要と特徴を示しつつ、そのメリットではなく仮想ネットワーク導入を検討する際に考慮しなければならないデメリットに焦点を当てて考察したいと思います。

仮想ネットワークの必要性

そもそもオンプレミスシステムの運用やiDC（データセンター）などが顧客別に数十台のサーバを提供するホスティングシステムを運用する上で、現状のネットワークをさらに仮想化しなければならない理由はありません。EthernetとTCP/IPによって発展してきた現在のコンピューティングネットワークは、すでにプロトコル階層によって必要十分に仮想化されているからです。

たとえば、データリンク層プロトコルはメタルケーブルやファイバーケーブルなどの物理的な単一伝送媒体上に複数の論理的な通信チャンネルを形成します。またネットワーク層プロトコルは単一、もしくは複数のデータリンクセグメント上に論理的なネサブネットを構成し、さらにIPアドレスで一意に識別される物理サーバ上にはTCP/UDPポートによって複数のエンドシステムが個別に動作しています。これらの状態はまさにネットワークコンピューティングシステムを仮想化していることにほかなりません。

現状のネットワークシステムをさらに仮想化しなければならないのはいったいどのような環境なのでしょうか。SDNや仮想ネットワークの技術について議論する場合、それら技術の特徴

や想定されるユースケース、そして性能比較に始終するばかりで、それらの技術を導入する本質的必要性についてはあまり議論されない傾向にあります。しかし、本来はその必要性と実際に適用されるべき環境と状況をまず最初に精査するべきではないでしょうか。

仮想ネットワークの多くは、従来のL2/L3ネットワーク上に仮想的なL2もしくはL3セグメントをオーバーレイし、さらにそのセグメント最大数をIEEE802.1Qの最大VLAN数以上にすることを目的としています。つまり、物理的なMACアドレス、もしくはIPアドレスによって一意に識別されるサーバ上に複数の仮想サブネットが存在し、かつ、単一のネットワークシステム内にそのサブネットが4,094以上にある環境が仮想ネットワークのターゲットです。すると現状考えられる仮想ネットワークのユースケースとは仮想マシンによって4,000以上のテナントを収容するレンタルサーバ事業者かIaaS事業者でしょう。ただし、VLAN数が4,000を超えるからといって必ずしもネットワークを仮想化しなければならないというわけではありません。4,000以上の契約が見込めるならば、あらかじめ2つのネットワークシステムを用意するか、必要になった時点で別途もう1つのネットワーク・システムを構築すれば良いだけです。

1つのネットワークシステムで4,000以上のテナントを集約しなければならないというのは、個々のテナントが利用する仮想マシン数が随時大幅に変動し、その追加や除去要求を数分で対処しなければならない場合のみです。仮想マシン環境の変更に数時間、もしくは数日の猶予があれば物理セグメントを再構築しサーバを新たに用意して対処することはできます。つまり現実的には仮想ネットワークを必要とするのは現在4,000テナントを超える、もしくは手が届きそうなパブリッククラウド事業者であり、国内ではパブリッククラウドシェアの上位5社程度(筆者調べ)でしょう。

ではこのような状況下で仮想ネットワークを議論することは無意味なのでしょうか。

複数のクラスタで構成されるシステムでは特定のクラスタ内通信の品質を維持するため、ネットワークをほかのクラスタから分離したい場合があります。現段階でそれほど多くのテナント契約がなくとも、今後、サービスやアプリケーションによっては1つのテナントが数十のVLANを必要とし、同時に敏速な仮想マシンの追加除去を要求してくる可能性が絶対にないとは言い切れません。また、将来的にまったく別の目的でネットワークを仮想化する必要性が発生するかもしれません。本来の目的では利用されず、当初想定していなかった用途で注目され活用されたという技術はいくつもあります[注1]。

また、仮想ネットワークやSDNを評価検証した結果、まったく別のアイデアやコンセプトが閃く可能性もあります。現状必要なくとも常に新しい技術を理解し把握しておくことはエンジニアにとって大きなチャンスであり、さらに誤った利用や導入の判断をしないためにも技術のメリットとデメリットを明らかにしておくことは重要なことです。

現在提案されている仮想化技術にはブリッジパスツリーを多重化し、ECMP[注2]によってL2スイッチ間帯域パスを柔軟に形成するファブリックモデルと、トンネル技術を用いてオーバーレイネットワークを構成するエンドポイントモデルがあります。ファブリックモデルはおもに新しいプロトコルを実装した物理的ハードウェアによって実現されるため、本来仮想ネットワークとは言い難い側面もあります。しかしECMPによる柔軟なマルチパスを形成することで仮想的に通信チャンネルの動的広帯域化と冗長化を提供するという意味で仮想ネットワークの範疇に入れてもかまわないでしょう。本連載では第1回でファブリックモデル、第2回でエンドポイントモデル、第3回でOpenFlowについてそれ

注1) たとえばMPLS(Multi-Protocol Label Switching)など。
注2) Equal-cost multi-path。SPF(Shortest Path First)に基づき、コスト値が同じ複数経路を探すプロトコル。

仮想ネットワークの落とし穴

ぞれの概要とデメリット、欠陥などについてお話します。

ファブリックモデル

ファブリックモデルはメッシュ状に相互接続したネットワークスイッチやブリッジによって構成され、耐障害性、冗長性と帯域拡張性を同時に実現する技術です(**図1**)。ただし、あくまでも物理スイッチによる実装であり、前述のようにネットワークシステム内の最大VLAN数制限を解決することはできません。また、メッシュ接続された物理スイッチ上に仮想的な多重パスツリーを形成するのでオーバーレイネットワークと言えなくもないですが、既存ネットワーク上に論理的なL2/L3オーバーレイを形成するわけでもありません。しかし、第2回で解説するエンドポイントモデルによる仮想ネットワークも結果的にはスイッチ機器によるアンダーレイネットワーク上に構成されるため、ファブリックネットワークの仮想化技術を知っておくことも必要です。

現在最も有力なプロトコルにはIETFが標準化作業をしたTRILL(Transparent Interconnection of Lots of Links)、IEEEで勧告されているSPB(Shortest Path Bridging)、そしてONFが提案するOpenFlowホップバイホップ形式があります。

OpenFlowホップバイホップ形式については第3回であらためて取り上げるとして、ここではTRILLとSPBについて解説します。

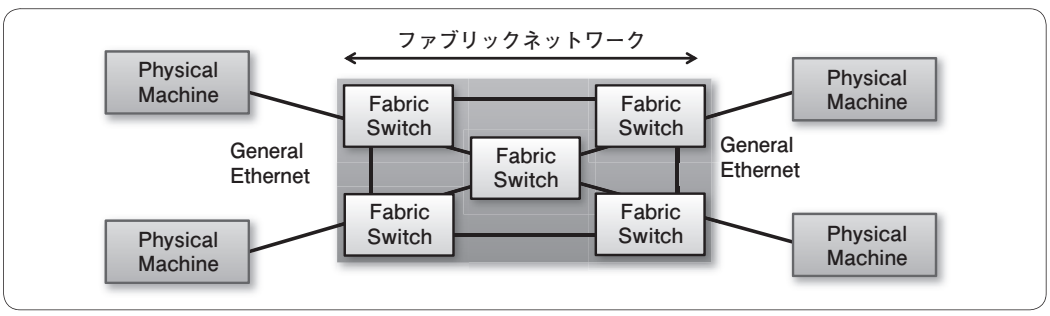
TRILL

TRILLはIETFのRFC5556、6326、6325、6327、6439によって標準化されたプロトコルであり、メッシュ状に接続されたスイッチファブリックにおいてブリッジループの発生を回避しつつ、リンクステート型のルーティングプロトコルによって、最短距離の選定とECMPによる複数経路パスを形成する技術です。基本的にどのスイッチがどのVLANへの到達性を持っているかという情報を経路情報とし、VLAN単位の仮想ネットワークをファブリック内に形成します。RFC6325では当該VLANへの到達性を示す経路情報交換プロトコルとしてIS-ISを用いることを提案していますが、実際の製品では必ずしもIS-ISが用いられているとは限りません。

TRILLプロトコルをサポートし、Ethernetファブリックを構成するスイッチをRBridge(Routing Bridge)と呼びます。各RBridgeは個別のIS-IS IDを持ち、そのIDをブリッジの識別子としてルーティング情報を交換します。同時に2バイトのニックネームを持ち、TRILLシグナリングによる経路表やTRILLヘッダでのRBridge識別子にはIS-IS IDではなくニックネームが使われます。

各RBridgeはどのVLANセグメントへの到達性を持っているかという情報をIS-ISでファブ

▼図1　ファブリックモデル



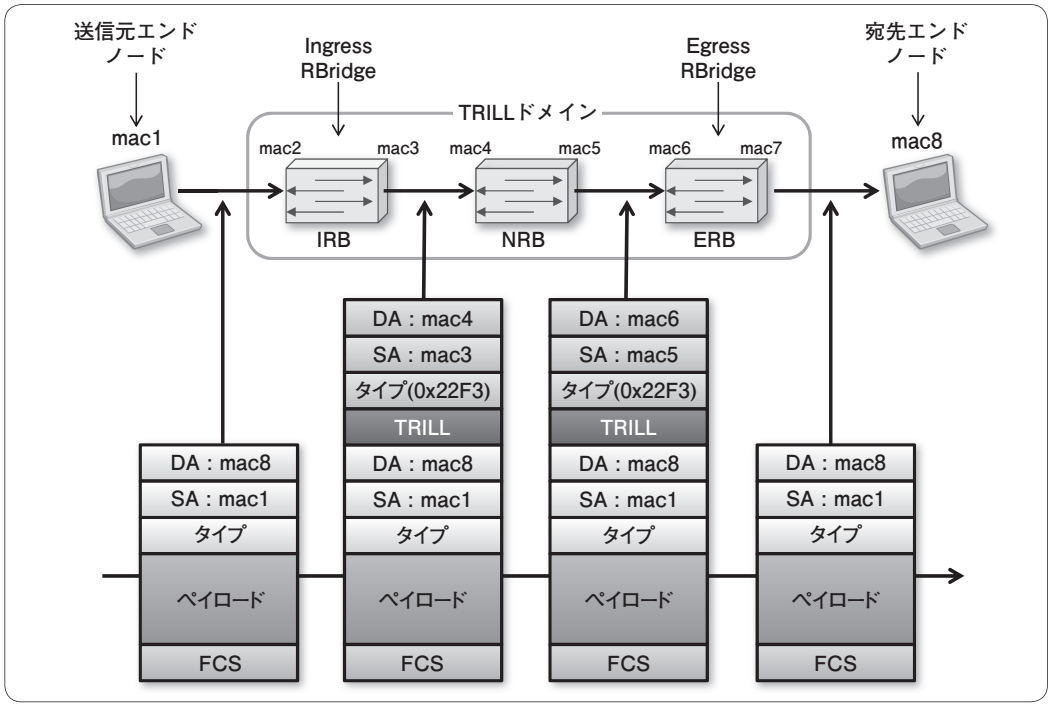
リック側に広告します。次に同じVLANセグメントへの到達性を持つRBridge群の中から代表ブリッジが選出され、その代表ブリッジをルートとするパスツリーを構成します。ARPやNDPのようなブロードキャスト、マルチキャストフレームは、その形成されたVLANツリーに対して配送されます。ノード間ユニキャストフレームは、ファブリックの入り口にあるRBridge(Ingress RBridge)がEthernetフレームにTRILLヘッダを付加します。このTRILLヘッダにはフレームの寿命を示すTTLフィールドがあり、RBridgeを通り抜けるたびにデクリメントされ、途中でTTLが0になったフレームを破棄することでブリッジループを回避しています。次にエンドノードへの直接到達性を持つRBridge(Egress RBridge)に向けてフレームを送出しますが、TRILLフレームの宛先MACアドレスはEgress RBridgeのMACアドレスではなく、個々のRBridgeの経路表にある隣接RBridgeのMACアドレスが利用され、経路途中のRBridgeを通り抜けるたびにこの宛先MACアドレスと送信元MACアドレスが差し替えられます(**図2**)。

したがってTRILLファブリック内でのあるノードからあるノードへの経路は経路途中のRBridgeしだいであり、エッジにあるIngressやEgressスイッチはその途中経路を知りません。逆に言うと、ファブリック内ではIngress/Egressスイッチに依存しない経路を形成することが可能であることからファブリック内ルーティングやECMP化の自由度が高いことを示しています。

互換性の問題

TRILLが持つ欠陥の1つは、すべてのRBridgeがすべてのエンドノードのMACアドレスを学習しなければならないため、MACアドレス学習負荷が大きいことと、個々のRBridgeが持つMACアドレステーブルサイズが肥大化

▼図2　TRILLによる転送



仮想ネットワークの落とし穴

するという問題です。1つのTRILLファブリックに膨大なノードが接続された場合、RBridgeのアドレステーブルが溢れ、テーブルからMACアドレスが除外されたり新しいMACアドレスが登録されなかったりといった問題が起こる可能性が高くなります。すると、通信の途中でRBridgeが該当ノードへの方向がわからなくなり、フレームがブラックホール[注3]へ吸い込まれる恐れがあるのです。

そのため、大規模なファブリックへ導入する場合、想定される最大ノード数のMACアドレスを十分格納できるテーブルサイズのモデルを選択する必要があります。

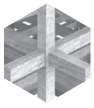
もう1つの欠陥とは前方互換性がまったくないことです。図2で示したようにTRILLによって運ばれるEthernetフレームにはTRILLヘッダとアウターEthernetヘッダが追加されますが、このTRILLヘッダはまったく新しく定義されたヘッダフォーマットであり、既存のスイッチ機器がこの新しいヘッダを認識することはできません。つまり、TRILLによってファブリックを形成する範囲のスイッチはまったく新しく設計し製造されたTRILLスイッチだけであり、既存のスイッチが途中に介在したりすることはできないのです。また、標準化よりも実装のほうが早く、ベンダによってはIS-IS以外のプロトコルを使ったり、IS-ISのTLVレコードを独自に拡張していたりもするため、ベンダ間での相互接続性もまったく期待できません。

将来的に標準化によってベンダ間の相互接続性が実現されるとしても、現状の実装を変更せざるを得ないため後方互換性の保証も怪しくなります。これはIETFの標準化作業が実装者によるボトムアップ形式の提案型であり、同じ目的のためにいくつもの標準や実装が生まれやすく、前後方互換性にそれほど神経質ではないことが原因なのかもしれません。

また、キャリアの広域Ethernetサービスなどを介して遠隔データセンターとファブリックを形成する場合も、その広域網内のすべてのスイッチがTRILLをサポートしていない限り、直接TRILLによるファブリックを遠隔地に拡張することはできません。キャリア網内のMAC-in-MAC形式でトンネリングされるのであればTRILLフレームは遠隔地に到達しますが、その場合、TRILLフレーム長はキャリアが提供可能なMTUサイズ以下である必要があります。現在、各ベンダが独自にこういった非互換性を吸収し、既存ネットワークや広域Ethernetサービスを介してTRILLファブリックを拡張するゲートウェイ製品を提供してはいますが、これもベンダ独自の実装であるためベンダ間の互換性はまったくありません。

このようにTRILLにおける致命的欠陥とは、ベンダ間と前方後方に対する非互換性です。現状のスイッチネットワークをすべて同じベンダのTRILLスイッチに置き換えるということは投資的観点から非現実的であり、またベンダロックインを確定してしまう恐れもあります。しかも既存機器とのファブリック的連携も不可能であり、利用できる製品寿命が短い(可能性がある)ことなどを考慮すると、導入には相当な勇気を必要とするでしょう。

SPB



SPB(Shortest Path Bridging)はIEEEによって標準化が進められているファブリックプロトコルであり、2006年頃に最初のドラフトが提示されました。SPBはSTP(IEEE802.1d)、RSTP(IEEE802.1w)、およびMSTP(IEEE802.1s)の後継プロトコルとして位置付けられており、前項で解説したTRILLとよく比較される技術です。基本的にはEgressブリッジをルートブリッジとするパスツリーを形成し、ループを排除すると供にブロックリンクの低減を実現しつつECMPも可能としています。

L2スイッチ、すなわちマルチポートブリッジ

注3) 網内で行き場を失ったフレームはその時点で破棄されるためこのように言われます。

によってポート障害やスイッチ障害による問題を回避するため、接続リンクを多重化してファブリックを形成するとファブリック内にブリッジループが発生します。EthernetヘッダはTTLなどのフレーム寿命を示すパラメータを持たないため、ブリッジループが発生するとブロードキャストやマルチキャストフレームなどは永久にループ内を巡回しつつコピーされ増殖してしまうのでネットワークにパケットストームが起こり、場合によってはネットワークが全落ちします。

このブリッジループ発生を回避する伝統的な技術がSTP(Spaning Tree Protocol)です。STPはファブリック内で1つのルートブリッジを決定し、各ノードはルートブリッジからの最短距離を順番に計算してパスツリーを形成し、選択されなかったリンクを論理的に遮断してループの発生を防ぐという方法です。STPでは各ブリッジから送られてくるBPDU(Bridge Protocol Data Unit)によってツリーの形成や死活確認を行っていますが、隣接ブリッジからのBPDUがある一定時間届かなくなるとそのブリッジがダウンしたと認識し、ツリーの再構成を行います。したがってツリー内のブリッジが障害を起こした時点から、BPDUを待つタイムアウトとツリーを再構成している間、ファブリックは非常に不安定な状態になります。この問題を解決するため、隣接ブリッジの死活監視をタイムアウトで行うのではなく、提案と合意というメッセージを交換することによって行うプロトコルがRSTP(Rapid STP)です。このメッセージ交換間隔をSTPのタイムアウトよりも短くすることで障害検知から復旧までの時間を短縮しているのです。

