Special Feature 01
RDBかNoSQLか？
Special Feature 02
プロキシサーバのしくみ
2014 March
03
[ソフトウェア デザイン]
OSとネットワーク、
IT環境を支える
エンジニアの総合誌
Software Design
2014年3月18日発行
毎月1回18日発行
通巻347号
（発刊281号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
1,280円

特集一、
Special Feature 01

RDBとNoSQL どちらを選びますか？

データベースの諸問題

真っ当に考えるDBの鉄則

Problems of database

特集二、
Special Feature 02

ネットワークエンジニアのための
プロキシサーバの教科書
基本機能からリバースプロキシまで
構築と運用の鉄則

短期集中連載

Mac as a desktop service
さらに踏み込む
Mac OS Xと仮想デスクトップ

Decide if you would like to choose RDB, NoSQL.

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63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。

Intel XDK

Intel製HTML5アプリ開発ツール「Intel XDK」

「Intel XDK」はIntel社が開発・公開している無償利用可能なHTML5アプリケーション開発ツールです。マルチプラットフォームに対応しており、HTML5/CSS3/JavaScriptによってWebアプリケーションを開発できるだけでなく、作成したアプリケーションを各モバイルプラットフォーム用のネイティブアプリに変換して提供できるという特徴を持っています。Intel XDKの前身となったのはAppMobi社が提供していた「jqMobi」をはじめとする一連のHTML5開発ツールです。Intelは2013年2月に同社からこれらのツール群を買い取り、統合的な開発環境としてまとめたうえで同4月にIntel XDKを発表しました。

Intel XDKではおもに次のような機能が提供されます。

- 高機能エディタ
- UI編集ツール
- デバッグ用コンソール
- 多数の機種に対応したデバイスエミュレータ
- モバイル向けにカスタマイズされたJavaScript UIライブラリ
- クラウドベースのビルドシステム

Intel XDKで開発したアプリケーションは、デバイスエミュレータを使用することによって実機を用意しなくても動作確認を行うことができます。エミュレータではデバイスの場所や向き、ネットワーク接続状態などを細かく指定できるため、実機に近い条件でテストすることが可能です。

ネイティブアプリへの変換は、Apache Cordovaのビルドシステムによって行われます。Apache Cordovaはモバイルアプリ開発フレームワークであるPhoneGapのオープンソース版で、HTMLベースのWebアプリケーションから、iOSやAndroid、Windows Phoneをはじめとした各種モバイルプラットフォーム向けのネイティブアプリを構築することができます。構築したアプリは、各プラットフォームのアプリストアでの配布・販売にも対応します。

さらに進化した「Intel XDK NEW」も登場

Intel XDKの公式サイトでは、すでに次期バーションとなる「Intel XDK NEW」も公開されています。Intel XDK NEWでは、従来のバージョンに対して次のような拡張が行われています。

- オープンソースのコードエディタ「Brackets」の採用
- 多くのフレームワークに対応した新しいUIビルダー
- 「Apache Ripple」ベースの新しいエミュレータ
- USB接続したAndroidデバイスのリモートデバッグやJavaScriptプロファイリング機能
- ユーザインターフェースの刷新
- node-webkitへの置き換えによってJavaとChromeへの依存性を解消
- Cordova 2.9のサポート

上記に加えて、ネイティブアプリのビルドシステムのランタイムエンジンとして「Crosswalk」を選択できるようになったことも大きな変更点です。CrosswalkはオープンソースのWebアプリケーションランタイムで、現在はAndroid版とTizen版が公開されています。Googleがオープンソースで開発している新しいレンダリングエンジンの「Blink」を採用することによって、極めて高速なレンダリングやWebGL対応などを実現しているという強みがあります。

また、CrosswalkはCordova APIをサポートしており、Cordova APIを使って作成されたアプリケーションはそのままCrosswalkのランタイム上で実行することができます。しかもIntel XDK NEWでCrosswalk向けのビルドを行った場合、自動的にCordova APIを取り込むため、Cordova向けのAPI追加などは必要ないとのことです。

Intel XDK NEWは、HTML5ベースのアプリ開発者の選択肢をさらに拡げるものになるでしょう。なおIntelでは、従来のIntel XDKの提供を2014年2月末で終了させ、その後はIntel XDK NEWをIntel XDKの名称で提供していく方針を表明しています。SD

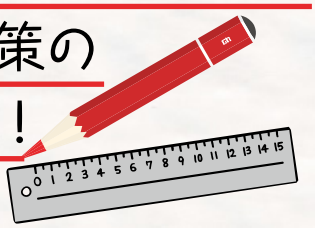
Intel XDK
http://xdk-software.intel.com/

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。

あなたを
合格へと導く
一冊があります！

効率よく学習できる
試験対策の
大定番！



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

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

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

濱本常義ほか　著
A5 判／ 592 ページ
定価 3,360 円（3,200 円＋税）
ISBN978-4-7741-6178-5

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

内田保男ほか　著
A5 判／ 512 ページ
定価 3,360 円（3,200 円＋税）
ISBN978-4-7741-6101-3

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

大滝みや子　著
A5 判／ 488 ページ
定価 2,394 円（2,280 円＋税）
ISBN978-4-7741-6112-9

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

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

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

Contents

Software Design Mar. 2014

1

第1特集 Special Feature 01

データベースの諸問題

RDBとNoSQL どちらを選びますか?

真っ当に考えるDBの鉄則

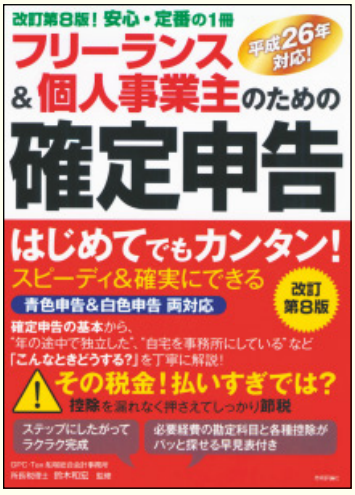
技術評論社の 確定申告本

平成26年3月締切分

初めてでも大丈夫！ マネして書くだけ確定申告 平成 26 年 3 月締切分

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

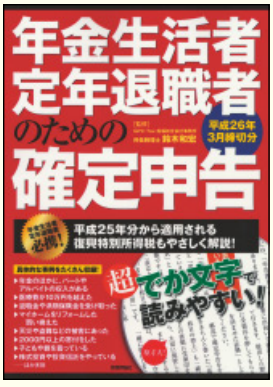
家族が働いていたり副業があるために確定申告が必要なサラリーマンのために、確定申告に必要な書類の作成方法を、簡単にわかりやすくまとめました。サラリーマンだけでなく、フリーランスや不動産オーナー、年金で生活している方などが確定申告を行う場合の書類の作成方法についても、個別のケースごとに詳しく解説しています。本書を参考にすれば、該当するケースを探して、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などに、特におすすめしたい1冊です！



フリーランス＆個人事業主のための確定申告 改訂第 8 版

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

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



年金生活者定年退職者のための確定申告 平成 26 年 3 月締切分

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

世界一ラクにできる確定申告

全自動クラウド会計ソフト「freee」で
仕訳なし・入力ストレス最小限！

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

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

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

Contents

Software Design Mar. 2014

技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

http://gihyo.jp/dp

GIHYO Digital Publishing 未来の"普通"を、今。

Google play

amazon.co.jp

※販売書店は今後も増える予定です。

法人などまとめてのご購入については
別途お問い合わせください。

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

ソフトウェア・デザイン［目次］

Contents

Software Design Mar. 2014

3

Column

Development

OS/Network

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

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

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

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

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

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

毎月18日発売

消費税アップによる
価格変更前の
ラストチャンス！
（4月号からご購入）

年間定期購読のご案内

富士山マガジンサービス版

年間購読なら
割引料金で
購読できます！

全国どこでも
直接お届け
しています！

1年購読（12回）

14,580円（税込み、送料無料）1冊あたり1,215円（5%割引）

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

［月額払い］
スタートしました！

デジタル版Software Design

Webで購入

家でも
外出先でも

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

お申込み方法

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

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

Software Design plus

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

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

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

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

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

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

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

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

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

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

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

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

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

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

サーバ/インフラエンジニア養成読本 仮想化活用編
Software Design編集部 編
定価 1,980円+税　ISBN 978-4-7741-5038-3

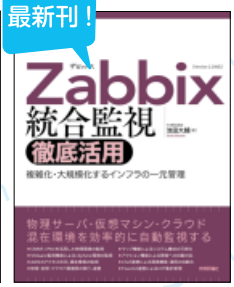
業務に役立つPerl
木本 裕紀 著
定価 2,780円+税　ISBN 978-4-7741-5025-3

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

最新刊！

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

最新刊！

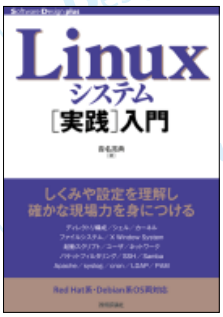


TIS㈱ 池田 大輔 著
B5変形判・384ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6288-1

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

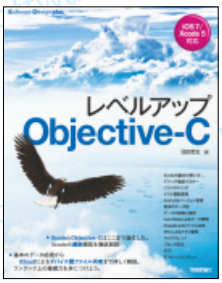
久保田 光則、アシアル㈱ 著
A5判・384ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-6211-9

ニコラ・モドリック、安部 重成 著
A5判・336ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-5991-1



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

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



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

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

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

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

水野 操、平本 知樹、神田 沙織、野村 毅 著
B5判・128ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-5973-7

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

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

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

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10回 自宅ラックのススメ、別府出張編

イラスト：髙野涼香

はじめに

自宅にラックやルータ、そしてサーバなどを設置して遊んでいると、そのうち、インターネット接続で、BGPを使用してみたいと思うこともあるでしょう。ということで、今回は少しおもしろいことができる集まりを紹介してみたいと思います。

BGPとは……?

ネットワークエンジニアならば、BGP（Border Gateway Protocol）は避けて通れない技術の1つになるでしょう。BGPとは、自律的にネットワークを構成するために使用する技術です。インターネットの世界で唯一の番号として、組織に1つもしくは複数の識別番号（AS番号）があり、その番号に基づいて経路情報の交換が行われます。具体的には、複数のネットワークに接続し、互いにアドレスブロック（経路情報）を交換し合い、ルーティングが行われます。この際に経路の選択の仕方により、負荷分散・経路最適化・ネットワークの冗長化などを行うことができます。紙面の都合上詳しくは書けないので、JPNICの「インターネット10分講座：BGP[注1]」や「Geekなページ：BGPを解説してみた[注2]」などを読んでいただければわかりやすいと思います。

IHANet（いはねっと）

筆者はグローバルAS番号を持っています。インターネット上で、BGPを触っていたり遊んでいたりしています。実際にAS番号を取得してインターネットに接続するのは、非常にハードルが高く、BGP接続が可能な事業者2社以上と接続する予定があるといった確認、使用する機材といった審査もあります。そこでどうしてもBGPを使ってみたい方に向けて、草の根的に構成されたBGPネットワークがあります。この集まりはIHANet[注3]（いはねっと）というユーザグループで、国内にあります（写真1）。これに参加することで、インターネットとはひと味違うBGP技術やつながりを楽しむことができます。また、実際に各ポリシーに基づき、接続・運用することもできます。実インターネットと直接BGPで接続されていないことから、コミュニティの外に迷惑をかけることがありません。ちなみに筆者は両方に接続していることから、間違えると迷惑をかけてしまうことは十分にあります（笑）。

IHANetでは、プライベートで利用可能なAS番号を用いて、BGP4＋プロトコルを使用し、相互に接続してIPv6の経路交換を行っています。接続にあたり、お互いのネットワークをポリシーに基づき接続されています。ここは、誰でも参加できますし、情報交換や意見交換などは、IRCやメーリングリスト上で行われています。参加を希望する方は、

注1） https://www.nic.ad.jp/ja/newsletter/No35/0800.html
注2） http://www.geekpage.jp/blog/?id=2009/2/20/
注3） http://www.ihanet.info/

一度のぞいてみてください。インターネットを構成するのと同じように、AS番号の割り当て、RR（Routing Registry）の提供、LG（LookingGlass）など、実際に運用されており、一通りのことは試してみることができます。

また、ネットワークにかかわる団体として、JANOG[注4]などもあります。2014年は別府でJANOG33 Meetingを開催しました（筆者も参加しました）。ネットワークエンジニアや技術に興味のある方々が集まることから、JANOGミーティング終了後の翌日など、みんなで集まって、IHANetメンバーが集まることが多く、勉強会やピアリングパーティー（お互いのネットワークを接続し合う）といったこともやっています。昔のインターネットもこういう感じで人のつながりがあり、ビールを飲みながらつながっていた時代もありました。beer and peerとか言われたりもしていますし、IX（Internet eXchange）などではまだこの文化は若干残っています。

BGPって何が楽しいの？

BGPは、技術的要素と、コミュニティー的な要素があります。インターネットは、お互いの組織・団体が各々のポリシーに基づき、協調・構成されたネットワークで、各組織の配下のネットワークはコントロールできても、インターネットは誰のものでもなく、各組織・団体によって、運用されています。すなわち、お互いが自立的に協調し合い、その方針に基づいたルールをもとに、技術的に接続します。何が言いたいかというと、人が集まり、組織を形成し、それを形にしたのがBGPだということです。こうやって、人同士がつながり、意見や情報交換などをしつつ、新しい世界ができていきます。いわゆる政治やコミュニティの上に成り立っている技術といっても過言ではありません。それを上手に転がすのがおもしろいものだと考えています。

注4）http://www.janog.gr.jp/

終わりに

BGPを触る場合は、BGPという技術が先か、コミュニティが先かという問題はありますが、結果として人がつながり、その上でインターネットというネットワークが形成されているということを忘れてはいけません。ということで、仲良くみんなで協調し合って構成されているものが、インターネットですね。実ネットワークでは難しくても、こういう集まりがあるということを皆さんに知っていただきたいです。そのきっかけにしてみてください。1人で寂しい思いをしている人も、表に出てきて楽しむのも、エンジニアの生き方の1つだと思います。

機材が自宅にそろったら、ますはこういう世界・活動を知ってもらいつつ、歩んでみてください。昔のインターネットが形成されていた時の雰囲気にも近いと思います。

さて、次回は最終回。ご期待あれ。SD

▼写真1　IHANet Meetingの様子

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

この個所は、雑誌発行時には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。

DIGITAL GADGET

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

Volume 183 >>>>>

流行だけでない ウェアラブルとモノのインターネット

身につける デジタルデバイス: ウェアラブル

世界最大の家電見本市International CES（Consumer Electronics Show）2014が、米国ラスベガスで2014年1月7～10日の4日間開催されました。

International CES
http://www.cesweb.org/

多くの家電製品の新製品が発表される中でとくに注目を浴びていたのは、4K高精細テレビ／カメラ、モノのインターネット（後述）、ウェアラブル（グラスウェア）の分野です。そのほかにもロボットやドローン（小型リモコンヘリコプター）、車載機器、車載アプリ、ネット接続自動車などが注目され、なかでも車業界の展示は過去最大規模とのことでした。

モノのインターネット IoT

IoT（Internet of Things）は、IPアドレスが枯渇し、IPv6の必要性が叫ばれていた頃にも話題に上りました。コンピュータやデジタルデバイスがネットにつながるのはあたりまえですが、従来ネットやデジタルとは関係なかったモノもネットにつながることによってさまざまな情報を得ることができます。

モノがネット接続されることによって、離れたところから操作できたり、離れたところからモノの状態を知ることができます。また、人の手では一度に1つしか操作、認知できないモノも、多数のデバイスを一度に操作したり、一度に状態を把握できるようになります。さらにモノとモノとが協調して何か物事を行っ

水泳用のウェアラブルデバイス「Instabeat」
http://www.instabeat.me/

Google Glassの対抗馬。メガネでスマートフォンを操作する「GlassUp」
http://www.glassup.net/

スマートフォンで操る紙飛行機「PowerUp 3.0」
http://www.poweruptoys.com/

※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。

たり、広範囲での状態／情報を知覚できるようになります。

単に便利になるだけではなく、人やモノの行動や生活の情報を蓄積することによって、今までにないサービスが実現できるようになるのです。

今回のCESではとくにCPUチップメーカーであるIntelのIoTへの力の入れようが目立ちました（http://makeit.intel.com/）。また最近、家庭内の冷暖房をインテリジェントにコントロールするデバイスやネットワークに接続される火災報知器を発売するNESTが、32億ドルでGoogleに買収されました。ネットビジネスが主な領域であったGoogleが、確実に現実世界の「モノのインターネット」に進出してきた気配がうかがえます。

IoT浸透の背景として、さまざまなセンサーデバイスがより安価に、長寿命で動作するようになり、今まで考えられなかったような「モノ」が真の意味で「スマート化」したことがあります。想像してみてください。薬瓶がネットにつながるデバイスになったとき、車のワイパーが広域ネットにつながったとき、冷蔵庫のビール棚がECサイトとつながったとき、トイレの電灯がネットにつながったとき……。身の回りにあるものがネット化したとき、デジタル化したときに何が起こるのかを。

その一方、プライバシーの課題も忘れてはなりません。これからは便利さの享受と、個人情報の開示とのバランスが重要になってくることでしょう。

今度こそ、本気のウェアラブル

Google Glassの登場のおかげ（せい？）かどうかわかりませんが、CES 2014では数多くのメガネ形ウェアラブルデバイスが台頭してきました。Google Glassもその他のメガネ形デバイスも、デバイスそのものだけでは完結せず、ネットと接続してどういう利用体験ができるのかが焦点になります。また、音声認識やタッチ操作、ウィンクなど目の動きによる操作など、操作性に関する事柄が使い勝手に大きく影響します。

また、公共の場でのカメラ撮影によるプライバシー問題をはじめとする社会的な要素など、さまざまな課題と可能性も積み重なっています。そのうえ、多くの人々に浸透するためにはファッション的な要素も不可欠でしょう。視力矯正のためのメガネはいまやファッションの一部となっている一方、レーシックなどの手術をしてまでメガネを外したいと考えている人もいます。

ウェアラブルという概念は、1992年頃、もともとはペースメーカーなどを身体に埋め込むことができる（ウェアラブル）という言葉からきたものです。マサチューセッツ工科大学（MIT）のメディアラボで最初に提唱されました。

言葉どおりに考えると、“ウェアラブル＝着ることのできる”という意味です。今では身につけることができるデジタルデバイス、身につけたままで操作できるデジタルデバイス、身体と一体化して情報を取得したり、身体性を拡張するデジタルデバイスと広く捉えられます。

人が常に身につけているものを考えてみると、何があるでしょう?

衣服、帽子、靴、腕時計、ネックレスやブレスレットのような装飾品、指輪、メガネ、カツラ、入れ歯、補聴器、心臓のペースメーカー、人工関節などなど。私たちが常に持ち歩き、一日平均150回は使っていると言われているスマートフォンも、ウェアラブルデバイスに近い存在なのかもしれません。ウェアラブル業界は、今後5年で190億ドル市場になることが予想されています。

これからのウェアラブルデバイス、これからのモノのインターネット

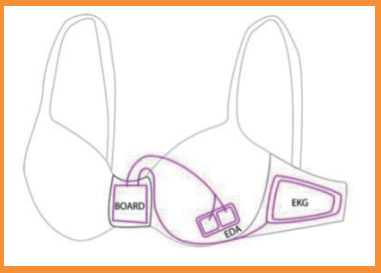
一度美しい映像を見てしまった人間は、すぐにその美しさに慣れてしまいます。Retinaディスプレイほどの解像度は必要ないかも?と思いつつも、しばらく使ってその細かさに慣れてしまうと、従来のディスプレイを見比べたとき、やけにぼやけたものに思えてしまいます。

また、超高精細な映像は平面であっても立体的に見えたり、奥行きを感じることさえあります。必要十分と思いつつも、無限の解像度を持つ自然の風景と見比べて、どこまでも高画質を追っていくのが人の特性なのかもしれません。

自動運転、エアバッグ、衝突検知、自動車庫入れ……車の世界は、私たちが子供の頃の図鑑やSF小説に描かれていた未来の自動車像をあらかた

SONYの活動量計
ウェアラブルデバイス「core」

モバイル専用QWERTY
キーボード「TREWGrip」
http://www.trewgrip.com/



Microsoftが提唱する
ブラジャー型ウェアラブルデバイス
（特許情報より）

実現してしまっています（まだ空を飛ぶ車は見かけませんが、すでに自家用の空陸両用車は販売されているそうです）。

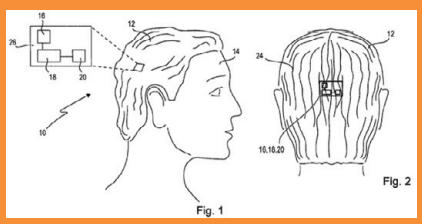
ウェアラブルの世界も、高精細ディスプレイの世界も、未来を描きつつ現実世界での革新を一歩一歩進めていくことが重要です。

デジタルカメラやバッテリー技術の進化により、身につけて撮影し続けるカメラが安価に作れるようになり、ライフログとしての映像記録が平易になりました。技術の進歩で今まで難しかったことができるようになるとともに、だれでも使え、安価になることで、もう一歩、違った使い方やさらなる活用方法が生まれてくるのです。

そして生活にとけ込むためには、常に持ち運べるか、毎日充電できるか、必要なときに素早く使えるか、使わないとき邪魔にならないか、ファッション性に優れており、装着して恥ずかしくないかなど、心配する事柄は山積みです。

あらゆる機能が載っており、スペック表にはすべて丸が付くスイスアーミーナイフのようなデジタルデバイスが優れているわけではありません。ある特定の機能が素晴らしく良かったり、よく考えられた使い込めるデザインが求められています。時代を担ってきた数々の新しい家電製品のデビューの場であるCES。来年もまた大きく流行が変化していることでしょう。来年のCESでも、未来的すぎない、すぐにでも使える興味深い新製品群を期待しています。

SD



SONYが提唱する
カツラ型ウェアラブルデバイス
（特許情報より）

gadget 1

Fin

http://www.wearfin.com/

指輪型ジェスチャーコントロールデバイス

Finは、RHL Vision Technologies社製の指輪型デバイスです。Bluetooth接続で、スマートフォンやスマートテレビ、車載機などと連携する指輪型のジェスチャーコントロールデバイスです。本稿執筆時点ではプロトタイプの試作／テスト中です。少し太め指輪を親指に装着して使います。防水で、micro USBで充電します。上下左右のスワイプ、タップ、そのほかユーザ固有の動きを感知して操作します。予価99ドルで2014年9月発売予定です。

gadget 2

Lumo Lift

http://www.lumobodytech.com/

猫背防止デバイス

Lumo Liftは小さな四角いセンサーを衣服につけておくことで、姿勢が悪くなったときにスマートフォンへ知らせてくれる小型専用デバイスです。歩数計測やカロリー計算もしてくれますが、主軸は猫背防止です。バッテリーは一度の充電で約5日間もつそうです。2014年夏前に発売予定で、予価69ドルとのこと。追加料金でカラフルな色合いのものが用意されています。

gadget 3

PrioVR

http://www.priovr.com/

全身センサー。ゲーム向けモーションキャプチャデバイス

PrioVRは身体各所に装着することで体の動きを認識するゲーム用のデバイスです。ヘッドマウントディスプレイと組み合わせて使えば完全にバーチャルな世界に没入できるかもしれません。腕と頭の動きだけのLZ版から、加えて足の動きも取得するLite版、さらに腰と肩、足首の動きまで取得できるPro版に分かれます。Kinectとは異なり、位置や環境にかかわらず、遅延が少なく正確にデータを取得できるのが特徴です。

gadget 4

iRing

http://www.ikmultimedia.com/products/iring/

楽器アプリ操作専用指輪

iRingはスマートフォンを触らずに操作するための、おもに楽器アプリ操作用の指輪です。横一列に3個並んだドット（円形）、あるいは三角形の形に並んだドットが印字された指輪です。この指輪を手にはめて、iPadやiPhoneの内側のカメラにかざして楽器アプリを操作します。SDKも公開される予定で、iRing対応のほかのアプリが登場するかもしれません。2014年春に24ドルで販売予定。

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

結城浩の再発見の発想法 10

Synchronous

Synchronous ——シンクロナス

シンクロナスとは

シンクロナス(synchronous)とは、処理がどのように行われるかを表す形容詞の1つです。日本語では「**同期的**」と言います。反対語は**エイシンクロナス**(asynchronous)で、こちらは「**非同期的**」と言います。asynchronousのはじめの“a-”は否定を意味する接頭辞です。アシンクロナスと呼ぶ場合もありますが、違いを強調するときには「エイ」と発音したほうが安全です。

さて、プログラムPが処理Sを開始したとします。制御がプログラムPに戻ってきたときに処理Sが**すべて完了している**なら、処理Sは**同期的処理**と言えます。このイメージ図を**図1**に示します。

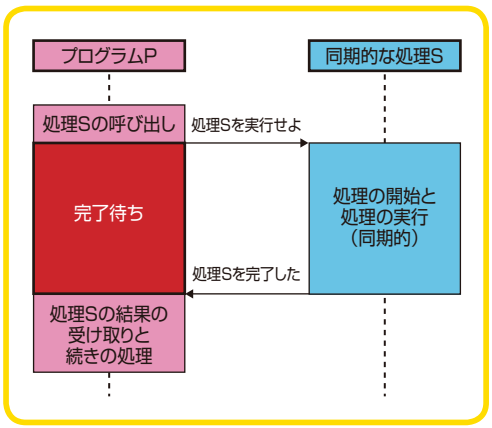
制御が戻ってきたときに処理がすべて完了しているのはあたりまえのように感じますが、**非同期的処理**と対比して考えると理解できます。プログラムPが非同期的処理Aを開始したとします。このとき、制御がプログラムPに戻ってきても処理Aは**すべて完了しているとは限りません**。処理Aは、プログラムPとは独立に処理が継続しており、処理Aが完了してからプログラムPに「はい、完了しました」とあらためて通知が行くのです。非同期的処理のイメージ図を**図2**に示します。

同期的／非同期的というのは複数のものが動作しているときによく登場する考え方です。

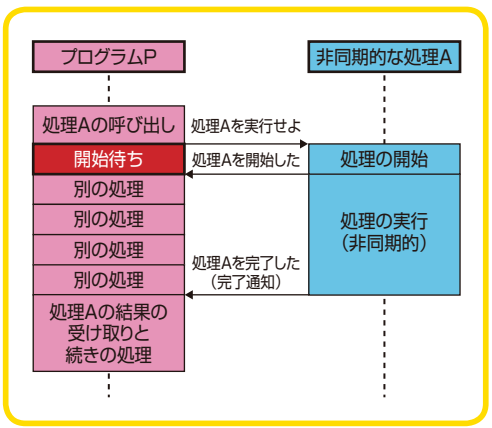
典型的例

同期的／非同期的処理の典型的な例は**入出力**です。たとえば「データをファイルに書き込む」

▼図1　同期的処理



▼図2　非同期的処理



という処理について考えましょう。

同期的処理Sの場合、データをファイルに書き込む処理がすべて終わるまでは制御が戻ってきません。言い換えると、制御が戻ってきたならデータはファイルにちゃんと書き込まれている保証があることになります。プログラムの構造は単純になりますので、これはメリットです。反面、制御が戻ってくるまでプログラムは待たされ、次の処理に進むことはできませんので、これはデメリットです。プログラムの応答性が悪くなりますので、ユーザは「重いな」と感じるかもしれません。

非同期的処理ではメリットとデメリットが逆になります。非同期的処理Aの場合、処理を開始した時点で制御が戻りますが、内部では書き込み作業を継続します。すぐに制御が戻ってくるので、プログラムは書き込み完了を待たずに次に進めます。しかし、制御が戻ってきたからといって書き込みが完了している保証がないので、プログラムの構造は複雑になりますし、書き込まれた情報を使う必要があるなら結局完了通知を待つ必要があります。非同期処理Aは多重に実行される可能性があり、エラー処理は複雑になることが多いでしょう。動作テストも難しくなります。

日常生活での同期的作業

実は日常生活でも同期的／非同期的の区別が重要になることがあります。

たとえば、あなたが誰かに作業を依頼したとしましょう。成果物の出来によってあなたの次の作業が変化するなら、あなたは同期的に作業を進めるしかありません。つまり、成果物の完成を待たなければならないということです。それに対して、成果物の出来にかかわらずあなたが自分の作業を進められるなら、非同期的に作業ができるでしょう。

複数人が共同作業を行う場合、互いに非同期的に作業できるようにしないと効率が悪くなります。そのためには、各人が他人の成果物に依存せず作業できるように分担を考える必要があります。一方、すべてが非同期的に動いていると全体のとりまとめが難しく、進捗状況がわかりにくくなる危険性もあります。

優先順位と同期的作業

複数人が作業を行うとき、各人は自分の作業の優先順位に注意しなければなりません。自分の作業の結果を誰かが待つことになるなら、その作業を優先して行う必要があるからです。つまりそれは**「誰かを待たせない」**工夫です。

逆に、自分が誰かの作業の結果を待つ状況になるのも好ましくありません。そのためには〆切を確認しておく必要があるでしょう。それは**「誰かから待たされない」**ための工夫です。

私たちはつい、自分が進めやすい作業から着手しがちです。しかし、集団での生産性を上げるには、「誰かを待たせない」また「誰かに待たされない」ように注意する必要があるのです。

完了通知の重要性

集団の生産性を上げるには「私の作業は完了しました」という完了通知が重要です。作業が完了しても、成果物を待っている人に伝わらなければ待ち時間が発生してしまうからです。

プログラムでは完了通知の方法が決まっています。それと同じように、日常生活でも完了通知の方法を合意しておくことは重要です。会って伝えるのか、メールで伝えるのか、電話で伝えるのか……前もって「プロトコル（約束事）」を決めておいて、無駄な待ち時間が発生しないようにしたいものです。

◆ ◆ ◆

あなたの周りを見回して、同期的作業や非同期的作業を探してみてください。作業をうまく分担することで、同期的な作業を非同期的な作業に変換することはできるでしょうか。また、「私の作業は完了しました」という完了通知を明確化し、無駄な待ち時間が発生しないようにできるでしょうか。ぜひ考えてみてください。SD

enchant ～創造力を刺激する魔法～

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

第11回 enchantMOONの誕生［後編］

取締役会の理解を得られないまま、なんとか独自開発のコンピュータを作ろうというプロジェクトがスタートしました。製品の名前は「enchant MOON」に決まりました。この名前は、企画者の辻と僕で決めた名前でした。使う人の思考を反射して輝く月のような端末。それが我々の作る端末が目指すべき姿だと。

ところがそうした崇高な志は、現場のプログラマとの衝突で脆くも崩れ去りそうな勢いでした。僕は企画者の辻とプロジェクトマネージャの増田から会議室で最終仕様書のプレゼンを受けていました。

仕様策定に潜む病

「ハイパーテキストとプログラミング、それがこの企画の最大のウリだろう。なぜそこを削るんだ？」

増田が答えます。

「それは……プログラマと相談した結果……」

僕はああ、なるほどな、と思いました。

「開発の現場に丸め込まれたか」

これはモノ作りの現場でしばしば出くわす組織的な病です。企画がどれだけ尖っていようと、否、尖っているからこそ、プログラマは自分の仕事を想像することが難しくなるのです。

モノ作り、特に尖ったもの、この世にないものを作ろうとすれば、次に待っているのはプログラマとの闘いです。ゲーム畑出身の僕は、この闘いを幾度もくぐり抜けてきました。プログラマをなだめ、脅しすかしして、なんとか理想とする仕様を組み込んでもらうのです。彼らがどうしても言うことを聞かないときの僕の必殺技は「そこをどけ、俺がコードを書く」でしたが、増田も辻もプログラミングができません。プログラマが無理だ不可能だやりたくないと言ったとき、言い返すことができなかったのでしょう。

時には僕自身がプログラマとしてそうした「尖ったものを拒絶する」側に回ったこともあります。プログラマは、見込みのない仕事はやりたくないものです。工数を見積もれと言われても、作ったことのないものや、やったことのない仕事に関しては工数を見積もることすらできません。あらかじめ見積もった工数に差異が出れば責められるのは自分ですから、自然と保守的になるしかないのです。

だがこれではダメだ。

僕はこう言うしかありませんでした。

「ハッキリ言って、いま君たちが作り、売ろうとしているものは、商品とは呼べない。完成する頃には新規性も薄れ、1台だって売れないだろう。これはGalaxy Noteから速度と利便性と通信機能を奪っただけの、劣化品だからだ。ハイパーテキストとプログラミング機能、手書きとプログラミングの融合、これこそが、この企画の最大の魅力であり、ウリだったはずだ。お絵描きができるだけの端末なら、万に一つも我々

に勝ち目はない」

増田は黙りました。かわりに辻が答えました。

「プログラミング機能は難しいので次世代機に載せようということになったんです」

「次世代機だって⁉」

僕は呆れました。ついさっきまで、取締役の連中を説得したと思ったらこの能天気さです。今だって奇跡的な綱渡りの上に予算が組まれているのです。次世代機なんて、この時点では夢のまた夢でした。

「そんなもの、一号機が成功しなけりゃ、あり得ないだろ？　投資だって何億だぞ。これがコケたら、会社は破産だ」

僕は絶望的な気分になりました。なにしろ企画を推進する当事者たちが、自分たちの考えた企画の真の価値を理解していないうえに、それを実現する方法が目の前にあるにもかかわらず、使おうともしないのです。彼らはただ、エンジニア出身の管理職としての僕に、どうすればエンジニアを説得できるか、仕様を詰める前にただ意見を聞けば良かっただけなのです。

「なぜ僕は君たちにこれを見せられてるんだ？」

僕は問いました。

「なぜ、僕と君たちなんだ？　なぜ、僕たちじゃないんだ？　どうして仕様を決めるプロセスに僕を参加させないんだ？　僕がその場に居たら、プログラマが無理と言っても、なんとかする方法を考え出すことができたはずだ。それが“僕のスキル”だからだ。僕は布留川くんほど堅実なコードは書けないし、辻さんほど斬新な企画を立てることはできない。だが“斬新な企画を実現する方法”を考え出すことについては、少なくとも君たちよりはずっとマシなアイデアを出せるはずだ。そうまでして僕を無視した結果が、これか」

怒りと絶望で、気が変になりそうでした。

僕は会議室を出ました。

増田が追いかけて来て、僕の手をつかみ、何か言いたそうに口を開きました。

僕はそれを遮り、言いました。

「いいか、君たちはとんでもない間抜けだ。ダイヤの原石を焚き火に放り込むようなことをやろうとしてる。一度でも炭になったダイヤは元には戻らない」

増田は顔を歪ませ、目に涙を浮かべました。

「馬鹿野郎、泣くな」

「すみません。開発者の気持ちや立場を考えると言い返せず、最後は清水さんがなんとかしてくれると思っていました」

「もし、まだ製品を少しでもマシなものにしたいと思うなら、明日、朝イチで会社に来い。とにかくこれじゃダメだ。徹底的にコンセプトを磨き上げる。開発者と話すのはそのあとだ」

翌朝、集まった3人でブレインストーミングが始まりました。この製品が目指す姿、人生のどの時点でどのように活用されるべきか、その将来像をまず徹底的に捉え直したのです。