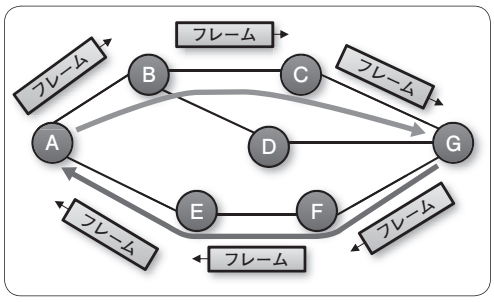
しかし、STPもRSTPもファブリック内で選出されたルートブリッジを頂点とする単一のパスツリーしか形成しないため、そのパスが常に各ブリッジ間の最短パスとは限らず、またループを回避するために無効化されたポート(ブロックポート)も多く、性能、耐障害性、コストパフォーマンスにおいて効率的ではありません。

そこで、ファブリック内にある複数のVLANによってリージョンを定義し、そのリージョン単位でパスツリーを形成することでポートの利用率を上げ、コストパフォーマンスを向上させたプロトコルがMSTP(Multiple STP)です。しかし、MSTPもリージョン単位でルートブリッジを選出し、そのルートブリッジを頂点とするパスツリーによってブロードキャストセグメントを構成することから、ファブリック全体の利用効率を向上させるとはいえ、リージョン内のパスツリーが常に各ブリッジに対して最適化されるているというわけではありません。そこで、新たに開発されたプロトコルがSPBです。

SPBはファブリックのエッジにあるIngress/Egressブリッジをルートブリッジとしたパスツリーを個々に形成します。したがって1組のエッジブリッジ間のパスは常に最短パスを選択することとなり、STP、RSTP、MSTPにおけるパフォーマンスの問題を解決しました。しかし、すべてのエッジブリッジをルートとするパスツリーを個別に形成すると、エッジ間の経路が行きと戻りで異なるパスを選択してしまう可能性があります。**図3**で示すようにブリッジAからBへのパスはA-B-C-Gが選択され、ブリッジBからAへのパスにB-F-E-Aが選択されたとすると、Aから送出されたブロードキャスフレームはBで折り返してAまで戻ってきてしまうのでブロードキャストストームが発生します。

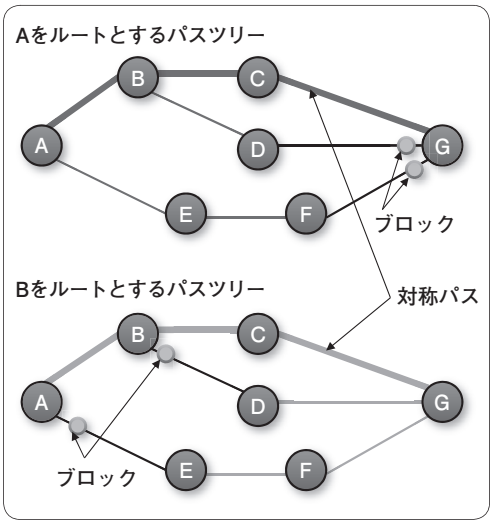
そのため、SPBでは**図4**で示すように常に対向するエッジブリッジ間のパスには同じ選択ポリシーを用いてパスを対称化し、ループが発生

▼図3　非対称パスにおけるループ



仮想ネットワークの落とし穴

▼図4　対称型パスツリー



することを回避しているのです。

MACアドレス学習問題

SPBにはオリジナルのEthernetフレームをIEEE802.1adに準拠したVLANタグでカプセリングするQ-in-Q方式（SPBV）と、IEEE802.1ahに従うアウターEthernetでカプセリングするM-in-M方式（SPBM）があります。Q-in-Q方式ではTRILLと同じようにファブリック内のすべてのブリッジがエンドノードのMACアドレスを学習する必要があり、TRILLと同じくアドレステーブルが枯渇するという危険性を秘めています。M-in-M方式の場合、エンドノードのMACアドレスを学習するのはエッジブリッジだけであり、ファブリック内のブリッジはSPBブリッジのMACアドレスだけを学習すればよく、アドレステーブルの枯渇という問題は発生しづらくなっています。そのため、どちらかというとSPBVはLAN内での利用が多く、SPBMは広域Ethernetを運用するキャリア・ネットワークへの適用が多くなっています（**図5**）。

しかし、クラウド環境においてはこのMACアドレス学習がSPBVとSPBMに共通する致命的欠陥になります。クラウド環境では物理サーバ上にいくつもの仮想マシン（以後VM）が起動されており、そのVMは頻繁にアップダウンを繰り返します。また、場合によってはVMが物理マシン上を移動する可能性もあります。TRILLの場合、VMやエンドノードが新たに追加されたり、直接接続するエッジRBridgeを移動したりした場合、そのエンドノードが新たなRBridgeの到達性範囲にポップしたことをESDAI（End Station Destination Address Information）メッセージによってパスツリー全体にブロードキャストします。したがってエンドノードのロケーション移動や追加に伴うMACアドレス学習を一瞬で完了することができます。

しかし、SPBによって構築されたファブリック網では基本的にエンドノードのMACアドレス学習をIEEE802.1Dに準拠した学習で行うため、各ブリッジ（SPBMの場合はエッジブリッジ）はエンドノードからのEthernetフレームが自身を通り抜けない限りそのノードのアドレスを学習することはできません。したがって、あるエッジブリッジに接続されているエンドノード上のVMが別のエッジブリッジに接続されているエンドノードにマイグレーションされた場合、そのVMへのEgressブリッジが変更したという情報がファブリック全体に伝播するにはある程度の時間を要します。その間、古いロケーション情報を持つブリッジにそのVM宛のフレームが届くと、テーブル情報と実際のロケーションが異なるためフレームはブラックホールへ吸い込まれることになります。

つまりSPBは高い柔軟性と弾力性が要求されるクラウド・コンピューティング基盤には不向きなファブリック技術なのです。ほとんどVMがマイグレーションせず、また追加や除去の少ないクラウド環境であれば有効ですが、そもそもそのような環境をクラウド化する必要もなく、クラウド化されていない環境にイーサネットファブリックのような仮想ネットワークは必要ありません。結果的にSPBはデータセンターやLAN上で運用されるクラウド基盤向けのファブリッ

ク技術ではなく、長期契約によって仮想回線を提供する広域Ethernetサービス用に限定される技術ということになります。

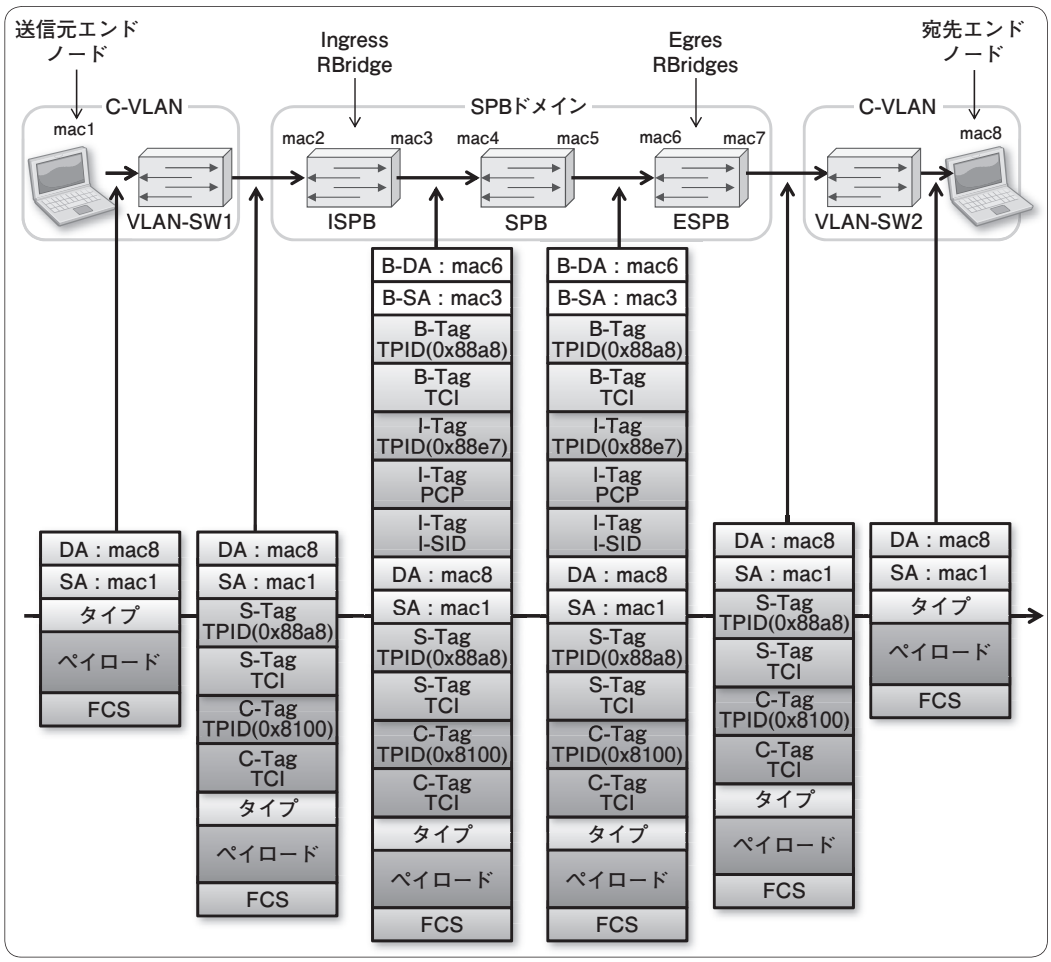
おわりに

TRILLやSPBのようなL2技術については古くからIEEEのドキュメントに親しんでいる人以外には中々把握しにくい内容かもしれません。本稿では誌面の関係上、詳細説明は割愛しましたが、もし興味があれば拙書『SDN/OpenFlowで進化する仮想ネットワーク入門』(インプレスR＆D)に詳しく書いてあるのでそちらを参照してください。

一般的に新しい技術を論じる場合、その技術や実装を提供する側が主導権を握る傾向があるためそのメリットについての議論に多くの時間とリソースを割き、デメリットについてはわずかに触れるだけにとどまることが多いでしょう。しかし、その技術や実装を導入する立場にとってはメリットよりもデメリットのほうが重要なのです。なぜならば筆者の経験上、デメリットを知らずして、もしくは目を背けたまま導入してしまうと高い確率で甚大な被害と尽きることのない大きな責任を背負うことになります。逆にデメリットを十分理解把握したうえで、メリットの部分を活かそうという姿勢での導入であれば、思いもかけぬ障害や困難に窮することが避けられるでしょう。

次回はVXLAN、NVGRE、そしてSTTに代表されるエンドポイントモデルの仮想ネットワークについて、その致命的欠陥をお話します。SD

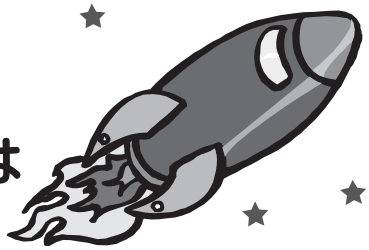
▼図5　SPBMによるフレーム転送



4

Debian Developer　やまねひでき　henrich@debian.org

ソフトウェアをDebian公式リポジトリに入れるには



Debian Hot Topics

はじめに

そろそろDebian7.0「Wheezy」のリリースが……と前回も書きましたが、ようやく5月4日（あるいは5日）になるとの発表がありました。残念ながら本稿にはギリギリ間に合わないため、7.0の特徴や新機能などの記事は次回にまわすことにして、今回は読者からリクエストがあった話題をとりあげてみたいと思います。

公式パッケージへのいざない

さて、この原稿と前後してリリースされているはずのDebian 7.0では6.0と比較して約12,800個のパッケージが新規に利用できるようになっています。しかし、自分が開発している（あるいは使っている）ソフトが、Debianの公式リポジトリにはまだ入っていない……というちょっと残念な状況は、これほど豊富なパッケージ数を誇るDebianであってもままあります。そんなときにはソースから入れておしまい、ではなくて「使っているソフトを開発者にお願いして公式パッケージとして取り込んでもらう」ということを考えてみるのはいかがでしょうか。

公式パッケージになることのメリット

「公式パッケージとして取り込んでもらう」のは大変そう……ソースから入れるほうが気楽で良いんじゃないの？という疑問を持たれる方もいることでしょう。しかし、公式パッケージになると、面倒な苦労を上回るさまざまなメリットがあるのです。ざっと挙げてみましょう。

- 依存している必要なパッケージも漏れなく一発でインストール完了
- 各所に存在するリポジトリミラーに配布されるので、インターネットにつながりさえすればどの環境でも、コマンド一発でインストール可能
- アップデートが楽（ソースで入れるとライブラリの不整合が出るなど、アップデートが煩雑になりがち）
- 自分以外の多くの人も使うため、広範囲なテストが行われ、問題点が洗い出される
- ほかの人が修正パッチを書いてくれる可能性が出てくる。セキュリティ問題も基本的に考慮してくれる

公式パッケージになるということは、世界各地に存在するDebianミラーにパッケージが用意されます。いちいちソースのバージョンを確認しつつ導入したり、導入手順書を延々と書いたり、オレオレパッケージ用のリポジトリを作成する必要はありません。「○×パッケージをインストール」ということだけを書けば良いので、何より簡潔でわかりやすくなります。しかも作業担当者は「楽」ができるわけです。

また、「公式パッケージになる」ということを「さまざまな環境／アーキテクチャ／利用方法でのテスト人員を無償で確保できる」と考えて

みてはどうでしょう？　これを費用を支払って実現しようとすると、膨大な金額になってしまうのは容易に想像できるかと思います。かなりコストパフォーマンスが良いと思いませんか？

さらに自分が手動で入れたソフトウェアは修正も自分で行わなければいけませんが、Debian公式パッケージであればセキュリティ修正を含む重大な問題はディストリビューション側がケアしますので、リスクを減らせます。

まとめると、

- 楽ができて
- コストメリットがあり
- リスクを減らせる

と良いことずくめなのです。いかがでしょうか、手間をかける価値があるとは思いませんか？では、その気になった、ということで次へ進んで具体的な手順を確認してみましょう。

取り込んでほしいソフトウェアを提案する際のお作法

単に「取り込んでー」「オッケー」というやりとりで済むほど、Debianの状況はシンプルでコンパクトなものではありません。若干の形式に従ったやりとり（プロトコル）を理解する必要があります。とはいえ、この短い記事を読み終えるころにはだいたい理解できるぐらいのものですので、肩の力を抜いていただいて結構です。流れとしては「事前調査」→「RFP/ITP[注1]の登録」→「パッケージ作成」→「アップロード」の4つに分かれます。では、それぞれのフェーズについて説明していきましょう。

事前調査をしよう

確認が必要なことは3つです。

- そのソフトウェアはDebianで配布可能なライセンスなのか？
- 開発元（upstream）は「非協力的」ではないか？
- 興味を持ってくれそうなDebian開発者はいるか？

★そのソフトウェアはDebianで配布可能なライセンスなのか？

まず、確認すべきことは「そのソフトウェアのライセンスはDFSG（Debian Free Software Guidelines）[注2]に従っているものか？」ということです。難しいことのように聞こえますが、DFSGは「オープンソースの定義」[注3]のもとになったものですので「≒そのソフトウェアがオープンソースソフトウェアであるか？」と考えていただければだいたい間違いがありません[注4]。

Debianのリポジトリに入るソフトウェアは3種類に分類されます。1つめはDFSGに準拠している（DFSG-freeと称します）ソフトウェア。これはDebianの「main」コンポーネントとして配布されます。

2つめは、ソフトウェア自体はDFSGに従っているものの、動作にDFSG-freeではない別のソフトウェアが必要になるもの。「Contrib」コンポーネントとして分けて配布する形になります[注5]。

そして最後に、ソフトウェアをそのまま再配布することは可能だが、変更して再配布が行え

注1）URL http://www.debian.org/devel/wnpp/ 参照。それぞれ「Request For Package」「Intent To Package」の略。RFPは「こんなソフトウェアがあるから誰かパッケージにしてくれないかなー（チラッチラッ」、ITPは「このソフトウェアを私がパッケージにします！宣言」だと思ってください。

注2）URL http://www.debian.org/social_contract#guidelines

注3）URL http://www.opensource.gr.jp/osd/osd-japanese.html　参照。オープンソースソフトウェア（OSS）はそもそもが意図的に作られたマーケティング用語であり、きちんとした定義があります。「OSSってソースがネットで公開されていて、タダで手に入るソフトウェアのことだよね？」などと言っている読者の方はいらっしゃいませんよね？

注4）なぜ「＝」ではないか、というとOpen Source InitiativeがOSSと認定していてもDebianではDFSG-freeではない、と考えられているライセンス（たとえば、CDDL。OpenSolarisなどに使われていたライセンス）があるからです。