ふつう、この手のブレインストーミングを開発が始まってからやることはあり得ません。しかし、そのとき僕たちに必要だったのは、自分たちが本当は何を作ろうとしているのか徹底的に考え抜き、知ることでした。それがわからなければ、その後のプロモーション計画も、マーケティングも、何も立てることができなかったからです。我々はなぜこの製品を作り、世の中に何を問いかけたいのか。

このときすでにハードウェアが非力であることはわかりきっていました。だからこそ、我々はハードウェアの能力の向こう側にあるべき、本当の理想像をできるだけ詳細かつリアルに描き出す必要があったのです。そのうえで、機能のロードマップを作り、何を優先して何を後回しにすべきか、それを決めるためのプロセスがこのブレインストーミングでした。

最終的に、子供が生まれてから老人になって死ぬまでの間にこの製品がどのようにかかわっていくか、議論を重ねた結果、人生のどの時点でも人々のクリエイティビティに寄り添う“オールライフコンピュータ”というコンセプトが固まりました。

コンセプトを最初から固め直し

カスタムOSに潜む罠

このコンセプトを持って、僕たちは現場の開発者たちと会合を持つことになりました。ところが現場はそれどころではありませんでした。CPUメーカーから渡されたオープンソースのAndroid OSの性能が想像以上に低すぎたのです。

「清水さん、ダメです。実機で前田ブロックを動かすと、1fpsも出ません」

「なんだって？」

僕は驚いてエンジニアリングサンプルをひったくるようにして見ました。たしかに、fps(フレーム毎秒)が0.3から0.8をうろうろしていました。

この製品にプログラミング機能を持たせるためには、ビジュアルプログラミング言語を動作させることは必須条件でした。ペン中心のタブレットで、プログラミングするのにキーボードが必須なんてことは不自然だからです。

このビジュアル言語は前田ブロックNEOと呼ばれていて、すべてHTML5上で動くenchant.jsの上で実装されていました。ところがターゲットとなる基盤でいざ前田ブロックを動かそうとすると、あまりにも遅すぎてとても実用的な速度で動かないのです。

現場はお葬式ムードです。増田も弱り切った顔をしています。もう投げ出したい、そんな雰囲気すら感じました。会議室を見回すと、誰もが諦めた顔をしていました。この企画はものにならない。もうどうにもならないんだと。

しかし僕は首を振りました。

「いや、なにか方法があるはずだ」

ハードウェアは非力とはいえ、少なくとも32ビットマシンで、GPUまで搭載されているはずです。思いどおりのパフォーマンスが出ないのは、前田ブロックかWebブラウザ、どちらかの実装がおかしいということになります。Androidの解析を担当したエンジニアが言います。

「たぶんWebブラウザがGPUを有効に使えていないのだろうと思います。ソースコードレベルでいろいろ試したけどダメでした。この人員でいまからWebブラウザのソースコードを改造してGPUに対応させるのは現実的に無理です。何ヵ月かかるか……ひょっとすると年単位かも……」

それを聞きながら、僕の頭の中に1つのアイデアが閃きました。間に合うかどうかわかりませんが、一か八か、試してみる価値はあると思いました。

「ケビン、Google V8でenchant.jsだけをGPUを使って動作させるVMを作ることはできないか？」

ケビンはドイツの名門、ミュンヘン工科大学からやってきた留学生です。enchant.jsのとくに難しい機能実装を担当しており、ARCきっての天才と呼ばれていました。まだ日本語に慣れていないため、僕の日本語がわからなかったらしく、首をかしげています。僕は同じことを英語で言い直しました。

「作ることは……たぶん、できます」

ケビンは片言の日本語でそう答えました。

「V8はプリコンパイル方式だ。うまくいけば、JIT方式のDalvikより速いかもしれない。ケビン、1週間やる。できるかどうか、答えを出してくれ」

僕は皆へ向き直り、言いました。

「よし、前田ブロックはケビンの“Eagle”に賭ける。ほかの者は基本機能の実装に全力を注いでくれ」

「イーグルって何ですか？」

「月着陸船だよ。この製品は、enchantMOONと呼ぶことにしたんだ。カスタムAndroidがSaturn-V(サターンファイブ)、その上で動くシェルがColumbia(コロンビア)、そしてJavaScriptによる

アプリ実行環境が、Eagleだ」

アポロ計画をモチーフにしようと考えたのはこの頃でした。我々のような小さな組織が人類を前進させるような新しい製品を目指して世界の一流企業に戦いを挑むという無謀なこの挑戦は、アメリカ合衆国という若い国家が、その総力を結集して月を目指したアポロ計画とイメージが重なったのです。

1週間後、ケビンが声を弾ませて報告に来ました。

「清水さん、15〜20fpsです。10倍……もしかすると50倍以上……高速化されてます」

ケビンは指示どおりわずか一週間でGoogle V8エンジンとGPUを組み合わせたEagleのプロトタイプを完成させたのです。ベンチマークの結果は、これなら勝負できる、と自信を深めるに充分なものでした。こうしてenchantMOONのプロジェクトは前へ動き始めました。

だめかもわからん

目先のトラブルを解決したとほっと胸をなで下ろして会議室を出た僕を、財務担当取締役の武市が待ち構えていました。僕は嫌な予感がしました。

「清水さん、もうだめかもわからん。予定よりキャッシュロスが激しいんよ」

会議室に入るなり、武市は僕に財務諸表を見せました。

「試作に関する金が予想外にかかりすぎとる。開発の進捗も人を投入しすぎとるし、このままいくと、身売りも選択肢の1つとして考えなあかんかもしれんよ」

そのとおりでした。気がつけば一番利益率の高かった部門であるARCが、これまでにないほど新製品にリソースを集中した結果、会社のキャッシュフローは急激に圧迫されていきました。しかしそれは必要な支出でした。MicrosoftやAppleが何千人という開発者をOSに投じているなか、OSで対抗しようとするのにたった5人でやろうとしているのです。これが1人でも欠けたら決して完成することはないでしょう。

やるならば、覚悟を決めなければなりません。僕は腹をくくりました。

「銀行から借りましょう。私が個人保証します」

「個人保証たって、何億やで。失敗したら……大丈夫か？」

個人保証といっても、何億なんていう借金を即金で返せるあてがあるわけがありません。この個人保証とは、失敗したら会社が潰れようが僕個人が一生かかっても支払うという、いわば奴隷契約のようなものです。

「ここが僕の正念場です。借りましょう」

それからしばらくして、メインバンクの重役が僕に面会したいという申し出がありました。おそらく借り入れの件だということで、武市も僕も張りつめたような緊張感のなか出迎えました。

「社長、お忙しいなか、お時間をいただき、ありがとうございます」

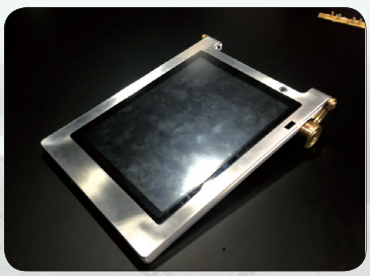
重役は、穏やかな表情で僕を見ました。

「ニュースで拝見しましたよ。国産のタブレット端末を作っていらっしゃるとか」

「恐れ入ります」

「大変興味深く拝見いたしました。期待しております」

じりじりとした時が過ぎる中、貸付けが実行された、という連絡が来ました。まさに薄氷を踏む思いで、経営者としてプロジェクトの峠を超えました。しかし先に待ち受けていたのが、さらなる困難だとはこのときの僕には思いもよらなかったのです。SD



ハードウェアの試作機(塗装前)

コレクターが独断で選ぶ!

偏愛キーボード図鑑

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

第11回

自分好みの手幅でタイピングできる

Kinesis Freestyle2

写真1　Kinesis Freestyle2

はじめに

今回はキーボードの真ん中あたりを境界に左右に分割されたキーボードを紹介します。キーボードを2分割することで、タイピングするときの左手／右手の置き場所がより自由になります。前回紹介したComfort Keyboard Originalは3分割されたキーボードでしたが、鉄板の上に固定する必要があり、自由度はその範囲しかありませんでした。今回紹介するKinesis Freestyle2(**写真1**)は、左右のパーツがケーブルでつながっているだけですので自由な位置取りが可能です。

Kinesis Freestyle2

Kinesis社が販売するキーボードです[注1]。Kinesisと言えば、コンタードキーボードが有名ですが[注2]、ほかにもいくつか種類があります。Freestyle2はそのうちの1つで、キー配列は普通の英字配列です。

特徴

次の特徴があります。

- 左右に分割された形状
- 軽めのキー荷重
- 充実したアクセサリ
- キーレイアウトは英字配列

キーボードはテンキーレスキーボードをちょうど真ん中から割ったような形となっています。分割された左右のパーツはケーブルでつながっており、その長さが9インチ(約20cm)のバージョンと20インチ(約50cm)のバージョンがあります。ケーブルは取り外しや交換ができないので、長い20インチのほうがお勧めです。もちろんキーボードを分割しないで使うこともでき、専用のフックで左右のパーツをつなげられます。このフックはキーボードの上部だけを止めるので、キーボードを扇状に配置できます(**写真2**)。

タイピングの際には、手幅や腕が窮屈にならないように左右のパーツを離せば、ゆったりしたフォームを取れます。また、左右のパーツの真ん中にスペースが作れるので、そこにトラックボールやマウスのような

写真2　左右のパーツをフックでつなげる

注1）http://www.kinesis-ergo.com/freestyle2.htm
注2）本連載第3回（本誌2013年7月号）で紹介しました。

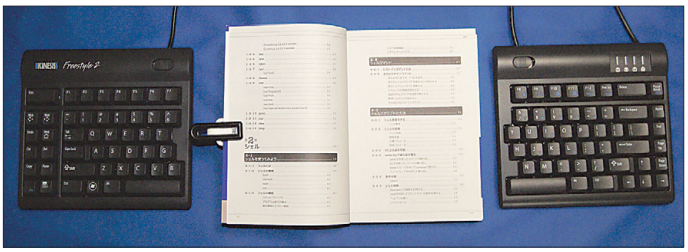


写真３　中央に書籍を置く

写真４　VIP3アクセサリを装着

デバイスを置いたり、書籍を置いたりできます（**写真3**）。

Freestyle2はメンブレンキーボードですが、タッチは軽めで長時間タイプしても疲れにくいです。よくあるメンブレンキーボードの場合、タッチは重めです。たとえばHappy Hacking Keyboard Liteだと約55gです[注3]。Freestyle2は軽く、約45gとなっています[注4]。これは、Happy Hacking KeyboardのProfessionalシリーズやメカニカルキーボードでよく採用されているCherry軸の中でも最も軽い茶軸や赤軸とほぼ同じ数値です。

別売ですが、アクセサリも充実しています。アクセサリは複数種類存在します[注5]。**写真4**はVIP3アクセサリを付けたものです。パームレストがつき、キーボードに傾斜がついています。傾斜は、5度、10度、15度と変えられます。

その他のアクセサリとして、パームレストのみのPalm Supportsや、パームレストがなく傾斜をつける機能だけに絞った省スペースタイプのV3などがあります。

キーレイアウトは、普通の英字配列ですが、Esc や Delete が大きいことと「Copy」や「Cut」などのホットキーがあることが特徴です（**写真5**）。左右のパーツを離さないでも使えるので、普通のキーボードと比較しても違和感なく使えます。

なお、Freestyleシリーズには古いバージョンの「Freestyle」と最新バージョンの「Freestyle2」の2種類が存在します。Freestyle2が発売されたのは比較的最近で、古いバージョンのFreestyleもまだ販売されています。新旧の一番の違いはキーボードの厚さです。Freestyle2のほうが薄くなっています（**写真6**）。

写真５　Esc とホットキー

入手方法

Freestyle2はKinesis社の日本代理店である㈱エジクン技研[注6]、もしくは日本のAmazon.co.jpから購入が可能です。値段は1万5,000円ほどです。旧バージョンのFreestyleも取り扱っていますが、在庫限りのようです。

◆ ◆ ◆

今回はKinesis Freestyle2を紹介しました。分割キーボードは使用する際のハードルはかなり低く、用途もいろいろあるので、「一定数以上の需要がある」と筆者は思っています。しかし、実際は製品の種類があまりないのが現状です。とくに日本語配列のキーボードの種類が少ない[注7]ので、手頃な価格で販売されることを願っています。SD

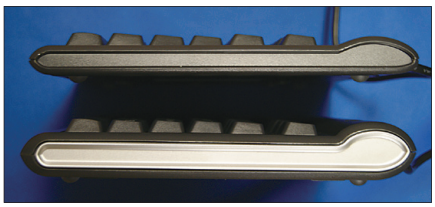


写真６　Freestyle2（上）とFreestyle（下）

注3）http://www.pfu.fujitsu.com/hhkeyboard/leaflet/keyspec.html
注4）https://www.kinesis-ergo.com/wp-content/uploads/2013/06/freestyle_vs_ms4000-800x3381.jpg
注5）http://www.kinesis-ergo.com/shop/freestyle2-for-pc-us/
注6）http://edikun.ocnk.net/
注7）パーソナルメディア㈱が販売しているμTRONキーボードがありますが、5万円以上と非常に高価です。

はんだづけカフェなう

ステッパーをはじめよう（前編）

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

ステッパーとは

ステッパーとは、ステッピングモーターあるいはパルスモーターと呼ばれるモーターの一種です。海外ではよくStepper（ステッパー）と表記されますが、日本ではたいていステッピングモーターと呼ばれます。ステッピングモーターは、モーターの軸を電気信号に合わせて一定角度ずつ回すことができるモーターです。

たとえばありふれた200ステップのステッピングモーターですと、1周が200ステップになっているので、1ステップ動かすと1.8°ずつモーターが回ります。簡単な制御回路で位置を割に正確に出すことができるので、正確さを要求されるメカには多用されます。

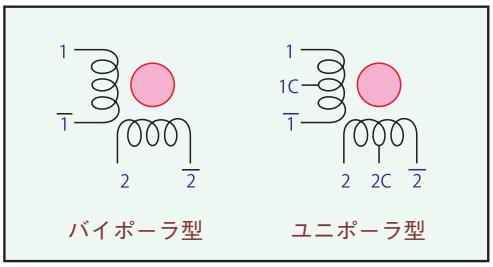
本誌2013年8月号の第1特集「3Dプリンタが未来を拓く理由」でも紹介しましたが、ステッピングモーターはインクジェットプリンタやスキャナ、あるいはクオーツ時計といった、手軽に正確な位置決めをしたい装置で多用されています。

ユニポーラとバイポーラ

ステッピングモーターには「ユニポーラ型」と「バイポーラ型」があります。回路記号を見るとわかりやすいので図1に記しますが、バイポーラは電磁石の両端から線が延びていて、ユニポーラでは加えて電磁石の真ん中からも線が出ています。つまり、バイポーラ型の電磁石で磁石の極性を切り替えるには、電流を流す方向を変える必要があります。一方、ユニポーラ型では、1Cから1または$\overline{1}$のどちらかに電流を流すかで磁石の極性を変えることができます。半導体は決まった方向の電流をON/OFFするのは得意ですが、反対方向に流すのは得意ではありません。

よって、ユニポーラ型のほうが使用する周辺回路を減らすことができるので、今回はユニポーラ型を例に話を進めます。

▼図1　バイポーラ型とユニポーラ型



ステッパーのしくみ

ステッピングモーターの内部には電磁石が入っていて、軸には永久磁石が取り付けられています。図2はステッピングモーターの内部を簡略化して記したものです。

電磁石に電流を流していない状態では、図2❶のように永久磁石は引っ張られていない自由な状態です。1の電磁石に電流を流すと、図2❷のように永久磁石は1の電磁石に引っ張られます。電流を流す電磁石を1→2→3→4→1と切り替えていくと、図2❷→❸→❹→❺→❻のように永久磁石が回転し、軸が回転することになります。

逆回転をするときには、1→4→3→2→1の順に電流を流していくことで、図2❻→❺→❹→❸→❷のように回転することとなります。

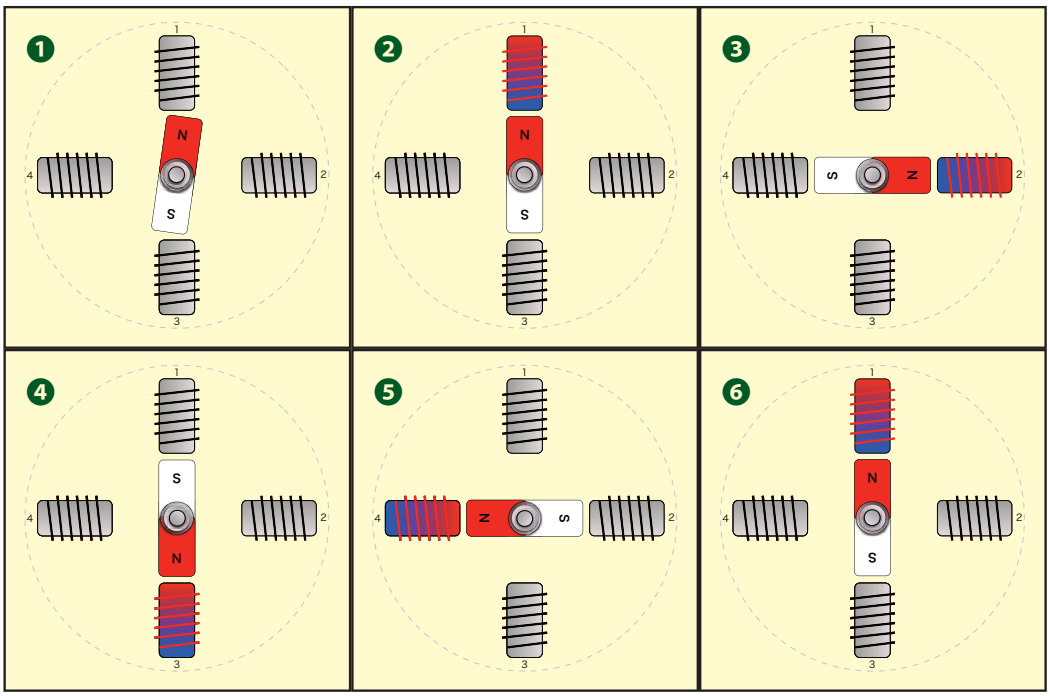
この電流を流すということを図にすると図3のようになります。図3❶のように接続して、スイッチを入れて電流を流すと図3❷のように電磁石が永久磁石を引っ張ります。電磁石に電流を流して磁石にすることを「励磁（れいじ）」と言います。このスイッチ操作をマイコンにさせれば、ステッピングモーターを意図したとおり回転させることができます。

ステッパーを動かす回路

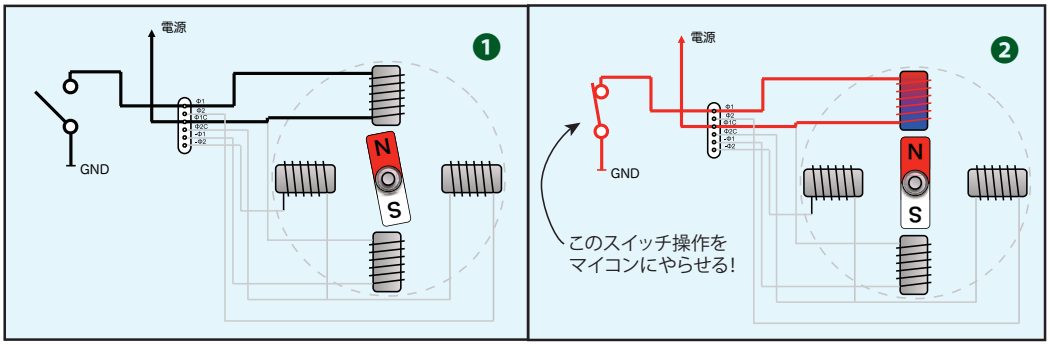
モーターは、マイコンに直接接続してはいけません。

この記事で紹介する、SPG20-1362という小型のステッピングモーターの場合、コイル抵抗は68Ωとあります。このモーターの電磁石（コイル）に5Vを加えると、100mAほど流れることが予想できます。マイコンのI/O（入出力）ピンに流しても良い電流はArduinoのATmega328Pで20mA、mbedなどに使われているLPCマイコンでたいてい4mAが最大となっています。つまり数十mAもの電流をマイコンのI/Oピンに流すとマイコンは壊れてしまいます。

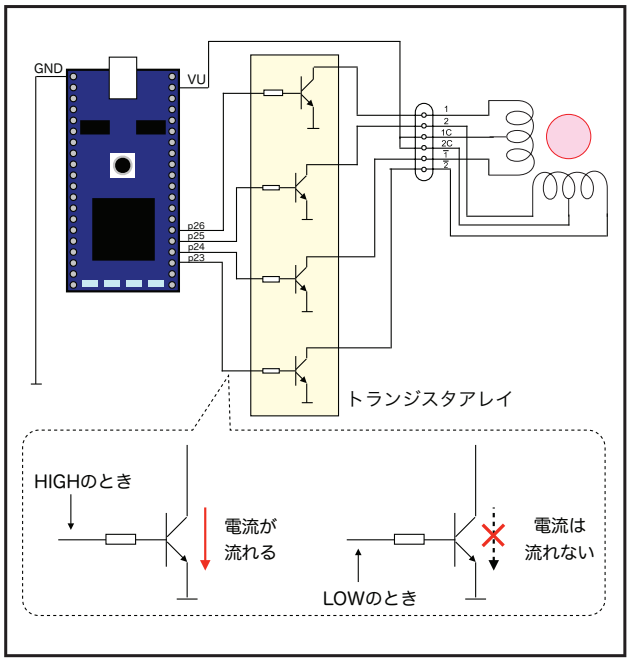
▼図2　ステッパーのしくみ



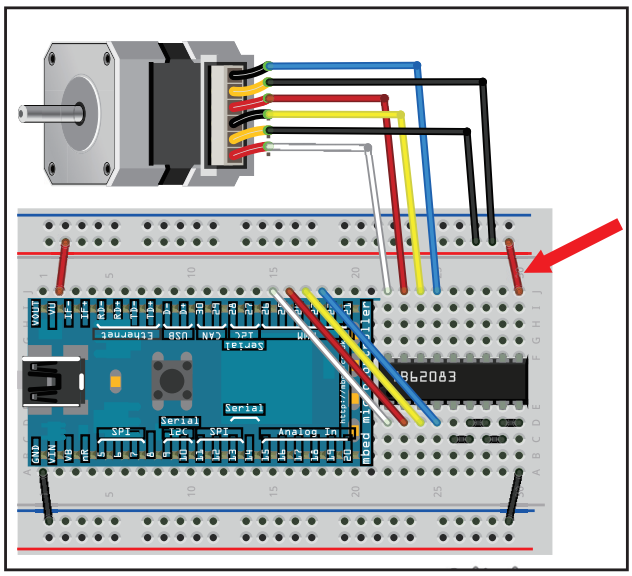
▼図3　ステッパーの励磁



▼図4　今回の配線図



▼図5　ブレッドボード上での配線



こういった場合、FET[注1]やトランジスタにスイッチの代わりをさせたりします。今回は、ブレッドボードでお手軽に試すことができるDIP[注2]で、価格も安いので、東芝のTD62083APGという8chトランジスタアレイを使用してみることにします(図4)。トランジスタアレイというのは1つのパッケージに複数のトランジスタが入っている素子のことで、この場合8chですので8個のトランジスタが入っています。このトランジスタは入力抵抗が2.7kΩということで、マイコンのI/Oピンが流さなければならない電流は1mA程度で済みます。一方で、出力電流は最大500mAと今回のモーターを回すのには十分な値になっています。もっと大型のステッピングモーターをコントロールするには、もっと大型のトランジスタやFETを用意する必要がありますので注意してください。

組み立て

では、実際に配線してみましょう。図5のように配線を行います。今回必要な材料は表1のとおりです。

筆書と同じように、ジャンパワイヤの色が長さで決まるのが好きでない方や、硬いジャンパワイヤが好きでない

注1)　Field effect transistor：電界効果トランジスタ

注2)　Dual Inline Package：2列に並んだリード線を備えたICの形状

▼表1　材料表(秋月電子通商での型番と執筆時の価格)

品名	型番	参考価格	URL
mbed	LPC1768	5,200円	http://akizukidenshi.com/catalog/g/gM-03596/
ステッピングモータ	SPG20-1362	250円	http://akizukidenshi.com/catalog/g/gP-04241/
ピンヘッダ	1x40	40円	http://akizukidenshi.com/catalog/g/gC-00167/
トランジスタアレイ	TD62083APG	100円	http://akizukidenshi.com/catalog/g/gI-01516/
ブレッドボード	EIC-801	250円	http://akizukidenshi.com/catalog/g/gP-00315/
ジャンパーワイヤ	EIC-J-L	300円	http://akizukidenshi.com/catalog/g/gP-00288/

方は、さまざまなタイプのジャンパワイヤが売られていますので試してみてください。

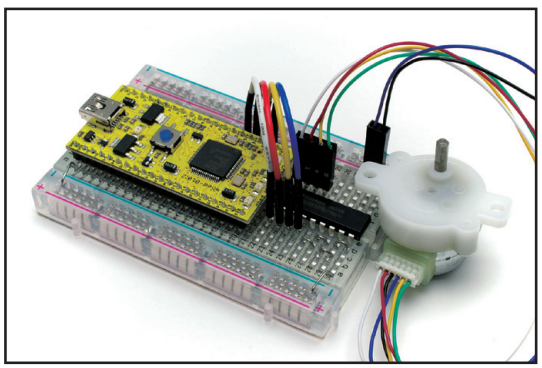
ステッピングモーターをブレッドボードと接続するには、ピンヘッダをステッピングモーターの線の先にはんだづけしなければなりません。

図4には記されていませんが、TD62083APGには、還流ダイオードというものが入っています。電磁石に流していた電流を切ったとき、逆起電力と呼ばれる力が働き、それまでに流れていたのと反対方向に電流が流れます。この電流はトランジスタを破壊してしまうことがあるため、還流ダイオードで逆向きの電流を逃がしてやることでトランジスタを保護します。**図5**中に赤い矢印で記した配線が、還流ダイオードの配線です。

今回使用しているステッピングモーターは12V用のものです。しかし、12Vの電源を用意するのも面倒ですし、5Vで動かすことでモーターの加熱などを和らげることができますので、今回は5Vで動かしたいと思います。また、100mA程度であればマイコンボードの電源出力端子から取っても問題ないため、今回はマイコンボードの電源出力端子から電源を取ってみたいと思います。トランジスタと同様に、今回は100mA程度をモーターを動かすために使うだけですので、マイコンボードの電源を使っています。もっと大きなステッピングモーターを使う場合は、トランジスタと同時に電源も検討する必要があることに注意してください。

筆者はmbedを使って組み立てましたが（**写真1**）、もちろんArduinoでも同様の回路でステッピングモーターを動かすことができます。

▼写真1　今回組み立てた例



ソフトウェア

次はソフトウェアです。@tedd_okano氏が書いたサンプルコードがありますので[注3]、使ってみたい方はダウンロードしてください。

ソフトウェアのやっていることはものすごく単純です。スイッチを順番に入れてモーターを回しています。0.01秒のウェイトをなぜ入れているかというと、あまり短時間電流を流してもステッピングモーターが追いつかないためです。こういう状態を脱調（だっちょう）と言います。同様に、あまり速く回転させるとトルク（回転する力）が下がります。

ソフトウェアを走らせると、モーターがぐるぐると回りだすはずです。waitの値を変えるとモーターの回る速さが変わるのが確認できるでしょう。

もっと手軽に

ここまではとりあえずステッピングモーターを回してみましたが、実際にステッピングモーターで機器の制御をするにはさまざまな要因を考える必要があります。たとえば、台形速度制御です。実際にステッピングモーターでモノを制御するときには、加減速を考えてやらなければなりません。滑らかに加減速をさせるためには台形制御という技術が一般的に用いられます。こういった制御をCPUで行うこともちろんできますが、ステッパーモーターコントローラなどと呼ばれる制御を委ねられるチップもあります。次回は、このステッパーモーターコントローラを使ってステッピングモーターをコントロールしてみたいと思います。SD

注3）http://mbed.org/users/okano/code/unipolar_stepper_motor_operation_sample/

PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは 2014 年 3 月 17 日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

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

01 1名

REALFORCE108UG-HiPro

プロフェッショナル仕様のキートップを採用したキーボード。打鍵の際に指がキートップ天面にフィットするよう深めに湾曲部をつけ、指がキーに馴染む形状になっています。タイピングでよく使用する手前から 2 列目～5 列目のキートップを高くすることで最前のキー列との段差を作り、誤って触れないよう考慮されています。インターフェースは USB、配列は日本語 108 配列（カナ刻印なし）、対応 OS は Windows 7 です。

提供元 東プレ **URL** http://www.topre.co.jp/

02 1名

JetFlash 380

標準 USB インターフェースとマイクロ USB インターフェースの 2 つの端子を搭載したデュアル USB メモリです。デスクトップ／ノート PC はもちろんのこと、USB OTG 対応のスマートフォンやタブレット端末でも利用可能です。色はゴールドで、容量は 16GB です。

提供元 トランセンドジャパン **URL** http://jp.transcend-info.com/

03 2名

超ブルーライト削減

簡単操作で PC やスマートフォンの画面から出るブルーライトを削減できるソフトウェア。ディスプレイの色合いを調整することで、PC では約 69％のブルーライト削減効果が実証されています。対応 OS は Windows 8.1/8/7/Vista/XP、Android 2.2 ～ 4.4 です。

提供元 ソースネクスト **URL** http://www.sourcenext.com/

04 4名

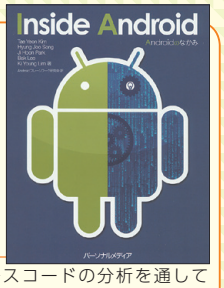
Treasure Data T シャツ

米 Treasure Data 社は 2013 年 11 月に、同社のロゴおよび Web サイトを刷新しました。それを記念して作られた同社の新しいロゴ入り T シャツ。サイズは M になります。

提供元 Treasure Data,Inc. **URL** www.treasuredata.com

05 2名

Android のなかみ



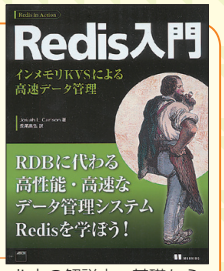
Tae Yeon Kim、Hyung Joo Song、Ji Hoon Park、Bak Lee、Ki Young Lim 著、Android フレームワーク研究会 訳／
B5 変形判、506 ページ／
ISBN = 978-4-89362-288-4

Android の内部構造を徹底解剖。ソースコードの分析を通して Android フレームワークの全貌に迫ります。構造と動作原理を理解すれば、Android に最適なソフトウェアの設計に役立ちます。

提供元 パーソナルメディア **URL** http://www.personal-media.co.jp/

06 2名

Redis 入門



Josiah L.Carlson 著、長尾 高弘 訳／
B5 変形判、360 ページ／
ISBN = 978-4-04-891735-3

OSS のキーバリューストア（KVS）の「Redis」の解説本。基礎から、リアルタイムデータの前処理、インメモリデータセットの管理、pub/sub（パブリッシュ／サブスクライブ）と設定まで解説します。

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

07 2名

詳説 C ポインタ

Richard Reese 著、菊池 彰 訳／
B5 変形判、248 ページ／
ISBN = 978-4-87311-656-3

C 言語のポインタについて、図とコードを多用して視覚的かつ直観的な理解を促します。メモリ構造と管理方法についても理解できるので、Java、C++、C# などのプログラマにも役立つ内容です。

提供元 オライリー・ジャパン **URL** http://www.oreilly.co.jp/

第1特集

データベースの諸問題

RDBとNoSQL どちらを選びますか?

真っ当に考えるDBの鉄則

「NoSQLが流行っているから、これでいこう」そんなに軽くていいのですか？　データベースは企業の実績を支える重要な資産です。リレーショナルデータベースを本当に活用していますか？　あなたに本当に必要なデータベースは何ですか？　本特集では、データベースについてRDBかNoSQLか根底から問いかけつつ、エンジニアの軸となる知識と技術を整理し紹介します。

CONTENTS

第1章 データベースの根源的誤解

Writer 奥野 幹也（おくの みきや）日本オラクル（株）
http://nippondanji.blogspot.jp/ ／ Twiter@nippondanji

NoSQLが普及し、その可能性が評価される昨今、RDB（リレーショナルデータベース）についていろいろな誤解をしていませんか。本章では、RDBにまつわるさまざま誤解を取り上げ、それらについて検証します。皆さんのシステムで、NoSQLを使うべきか、それともRDBを使うべきか、一度じっくり考えてみませんか。

NoSQLはRDBの置き換えにはならない

リレーショナルデータベース（以下RDB）でないデータベースを総称してNoSQLと呼ばれていますが、それらは非常に高い人気を博しています。NoSQLというカテゴリのデータベース製品が台頭して久しく、本稿を読んでいただいている皆さんの中にもすでに使ったことがある、あるいは利用を検討しているという方も多いのではないでしょうか。とくに新しいものが好きな人、RDBを使った経験上で困難にぶち当たったことがある人は、ユートピアを求めてNoSQLに手を出そうとしてらっしゃるかもしれません。

しかしながら、世の中にはそのような都合の良い世界というものは存在せず、NoSQLにはNoSQLなりの問題や課題があるものです。そこで、今回はデータベースにまつわる、人が陥りやすい誤解について解説します。皆さんがデータベースを選定するうえでヒントになれば幸いです。

RDBは時代遅れという3つの誤解

「RDBはもう古い。今はもうNoSQLの時代だ」そんなセリフを聞いたことはないでしょうか。一般的に、新しいもののほうが正しい、あるいは将来に渡って長く使うことができるというようなコンセンサスが世の中にはありますが、RDBに関してはそれは当てはまらないと言わざるを得ません。RDBはかなり歴史のあるソフトウェアではありますが、現在進行中で最もよく使われているデータベースソフトウェアであり、なおかつ今後も廃れることはないと考えられるからです。

SQLより前の時代のデータベースとは何か

当然のことですが、SQLを扱えるデータベースソフトウェア、すなわちRDBがこの世に登場する前は、この世にあったデータベースソフトウェアはすべてNoSQLと呼ばれるカテゴリに含まれるものでした。なぜならば、まだSQLがこの世に登場していなかったからです。とくに階層型データベースやネットワーク型データベースがよく利用され、これらはトランザクションも実装しているものでした。しかしそれらのデータベースにはさまざまな課題や問題があり、それを克服するためにRDBが発明されたという経緯があります。したがって、SQLが時代遅れというよりは、NoSQLが先祖返りという見方もできなくはないでしょう。

RDBには確固たるデータモデルがある

リレーショナルモデル最大の利点は、その背後に集合論と論理学（述語論理）という確固たる数学的モデルが存在する点です。NoSQLと呼ばれるカテゴリのデータベースには、プログラ

ミングに役立つ便利なデータ構造を詰め合わせたようなものもあります。そういったデータベースソフトウェアは確かに便利ではあるのですが、そのバックグラウンドとなる数学的モデルがないため、応用が利きにくいですし、将来に渡って陳腐化させないのは難しいと言わざるを得ません。数学的モデルがないことの何が問題なんだ？——と思われるかもしれませんが、数学的モデルに基づいているというのは、データベースソフトウェアにとって重要です。

・さまざまな数学の公式や定理を応用できる
・複雑なクエリを表現できる
・データモデルの正しさがこの先も否定されることがない

しっかりとしたデータモデルが存在することの意義はとても大きいです。リレーショナルモデルの背後にある数学的なモデルは、現時点ではもちろん正しいものですから、時代遅れではありません。また、集合論のようにベーシックな概念は、将来的にも正当性が否定される可能性はまずありませんので、末永く付き合っていくことができます。

日々進歩するRDB

リレーショナルモデルの歴史は古く、そのアイデアに目新しさはもはや皆無であると言っても過言ではありません。しかしながら、RDBはデータベースアプリケーションの最前線にいまだ居座り続けています。というのも、RDB自身が常に進化しているからです。とくに実装面では常に洗練され、性能の向上あるいは新しい機能の追加の勢いはとどまることを知りません。

RDB製品の進化はなぜ起きるのでしょうか？——単なる既存の製品に対する改良というようなものでは説明のつかない部分があります。その背景にあるのが次の2点です。

ハードウェアの進化

RDBが進化する原因の1つは、やはりコンピュータハードウェアの進化があると考えられます。コンピュータハードウェアは常に進化し続けていますから、それを最大限活用するにはソフトウェアも追随しなければなりません。CPUが複数のコアを搭載するのが当たり前になって久しいですが、ソフトウェアの性能を最大まで引き出すには、効率よくCPUコアを利用できるようなアーキテクチャになっていなければなりません。各種RDB製品では、並列処理を効率的に行うために、ボトルネックの解消が日々精力的に行われています。

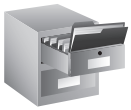
また、最近はSSD(Solid State Drive)が普及してきましたが、それによりデータベース製品のトレンドも変化を受けています。ディスクI/Oがボトルネックというのは、従来からデータベースにとっては常識だったのですが、その傾向にも変化が見られるようになって来ました。とくにPCI Express接続のSSD製品を搭載していると、先にCPUのほうが限界を迎えることが多くなってしまいます。豊富なコアがあるにもかかわらずにです。次世代インターフェースであるNVMe(Non-Volatile Memory Express)が普及すれば、その傾向はより顕著になるかもしれません。

ニーズの変化

ハードウェアの進化という内向的な動機とは対極的に、ニーズの変化という外向的な要因もあります。ここ数年、ビッグデータというバズワードが世間を賑わしています。ビッグデータがバズワードであるのは間違いありませんが、データベースソフトウェアが管理しなければいけないデータが年々増えてきているのはまぎれもない事実です。パーティショニングやインデックス、あるいはストレージレイヤーの進化をはじめとして、分散型やカラム指向のRDBといったものも登場してきました。いかにして効率的に大きなデータ、あるいは大きなトラフィック

を捌くことができるかというのは、これからもデータベースにとって最優先の課題であり続けるでしょう。

今のところ、RDBはハードウェアの進化、ニーズの双方に対してうまく追随しているように見えます。その状態が続く限り、RDBが時代遅れになることはないでしょう。



RDBが遅いという3つの誤解

NoSQLデータベースの利用を検討している人には、漠然とRDBが遅いという考えを持っている人が多いように見受けられます。確かに、SQLの解析やアクセスコントロール、トランザクションなどがある分、単純なしくみでアクセスできるKVSなどに比べるとオーバーヘッドは大きくなる傾向があります。しかしその一方で、SQLは複雑なクエリを表現するのが比較的容易だという利点がありますので、一概にどちらが速いかというのは単純に比較することはできません。

JOINが遅いという誤解

RDBにおいて散々言われてきた誤解があります。それがJOINが遅いというものです。

まず最初に、JOINと一口で言ってもさまざまなものが存在するという点に注意してください。2つのテーブルからそれぞれ1行ずつフェッチするだけで済むようなJOINであれば、インデックスが利用できれば極めて高速に処理できるでしょう。反対に、大量の行をJOINするようなものなら遅くなって当然です。これは使い方の問題であり、JOINそのものが遅いのではありません。当然使い方を誤れば遅くなりますが、それはJOINだけでなくどのような演算でも言えることです。

また、もうひとつ覚えておいていただきたいのは、RDBほどJOINを高速に処理できるソフトウェアはないという点です。たとえばKVSに**図1**のような値が格納されている場合を考えてみましょう。

listで始まるキーには、userで始まるキーのリストが格納されています。たとえばlist:1をgetしてから、対応するuser:1,user:2,user:3をmgetすれば、KVSでもJOINっぽい処理ができるというふうに考えることができます。ところが、このような処理の仕方は効率がよくありません。

この例のように3件程度ならたいした負荷ではありませんが、1000件、100万件などを処理しようとするとキーのやりとりだけでネットワークの帯域をかなり消費してしまいます。また、KVSにはトランザクションがありませんので、list:1をgetしてからuser:Xが更新されてしまうかもしれません。KVSでJOINっぽい処理をしようとすると、性能だけでなく、データの整合性の面でも問題が生じてしまうことになります。

▼図1　KVSにおけるJOINの例

key	value
list:1	1,2,3
list:2	3,4,5
user:1	A
user:2	B
user:3	C
user:4	D
user:5	E

CAP定理という最大の誤解

RDBが遅いという根拠として、度々CAP（Consistency：一貫性, Availability：可用性, Partition-tolerance：分断耐性）定理が持ち出されることがあります。しかし、CAP定理を根拠にするのは危険です。なぜならばCAP定理には致命的な問題があるからです。

最大の問題は、CAP定理はそもそも定理と呼べるものではないという点です。定理と名乗るからには数学的な意味での定理になっていな

ければ誤解を生んでしまいますが、CAP定理は定理としての要件を満たしていません。定理というのは公理、あるいは別の定理から論理演算によって導かれるものでなければならず、その論理演算の過程を証明と言います。しかしながら、CAP定理にはそのような証明はありません[注1]ので、定理とは呼べないのです。経験則、よくても仮説止まりでしょう。「CAP仮説」という名称ならば異論はありませんが、定理でないものを定理と呼ぶのは不適切です。

誤った前提から出発すると、どのように議論あるいは考察しても正しい結論は導かれないでしょう。CAP定理はあくまでも参考程度の仮説として考えておいたほうが無難です。将来的にはCAP定理の殻を破ったデータベースソフトウェアが登場するかもしれません。

性能の見方についての誤解

性能について検討するうえで欠かせないのが、スループットとレスポンスについての理解です。よく「NoSQLは速い!!」みたいな解説記事を巷で見かけますが、根拠となっているベンチマークを見ると、レスポンスについてのものである場合が多いです。

レスポンスとは、ある処理の開始から完了するまでの時間です。もちろんレスポンスが良い(時間が短い)に越したことはありませんが、データベースサーバにより求められるのはスループットです。スループットとは単位時間あたりにどれだけの処理を実行できるかという指標で、同時にどれだけの処理を実行できるかという性能にも大きく関係してきます。最近はCPUのコア数も増えて来ましたから、それらをいかに効率的に使いこなせるかというのもデータベースソフトウェアにとって重要な課題です。

ベンチマークを見るときは、どのように計測したか、値にどんな意味があるのかということをふまえて吟味しましょう。

注1) 証明っぽいものならありますが。

リレーショナルモデルについての3つの誤解

RDB最大の特徴は、何と言ってもそのデータモデルです。リレーショナルモデルはとてもよくできたデータモデルであり、複雑なデータ構造を必要とするデータベースアプリケーションを構築するのに向いています。ところが、実際の現場ではリレーショナルモデルがおろそかにされがちで、その結果多くの人がRDBをうまく使いこなせないという状況に陥っているようです。

データベースは単なる入れ物ではない

開発者の中には、「データの加工やロジックはアプリケーション側で頑張るから、データベースは何でも良い」とか「インデックスを付けておけば何とかなる」と考えてしまう人がいるようです。確かに、突き詰めればアプリケーション側で実装すれば何でもできてしまうわけで、このような主張もある意味では正しい側面もあるのは否定できません。ただし、どのようにデータベースを使うかで開発のたいへんさがまったく違ってしまうという点が重要です。具体的には、DB設計のよし悪しによってクエリの書きやすさが変わってきます。

RDBに限った話ではありませんが、道具は正しく使ってこそ便利なものとなります。たとえば料理をするとき、スライサーは根菜などを薄くカットするのに向いていますが、刺身を作るのには向いていません。出刃包丁は魚を捌くのに便利ですが、根菜をスライスするのには向いていないでしょう。調理器具と同じように、データベースも正しく使ってこそ真価を発揮し、便利な道具となります。材料(テーブル)に応じて、適切な器具(クエリ)を選んで加工することで最適な性能が得られるのです。データベースを単なるデータの入れ物だととらえて使い方を間違えると、労力の割には報われないことになってしまいます。

どのようにデータベースを設計すべきか、またはクエリを書くべきかということを理解するには、データモデルについての理解が不可欠だということを覚えておいてください。RDBでは、リレーショナルモデルです。

そもそもリレーションとは何か？

リレーショナルモデルについての非常にポピュラーな誤解は、リレーショナルモデルがテーブル同士の関係を表現するデータモデルであるというものです。大事ですのでもう一度言いますが、リレーショナルモデルはテーブル同士の関係を表すデータモデルではありません。

ではリレーションとはいったい何でしょうか？大学などで習ったという人にとっては常識とも言えるのですが、現場ではリレーショナルモデルはおろそかにされがちです。この問いに答えられるようになってはじめて、リレーショナルモデルを理解する第一歩を踏み出すことになります。

実は、リレーションとは、リレーショナルモデルにおける演算の単位、つまりデータの集合なのです。SQLではテーブルがリレーションに相当します。リレーションに対して演算を行うデータモデルだから、リレーショナルモデルという名前なのです。

テーブルとリレーションは対応する概念ですが、SQLとリレーショナルモデルでは用語が少しずつ違っています。SQLとリレーショナルモデルの用語の対応を**表1**に記しておきます。

リレーショナルモデルとSQLで名称が異なるのは紛らわしいのですが、これには明確な理由があります。テーブルとリレーションは対応する概念ではありますが、性質に違いがあるのです。性質が異なるから名称も異なるというわけです。本稿では詳しくは述べませんが、リレーションはテーブルと違って次のような性質があります。

- NULLがない
- 重複する要素を含まない
- タプル同士の順序がない
- 更新できない

本稿では深くは触れませんが、これからデータベースを学ぼうという人は、ぜひリレーショナルモデルについてもっと詳しく勉強してみることをお勧めします。

▼表1　SQLとリレーショナルモデルの対応

SQL	リレーショナルモデル
テーブル（表）	リレーション（関係）
カラム（列）	アトリビュート（属性）
ロー（行）	タプル（組）

リレーショナルモデルの限界

リレーショナルモデルについて詳しくなると、その得手不得手もわかるようになります。これはどのようなデータモデルにも言えることですが、万能なものはこの世には存在しません。リレーショナルモデルは優秀なデータモデルであることは間違いありませんが、当然の如く限界、つまりデータモデルを適用できないケースがあります。ズバリ、リレーショナルモデルで表現できるのは「集合」として表現できるデータ構造に限られるのです。たとえば多重集合やグラフ、ツリー、履歴といったデータ構造は、本来リレーショナルモデルでは扱うことができない分野です。

ところが、SQLはリレーショナルモデルの垣根を超えてそういったデータ構造も扱えるようになっています。その点を理解していないと、テーブルとしてデータを表現できたけどなぜかクエリをうまく書けないというような状況に陥ったとき、その理由がわからず苦労するでしょう。

リレーショナルモデルが扱えないデータ構造は、SQLなら表現可能とは言え、RDBが得意とする分野ではありません。そのようなデータ構造がメインであれば、ほかのデータモデルを持ったデータベースソフトウェアの利用を検討する価値はあります。

ストアドプロシージャについての誤解

とくに昔からRDBを使っている人に多いのですが、ストアドプロシージャこそがRDBの真骨頂だと考えている人がいるようです。残念ながら、それはとんでもない誤解だと言わざるを得ません。

リレーショナルモデルは集合論をベースとしたデータモデルです。1つ、あるいは複数の集合(＝リレーション)を用いて演算した結果、新たな集合を得るというのが根本的な考え方です。このようなモデルの最大の利点は、ループや分岐といった手続き型の処理を排除できる点にあります。ループの代わりに集合をベースとした演算をすることで、シンプルにクエリを表現することが可能になるのです。

ところが、ストアドプロシージャはループや分岐を使って処理を表現する方法ですから、ストアドプロシージャに頼っていてはリレーショナルモデルの利点をすべてぶち壊してしまうのです。ただし、リレーショナルモデルにうまく適合できないデータをSQLで扱うような場合には、ストアドプロシージャはたいへん強力なツールとなります。この点を理解していれば、いつストアドプロシージャを使うべきかを判断できるでしょう。

DB設計についての4つの誤解

RDBに限った話ではありませんが、DB設計は極めて重要かつ難しいテーマです。にもかかわらず、リレーショナルモデルへの理解不足から、RDBを使っている現場ではDB設計についての誤解が横行しているようです。その結果、状況をより悪くしてしまっているケースが多々あります。

正規化は重要でない?

一番よくあるのが、正規化は必要ない、あるいは重要ではないという誤解です。RDBを使うならば、正規化はしっかりと行うべきです。正規化はリレーショナルモデルにおける極めて重要なDB設計理論です。正規化をすることで、データの重複を排除し、その結果データの不整合を防ぐことができます。重複がないということは、あるデータを更新しようと思ったら、一ヵ所だけを更新すれば良いということです。

反対に、重複がある場合には同じデータが出現する複数ヵ所すべてを更新しなければなりません。このとき、一ヵ所でも更新漏れがあると、本来同じ値であるべきデータに食い違いが生じてしまうことになります。正規化されていないテーブルを更新し、その結果不整合が生じた状態を**図2**に示します。

図2はとある会社の従業員の名簿をテーブルとして表したものです。従業員名、所属する部署、部署の代表電話(内線番号)が含まれています。あるとき、鈴木さんが営業部へ配属変更になったので名簿を更新しました。ところが、そのとき部署名は変更したのですが、電話番号を更新するのを忘れてしまいました。長年この会社に勤続している人なら、どの部署がどの内線

▼図2　更新による不整合の例

EMP_NAME	DEPT_NAME	DEPT_PHONE
山田太郎	営業	1100
鈴木一郎	マーケティング	1200
山田花子	マーケティング	1200
佐藤二郎	営業	1100
山本三四郎	経理	1300

⇒

EMP_NAME	DEPT_NAME	DEPT_PHONE
山田太郎	営業	1100
鈴木一郎	営業	1200
山田花子	マーケティング	1200
佐藤二郎	営業	1100
山本三四郎	経理	1300

番号なのかを記憶しているかもしれません。しかし、まったく知らない人が更新後のテーブルを見て、営業部の内線番号が何かということを判断できるでしょうか？「1200よりも1100のほうが出現回数が多いから1100かもしれない」と推理することはできますが、それはあくまでも推理であって、正しいという保証はないのです。

このように、いったんデータに不整合が生じてしまうと、データそのものから「本当に正しい値は何か」ということを導き出すことはできなくなってしまいます。つまり、不整合が生じたデータベースでは、クエリが正しいデータを返すとは限らないのです。これはデータベースにとって致命的です。

DB設計で「重複がないこと」を保証できる正規化は、RDBの運用をぐっと楽にしてくれる強い味方なのです。

いったん設計したら変更してはいけない？

ウォーターフォール型の開発で問題になりがちなのは、先にDB設計を済ませてしまい、その後そのスキーマを使って開発を行うというものです。プログラムがバグから逃れられないのと同様、DB設計も最初から完璧なものを作ることはできません。とくにテーブル数が多ければ多いほど、検討する必要がある組み合わせが増えてしまうため、DB設計に欠陥が生じやすくなってしまいます。コードを書いてみたらあらためてDB設計の欠陥に気がついたということもあるでしょう。

DB設計の変更は手間がかかりますが、欠陥が発覚したら修正をすべきです。時間はかかってしまいますが、アプリケーションとデータベースをセットでリファクタリングする必要があります。問題を放置してしまうとバグの温床になりますし、その欠陥をカバーするためのコードは後から技術的負債となって重くのしかかって来ることになります。

スキーマレスならDB設計がいらないのか？

「スキーマレスなNoSQLを使えば、DB設計に頭を悩ませなくても良い」というような発言を耳にしたことはないでしょうか。残念ながら、それも誤解と言わざるを得ません。スキーマを柔軟に変更できることと、DB設計をしなくても良いことはイコールではないからです。

ドキュメント型データベースを使っていても、クエリを記述する際にはどこにどのようなデータが格納されているかという情報が必要です。ドキュメント型データベースに問い合わせを行う際には、ユーザが定義したデータ構造に従ってクエリを記述することになります。つまりユーザが事前に「ドキュメント内のどこにどんなデータを格納しておくか」ということを決めておかなければならないのです。スキーマにその情報が表れない分、スキーマレスのほうが返ってクエリを書くのが難しいかもしれません。

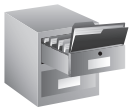
スキーマレスはDB設計を簡単にする万能薬ではありません。確かにいつでも柔軟に好きなようにデータの構造を変更できますが、ただそれだけなのです。ドキュメント型データベースでも、RDBと同じように、データの重複という問題がついてまわります。RDBとは違い正規化によって解決できない分、ドキュメント型データベースでは解決が難しいでしょう。また、たとえ正規化を行ったとしても、ドキュメント型データベースではJOINできませんのでクエリの記述は難解になってしまうでしょう。さらに、データ構造の変更を防ぐ手立てがないという点も問題です。ドキュメントのデータ構造の変化に対して制約を付けることができないため、何かの手違いでデータ構造が変わってしまい、データの更新漏れが起きる可能性があります。

ドキュメント型データベースは、スキーマレスであるため扱いやすい点もありますが、反対にスキーマレスであることが欠点にもなるのです。

O/Rマッパーを使えばDB設計は適当でよいのか?

最近はO/Rマッパーを備えたフレームワークを使った開発が活発ですが、「オブジェクトにマッピングしてしまえばDB設計はいろいろとボロがあってもプログラムで何とかできる」というふうに考えてしまってはいませんでしょうか?……残念ながら、これも大きな誤りです。

その理由は、たとえO/Rマッパーを使った場合でも、その背後ではSQLが実行されているからです。もしDB設計をおざなりにしてしまって、たとえば正規化を行わなかったとすると、重複による不整合が生じてしまうリスクが残ってしまうのは、O/Rマッパーを使っていようがいまいが同じなのです。



まとめ「データと向き合い、使い分ける」

データベースにまつわるさまざまな誤解について解説してきましたが、いかがでしたでしょうか。心当たりのあるものはあったでしょうか?

NoSQLデータベースは、使いどころを間違えなければ強力なデータベースソフトウェアであることは否定のしようはありません。NoSQLの人気は増す一方のように見えますし、新しいNoSQLソフトウェアも次々に登場しています。一見すると、NoSQLはRDBに取って代わる新しい世代のソフトウェアだと思ってしまいがちです。しかし、そのような考えが誤解であることは、本稿で説明したとおりです。

とはいえ、RDBも万能ではありませんので、いかにうまくNoSQLと使い分けるか、あるいは併用するかがポイントとなります。使い分けるべきポイントを見極めるには、やはりRDBや各種NoSQLについて正しく理解することが重要です。リレーショナルモデルの範疇はどこか、リレーショナルモデルに収まらないデータはどこまでならSQLで対応可能か、対象のデータに対応したデータベースソフトウェア(NoSQL)は存在するか、といったことを理解して初めて、正しい選択を行うことができるでしょう。SD

Column

DB設計に潜む罠

筆者は、仕事がらさまざまなテーブルやクエリを目にします。クエリのパフォーマンスチューニングで調査をしていると、多くの場合はそのクエリがアクセスするテーブルの設計に問題を抱えていることに気づきます。そして、その兆候はすべてクエリに表われています。その結果パフォーマンスに問題を抱えることになってしまうのです。

たとえば、カラムがNULLを許容するためにCOALESCEやIFNULLが多数用いられていたり、テーブルにフラグを表すカラムが多数あるため検索条件が複雑怪奇になっていたり、OLTPなのにクエリの中で関数(MAXやMIN、COUNTなど)が用いられていたりといったものです。これらはすべてリレーショナルモデルのセオリーを無視したテーブル設計が招いた結果であり、こういった兆候を見かけたら、クエリをどうにかしようと考える前にまずはDB設計を見直すべきでしょう。

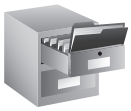
とはいえ、多くのユーザにとってDB設計を変更するという決断は勇気が要るものであり、変更するという判断はなかなか取られにくいものです。そんなとき筆者は、インデックスやクエリの書き換えでどうにか逃げるような対策をアドバイスしています。DB設計がリレーショナルモデルのセオリーから外れていても、インデックスを工夫したりストアドプロシージャを用いたりすることで、何とかなってしまうケースは多いです。

しかし、当然そういったアドバイスは次善の策であり、その場しのぎであるため、対策が負債となって残ってしまうでしょう。RDBが本来のパフォーマンスを出すには、根本的にリレーショナルモデルに沿ったDB設計が不可欠なのです。

第2章 データベース設計における地力をつけよう！
――「正規化」再考

Writer 小野 哲（おの さとし） フリーライター／街のRDBMS研究家

RDBは数学的な理論を元に実装されたものであり、正規化はその理論の中で定義されています。実用的なDBを作る際には、あえて正規形を崩す必要もありますが、理にかなった崩し方をするためにも、理論を正しく理解しておきたいものです。本章では、理論に沿った正規化から現実的なDB実装まで一気に解説します。



正規化を意識していますか？

正規化という言葉。情報処理試験以来耳にしていないのではないでしょうか。普段なにげなく行っているデータベース（以下、DB）設計ですが、案外正規化について知らずに設計していたりするものです。実際にDB設計をする際、いちいち「これは第三正規形まで到達している」とは考えずに一気に表構造に落とし込むものです。「正規化なんて全然意識していないけど、設計できているし……」という人がほとんどですよね。

なのに、なぜここで正規化の話をするのか？

ズバリ！正規化の意味を理解することでDB設計における地力をつけるためです。正規化はリレーショナルデータベース（以下、RDB）の発生と進化の歴史そのものであり、それを理解することはDBを理解することなのです。

加えて、近年ビッグデータの流れに伴い、NoSQLの話題が多く、一部ではRDBは過去の技術であるとまで言われています。しかしながら、正規化について深く考察すると意外にもNoSQLにおけるDBはどのように設計すべきなのかが見えてくるものです。

歴史からの考察

RDBの歴史が、1970年に発表されたコッド（F.Codd）の論文「A Relational Model of Data for Large Shared Data Banks（大規模共有データバンクのデータ関係モデル）」からスタートしたのは誰もが知っています。この論文は集合論を発展させたリレーショナルデータモデルを説明したものでしたが、そこには早くも正規化のことが触れられています（1.4章 正規化）。

コッドの理論はDBの正規化を段階的に進化させました。それがいわゆる第一正規形から第三正規形に至る基本的な正規形でした。コッドの理論を元にデータを表現／操作する最低の条件は、第一正規形です。その理由は、データの集合をスカラ値で表現することで集合演算が初めて可能になるからです。詳しくは後で述べます。

第二、第三正規形では集合演算の必須条件ではなく、データの追加／更新／削除時の異常を回避するための論理が重要視されます（1971年のコッドによる論文）。

その後、ボイスコッド正規形がボイス（Raymond F.Boyce）とコッドによって定義され、第四、第五正規形がロナルド・ファギン（Ronald Fagin）によって定義されます。最初の論文から約7年で第五正規形までを定義するに至り、ひとまずの完成を見ることになりました。

それと並行して、コッドのリレーショナルモデルの理論（**リレーショナル理論**）を使えば、数学的にデータの構造を定義／操作できることがわかると、世界の研究者によりこぞってリレーショナル理論が実装されていくわけです。その実装系の代表がSQLなんですね。ちなみに、コッド率いるIBMチームがSystem Rを世に出した

のを皮切りに、コッドの論文を読んで影響を受けたラリー・エリソン(Larry Ellison)がOracleを、マイケル・ストーンブレーカー(Michael Stonebraker)がIngresを創り出し、SQLを実装したのです。そうやってRDBMS(Relational Database Management System)が完成されていったのです。

数学的な美しさか、実用性か

リレーショナル理論から発展していったRDBMSは、実装が進化するにしたがってDBの世界を一新しました。SQLは簡単にデータモデルを記述/操作でき、従来の階層型DBやネットワーク型DBと違って、格段に使いやすくわかりやすいものだったからです。

ところが、RDBMSにおけるデータのモデリング(リレーショナルモデル)はデータの正規化を運命づけられているため、テーブルはどんどん細分化する傾向にありました。あまりに正規化し過ぎてシステムのパフォーマンスを落とすこともありました。そのため、現場ではわざと正規化を崩すことも行われますが、データ構造を改変するより別の方法を模索することもできます。

(1)CPUやメモリなど物理的な性能を上げる
(2)特殊なテーブルの管理(パーティショニング、マテリアライズドビューなど)をする

これらの方法を**スケールアップ**といいます。いずれの方法も根本的なデータの構造は変えずに、追加的にパフォーマンスを上げられます。ハードウェアの技術が発達にするにつれ、ハードウェアコストが劇的に低廉化し、スケールアップが容易になり、致命的な問題は目立たなくなりました。ところが、「それらの方法すら行き詰ったら?」という命題が依然として残ったままです。スケールアップの限界をどうするかということですね。

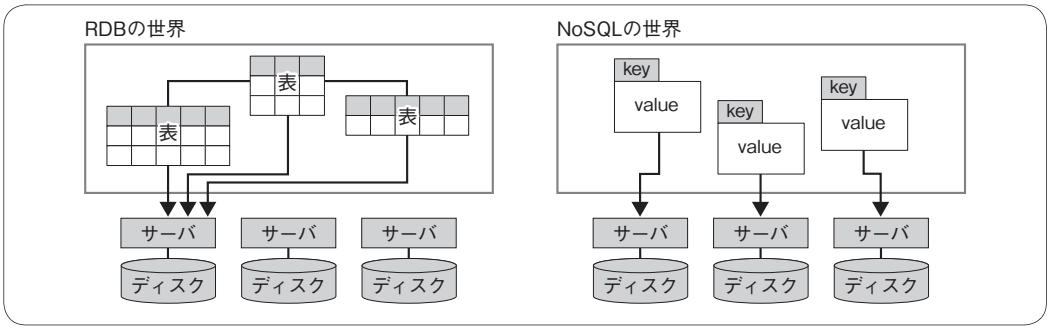
これはすなわち、近年流行のビッグデータに対する命題と同義でした。動画などのデータの巨大化、SNSなどの爆発的なデータの増加などが現実的になり、DBシステムがスケールアップで対処できないことが多くなったのです。ようやく問題点が目立ちはじめました。

この命題に対するベターアンサーは**分散**です。簡単にいうと、DBサーバを分散し並列度を上げることです。これを**スケールアウト**といいます。(2)のパーティショニングなどは分散化可能な技術ですが、RDBMSのように正規化されそれぞれの表が強い制約で結ばれている場合には、分散は限定的となり、大規模な分散は難しくなります。つまり、正規化すればするほど分散が困難になるというパラドックスが目立つことになりました。ここに至って初めて非正規化の有用性が出てきたのです。

スケールアウトを可能にしたNoSQL

分散の実際の方法は意外に簡単です。データ同士の依存性をなくし、データをまとまった単位で分散することです。それにより書き込みも読み込みも分散されるため、並列度が上がりパフォーマンスが向上します(**図1**)。実はこれが

▼図1　分散が難しいRDB、分散しやすいNoSQL



NoSQLに求められていることであり、NoSQLが得意とするところなのです。

NoSQLの基本構造は、KVS（Key-Value-Store）という、キーとデータで構成されるシンプルな方法でデータを保存することです。キーに対しデータの中身はアイデアしだいで何でもアリで、配列やリスト構造、大規模な動画データなどでもかまいません。それぞれのデータが論理的に独立していることで、データの分散が容易になるわけです。NoSQLが非正規化を好む理由はそこにあります。NoSQLの登場によりスケールアウトが現実的になったのです。

「あれかこれか？」ではなく「あれもこれも！」

ところが仮に、正規化されたデータをNoSQLで扱った場合、それはRDBMSで管理するのと同じことになり、たちまち正規化のパラドックスに陥ります。関係とその制約、一貫性維持などを必要条件にした場合は、NoSQLであってもRDBMSと同じくスケールアウトが難しくなります。正規化を崩すという半端なレベルではなく、データをあえて非正規化することにNoSQLの本領があるというのは、理屈がわかるとおもしろいですよね。

ですので、RDBMSかNoSQLかという二者選択の議論は実に不毛で、ユーザの要件によって選択は変化するということがわかりますし、RDBMSもNoSQLも一緒に使う「あれもこれも」というのが賢い選択だということです。

正規化のおさらい

では、ここから、おさらいのつもりで正規化についての基礎を学びましょう。読者の中には、実務でDB設計をしている人も多いと思います。正規化やRDBMSについての書物を読むと、なんとなく難しい数学的表現でいやになってしまう方もいるかもしれません。ドメインとかタプルとか関数従属とか候補キーとか、それらの言葉に翻弄されていた人も、この機会に苦手意識を克服しましょう。そのためには、まず基本的な用語を攻略します。これから一気に説明する用語さえ押さえれば、ありがたい書物の正規化についての記述もリレーショナル理論の記述もスラスラと理解できるようになるはずです。理論武装ができれば普段のDB設計にも磨きがかかるでしょう。

集合論、ドメイン、タプル

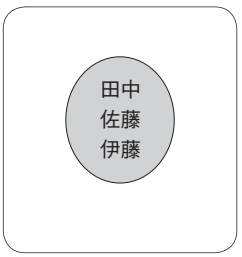
ある会社に**図2-1**のような社員の集合があるとします。一見して社員名の集合だとわかります。集合は一般的に{x|xは社員}のように表現します。この場合は、社員という集合を表してしています。この集合を**ドメイン**といい、**図2-2**のように表現します。

ここに部署というドメインが加わるとします（**図2-3**）。この2つのドメインの属性の値同士の組み合わせを**直積**といい、次のように表せます。

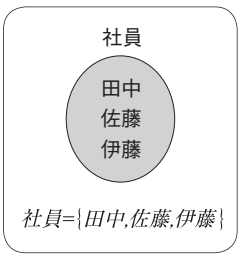
$$社員 \times 部署 = \{(田中,開発),(田中,営業),(佐藤,開発),(佐藤,営業),(伊藤,開発),(伊藤,営業)\}$$

この直積の集合の各要素を**タプル**（tuple＝組）といい、直積の集合の任意の部分集合を**リレーション**といいます。リレーションは次のように

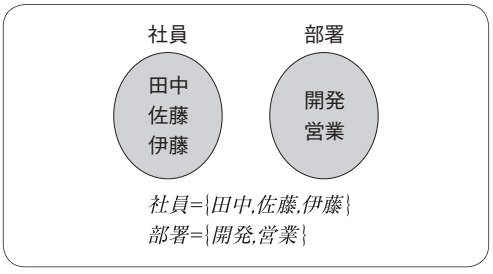
▼図2-1　社員の集合



▼図2-2　社員ドメイン



▼図2-3　社員ドメインと部署ドメイン



表現します。

$R=\{(田中,営業),(佐藤,営業),(伊藤,開発)\}$など

ドメインの個数を**次数**といい、次数nのリレーションをn項のリレーションと表現します。今回の例では2項のリレーションということになります。ところで、現実の社員-部署の関係は表で表せますね。たとえば**図3-1**のような表があったとします。

この表そのものがリレーションです。リレーショナル理論の実装系であるRDBMSの世界では表であるリレーションを**テーブル**といい、ドメインの値を**列**(column)といい、タプルを**行**(row)といいます。**表1**のように比較するとわかりやすいかもしれません。

さらに、RDBMSの世界では社員や部署というドメインを**属性名**といい、表の名前を**表名(リレーション名)**といいます。この例ではさしずめ社員表と名付けられますね。ちなみに表題に相当する部分を**リレーションスキーマ**といい、タプルの集まりであるデータの中身のことを**インスタンス**といいます。

主キー、外部キー、制約

さて、社員表のタプルは一意であるように見えます。しかし、同名の社員が存在すると仮定すると、そのタプルは一意にはなりません。そのため、一意にすべくなんらかのしかけが必要です。そこで識別番号のようなものを与えれば一意のタプルを作れます。ここでは社員番号という列名を付け足すことにします(**図3-2**)。

これでタプルを一意にすることができました。このようにタプルを一意に識別できる属性を**キー**(一般的に**候補キー**)といいます。候補キーは複数の組みであってもかまいません。候補キーの中で1つを選んで代表的なキーにする場合、それを**主キー**といいます。主キーは不定な値であるNull値を許さないという約束があります。それを**主キー制約**といいます。

部署表が別にあったとすると、社員表と部署表は社員番号でつなげられます(**図3-3**)。

部署表の部署番号は主キーです。社員表の部署は、部署表の部署番号を参照します。このようなキーの関係を**外部キー**といいます。このとき社員表の部署番号は、部署表の部署番号の外部キーとなります。

▼図3-1　社員表

社員	部署
田中	営業
佐藤	営業
伊藤	開発

$R=\{(田中,営業),(佐藤,営業),(伊藤,開発)\}$

▼図3-2　社員表(社員番号を追加)

主キー →

社員番号	社員名	部署
S1	田中	営業
S2	佐藤	営業
S3	伊藤	開発

▼表1　リレーショナル理論とRDBMSにおける用語の対比

ファイルの世界	リレーショナル理論の世界	RDBMS実装系の世界
ファイル	リレーション	テーブル
レコード	タプル	行
フィールド	ドメインの値、属性	列

▼図3-3　社員表と部署表をつなげる

社員

社員番号	社員名	部署番号(外部キー)
S1	田中	B1
S2	佐藤	B1
S3	伊藤	B2

部署

部署番号(主キー)	部署名
B1	営業
B2	開発
B3	伊藤

外部キーとして定義した場合、部署番号に登録されていない番号、たとえばB4を社員表に登録した際にはエラーとすることができます。これによって論理的に矛盾するデータを防御する構造ができあがります。このような制約を**外部キー制約**といいます。主キー制約や外部キー制約などデータの論理的な矛盾を起こさないように制約をかけることを**一貫性制約**、または**整合性制約**といいます。この関係の制約を定義できるのがRDBMSの大きな特徴です。

リレーショナル代数、リレーショナル論理

コッドによって提唱された重要な数学が、リレーショナル代数とリレーショナル論理です。これを説明するには誌面の問題がありますので、さらりと触れるだけにします。

一般的に私たちはテーブルの操作をするときに、SELECT文を使います。ありとあらゆる演算ができるので便利ですが、そもそもこのSELECT文はリレーショナル代数によって成り立っているものなのです。たとえば、「社員番号がS01の人の名前と部署を求める」といった問いをリレーショナル代数で表現すると次のようになります。

$$\pi_{社員名,部署}\left(\sigma_{社員番号='S01'}(社員)\right)$$

πを射影、σを選択といいます。次のSQL文と同じ意味です。

```
select 社員名,部署 from 社員表 where
社員番号='S01'
```

一方リレーショナル論理は通常の集合論的な記述のしかたです。たとえば集合の直積を表現するときは次のようになります。社員表と部署表の直積だとすると。

$$社員 \times 部署 = \{t * u \mid t \in 社員 \wedge u \in 部署\}$$

「tとuのタプルのANDをとった集合」という意味です。リレーション論理の中にもタプルリレーション論理式とドメインリレーション論理式があり、代数と同じくデータの操作系を表現できます。論理式の場合は代数より集合論的になります。参考までにタプルリレーショナル論理式で「社員番号がS01の人の社員番号を求める」を記述すると次のようになります。

$$\{t^{(1)} \mid (\exists u)(社員(u) \wedge u[社員番号] = \text{'S01'} \wedge t[社員番号] = u[社員番号])\}$$