注5）たとえば、flashplugin-nonfreeパッケージはソフトウェア自体はDFSG-freeですが、DFSGに合致しないプロプライエタリなAdobe Flashに依存するので、contribコンポーネントに収録されています（名前と若干ギャップがありますが……その点はスルーで）。contribに収録されているものはデフォルトでは利用できないのでユーザが設定を有効にする必要があります。

ないなどの制限があるもの。これは「non-free」コンポーネントに収録されます[注6]。

そして、上記の3区分以外のソフトウェアはDebianでは配布していませんし、残念ながら配布自体もできません[注7]。このライセンスに関する判断は煩雑ですので、**図1**に簡単なフローチャートを挙げました。参考にしてください。

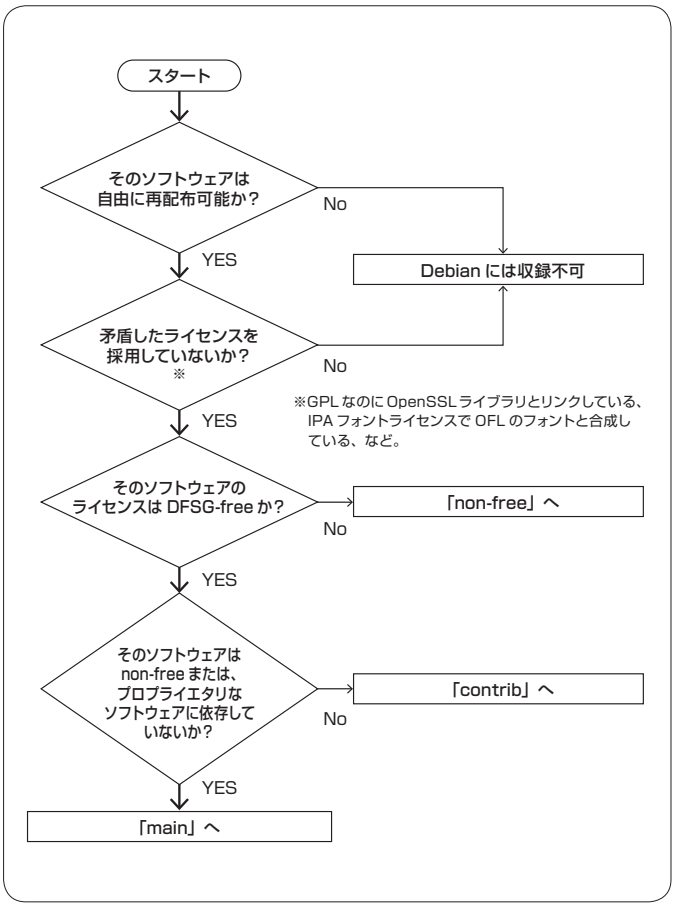
とはいえ、ライセンス周りは複雑怪奇なことがあり、なかなかわかりづらいので、判断に迷うことがあれば1人で悩まずにメーリングリスト[注8]で尋ねてみることをお勧めします。

★開発元(upstream)は非協力的ではないか？

読者の中には「協力的かどうか」じゃないの？と思った方もいらっしゃることでしょう。非協力的というのは、ビルドエラーの無視とか、Debianがらみで送ったパッチを意途的に無視とか、わざわざDebianにだけ発現するトラップをしかける[注9]などの行為がそれにあたります。

DebianではIntel/AMD以外のマイナーなCPUをサポートしていたり、カーネルもFreeBSDカーネルをeglibc環境に持ってきて使っていたりなど、upstreamの開発者が想定している「一般的ではない」環境でビルドと使用を行うことがあります。こんな場合にバグ報告などをupstreamに送って協力を仰いだりするわけですが、ごくたまに「そんな環境知らねぇし！」と協力をまったく拒否するような人もいるわけです[注10]。こうなってしまうとDebianとしてはメンテナンスするリスクとコストが高いと判断し、基本的には「何とかしてそのソフトウェアは使わない」方向に向かってしまいます。そんなソフトウェアをパッケージにしてもしかたがないので、こ

▼図1　Debianへの収録におけるライセンス確認の簡易フローチャート



注6）ちなみにnon-freeに収録されているソフトウェアは「Debianの一部ではない」というのが公式見解です。このあたりはあまり手をかけずにプロプライエタリなビデオドライバが入っているrestrictedリポジトリが使えたり、Canonicalという企業がバックについてpartnerリポジトリを持っていたりするUbuntuと比較されるところです。楽に使えるのが一番ではあるのですが、Debianの目的は「フリーなOSを開発すること」ですので、この辺は「目的と姿勢の違い」と理解してください。

注7）一部のプロプライエタリなソフトウェア(Adobe Flash、Oracle Javaなど)は再配布が許可されてないので「無理」です。

注8）debian-devel@lists.debian.or.jp

注9）昔、Debianのパッケージメンテナと険悪になった開発者が、メンテナの開発環境では発現しないよう細工を施し、ほかの環境では「このパッケージ使うよりうちから直接ダウンロードした方が良いぜ！」と表示されるようにトラップをしかけたことが過去に1回だけあったのです……。

注10）有名どころではglibc開発者だったUlrich Drepper氏がバグ報告に対して「このクソなアーキテクチャ以外ではちゃんと動く。ARMのためだけにパフォーマンスを犠牲にするなんてことはしないね」「自分に給料を払ってるわけでもないのに命令するな」などの素敵な語録を残し、Debianがglibcからeglibcに移行するきっかけを作ってくれました。

▼図2　RFPのフォーマット

```
To: submit@bugs.debian.org
Subject: RFP: birdfont -- TTF, EOT & SVN font editor  ←わかりやすくRFP/ITPと先頭に付ける

Package: wnpp  ←必ずwnppにする
Severity: wishlist  ←重要度は「要望(wishlist)」
X-Debbugs-CC: debian-devel@lists.debian.org, pkg-fonts-devel@lists.alioth.debian.org
                                                    ↑登録と同時に同報する宛先
※ここまでがBTSの既定フォーマット。以下がRFP/ITPの既定フォーマット
   Package name: birdfont  ←パッケージ名
   Version: 0.8.0  ←バージョン
   Upstream Author: 2012 Johan Mattsson <johan.mattsson.m@gmail.com>  ←作者
   URL: http://birdfont.org/  ←入手先
   License: GPL-3  ←主なライセンス

    Description: TTF, EOT & SVN font editor  ←説明
  Birdfont is a free, open source font editor that lets you create outline
  vector graphics and export ttf, eot & svg fonts.
```

の点はクリアしておく必要があります。

★興味を持ってくれそうなDebian開発者はいるか？

そして、興味を持ってくれそうなDebian開発者を見つけること、が重要になります。なぜかというと「リポジトリへパッケージを新規にアップロードできる権限を持っているのは公式Debian開発者だけ」だからです[注11]。

しかし、実際のところ、Debian開発者はメールベースでの処理を大量に行っているので、お作法に従ってRFP登録したとしても、アピール下手な場合は残念ながらメールの山に埋もれて検討すらしてもらえない可能性もあります。方法は問いませんので「パッケージにしてメンテしてくれそうな人を見つけて直接アタックしてみる」[注12]か「そのソフトウェアが有用であることをDebian JPのメーリングリストで発信する」などを行いましょう。

RFP/ITPの登録をしよう

RFPは**図2**のようなフォーマットになります(ITPも同様です)。

To:とかSubject:ってメールみたいだな？と思われる方がいるかもしれません。はい、メールです。実はDebian BTSは世にも珍しい「メールベースのバグトラッキングシステム」ですので[注13]、submit@bugs.debian.org宛にフォーマットに従ったメールを送ることで、パッケージ作成依頼の登録が行えます。これで「RFP/ITPの登録」は完了です。そして、この後「パッケージ作成」→「アップロード」となります……が、ページ数も尽きてしまうので、この説明は別の機会に譲りましょう。

最後に

この記事を読んでくれた方で、パッケージにしたほうが良さそうだな、というソフトウェアが思い当たりましたら、ぜひメーリングリストなどでご相談／情報共有いただけるとうれしく思います[注14]。Debian(とその派生ディストリビューション)を一緒に改善していきましょう:-) SD

注11) この理由はパッケージのインストールはroot権限で行われる(つまりパッケージのメンテナスクリプトに命を預けている)、という点から考えると理解しやすいかと思います。考えたくはないですが、悪意を持ったパッケージが気軽に公式リポジトリに登録できる状態ですと、いったん被害が起きると甚大なことになります。

注12) メール／Twitter／勉強会などのイベントで直接、というのが考えられます。

注13) バグ報告と操作記録はhttp://bugs.debian.orgで「参照」できます。が、操作自体はすべてメールベースで実施します。一応「reportbug」というツールも準備されていますが、結局のところSMTPを使って処理を行います。

注14) もちろん、定期的に開催されているイベント(http://www.debian.or.jp/community/events/)などで尋ねてみるのもありです。イベント情報はTwitter：@debianjpなどを参照ください。

第38回

Ubuntu Monthly Report

Ubuntu Touch

先日スペインのバルセロナで行われたMWC2013 で発表されたUbuntu Touch は大きな注目を集めました。今回は開発者向けイメージのインストール方法と開発環境について紹介します。

長南 浩　chonan@progdence.co.jp

第3のモバイルOSを狙うUbuntu Touch

2013年2月、スマートフォンとタブレット向けの新OSであるUbuntu Touchの開発者向けプレビュー版[注1]がMWC2013に先駆けて公開されました(**図1**)。今日UbuntuはLinuxディストリビューションとして有名になりましたが、その一方でテレビ向けのUbuntu TV[注2]やクレードルに接続するとUbuntuのデスクトップ環境が使えるUbuntu for Android[注3]など、さまざまな分野での取り組みがCanonicalを中心に進められています。今回のUbuntu Touchはスマートフォンとタブレット向けにiOS、Androidに続く、第3のモバイルOSを目指す野心的な試みといえます。

現在、公式にサポートされているデバイスは、Google が開発者向けに販売しているデバイスのうち、Nexus 4、Nexus 7、Nexus 10、Galaxy Nexusの4機種です。これ以外の機種への移植や動作報告などはXDA Developers[注4]などの開発者フォーラムで議論や報告が行われているようです。

注1) https://wiki.ubuntu.com/Touch
注2) http://www.ubuntu.com/devices/tv
注3) http://www.ubuntu.com/devices/android

図1　Nexus 10でのUbuntu Touchのホーム画面

実デバイスへのインストール

インストール方法については、公式Wiki[注5]のページに詳しく掲載されていますが、デバイスのブートローダをアンロックしたうえで、Ubuntuが動作するPC上からOSイメージを書き込む方法でインストールします。事前に、**図2**のようにツールをインストールし、ブートローダをアンロックして開発者モードが有効となったデバイスを、USBケーブルで接続し、

```
$ sudo phablet-flash -b -l
```

とすると、最新のOSイメージをダウンロードした後にデバイスに書き込みを行います。OSのイ

注4) http://www.xda-developers.com/
注5) https://developers.google.com/android/nexus/images

メージは500MB以上あるため、高速なインターネット環境が必要です。デバイスへのOS書き込みにはけっこう時間がかかるため、途中でハングアップしたかのように見えますが、終了するまで気長に待ってください。

書き込みが終了すると、デバイスが再起動され、Ubuntu Touchが起動します。なお、現時点ではUbuntu Touchのイメージは毎日更新されており、phablet-flashコマンドの「-l」オプションで明示的に最新のファイルがダウンロードされます。

また、一度Ubuntu Touchをインストールしたデバイスをもとのに戻すには、Googleが公開している工場出荷時のイメージ[注6]を入手して書き戻す方法があります。現状のUbuntu Touchはあくまで開発者向けという位置づけですので、事前にリカバリ手順を確認したうえでインストール作業を行ってください。とくにNexusデバイス以外のものについては、デバイスがまったく動作しない状況（文鎮化）となる可能性があるので、サポートされているデバイスにインストールすることを強く推奨します。

開発環境の構築

そんなUbuntu Touchですが、アルファ版ながら、アプリケーション開発のためのUbuntu SDKが公開されています。インストールはUbuntu 12.10、12.04LTSの場合は**図3**のように、またUbuntu 13.04 の場合は**図4**の一連の操作でインストールできます。また、開発環境としてQt Creator（**図5**）を使いますが、上記の操作中に自動的にインストールされます。

PCにデバイスを接続した状態でQt Creatorを使うと、デバイス上のパッケージをapt-getで管理したり、sshでデバイスへログインするといった操作をTools→Ubuntu→Deviceメニューから行うことができます。現状ではphablet-flashコマンドでインストールされるイメージには、日本語のフォントは入っていないので「fonts-takao*」をapt-get→installからインストールしておくといいでしょう。ただしこの際にデバイスはWifi経由でパッケージファイルを取得しようとするので、事前にWifiの設定を済ませておく必要があります。また、アプリケーションを開発し、デバイスにインストールするのであれば、Developer Connectionを有効にしておきましょう（**図6**）。

Ubuntu Touch アプリの特徴

Ubuntu Touchアプリケーションは Qt Quick フ

図2　事前にツールをインストールする

```
$ sudo add-apt-repository ppa:phablet-team/tools
(PPAインストールのための確認が行われます)
$ sudo apt-get update
$ sudo apt-get install phablet-tools android-
tools-adb android-tools-fastboot
```

図3　Ubuntu 12.10、12.04LTSをインストールする場合

```
$ sudo add-apt-repository ppa:canonical-qt5-
edgers/qt5-proper
$ sudo add-apt-repository ppa:ubuntu-sdk-team/ppa
$ sudo apt-get update
$ sudo apt-get install ubuntu-sdk notepad-qml
```

図4　Ubuntu 13.04をインストールする場合

```
$ sudo add-apt-repository ppa:ubuntu-sdk-team/ppa
$ sudo apt-get update
$ sudo apt-get install ubuntu-sdk
```

図5　Qt Creator

Welcome画面にはUbuntu Touchでの開発ドキュメントへのリンクが表示される

注6）https://developers.google.com/android/nexus/images

レームワーク上で動作し、QMLというUIに特化した言語で記述されます。Windowsストアアプリ開発や、WPF(Windows Presentation Foundation)でのXAML(Extensible Application Markup Language)、Androidでのレイアウト XMLに相当するものですが、JSONのような文法で比較的単純な構造になっています。開発環境もLinux環境にありがちなソースファイルをエディタで編集してコンパイラやインタプリタに処理させるのではなく、Qt CreatorをIDEとした、よりモダンなスタイルでの開発を行うことができます。Visual Studioほど万能ではないのです

図8　サンプルアプリケーションをQt Creatorで実行したところ

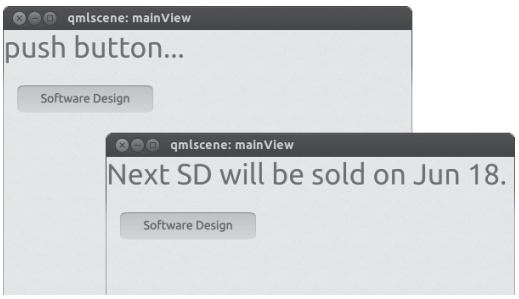


図6　Qt Creatorからデバイスにapt-getコマンドを発行できる

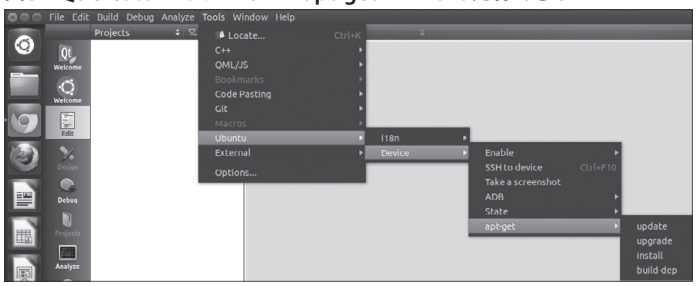
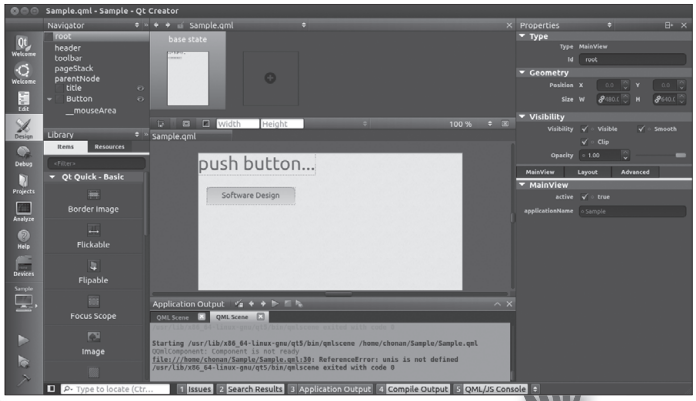


図7　Qt CreatorでのUIデザイン



が、ボタンやテキストボックスなどの配置をGUIで調整することもできます(**図7**)。

QMLのコードはUIを記述するだけではなく、ボタンが押されたときの処理などをJavaScriptで記述できます。一般的にある程度規模が大きいアプリケーションになると、UIとその処理を分離することが求められますが、その場合にはJavaScriptの関数を別ファイルに用意して、QMLファイルの冒頭で適宜importするスタイルとなります。ちょっと乱暴な言い方かもしれませんが、AndroidでのDalvik仮想マシンの代わりにUbuntu TouchではQt Quickフレームワークを使用し、JavaScriptとHTML5でアプリを開発するのが定番開発パターンとなりそうです。