これは次のSQL文と同じ意味です。

```
select 社員番号 from 社員 where 社員番号
='S01'
```

リレーショナル代数／論理で記述できるデータ操作は、実際のRDBMSでもできなくてはなりません。最低限これができるデータ操作系を持ったRDBMSを**リレーショナル完備**といいます。今、世の中に存在するSQL処理系の多くが、リレーショナル完備といえるでしょう。またSQLの言語仕様が手続き型ではない理由もそこにあるわけです。

このようにコッドが提唱したリレーショナル理論は、最初からデータの操作をすべて数学的に説明していました。そのあとづけでSQLが実装されていったという事実をリアルに感じて

▼図4　第一正規化

① 社員

社員番号	社員名	部署	好きなもの
S1	田中	営業	日本料理、車
S2	佐藤	営業	酒、たばこ、ギャンブル
S3	伊藤	開発	本、映画

繰り返しデータを排除する

② 社員

社員番号	社員名	部署	好きなもの
S1	田中	営業	日本料理
S1	田中	営業	車
S2	佐藤	営業	酒
S2	佐藤	営業	たばこ
S2	佐藤	営業	ギャンブル
S3	伊藤	開発	本
S3	伊藤	開発	映画

いただけたでしょうか。

第一正規形 ―― 繰り返しデータの排除

一気にリレーショナル理論のおさらいをしたところで、ここから正規形のお話に入ります。ここまでの話でデータの最小単位はドメインの値であるということがわかりました。その値は分割できない単位であり、数学的には**スカラ値**といいます。複数存在する値や配列などの値はスカラ値ではありません。もしデータの中にスカラ値ではないものが含まれていたら、それを排除しなくてはなりません。

図4-①のような表があったとします。「好きなもの」の値は複数存在しています。このようなデータはリレーショナル理論では扱えず、それによって成り立つRDBMSでも基本的に扱えないことになっています。このようなデータの形を**非正規形**といいます。

このデータを最低限RDBMSで扱える形にする場合は、スカラ値の集合にする必要があるので、**図4-**②のようにすればいいことがわかります。これを**第一正規形**といいます。正規化されたデータは複数の行になります。

関数従属性とは

次の説明の前に、従属性について説明しておく必要があります。あるリレーションの中で属性同士の関係が「Aが決まればBが一意に決まる」という関係が成り立つとき、BはAに従属しており、ちょうどy=f(x)といった関係になりますね。それを**関数従属性**といいます。A→Bと表現し、BはAに関数従属していると表現します。社員番号→社員名という関係です。

一般に関数従属性はキーとそれ以外のものを関連付ける場合に存在します。タプルを一意にできる属性を持ったキーを**スーパーキー**といいます。社員番号はタプルを一意に決定できますので、スーパーキーといえます。ちなみに、候補キー・主キーもスーパーキーです。

また、社員番号は社員名を一意に決めますが、1人の社員に複数の社員番号は存在しませんね。完全に1対1の場合、BはAに**完全従属**しているといいます。

一方、部署と社員の関係を見てみましょう。営業に対して{田中,佐藤}と複数対応しています。Aが決まるとBが複数決まりますね。これを**多値従属性**といい、A ↠ Bと表現します。以上のリレーショナル理論の用語を覚えておくと、正規化を理解するのに役立ちます。

第二正規形 ―― 部分関数従属性の排除

では、第二正規形の説明です。**図5-**①のような売上のデータがあったとします。繰り返しがないので第一正規形を満足しているといえますね。

しかし、顧客だけではタプル(あるいは行)を一意にすることはできません。「顧客」と「商品」がセットになって初めてタプルを一意に識別できます。このとき{顧客,商品}が主キーであるといえます。関数従属性からすると{顧客,商品}→数量、{顧客,商品}→単価が成り立っています。しかし、一方で「単価」は「商品」単独のキーによって関数従属していることもわかります。つまり、商品→単価という関数従属です(**図5-**②)。

このように候補キーの1つが別の関数従属性を持っている場合を**部分関数従属**といいます。このような場合、データを更新することで矛盾が発生することがあります。1つのタプルのデータの単価を10000円に更新したとすると、別のタプルの単価は依然20000円のままです。このデータの矛盾を**更新時異常**といいます。データの矛盾をなくすためには、テレビを含んだタプルすべてを更新しなくてはなりません。手間がかかります。

そこで、この部分関数従属を排除します。部分関数従属を別のテーブルに分離することにより**第二正規形**となります(**図5-**③)。これによってテレビの単価が変わったとしても1ヵ所を更新するだけで良いので、更新時異常を回避できます。このような状態を**一事実一箇所**(1 fact 1 place)といい、正規化での重要な原則の1つです。

第三正規形 —— 推移関数従属性の排除

図6-①のような表があったとします。部分従属性はなさそうなので、第二正規形の条件は満たしています。この表では、部署が決まると勤務地が決まるものとします。なんとなく気持ちが悪い感じがしませんか？　この違和感の正体は何でしょうか。

関数従属を見てみましょう。図6-②のようになります。第二正規形の条件は満たしながらも1つのタプルに複数の関数従属が含まれています。社員番号→部署、部署→勤務地という従属です。社員番号が決まれば部署が決まり、部署が決まれば勤務地が決まるといったX→Y、Y→Zという推移的な関数従属です。これを**推移関数従属性**といい、**第三正規形**はこの推移関数従属を排除したものをいいます。

Xを主キーとすれば、Y→Zの関数従属性を分割します。図6-③のようにすれば、第三正規形が得られます。これにより、部署の勤務地が変更になったら、社員表の該当データすべてを更新しなければならないという更新時異常を回避できます。

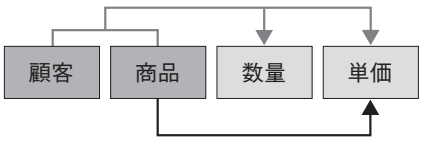
第三正規形まで実施すれば、現実的な正規化

▼図5　第二正規化

① 売上

顧客	商品	数量	単価
A事務所	テレビ	1	20000
A事務所	ラジオ	2	3000
B事務所	テレビ	1	20000
B事務所	USBメモリ	1	1500

②



部分関数従属性を排除する

③ 売上

顧客	商品	数量
A事務所	テレビ	1
A事務所	ラジオ	2
B事務所	テレビ	1
B事務所	USBメモリ	1

商品

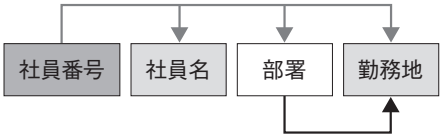
商品	単価
テレビ	20000
ラジオ	3000
USBメモリ	1500

▼図6　第三正規化

① 社員

社員番号	社員名	部署	勤務地
S1	田中	営業	東京
S2	佐藤	営業	東京
S3	伊藤	開発	神奈川

②



推移関数従属性を排除する

③ 社員

社員番号	社員名	部署
S1	田中	営業
S2	佐藤	営業
S3	伊藤	開発

勤務地

部署	勤務地
営業	東京
開発	神奈川

はほぼ終了とみなして良いでしょう。これ以上の正規化は業務のルールなどでやるかやらないかが決まりますし、場合によっては第三正規形をあえて崩す場合もあります。現場で活躍するSEはDB設計する段階で、無意識に正規化を行っているものです。ただ、正規化のルールや原則などを知っていると、設計時に理論的なチェックを行えるようになりますし、正規形を崩す場合にも意識的にできます。その意味では、その先の高次の正規化についても知識として知っておくと良いでしょう。

ボイスコッド正規形 ――非キー→候補キーの排除

さて、**ボイスコッド正規形**の説明です。Wikipediaによると「関係(リレーション)上に存在する自明でないすべての関数従属性の決定項が候補キーである」と定義されています。まったく何のことかさっぱりわかりませんね。でも大丈夫です。

「自明でない関数従属性」というのはめんどうな言い方ですが、なんてことはありません。{社員,部署}→社員や{社員,部署}→部署など、説明しなくてもわかっている関数従属性を**自明な関数従属性**といいます。ですので、通常の形にしておいて「自明でないすべての関数従属性の決定項が候補キーである」ということは、ありていにいえば、候補キー→非キーの関係です。すべての非キーの属性が候補キーに完全従属するということです。まあ、当然のようなことですが、この逆を考えれば良いのです。この条件を崩すものはなんでしょうか?

そうです。非キー属性が候補キーを決定づけるものです。すなわち候補キーが非キーに関数従属する場合です。こういう形はボイスコッド正規形ではないといえます。

これをうまく説明できる例は現実にはあまりないので、教科書などでよく見かける例で説明します。**図7-**①は社員が適正にしたがって受けるセミナーを決める表だとします。社員とセミナーで受ける講師が決まり、社員は複数のセミナーを受けられます。しかし、講師は1つのセミナーしか担当しないとします。

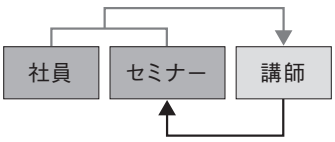
このとき**図7-**②のような関数従属性が成り立ちます。候補キーである{社員,セミナー}が講師との関係従属性が成りたち、部分関数従属性も推移従属性も存在しないことから、これは第三正規形であるといえます。しかし、非キーである「講師」と候補キーの一部である「セミナー」が関数従属性にあることから、この表はボイスコッド正規形ではないといえます。

▼図7　ボイスコッド正規化

①

社員	セミナー	講師
田中	論語	孔子
田中	政治学	孟子
佐藤	政治学	韓非子
伊藤	自然言語処理	老子

②



非キー → 候補キーの関係を排除する

③

社員	講師
田中	孔子
田中	孟子
佐藤	韓非子
伊藤	老子

講師	セミナー
孔子	論語
孟子	政治学
韓非子	政治学
老子	自然言語処理

この例では{社員,セミナー}が主キーであることから、「社員」と「セミナー」が決定しないと「講師」の登録ができないことになります。また社員の田中を削除すると、講師の孔子と孟子が存在しなくなってしまうという更新時異常が想定されます。

これらをボイスコッド正規形にするには、次のような分解を行います。社員→講師、セミナー→講師、講師→セミナー。ただ、セミナーと講師の関係は1つで良いので、社員→講師と講師→セミナーとの関係だけで良いことになります。したがって、**図7-**③のようになります。これによって先の更新時異常も解決します。

第四正規形 ——自明でない多値従属性の排除

第四正規形はボイスコッド正規形と同様に、現実的にはあまり遭遇しないものです。理論上あるという程度で覚えておけば良いでしょう。**第四正規形**の定義は「関係に存在する多値従属性は、自明であるか、決定項がスーパーキーである」ということです。またわけのわからない定義ですね。

関数従属性はA→Bのように「Aが決まれば一意にBが決まる」ということでした。これに対して「Aが決まれば複数のB(多値)」が決まることを多値従属性といいましたね。A↠Bと表します。

では、自明である多値従属性の反対である「自明でない多値従属性」とはどういうものなのでしょう。それはA↠Bという多値従属性のデータと別に、A↠Cという多値従属性を持ったデータが混在していることを言います。A↠B|Cと表します。自明でない多値従属性は対象性を持っているので、**対象性のある多値従属性**ともいいます。そういったものは第四正規形ではありません。

図8-①のような表があるとします。これらすべての属性をすべて組み合わせて主キーにしたとすれば、第三正規形もボイスコッド正規形もいずれの条件も満たしています。しかし、このデータには社員↠スキル、社員↠資格の多値従属性はあっても、スキルと資格の間には何ら従属性がなく独立したものです。つまりA↠B|Cの形となり、第四正規形ではありません。

そこで、これらをA↠B、A↠Cの形に分解します(**図8-**②)。こうすることにより、対象性のある多値従属性が排除され、自明な多値従属性を持った表に分割されました。これで第四正規形を満たしたものとなります。

さて、勘の良い人はすでに気がつきましたね。分割した2つの表を自然結合したらどうなるのか？　そうです、もとの表に戻りますね。難し

▼図8　第四正規化

①

社員	スキル	資格
田中	コミュニケーション	税理士
田中	顧客開拓	税理士
田中	コミュニケーション	珠算一級
田中	顧客開拓	珠算一級
佐藤	コンサルティング	漢字検定二級
佐藤	新規事業開拓	漢字検定二級
佐藤	コンサルティング	英語検定二級
佐藤	新規事業開拓	英語検定二級
伊藤	基本設計	情報処理技術者一種
伊藤	要件定義	情報処理技術者一種

自明でない多値従属性を排除する

②

社員	スキル
田中	コミュニケーション
田中	顧客開拓
佐藤	コンサルティング
佐藤	新規事業開拓
伊藤	基本設計
伊藤	要件定義

社員	資格
田中	珠算一級
田中	税理士
佐藤	英語検定二級
佐藤	漢字検定二級
伊藤	情報処理技術者一種

いことをいっていますが「自然結合された表は第四正規形にはならないよ」といっているのと同じなんですね。なあんだという感じですね。

第五正規形 ―― 結合従属性を保つ

では第五正規形について述べます。第四正規形までくると、これ以上分解できないという単位に近くなりました。これ以上、正規化できるのだろうかと思われます。

第四正規形の場合A↠B|Cの自明でない多値従属性をA↠B、A↠Cまで分解しました。それら2つを結合すると元に戻ります。それを**結合従属性**といいます。結合従属性が保たれている場合は**第五正規形**を満たしているといいます。しかし、2つの結合だけでは元に戻らない場合があります。

第四正規形の場合は、独立した多値従属性だったのですが、A↠B、A↠C、B↠Cというようにそれぞれが関連した多値従属性をもっているとき、A↠B、A↠Cの表結合だけでは

▼図9-1　結合従属性が保たれていない例

①

社員	スキル	担当業務
田中	コミュニケーション	営業
田中	顧客開拓	営業
佐藤	コンサルティング	営業支援
佐藤	新規事業開拓	企画立案
伊藤	基本設計	システム開発
伊藤	要件定義	営業支援

自明でない多値従属性を分解

② 社員_スキル

社員	スキル
田中	コミュニケーション
田中	顧客開拓
佐藤	コンサルティング
佐藤	新規事業開拓
伊藤	基本設計
伊藤	要件定義

社員_担当業務

社員	担当業務
田中	営業
佐藤	営業支援
佐藤	企画立案
伊藤	営業支援
伊藤	システム開発

結合

```
SELECT "社員_スキル"."社員"
, "社員_スキル"."スキル"
, "社員_担当業務"."担当業務"
FROM "社員_担当業務", "社員_スキル"
WHERE "社員_担当業務"."社員"
= "社員_スキル"."社員"
AND "社員_スキル"."社員"
= "社員_スキル"."社員"
```

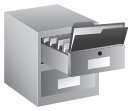
③

社員	スキル	担当業務
田中	コミュニケーション	営業
田中	顧客開拓	営業
佐藤	コンサルティング	企画立案
佐藤	新規事業開拓	企画立案
佐藤	コンサルティング	営業支援
佐藤	新規事業開拓	営業支援
伊藤	基本設計	システム開発
伊藤	要件定義	システム開発
伊藤	基本設計	営業支援
伊藤	要件定義	営業支援

元に戻りません。B ↠ Cという表を作り、それも含めて結合を行うことで元の状態に戻ります。それで第五正規形を満たしたといえます。

図9-1-①を見てみましょう。これを第四正規形で行ったように、自明でない多値従属性を分解します(**図9-1-**②)。次に、この2つの表を結合するとどうなるでしょうか(**図9-1-**③)。灰色で示した行のように、最初に定義したものではないデータが発生しています。これはB ↠ Cという関係を考慮しないために起こる現象です。

ですので、その表(スキル_担当業務)を追加し、再び結合してみます(**図9-2**)。無事に元の表と同じになりました。結合従属性が保たれたことにより、第五正規形を満たしたものであるといえます。



実際に正規化をしてみよう

第五正規形までを俯瞰的に見てみると、現実の設計業務においては第三正規形までで十分だということがご理解いただけたでしょう。第三正規形より高次の正規形は意識しないでも分解してしまっていることが多いので、普段見慣れないだけなのです。

理屈がわかったところで、ここで実際に正規化をやってみながら、正規化のステップを確認していくことにしましょう。さらにDBを構築し、現実の業務に合わせた検証までのステップを説明したいと思います。**図10-1**のような「作業報告書」を例題とします。

非正規形から第一正規形へ

ある業務のDBを構築する際には、実際に業務で使用している伝票や台帳などの書類を見るのが一番です。DBに必要な項目はほとんどそこにあります。その項目それぞれがドメインであり、項目の集まりがタプルであり、伝票自体がリレーションだと考えれば良いわけです。

では始めましょう。最初の段階では非正規化状態で良いので、項目を並べてみます。ExcelやCalc(OpenOffice.org(以下、OOo)、Libre Office)などの表計算ソフトを使って1つのシートにデータ項目を並べていくとわかりやすいかもしれません。**図10-2**のような表が書けたの

▼図9-2　第五正規化(結合従属性が保たれている)

社員_スキル

社員	スキル
田中	コミュニケーション
田中	顧客開拓
佐藤	コンサルティング
佐藤	新規事業開拓
伊藤	基本設計
伊藤	要件定義

社員_担当業務

社員	担当業務
田中	営業
佐藤	営業支援
佐藤	企画立案
伊藤	営業支援
伊藤	システム開発

スキル_担当業務

スキル	担当業務
コミュニケーション	営業
コンサルティング	営業支援
基本設計	システム開発
顧客開拓	営業
新規事業開拓	企画立案
要件定義	営業支援

↓

社員	スキル	担当業務
田中	コミュニケーション	営業
田中	顧客開拓	営業
佐藤	コンサルティング	営業支援
佐藤	新規事業開拓	企画立案
伊藤	基本設計	システム開発
伊藤	要件定義	営業支援

結合

```
SELECT "社員_スキル"."社員"
        , "社員_スキル"."スキル"
        , "社員_担当業務"."担当業務"
FROM "社員_担当業務", "社員_スキル", "スキル_担当業務"
WHERE "社員_担当業務"."社員" = "社員_スキル"."社員"
AND "スキル_担当業務"."スキル" = "社員_スキル"."スキル"
AND "社員_担当業務"."担当業務" = "スキル_担当業務"."担当業務"
```

ではないでしょうか。

非正規化データは表計算ソフトを使って普通に表現できるものですが、RDBMSでは扱えないわけですから正規化をしていきます。まず第一正規形は繰り返しを排除することでした。**図10-3**のようになりますね。

第二正規形へ

次に第二正規形です。この表の中で主キーを見極めましょう。主キーはタプルを一意に特定できるものですから、ここでは{日報番号,日時,プロジェクト,作業項目}ということになりそうです。

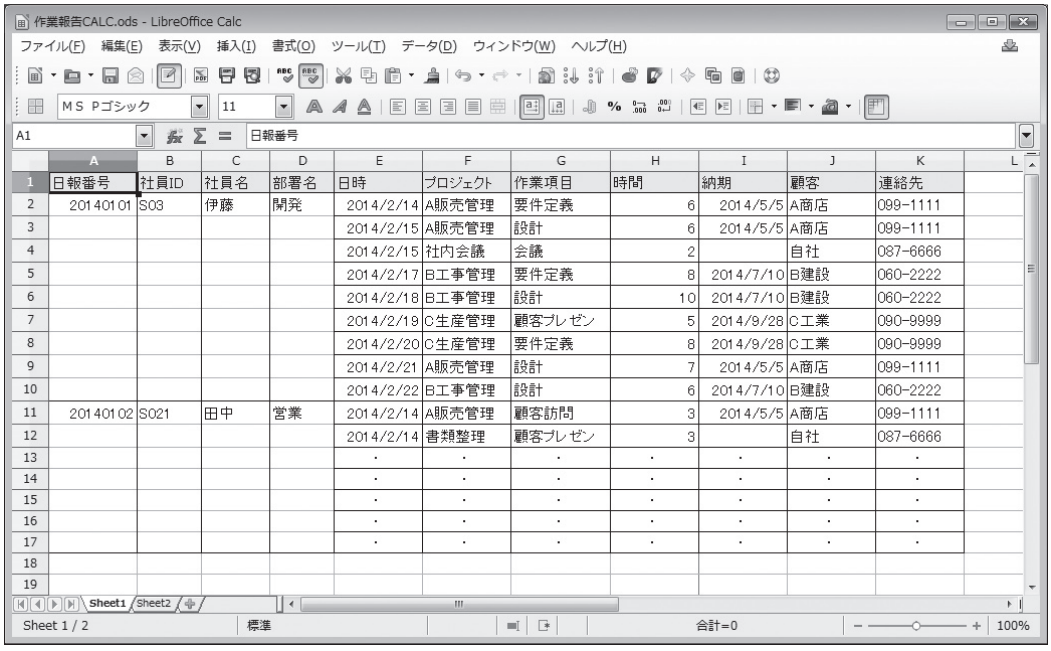
第二正規形は候補キーの中から部分関数従属するものを探すことでした。{日報番号,日時,

▼図10-1　作業報告書

作業報告書

日報番号	20140101		部署名		開発	
社員ID	S03		社員名		伊藤	
日時	プロジェクト	作業項目	時間	納期	顧客	連絡先
2014/2/14	A販売管理	要件定義	6	2014/5/5	A商店	099-1111
2014/2/15	A販売管理	設計	6	2014/5/5	A商店	099-1111
2014/2/15	社内会議	会議	2		自社	087-6666
2014/2/17	B工事管理	要件定義	8	2014/7/10	B建設	060-2222
2014/2/18	B工事管理	設計	10	2014/7/10	B建設	060-2222
2014/2/19	C生産管理	顧客プレゼン	5	2014/9/28	C工業	090-9999
2014/2/20	C生産管理	要件定義	8	2014/9/28	C工業	090-9999
2014/2/21	A販売管理	設計	7	2014/5/5	A商店	099-1111
2014/2/22	B工事管理	設計	6	2014/7/10	B建設	060-2222

▼図10-2　表計算ソフトで項目を並べる（LibreOffice Calc）



	A	B	C	D	E	F	G	H	I	J	K	L
1	日報番号	社員ID	社員名	部署名	日時	プロジェクト	作業項目	時間	納期	顧客	連絡先	
2	20140101	S03	伊藤	開発	2014/2/14	A販売管理	要件定義	6	2014/5/5	A商店	099-1111	
3					2014/2/15	A販売管理	設計	6	2014/5/5	A商店	099-1111	
4					2014/2/15	社内会議	会議	2		自社	087-6666	
5					2014/2/17	B工事管理	要件定義	8	2014/7/10	B建設	060-2222	
6					2014/2/18	B工事管理	設計	10	2014/7/10	B建設	060-2222	
7					2014/2/19	C生産管理	顧客プレゼン	5	2014/9/28	C工業	090-9999	
8					2014/2/20	C生産管理	要件定義	8	2014/9/28	C工業	090-9999	
9					2014/2/21	A販売管理	設計	7	2014/5/5	A商店	099-1111	
10					2014/2/22	B工事管理	設計	6	2014/7/10	B建設	060-2222	
11	20140102	S021	田中	営業	2014/2/14	A販売管理	顧客訪問	3	2014/5/5	A商店	099-1111	
12					2014/2/14	書類整理	顧客プレゼン	3		自社	087-6666	
13					・	・	・	・	・	・	・	
14					・	・	・	・	・	・	・	
15					・	・	・	・	・	・	・	
16					・	・	・	・	・	・	・	
17					・	・	・	・	・	・	・	
18												
19												

プロジェクト,作業項目}の属性で関数従属しているものは何でしょうか。

日報番号→ {社員ID,社員名,部署名}
プロジェクト→ {顧客名,連絡先,納期}

となります。これらの属性を別の表に分割しましょう。**図10-4**のようになります。灰色になっている部分が主キーです。

第三正規形へ

次に第三正規形への落とし込みです。第三正規形は推移関数従属を探すことです。日報表を見ると、日報番号→社員ID、社員ID→ {社員名,部署名} という推移関数従属を見つけられます。またプロジェクトを見ると、プロジェクト→ {顧客名、顧客名→連絡先} という推移関数従属も見つかりました。これらを分解します(**図10-5**)。

これで第三正規形まで落とし込めました。この時点でボイスコッド正規形、第四正規形、第五正規形ともに満足しています。現実の設計では第三正規形までで終了と思って差し支えありません。

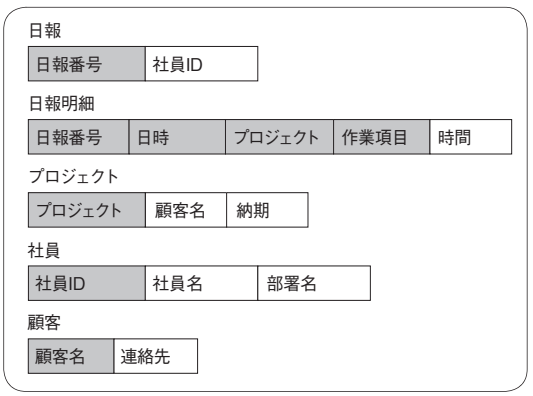
ですが、あと一歩踏み込みましょう。実際の設計ではこれで終わりではありませんね。候補キーに変更が予想される項目があります。たとえば、「プロジェクト」や「顧客名」などです。これらは通常IDと名称に分けるのが普通です。そうすることでプロジェクト名や顧客名を問題なく更新できるからです。「作業項目」なども名称変更の可能性がありますので、念のためIDと名称に分けることにします。さらに、主キー

▼図10-3　第一正規形

日報番号	社員ID	社員名	部署名	日時	プロジェクト	作業項目	時間	納期	顧客	連絡先
20140101	S01	伊藤	開発	2014/2/14	A販売管理	要件定義	6	2014/5/5	A商店	099-1111
20140101	S01	伊藤	開発	2014/2/15	A販売管理	設計	6	2014/5/5	A商店	099-1111
20140101	S01	伊藤	開発	2014/2/15	社内会議	会議	2		自社	087-6666
20140101	S01	伊藤	開発	2014/2/17	B工事管理	要件定義	8	2014/7/10	B建設	060-2222
20140101	S01	伊藤	開発	2014/2/18	B工事管理	設計	10	2014/7/10	B建設	060-2222
20140101	S01	伊藤	開発	2014/2/19	C生産管理	顧客プレゼン	5	2014/9/28	C工業	090-9999
20140101	S01	伊藤	開発	2014/2/20	C生産管理	要件定義	8	2014/9/28	C工業	090-9999
20140101	S01	伊藤	開発	2014/2/21	A販売管理	設計	7	2014/5/5	A商店	099-1111
20140101	S01	伊藤	開発	2014/2/22	B工事管理	設計	6	2014/7/10	B建設	060-2222
20140102	S02	田中	営業	2014/2/14	A販売管理	顧客訪問	3	2014/5/5	A商店	099-1111
20140102	S02	田中	営業	2014/2/14	書類整理	顧客プレゼン	3		自社	087-6666
・	・	・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・	・	・

▼図10-4　第二正規形

▼図10-5　第三正規形



を参照している外部キーに(FK)と書き、明確にしておきましょう(**図10-6**)。

これで正規化が完了した段階ですが、表同士の関係が一目見てわかるわけではありません。正規化により表も細かく分割されましたので、人に説明するのもたいへんそうです。そこで通常は、DBの設計にはER図を描き、設計書類とし、レビューやチェックを行います。

ER図を描き ビジネスルールを検討

誌面の関係でER図の描き方を説明する余裕はありませんので、正規化された表をER図に落とし込んだイメージ(**図10-7**)で説明しましょう。

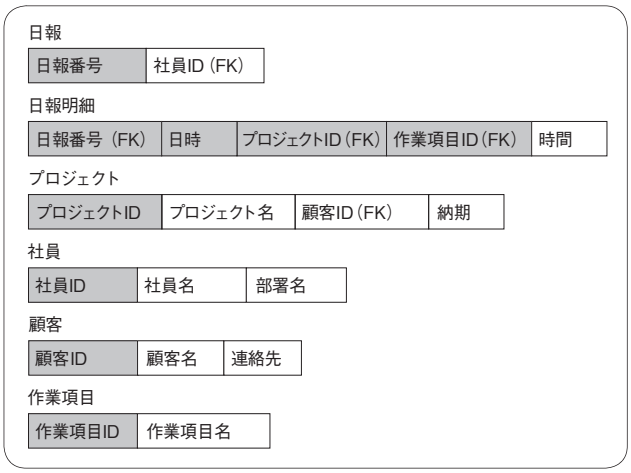
ER図を描くことにより正規化されたデータがどのような関係で結び付いているのかが、よくわかります。そして、これを眺めることでどのような制約をかけたほうが良いのかが、見えてきます。たとえば業務と照らし合わせながら、それぞれの表の制約などを決めていきます。具体的には、たとえば次のようなレビューを行います。

①「日報」「プロジェクト」「作業項目」は日報明細に対して、1：Nの関係にある。そしてNull制約を定義し値がNullであることを許可しない。これによりデータの欠け落ちを防御できる

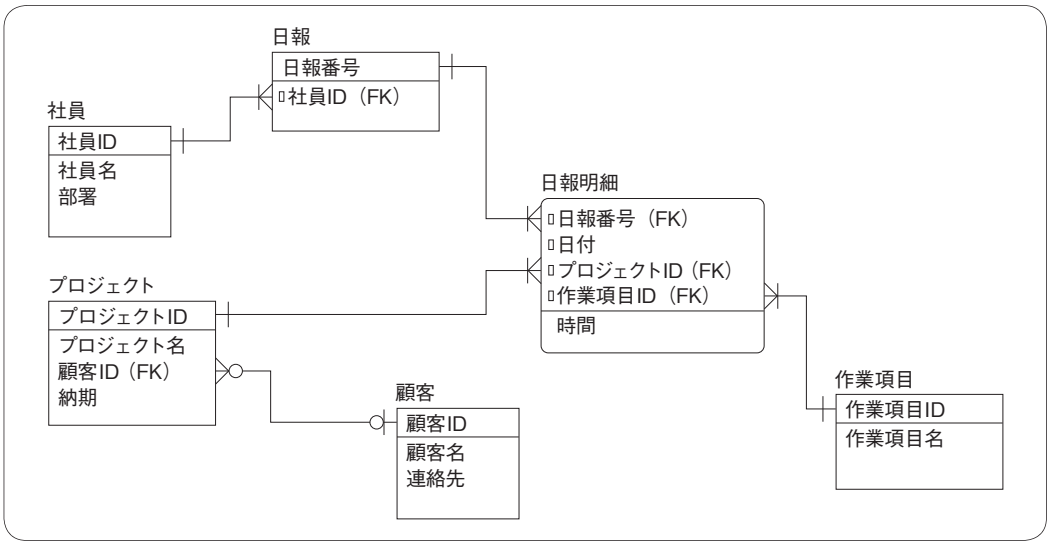
②プロジェクトと顧客の関係は、0または1:Nにし、顧客が決まらなくても架空のプロジェクトを発足することができる

また、これと同時に正規化された構造の中で、現実のビジネスルール

▼図10-6　IDを追加し、外部キー(FK)を明記する



▼図10-7　ER図



との矛盾がないかを検証します。たとえば次のような検証です。

❶部署は社員表の中で管理しているが、社員が異動により部署が変わった場合、過去の日報明細は現在の部署で集計されてしまうがそれでいいのか？
❷日報明細において主キーが{日報番号,日付,プロジェクトID,作業項目ID}であるが、作業項目が決まらなくても日報明細に書いておくという業務は存在しないのか？
❸❷と同様に同じ日付で同じ作業項目の追加記入はないのか？　追加があった場合は同じ作業項目の時間を上書きすればいいのか？

いかがでしょう。業務がからんでくると、どんどんデータ構造についての矛盾や疑問が浮かんできます。それをどのようにビジネスルールにしていくかで、DBの設計は変化するものです。ここでは、話が複雑になるので設計レビューは済んだことにして次のステップに進みます。

データベースを構築

次は実際のDBを構築する段階です。ER図を元にDB定義であるDDLを書いていきます。CREATE TABLE、ALTER TABLEなどのSQL文を列挙するわけです。そして、できあがったDDLをPostgreSQL上で実行すれば、DBが構築されます。今回は例としてPostgreSQLを使用しました。

さて、ここで提案したいことがあります。ER図から1つ1つSQL文を組立てていくのは勉強のためには良いのですが、ER図の変更やスキーマの変更があるたびに、DDLを組立てなおさなくてはなりません。設計段階では変更はかなりの頻度で発生しますので、忙しい現場だとたいへんなストレスになります。

そこで、ツールを使います。最近はER図を作成するツールがいくつかあり、DDLを自動的に出力してくれるものもあります。そういった機能を使うと変更があってもスムーズに対処できます。ちなみに今回著者が使ったのは「A5:SQL MK2」というツールで、無料でダウンロードできます[注1]。

A5:SQL MK2はDDLを自動出力してくれるとともに、DBとの接続もできるので、出力したDDLをそのまま実行してDBを構築できます。DBの中身を参照更新できる機能も付いているので、テストデータを作成したり参照したりでき、非常に優れたツールです。今回テストしたDBはこのツールで構築しました。

DBに接続してテストデータを作成

DBが構築できたら、設計は終了かというと、そうではありません。理論的に成立していても、データを入力していく中で、いろいろなことが見えてくるものです。できあがったデータ構造の使い勝手なども含めて検討する必要があるのです。

ですので、なるべく簡単にテストデータを入力して検証作業が行える環境が必要になります。著者のお勧めは、オープンソースであるOOoやLibreOfficeのBaseなどのDBソフトをフロントエンドにしてバックエンドに接続し(今回はPostgreSQL)、簡易的な入力画面を作ってしまうことです。ちょっと手間はかかりますが、テスト用アプリケーションプログラムを書くのに比べたら、検証できるまでにかかる早さは比較になりません。

ぜひともやりたいことは、**図10-8**のように一画面に関連するテーブルを全部入れてしまうことです。見通しが良いので簡単にストレスなくデータ作成や検証ができます。

こういった入力画面を作る際のちょっとしたコツを述べます。日報と日報明細のテーブルはともに日報番号で関連付けられているので、日報の行の選択が変わるごとに同じ日報番号の明細データが表示されるようにします。

日報明細のプロジェクトIDや作業項目IDなどの入力は、コードだとわかりにくいので、リ

注1）http://www.wind.sannet.ne.jp/m_matsu/developer/a5m2/　このソフトウェアは無料ですが寄付を歓迎しています。

ストボックスコントロールに変更し、ドロップダウンで実際の項目名称が選択できるようにします。また、日付項目もカレンダーを使って入力できるようにします。これにより入力の効率が格段に上がります。