サンプルアプリケーション

サンプルとして、ごくごく単純なアプリケーションを書いてみました(**リスト1**)。ボタンを押すと、次の本誌の発売日を表示させるというものです。MainView上にラベルとボタンを配置して、ボタンが押されたらラベルの文字列を変化させるというアプリケーションです(**図8**)。

ボタンが押された場合の処理はJavsScriptを埋め込んでいますが、それを含めて60行程度の分量です。iOSやAndroidでのアプリ開発に比べて簡単に記述できることがわかります。JavaScriptを使うという特性をうまく利用すると、スムーズに開発を進めることができそうです。とくにHTML5と関連テクノロジを追いかけている開発者は比較的簡単にUbuntu Touchアプリを書くことができるでしょう。

掲載したサンプルアプリはQt Creator経由でデバイスにインストールして動作させることができます。

おわりに

今回はUbuntu Touchのインストールと開発環境を駆け足で紹介しました。モバイルOSについてはUbuntu Touch以外にもFirefox OSやTizenなどの新しい製品が名乗りを上げるなどMobile World Congress 2013前後から、にわかに活気がでてきました。

iOSやAndroidの2強が支配するモバイルOS分野で、Ubuntu Touchがどれだけの地位を築くことができるのかは今後も注目しておきたいところです。SD

リスト1　サンプルアプリケーション

```
import QtQuick 2.0                          // Qt Quick v2 フレームワークを使用
import Ubuntu.Components 0.1               // Ubuntu Touch 用の部品をインポート

MainView {                                  // MainView の中に Label と Button をそれぞれ1個配置
    objectName: "mainView"                  // ビューの名前を設定
    applicationName: "Sample"               // アプリケーション名を設定
    id: root                                // ビューのid を設定

    width: units.gu(60)                     // ビューの幅と高さを設定
    height: units.gu(80)                    // units.gu() を使うことで解像度の違いを吸収できる

    property real margins: units.gu(2)      // マージンとボタン幅のサイズを設定
    property real buttonWidth: units.gu(9)

    Label {                                 // 上部の文字列ラベルの設定
       id: title                            // ラベルのid。title.text などとして参照・変更される
       ItemStyle.class: "title"             // ラベルのスタイルを設定
       text: "push button..."               // ラベルの文字列
       height: contentHeight + root.margins
       anchors {                            // 上面、左右面を親要素(=mainView)に合わせる
           left: parent.left
           right: parent.right
           top: parent.top
       }
    }
    Button {                                // ボタンの設定
        x: units.gu(2)                      // ボタンの表示位置を x: y: で設定
        y: units.gu(8)
        width: units.gu(20)                 // ボタンの大きさ(幅・高さ)を設定
        height: unis.gu(4)
        text: "Software Design"             // ボタンの文字列
        onClicked: {                        // ボタンがクリックされたときの処理(Javascript埋込)
            // 月の名前を配列に格納
            var monthName = [
                        "Jan","Feb","Mar","Apr",
                        "May","Jun","Jul","Aug",
                        "Sep","Oct","Nov","Dec"];
            // 現在の日付を格納
            var currentDate = new Date();
            // 表示月は後ほどセットするが、あらかじめ宣言しておく
            var nextDate;
            if ( currentDate.getDate() <= 18 ) {
                // 現在の日付が「18日以前」の場合には当月を表示月とする
                nextDate = currentDate; // 日は考慮しないので現在の日付をそのまま採用する
            } else {
                //「19日以降」の場合には翌月を表示月としたいのだが...
                if (currentDate.getMonth() == 12) {
                    // 当月が12月の場合は「翌年の1月」を翌月として表示月にセット
                    nextDate = new Date( currentDate.getFullYear() + 1,
                                0,      // 月は0(1月)～11(12月)で指定するので、0(1月)を設定
                                1 );    // Dateコンストラクタは日まで指定するので1日を設定
                } else {
                    // それ以外の場合には、「同年の1月後」を表示月にセット
                    nextDate = new Date( currentDate.getFullYear(),
                                currentDate.getMonth() + 1,
                                1 );
                }
            }
            // 表示月と月名配列をもとに title ラベルの文字列を更新
            title.text  = "Next SD will be sold on ";
            title.text += monthName[currentDate.getMonth()] + " 18.";
        }
    }
}
```

第9回

オープンソースが本当に当たり前になるとき

小島 克俊
KOJIMA Katstoshi
レッドハット㈱グローバルサービス本部
プラットフォームソリューション統括部
ソリューションアーキテクト

はじめに

レッドハットの小島克俊です。プラットフォーム製品のソリューションアーキテクトです。レッドハットの製品とサービスについてお客様やパートナー企業へ技術面からそれらの利点を伝えています。かつてUNIXサーバの販売量で世界一だった日本企業に所属していました。今回はそのころの話をおもにします。UNIXとかクラウドなどについてです。

きみはUNIXをわかっていない

あなたはUNIXをご存じでしょうか。

UNIXとLinuxの違いをPOSIXを引き合いに出してうまく説明できたり、Mac OS XはUNIXだとか知っていれば、ずいぶんなUNIX好きだと思います。そんなあなたがUNIXに習熟したエンジニアだったとして「きみはUNIXがわかっていない」と言われたらどう感じるでしょうか。

UNIXという言葉が非常にミステリアスな何かすごい技術として日本でも知られるようになったのはサン・マイクロシステムズ社(以降、サン社)がUNIX製品を販売しはじめたときです。サン社が大学サークルから発展してベンチャー企業になった1980年代のころのことです。

当時UNIXに多くの機能を追加していたビル・ジョイ氏はEthernetを利用した共有ファイルシステムを作ることに夢中になっていました。ビル・ジョイ氏はviエディタを作り、実際にUNIXの重要な部分を開発した人であり、現在ではUNIXの神の領域にいると評価されている人です。今でこそコンピュータ同士で同じファイルが同時に操作できることは当たり前ですが、そのころは先進的な考え方でした。

技術オタクとして社内で知られた大先輩がサン本社への調査を命じられます。そこではビル・ジョイ氏が対応にあたりました。そこで先輩はネットワークでつないだファイルシステムを作る、ということの実現について情熱的な説明を受けます。あまりに熱心に説明するのを聴いて、筆者の先輩もエンジニアとしての知見を総動員させて考え、冷静に評価したそうです。

ステートレスなUDPでファイルシステムを作るという説明に不安を感じた先輩はビル・ジョイ氏にこう言いました。「きみはUNIXをわかってない」と。そう言われたビル・ジョイ氏はかなりショックを受けたらしく、さらに真剣に説明を試みたそうです。時間にして1時間程度でしょうか。あまりに熱心に語るビル・ジョイ氏を見て「なかなか見所がありそうなアイデアだ」と言ったその先輩は、今ではこのときのことを一生の不覚と語ります。

ご存じのようにこの機能は現在ではNFSと呼ばれ、当たり前のように利用されています。

日本に戻った先輩は上司へ「技術的に先進性があり、かつ実現の見込みあり」と報告しました。

UNIXを作っている最中の神に会い、意見したことを先輩は恥じているのですが、それは時間が経ってから評価が決まったことです。先輩が気にしているやりとりがあったにしても、そんなことはどうでも良いくらいUNIXの輸入販売は結果として大成功を収めます。

シリコンバレーでは起業しても100のうち3つも残らないといわれますが、先輩らの技術調査の対象になった企業は10に3つくらいが当たりだったそうです。

クラウドが始まる

2000年代に入り、Linuxでのビジネスが始まったころ、別な大先輩がこんなことを言いました。「おい、トムがラリーに買収されたぞ」。

トム・シーベル氏はもともとオラクル社の天才営業といわれた人で、袂(たもと)を分かちシーベル・システムズ社(以降、シーベル社)を起業しました。その企業がオラクル社に買収されたのでした。シーベル社はCRM分野そのものを開拓するような製品を作っていました。販売営業員らのノウハウを顧客(会社や個人)にひもづけて記録し、シェアするしくみを提供する製品です。営業員の退職によるノウハウ損失も少なくなりますし、営業活動においてとても有効なツールとなります。その製品はセールス・フォース・オートメーションという名称でした。

その後、クラウドコンピューティングの代表格となるセールスフォース・ドットコム社のサービスはシーベル社製品と同様の機能を提供することからスタートしています。

実はその先輩、起業間もないシーベル社に派遣され、製品の日本語化を進める作業のとりまとめを担当した経験がありました。日本での販売拡大を考えていたトム・シーベル氏から先輩は大きな期待をよせられ、エンジニアとしては格別な待遇を受けていたのです。

当時はストックオプションを選んだ社員がカローラで通勤していたのにある日突然ポルシェで来た、みたいなことも日常茶飯事だったそうです。そんな栄華を誇ったシーベル社がクラウドサービスが登場しはじめたころに買収されてしまったのは象徴的な話だと思います。

ハードウェアの変化

アプリケーションのGUIで[保存]を意味するアイコンにフロッピーディスクが使われていることがあります。しかし実際にフロッピーディスクを見たことがないエンジニアも増えています。最後にフロッピーディスクを使ったのはいつでしょう。筆者は覚えていません。同様にモデムを見たことがないエンジニアが増えています。

ハードウェア技術はとにかく速く変化し、どう使ったらいいかすらわからないものが出てきます。

たとえばIntel Xeon Phiのようなものも出てきました。PCI拡張カードに通常のCPUと同じ命令が処理できるコプロセッサです。使い方がまだ確立していません。

レッドハットの製品では何PB(ペタバイト)でも1つのシステムとして管理できるストレージ製品が増えました。サーバとネットワークが価格の割に性能がよくなったことで実現したソリューションです。PBクラスのストレージを使いこなすことができる人はまだ多くありません。今までそんなサイズのストレージが普通にはなかったからです。これからいろいろな事例ができていくことでしょう。とても楽しみです。

オープンソースソフトウェア

そんな目まぐるしく変化する業界最先端で走り続けた先輩達からのアドバイスはこうです。「チャレンジしてもないことで人を批判するのはダメ。評論家と同じで絶対に世界を変えられない。とにかくチャレンジしろ」と、アメリカンなプラグマティズムを地でいくような感じです。確かに、真似をするためのリファレンスがないとき、どうすべきか自分で考えて対処する裁量が大きいほど個人の能力が発揮され、良い結果がついてくるような気がします。

この試練はすごいな、という例を1つ。大先

輩らと同じ部署に入社した新卒2年生の話です。

サウジアラビアの日本企業に納品したシステムが故障し、単身現地に派遣されたものの検査器具がないので東京の会社に連絡したところ「サガセ」とだけ電報を受けます。近くの別な日系企業でオシロスコープを借り、ジープを飛ばして持ち前の技術力で問題個所を特定したところ、1つのICチップが故障していたことがわかりました。再度、会社に連絡すると「ナオセ」と電文です。その新人はこんな状況を「おもしろい」と感じました。稀(まれ)な人材ですね。

その後、その人はお客様から信頼される仕事を数多くこなし、若くして執行役員にまでなりました。大先輩たちもなるほどと思える人事です。

さて、その人が技術部署の長となり、筆者が所属しているときのことです。筆者はオープンソースに関わる製品やサービス、仮想化技術について調査を担当していました。あるときRuby技術認定資格の制度開始を支援をするチャンスがありました。部署長は「とにかくチャレンジしろ」と後押ししてくれました。オープンソースに期待され、コンセプトに共鳴してもらえたと記憶しています。そして数年後、制度は実現しました。楽しい仕事でした。

それから

しばらく経ちました。オープンソースソフトウェアのいいとこ取りをした製品を作り、保守サービスを提供する企業に今筆者は所属しています。レッドハットのソフトウェア製品は設計図にあたるソースコードが公開されています。つまり製品そのものに企業秘密がありません。ものを売るというよりはアイデアやノウハウを売っています。とにかくチャレンジングな企業だと思います。

サン社のUNIXもsendmailなどBSDに含まれる公開された既存ソフトウェアをパッケージの一部として製品にしたものでした。結局、今も昔も知性の集約をするには同じ手法が活かされているのだと思います。

それはノイマン型なのか

ある企業でメインフレーム以外のコンピュータを導入検討するとき、「それはノイマン型なのか」という質問があったときいたことがあります。

技術的に理解している人にとっては質問に違和感があるものの、エグゼクティブ向けの説明にそういった言葉があり、基準のようにとらえられた結果の質問だったとも思われます。当たり前だからあえて確認しない不自然な質問です。

今は「それはオープンソースなのか」といった質問がときおりあります。

ソフトウェアの生産性と品質の両立を考えるとソースコードが公開されていることは重要です。この質問自体に違和感を感じるようにオープンソースであることが当たり前になるときがきっとくるな、と感じています。

書籍紹介

今までの話が面白いと感じた皆さんはこんな書籍も楽しめると思います。

『目利き――シリコンバレーのスター経営者たちが最も信頼する日本人』

桐山秀樹、日経BP社、1999年、1,470円

大先輩達が口をそろえて最もおっかないけど尊敬できる上役だったという方が主人公です。歴史的記述として楽しめます。

『Ruby技術者認定試験 公式ガイド』

伊藤忠テクノソリューションズ(著)、Rubyアソシエーション(監修)、ITpro(編)、日経BP社、2009年

資格制度事業化のとき、書籍出版にもチャレンジさせていただきました。Amazonで少しだけ立ち読みできます。

『ソフトウェア・グラフィティ』

岸田孝一、土屋正人、石曽根信、中小路久美代、石井達夫(著)、中央公論事業出版、2012年、1,785円

VAXによるUNIXのビジネスの話が含まれています。日本でのGNUの歴史も読み取れます。

SD

第15回 Linux 3.9の新機能 ～dm-cache～

今回はLinux 3.9の新機能“dm-cache”について解説します。dm-cacheはSSDをキャッシュとして使用することで、ディスクブロックのパフォーマンスを向上するための機能です。

dm-cache

SSDは一度使うと病みつきになってしまうような速度を誇っています。みなさんの中にもHDDの速度には戻れないな……と思っている方もいらっしゃるのではないでしょうか。ただ、この高速なSSDにも、まだまだ容量が小さく価格が高いという弱点があります。HDDで2TBが買えるような値段で、SSDではまだ250GBといったところでしょうか。

HDDの容量とSSDの速度、この2つの利点を活かすべくSSDをキャッシュとして活用するためのパッチが各種投稿されてきていました。“flashcache”や“bcache”といった名前に聞きおぼえがあるのではないでしょうか。本稿では、その中からカーネルにマージされたdm-cacheの動きを中心に見ていきます。

Device Mapper

dm-cacheはその名の中に“dm”という言葉を含んでいるとおり、Linuxの“Device Mapper(dm)”の機能を用いて実装されています。このDevice Mapperとはどのような機能なのでしょうか？

Device Mapperとは、仮想的なブロックデバイスを作り、その仮想デバイスへのアクセスを設定されたテーブルをもとにほかのブロックデバイス[注1]へのアクセスに変換するしくみです。これを使って、LVM2、multipath-tools、dmraid、cryptsetupといった機能がLinuxでは実装されています。

では、まず筆者の環境で設定してあるLVM(Logical Volume Manager)を例にして、具体的にこのDevice Mapperについて見てみましょう。LVMは論理ボリュームマネージャ(Logical Volume Manager)というもので、複数の物理デバイス、パーティションをまとめて、そこから論理的なディスク領域を作ることができる機能です。たとえば、1つのディスクの大きさにとらわれずに、大きなサイズのファイルシステムを作り出すことができます。

LVMの物理デバイス

物理デバイスに近いところから見ていきましょう(**図1**)。pvsコマンドで、LVMで使われている物理デバイスを見ることができます。この

注1) 実際のHDD、SSDはもちろんのこと、また別のDevice Mapperによる仮想ブロックデバイスなどに変換することもできます。

3つのパーティションsdb3、sdc1、sd2はpv createというコマンドでLVM用に初期化が行われています。LVM用に初期化された、これらの物理ボリューム(PV)をボリュームグループ(VG)に所属させます。ここではすべてのPVを、vg2というVGに所属させています。

最後にVGから論理ボリューム(LV)を切り出します。この論理ボリュームを、通常のパーティションかのようにファイルシステムを作ったりと利用できます。LVMによって/dev/"ボリュームグループ名"/"論理ボリューム名"というデバイス(へのシンボリックリンク)が作られています。

シンボリックリンクの先が"dm-<番号>"となっていることからも、LVMがDevice Mapperを使っていることがはっきりとわかります。これらのデバイスをDevice Mapperの側から見てみましょう。dmsetup lsでDeviceMapperによる仮想デバイスを見ることができます(**図2**)。ここで括弧の中に数字が書かれています。これが相当する仮想デバイスのデバイス番号となります。Device Mapperによって、作成される/dev/mapper下のシンボリックリンクと/dev/dm-*とを見てみると、対応していることが確認できるでしょう。"dmsetup table"コマンドを使うことでDevice Mapperの「テーブル」を見ることができます。Device Mapperはこのテーブルに従って仮想デバイスへのアクセスを変換していきます。たとえば、"vg2-storage"を見てみましょう。このデバイスのテーブルは**表1**のようになっています。

▼図1　LVMの表示

```
# pvs
  PV         VG   Fmt  Attr PSize   PFree
  /dev/sdb3  vg2  lvm2 a--  147.96g      0
  /dev/sdc1  vg2  lvm2 a--  298.09g      0
  /dev/sde2  vg2  lvm2 a--  931.69g 819.09g
# vgs
  VG   #PV #LV #SN Attr   VSize VFree
  vg2    3  10   0 wz--n- 1.35t 819.09g
# lvs
  LV                VG   Attr      LSize   Pool Origin Data%  Move Log Copy%  Convert
  Walbrix           vg2  -wi-a----  14.65g
  debian-root       vg2  -wi-a----  20.00g
  debian-swap       vg2  -wi-a----   2.00g
  genbutu           vg2  -wi-a----  15.00g
  gentoo-fbsd       vg2  -wi-ao---  20.00g
  gentoo-fbsd-broken vg2  -wi-a----  20.00g
  gentoo-fbsd8      vg2  -wi-a----  20.00g
  gentoo-linux      vg2  -wi-a----  20.00g
  nix               vg2  -wi-a----  30.00g
  storage           vg2  -wi-ao--- 397.00g
$ ls -l /dev/vg2
合計 0
lrwxrwxrwx 1 root root 7  4月  8 22:23 Walbrix -> ../dm-4
lrwxrwxrwx 1 root root 7  4月  8 22:23 debian-root -> ../dm-7
lrwxrwxrwx 1 root root 7  4月  8 22:23 debian-swap -> ../dm-8
lrwxrwxrwx 1 root root 7  4月  8 22:23 genbutu -> ../dm-3
lrwxrwxrwx 1 root root 7  4月  8 22:23 gentoo-fbsd -> ../dm-5
lrwxrwxrwx 1 root root 7  4月  8 22:23 gentoo-fbsd-broken -> ../dm-6
lrwxrwxrwx 1 root root 7  4月  8 22:23 gentoo-fbsd8 -> ../dm-2
lrwxrwxrwx 1 root root 7  4月  8 22:23 gentoo-linux -> ../dm-1
lrwxrwxrwx 1 root root 7  4月  8 22:23 nix -> ../dm-9
lrwxrwxrwx 1 root root 7  4月  8 22:23 storage -> ../dm-0
```

▼表1　vg2-storageのテーブル

開始セクタ	サイズ	マップ方法	マップのパラメータ
0	310304768	linear	8:19 384
310304768	340656128	linear	8:33 284477440
650960896	8388608	linear	8:33 182454272
659349504	173219840	linear	8:66 2055

/dev/vg2/storageの0～310304767セクタ[注2]までが、linearという方式で"8:19 384"というパラメータを用いてマップされます。linearではパラメータは"<デバイス番号> <オフセット>"という解釈となります。つまり、ここでは/dev/vg2/storageの0セクタ目が/dev/sdb3(8:19)の384セクタ目に、1セクタ目が385セクタ目に、というように310304767セクタまで対応していきます。そして/dev/vg2/storageの310304768セクタからは、テーブルの次のエントリになり、/dev/sdc1(8:33)の284477440セクタ目からにマップされていく、となります。

注2) 1セクタは512バイト。

▼図2 Device Mapperの表示

```
# dmsetup ls
vg2-gentoo--fbsd--broken        (253:6)
vg2-debian--swap        (253:8)
vg2-debian--root        (253:7)
vg2-storage     (253:0)
vg2-nix (253:9)
vg2-Walbrix     (253:4)
vg2-gentoo--fbsd        (253:5)
vg2-gentoo--linux       (253:1)
vg2-genbutu     (253:3)
vg2-gentoo--fbsd8       (253:2)
$ ls -l /dev/dm-* /dev/mapper/*
brw-rw---- 1 root disk 253,   0  4月  8 22:23 /dev/dm-0
brw-rw---- 1 root disk 253,   1  4月  8 22:23 /dev/dm-1
brw-rw---- 1 root disk 253,   2  4月  8 22:23 /dev/dm-2
brw-rw---- 1 root disk 253,   3  4月  8 22:23 /dev/dm-3
brw-rw---- 1 root disk 253,   4  4月  8 22:23 /dev/dm-4
brw-rw---- 1 qemu qemu 253,   5  4月 11 13:16 /dev/dm-5
brw-rw---- 1 root disk 253,   6  4月  8 22:23 /dev/dm-6
brw-rw---- 1 root disk 253,   7  4月  8 22:23 /dev/dm-7
brw-rw---- 1 root disk 253,   8  4月  8 22:23 /dev/dm-8
brw-rw---- 1 root disk 253,   9  4月  8 22:23 /dev/dm-9
crw------- 1 root root  10, 236  4月  8 22:23 /dev/mapper/control
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-Walbrix -> ../dm-4
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-debian--root -> ../dm-7
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-debian--swap -> ../dm-8
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-genbutu -> ../dm-3
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-gentoo--fbsd -> ../dm-5
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-gentoo--fbsd--broken -> ../dm-6
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-gentoo--fbsd8 -> ../dm-2
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-gentoo--linux -> ../dm-1
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-nix -> ../dm-9
lrwxrwxrwx 1 root root        7  4月  8 22:23 /dev/mapper/vg2-storage -> ../dm-0
# dmsetup table
vg2-gentoo--fbsd--broken: 0 41943040 linear 8:33 221562880
vg2-debian--swap: 0 4194304 linear 8:33 178259968
vg2-debian--root: 0 20971520 linear 8:33 157288448
vg2-debian--root: 20971520 20971520 linear 8:33 263505920
vg2-storage: 0 310304768 linear 8:19 384
vg2-storage: 310304768 340656128 linear 8:33 284477440
vg2-storage: 650960896 8388608 linear 8:33 182454272
vg2-storage: 659349504 173219840 linear 8:66 2055
vg2-nix: 0 62914560 linear 8:66 173221895
vg2-Walbrix: 0 30720000 linear 8:33 190842880
vg2-gentoo--fbsd: 0 41943040 linear 8:33 115345408
vg2-gentoo--linux: 0 41943040 linear 8:33 2048
vg2-genbutu: 0 31457280 linear 8:33 83888128
vg2-gentoo--fbsd8: 0 41943040 linear 8:33 41945088
```

いろいろなマップ方式

先ほど「linearという方式で」と書いたとおり、Device Mapperにはlinear以外にも多くのマップ方式を備えています(**図3**)。

簡単なものでは、striped、mirror、crypt、verifyといったものがあります。stripedは複数のデバイスにデータをストライプ状にばらけさせることができます。たとえば、ストライプのサイズを128とすると仮想デバイスの0〜127セクタが/dev/sda1に、128〜255セクタが/dev/sdb1に、256〜383セクタはまた/dev/sda1に……といったようにマップします。I/Oが複数のデバイスにばらけるのでスループットの向上が期待できます。

mirrorはその名のとおり、複数の物理デバイスにデータのミラーを作ることができます。cryptでは透過的に下位のデバイスへの暗号化を施します。verifyは読み込み専用のデバイスを作ります。このデバイスから読み込まれたデータは、「ハッシュ用デバイス」に記録されたハッシュによって検証が行われます。もしディスクへの改ざんなどがあってハッシュが一致しない場合、そのI/Oは失敗します。Chrome OSではこの方式を用いて、Verified bootを実現しようとしています。

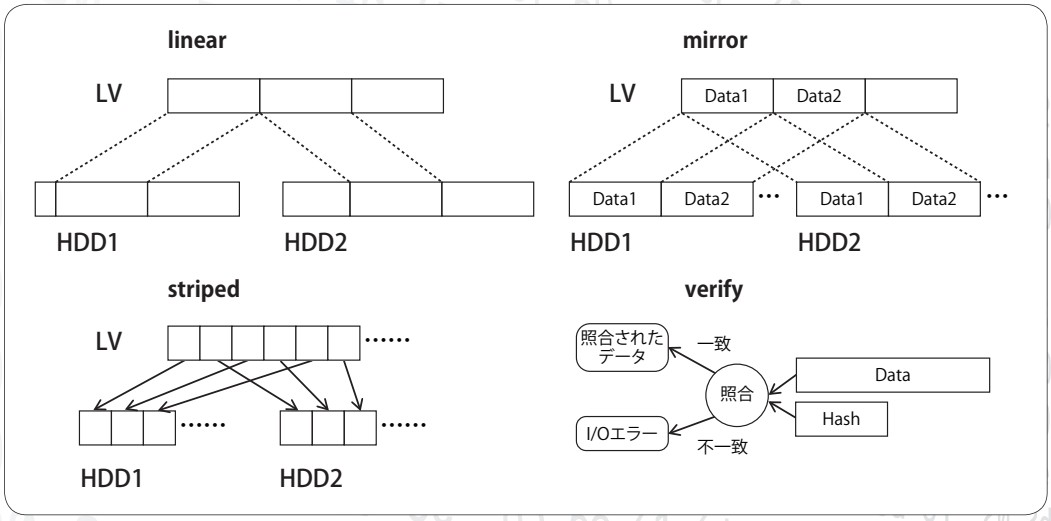
変わったものではerror、zero、delay、flakeyといった方式があります。errorは常にI/Oがエラーとなり、zeroでは常に0が読み込まれ書き込みは無視し、delayは指定されたミリ秒だけI/Oを遅延させます。flakeyは一定の間隔で、正常状態と異常状態とを交互に繰り返すようなデバイスを作ることができます。異常状態の間は書き込みを無視する／読み書き時に指定したバイトのデータを書き換えて伝える、といった設定ができます。これらの方式はおもにデバッグ用、テスト用として使われることが多いです。

snapshot関連のマップ方式

そしてsnapshot関連のマップ方式があります。LVMにてスナップショットを作成する様子を見てみましょう(**図4**)。

まずスナップショットの元となるLV baseがあります。スナップショットsnap1を作ると、baseのテーブルはsnapshot-originを使ったマップに変えられます。また、もともとbaseで参照できていた領域はbase-realという名前の新しいマップに変えられます。そしてsnapというLVと、snap-cowというLVの2つも新しく作られます。

▼図3　Device Mapperのマップ方式



snapはsnapshotというマップであり、snap-cowはlinearのマップとなっています。snapshot-originは元の領域をパラメータとしてとります。baseに書き込むと、同時にbase-realにあった古いデータがsnap-cowへと書き込まれます。snapshotは元の領域および古いデータが保存される領域をパラメータにとります。この2つの領域のデータをマージして、昔のデータを見せることでスナップショットを実現しています。

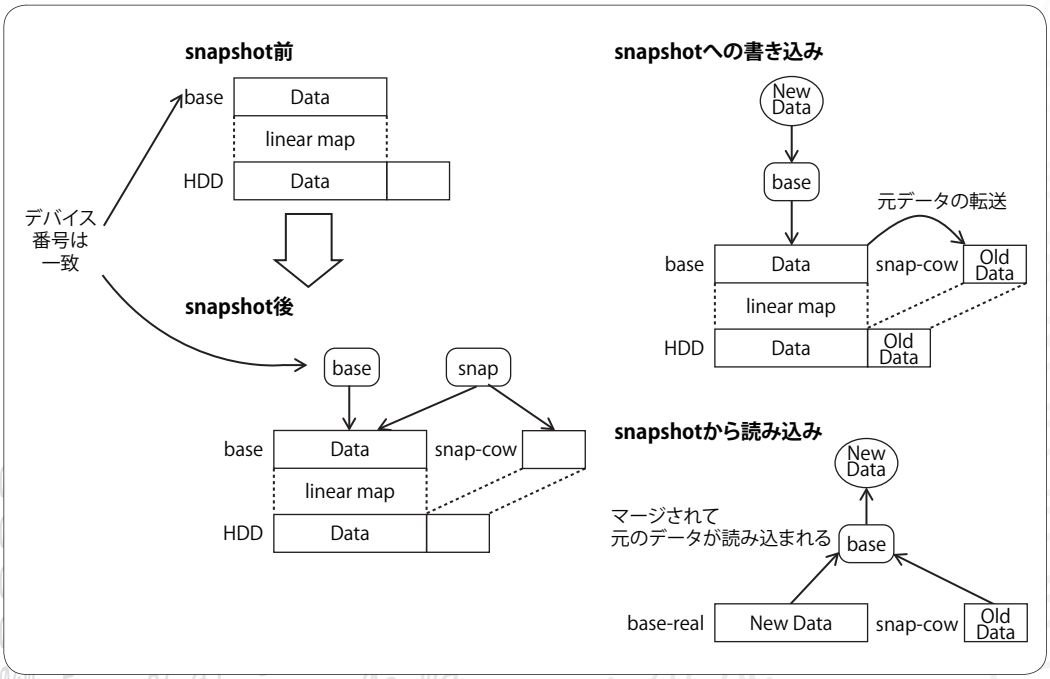
dm-cache

これらのマップ方法と同じように今回紹介するdm-cacheも実現されています。dm-cacheのテーブルは**図5**のようなパラメータで構築されます。

このパラメータからわかるように、dm-cacheでは3つのデバイスを使用しています。オリジナルデータデバイスはキャッシュの元となる、大容量で遅いデバイス(HDDなど)であり、キャッシュデバイスはデータをキャッシュする小容量で速いデバイス(SSDなど)です。メタデータデバイスにはキャッシュ上のデータだけが更新されていて、オリジナルデータが更新されていない状態である、などの管理用データが記録されます。

featureには現状writebackもしくはwritethroughのどちらかが指定されます。writebackの場合、キャッシュされているデータに対して書き込みがあったときに、キャッシュにだけ書き込みを行い、dirtyフラグを立ててI/O処理を終了します。オリジナルデータデバイスへの更新は、このdirtyフラグをもとにして後で行われます。

▼図4　snapshot



▼図5　dm-cacheのテーブルパラメータ

```
cache <メタデータデバイス> <キャッシュデバイス> <オリジナルデータデバイス>
<ブロックサイズ>
<feature引数の数> [<feature引数>]*
<ポリシー> <#ポリシー引数の数> [ポリシー引数]*
```

writethroughの場合、オリジナルデータに書き込みを行ったあとにキャッシュデータを更新し、I/O処理を終了します。

ポリシーにはデータをキャッシュにのせる／キャッシュから消すといった基準を決定するポリシーの名前を指定します。今のところmultiqueueというポリシーと、cleanerというポリシーとがあります。しかし、cleanerのポリシーの方はすでにキャッシュに入っているかどうかを返し、キャッシュへやキャッシュからの移動はまったく行わないキャッシュの動作には役に立たないポリシーになっています。このcleanerポリシーはdirtyなキャッシュをオリジナルデータデバイスに書き戻していくので、キャッシュを“clean”にして解放するために使えます。

multiqueueポリシーの動作

それではデータの動きを見ながら、multiqueueポリシーの動作を見ていきましょう。multiqueueではpre_cacheとcacheという2種類のキューがあります(**図6**)。

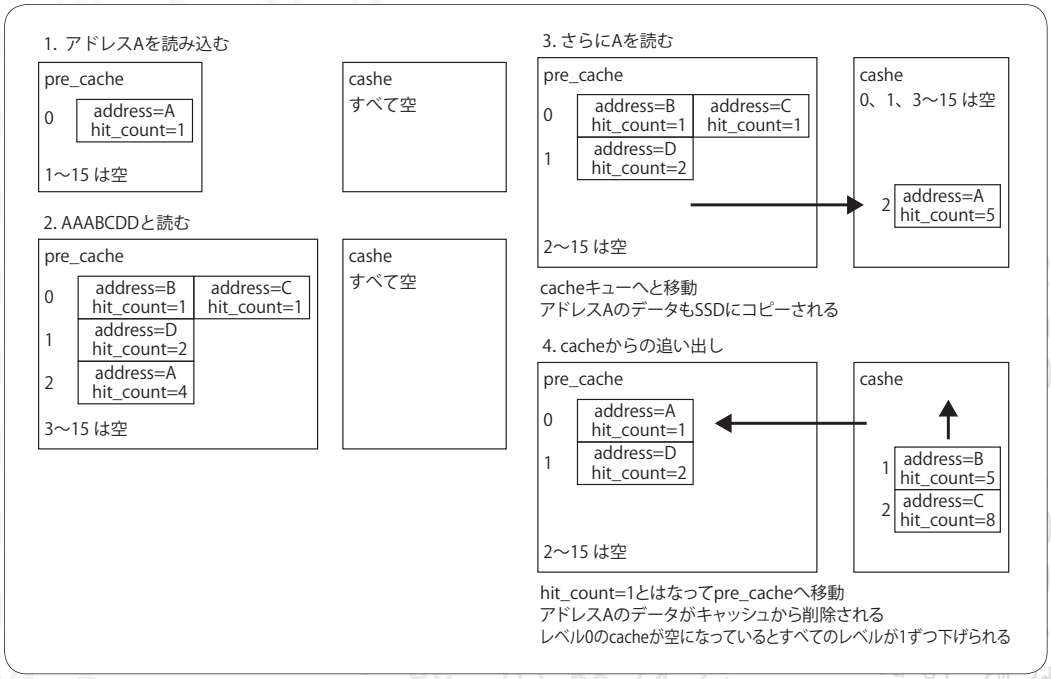
最初にデータにアクセスすると、そのアドレスへのアクセスエントリが作成され、pre_cacheキューに挿入されます。アクセスエントリにはhit_countというアクセスされた回数をカウントした変数があります。pre_cacheキューもcacheキューも内部が16段階(レベル)に分かれています。

アクセスエントリがキューのどのレベルに入るかはlog_2(hit_count)によって決定されます。ここでは最初のデータアクセスで“hit_count=1”ですのでレベル0に入ります。pre_cacheキューに入ってもまだデータのキャッシュは行わず、オリジナルデータデバイスからデータを取得してもらいます。