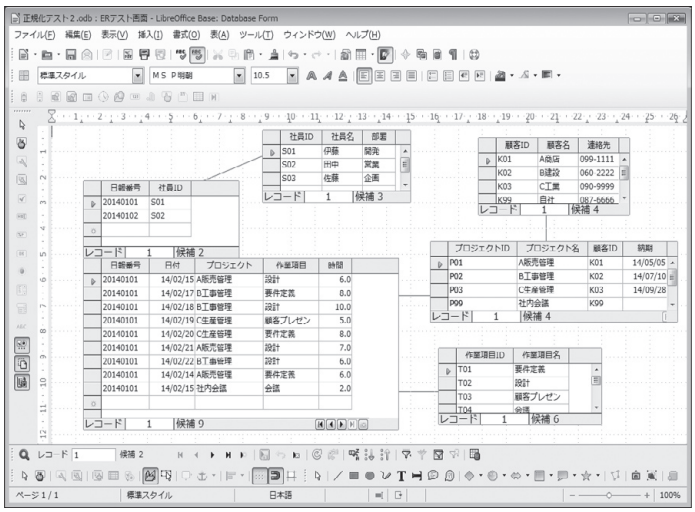
テーブルの関係が一目でわかるように、関連するテーブル同士を線で結ぶようにお絵かきすると良いですね。誌面の関係でBaseを使った入力画面の作成の方法は、別の機会に述べたいと思います。

ビジネスルールを検証、正規形を崩すこともある

さて、このようなテスト用の入力画面を用意して何をするかというと、各種の整合性／一貫性制約のチェックを行うと良いです。制約が緩過ぎないか、きつ過ぎないかなどです、たとえば次のようなチェックを行います。

(1)社員表の社員を登録していなくても、日報の社員IDを登録できるか？
(2)顧客IDが決まらなくても、プロジェクトは登録できるか？
(3)日報明細にプロジェクトが存在するのに、該当のプロジェクトを削除できるか？
(4)主キーにNullが入力できてしまわないか？

▼図10-8　テスト用データ入力画面（LibreOffice Base）



など、さまざまなテストが考えられますよね。

そしてさらに、現場のビジネスルールにあっているかどうか、チェックしながらテストをします。これによりさらに正規化するか、わざと正規化を崩すかということも見えてくるはずです。

たとえばER図の設計レビューの際に、問題になった「部署変更に伴う過去のデータの集計」について、この検証の時点で「やはり過去の部署は過去の部署として集計したい」と仕様が変更になった場合、構造を修正する必要があります。具体的には日報の中にも「部署」という属性を設けて過去と今を意識的に分けることになります。これは第三正規形ではなくなり第二正規形への逆戻りとなるわけですが、ビジネスルール上は解決します。ER図を描き直し、DBを再構築して、またテストし、検証します。その繰り返しです。それが設計の現場の日常なのです。

まとめ

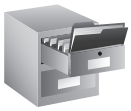
以上、駆け足で正規化の理論と実際を見てきました。読者の理論武装に役立てれば幸いです。

正規化は理論として成り立っており、実際の設計においても有効であることは間違いありません。しかし、最終的に決定する正規化のレベルは、やはり現実の業務のありかたで決まるということです。これはNoSQLの選定や設計でも同じことです。誤解を恐れずに言えば、NoSQLの最大の売りである高速性と分散性は、非正規化による現実との折り合いの結果ではないでしょうか。そういったことをふまえながら正規化について再考すると、RDBかNoSQLかという議論がいかに不毛なものかをご理解いただけるのではないかと思います。SD

第3章 分散DBの適用範囲とは ——その概要と将来を見据えて

Writer 神林 飛志(かんばやし たかし) 上新 卓也(うえしん たくや) (株)ノーチラス・テクノロジーズ

前章のRDBから一転して、本章ではNoSQLについて解説します。前半では、分散DBという視点から、現在のNoSQLがどんな領域に適用できるかを整理します。後半は、高いスケーラビリティと可用性だけでなく、RDBのような一貫性を実現したGoogle F1を紹介します。そこから分散DBの将来像を展望します。



分散DBの定義／位置づけ

現状の分散DBは、HBase、MongoDB、Riakを筆頭にOSS系を中心に発展してきたNoSQLに現在は代表されています。これ以外にもVoltDB、DynamoDB(AWS)、F1(Google)も分散DBであれば外せない製品ですが、後者は純粋にはNoSQLとは言いがたいので、いったんは枠の外においておきましょう。

一方、現状で一般に使われているデータベースはRDBMSです。Oracle、MySQL、PostgreSQLあたりが主流でしょう。従来は商用製品が主流でしたが、最近ではOSS系も使われつつあります。もともとNoSQLは、RDBMSでは処理しきれないデータ量、アクセス頻度を処理するために考案され、実際のユースケースとしてはとくにWeb系のしくみとして発展してきました。したがって、出自から見れば、NoSQLとRDBMSはそれぞれの「最もパフォーマンスが発揮できる場面」が異なり、本来は相互に補完関係にあります。しかし、それぞれ独自の発展と、運用上できれば単一のDBで管理したいというユーザニーズもあり、徐々に競合関係になりつつあります。

以上をふまえ、お題である「分散DBの適用範囲」について、「現状の分散DBはどこに使えるのか？」という点を考えていきたいと思います。また、今現在も世界各地(除く日本)で最も開発が競争的に行われているものの1つはこの分散DBですので、現状だけではなく、今後の展望もふまえて分散DBの適用範囲を考えていきましょう。なお、本章では、HBase、MongoDB、Riakあたりをまとめてと NoSQLという言い方をさせていただきます。

NoSQLの定義をどう考えるのか？

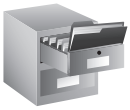
もともとはNot Only SQLという表現が、そもそものNoSQLのことの起こりです。しかし、もはや現状ではどのNoSQLも、SQLライクのインターフェースがそれらのエコシステムで準備されていることが多く、Not Only SQLという枠でくくるやり方はほとんど無意味になっています。ここではNoSQLの出自がRDBMSの限界を超えたところという点から、その定義を考えていったほうが良いでしょう。

RDBMSの限界は、大規模なデータの取り回しでパフォーマンスがボトルネックになりやすいという点です。これに対して、NoSQLは、規模が足りないときには、ノードを追加してクラスタをスケールアウトさせるという戦略をとって問題をクリアしています。

とはいえ、すべてに万能のしくみはありません。NoSQLも例外ではありません。ある機能やパフォーマンスを獲得すれば、その一方で捨てざるを得ないものもあります。NoSQLがRDBMSの限界を超えるパフォーマンスを得るために、

捨てた大きな資産の1つが一貫性になります。一貫性はRDBMSで最も重要かつ重厚なしくみの1つです。これをNoSQLでは、ほぼきれいさっぱり捨て去りました。

ただし、まったく捨て去ってはそもそもデータベースとして使い物にはならないため、最低限の“一貫性”のみを保持しています。一般に結果一貫性(Eventually Consistency)と言われるものです。これは「最終的にはいつかデータの整合性は合うはずです」という最低限の保証です[注1]。RDBMSのパフォーマンスを凌駕する読み込み／書き込みを獲得する一方で、一貫性をある程度犠牲にした“分散DB”をここではNoSQLという定義に当てたいと思います。



分散DBはどこに使えるのか?

まずもって分散DBがRDBMSの持つような一貫性を放棄している以上、一貫性を必要としているしくみには基本的には分散DBは使えません。後述するようにこのあたりは今後大きく変わると思われますが、現時点では分散DBはRDBMSのような一貫性を必要とするような用途には使えないでしょう。一貫性を必要としない領域ということであれば、とりあえず仔細かまわずデータをためておけ、という分野や、ほかのデータとの整合性は問わないが、とりあえず準備されているデータをとにかく見せろという分野に絞られていきます。これは現状では、Webのログの収集、組込み系のしくみから自動的に転送されるログの収集、これに類似する各種ライフログの収集、また、できあがったデータの閲覧、といった分野になっていきます。

上記の領域は、今流行の「ビッグデータ」の領域と重なります。ビッグデータの領域とは、過去の技術では解析できなかった大量のデータを収集／解析し、今まで発見されていなかった事象や傾向を見つけ出し、積極的に利用していこうという動きです。このビッグデータの基盤として、現在の分散DBは適当ですし、実際ビッグデータの隆盛と分散DBの流れは軌を一にしています。つまりデータを大量にためて、解析し、結果をバシバシ見せろ、というしくみですね。

さてそれでは、今まで十把一絡げにNoSQLと評していた分散DBをより細かく見ていきましょう。単純に大量のリードとライトに適しているといっても、各製品により癖があります。そのあたりを大まかになりますが、適用範囲ということで追っかけていきましょう。

HBase

HBaseは、Apache Software Foundation(ASF)により開発が行われているカラム指向データベースです。Google社によるBigtableを参考に、同じくASFにより開発が行われているApache Hadoopの分散環境上に構築され、大規模データに対する低遅延なランダムアクセスが可能です。

データモデル

HBaseのテーブルは、行を特定するためのRowKey、カラムの種類に応じた複数のColumnFamily、カラムを特定するためのQualifierからなります。ColumnFamilyはテーブルを作成する際に決定する必要がありますが、そこに含まれるQualifierは必要に応じて指定できます。RowKey、ColumnFamily、Qualifierの組み合わせで、1つのCellを特定します。このCellがRDBMSにおけるテーブルの特定行の列に相当しますが、Cellは複数バージョンのデータを持てます。また、各行はRowKeyでソートされています。

RowKeyを指定して1行ずつデータを取得(Get)できるほか、RowKeyの範囲で一気にデータを取ってくること(Scan)ができます。GetやScanの際にはCellのバージョンの範囲を指定したり、Filterをかけたりすることで取得する

注1) なお、分散系の一貫性という概念とRDBMSでいう一貫性の概念は本質的に異なるものですので、両者を峻別せずに一貫性という言葉を使うことは極めて不適切です。ですが、ここでは、データの最低限の整合性を保つという非常に広い意味で“一貫性”という言葉を使わせてください。

データの絞り込みが可能です。

データの書き込み(Put)を行う際には1行ずつ処理することになりますが、この1行の操作に対しては“一貫性”を保証しています。つまり、書き込み操作が完了したら、すべてのクライアントからは最新のデータを取得できるということです。また、CAS(Compare And Swap)操作や数値のインクリメント処理もサポートしています。

シャーディング

HBaseのテーブルはRowKeyの範囲でリージョンと呼ばれる単位に分割し、これを各ノードに分散配置することでシャーディングを行います。リージョンのサイズが設定値よりも大きくなると自動的にリージョンの分割が行われ、それぞれノードに配置されます。標準ではセカンダリIndexをサポートしていないため、想定するアクセスパターンによってRowKeyの設計を厳密に行う必要があります。

たとえば、テーブルへのランダム書き込みはほとんどメモリ内の処理だけで終わるため非常に高速です。しかし、前述したようにHBaseのテーブルはRowKeyの範囲で分割されたリージョンごとに分散配置されており、RowKeyの値が近い大量のデータを書き込もうとすると特定のノードに書き込みが集中するため、そのような状況が発生しないようにRowKeyがうまく分散するようにしなければなりません。

逆に読み込みについてはScanが高速に処理できるため、同時に読み込む必要のあるデータはRowKeyを近い値に設定しておくべきです。

このように、書き込みと読み込みのどちらが多いのか、どのようなデータを同時に読み書きするのか、などといったアクセスパターンに応じてRowKeyを設計していきます。

冗長化

HBaseは下層にあるHDFS(Hadoop Distributed File System)を利用することでデータの冗長化を行っています。Putなどのデータ操作はHDFS上に先行書き込みログ(WAL：Write Ahead Log)として書き込まれます。また、データはある程度メモリ内で処理され、設定されたデータ量を超えると、HDFS上にHTableファイルとしてフラッシュ(書き込み)されるようになっています。HDFSへの書き込みは複数個レプリカが作成されるため、特定のノードがダウンしてもWALとHTableからリージョンを復元できます。

使いどころ

HBaseはランダム書き込みとRowKeyの範囲を指定したシーケンシャルリードが得意です。たとえばLINEやFacebookなどのメッセージ機能のようなデータを格納するには、ユーザIDをRowKeyの頭に付けておくことで、多数のユーザによって書き込みを分散すると同時に、各ユーザのメッセージについては近くにまとまることになりますので、あるユーザのメッセージをまとめて持ってくることが容易にできます。ユーザごとのアクセスログを記録する、という用途も同様の考え方が使えます。

さまざまな切り口でアドホックなクエリを利用する場合には向いていません。セカンダリIndexが利用できないため、RowKey以外で検索する場合にはテーブルをフルスキャンしなければなりません。どうしてもRowKey以外の検索が必要な場合には自分でIndexを管理する必要があります。ただし、クエリの結果を取得するのに時間をかけてもいい状況であれば、MapReduceやHive、Pigなどと連携することで取得することは可能です。

MongoDB

MongoDBはMongoDB社によって開発されているドキュメント指向データベースです。JSON形式[注2]で表された「ドキュメント」を単位

注2) 内部的にはJSONのバイナリフォーマットであるBSON形式で格納している。

としてデータを格納します。RDBMSのようにスキーマが決まっているわけではなく、それぞれのデータを独自のドキュメントとして保存できます。いわゆる非構造化データをスキーマレスで扱えるわけです。そのため、あらかじめスキーマを規定できないようなシステムや、システムの成長に合わせてスキーマが拡張していくようなケースで利用できます。

データモデル

MongoDBのデータベースの中には、「コレクション」と呼ばれるテーブルに相当するものがあり、そこにドキュメントを格納していきます。ドキュメントに"_id"という特殊なキーの値を付けることで特定のドキュメントを取得できるようになります。"_id"フィールドがない場合には自動的にObjectId型の値が設定されます。

フィールドの値や範囲などの条件を組み合わせたクエリを実行できたり、また必要なフィールドのみを指定してデータを取得できたりと、柔軟な検索が可能です。クエリ言語自体がJavaScriptであるため、演算子を定義したり、Indexの利用はできなくなるものの、問い合わせ処理自体を実装したりすることもできるようになっています。

コレクションには"_id"フィールドに対してIndexが作成されています。またセカンダリIndexを作成できるので、クエリの高速化が期待できます。利用できるIndexは、Bツリー、ハッシュ、2次元Index、地理空間Index、およびそれらの複合Indexなどが挙げられます。

データの解析など、集計を行う場合には標準的な集約関数が利用できるほか、MapReduceも利用できます。

シャーディング

MongoDBは、Shardキーを元に水平分割されていきます。Shard内のデータが設定されたデータサイズを超えると、そのShardが自動的に分割されます。Shardキーは自由に決定できますが、この選択は非常に重要になります。たとえば、キーに指定したフィールドの取り得る値の種類が少ない場合には、たかだかその種類の数までしか分割されないことになり、データが分散されなくなってしまいます。

Shardキーを含むクエリは指定されたデータを含むShardのみでクエリを実行することで負荷分散できます。逆にShardキーを含まないクエリは全Shardでクエリを実行してからマージする必要があります。また、Shardキーを含まない場合に問題が生じるクエリがあるので注意しましょう。

一度設定したShardキーは変更できないので、事前に十分検討する必要があります。

冗長化

MongoDBで冗長構成をとるには、各ShardをPrimaryノード1台とSecondaryノード複数台を1組にしてReplicaSetという構成を作ります。Primaryノードに対する更新はSecondaryノードにレプリケートされ、各ノードでデータを冗長に持つことになります。

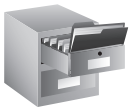
Primaryノードがダウンしてしまった場合には、残りのノードでPrimaryを選出し昇格します。Primaryノードの選出には投票権を持つノードによる多数決で行われます。投票権を持つノードは、ReplicaSetの中に奇数台必要で、過半数をとったノードがPrimaryとなります。ReplicaSetに奇数台のノードが準備できない場合には、Arbiterという、データを保持せず投票権のみを持つノードを追加して奇数台の構成にしておきます。

ReplicaSetを構成するノードは最大12台までで、投票権を持つノードはReplicaSet内に最大7台までとなっています。

使いどころ

MongoDBは、非構造化データを大量に扱う場合に有効です。スキーマレスですので、フィールドを追加するのもドキュメント形式を変更するだけです。RDBMSにおけるJOINのような

処理も、非正規化やドキュメントを入れ子状にして持つことで実現できます。セカンダリIndexも使えるので、あらかじめクエリを想定できるのであれば、そのクエリを高速に実行できます。さまざまな形式のデータをいったんMongoDBに入れておきさえすれば、出力時の柔軟なクエリやMapReduceを使って目的のデータを取得できるでしょう。



Riak

RiakはBasho Technologies社によって開発されているKey-Value Storeです。もともとはAmazon Dynamoの論文を参考に実装されていて、耐障害性や運用の容易性、スケーラビリティが特徴として挙げられます。データへのアクセスはHTTP REST API、または一部制限はありますが、Protocol Buffersも利用できます。

データモデル

データのキーは、バケットによって分類されています。同じキーであっても、別々のバケットに入っていれば違うものとして扱われます。

HTTP REST API経由で値にアクセスする際にはURLは次のようになっています。

```
http://${SERVER_NAME}:${PORT}/riak/
${BUCKET}/${KEY}
```

値をPUTする際にContent-Typeを指定することで、テキストやJSON形式のデータ、HTML、XML、また画像ファイルなどHTTPで扱える任意の形式のデータを格納できます。

キーとキーを関連付ける、リンクと呼ばれる機能もあります。あるキーから、リンクをたどってリンク先の値を取得できます。

また、MapReduceも利用できますし、バックエンドストレージによってはセカンダリIndexを利用できるので、これらを組み合わせることで任意のクエリを実行できます。

シャーディング

Riakにはマスタノードがなく、すべてのノードが対等であるため、SPoF(Single Point of Failure)が存在しません。クラスタにノードが追加されると自動的にデータのリバランスが行われます。逆に、ノードがダウンした場合であっても、ほかのノードが肩代わりすることで処理を継続できます。

データは、Consistent Hashingにより分散されます。バケットとキーの組み合わせから、160bitのハッシュ値を生成します。この160bitの整数空間をRiakではRiak Ringと呼び、どのノードにデータを置くのかを決定するのに利用します。

この整数空間を均等にいくつかのパーティションに分けて各パーティションが担当する値の範囲を決めておき、この生成したハッシュ値を範囲に含むパーティションがデータを持つことになります。これらのパーティションは、仮想ノードもしくはvnodeと呼ばれます。物理ノードはこの仮想ノードを、順番に複数個担当することになります。

たとえば、16パーティションに設定された4ノード(node[0-3])のクラスタであれば、各パーティションが受け持つ範囲の大きさは$2^{160}/16$、node0は0、4、8、12番めの4つのパーティションを受け持ちます。

冗長化

データを格納する際には、複数個のレプリカを作成します(レプリカの数はパラメータNで指定します)。レプリカの配置先は各パーティションの次のパーティションを受け持つノードになります。2つのレプリカを作成する場合(N=2)には次、3つの場合(N=3)にはさらにその次、というように次へ、次へとレプリケートされていきます。デフォルトでは初めのものを含めて全部で3つのレプリカを作る(N=3)ように設定されています。

データの書き込みの際には、いくつのレプリカが作成されたら書き込みが完了したとみなすのかを設定できます(パラメータWで指定します)。全部で3つのレプリカを作成するよう設定されている場合でも、1つレプリカが作成された時点で完了とみなす(W=1)こともできますし、3つとも作成されないと完了とみなさない(W=3)ように設定することもできます。

データを読み込む際には、レプリカがあるどのノードからでも読み込みができます。また、複数のノードからデータが読み込めた時点で読み込みができたとみなすように設定することもできます(パラメータRで指定します)。

これら3つのパラメータは、データの整合性とパフォーマンスの兼ね合いによって設定する必要があります。たとえば、N=3、W=3、R=1であれば、書き込み完了の時点ですべてのレプリカが作成できているので、どのレプリカから読み込んでも最新のデータが取得できますが、書き込みが遅くなってしまいます。また、N=3、W=1、R=1とすると、書き込みが完了してもすべてのレプリカが作成できているとは限らないので、古いデータを読んでしまうかもしれません。N=3、W=1、R=3であれば、3つ読み込んだ中に最新のデータが見つかりますが、読み込みが遅くなります。

データの整合性が問題となる状況であれば、W+R>Nとなるようにそれぞれの値を設定する必要があります。W、Rを指定しない場合、デフォルトでN/2+1となるようになっています。N=3であれば、W=2、R=2となります。

使いどころ

Riakは耐障害性に優れているため、可用性が優先されるシステムに向いています。たとえばユーザセッションの情報を格納したり、Amazon S3のようなストレージサービスとして利用したりできるでしょう。実際、S3 API互換のストレージサービスとしてRiakをベースとしたRiak CSというプロダクトがBashoよりリリースされています。

ビッグデータ的な使い方以外では?

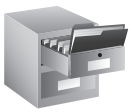
以上が、現在、隆盛を極めている分散DBの概要と想定されている用途です。では「ビッグデータ以外の分野では適用できないのか?」というのが、当然ですが次の注目点になります。まぁ日本ではビッグデータの市場は喧伝されているほど大きくないのが実情ですし、「Web系以外の一般業務に使えないのか?」ということは普通に検討されます。

まず、結論から言いますと、現状の分散DB、すなわちNoSQLはそれ以外の分野では、まずいっさい使えません。使えません、という言い方より「使いません」という言い方が正しいでしょう。要は、RDBMSを使います。

業務システムはやはりRDBMS

ビッグデータ系以外の領域であれば、これはもう一般的なごく普通の業務領域になります。昔からありますし、今後もあります。こういった領域では現状ではRDBMS一択です。業務的な要請でRDBMSで想定される一貫性が担保されている必要がありますし、また、現状のツール群や今までのノウハウ、現在までに蓄えられてきた資産もあります。これを全部捨て去って、業務システムをRDBMSから分散DBに移すには、メリットよりもデメリットのほうが大きいでしょう。もちろん、RDBMSにはある程度のスケールアウトの壁があるので、なんとか分散DBを導入したいという動きもあります。巨大になった商品マスターをRDBMSからHBaseに移すというプロジェクトもありましたが、結局「HBaseの上にRDBMSと同じ機能を一から作る結果」になり断念ということのようです。HBaseも良い製品なのですが、RDBMSと同じ土俵に乗っては、このように分の悪い勝負になってしまいます。

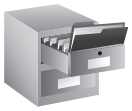
RDBとNoSQLどちらを選びますか？
真っ当に考えるDBの鉄則



では将来もだめなのか？

NoSQLで業務系システムを構築するのは、いろいろとハードルが高い、というのが現実です。では将来はどうなのか？　最先端の分散DBの状況を見ながら、考えていきましょう。

現在のところ、分散DBの開発のフィールドは、ほぼクラウド環境が前提になりつつあります。これは、よりスケールアウトさせるビジネスがクラウドで展開されていて、それを支えるデータベースが必要になるからです。クラウドでの技術開発という点では、やはりGoogleが相当先に進んでいます。分散DBの今後の発展の方向性を見定めるには、残念ながらGoogleの研究を見ていくことが第一でしょう。最新の動向は外に出さないというのが、Googleの流儀のようですが、最近では現在の潮流をふまえたうえでいろいろと公表しています。このあたりを見ながら、この先の分散DBの特質や使い方への示唆や、ほぼ確実に到来するであろうアーキテクチャを見ていきましょう。



Google F1

GoogleのメインビジネスであるAdWordsビジネスを支えている分散DB、これがF1です。この概要が公表されています[注3]。

F1の特長

最大の特長は、NoSQL並みの高いスケーラビリティと高可用性を確保しつつ、かつRDBMSの提供するACIDトランザクションと同等の一貫性を確保している点です[注4]。NoSQLとRDBMSの両者の弱点をなくした、まさに次世代のデータベースと言えるでしょう。以下に、そのアーキテクチャを中心に注目すべきポイントをかいつまんで見ていきましょう。

基本的アーキテクチャ

F1は、トランザクション管理とクエリ処理を行い、クライアントとの通信を制御するF1サーバと、そのクライアント、そしてその下位でトランザクション実行とデータレプリケーションを担当するSpannerで構成されています。Spannerのさらに下位にファイルシステムとしてCFS(Colossus File System)があります。

F1サーバにデータが保存されることはなく、すべてのデータは基本的にSpannerに保存されます。F1サーバもSpannerもそれぞれ分散アーキテクチャになっており、とくにSpannerは複数のデータセンターにまたがってデータの制御を行っています。F1サーバはまず同じデータセンター内部にあるSpannerサーバに問い合わせを行い、ロードバランスや可用性の確保が必要な場合に、外部のデータセンターのSpannerサーバに接続を行います。

F1サーバ自体はステートレスであり、クライアントは任意のF1サーバと接続できます。ただし、ロックをとるトランザクションを行うときは例外で、そのトランザクションの実行時は、特定のサーバにバインドされます。

また、クエリ処理は、予想されるレイテンシーを考慮し、分散処理と集中処理を選択して効率の良い処理を実行します。また必要に応じてMapReduceも実行します。基本的にクエリの制御はすべてF1サーバで処理されますが、MapReduceについては、アウトプットのボトルネックを発生させないために、クライアントから直接Spannerにアクセスさせて制御しているようです。

Spanner

さて、トランザクションの根幹を担っているのがSpannerと言われるミドルウェアです。Spannerの提供する機能は、下位レベルのストレージ制御、データキャッシュ、データレプリ

注3) http://research.google.com/pubs/pub41344.html

注4) なお、ここでの一貫性は、トランザクションの意味でのserializableを意味しています。

ケーション、可用性確保、データ分割(シャーディングと移動)、ロケーション管理、そしてトランザクション機能です。分散トランザクションのプロトコルは2PC(2 Phase Commitment)をPaxosプロトコルで実行するしくみになっています。

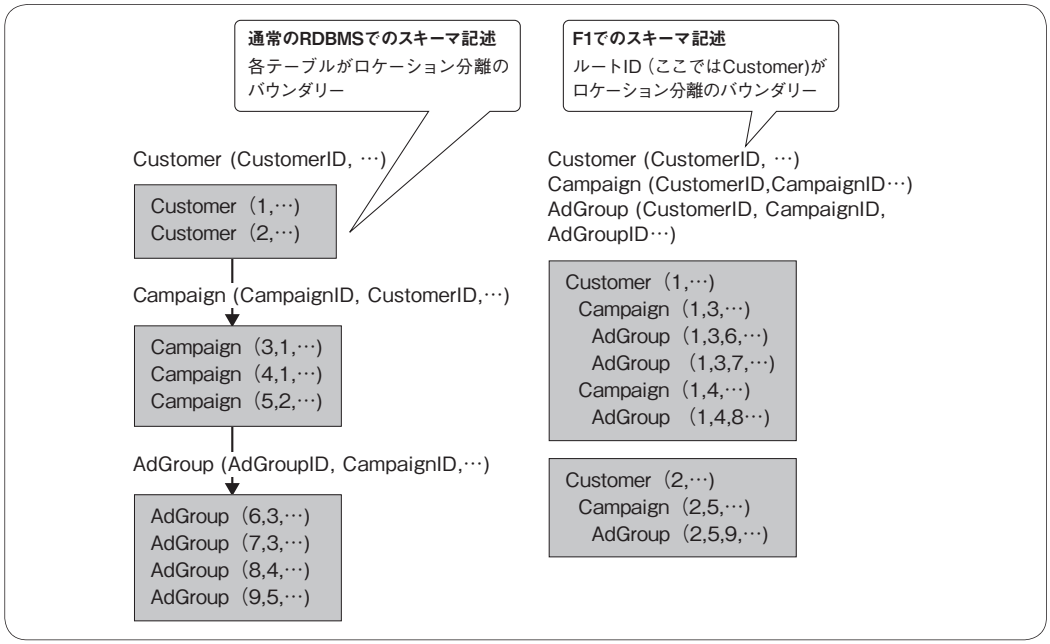
このしくみが公開されたときには、非常に狭い一部の界隈に、激震をもたらしました。後述するように、Spannerは、いわゆるSnapshot Isolationに準じるトランザクション制御を行っており、First-Commit-Winルールを採用しています。この手法はTimestampの順序確保が必要な条件なのですが、分散処理では時刻が合わない場合を前提にする必要があり、そのためなかなか分散環境での実装が困難でした。この課題をSpannerでは、ハードウェア(GPSと原子時計を利用)で解決するという卑怯技を使って乗り越えています。あぁ、やっぱりソフトウェアでは無理なのか。アルゴリズムでは無理なのか。Googleよ、おまえもかと、この道で頑張ってきた人たちが天を見上げる事態になりました。Googleがこの方法を示したことは、一種のパンドラの箱を開けることになっています。いや、ソフトウェアでできないならハードウェアで解決って、それアリですか……。現在、各社(除く日本のIT企業)が競ってこの分野に注力している結果になっているようです。

データモデル

F1のデータモデルは、基本的にテーブルモデルをベースにしたRelationalモデルと言って良いでしょう。ただし、各テーブルが独立にあるのではなく、行単位で各テーブルの行が挟み込まれるような階層構成になっています(**図1**)。

このテーブル構成のメリットは、キーが同一の場合、ある行から下位の階層の行へのアクセスが非常に速いという点です。また関連するデータの物理ロケーションが特定のノードに集中するため、不要に分散トランザクションをする必要もありません。とくに更新処理でのオーバーヘッドが軽くなります。筆者の見るところ、これはAdWordsの業務要請のアクセスパターンには最適な構成になっていますが、横断的な更新処理が頻発するような場合は、相当のコスト

▼図1　RDBMSのデータモデルとF1のデータモデル



が発生するように思われます。F1ではデータセンター間のトランザクションもサポートしているので、レイテンシーを下げるためにいろいろと工夫しているのでしょう。

各カラムに格納されるデータタイプはProtocol Buffersの形式になっています。このレベルだけ見るのであれば、Objectベースのデータベースとも言えます。Protocol Buffersがベースになっていることで、O/Rマッピングのオーバーヘッドを最小にすることが可能になっています。ただし、その代償としてProtocol Buffersの内部のフィールドに直接アクセスしたり、Indexを行ったりする場合には通常のプリミティブなタイプとは異なり、多少コストがかかっています。

IndexについてはローカルIndexとグローバルIndexの2つの要素から構成されています。ローカルIndexは単一の物理ロケーション、すなわちSpannerサーバに格納され、Index先の行と同一サーバに同居しています。また、グローバルIndexは複数のSpannerに分散配置されています。これは結果として、とくに大量のIndex更新処理のパフォーマンス劣化を引き起こします。ここはまだ、Googleでもいろいろと手段を検討しているようです。分散DBでのグローバルIndexは、その更新処理とあいまって決定的な良い解決案がないのが現状ですが、Googleも苦戦をしているということでしょう。

スキーマ変更

まずAdWordsのビジネス自体がGoogleの根幹の1つですので、当然データベースのダウンや停止は許されず、高い可用性を要求されます。このような要求の中で必要に応じてスキーマ変更をノンブロッキングで行う必要があります。NoSQL系のデータベースでもスキーマ変更が鬼門であることは周知のとおりです。F1では、スキーマ変更はかなりトリッキーな手法を採用しています。まず、スキーマ変更は、各サーバで非同期処理で行います。これによりノンブロッキングが確保できますが、当然データ不整合の原因となります。そこで、複数のスキーマを同時にアクティブにし、各スキーマ間で互換性があるように、与えられたオペレーションを操作します。スキーマ移行が完了した段階で、古いスキーマの使用を放棄します。この手法はおそらく、AdWordsのデータモデルを運用する場合には可能ではあると思われますが、一般的な分散DBではデータ不整合を防ぐために、相当のワークアラウンドをアプリケーションに課すことになり汎用的とは言えないでしょう。

トランザクション

F1ではACIDトランザクションを実現しています。複数データセンターにまたがる大規模分散DBでの分散トランザクションを、低遅延かつプロダクションレベルで、ACID要件として実現しているケースは世界で最初のものでしょう。以下に詳細を解説します。

まず、実現されるオペレーションを、複数の読み込みとそれに続く、必要に応じた書き込みと類型化したうえで、複数のトランザクションの手法を提供しています。

①Snapshotトランザクション

いわゆるリードオンリー・トランザクションになります。Timestampを基準にしたSnapshotを利用します。通常はローカル・レプリカを参照し、時刻としては5〜10秒の遅れた時刻を利用します。ただし、明示的に時刻を指定することも可能ですし、また現在時刻を指定することも可能です。Spannerの原子時計が提供する時間同期の面目躍如です

②悲観トランザクション

ロックをとるトランザクションになります。リードロックの場合は、共有ロックと排他ロックを選択することが可能です

③楽観トランザクション

典型的なオペレーションに対応したトランザクションです。リード時点では、ロックをとらず、書き込み時点でロックをとり書き込みを行いま

す。このとき、リードした行について読み込み時点での最終更新時刻より後で、更新処理がコミットされたことが検出された場合は処理をabortします。一般にSnapshot Isolationと言われるトランザクション・プロトコルで、最後に書き込みをコミットしたトランザクションが有効になるので、First-Commit-Winルールと言われます

F1ではデフォルトでは、楽観トランザクションになります。この手法の最も大きいメリットは、リードがノンブロッキングなのでパフォーマンスが劣化しないという点でしょう。一方デメリットは、F1の楽観トランザクションが比較的ナイーブなSnapshot Isolationの実装なので、固有のanomalyが発生することでしょう。よく知られているのは、いわゆるphantomと言われるもので、とくにInsert phantomは除去不能です。F1では、ここではデータモデルの特性をうまく利用してこの問題を解決する方法を提供しています。各テーブルが挟み込み形で階層構造になっているため、更新処理のあるテーブルの行の上位のテーブルの行をロックすることでInsert phantomを除去できます。通常よくある回避策はテーブルロックや、Indexを利用するものですが、F1ではスキーマ特性を利用しています。

トランザクションの書き込み実行時のプロトコルは2PCになっています。この合意形成はPaxosによっています。Paxosは分散環境の合意形成の手法としては、現在、最もデファクトに近いアルゴリズムです。ただし、その原論文が難解であることでも有名で、またさまざまなバリエーションがあり、相当に複雑です。ここでは以下のスライドを参考にしていただいたほうがいいと思います。

まず、日本の分散合意業界の若手エースの@kuenishiさんのスライドが以下。

```
http://www.slideshare.net/kuenishi/ﾌ
genpaxos-1679212
```

これまた、同じく日本の分散処理業界のホープの@nobu_kさんも貴重なスライドを公開してます。こちらもお勧めです。

```
http://www.slideshare.net/pfi/paxos-ﾌ
13615514
```

こちらはUstreamでの解説講義つきです。

```
http://www.ustream.tv/recorded/23776788
```

クエリ処理

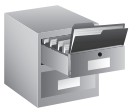
F1では分散クエリのしくみを導入しています。単ノードで処理可能な場合は、手元のノードで処理しますが、分散クエリのほうがより効率的だと判断される場合は分散処理されます。データはすべてSpannerに保存されているので、処理はすべてリモートアクセスになります。横断的に結合処理を行う場合は、複数のSpannerサーバにアクセスすることになります。そのためクエリ処理の実行パフォーマンスは従来のRDBMSと異なりネットワークの遅延に大きく左右されます。