アクセスがあるたびにhit_countが1つずつ増加し、新しいhit_countに応じたレベルのキューにエントリが挿入されなおします。こうしてデータが何度もアクセスされているとやがてデータをキャッシュへと移す(promoteする)ときがやってきます。判定基準は**図7**のようになっています。

基本しきい値は、cacheキュー内のhit_count

▼図6　pre_cacheとcacheのデータの動き



が少ないものから20個のアクセスエントリのhit_countの平均値となっていて、定期的に更新されています。読み書きによる調整は、そのアクセスが読み込みである場合に4、書き込みにある場合に8となっていて、読み込みのほうがより早くキャッシュに乗るようになっています。キャッシュからの削除のことを考えると、書き込みの場合にはキャッシュ内のデータをオリジナルデータデバイスに書き戻す手間がかかります。そのため読み込みよりも書き込みのほうがしきい値が厳しく設定されているわけです。

やがて、このデータがアクセスされなくなりキャッシュが別のデータで埋まってくると、アクセスエントリがcacheキューの先頭へと移動していき、先頭に到達してさらにキャッシュされる新しいエントリがでてきた時点でキャッシュからデータが追い出されます。

dm-cacheのmultiqueueポリシーの大雑把な流れは以上のようになっています。細かいところではもう少し調整が行われています。たとえば、I/Oパターンを調べておいて、そのI/Oがシーケンシャルなものかランダムなものかを判定します。大規模で連続したI/Oであれば、HDDであっても良いパフォーマンスとなるので、この場合はオリジナルデータデバイスから読み書きするようになっています。

まとめ

今回はHDDとSSDの利点を活かし、大容量で高速なデバイスを実現するdm-cacheという新しいDevice Mapperのマップ方式を紹介しました。ポリシーの切り替えができるように設計されているので、自分でポリシーを書いてみるのもおもしろいかもしれませんね。

さて、おまけとして最近(4月初めごろ)投稿された変わったパッチを2つ紹介しましょう。

1つめはチューリングマシーンのサポートです[注3]。まだまだ作成中のようでconfig項目と簡単なMakefileのみのパッチですが、無限に長いテープをLinuxでサポートするようです。

2つめはevent subsystemのevent typeにconferenceを追加しようというものです[注4]。event subsystemは入力デバイスの1つで、たとえばマウスやキーボードといったデバイスから入力をイベントという形式で受け取ります。たとえば、筆者のマシンであれば**図8**のようなコマンドで、マウスを動かすとイベントが出力されている様子を見ることができます。このパッチではカーネルサミット、LinuxConといったLinuxのイベントをこのeventシステムで取り扱うことができるように、これらのイベントに定数を設定しています。こちらはstableカーネルメンテナのGreg KHからAcked-Byがついているのでもしかしたらマージされるかもしれませんね[注5]。どちらも奇妙なパッチですが、脚注のURLを見れば投稿された日付がわかるかと思います;-)SD

注3) https://lkml.org/lkml/2013/4/1/11
注4) https://lkml.org/lkml/2013/4/1/181
注5) https://lkml.org/lkml/2013/4/1/355

▼図7　キャッシュ移動の判定基準

```
hit_count ≧（基本しきい値）＋（読み書きによる調整）
```

▼図8　デバイスからイベントを受け取る

```
sudo dd
if=/dev/input/by-id/usb-ELECOM_ELECOM_USB_mouse_with_whell-event-mouse
of=/dev/stdout |hexdump -C
```

June 2013

NO.20

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

IT業界や仕事以外にも目を向けよう—IT技術者のこれから

今回は、3月に徳島で行いました研究会の模様をお伝えします。また、6月より新年度を迎えるにあたり、新規会員を募集致します。

jus研究会徳島大会の報告

■IT技術者とITコミュニティのこれから
【日時】2013年3月9日（土）16:15～17:00
【会場】とくぎんトモニプラザ
（徳島県青少年センター）
【講師】中野 秀男（大阪市立大学名誉教授／
大阪市ITアドバイザ／中野秀男研究所）
【司会】法林 浩之（日本UNIXユーザ会）

2012年度としては最終回となるjus研究会を、jusの行事としては初開催となる徳島県で行いました。今回の講師には大阪市立大学名誉教授の中野先生をお迎えし、IT技術者やITコミュニティのあり方についてご講演いただきました。今回も展示会場内にセミナーブースが設けられていたために観覧者が多く、47人の方にご参加いただきました。

■他業界とのコラボレーションを見据えること

はじめに中野先生から自己紹介があり、自身と徳島の関わりとして、徳島県方面の組織をインターネット（当時はJUNET）につなぐことに尽力した話が紹介されました。その後、本題に入りましたが、先に結論として、「ITは当たり前の時代に入ったのでそれ以外の得意分野を何か作ったほうが良い。そしてほかの分野の人と協業するためにコミュニケーションやコラボレーションの能力を身につけるべきである」という意見が提示され、その具体例をいくつかのプロジェクトを交えつつ紹介されました。

たとえば、4月にオープンする大阪駅北側の梅田北ヤード地区（通称：うめきた）の開発プロジェクトにおいては、中野先生は都市計画や建築の専門家たちと一緒に街のコンセプト作りを考えました。また、ここに入居するVislab Osakaは可視化にまつわる技術とノウハウを結集した組織ですが、可視化を実現するにはコンピュータの知識だけでなく、アーティストやディレクターなどとコミュニケーションする能力も問われます。大阪府立大学が取り組む植物工場というプロジェクトは、空調や照明などを制御して完全人工光型の植物栽培を実現すべく研究していますが、これも生物や農業に携わる人々とのコラボレーションが必要です。このようにITは幅広い分野で活用されるようになってきていますが、それに伴って発生するセキュリティやプライバシーの問題を考慮せずにシステムが作られることも多いので、たとえばそういうところにIT技術者の出番があるだろうというのが中野先生の見解です。

■好きなことはコミュニティ活動に見いだす

また、後半ではボランティアに対する考えも示されました。自分の好きなことが生業になれば幸せですが、必ずしもうまくいくとは限りません。そこで、稼ぐための仕事とは別にボランティアという形で好きなことをやるのも1つの方法で、実際にITコミュニティにはそういう人がたくさんいます。中野先生はそういう関わり方を肯定しつつも、それを

先々どのようにしていくか（やがては生業にしたいのかどうかなど）のビジョンを持って活動してほしいとコメントされました。そして最後に、ITの世界は勉強を続けていくことが必要であるとし、今後もイベントに出かけて勉強を続けていきたいという意思を示されました。

中野先生は4月から帝塚山学院大学の教授に着任されるとのことで、これからもまだまだ頑張る姿を見ることができそうです。なお、今回の講演は次のURLで映像を見ることができます。また、中野先生自身により資料も公開されていますので、併せてご参照ください。

- **講演の映像**
  http://www.ustream.tv/recorded/29845788
- **講演で使われた資料**
  http://nakanohideolab.jp/wp/contents/rejume

2013年度新規会員募集

jusは6月に新年度を迎えます。jusへの入会は随時受け付けていますが、新年度を迎えるにあたり、とくに積極的に会員募集を行っています。jusの活動に賛同される方々のご参加をお待ちしております。

jusは、本会の主旨に賛同する個人もしくは組織が会員となっており、個人会員、学生会員、法人会員、賛助会員から構成されています。それぞれの入会資格、年会費は、**表1**のとおりです。

jus会員の主な特典としては、次のものがあります。

- **勉強会、ワークショップなど、jusの開催するイベントへ（表2）の参加費割引（法人会員／賛助会員は、その組織に所属する方すべてが対象となります）**
- **イベント告知などを載せた会員向けニュースレターの配布**
- **会員向け機関誌「/etc/wall」の配布**

jusへの入会をご希望の方は、office@jus.or.jpまでお問い合わせください。また、jusのWebサイト（http://www.jus.or.jp/）にも入会案内や入会申込フォームを用意していますので、そちらもご覧ください。これからもjusをよろしくお願いします。SD

▼表1　jusの入会資格／年会費

種別	資格	初年度年会費	次年度以降
個人会員	個人	7,000円	6,000円
学生会員	学生	4,500円	3,500円
法人会員	組織（営利団体）	95,000円	90,000円（1口あたり）
賛助会員	個人または組織（非営利団体）	45,000円	40,000円

▼表2　2013年度jus行事予定

行事	開催時期	開催場所
定期総会	7月	東京都内
Lightweight Language Matsuri	8月24日	すみだ産業会館
Internet Conference 2013	10月24～25日	慶應義塾大学 三田キャンパス
関西オープンソース 2013	11月8～9日	大阪南港ATC
Internet Week 2013	11月後半	東京都内（予定）
勉強会	随時	東京、大阪、名古屋など
研究会	年6回程度	全国各地
ワークショップ	年2回程度	—
運用研究会	不定期	—
その他協力イベント	オープンソースカンファレンス、Developers Summit、TechLIONなど	

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18回 Hack For Japanスタッフ座談会［前編］

東日本大震災から2年が経過しました。ITで復興支援を考えるHack For Japanとして活動を続けていくにあたり、「これまでの振り返りと今後に向けた決意」をテーマにスタッフで座談会を開きました。当日は約2時間、盛りだくさんの内容となったため今号と次号の2回に分けてお送りします。

昨年度の振り返り

高橋：まずは昨年度、3月までの振り返りから話していきたいと思います。印象に残ったイベントや出来事などあればお願いします。

宮城、福島での活動

佐々木：石巻の「IT Bootcamp[注1]」に皆で行けたのは面白かったですね。あとはHack For Fukushimaとしては活動が停滞しているので、そこは反省すべき点だったと思います。どうして停滞したかを考えてみると、昔は福島は静かなところなので結構のんびりと仕事をしていたんですね。震災後の1年目は良くも悪くもいろいろな人が来るし雑音も入るしで、かなりハイペースになっていたんです。いろんなものが。2年目は余韻で土日は休もうかという雰囲気になっていたように思います。そして3年目は元どおりのペースでできることをやっていこうかという感じになり、そういった反動でゆったりしすぎていた感はありますね。

高橋：会津では今年はHack For Japanのイベントをやることができていなかったですね。

佐々木：確かに会津ではやっていなかったのですが、会津大学のイベントや「ABC[注2]」もありましたよね。そちらのほうの準備に追われていた感はあります。あとは石巻のBootcampも結構準備に時間はかかりました。

関：福島全体としてほかのコミュニティなどからのITによる支援はあったのでしょうか。

佐々木：福島は広いので地域ごとに独立していて、そのハブになるような人がいなかった状況です。あとはイベントをやっても人を集めるのが大変になってきたように感じます。たとえば、モバイルクリエイターズサミットで実施したビジネスプランコンテストは賞金も用意して復興3県（岩手、宮城、福島）で開催しました。現地では地元の人が協力してくれて集まるのですが、東京で決勝戦をやったときは事前登録で100人を超すのが大変でした。東京でモバイルコンテンツ系のイベントをやると200人くらいは楽に集まるのですが、復興系のイベントで人を集めようとするとなかなか大変です。IT業界と復興支援のボランティアをやっている人とは意外と距離があるのではないかと感じています。

鎌田：去年「エフスタ[注3]」さんのイベントをYahoo! Japanでやったときは集客は盛況でした。

小泉：どういうリーチのさせ方をするかにもよると思います。今年の3月11日に仙台で「ITで日本を元気に」の佐々木賢一さんがやったイベントでは、平日にも関わらず150人くらい来ていました。以前ほどは集まりにくくなっているとは思うのですが、持って行き方次第だと思います。

OpenStreetMapの課題

岩切：私の活動報告としては、岩手で3月、5月、9月にOpenStreetMap（以下OSM）のイベントを3回

注1）2012年7月に石巻の高校生に3日間でアプリ開発を学んでもらったイベント。
注2）日本 Androidの会主催のイベント、Android Bazaar and Conferenceのこと。2012年10月はICT ERA＋ABCとして仙台で開催され、Hack For Japanでも2つのセッションを担当。
注3）福島で活動しているITエンジニアのためのスキルアップ応援コミュニティ。

参加者紹介（Hack For Japanスタッフ）

及川 卓也

Hack For Japanの立ち上げメンバーの1人。普段はGoogleでChromeを担当しているほか、知り合いのスタートアップやNPOに助言を与えたりしている。今回はGoogle Hangoutで参加。

小泉 勝志郎

サンキュロットインフォ代表。スマートフォンアプリ開発関連の事業を行う。震災で失業者が多い南相馬でアプリ開発者育成をした。震災復興では「うらと海の子再生プロジェクト」にてIT関連を担当。

岩切 晃子

（株）翔泳社勤務。毎年開催されているデブサミを運営。岩手県釜石市出身でHack For Japanの活動で岩手とのパイプ役として奮闘している。

やりました。世界で活躍するOSMマッパーの方々を岩手で案内することもやりました。ほかには、ITの縛りを除いて岩手で頑張っている人を応援するイベントをやってくれないかと岩手側から打診を受けて、「いわて未来Meetup」を11月に盛岡で、2013年3月には東京でもやりました。

OSMについてはもう一度総括が必要ですね。OSMの東京側の人の熱意はあるのですが、地元ですでに復興支援のことで頑張っている方々にさらにOSMの作業をやってもらうというのは、ハードルが高いのではないかと感じています。本当に必要なのは岩手側で何か困ったことが起きたときに、技術者同士が連携できるしくみが求められていると思うのですけれども、2013年はそれに取り組めたらいいなと思っています。

一方、「いわて未来Meetup」で感じたことは、スタートアップの仕事を流されてしまった人たちが被災地でもう一度起業したり、何かプロジェクトを起こしたりということをやっているところではITはすごく使われているということです。Microsoftが支援して学校を開いてくれたり、Matz[注4]が来てくれたりとニュースには事欠かないのですが、そういうことを上手く使って東京で稼ぎたい人たちをつなぐ活動もしていきたいです。

高橋：OSMで復興する街を記録するイベントは、今度は石巻でもやりますよね。

関：OSMとして復興マッピング活動は初めての試みだったし、可能性はまだすごく感じています。反省としては、やはりコミュニティ活動なので地元で理解して主導してくれる人がいないとなかなか上手くいかないということです。いつまでも遠隔地からのサポートでは続かないです。

一方で、復興マッピングの活動が仕事に結びつくかというと、なかなかお金にはならないので継続するためのハードルが高いという課題もあります。出口としては“印刷”が1つのポイントだと思っています。今でも手書きの地図を配布している人がいたりと、紙地図のニーズは高いので、印刷ツールがあると役に立つと思っています。そのためのツールの整備はやりたいです。お金にならなくてもそれが役に立てば良いと思っています。次の石巻ハッカソンでは印刷して配れるところまでは見せたいですね。結局、地図を充実させても使えなければ意味がないので、シンプルに小さく回るプロセスを1つ作ろうかなというところです。

OpenDataへの取り組み

関：ほかに挙げられるのは、本連載でも何度か取り上げているように、OpenData関連のハッカソンが発展していっていることです。この活動をやっているうちに防災関連のさまざまなところから声をかけ

注4）Rubyのまつもとゆきひろ氏。

参加者紹介（Hack For Japanスタッフ）

鎌田 篤慎
普段はヤフー（株）が公開するAPIなどの利用促進、デベロッパリレーションなどを業務としている。Hack For Japanでは復旧復興支援データベースAPIへの改善要望をまとめ、国に提言した。

佐々木 陽
会津若松の（株）GClueの代表取締役。Android、iOSアプリケーション開発が主な事業。未来の主戦力となるエンジニアを育てるため、大学生などに教える活動を10年間行っている。

佐伯 幸治
Hack For Japanではコピーライティングをおもに担当。普段はフリーランスとしてWebや紙媒体の編集制作・コピーライティングに携わっている。

てもらいました。たとえば消防庁からは、大規模災害時におけるSNSなどによる緊急通報のあり方を検討する委員会に呼んでもらって、大規模な災害が起きたときに電話が通じなくなっても緊急通報がTwitterなどから送れるようにしようというワーキンググループに入れていただきました。あとは内閣府がやっているOpenData推進のIT戦略会議の委員をやらせていただいたりと、しくみを変える部分からいろいろ関われるようになったという変化が起こっています。

岩切：昨年6月の復旧・復興データベースAPIハッカソンと、今年の2月にもやったOpenData のハッカソンは国からも人が来ていたし、東京でできる意義のある活動だったかなと思います。

関：経産省の人たちも「あれは良かった」と言ってくれています。

鎌田：OpenData活動については、震災当時データがオープンではなかったために力が発揮できなかったエンジニアがすごく多かった印象を持っているので、ここはお手伝いしていきたいと思っています。全体としては、ハッカソンであまりにも成果を求めすぎていたのが1年目で、2年目はその中で試行錯誤していたということかなと思うのですが、ハッカソンを継続的にやっていくことでエンジニア同士の緩やかなつながりを持ち続けるコミュニティができ上がっていくことのほうが、今後の震災に備えるという点で意義があるかなと思います。OpenDataもその流れでどんどん出て行くような流れを作れれば、次に何かあったときへの備えという点で、人の面でもデータの面でも形付けられるのではないかと思っています。先日読んだCode for AmericaのHacking the Hackathonという記事の中でも、「アウトプットを求めすぎず、コミュニティの形成が大切」ということが挙げられていて、こういう活動を通してエンジニア同士がつながっていくことや、メンターを探そうという意識はいろいろな課題を解決する枠組みになるのではないかという気がしています。