ネットワーク遅延を向上させるためにF1では、パイプライン処理とバッチ処理を利用しています。たとえば結合処理は、個々のキーをまとめてバッチ的に各分散ノードにプッシュダウンして、各ノードで並列に処理を行います。この処理はパイプライン化され、ストリーミング的に処理されます。これは非同期の並列処理になります。高いパフォーマンスを維持することが可能ですが、その代償として個々の問い合わせの実行順序は保証されません。なお、データのパーティショニングは細粒度で行われているため、個々のノードのリソース競合が起こりにくくなっています。したがって、処理性能をリニアにスケールアウトすることが可能です。

クエリの実行計画の策定が非常に特徴的になっています。通常、分散DBでは、データの分散配置が一定のセマンティクスをもってパーティショニングされることを利用して、クエリのプッ

シュダウンを効率的に行いますが、F1ではこの戦略をとっていません。Spannerがデータのパーティショニングをランダムかつ動的に行うため、F1側でクエリ実行時にデータの再パーティショニングを行う必要があります。データ配置のパーティショニングはハッシュ・パーティショニングになっており、結果、統計データの取りようがなく、通常の最適化戦略がとれません。確かに、結合戦略としては効率的な分散ハッシュ結合を積極的に選択できますが、普通に考えれば、この手の戦略は完全に力技で、再パーティショニング時に簡単にネットワークが溢れてしまいます。普通にアウトです……が、Googleでは、これまたハードウェアで解決しているようです。ネットワーク機器を相当に強化して強引に処理しきっているようです。

従来のセオリーはネットワークの遅延コストをほかのリソースと比べて桁違いのオーバーヘッドと見て、戦略を立てます。とくにF1のようなマルチデータセンター間の処理が前提であれば、なおさらです。ですが、Googleはハードウェアを含めた環境をまるごと強化し、「ゲームのルール自体」を変えようとしているように見えます。実際、世界各地でダークファイバーを買いまくって、土管だけ借り、交換機は独自のものを持ち込んでいるというところまでやっているようです。ここまで前提が変わるのであれば、確かにこのような戦略も今後はありなのかもしれません。



今後の分散DBの方向性

さてF1の外観を見てきたところで、本筋に戻って、今後の分散DBの潮流と利用の方向性について検討してみましょう。

トランザクション

上記のように、すでにF1のような分散DB上でのRDBMS的な一貫性が担保されている以上、今後の分散DBの潮流は間違いなく、一貫性をサポートしたものになっていくでしょう。

分散トランザクションの手法としては、Snapshot IsolationとPaxosをベースにした2PCによる同期が当面の間は分散トランザクションの主流になると思われます。ただし、F1の手法がそのままデファクトになるかと言えば、さすがにそうではないでしょう。Snapshot Isolationにしても、F1のようにアプリケーションに制約をかけることなくserializableな実行が可能なアルゴリズムを利用するようなさまざまなバリエーションが検討されています。また、同期処理についても負荷の高い2PCではなく、よりlazyに同期処理を実行する手法も多数検討されています。いずれにしろ、従来のRDBMSと同等の一貫性をサポートしていくという流れは決定的なものに見えます。

クエリ処理

F1では分散クエリの結合戦略が、全面的にハッシュベースで行われている点が興味深い点です。これは従来のセマンティクスベースを利用した分散クエリの処理の理論とは違う方向です。従来の分散クエリの基本は、クエリのロジックとパーティショニングのロジックを合成して論理計算を行い、クエリ実行のコストが最も低い実行計画を作成し、これをもとに問い合わせ実行を行っていました。このコスト計算では従来はネットワークコストをかなり高めに見積もって計算するのが実態で、結果、できるだけデータのネットワークでの転送を抑える結合戦略が通常は選択されます。

これに対して、F1のハッシュベースのクエリ実行の手法は、デメリットはネットワーク負荷が簡単に上がってしまう点ですが、他方、大きなメリットとして分散のパーティショニングのロジックの設計に対する大きなフリーハンドを享受できます。現在の分散クエリの実行のスタイルでは、データ分散のロジック（Predicate）を、事前に発行され得るクエリをある程度先読みしながら設計していかないと分散のパフォーマンスが出ません。これは相当高度な設計技術が要求されます。

F1ではハッシュベースのメリットを享受しつつ、デメリットのネットワークパフォーマンスの負荷をハードウェアで回避するという手法を導入しています。結果として、設計によるパフォーマンス向上策を打つ必要がなく、実は現在の分散クエリ実行の難題への1つの解になっている気もします[注5]。

現状のGoogle並みのネットワーク環境が一般に享受できるようになれば、分散DBでのクエリ最適化戦略についても、よりユーザビリティの高いものに進化していくでしょう。

全体的なアーキテクチャから推測されること

分散処理の理論から見ても、大量のデータを分散処理させつつ、パフォーマンスの向上と一貫性の確保を両立させることは非常に困難です。また、実際のNoSQLを見ても実現できていないことは明らかです。この壁をF1は、ハードウェアの発展と、ミドル層の制限をアプリケーションに制約を課すことでクリアしています。ハードウェアでは、同期処理に原子時計とGPSを導入し、ネットワークのトラフィックが増大しても処理しきってしまうスイッチ群やネットワーク環境を利用しています。またアプリケーション層では、スキーマ構造をある程度制限することでナイーブなトランザクションプロトコルでも不整合が発生しないようにしています。

つまり、データベースという枠を取り払い、トータルのシステムとして分散DBの課題にあたることで解決案を導き出していると言えます。これは、今後の分散DBのあり方の1つにはなるでしょう。すなわち、ソフトウェアとしての分散DBではなく、ハードウェアの環境までも含めた分散DBというものが1つの形になるかもしれません。どのような形で提供されるかは、想像の域を出ませんが、一種のアプライアンスの形で提供されるか、またはクラウドサービスの形でのみ提供されるようなものになるかもしれません。また、ハードウェアベンダが、特定のハードウェア環境とセットで分散DBを提供してくる可能性もあるでしょう[注6]。

今後の適用領域

分散DBで一貫性が確保される以上、現在の単ノードでのアーキテクチャを基本とするRDBMSベースで稼働しているシステムは、そのほとんどが分散DBへ移行させることが理論上は可能になります。すなわち、現行の大半のシステムが、分散環境でのスケールアウトと高いパフォーマンスのメリットを享受できるようになると思われます。

ただし、分散DBの制約をさまざまなやり方で克服していく必要があるでしょう。この課題の乗り越え方については、Google F1は非常に示唆に富んでいます。理論的な課題になっている部分を理論で解決するのではなく、ハードウェアを解決する手法として援用している点は見逃せません。この結果、問題点を設計手法で回避する労力が軽減されています。これは複雑になった理論をいったん、簡易にリセットしている感じも受けます。

その一方で、スキーマを利用した回避策も実施しています。これはアプリケーションの開発とミドル層の制御が、分散DBでは表裏一体になっていることの現れでもあります。ハードウェアでの解決とは別に、分散DBでは設計／実装については今までのRDBMS以上に、配慮すべきことが増えるということでもあります。

分散DBの将来は、「確かに分散DBで通常の処理はできるようにはなった。ただし、本来ある分散DBの課題をハードウェアなり、ソフトウェアなりさまざまな形で解決している以上、その手法を考慮した設計／実装の手腕が求められることにもなってきた」ということになるでしょう。SD

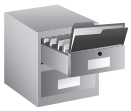
注5）ハードウェアでの解決は、解決にはなってないという考え方もありますが。

注6）実際にそういう動きがあるという噂も耳にします。

第4章 分散からあえてRDBへ
――『過負荷に耐えるWebの作り方』のエピローグ

Writer 林 哲也（はやし てつや）（株）パイプドビッツ　取締役CTO

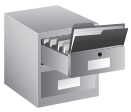
国民的アイドルグループ選抜総選挙を支えたWeb投票システムは、当初KVSであるVoltDBをデータストアとして利用していました。しかしシステムの運用が軌道に乗るにつれRDBであるPostgreSQLに移行しました。本章では、Web投票システムの事例を紹介しながら、KVSのメリット・デメリットを説明し、適材適所でのデータベース利用を解説します。



はじめに

本誌の読者の皆さんはVoltDBのことをご存じの方も多いことでしょう。インメモリDBでスケールアウト可能、しかもSQLでデータ操作が可能といった特長があります。しかし、実際に本番稼働システムとして利用した事例は少ないのかもしれません。VoltDB社の顧客紹介ページには、さくらインターネット社がありますが、「VoltDB」とGoogleで検索すると検索候補に「voltdb akb48」と出てくるように、3年前に選抜総選挙でVoltDBを利用した事例が取り上げられることも多いようです。

システムを担当した当社が、選抜総選挙でどのようにVoltDBを活用したかについては拙著『過負荷に耐えるWebシステムの作り方』（技術評論社）で紹介しています。そこでは主に2011年に開催された第3回選抜総選挙のシステムについての紹介でしたが、その後の投票ではVoltDBではなく、汎用的なRDBであるPostgreSQLに切り替えました。今回は、移行に至った理由を、VoltDBの特徴の説明を通じてご紹介します。



VoltDBとは

VoltDBは、IngresやPostgreSQLに携わったMichael Stonebrakerにより設計され、VoltDB社により開発が行われているインメモリデータベースです。VoltDBには次の特長があります。

- インメモリデータベースなので高速シャーディングにより、スケーラブル
- 管理オーバヘッドが低いので高速
- データの操作にSQLを利用可能
- 他のNoSQLと異なり、ACID特性を有している

VoltDB社の検証結果によると、通常のRDBと比較して約45倍の性能を示しています。また、スケーラビリティに関してはサーバを1台から12台まで増やすことで性能が10.6倍にまで向上するなど、かなり線形に性能が伸びることが示されています（VoltDB社のホワイトペーパーより）。次のように管理オーバヘッドを大幅に削減することで、より多くの処理時間を実際のデータ処理に割り当てることができるため高速に処理することが可能になっています。

- インメモリにすることでバッファ管理が不要
- 各サーバ（シャード）上のシングルスレッドで動くストアドプロシージャによりデータ操作することで排他制御を簡素化
- 各シャードを複数のサーバに複製することでトランザクションログへの書き込みが不要

シャーディングは他の多くのNoSQL製品と同様にハッシュテーブルにより行われます。例えばメールアドレス、名前、パスワード、会員ランク、現在のポイントを持つ会員マスタテー

ブルを3台のVoltDBサーバに分散させると**図1**のようになります。ここではシャードを二重化(A、B、Cの複製をA'、B'、C')しているので、たとえ1台のサーバ(例：ノード3のシャードC、B')がダウンしても、他の2台のサーバにはすべてのデータ(シャードA、B、C')があるためにシステムは稼動し続けられます。

特定のシャード内で完結する処理は非常に高速に完了します。それは、SQLのWHERE句で「シャードキー＝値」を指定するような場合です。このテーブルの例で言うと、

```
select ポイント from 会員マスタ where
メールアドレス = 'suzuki@example.com';
```

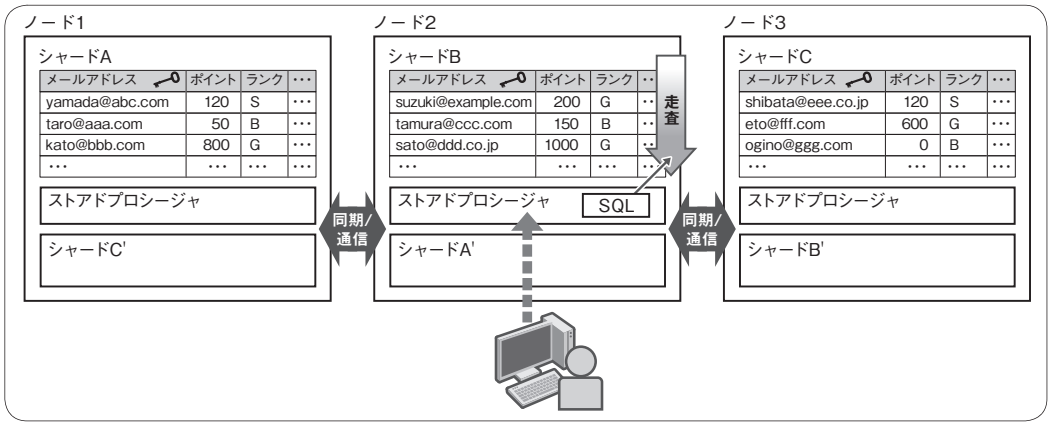
や

```
update 会員マスタ set ポイント =
ポイント + 100 where メールアドレス =
'suzuki@example.com';
```

といった処理は、レコードの走査がシャードB内だけで行われるので高速に処理されます。

VoltDBはクライアントから呼び出されるストアドプロシージャでデータの操作を行いますが、**リスト1**のようなプロシージャを書くこと

▼図1　VoltDBのパーティションテーブル(シャーディング)処理



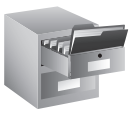
▼リスト1　ストアドプロシージャによるトランザクション処理

```
package jp.co.pipe.procedures;
import org.voltdb.*;

public class UpdatePoints extends VoltProcedure {

  public final SQLStmt selectsql = new SQLStmt("SELECT points FROM members WHERE email = ?;");
  public final SQLStmt updatesql = new SQLStmt("UPDATE members SET points = points - ? WHERE
email = ?;");

  public VoltTable[] run( String email, long points ) throws VoltAbortException {

    // メールアドレスで指定された該当会員のポイント数確認
    voltQueueSQL(selectsql, email);
    VoltTable queryresult = voltExecuteSQL()[0];

    // 所持ポイント数が、消費するポイント数よりも少なければ処理中断
    if (queryresult.getRowCount() == 0 || queryresult.fetchRow(0).getLong(0) < points) throw
new VoltAbortException();

    // ポイントの消費
    voltQueueSQL(updatesql, points, email);
    return voltExecuteSQL();
  }
}
```

で、トランザクション処理を行うことができます。例では、指定のポイント以上を保持しているかどうかを確認(select)してから、指定ポイントを減算(update)しています。このストアドプロシージャは、シャードに対してシングルスレッドで動作するために、処理中のレコードが他のクライアントから操作されることがありません。

一方、たとえば会員ランクがGold(G)の会員のポイントを集計(sum)することを考えると、全てのシャードで会員ランクがGのものをピックアップするためのレコード走査と、各シャードの結果セットのサーバ間での伝送が発生します。VoltDBでは(少なくとも選抜総選挙で使ったバージョン1.3では)、結果セットの転送量が10MBを超えるとエラー(「Output from SQL stmt overflowed output/network buffer of 10mb.」)が発生してしまったので、その場合はデータの範囲を制限する必要があります。複数シャードにまたがるデータの大量取得や集計関数を使うと、このエラーが発生するようです。

これらの特徴を理解することは、VoltDBを適用するのに得意な分野、苦手な分野を判断するのに役立ちます。



選抜総選挙システムでのVoltDBの使い方

選抜総選挙では投票結果を記録するためにVoltDBを使用しました。シリアル番号、投票先候補者ID、投票日時だけの非常に単純なテーブルが1つだけです。シリアル番号が主キーであり、シャードキーです。このテーブルに対して次の処理が行われます。

①投票(レコードの挿入。主キーによる同一シリアル番号による重複投票のチェック)
②投票後の集計(候補者IDでグループ化した投票数合計)

①の投票はVoltDBの特性を活かして、レコードが挿入されるべきシャード上で重複チェックをし、問題が無ければ挿入するという処理を高速に行うことができます。しかしながら、②の集計では問題がありました。前項で説明したとおり、集計は複数シャードに対する処理であり、結果セットのサーバ間通信が発生します。候補者IDが150程度であったので結果セットは大きくないだろうと思っていたのですが、結果セット転送量に関するエラーが発生してしまいました。そのため、集計用一時テーブルを用意し、エラーが起こらない範囲に投票期間を3つに分けて、それぞれ集計して一時テーブルに書き込み、再度、その一時テーブルから最終結果を集計するといった手間がかかりました。それでもこの作業は最終集計時の一度だけですので、十分にVoltDBの高速処理のメリットを享受できました。

VoltDBの得意な分野と苦手な分野

「SQLが使える」「テーブルのJOINができる」「トランザクションが使える」「シャーディング」等の観点で、他のKVSよりも強みを持つVoltDBですが、その性能を活かせるのは、やはりシングルパーティション(シャード)での処理です。シャードキーを指定したデータの操作となります。

SQLが使えるからと言ってRDBが使えるさまざまなSQLをVoltDBで処理しても、非効率あるいは低速で処理される場合があります。集約関数を使ったり、WHERE句でパーティションキー以外のものを指定したりする場合には、VoltDBが最適かどうかを検討する必要があるでしょう。

PostgreSQLへの移行

上記のように弊社が最初に担当した投票ではVoltDBを使いましたが、その後の投票では「性能」「運用性」「開発保守性」の観点から、RDBであるPostgreSQLにデータベースを変

更しました。

当初は過大な性能要求がされていたためVoltDBを選択しましたが、実際には想定の100分の1程度の処理頻度でしたので、PostgreSQLでも十分に対応可能だと考えました。他の業務の関連もあり、実際にVoltDB 1.3とPostgreSQL 9.0上で44万件の顧客テーブルを使い、①重複しているメールアドレスがあればフラグを立てる、②1000行のメールアドレスを含むファイルを1行ずつ読み込んでメールアドレスが一致するレコードにフラグを立てる、という2つの性能検証を行いました。その結果は**表1**のとおりです。②はVoltDBのデータ操作は速いのでしょうが、その操作(Javaによるストアドプロシージャ)の呼び出しコストが高いのだと考えます。

運用性に関して、当社は「情報資産の銀行」の事業コンセプトのもと、いくつかの製品やサービスを提供していますが、データベース基盤としてPostgreSQLを幅広く使っています。そのため、PostgreSQLの運用・監視や障害対応には慣れていますが、VoltDBの経験は不足していました。そのためにVoltDBが性能的には良い選択肢でも、システム運用メンバーの不安や負荷が増えてしまいます。

また開発保守性に関しても、VoltDBはデータ操作にSQLを使えるものの、シングルやマルチパーティション(シャード)を意識してストアドプロシージャを開発したり、JDBCではなく独自のクライアントライブラリを使ってアクセスしたりと、複数の開発者で開発する場合には教育や保守コストも増えてしまいます。

今回は会社組織の仕事のため上記の理由からPostgreSQLに移行する判断をしましたが、自分一人で仕事をするのであれば技術的な興味からVoltDBを使い続けたかもしれません。

▼表1　VoltDBとPostgreSQLの処理速度テスト

処理	VoltDB	PostgreSQL
①重複チェック	125秒	188秒
②一致アドレス	11.2秒	8.2秒

まとめ「適材適所でDBMSを選択する」

データの持ち方、データベースにはさまざまな種類や製品があり、一貫性を持ちトランザクション処理が得意であったり、大量レコードの集計処理が得意であったり、それぞれに得意分野があります。MySQL、PostgreSQL等のRDBは汎用性が高く、そのために幅広い用途で使われています。数百万～数億レコードといったデータの集計処理であれば列指向データベースの方が適していますし、単純にキーに対する属性値の取得であればKVSが適しています。VoltDBはリアルタイムで訪問者の確認や更新等、パーティションキーで指定可能なデータに対する操作を、SQLやトランザクション処理する場合に適していると思います。

また、必ずしもDBMSだけが選択肢ではなく、YAML、XML、JSONといったファイルへの読み書きおよび共有や、プログラム中の変数や定数としてメモリに保持して各アプリケーションサーバ間で通信するなど、さまざまなデータ保持・管理の手法の中から用途に最適なものを選択します。上記選抜総選挙のシステムでも、VoltDBのレプリケーションだけではなく、ファイルへの書き出しも行うことで冗長性、信頼性を高めています。

それぞれの得意分野と、なぜそれが得意なのかの技術的な理由を理解し、開発するシステムに最適な製品を選択することが大切だと考えます。また、処理速度だけではなく、運用性、信頼性、可用性、保守性、データ一貫性といった観点も考慮に入れる必要があります。SD

第5章 RDBと比べてわかるMongoDBを利用する際の注意点

Writer 桑野 章弘(くわの あきひろ)(株)サイバーエージェント
Twitter@kuwa_tw／URL http://d.hatena.ne.jp/akuwano/

本章ではNoSQLの1つであるMongoDBを利用する際のポイントを紹介します。サイバーエージェントでは、同社のソーシャルゲームサービスでMongoDBを利用しており、また、ほかのサービスではRDBの運用経験も豊富です。RDBの特徴や使い方と比べることで、MongoDB特有の注意点が見えてきます。

はじめに

MongoDBについて、みなさんはどのような印象をお持ちでしょうか。SNSなどでは、「開発しやすいデータストアですね！」という意見と同時に、「大規模に運用すると痛い目をみるデータストア」という意見が散見されます。みなさんも同様の印象をお持ちではないでしょうか。筆者はMongoDBを使ってきた感想として、これらの意見は一面では合っていると思うし、ある面では正しくないな、もったいないな、と考えています。

MongoDBに限らず、NoSQLと言われるプロダクトは、得意／不得意がハッキリ分かれる傾向や、「これをやると失敗する。でもこれに使うならうまく使える」といった傾向が強いように思います。とくに、RDBで実現できない／しづらい部分を実現するためにNoSQLが生まれたこともあり、RDBとは性質が異なる傾向が強いです。

そこでこの章では、MongoDBを運用してきた中で得た「うまく使うためにはどうしたらいいのか？」というノウハウについて話していきましょう。

MongoDBの特徴

それでは、最初に簡単にMongoDB[注1]の特徴をまとめていきましょう。

MongoDBは、MongoDB社[注2]が開発しているオープンソースのデータストアです。MongoDBという名前の由来は、「ばかでかい」という意味の英単語“humongous”の“mongo”という部分からきています。おもにスキーマレスを始めとした「開発のしやすさ」や、シャーディングを始めとした「スケールアウト」を最初から考慮に入れて開発されています。

そんなMongoDBで実現できることとは何でしょうか。RDBと比較しつつ紹介していきます。

柔軟なデータ構造

まず、MongoDBは柔軟なデータ構造を実現できるドキュメント指向データストアである、という点が挙げられます。通常、RDBはテーブルとテーブル間のリレーションでデータ構造を表現します。MongoDBはデータ構造の変更の自由度が高く、JSONに似たBSON(Binary JSON)形式でデータを扱うため、リスト構造やネストの深いデータ構造も表現可能です。

もう1点、スキーマレスという特徴があります。RDBの場合、テーブル構造はそのテーブル内で同じでなければならず、変更する際にもすべての行を変更しなければなりません。しかし、MongoDBのコレクション(RDBでいうテーブル)内の構造は各オブジェクトで同一である必要はありません。たとえば、**リスト1**と**リスト2**のデータ構造が同一コレクション内に存在しても問題ありません。

注1） http://www.mongodb.org/

注2） http://www.mongodb.com/

サーバ冗長化

RDBサーバで苦労することの1つに、サーバの冗長化があります。最近は、MySQL-MHA[注3]などの普及により、冗長化の敷居は下がってきていますが、構築にはネットワークやDBサーバに関する知識が必要です。MongoDBではレプリカセットというしくみが標準で用意されています。

レプリカセットとは、複数のmongodプロセス[注4]でデータの同期とサーバの冗長化を行う機能のことです。RDBにおけるレプリケーションと冗長化を兼ね備えた機能と言えます。サーバの種別はプライマリとセカンダリとアービターに分かれており、更新はプライマリサーバに向けて、参照はセカンダリサーバに向けることもできます。プライマリに障害が起きた際、新たなプライマリの選出はレプリカセットの各サーバによる投票によって行います(**図1**)。レプリカセットには最低3つの投票権を持つプロセスが必要となります。

レプリカセットはスプリットブレイン(ネットワーク分断)対策で、レプリカセットメンバの過半数がいるクラスタ以外はプライマリにならないようになっています。**図1**では3台のレプリカセットのうち1台が壊れていますが、これでもレプリカセットは問題なく処理継続できます。ですが、もう1台壊れて稼働台数が1台になると1/3になり、レプリカセット内で過半数を取れなくなり、処理継続は不可能になります。

そのためサーバ台数は増える傾向にありますが、アービターサーバという実データを持たない投票権のみ持つ軽量プロセスを使うこともできます。

データのスケールアウト

もう1つ、RDBサーバで苦労することとして、データのスケーリングがあります。たとえば、あるサービスでユーザ数が増えてきたのでデータを分割しないといけないというケースがあります。この場合には、データベースを分けて、ハッシュ管理用のデータを作って、ユーザIDのレンジごとに分けるなどして、各RDBのテーブルを分割して振り分けを行います。これらは下手に運用に入ってしまうと、ずっと手作業で行わないといけないなど、技術的負債になり得るものです。

MongoDBの場合、スケールアウトに対する答えとしてオートシャーディングが用意されています。これはデータの受け持ち範囲を各サーバ(もしくはレプリカセット)単位に分けることでスケールアウトするしくみです。アプリケーション側はmongosプロセスに接続するだけなので、バックエンドのサーバをいくら増やしても変わらずにデータにアクセスできます(**図2**)。

分散データ解析

昨今、ビックデータの話をよく聞くようになり

注3) 正式名称はMaster High Availability Manager and tools for MySQL。MySQLのマスターの冗長化を行うためのソフトウェア。

注4) MongoDBの主となるサーバプロセス。実際のデータ、インデックスを保持している。

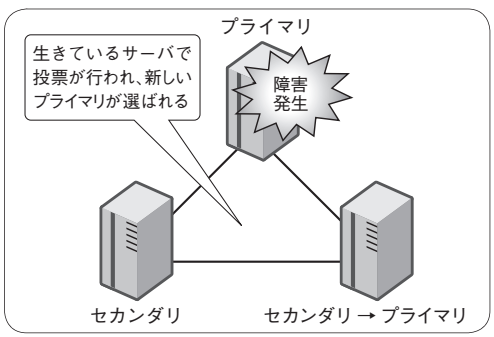
▼リスト1　データ構造の例

```
{
  userId : 'kuwa_tw',
  sex : male,
  favoritefood : [ 'curry', 'mabodofu' ]
}
```

▼リスト2　データ構造の例2

```
{
  userId : 'kakerukaeru',
  favoritefood : [ 'curry' ]
  job : {
    name : “ServerSideEngineer”,
    level : 10
  }
}
```

▼図1　レプリカセットにおける障害発生時の動き



ました。Hadoopなどのソフトウェアを使い、MapReduceでデータ解析を行っている方も多いのではないでしょうか。データウェアハウス的にRDB上で解析している場合もあると思いますが、RDB単体ではMapReduceを行うことはできません。ここまでお話すれば、「またか……」と思うかもしれませんが、MongoDBではAggregation Frameworkを使用することで、MapReduceを非常に簡単に使用できます。データ処理を行うデータストア上でMapReduceが行えるというのは、データ取得からデータ解析までをノンストップで行えるため非常に有用です。

RDBとMongoDB どちらを使えばいい?

前述したとおり、MongoDBではレプリカセットで冗長化を行え、シャーディングでスケールアウトを行え、AggregationFrameworkを使えばMapReduceまで行える万能なデータストアであると言えます(それ以外にも全文検索なども行えるのです！　まだ日本語対応していませんが……)。こんな話をすると、「じゃあ、MongoDBで全部やったらいいの？」と思われるかもしれませんが、世の中そんなにいい話はありません。何も考慮せずにただ使うと痛い目を見ることでしょう。そこでRDBとMongoDBをどのように選択していくのか、このあたりを話していきます。

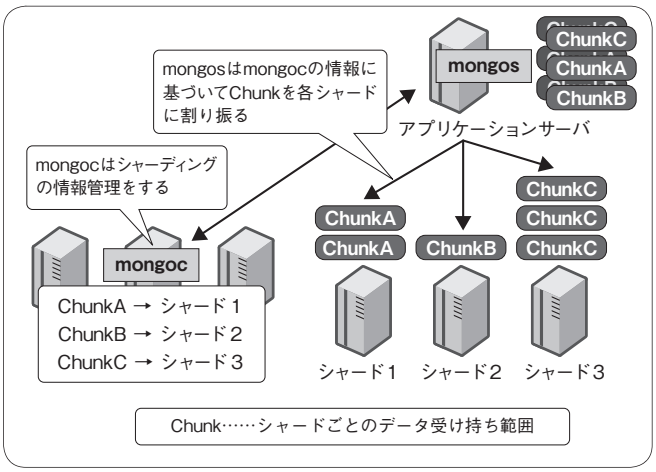
MongoDBとRDBの比較

MongoDBとRDBを比較していきましょう。**表1**に簡単にまとめました。

MongoDBに限りませんが、NoSQLはクセの強いデータストアです。MongoDBの場合、書き込みはデータベースレベルライトロックのため、大量の書き込み処理には向いていません。ですが、読み込みはデータオブジェクトがキャッシュされている限りは高速にアクセスできるため、読み込み処理が多いソフトウェアには有用です。

次に、MongoDBはアクセスしたデータオブジェクトを、MMAP(MemoryMappedFile)を利用してキャッシュします。そのためメモリが潤沢に存在し、OSによるガベージコレクションがうまく動いている間は、非常に高速にデータへのアクセスを行います(**図3**)。ですが、単位時間のデータアクセス量がメモリ量を超え続けると、ディスクとメモリ間のアクセスが大量に発生することになります(**図4**)。それによりMongoDBのスローダウンが発生してしまいます。これを避けるために

▼図2　シャーディングの概要



▼表1　MongoDBとRDBの機能比較

	MongoDB	RDB
アクセスパターン	RDB以上に読み込み処理向き。データベースレベルのロックのため大量の書き込み処理には不向き	読み込み処理向き。おもに行レベルのロックなのでMongoDBよりは書き込み処理向き
データ方式	BSON	テーブル
水平分散	シャーディング	単体ではアプリケーション実装が必須
冗長化	レプリカセット	単体ではない(MHAなどを利用することで可能)
得意なアクセスパターン	ホットデータがあるもの	ホットデータ＆全体解析
重視するリソース	メモリ	メモリ(だが、ストレージ、CPUも必要)

は、「1回にアクセスするデータ量を決める」「そもそもデータを一定以上持たない」という戦略が考えられます。「1回にアクセスするデータ量を決める」場合はアクセスパターン分析とスキーマデザインを適切に行う必要があります。「そもそもデータを一定以上持たない」場合はCapped Collection[注5]やTTLCollection[注6]を使う、もしくは作業用データのみ毎回取得して作業後破棄する、などの処理を行う必要があります。

こういうときはMongoDBを使え!

RDBとMongoDBの違いをふまえたうえで、それぞれにマッチするパターンについてまとめます。まず、MongoDBの得意なユースケースを次のようにまとめました。こちらについて説明していきましょう。

・ゲーム系
・一時データ処理

ゲーム系

ゲーム系が得意である大きな理由として、必要な初期実装が簡単だということが挙げられます。BSONオブジェクトを使用することで、アプリケーションとのデータのやりとりをJSONで行えます。とくに最近、流行しているNode.jsでの開発に関しては、サーバからデータストアまでをJSONでワンストップで扱えるため、非常に簡単に実装できます。手前味噌ではありますが、弊社の提供している「ピグライフ」「ピグワールド」といったサービスもすべてNode.js + MongoDBという構成で行っており、高速な開発スピードを維持するのにMongoDBが一役買っています。

もう1点はゲームのデータの中でアクセスされるデータは、一部のアクティブなユーザのものがほとんどである、という点です。そのようなデータ(ホットデータ)に関しては、MongoDBのキャッシュ機構がうまく働くため、パフォーマンスを確保でき、MongoDBの良さを十分引き出せます。

一時データ処理系

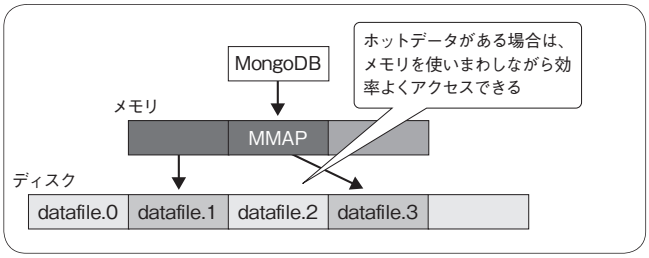
もう1つMongoDBが得意とするユースケースは、一時的なデータ処理系です。「1回にアクセスするデータ量を決める」「そもそもデータを一定以上持たない」という戦略で行きましょうと前述しましたが、まさに一時的なデータ処理系はそれに沿った使い方です。たとえば、次のようなものになります。

・少量かつ定量のデータを取得するための一時DB
・大規模データを処理しないMapReduce処理

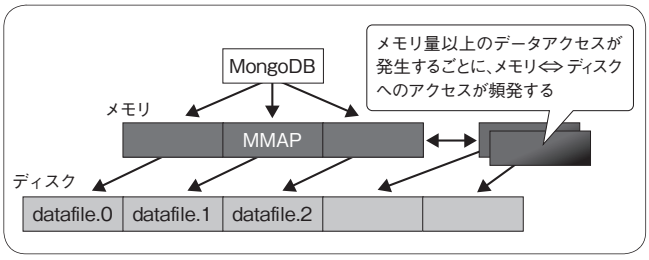
Fluentdなどでログを貯める用途として使う場合や、少量のデータを取得するためのデータベースとして使う場合には、CappedCollectionや、TTLCollectionを使用することでデータ量の肥大化を避けられます。それにより、MongoDBを高速に保ちつつ、リアルタイムに近いログ解析クエリを投げることができます。

注5) 古いデータを削除することで、データサイズを一定に保つことができるコレクション。
注6) 定めた期間よりも古くなったデータを自動で削除するコレクション。

▼図3　MMAPでの効率の良いキャッシュの例



▼図4　MMAPでのキャッシュあふれの例



もう1点、「AggregationFrameworkを使用することで、現在入っているデータに対してMapReduce処理を行える」ということも先ほど述べました。ですが、大量のデータに対してMapReduceを行うのはスローダウンの可能性があります。そのため、大元のデータはほかのデータストア（RDBなど）に貯めておきます。MongoDBはデータ削除、データインポートともに高速なため、毎回必要な分だけMongoDB側へインポートしてMapReduceを行えば、簡易かつ結果を利用しやすいしくみを構築できます。

もし、「統合的なログ解析システムのバックエンド」を構築したい場合には、MongoDBではなく大規模なRDBやHBase、Cassandra、HDFSなど、それにあったものを使用するようにしましょう。

こういうときはRDBを使え!