関：まさにコミュニティが大事だと思っています。最近はOpenDataハッカソンをやると毎回来てくれる人もいますし、何よりいいのはエンジニアと行政とGLOCOM[注5]のような公共を改善しようとしている人たちが集まった結果、顔見知りが増えてきて動きやすくなっています。

コミュニティ形成の重要性

岩切：そういう活動を見て「参加したい」と思ってくれる人も増えるような気がしていて、デブサミ[注6]でITによる復興支援系のコンテンツをいくつか入れたのですが、集客は悪くなかったです。皆さんの関心は冷めてはいない、一時ほどの熱はないもののむ

注5）国際大学グローバル・コミュニケーション・センター。「情報社会学」と総称される情報社会の諸側面の学際的・総合的研究を行う研究所。
注6）翔泳社主催のデベロッパーズサミット。Hack For JapanでもOpenDataに関するセッションを担当。

参加者紹介（Hack For Japanスタッフ）

関 治之
Georepublic Japan社CEO。Geo Developerとして位置情報系のサービスを数多く立ち上げてきた。Hack For Japanでは、復興マッピングやオープンデータハッカソンなどを実施している。

石野 正剛
震災直後に福島第一原発から放出される放射性物質と風向きを地図上に可視化するスマホアプリ「風@福島原発」を開発した。富士通（株）でソフトウェアのUXデザインを担当している。

高橋 憲一
普段は（株）スマートエデュケーションのエンジニアとしてiOSやAndroidの子供向け知育アプリ開発を行っている。最近は東北TECH道場の講師として宮城県の石巻を頻繁に訪れている。

しろ冷静に見ることができている人が増えていると思いました。

関：あのセッションを聴いて、spending.jpのしくみを使って自分の街の税金がどのように使われているか可視化する“Action[注7]”をしてくれた人がいたのも良かったですね。

岩切：佐伯さんの復興イベントカレンダー[注8]もいいですね。

佐伯：復興イベントカレンダーを作った理由は、自分が東京で何ができるのかという思いがあったのと、イベントの日付が手軽に作れるしくみを作っておくと減災に役立てることができるかと思って、コンパクトに作れるしくみを運用してみようと考えました。これが形になれば被災地で使える可能性もあると思ってやっています。あとは、ちょこちょことネタを落としていくとコミュニティの活性化につながるかなという気がしてやっています。

高橋：石野さんのスキルマッチング[注9]のほうは何かありますか？

石野：12月から登録を開始して、Facebookではあまり盛り上がらなかったので登録者メーリングリストを作りました。今のところ25人の登録があり、スキルの高い方にも登録していただいています。もし被災地でITの手を借りたいという方がいたら、すぐにでも申し出ていただければと思っています。

佐伯：思った以上に知られていないのではないでしょうか。

石野：まだ登録者が少ない状況で、あまり大っぴらに言えないかなというのもあります。

高橋：そこは鶏と卵の話に近い気がしますね。

佐伯：「スキルマッチングやってます」ということをほかの人に向けてもっと伝える必要があると思います。

岩切：「遠野まごころネット[注10]」が東京支部を作って活動していて、毎週水曜に「まごころカフェ」という場で今の復興支援の話をやっているのですが、すごく良いコンテンツなのにUStreamを使える人がいないので広く公開できないといった、プログラミング以前に、ITに詳しい人が必要とされている場面もあるので、潜在的なマッチングの需要はかなり多いと思います。

◆◆◆

次号の後編では、福島の問題に対して我々ができること、教育活動、今後に向けた決意についての話を展開していきます。SD

注7）今年のデブサミのテーマ。セッションを聴いて終わりではなく、実際に行動につなげてほしいという想いが込められている。

注8）都内＆近郊で開催される復興イベントをお知らせする取り組み。Facebookのほか、TwitterやGoogleカレンダーでも運用中。URL http://www.facebook.com/fukkouevent

注9）復興支援などに対してITを活かして貢献したい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。URL http://blog.hack4.jp/2012/12/blog-post.html

注10）東日本大震災で被災した岩手県沿岸部の被災者の方々を支援するべく、遠野市民を中心として結成されたボランティア集団。URL http://tonomagokoro.net/

温故知新
IT むかしばなし

第23回

MP/M

たけおかしょうぞう　TAKEOKA Shouzou　take@takeoka.net

はじめに

8bit時代にマルチユーザOSは存在しました。ザイログのZ80より古い、2MHzのインテル8080 CPUで動作する「MP/M」は、「CP/M」のマルチユーザ版で1979年に発売されました。

CP/M

CP/MはMS-DOSの手本になったOSで、プロンプトは「A>」です。CP/Mは1974年頃から開発され、1977年頃世界的に有名になっていた8080（Z80）用のOSです。当時、スタティックRAMは1Kbit/1チップが主流で、やっとDRAMは4Kbitのものが標準になり、Apple IIのような先進的な機械が4K〜12KB程度を実装していました。

MP/M

そういう時代に、MP/Mは8080（Z80）で、32KB以上のメモリを使用し、16ユーザを同時サポートしたのです。端末は、通常はシリアルポートに接続し、16台の端末を接続できました。MP/M II（8080版）では、最終的にメモリのバンク切り替えをサポートし、400KBのメモリが使用できました。

MP/MはCP/Mのシステムコールのすべてをほぼそのままサポートしたので、CP/Mの豊富なソフトウェアがそのまま使えました。

MP/Mは、1人のユーザが複数のプログラムを同時に使うことも可能で、あるプログラムが入力待ちのとき、キーボードから「^D」を入力すると、そのプログラムはサスペンド（デタッチ）され、コマンドプロンプト「A>」に戻ります。そこで別のプログラムを起動し、実行できました。そのプログラムでも、「^D」を入力すると、そのプログラムをサスペンドできます。

そしてコマンドプロンプト「A>」で、「^D」を押下すると、先にサスペンドされたプログラムの実行が再開されます。プログラムの再開は、ATTACHコマンドで明示的に行うこともできます。これは、UNIX/Linuxのcsh/bashの「^Z」によるsuspend、jobコントローラと同様のものです。

UNIXライクなコマンド

MP/Mには、UNIXのatコマンドに相当するSCHEDコマンドがあり、特定の日時に所望のコマンドを実行することが可能でした。UNIXのkill相当のABORT、psのようなMPMSTATというコマンドもありました。プリンタのスプーラを備え、SPOOLコマンドで印刷ジョブをいくつも投入できました。

MP/Mでは、おもにフロッピーディスクが使用されていたので、メディアを取り替える前にDSKRESETコマンドを使用することが必須になっていました。これはUNIXのumountと似たような意味です。また、「QUEUE」というタスク間通信の機能も備えており、それを使用して、タスク間の排他制御、同期、通信が行えました。

リロケータブルバイナリ

8080（Z80）にはメモリのアドレス変換を行うような高級なハードウェアはありません。しかし、マルチタスクのためには、プログラム（とデータ）を物理メ

モリの異なったアドレスに配置しなければなりません。

そこで、MP/Mのプログラムのバイナリは、再配置可能なリロケータブルバイナリとし、再配置のための情報が付加されたものとなっています。プログラムの実行開始時に、MP/Mは必要な物理メモリを確保し、そこに、プログラムのバイナリをロードします。そのとき、再配置情報を参照し、バイナリ中のアドレス情報を適切に書き換えながらメモリ上に配置していきます。

MP/Mのリロケータブルバイナリ形式は非常に単純で、256バイト(ページ)単位で再配置可能です。8080のアドレス空間は16bitです。メモリへのロード時には、16bitアドレスの上位8bit(1バイト)だけを、書き換えます。プログラムバイナリ中のどのバイトがアドレスの上位バイトを保持しているかは、ビットマップで表現しています。プログラムコードの各バイトを1bitで表したビットマップを作り、書き換えるべきアドレス上位が入ったところを示す1bitを、1として表すのです。ロード時に、先頭となった物理番地に対していくらの値を加えるかは、対象であるバイトの中に書いておきます。この形式をCP/Mファミリでは「PRL(Page Relocatable)」と呼んでいます。

RPLとDDT

PRLは、MP/Mから出現し、開発ツールも同時に提供開始されましたが、実は、CP/M1.4の時から存在するDDTというデバッガは、PRLと同じ手法を使用していました。

DDTは、通常のCP/Mコマンドとして0x100番地からロードされ実行されるのですが、実行開始直後に、自分の本体をメモリの最後尾(OSの直前)に、PRLと同じ手法で最配置し、その後、デバッグ対象のプログラムを0x100番地からロードして、デバッグを開始していました。

MP/Mの開発元であるDigital Research社からMP/Mとともに供給される標準コマンドは、PRLに変更され、複数の人が、エディタを使用しながら、アセンブルを行う、ということが可能になっていました。

協調型のシステム

MP/Mのシステムは、ハードウェアにメモリプロテクション機構もなく、OSにも、アクセス権限やその管理についての考えがほとんどありません。かろうじて、ファイルの存在をほかのユーザからやや見えなくすること、Read onlyにすること、パスワードでファイルアクセスをさせない、という機能があるだけです。つまり協調型のマルチユーザ／マルチタスクシステムということです。1979年のクロック周波数2～4MHzの8bitのマシンをホストとして、同じチームで資源を共有するという考えなのでこの程度の実現が妥当だったと思われます。

ちなみに、2MHzの8080は、Intel Core i7 2600Kの「100万分の2.6」ぐらいの性能です[注1]。8086用のCP/M86をもとに、1981年にはMP/M86がリリースされていました。MP/M86は8086のセグメント機能を利用していたので、8080用のMP/Mよりは実現がすっきりしていましたが、機能的には8080版とほぼ変わりがありません。

MP/Mには、CP/Netというネットワーク拡張があり、CP/Mをクライアントにして、MP/Mサーバ上のファイルを操作できるようになっていました。

Concurrent CP/M

MS-DOSに負けかけていたDigital Research社は、MP/M86をもとにGUIを付け、シングルユーザでマルチタスクの「Concurrent CP/M」をリリースし、その後それは、「Concurrent DOS」となっていきました。

ちなみに、Z80pack[注2]というサイトから、Z80シミュレータと、CP/M2.2, MP/M IIなどが1つにパッケージされたものをダウンロードできます。Ubuntuでコンパイル＆実行したところ、CP/M2.2、CP/M3、MP/M IIどれも、とても調子いいです。SD

注1) 英語版Wikipedia:「Instructions per second」の項(http://en.wikipedia.org/wiki/Instructions_per_second)による。

注2) http://www.autometer.de/unix4fun/z80pack/index.html

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。

Event

enchantMOON予約開始イベント
「enchantMOON Night」レポート

㈱ユビキタスエンターテインメント（以下、UEI）が、手書きメモに特化した独自OS搭載コンピュータ「enchantMOON（エンチャントムーン）」の発売と販売価格の決定を発表した4月23日。同日の19時より、一般の方を招待してのタッチ＆トライイベント「enchantMOON Night」を開催した。

会場となった五反田のゲンロンカフェは19時の開始時には満員となっていたが、その後も続々と来場者が詰めかけ、急遽イベントを2回に分けて行うほどの盛況ぶりだった（来場者数は250名との公式発表）。

現状のタブレットの不満を解消する

詰めかけた聴衆はUEIの清水亮CEOの熱弁に耳を傾けていた。なかでも聴衆の共感を得ていたように思えたのは、「ここに来ている皆さんはiPadもしくはAndroidタブレットを使ったりしていて、何らかの不満があるんじゃないでしょうか」と問いかけたときだ。指では思ったように書けない。描いている線の追従性が悪い。enchantMOONはこの不満を解消することに注力されている。その最たる行為がOSレベルでのチューンナップだ。AndroidのアーキテクチャをベースにしながらenchantMOON専用のOS「MOONPhase」を開発。最短反応遅延は0.05秒を実現している。

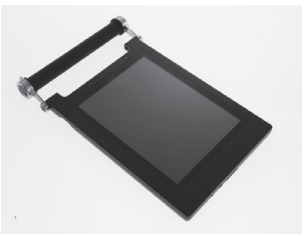
紙とペン――思考するための道具

なぜそこまで手書きにこだわるのか。

ITの最先端にいる人たちでも、紙（あるいはホワイトボードなど）とペンが手放せない状況を紹介。かたわらに高性能なPCがあるプログラマでさえ、紙とペンがあれば自然にそこで筆算してしまう。冗談めかして紹介されるが、そこには思考の過程ではPCよりも高速に処理できる手書きの良さが詰まっている。考えを手で描き、目で確認してまた考える。このプロセスが大切だということで、清水氏はこれを「思考の動脈と静脈」と表現。

▼ユビキタスエンターテインメント　代表取締役社長兼CEO　清水亮氏

▼enchantMOON



さらに、iPhone発表の壇上でスティーブ・ジョブズが「指」を「だれもが生まれながらに持つ世界最高のデバイス」と言ったことについて、Appleフリークでもある清水氏に「あれは間違い」と言わしめるほどにペンの重要性が説かれた（きっとジョブズのことだから、将来「あれは間違いだった」と撤回するのを想定したうえでの発言だったんじゃないか、と補足していたが）。

一方、紙となるディスプレイは表示解像度こそXGAとiPad miniと同等だが、ペン入力の分解能は815ppiという精細さを持つ。つまり、ディスプレイの5倍以上の解像度を持つデジタイザ（読み取り解像度5120×3840ドット相当）が搭載されているということで、非常に細かい文字も認識できる能力を持っている。

だれもがプログラマになれる

enchantMOONのユーザレベルを年齢で想定すると、3歳くらいでハイパーテキストの扱いができるようになり、6歳くらいでMOONBlock（後述）によるプログラミング、12歳にもなればJavaScriptを使い、16歳以上ならモジュールのコーディングまでできるようになるだろうとしている。ここで見られるように、enchantMOONはプログラミング教育での役割にもフォーカスしている。

enchantMOONに備わるビジュアルプログラミング機能の言語は「MOONBlock」と呼ばれる。だれでも簡単に、いつでも思いついたときにプログラミングできるようにと考えられたものだ。このMOONBlockの存在が他のタブレットデバイスと一線を画す、“思考と創造のためのデバイス”と感じさせる要因の1つだろう。

この機能の搭載は将来の展望にもつながる。出てきたキーワードは増井俊之氏の「全世界プログラミング」だ。ブロック型プログラミング言語によって老若男女問わず誰もが身の回りのものをプログラミングできるようにすることで、enchantMOONがホームネットワーク、スマートフォン、カーナビ、デジタルカメラなどといった機器とやり取りするハブになるという構想。この実現に向けて、まずは「MOONBlock」をオープンソースとして提供することが予定されている。

CONTACT　㈱ユビキタスエンターテインメント
URL http://www.uei.co.jp

Software

KDDIウェブコミュニケーションズ、クラウド電話API「Twilio」の提供を開始

㈱KDDIウェブコミュニケーションズは、4月17日、米国Twilio社が提供するクラウド電話API「Twilio」（トゥイリオ）の、日本での提供を開始した。

クラウド電話APIとは、Webサイトやアプリケーションに実装することで、インターネット上から電話をかけたり、電話を受けたりができるようになるもの。今回提供されるTwilioでは、電話の送受信機能、アップロードした音声を再生する再生機能、文字列を音声に変換できる音声合成機能、ネットワークを通して音声データを送受信するVoIPが利用可能になる（メッセージ送受信機能は提供開始予定）。Twilioは利用開発者数15万人、サービス提供国40ヵ国以上の実績を持つサービスで、KDDIウェブコミュニケーションズが業務提携を結び、サービス提供に必要な各種設備や日本国内の通信事業者との接続を行うことで国内提供開始に至った。これにより、オンラインとオフラインを連携したサービスを短期間に低価格で実現できるとしている。

Twilioの利用には電話番号利用料として月額490円（税込）と、従量制の通信料が必要となる。

CONTACT ㈱KDDIウェブコミュニケーションズ
URL http://kddi-web.twilio.jp

Service

ブレイン、ホスティング管理画面から社内SNS「CHITCHAT!」のユーザー括登録ができる機能を提供開始

㈱ブレインは、㈱ハイパーボックス、㈲ネットグルーヴワークスと提携し、両社が提供するホスティングサービスから、社内向けSNS「CHITCHAT!」のユーザ登録が一括でできるしくみを4月19日より提供開始した。

CHITCHAT!はブレインが提供する社内向けSNSで、組織内のホウ・レン・ソウを効率的かつ効果的に行うことを目的に開発された。操作画面はタイムライン形式を採用し、グループ設定、画像投稿、チャットといった機能を完全無料で提供している。

これまで、社内SNSのユーザ登録作業は1件1件個別に行っており、規模の大きい企業は登録作業に大きな手間がかかっていた。そこで、社内SNSが同一ドメインのメールアドレスで運用されている点に着目し、メールアドレスを管理するホスティングの管理画面とのシステム連携を考案した。このAPI連携により、社員にメールアドレスを発行するのと同様に、ホスティングの管理画面から同一ドメインを持つ社員全員に対して一括でSNSのユーザ登録が可能になった。社内SNSとホスティングのAPI連携は業界初の試みだという。