それでは次に、RDBが得意とするケースについて説明していきます。具体的には次の場合になります。

・SNS／ブログサービスなど
・課金システム

SNS／ブログサービスなど

まずは、SNSやブログサービスなどです。この場合は何が問題になるかというと、やはりメモリの使い方になります。SNSやブログサービスはアクセスされるユーザが多岐に渡ります。ゲームであればゲームをやっていないユーザのデータにはアクセスが行くことはありません。しかし、SNSやブログといったサービスは広いユーザアクセスが発生するため、MongoDBであればデータファイルのMMAPキャッシュの入れ替えが多数発生してしまいます。これはMongoDBが苦手とするアクセスタイプです。RDBであればバッファプールやクエリキャッシュといったインテリジェンスなキャッシュ機構が存在するため、ある程度広い範囲のアクセスも担保できます。

そして、MongoDBの場合、セカンダリインデックスの作成に対応してはいますが、現状はインデックスの使用は1クエリで1インデックスという制約があるので複雑なクエリを投げるのが難しいのも、これらのサービスに向いていない点になります。

課金システム

もう1つは課金システムなどの堅牢なシステムへの利用です。MongoDBにはRDBと同等レベルのトランザクションはありません。一部のAtomic処理は対応しているのですが、完全なACIDを保証していないため、完璧なロールバック、コミットを使用するためには自分たちで実装する必要があります。信頼性の担保という点ではRDBにはかないません。そのため堅牢なシステムや、いわゆる「枯れた」システムに対してはRDBを使用するのが無難であるといえます。

MongoDBの運用

それではMongoDBの実際の運用について解説していきましょう。

基本的な考え方

まず、基本的な考え方になります。RDBの運用時の考え方とは異なる部分をまとめてみました。

・トランザクションはない（Atomic操作は一部存在する）
・メモリは潤沢に用意すべし
・単純にクラスタ台数を増やすことでスケールアウト可能
・正規化は必ずしも良いものではない

トランザクションと、メモリの話は先ほども述べたので割愛しましょう。

クラスタ台数を増やすことでスケールアウトが可能ということについて説明します。シャード分割が行われることで、1シャードが受け持つデータ量はシャード数分に分割されます。結果、1シャード当たりのデータ量は減ります。加えてシャードが増えることで各シャードでメモリが

使用できます。そのため全体のメモリ使用量が増え、キャッシュできるデータ量が増えます。これはメモリを潤沢に用意する、というのと似た話になります。

正規化が必ずしも正しくないというのは、「メモリにおさめておきたいのか」「クエリ回数を減らしたいのか」といった部分に起因する問題です。たとえば、**リスト2**のデータをUserコレクションとJobコレクションに分けた場合の例を**リスト3**に提示します。この場合、Jobコレクションへのアクセス頻度が高い場合は、メモリに収まるデータ量が多くなるため効率が悪くなるわけです。このように1コレクションにデータを詰め込んだほうが効率がよくなる場合も多いので、正規化するかどうかは慎重に考えるようにしましょう。

そのほかに気になる点

学習コスト／必要なスキル

MongoDBは使えるようになるまでの敷居が低いデータストアです。各種設定については、設定値をあまり複雑に持たないというポリシーで開発しているようで、基本的な設定を入れればデフォルト値でもちゃんと動くところはすばらしいです。逆に、ここは設定させてくれという設定値(キャッシュポリシーなど)もありますが。

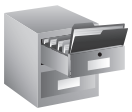
必要なスキルに関しては、クエリがJavaScriptであるため、難しいクエリを投げるようになればなるほどJavaScriptに対する知識が必要になります。ですが、必要になるといっても、1つ1つのクエリはそこまで難しいものではないので、すぐに慣れるはずです。

▼リスト3　リスト2のデータを分けた場合の例

```
Userコレクション
{
  userId : 'kakerukaeru',
  favoritefood : [ 'curry' ]
}
Jobコレクション
{
  userId : 'kakerukaeru',
  job : {
    name : “ServerSideEngineer”,
    level : 10
  }
}
```

費用／サポート

費用としてはオープンソースソフトウェアなので、使用するだけであればもちろんお金はかかりません。MongoDBのユーザコミュニティ(MongoDB-JPなど)も国内に複数存在します。ですが、MongoDB社で提供している有償版のサポートを使用したり、サポート契約を結ぶ際には費用がかかります。開発元からの情報は役に立つことも多いと思います。ただ、有料の情報がなかったとしても開発者のJIRAのページや、メーリングリストの情報などはとても有用です。そして、最近では、NRI社がMongoDBのサポートを始めたというニュースもあります。まだノウハウがなく使用に不安を抱える状態の場合はこういったサポートの使用検討を行うのも良いと思います。



MongoDBとRDBはもっと仲良くできるはず

ここまでいろいろな観点で、RDBとMongoDBの比較をしてきました。そのため、みなさんは「RDBとMongoDBは相容れないもの？」「どっちを使えばいいの？」と思っているのではないでしょうか。これらのプロダクトは、領域が重なっている部分も少なくありません。それによってどちらかを選択しなければならない場合はあります。ですが、前述のとおりNoSQLは得意、不得意が分かれるものです。RDBを使ったほうが良い場合はRDB、MongoDBを使ったほうが良い場合はMongoDB、だけではなく、1サービス内でも動的なパラメータ変更部分はMongoDBを使い、書き込みが多かったり、大事なデータが入っている部分はRDBを使う、というふうにシステム構成としてベストな方法を考察することが必要だと考えています。

この章がNoSQLとそしてRDBとも、仲良くできる一助になりますことを祈りつつ、この章を終わらせていただければと思います。SD

BOOK no.1 Androidのなかみ
Inside Android

Tae Yeon Kim、Hyung Joo Song、Ji Hoon Park、Bak Lee、Ki Young Lim【著】／Androidフレームワーク研究会【訳】
B5変形判、 506ページ／価格＝3,800円＋税／発行＝パーソナルメディア
ISBN＝978-4-89362-288-4

Androidは、Linuxカーネルを中心にランタイム、各種ライブラリ、アプリケーションフレームワークなどから成るフレームワークとして提供されている。その内部を解説する1冊。プロセスの動作や各種フレームワークの役割や利用方法について解説する。これはAndroidデバイスの開発者には必須の情報だ。フレームワークが提供する機能を知ることで、既存機能を利用できるところ、自分が開発すべきところが明確になる。また、アプリ開発者がAndroidに最適な開発方法を身につけるためにも役に立つという。本書を理解するには、システムコール、C、Javaの知識が必要だが、アプリ開発の幅を広げるために、一度挑戦してみてはどうだろう？

BOOK no.2 Redis入門
インメモリKVSによる高速データ管理

Josiah L.Carlson 【著】 ／長尾 高弘 【訳】
B5変形判、 360ページ／価格＝3,400円＋税／発行＝KADOKAWA
ISBN＝978-4-04-891735-3

KVS（Key Value Store） の実装は、本書のRedisをはじめとして山ほどある。roma、kumofs、VoltDBなどさまざまだ。okuyamaは本誌でも読み切り記事があった。KVSはリレーショナルデータベースと異なり、シンプルなしくみだからだろうか、多くの開発者が技術力を競っているように見える。本書はインメモリ型KVSと分類され、高速処理で評判の高いRedisを導入からシステムでの利用までかなり詳しく解説している。スクリプト言語luaとの連携まで一通りカバーしており、最初の1冊としてお勧めだ。

BOOK no.3 詳説Cポインタ

Richard Reese 【著】 ／菊池 彰 【訳】
B5変形判、 248ページ／価格＝2,200円＋税／発行＝オライリー・ジャパン
ISBN=978-4-87311-656-3

C言語の壁と言えば「ポインタ」の理解と昔から言われてきているが、そのためにたくさんの「ポインタ攻略」の書籍が出ている。本書はその中でもさまざまな内容を網羅的に扱った全部入りとも言えるもので、動的メモリ管理、関数、配列、文字列、構造体のそれぞれの章で例文を挙げながら詳しく解説されている。初心者向けのやさしい本が多い中で、本書では最初からmallocに代表されるC言語のヒープ領域でのポインタの使い方が出てくるなど、少し難しめに感じた。また、図でも解説されているが、あまり多くなく単純なものが多い。最初から読み進めていくよりも、わからないところを重点的に学習するのには良い教材ではないだろうか。

BOOK no.4 インフラ／ネットワークエンジニアのための ネットワーク技術&設計入門

みやたひろし 【著】
B5変形判、 416ページ／価格＝2,980円＋税／発行＝SBクリエイティブ
ISBN＝978-4-7973-7351-6

本書のスタンスは「自分自身でネットワークをイチから構築して運用してみないと、インフラ／ネットワークエンジニアとしての基礎力と対応能力を身につけることは難しい」というもの。既存のネットワークを相手に、与えられたマニュアルどおりに運用しているだけだと感じたら、本書は絶好のステップアップの手引きになる。ネットワーク構築の要となる基本設計について、豊富な図とともに1冊まるまる使って丁寧に解説している。基本設計の流れと本書の章構成が対応づけられ、しっかりした設計手法が自然に習得できるだろう。本書を教科書にネットワークを構築してみてほしい。経験に裏付けられたものこそが、自身の武器となるはずだ。

第2特集

ネットワークエンジニアのための プロキシサーバの教科書

基本機能からリバースプロキシまで 構築+運用マニュアル

本特集ではプロキシの教科書と題して、プロキシの基礎、リバースプロキシの実際、クラウド上でのプロキシサーバの3つに分けて解説します。

CHAPTER1では、プロキシの歴史を振り返り、フォワードプロキシをメインに動作と利用方法を解説しています。

CHAPTER2では、リバースプロキシに焦点を当て、サーバの代わりにクライアントとのやりとりをする基本的な機能から応用まで、実例を交えて解説します。

CHAPTER3ではプロキシの応用例として、プロキシのテクノロジを利用したクラウドサービスとその利用方法を紹介します。

CHAPTER 1

歴史をひもとけば見えてくる プロキシサーバの役割と変遷

株式会社データホテル
伊勢 幸一(いせ こういち)

プロキシを知るためには、まずその役割が必要になった背景を知る必要があります。インターネットの歴史を振り返り、フォワードプロキシ/リバースプロキシの誕生とその役割を確認しておきましょう。また、フォワードプロキシの動作と利用方法も解説します。

歴史から見るプロキシ

インターネットの歴史を振り返ると、常にデータを処理し情報を発信するサーバと、その情報を受け取ってさまざまに活用するクライアントという情報処理プログラムのコラボレーションに牽引されて進化してきました。たとえば、メールサーバとメーラー、FTPサーバとFTPクライアント、NNTPサーバとNetNewsリーダなどが古典的な例でしょう。そして、これらサーバ-クライアントシステムに大きな変革を与えたコラボレーションがWWW(World Wide Web)を形成するHTTPサーバとHTMLブラウザでした。

HTTPサーバ-クライアントがインターネットを席巻する以前、一時的にFTPサーバコンテンツをナビゲートするGopherとArchieや、テキストの全文検索を提供するWAISなどのサーバ-クライアントシステムも存在しました。しかし、サーバ-クライアントの相関関係がGopherやArchieのようなツリー型ではなく、メッシュ型で形成されるHTTPサーバ-クライアントの斬新なアイディアと、HTMLマークアップ方式を採用することによる簡易性と拡張性に圧倒され、WWWが出現するやいなや鳴りを潜めてしまいました。

今では考えられないことですが、当時インターネットを利用できる学術機関や一部の組織ではLANに接続されたコンピュータにはグローバルIPアドレスが与えられ、そのコンピュータはLANとインターネットの境界に設置された単なるIPルータを介して直接インターネット上のサーバや他のコンピュータと通信を行っていました。つまり、それらのコンピュータはインターネット側からでも直接アクセスすることができたのです。この頃、インターネットの利用者は軍、行政組織、企業の研究所、学術機関などに限られ、不特定多数によるDoS攻撃、不正侵入、情報改ざんなどのインターネットセキュリティがそれほど問題視されていなかったのです。

インターネット接続の増加

日本国内で一般の利用者に対するインターネット接続サービスが開始されたのは1993年ですが、この頃からインターネットを利用する事情に変化が生じてきました。日本をはじめ、世界中に商用接続サービスが展開されていく中、まず一般の民間企業がHP(ホームページ)と呼ばれる自社Webサイトを立ち上げ始めます。また、同時に企業や個人がISPと契約してインターネットへのアクセスサービスを受けることによって、Webサイトへのアクセスであるトラフィックがインターネット上を駆け巡ります。

この時点でインターネット上にあるWebサイトは企業や店舗のプロモーションのための情報

発信だったり、試験的な運用や学術機関が研究情報を共有するための非営利目的サイトがほとんどでした。それでも北米スタンフォード大学の学生が独自に運営していたキャンパス周辺レストランのメニューリストHPや国立がんセンター(現独立行政法人国立がん研究センター)が配信していた気象衛星ひまわりの撮影画像、奈良先端科学技術大学院大学が運営するShikaサーバで公開されていたPLAYBOYとPENTHOUSEのカバーガール写真集などが筆者の印象によく残っています。また、分割前のNTTが運営していたWebサイトでは「日本のWWWサーバ」というハイパーリンクリストがあり、当時日本で稼働していたWebサーバサイト数はブラウザ画面一枚に収まる程度でした。

このような従来のテレビ、ラジオ、新聞、雑誌とはまったく異なり、膨大な情報群からパソコン上のブラウザで利用者の意志とアクションによって利用者が選択する情報にアクセスするというコミュニケーションパラダイムに世界中の人々が興奮したのです。

セキュリティとIPアドレスの枯渇問題

自動的にインターネットにアクセスする利用者も爆発的に増加し、不本意ではありますが中には悪意をもって他組織のコンピュータにアクセスして侵入したり、内部情報にアクセスしたりする事例も発生してきます。そこで、企業や組織はインターネット側から簡単に社内のコンピュータにアクセスできないようなしくみを必要としました。

と同時に、この頃すでにIPv4アドレススペースの枯渇が懸念され始めます。先に述べたように組織内LANに接続されていたほとんどのPCはグローバルIPアドレスを持ち、それぞれが個別にインターネット上のサーバにアクセスしていました。中でも一部のUNIXとTCP/IPという先進的技術をターゲットシステムとした企業では、北米NIC(Network Information Center、現InterNIC)から直接クラスBのアドレスブロック[注1]の割り当てを受けていたりしていました。現在でいう/16プレフィックスのアドレス空間を、数十人しかいないベンチャー企業がLAN内部で利用していることなどあたりまえの状態だったのです。

そこで、1996年にIETFのRFC1918によって、LAN内で利用するために特化したプライベートアドレスが提案され、内部LANとインターネットの境界をIPルータではなく**何らかのゲートウェイによって遮断し、中継する技術**が必要とされました。そのネットワークをIP的に遮断し、かつ中継するゲートウェイがNATやファイアウォール、もしくはHTTPプロキシサーバだったのです。

NATの登場

これら3種類のゲートウェイはそれぞれ内部LANとインターネットを何らかの法則によって遮断し中継することを目的としていますが、方法やしくみが異なります。最初に出現したのは小型のルータ的外観であるNATアプライアンスでした(**図1上**)。これは単にIPヘッダにある宛先と送信元のIPアドレスを書き換え、内部のコンピュータに対してはインターネットではなく内部のサーバにアクセスしているかのように見せかけます。同時にインターネット側のコンピュータにはグローバルIPを持つコンピュータと通信しているかのような働きをし、IPパケットの中身に対して何らかの処理を行うことはありません。これはLAN内のプライベートアドレスを持つコンピュータから、グローバルアドレスを持つインターネット上のサーバとのパケット送受信を中継しているだけであり、アドレスの書き換えを除けば従来のIPルータとなんら変わりはありません。後にIPアドレスだけではなく、TCPやUDPヘッダのポート番号も変換するNAPT(一般的にはIPマスカレード)も登場してきますが、基本的にデータの中身にかかわらずアドレスやポート番号を機械的に変換する意

注1) 65,534台のサーバが接続可能。

味では同じです(**図1下**)。

ファイアウォールとプロキシの登場

企業がインターネット接続する際に懸念された、内部情報の漏洩や内部コンピュータへの侵入を発見して遮断する目的で導入されたセキュリティのしくみがファイアウォールです。これはNATのように機械的なアドレス変換を行うだけではなく、IPヘッダやその中にあるTCPポート番号などをチェックし、転送が許可されていないパケットを破棄して管理者に通知するといった処理を行います。

IPルータやNATのように機械的な自動処理ではなく、パケットの中身を調べる処理や破棄、転送、ロギングといったアクションをすべてのパケットに対して行うため、ゲートウェイのCPUに対する負荷も高く、転送性能が著しく低下します。LAN内のユーザ数が多い場合、Webサーバからのレスポンスを常に数秒から数十秒待たなければならなくなってしまい、利用者にとってはストレスの高い環境にならざるを得ません。

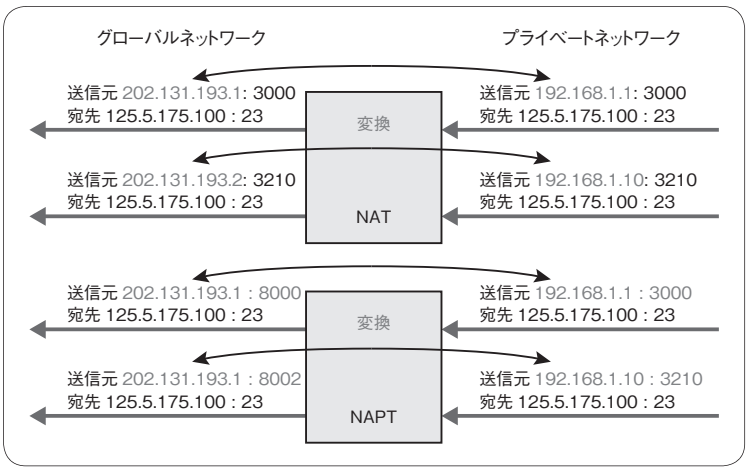
そこでファイアウォールの内側にプロキシサーバを設置し、パケットの中継だけではなく、データのキャッシュを行うことによってインターネットとの通信量を削減する手法が取られました(**図2**)。

プロキシの利用目的

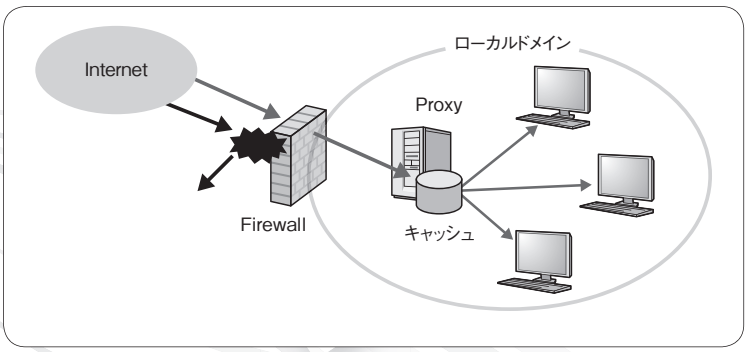
通常プロキシサーバは、NATやファイアウォールのようにコンピュータ間の通信を中継するだけではなく、通信しているデータを一時的にキャッシュします。そして他のクライアントからであっても同じURIへのアクセスであれば、インターネット側にアクセスリクエストを転送せず、ローカルストレージにあるキャッシュを返します。このしくみでクライアントへのレスポンス遅延を軽減させると同時に、通信回線の有効利用を図る目的で導入されました。

というのは、1993年当時のインターネット黎明期において、インターネットへの接続料金は非常に高価でした。筆者の記憶では、IIJから提供されていた64kbpsデジタル専用線での接続料金は月額50万円以上であり、128kbpsでは80万円以上の料金がかかっていました。現在、インターネット

▼図1 NATとNAPT



▼図2 ファイアウォールとプロキシサーバ



の接続料金はNTT東西の光フレッツ100MbpsでISP料金とを合わせても月額3,000円程度です。つまり、この20年でインターネット接続料金はなんと26万分の1にも下がっているのです(64kbpsと100Mbpsで比較)。

それほどインターネット接続は非常に高価であり、かつ64kbpsという非常に狭い帯域を何十人もの利用者で共有する必要がありました。当時のWebコンテンツはテキストベースのページがほとんどであったとはいえ、すべての利用者がブラウザ上のハイパーリンクをクリックするたびに64kbpsの細い回線の中にIPパケットを往来させることなど現実的ではありません。また、Webサイトへのアクセスにもパレートの法則[注2]が当てはまり、インターネットとの通信の80%はわずか20%のサイトへのアクセスであると推定され、この20%のWebサイトのデータを内部でキャッシュしておけば回線の利用効率を劇的に向上させることも可能です。

当時はBlogやSNSなどのサービスもなく、ほとんどのWebサイトは運営者が手動で更新していたためそれほど頻繁に内容が変わるわけでもなく、通常の利用形態であれば常にWebサイトからデータを読み込む必要もあまりありませんでした。最新の情報を取得することより、ストレスのないレスポンスのほうが優先されたため、プロキシサーバによるキャッシュはインターネットの利用効率に絶大な効果をもたらしました。

もちろんプロキシサーバ自身をファイアウォールとして運用することも可能ですが、それにはサーバOSやHTTPサーバに潜在するセキュリティホールをよく熟知し、ネットワークやプログラムに対する知識や経験が豊富な管理者が常にパッチを適用したりログから不正アクセスを摘出したりする必要があり、通常の組織でそこまで人的リソースを割くことは難しいでしょう。したがって、ファイアウォールは商用のパッケージを利用し、プロキシサーバはもっぱらキャッシュサーバとして運用されていました。稀にこれらインターネットアクセスに関する技術的知識を持つ組織では有償パッケージなどは使用せず、オープンソースを使ったファイアウォールやプロキシサーバ、SOCKSサーバなどを利用してインターネットアクセスを可能にしていた団体もあったことは事実です。

注2) 全体の数値の大部分は、全体を構成するうちの一部の要素が生み出しているという説のこと。

フォワードとリバース

歴史を見てきたとおり、インターネット黎明期におけるプロキシサーバとは基本的にフォワード型のプロキシサーバであり、プロキシといえばシステムの利用者がインターネット上のコンテンツを閲覧するために介在するサーバのことを示していました。したがって現在のように**フォワードプロキシ**という単語も存在していませんでした。

その後、さらにインターネットの利活用が拡大し、かつてWebブラウザ1ページ分しかなかった国内のWebサーバも、さまざまな企業や組織、そして個人までもがWebサーバを運用するようになっていきます。また単なるプロモーション目的ではなく、利用者の求める商品やサービスをWeb上で提供するE-コマース、ニュースや記事の配信、それらインターネット上のサービスやコンテンツとその利用者をマッチングするディレクトリサービス、検索サービスなどが出現してきます。

すると有用な情報を配信する人気の高いWebサーバにインターネット上の利用者が集中するようになり、今度は配信側が1台のサーバではアクセスを捌き切れなくなってきました。そこで、DNSラウンドロビンなどを活用し、単一のURLに対して複数のサーバにアクセスを分散して処理する方法が導入されてきます(**図3**)。しかしDNSによる分散手法には問題がありました。いわゆる「DNS更新遅延」という問題です。マスターレコードを持つDNS権威サーバが提供するFQDNとIPアドレスのアソシエーション

情報はクライアントが権威サーバから直接取得するよりも、インターネット接続サービスを提供するISP側に設置されたDNSキャッシュサーバという仲介者を介して取得されます。そのキャッシュには有効期限が設定されており、オリジナルの権威DNSサーバ側のレコードが更新されても、キャッシュサーバ側が有効期限切れで破棄するまでキャッシュにある古いレコードがクライアントに返されてしまいます。そこで、WWWサービス側のリニューアルやサービスの追加変更に伴うシステム構成の変更を利用者が使えるようになるまで時間がかかってしまうのです。

配信側システムの変化

そこで取り入れられた技術がロードバランサであり、シスコシステムズ社のLocalDirectorやアルテオンウェブシステムズ社のACE directorといったアプライアンス製品が導入されました。ロードバランスの導入により、1つのグローバルIPアドレスでも複数のHTTPサーバによって大量のアクセスを処理するという情報発信形式が確立され、これが標準的なWebサーバのシステム構成になっていきます。サーバの追加、変更、削除はロードバランサの設定によって実行されるので、DNSラウンドロビンでのレコード更新遅延時間に影響されるという問題はありません。

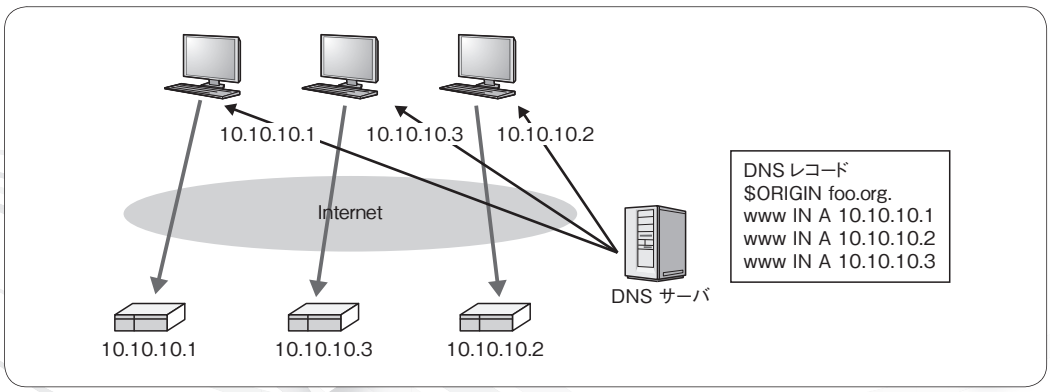
Webサイトが流通サービスやコンテンツ提供などのビジネス用途に活用されはじめると、単に静的なHTMLドキュメントを配信するだけではなく、クライアントからの入力に従った動的ドキュメントを返すインタラクティブサービスが増加します。この際、動的なドキュメントを生成するのがNCSA httpdでサポートされたCGIプログラムでした。現在ではさまざまなプログラム言語によって書かれていますが、当時のほとんどのCGIはC/C++言語、Perlそして Javaによって書かれていました。

ところが、インタラクティブサービスがリッチになるにつれ、画像や静的コンテンツを配信するサーバと同じサーバ上でCGIプログラムも実行すると負荷が大きく、サーバの配信能力に問題が生じます。そこで、ブラウザとのHTTP通信を司るWebサーバと、CGIプログラムを実行するアプリケーションサーバを分離するシステム構成が取られます。さらに、アプリケーションが取り扱うデータを格納するデータベース(DB)サーバや静的データを格納するストレージも分離され、システム構成はフロントエンド、アプリケーション、バックエンドという3段階の構成が取られるようになりました(**図4**)。

リバースプロキシの登場

そして、このフロントエンドであるHTTPサーバはブラウザとの通信だけではなく、サー

▼図3　DNSラウンドロビン分散



ビスや処理によってアプリケーションサーバを選択したり、プリレンダリングされたHTMLドキュメントや画像などの静的データをキャッシュしてバックエンド側データベースやストレージシステムへのアクセス負荷を軽減する役割を担うようになります。つまり、先に述べたインターネット側のコンテンツをローカル側のHTTPサーバでキャッシュし、回線帯域の有効利用とレスポンスの向上を図ったプロキシサーバと、ちょうど逆の働きをするのです。

この逆向きのキャッシングと中継を行うプロキシサーバが**リバースプロキシサーバ**です。それまでプロキシにはフォワードもリバースもありませんでしたが、この配信用プロキシサーバの出現により、従来型のプロキシサーバをフォワードプロキシ、逆向きのプロキシをリバースプロキシと呼ぶようになったのです(**図5**)。

わかりやすく言うと、運用者から見てデータを受信するためのプロキシがフォワードで、データを送信するためのプロキシがリバースです。

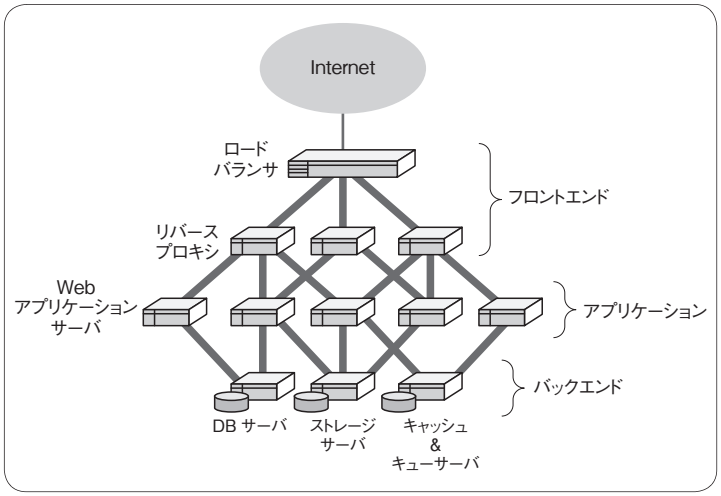
プロキシサーバの動作と利用法

リバースプロキシについてはChapter 2で詳細に解説されるので、ここではフォワードプロキシを例にプロキシサーバの動作のしくみと利用法を**図6**を用いて簡単に解説します。

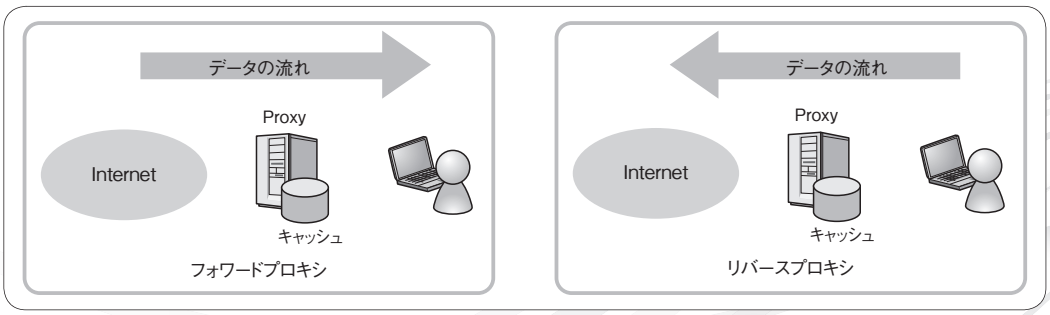
プロキシを使うための設定

通常ブラウザはあるインターネット上のWebサーバが持つhttp://foo.org/index.html というURLにアクセスしようとする場合、そのURLの最初のフィールドにあるFQDN(foo.org)の名前を解決します。そして、そのFQDNに一致するIPアドレス宛のパケットを、インターネットへのデフォルトルートであるファイアウォールへ送信します(**図6**のグレーの矢印)。したがって、任意のURLへアクセスする場合、インターネットとの境界にあるルータへ直接送信されてしまうため、そのままではHTTPリクエストパケットがプロキシサーバに届けられません。

▼図4　Webサーバ構成



▼図5　フォワードプロキシとリバースプロキシ



プロキシサーバに届けるためには、ブラウザ側で明示的にプロキシサーバを介してターゲットにたどり着く設定をする必要があります。

世界初のHTMLブラウザであるMosaicの初期バージョンにはプロキシサーバの設定機能がありませんでしたが、Version 2.4からプロキシ設定がサポートされました。その後にリリースされたNetscape Navigatorなどのブラウザでは、標準でプロキシサーバの設定が実装されるようになりました。ブラウザのプロキシ設定フィールドにプロキシサーバのFQDNかIPアドレスを指定しておくと、どのサーバへのHTTPリクエストであってもまずはプロキシサーバへそのリクエストパケットが届けられるというしくみです。また最近では、プロキシサーバの指定をブラウザ単位ではなくOS側のシステム設定として行うのが一般的になっています。

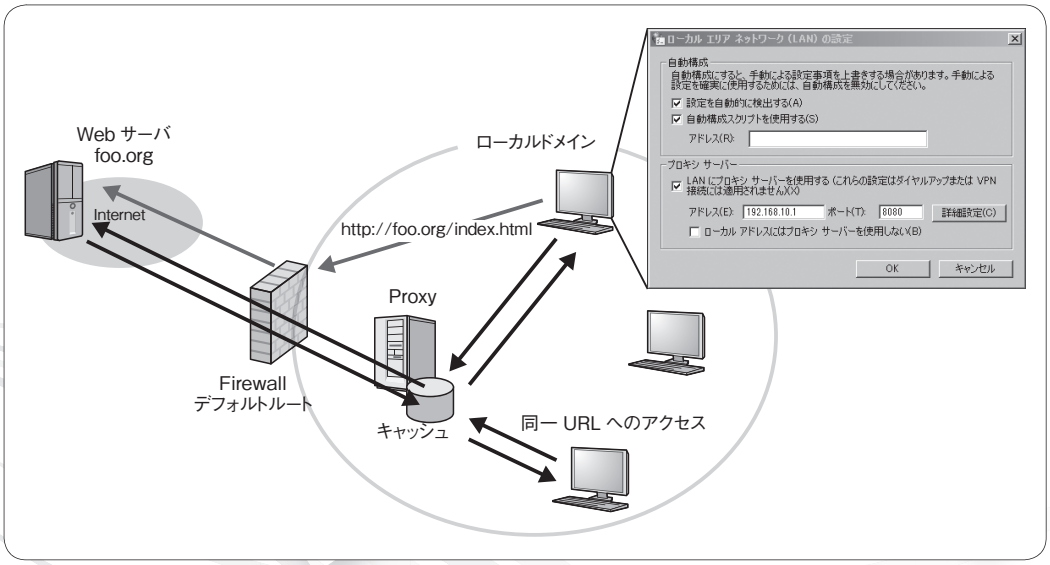
通常HTTPサーバへのリクエストはTCPポート80番をターゲットとして送信されますが、同じサーバで組織向けのWebサーバなどが稼働している場合、そのサーバとプロキシサーバを区別するため80番以外のポート番号でプロキシサーバを起動することがあります。そのため、通常ブラウザやシステムでのプロキシサーバ指定にはFQDN/IPアドレスとポート番号によってプロキシを設定できるようになっています。

ブラウザが送るインターネット上の任意のサーバに対するHTTPアクセスリクエストを受け取ったプロキシサーバは、そのリクエストパケットをルータを介して本来の宛先であるHTTPサーバに転送し、HTTPサーバからのレスポンスを受け取ります(**図6**の黒い矢印)。その受け取ったデータをそのままローカルディスクにキャッシュとして記録すると同時に、そのリクエストを送ってきたクライアントブラウザに戻します。クライアントに届くIPパケットの送信元はプロキシサーバのIPアドレスですが、TCPペイロードの中身はオリジナルのHTTPサーバが返信してきたHTTPデータです。したがって、その中身を解釈しブラウザ上に表示すると、あたかもオリジナルのHTTPサーバからの返信を直接受け取ったかのような表示が得られます。

キャッシュの働き

次に、同じURLに対して別のブラウザからHTTPリクエストがプロキシに届けられた場合、プロキシサーバはまずキャッシュ領域をチェックします。同じURLのファイルがあればそれを

▼図6　プロキシ設定とアクセス



読み出してブラウザに返信し、キャッシュになければ前と同じようにインターネット側のオリジナルサーバへそのリクエストを転送します。

キャッシングというと複雑な処理を連想しますが、HTTPコンテンツは基本的にオブジェクト形式になっているので、もっとも簡単なキャッシング方法はキャッシュ領域にURLと同じファイルを作成し、そのファイルにオリジナルサーバから送られてきたHTMLメッセージをそのまま書き出すだけでも機能します。キャッシュの有無をチェックする場合でも、ブラウザから送られてきたメッセージの中にあるURLと同じ名前のファイルがあるかどうかだけをチェックすれば良いので、性能やセキュリティさえこだわらなければ、多少のプログラミング経験があると誰でも比較的簡単に作成することができます。

キャッシュの作成方式にハッシュを使ったり、オリジナルサーバのHTMLツリー構造をそのまま再現してみたり、キャッシュの生存期間の調整や頻度の高いキャッシュをSSDディスクなどの高速ストレージに配置するなど、いろいろ独自に試してみるのも楽しいことなので、プログラミング講習などの良い課題になるでしょう。