CONTACT ブレイン㈱
URL http://www.blayn.co.jp

Service

ミックスネットワーク、大規模向けWebサイト構築運営プラットフォーム「SITE PUBLIS Multisiteライセンス」を提供開始

㈱ミックスネットワークは、複数のWebサイトおよびWebサービスを一元管理できる大規模向けWebサイト構築運営プラットフォーム「SITE PUBLIS Multisiteライセンス」を、4月30日より提供開始した。

同サービスは、複数サイトにおける運用の一元化、レイアウト／デザイン、コンテンツ資産やデータ活用などを集中管理できる機能を搭載している。数多くの製品ポータルサイトや海外法人を持つ企業、多くの関連法人や外部組織を持つ官公庁、自治体などでの利用を想定している。

同サービスは、各サイトで分断されていた構築／運用を横断的に行え、個別に集積されていたマーケティングデータやコンテンツ資産も集約管理することができる。またブロック部品を組み合わせてWebサイトを編集する独自の「ブロック方式」で、スマートフォン、モバイル、タブレット、PCなどのマルチデバイスへの対応を複数サイト全体で行える。

価格は、1ライセンス（10サイト）で1,500万円（税別）～となっている。

CONTACT ㈱ミックスネットワーク
URL http://www.micsnet.co.jp

Service

IDCフロンティア、共同でソリューションを開発／提供する販売パートナー制度を展開

㈱IDCフロンティアは、パートナーとサービスやソリューションを共同開発し、データセンターおよびクラウドサービスの販売、提供を行う「IDCフロンティアパートナープログラム」（以下、IDCFパートナープログラム）を4月1日より開始した。

同プログラムは、IDCフロンティアがパートナー企業と共同で、同社の各種サービスとパートナー企業の機器／サービスなどを組み合わせたソリューションを開発し、販売を行うというもの。すでに、アイウェイズ㈱のクラウドとのデータ連携を行う「DataBridge」などのサービスが展開され始めている。

同社はパートナー企業に対して、システムの構成例も含めた販売案件情報の蓄積、成功モデルの共有、サーバなど検証環境の無償提供といった支援を実施していくという。同社が持つITインフラ提供に関するノウハウを共有することにより、パートナー企業はデータセンターとクラウドサービスを組み合わせた拠点間分散やハイブリッド構成などの高度なソリューションや付加価値サービスを提供することが可能となる。

制度の開始当初は独立系の開発会社やシステムインテグレータなどを中心に約70社で展開する。データセンターおよびクラウドサービスにパートナー各社のさまざまなソリューションを加えて提供することで、ユーザのITインフラの革新とより一層の販売拡大を狙う。

■IDCFパートナープログラムの特徴

①パートナー各社とのサービスやソリューションの共同開発／提供
②案件ごとにリセールまたは取次形態の選択が可能な柔軟な契約形態
③パートナー企業へサービスの検証環境無償提供およびエンジニアの技術支援
④データセンターとクラウドを組み合わせたハイブリッド構成の販売／取次が可能

■パートナー企業一覧

http://www.idcf.jp/cloud/partner/

CONTACT ㈱IDCフロンティア
URL http://www.idcf.jp

Topic

The Linux Foundation、「OpenDaylightプロジェクト」の立ち上げを発表

非営利団体The Linux Foundationは4月8日、「OpenDaylightプロジェクト」の立ち上げを発表した。

このプロジェクトは、コミュニティがさまざまな企業の力を借りながら、Software Defined Networking（SDN）に関するよりオープンで透明性の高いアプローチの構築を目指す共同開発オープンソースプロジェクト。次の企業がプラチナおよびゴールドメンバーとしてプロジェクトに参加している。

■参加企業

- Big Switch Networks
- Brocade
- Cisco
- Citrix
- Ericsson
- IBM
- Juniper Networks
- Microsoft
- NEC
- Red Hat
- VMware

これらの企業がソフトウェアや開発人材の提供を通じて協力しあい、今後のオープンなSDNプラットフォームを構築していく。

具体的には、2013年の第三四半期にOpenDaylightの最初のコードをリリースすることを予定している。また、オープンコントローラ、バーチャルオーバーレイネットワーク、プロトコルプラグイン、スイッチデバイスの改良といった寄付やプロジェクト展開も予定している。

いくつかの企業や団体がすでに開発貢献や主要技術のオープンソース化の意向を示しており、これらの開発貢献や主要技術はOpenDaylightの技術運営委員会（OpenDaylight Technical Steering Committee）によってレビューされ、採用が決定される。

同プロジェクトには、今後も企業や学校、個人から製品やコードが提供される予定。また、オープンソースの開発コミュニティからのコードの提供は常に歓迎しているという。

CONTACT The Linux Foundation
URL http://www.linuxfoundation.jp

Software

キヤノンITソリューションズ、「ESET Endpoint Security for Android」のモニター版の提供を開始

キヤノンITソリューションズ㈱は、5月7日より、法人で利用可能なAndroid用総合セキュリティプログラム「ESET Endpoint Security for Android」と、Android端末を管理できるクライアント管理用プログラム「ESET Remote Administrator」のモニター版を提供開始した。

「ESET Endpoint Security for Android」は、企業での業務端末として利用が増えているAndroid OSに対応したESETの総合セキュリティプログラム。Android OSを標的としたマルウェアからの保護だけではなく、盗難対策、パスワード保護などの機能を備えている。

また、企業内のサーバにクライアント管理用プログラム「ESET Remote Administrator」を導入することで、Android端末の各設定やログ情報の閲覧、オンデマンド検査の実行、ウィルス定義データベースのアップデートといった操作をシステム管理者からリモートで実行することができる。WindowsやMac、Linuxといったほかの OSと合わせて一括管理が可能。

これらのプログラムを次の期間、モニター版として提供する。

■モニター版プログラムの提供について

提供期間：
2013年5月7日～2013年5月24日
評価レポート受付期間：
2013年5月7日～2013年5月31日
プログラム申込方法：
2013年5月7日より次のURLから申込受付開始
http://canon-its.jp/eset/eesa/

ESET Endpoint Security for Androidの動作環境は、対応OSがAndroid 2.0/2.1/2.2/2.3/3.0/3.1/3.2/4.0/4.1、メモリは1MB以上。microSDカードなどの外部媒体へのインストールには対応していない。

クライアント管理用プログラム「ESET Remote Administrator」のモニター版プログラムの動作環境については、http://canon-its.jp/eset/eesa/ を参照のこと。

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

Software

グレープシティ、日本語技術サポート付きjQueryウィジェットとコンポーネントセットをサブスクリプションで提供

グレープシティ㈱はプラットフォームごとに幅広い分野のコンポーネントを数多く収録したスイート製品「ComponentOne Studio」シリーズの新バージョンおよびjQueryウィジェット集「Wijmo Professional」をサブスクリプション方式にて5月29日に発売する。

ユーザライセンス価格はEnterpriseで、初回費用が157,500円（税込）。更新料は初回費用の40%。Webアプリケーションを開発する場合は別途コアサーバライセンスが必要。

ComponentOne Studio 2013J

ComponentOne Studioは、WindowsフォームからHTML5対応のASP.NETアプリケーションを開発したり、WPFなどのプラットフォームで業務システムを開発したりできるコンポーネントスイート製品。データグリッドや帳票、チャート、各種UI部品などをバランスよく収録している。

新バージョンではパッケージ販売を廃止し、サブスクリプション方式を採用した。定められた契約期間中の無償バージョンアップや日本語による技術サポートを無制限に受けることが可能となる。パッケージ製品と比べて機能追加や新環境への対応、不具合修正がこれまでよりも格段に早くなるという。

ライセンス体系が異なるだけでコンポーネントのクラス構造や機能はこれまでと変更はなく、上位互換を保証する。

Wijmo Professional 2013J

HTML5アプリケーション開発用の新製品「Wijmo Professional」も同時にサブスクリプション方式で発売する。同製品はjQuery UIにはない高機能ウィジェット集。jQuery自体はオープンソースとして利用可能だが、同製品ではウィジェットを販売するだけでなく、日本語による技術サポートや日本語ドキュメントといったサポートサービスも提供する。

CONTACT グレープシティ㈱
URL http://www.grapecity.com/tools

Letters from Readers

機械学習恐るべし

4月に行われた第2回電王戦でついにコンピュータが人間を圧倒しました。機械学習のおかげで、開発者自身の将棋の強さに関係なく、コンピュータが独自で強くなっていけるとのこと。囲碁だけはまだプロ棋士には及ばないレベルと言われていますが、この様子では囲碁でもコンピュータ優勢の時代が来るのは、そう遠くないかもしれません。

2013年4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 裏口からのプログラミング入門

異業種からソフトウェア開発の世界に入り、活躍している若手エンジニア、プログラマの方々に、どのようにプログラミングを勉強し、仕事に活かせるようになったのか、紹介してもらいました。4月から入社した新人エンジニアの方、今後の学習の参考になりましたか?

IT業界とは関係のない世界から転身された方の体験談から、ひょっとしたら自分にもできるかもしれないという夢や自信をいただいたような気がします。

宮崎県／maehrmさん

人の経験談や勉強法はとても参考になるし、モチベーションアップにつながる。

東京都／chimさん

プログラミング雑誌としては「裏技」の記事ですね! 意表を衝かれました。

熊本県／しゅさん

なかなか真似はできないとも思われますが、おもしろく拝読しました。

富山県／QKobさん

IT業界には文系エンジニアも多いでしょう。技術をどのように身につければいいか悩んでいる人もいると思います。今回の特集を参考にしてください。

第2特集 ソフトウェア開発に効くSmall Objectをご存じですか?

お馴染みトム・エンゲルバーグ氏によるオブジェクト指向再入門の記事をお送りしました。

オブジェクト指向の目的やいいところがわかりやすく書いてあると感じました。

福岡県／Ashleyさん

最近の関数型ブームに逆らってオブジェクト指向の内容が読めてよかった。

東京都／藤田さん

どんなふうにシステムを構築していくことが理想なのか、そういう目標地点がわかり、ためになりました。

兵庫県／ポーさん

オブジェクト指向は昔から叫ばれつつも、なかなか身につけられない技術の1つですね。本特集で実践の勘どころがつかめたでしょうか。

特別企画 Luminous Studioが変えるゲーム開発の舞台裏

スクウェア・エニックスで利用されているゲームエンジンLuminous Studio。Skeedとの協業でさらに進化しました。その機能について解説しました。

よく知らなかったのだが、ゲームエンジンって描画ロジックのライブラリかと思ってたら、リソースDBなんですね。

神奈川県／速水さん

スクウェア・エニックスのイメージが変わった。

東京都／hiddenさん

大規模オンラインゲームともなると開発資産の管理／共有が大きな課題となります。それをどう解決しているのか、ユーザ側からは想像のつかない領域ですが、その一端が垣間見れました。

特別企画 ゲーム開発の舞台裏座談会

Skeedの金子勇氏、スクウェア・エニックスの橋本善久氏などLuminous Studioの開発に関わったエンジニアによる座談会の様子をお伝えしました。

技術の問題点などが話題にあがっていて楽しく読めました。

滋賀県／東川さん

ゲーム開発については知識がありませんでしたが、読んでいて楽しかったです。

東京都／佐伯さん

大規模開発ならではの課題に取り組む様子が生々しく語られていました。巨大なデータの管理など、ゲーム開発には業務システム開発とは違った悩みがあるものだと感じました。

連載

「温故知新ITむかしばなし」のPrologのお話が興味深かったです。ICOTの成果が低いのは目標と具体的な成果の乖離を上手に説明できなかったからではないでしょうか。並行処理は大切な機能なので、今後その成果が活かされて再評価されるといいですね。

大阪府／出玉のタマさん

世界で評価されているのに、国内での評価が低いというのは、まるで初期のRubyのようですね。利用事例もありますし、今後、Rubyのように逆輸入のようなかたちで国内でも評判になるかもしれません。

フリートーク

総集編のPDF化がすごくうれしいです。SDは、今は使わないけど後から必要となる内容が結構あるので、気になる記事がある時は買うようにしています。しかし、保管場所に苦慮していました。こうしてPDF化されるとわかっていると逆に安心して講読できるようになります。

東京都／杉原さん

5月25日には創刊号からの記事を収録した第2弾も発売しますので、ぜひ手に取って下さい。しかし、電子版が売れることで、紙のバックナンバーが捨てられていくのは、少しさみしかったりもします。

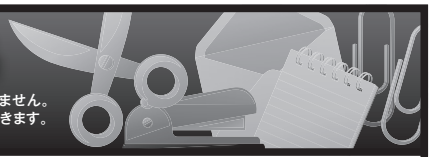
最近入社してから初めて部署が異動になりました。新しい現場は想像以上に自分の知らないことだらけで悪戦苦闘な日々です。貴誌を読んでいたおかげで普通に仕事していたなら知らなかったこともあり、助けられております。しかし、手を動かさずに読むだけだったので、あまり理解しきれていないということも痛感する今日この頃です。

兵庫県／yoneさん

担当も勉強のために記事の内容を実機で試してみようと思いつつ、実際にはできていません。やはり、自分のものとするには実践が必要ですね。

エンジニアの能率を高める一品

仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。



dicapac W-P2

2,992円（税込）／大作商事　http://www.daisaku-shoji.co.jp

※製品にスマートフォンは付属しません。

もうすぐ梅雨の時期ということで、雨の日も安心なスマホ用防水ケースを紹介します。dicapac W-P2はどのメーカーのスマホにも使えるビニール製の防水ケースです（縦128mm×横65mm×厚み13mm以内が快適に使える目安）。防水という面では申し分なくシャワーをジャバジャバ浴びせても（写真）、水にドブンと浸けてもスマホは濡れません。さすがに水中では無理ですが、ケースと手が濡れている程度なら、ケースの上からタッチパネルの操作ができます。ケースの中に空気が入っているとうまく操作できないので、しっかり空気を抜くことが効率よく使うポイントです。これなら台風の日でも屋外でスマホを使えます。首からかけるストラップが付属しているのですが、雨の日は傘をさしていて手がふさがっていることが多いで、これは便利だと思いました。（読者プレゼントあります。16ページ参照）

▲シャワーでびしょびしょ濡らしても、タッチパネルの操作は可能

4月号のプレゼント当選者は、次の皆さまです

①ロジクールタッチマウスt620 ..... 神奈川県　田中孝治様
②Boombero Wireless Speaker ..... 愛知県　加藤謙一様

★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
July 2013

2013年7月号
定価1,280円　176ページ

6月18日発売

［第1特集］データの目利きになりたい!

機械学習+データ分析「超」入門

なんらかのITシステムを運用していれば、自然にログが膨大な量に嵩んでいきます。Googleサーチエンジンレベルのビッグデータではありませんが、一般的なエンジニアでも身の回りにあるデータから意味を見いだすために、統計的な分析をすることが多くなっています。フリーの統計ツール「R」やテキストマイニング、データマイニング、Apache Mahout、fluentdなどなど、「データをうまく扱う技術」をわかりやすく解説します。

［第2特集］自分の定規を持っていますか?

原因追及「ベンチマーク」テクニック

サーバやネットワークが遅い、ボトルネックはどこに? そんなとき各種ベンチマークツールが役立ちます。適用の勘所、データの読み方などチェックポイントを紹介します。

［一般記事］

サーバラック・ケーブリング入門

※特集・記事内容は、変更になる場合もあります。あらかじめご了承ください。

休載のお知らせ

「Androidエンジニアからの招待状」（第38回）、「システムで必要なことはすべてUNIXから学んだ」（第10回）、「IPv6化の道も一歩から」（第7回）は都合によりお休みいたします。

SD Staff Room

●ちょうど5月号の発売時に後記を書いている。特集や新連載の反応を見るに、といってもTwitterをモニターしているだけだけど、けっこう手応えがあってうれしい。特に本は重要。いまをときめく若手エンジニアがタネンバウム先生の本を勧めている。難解で高価格だけど、それだけの価値がある本は長生き。そして影響を与え続ける。（本）

●テレ東の金曜夜が熱い! 「みんな! エスパーだよ! 」は夏帆のお色気シーンや真野恵里菜、神楽坂恵といった完璧な女子陣に加え、イジリー岡田、安田顕、栗原類とツボな方々が勢揃い。「ヴァンパイア†ヘブン」は大政絢と本田翼! のコケティッシュな魅力がハンパない。殿（しんがり）は牙狼の新シリーズ。週末の楽しみが増えた。（幕）

●ついにenchantMOONが正式にリリースされました。スマホやタブレットの選択肢は増えましたが、自分にはどれも同じに見えてイマイチ。が、久々に心が躍るデバイスが出てきた感じです! もちろん5月号からはじまった連載「enchant」でも清水氏に語っていただきますよ。乞うご期待!!（キ）

●総集編の制作で大量のテキストデータを扱うときに正規表現がとても役に立ちました。5月号の特集は自分にとってタイムリーなネタでした。正規表現はエンジニアでなくても役立つので、インターネットやオフィスツール同様、教育現場でコンピュータ関連の一般教養として学ぶ場があってもいいように思います。（よし）

●父親の調子が悪くなり病院に行ったら即入院に。当初は1週間入院すれば大丈夫ということでしたが、数日後隠れ病が原因でかなり危険な状態となり、母親とあわてて病院に駆けつけました。幸い回復しましたが当然入院生活は長引くことに。親孝行しないと、というか親不孝しないようにしないといけないなと実感中です。（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
2013年6月号

発行日
2013年6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。