プロキシサーバの変遷

HTTPプロキシサーバはWWWがインターネット上に登場した時点から存在しています。1991年に公開された世界初のHTTPサーバであるCERN(欧州原子核研究機構)版には最初からプロキシ機能が付随していました。1993年に公開された世界で2番目のHTTPサーバはNCSA(米国立スーパーコンピュータ応用研究所)版ですが、不思議なことにこちらにはプロキシ機能は実装されていませんでした。その代わり初めてCGI機能が実装されたのがNCSAであり、NCSAは同時にMosaicを開発中だったのでWWWの閲覧よりも配信に特化したサーバを作りたかったのかもしれません。同じ頃、プロキシ専用サーバとして電総研(現産業技術総合研究所)の佐藤豊氏が開発したDelegatedサーバがあります。これは当時統一されていなかったHTML内の日本語コードを変換し、ブラウザがサポートする日本語コードに合わせて表示するという利用方法もありました。

その後、CERN版もNCSA版もアップデートが滞るようになり、NCSAユーザの有志によってNCSA版に対してプロキシなどさまざまな機能を追加するためのパッチが実装されデプロイされていきます。そしてついにNCSA版が開発中止となった後、これらNCSA版に対するパッチ群を元にApacheサーバがリリースされました。Apacheは「とあるパッチ的サーバ(A patchy Server)」が語源であり、北米先住民部族の名称とは関係ないとされていますが、単なる噂に過ぎないようです。Apacheにはプロキシキャッシュ機能が実装されていたため、従来のプロキシサーバのほとんどはApacheに差し替えられていくようになります。

CERN、NCSA、Apacheは基本的にオープンソースベースの実装でしたが、Apacheがリリースされたころ、Netscape Communications社、Microsoft社、Sun Microsystems社(現Oracle)といった各ベンダーから独自の有償HTTPサーバがリリースされ、当然のことながらプロキシ機能が実装されていました。しかしコードはオープンソースではなく、かつ有償パッケージだったので、各ベンダーのユーザ企業がほかの製品とともに利用する例もありましたが、ほどんどのWebサイトはオープンソースであるApacheが使われていました。2013年末の段階でもいまだApacheがHTTPサーバシェアのトップを守っています。

また、コンテンツを配信するのではなく、コンテンツキャッシュやSSL処理を主たる目的としたHTTPサーバも出現します。オープンソースで公開されているSquidが代表的なHTTPキャッシュサーバの実装でしょう。Squidは水平型分散と垂直型分散の両方のキャッシュ体制をサポートし、おもにフォワードプロキシでは

なく、大規模なWebサイトでのリバースプロキシとして利用されてきました。しかし、Apacheのキャッシュとロードバランシング機能の拡充やLinux OSが提供するLVS負荷分散機能の安定化、非常に高速なリバースプロキシの出現、分散型KVSキャッシュシステムなどの台頭により、その構築運用の複雑さゆえ、次第に姿を消していく傾向にあるようです。

ここ数年で最もよく利用されている実装といえば、リバースプロキシで利用されるNGINX(エンジン・エックス)でしょう。これは処理速度と並列化に優れた実装であり、大量のHTTPリクエストを遅延なく処理して結果を返さなければならないようなWebサイトで多く利用されています。

現在、インターネット接続回線の高速化と低価格化のお陰で、回線帯域の節約とキャッシュによる高速なレスポンスを提供するフォワードプロキシはそれほど需要が多くなくなってきています。個々のブラウザ側でキャッシュデータを持つようになったこともありますが、非常にリアルタイム性の高いインタラクティブコンテンツの増加により、Webサイトのコンテンツをゲートウェイ側でキャッシュすることにあまり意味がなくなってきました。しかし、今後高解像度の画像、音声、動画などインターネット側からダウンロードするには多少ストレスがあるコンテンツを主軸とした何らかのサービスが展開されるなら、従来とは別の目的で再びフォワードプロキシが活用される場面が来るかもしれません。筆者はもう25年以上、このインターネットと付き合ってきましたが、いまだ、明日何が起こるか想像もつかない世界です。SD

CHAPTER 2

基本から応用まで リバースプロキシの用法と実例

LINE株式会社　佐野 裕(さの ゆたか)
Twitter @sanonosa

フォワードプロキシではクライアントの代わりにサーバとのやりとりをプロキシサーバが代行するのに対し、リバースプロキシではサーバの代わりにクライアントとのやりとりをプロキシサーバが代行します。
リバースプロキシはWebサーバやDBサーバなどと比べて存在は地味ですが、ちょっと大きいWebサイトを運用するうえでは必須も言えるほど重要な存在です。

はじめに

リバースプロキシと一口に言ってもさまざまな用途で使われています。リバースプロキシという名称を聞いたことがなくても、コンテンツキャッシュ、負荷分散(ロードバランス)、もしくはSSL(Secure Sockets Layer)アクセラレーションといった用語は聞いたことがあるかと思います。これらもリバースプロキシの考え方を応用したしくみとなります。そのほか、コンテンツ圧縮やシングルサインオンといった用途でリバースプロキシが使われることがあります。

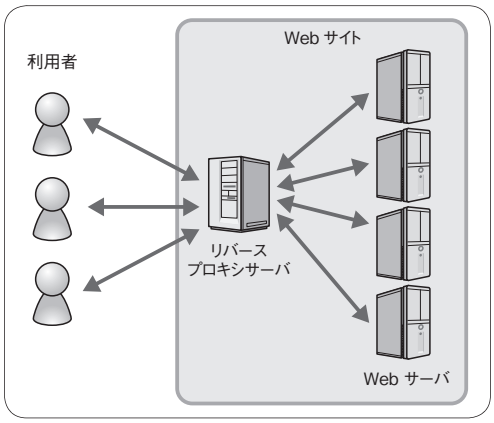
前置きが長くなりましたが、本章ではさまざまなリバースプロキシで使われている用法を紹介していきます。なお、つい数年前までリバースプロキシの用途でSquidが用いられることが多かったですが、最近は有力なオープンソースがさまざま登場してきましたので、今回はその中からVarnish CacheとNGINXでの実例も紹介します[注1]。

注1) 今回紹介する実例はCentOS 5.8での例となります。OSやディストリビューションなどによって多少設定方法が変わることがありますのであらかじめご了承ください。

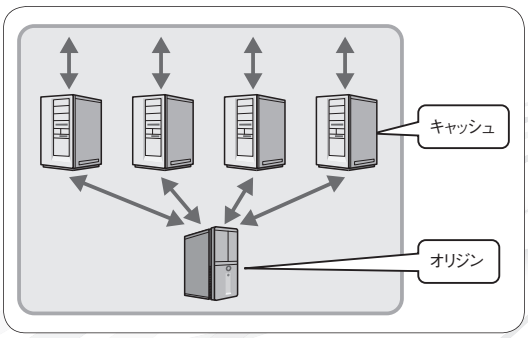
コンテンツキャッシュ

リバースプロキシの代表的な用途としてコンテンツキャッシュが挙げられます。コンテンツ

▼図1　AWSのElastic Load Balancingの動作



▼図2　コンテンツキャッシュ



キャッシュはコンテンツ配信元であるオリジンWebサーバから出力されたコンテンツをキャッシュしておき、クライアントからのリクエストに対して即座に応答することでレスポンスタイムを向上させられます(**図2**)。

ここでは高速なリバースプロキシとして最近広く使われているVarnish Cacheを試してみたいと思います。公式ページに行くと、「Varnish Cache is really, really fast. It typically speeds up delivery with a factor of 300 - 1000x, depending on your architecture.」と書いてあります。よほど高速動作に自信があるのでしょうね。

Varnish Cacheではvarnishとdefault.confに設定を記載します。

まず、Varnish Cacheをインストールした直後は待ち受けTCPポートが6081番となっていますので、これを80番に変えます。

```
/etc/sysconfig/varnish

VARNISH_LISTEN_PORT=6081
        ↓
VARNISH_LISTEN_PORT=80
```

次にバックエンド側に配置されるオリジンサーバのホスト情報と接続ポート番号を記載し、かつ「sub vcl_recy」のコメントアウト(#)をすべて削除します。**リスト1**の例ではオリジンサーバのIPアドレスは192.168.0.1、TCPポートは

▼リスト1　/etc/varnish/default.conf

```
backend default {
  .host = "192.168.0.1";
  .port = "8080";
}

# Below is a commented-out copy of the default VCL logic.  If you
# redefine any of these subroutines, the built-in logic will be
# appended to your code.
sub vcl_recv {
    if (req.restarts == 0) {
        if (req.http.x-forwarded-for) {
            set req.http.X-Forwarded-For =
                req.http.X-Forwarded-For + ", " + client.ip;
        } else {
            set req.http.X-Forwarded-For = client.ip;
        }
    }
    if (req.request != "GET" &&
       req.request != "HEAD" &&
       req.request != "PUT" &&
       req.request != "POST" &&
       req.request != "TRACE" &&
       req.request != "OPTIONS" &&
       req.request != "DELETE") {
         /* Non-RFC2616 or CONNECT which is weird. */
         return (pipe);
    }
    if (req.request != "GET" && req.request != "HEAD") {
        /* We only deal with GET and HEAD by default */
        return (pass);
    }
    if (req.http.Authorization || req.http.Cookie) {
        /* Not cacheable by default */
        return (pass);
    }
    return (lookup);
}
以下略
```

8080番となっています。

以上の設定を行ったらVarnish Cacheを起動します。

```
# service varnish start
```

コンテンツキャッシュが無事作動しているか確認する方法にはいろいろありますが、次の例では、クライアントからのアクセスに対してキャッシュがヒットしたかどうかを、hitかmissというわかりやすい表記で確認ができます。

```
# varnishncsa -F '%U%q %{Varnish:handling}x'

/index.html hit
/common.js hit
/logo.jpg  miss
/getcode.php miss
```

また、Varnish Cacheではさまざまな方法でVarnish Cacheの稼動状況が確認できます。**図3**の例はアクセスログの確認です。

図4の例はVarnish Cacheの統計情報の確認です。

負荷分散（ロードバランス）

一般的にWebサーバ群は安価なサーバを複数並べるスケールアウト構成を取ります。スケールアウト構成ではクライアントからのリクエストを各Webサーバに振り分ける役割が必要となります。またWebサーバを常時死活監視（ヘルスチェック）して何らかの原因で停止している場合は振り分け先から除外するしくみも併せて必要となります。この負荷分散用途としてリバー

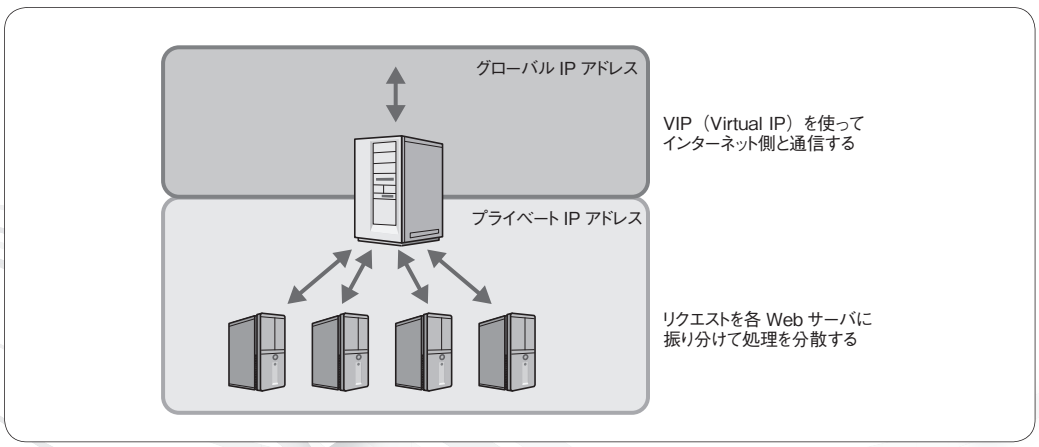
▼図3　アクセスログの確認

```
#  varnishncsa

192.168.1.11 - - [04/Jan/2014:21:03:54 -0800] "GET http://10.0.0.10/ HTTP/1.1" 304 0 "-" ➲
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) ➲
Chrome/31.0.1650.63 Safari/537.36"

#varnishtop
list length 49

     3.15 CLI            Rd ping
     0.93 CLI            Wr 200 19 PONG 1388898292 1.0
     0.81 CLI            Wr 200 19 PONG 1388898289 1.0
     0.80 VCL_return     deliver
     0.74 SessionClose   timeout
     0.74 StatSess       192.168.1.11 50335 0 1 1 0 0 0 315 0
     0.70 CLI            Wr 200 19 PONG 1388898286 1.0
     0.59 CLI            Wr 200 19 PONG 1388898283 1.0
     0.52 SessionOpen    192.168.1.11 50335 :80
     0.52 ReqStart       192.168.1.11 50335 1860921082
     0.52 RxRequest      GET
     0.52 RxURL          /
     0.52 RxProtocol     HTTP/1.1
     0.52 RxHeader       Host: 192.168.1.3
     0.52 RxHeader       Connection: keep-alive
     0.52 RxHeader       Cache-Control: max-age=0
     0.52 RxHeader       Accept: text/html,application/xhtml+xml,application/xml;q=0.9,➲
image/webp,*/*;q=0.
     0.52 RxHeader       User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 ➲
(KHTML, like G
     0.52 RxHeader       Accept-Encoding: gzip,deflate,sdch
     0.52 RxHeader       Accept-Language: ja,en-US;q=0.8,en;q=0.6
     0.52 RxHeader       If-None-Match: "e-4ef27896eda28"
     0.52 RxHeader       If-Modified-Since: Sat, 04 Jan 2014 16:29:55 GMT
     0.52 VCL_call       recv
     0.52 VCL_return     lookup
     0.52 VCL_call       hash
```

▼図4　Varnish Cacheの統計情報

```
% varnishstat

           8         0.00         0.06 client_conn - Client connections accepted
         808         0.00         6.52 client_req - Client requests received
         807         0.00         6.51 cache_hit - Cache hits
           1         0.00         0.01 cache_miss - Cache misses
           1         0.00         0.01 backend_conn - Backend conn. success
           1         0.00         0.01 backend_recycle - Backend conn. recycles
           1         0.00         0.01 fetch_length - Fetch with Length
          10          .           .   n_sess_mem - N struct sess_mem
           1          .           .   n_sess - N struct sess
           1          .           .   n_object - N struct object
           3          .           .   n_objectcore - N struct objectcore
           3          .           .   n_objecthead - N struct objecthead
           2          .           .   n_waitinglist - N struct waitinglist
           1          .           .   n_vbc - N struct vbc
         100          .           .   n_wrk - N worker threads
         100         0.00         0.81 n_wrk_create - N worker threads created
           1          .           .   n_backend - N backends
          20          .           .   n_lru_moved - N LRU moved objects
           1         0.00         0.01 n_objwrite - Objects sent with write
           8         0.00         0.06 s_sess - Total Sessions
         808         0.00         6.52 s_req - Total Requests
           1         0.00         0.01 s_fetch - Total fetch
      254951         0.00      2056.06 s_hdrbytes - Total header bytes
          14         0.00         0.11 s_bodybytes - Total body bytes
           4         0.00         0.03 sess_closed - Session Closed
         808         0.00         6.52 sess_linger - Session Linger
         108         0.00         0.87 sess_herd - Session herd
       31865         0.00       256.98 shm_records - SHM records
        1129         0.00         9.10 shm_writes - SHM writes
           1         0.00         0.01 backend_req - Backend requests made
           1         0.00         0.01 n_vcl - N vcl total
           1         0.00         0.01 n_vcl_avail - N vcl available
           1          .           .   n_ban - N total active bans
           1          .           .   n_ban_gone - N total gone bans
           1         0.00         0.01 n_ban_add - N new bans added
         808         0.00         6.52 hcb_nolock - HCB Lookups without lock
           1         0.00         0.01 hcb_lock - HCB Lookups with lock
           1         0.00         0.01 hcb_insert - HCB Inserts
         124         1.00         1.00 uptime - Client uptime
```

▼図5　負荷分散



スプロキシが用いられます(**図5**)。

リバースプロキシを用いた負荷分散の副次的効果としてセキュリティが向上します。インターネット配下にWebサーバを直に置かず、リバースプロキシサーバを間に挟むことによって外部からWebサーバへの直接的なアクセスを防ぐことができます。

ここでは最近Webサーバやリバースプロキシの分野で人気急上昇のNGINXを試してみたいと思います。NGINXはWebだけでなくHTTPやメールのリバースプロキシとしても使われます。

NGINXではNGINX.confを書き換えることで設定を行います(**リスト2**)。

設定例では、各Webサーバを「backend」という名前で定義し、80番ポートでlistenするように記しています。

ここまでの設定を行ったらVarnish Cacheを起動します。

```
# service nginx start
```

NGINXの場合は、死活監視の設定を行わなくてもWebサーバが落ちている場合は自動的に振り分け先から除外されます。商用バージョンであるNGINX Plusではhealth_checkというディレクティブが有効となり、たとえば5秒ごとに死活監視を行うといった設定が行えるようになります。**リスト3**が商用バージョンであるNGINX Plusでのヘルスチェック設定の例です。

コンテンツ圧縮

HTTPでは、配信するコンテンツをgzip圧縮してから送信し、Webブラウザ側でそのコンテンツを解凍するしくみがあります。このしくみによりネットワーク通信量を削減できます。しかしWebサーバ側で都度gzip圧縮を行うとWebサーバにCPU負荷がかかります。リバースプロキシを用いたコンテンツ圧縮機能ではこの処理をリバースプロキシが代行することで、Webサーバはそのままでコンテンツ圧縮が行えます。

ここでは再度NGINXでの設定を試してみたいと思います。コンテンツ圧縮の場合も先ほどと同様にnginx.confを書き換えることで設定を行います(**リスト4**)。

設定例を見ていただければわかりますが、gzip offをgzip onに書き換えてNGINXを起動することでコンテンツ圧縮が可能となります。

まずgzip off、すなわちコンテンツ圧縮がかかっていない状態で試してみます(**図7**)。

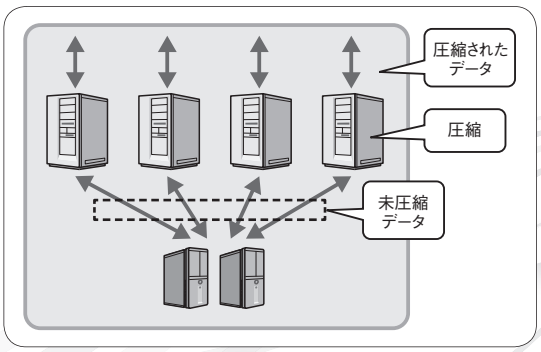
▼リスト2　/etc/nginx/nginx.conf

```
http {
  upstream backend {
    server 192.168.0.1:80;
    server 192.168.0.2:80;
    server 192.168.0.3:80;
    server 192.168.0.4:80;
  }

  server {
    listen 0.0.0.0:80;
    server_name www.example.com;

    location / {
      proxy_pass http://backend;
    }
  }
}
```

▼リスト3　NGINX Plusでのヘルスチェック設定例

```
location / {
    proxy_pass http://backend;
    health_check interval=5 fails=1 passes=1 uri=/;
}
```

▼図6　コンテンツ圧縮



今後はgzip onにしてみるとどうなるでしょう。出力結果が変わりコンテンツが圧縮されて配信されるようになりました(**図8**)。

今回紹介したコンテンツ圧縮機能はgzip圧縮を用いる方法です。gzip圧縮の特性上HTML、JavaScript、CSSなどのテキストデータに対する通信であれば圧縮率が上がり結果的に送信するデータ量が大きく削減されますが、画像データなどのバイナリデータの場合はgzip圧縮率が低く送信するデータ量はあまり変わりません。よってコンテンツ圧縮機能を用いる場合、通常は設定例のようにプレーンテキスト、CSS、XML、JavaScriptなど、gzip圧縮効果が高いファイルのみコンテンツ圧縮対象とします[注2]。

SSLアクセラレーション

低スペックのWebサーバに対してSSLのようなCPUリソースを多く使う処理が集中するとハードウェアリソースがすぐに枯渇してしまいます。このような場合はWebサーバのCPUを増強するハイスペック化によって対処することもできますが、SSLアクセラレーター(もしくはSSLオフローダーとも呼ばれる)用途としてリバースプロキシを用いることで、Webサーバはそのままで SSL対応強化が行えます(**図9**)。

SSLアクセラレーションはリバースプロキシの仲間ですが、ハードウェアとソフトウェアが一体化したアプライアンスサーバとして販売されていることが一般的です。

シングルサインオン

シングルサインオンは、複数システムにおける認証を1つに集約することで、ユーザは1回認証を受けるだけですべてのシステムを扱うことができるようになります。

リバースプロキシ型シングルサインオンでは、クライアントとのやりとりをリバースプロキシが代行するしくみであるため、たとえば「リバー

注2) NGINXではgzip onするとhtmlファイルが標準的にgzip圧縮対象となりますので今回の設定例からは除外してあります。

▼図7 コンテンツ圧縮がかかっていない状態

```
# curl -H "Accept-Encoding:gzip,deflate" http://(NGINXサーバのIPアドレス)/
Welcome!
test test test test test test test test
```

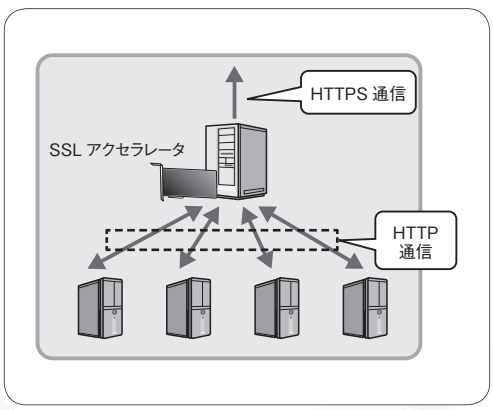
▼図8 コンテンツ圧縮がかかった状態

```
# curl -H "Accept-Encoding:gzip,deflate" http:// (NGINXサーバのIPアドレス)/
+O?I??MU(J,60??*I-.Q '?????<
```

▼リスト4 /etc/nginx/nginx.conf

```
#gzip off
gzip  on;
gzip_comp_level 9;
gzip_disable "MSIE [1-6]¥.";
gzip_http_version 1.1;
gzip_min_length 1000;
gzip_proxied off;
gzip_types text/plain
          text/xml
          text/css
          application/xml
          application/xhtml+xml
         application/rss+xml
          application/javascript
          application/x-javascript;
```

▼図9 SSLアクセラレーション



スプロキシにアクセスしてきたユーザに対して認証を行い、通過しないと配下のサーバに通信を代行しない」といったしくみを構築できます。

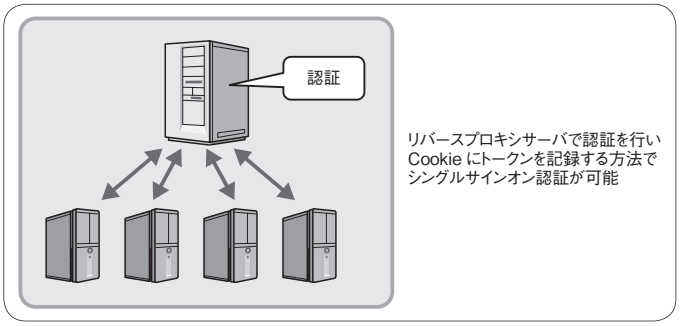
最後に

一口にリバースプロキシと言っても、非常にさまざまな使われ方をされていることがおわかりいただけたかと思います。その中には今回リバースプロキシとして紹介しましたが、世の中一般的にはキャッシュサーバやロードバランサなどと呼ばれていてリバースプロキシであることがあまり認識されていないものもあったかと思います。

リバースプロキシの役割は、サーバハードウェアやネットワーク回線の進化により時代とともに位置づけが変わってきたと思います。昔はサーバのパワーが低くネットワーク回線も細かったので、限りあるハードウェアやネットワーク資源を節約する目的でリバースプロキシが用いられていたように思います。しかし最近は余りあるサーバのハードウェアパワーと太いネットワーク回線により、高速なレスポンスをさらに高速にするためにリバースプロキシが使われているような気がします。

今後ハードウェアとオープンソースがどのように進化していくのか楽しみですね。SD

▼図10　シングルサインオン



CHAPTER 3

クラウドサービスで提供される プロキシサーバの利用方法と応用例

株式会社ハートビーツ　馬場 俊彰（ばば としあき）
Twitter @netmarkjp

本章ではプロキシの応用例として、プロキシのテクノロジを利用したクラウドサービスとその利用方法を紹介します。便利なサービスがたくさんありますのでこの機会にぜひ触れて感じてみてください。

はじめに

筆者が勤めるハートビーツでは、インターネットサービスのためのシステムの運用管理・監視をアウトソースで請け負っています。ここ数年は新規のシステムの9割以上がクラウド基盤を利用しています。クラウド基盤ではさまざまなサービスが提供されていますが、その中にはプロキシの技術を利用したものが数多くあります。そこで本章ではクラウドにおけるプロキシ型サービスを中心に、Amazon Web Services（AWS）を題材にプロキシサーバの応用例を紹介します。

今回紹介するのは下記のサービスですが、ぱっと見でプロキシというテクノロジの応用範囲の広さを感じていただけると思います。

・クラウド型ロードバランスプロキシ
・クラウド型コンテンツキャッシュプロキシ
・クラウド型セキュリティプロキシ
・クラウド型メールプロキシ
・クラウド型ストレージプロキシ

クラウド型ロードバランスプロキシ

リバースプロキシを利用してロードバランス機能を提供するサービスは、多くのパブリッククラウドサービスで提供されています。代表的なサービスとしてAWSのElastic Load Balancing（ELB）があります（**図1**）。HTTP/HTTPSのバランサとして使うことが多いですが、TCP/TCP+SSLのバランシングもできます。

L7ロードバランサとして見ると機能的には非常にシンプルで、sorryサーバ設定機能や代替コンテンツ返却機能、バックエンド側でのリトライの機能はありません。

分散方式は一般的にround robinやleast connectionなどがありますが、ELBにおいてはElastic Load Balancing Developer Guide（API Version 2012-06-01）によると、

By default, a load balancer routes each request independently to the application instance with the smallest load.

とのことです。しかし筆者が確認した範囲では、バックエンド側インスタンスのCPU使用率を見てくれるわけではなさそうですので[注1]、あまり過度な期待はしないほうがよさそうです。

なおELB自体の冗長化について検討したい場合は、同じくAWSのAmazon Route 53というDNSサービスを利用して、DNSサーバからのヘルスチェックとDNSレベルでのフェイルオーバーを設定できます（**図2**）。いわゆるグローバ

注1）複数のインスタンスをELB配下にぶらさげた状態で「dd if=/dev/zero of=/dev/null bs=1M count=`expr 1024 '*' 1024 '*' 1024`」として意図的に特定インスタンスの負荷を上げて5分ほど様子をみた結果、各インスタンスに振り分けられるリクエスト数に変動はありませんでした。

ルサーバロードバランス(GSLB)機能のように、Route 53がELBに対してヘルスチェックを実施し、応答が悪いELBを切り離してくれます。

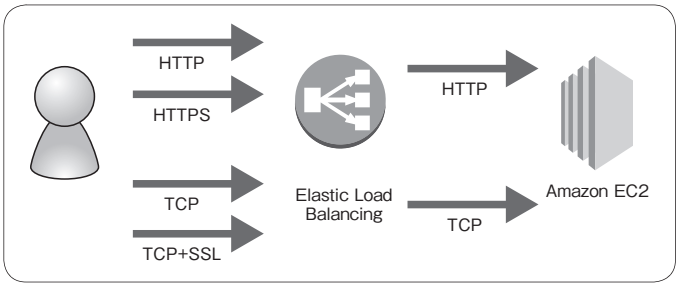
自動スケーリングがクラウドらしくてイイ

筆者がとてもクラウドらしくていいなと思うのは、ELB自身の自動スケーリング機能です。ELBのキャパシティはユーザがとくに指定する必要も(方法も)なく、AWS側で自動的に決定されます。そしてアクセスが増えた場合には自動的にキャパシティが向上し、ユーザがあらかじめキャパシティを指定する必要がありません。

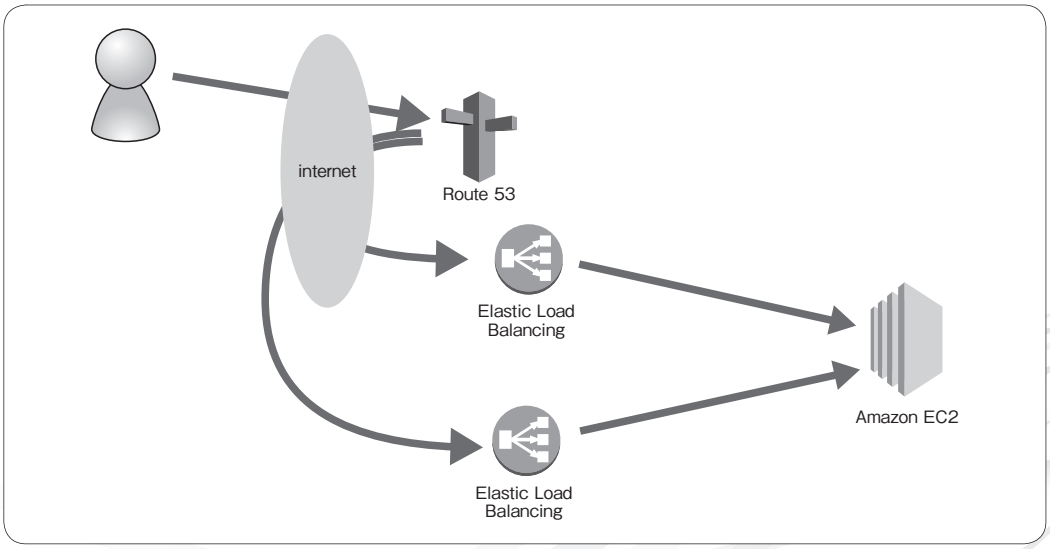
最近はHTTPSでのサービス提供が当たり前になってきています。サイトを全面的にHTTPSに移行するとSSL暗号化・復号化の負荷が問題になりがちですが、ELBを利用することでその部分をまるっとAWSに任せることができます。

ただしELBの自動スケーリングは瞬時にスケーリングするわけではないので、処理が完了するまでの間は利用者に対してELBの処理性能が不足した状態になります。あらかじめ(ELB的に)突発的なトラフィックが流れることがわかっている場合には、サポート窓口から暖機運転の依頼をできます(サポートレベルビジネス以上)。この暖機運転によりあらかじめELBのキャパシティを増やしておくことができ、アクセスのとりこぼしを防ぐことができるかもしれません。ただしTwitterでのバズりなど予測不可能な突発アクセスに対処するためには、すべて

▼図1　AWSのElastic Load Balancingの動作



▼図2　AWSのAmazon Route 53の動作



ELBに依存するのではなく後述するCloud Frontも併用しましょう。

IPアドレスやSSLクライアント証明書による接続元制限や同時接続数制限、一定時間内の接続数制限のような細かいアクセス制御はできませんが、そのぶんシンプルで使いやすくなっています。このシンプルさが目的にマッチするようであればぜひご活用ください。

クラウド型コンテンツキャッシュプロキシ

多数のユーザへの高速なコンテンツ配信を実現するためのソリューションとしてContents Delivery Network(CDN)というサービス形態があります。コンテンツキャッシュサーバをインターネット上に多数配置することで、ユーザに迅速にコンテンツを配布できるようになります。

オンデマンドで利用でき、費用的・工数的にたいへん利用しやすいサービスとしてAWSのAmazon CloudFrontがあります。CloudFrontを利用することで、短時間で従量課金制のCDNサービスをセットアップし利用できます。

CloudFrontはプロキシサーバとして動作し、レスポンスのキャッシュ制御命令に基づきコンテンツをキャッシュします。CloudFrontがユーザからのアクセスを受付・配信することで、サーバ側のリソース(おもにCPU・メモリ)やネットワーク帯域を節約できます。

AWS推奨の鉄板構成は前段にCloudFrontを置き、動的サイトのバックエンドはELB→EC2、静的コンテンツはS3に配置してCloudFront経由で配信というもので、通常のWebサイトであればこの構成が一番ユーザレスポンスがよくなると思います(**図3**)。

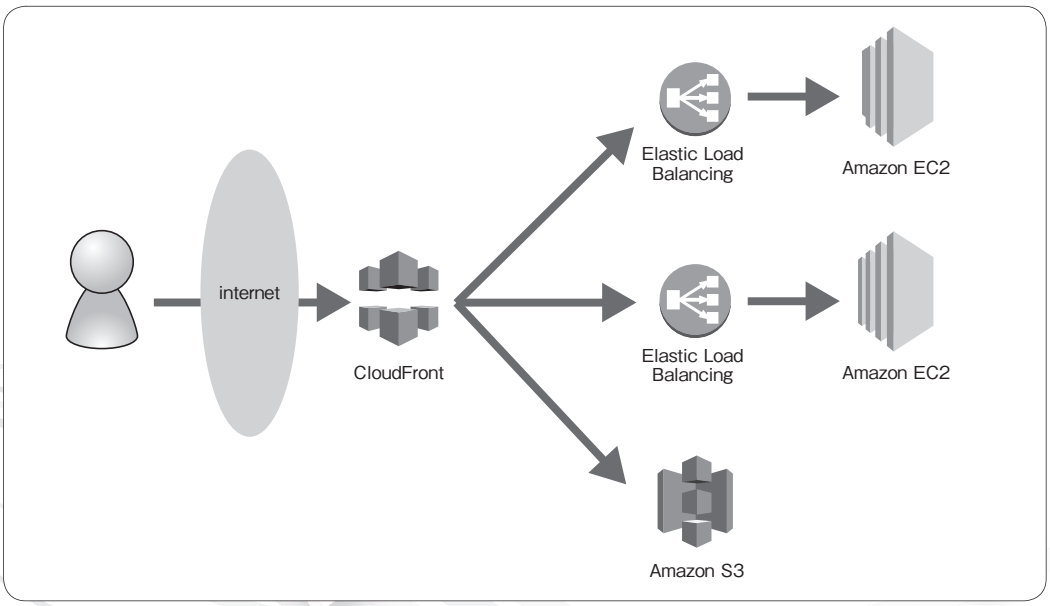
CloudFrontを物理サーバと組み合わせて使う

突然ですが、

「ホスティングサーバで運用しているあのサイトが、TVで取り上げられることがきまりました。3日後に」

と言われたらどうしますか？　困りますね。そんなときに使える、既存のサイトと組み合わせて、必要なときに必要なだけ使えるCloudFront

▼図3　CloudFrontの構成



の使い方を紹介します。

①CloudFrontを前段に配置する
②コンテンツを書き換えて特定の静的ファイルのみCloudFrontで配信する
③リダイレクトして特定の静的ファイルのみCloudFrontで配信する

CloudFrontは今や独自ドメインのSSL証明書やルートドメインホスティング、カスタムエラーページ、POSTメソッドにも対応しとても使いやすくなっています。HTTPSについては、「*.cloudfront.net」というFQDNのものであれば、証明書の手配なども必要なくすぐに利用できるのでたいへん捗ります。

「CloudFrontを」というよりはCDNを利用するうえで1点注意しなければならないのは、キャッシュ制御についてです。キャッシュの破棄(= purge)に少し手間がかかるので、キャッシュ制御を柔軟に実施する必要がある場合にはCDNを使わずに、Varnishなどを利用してうまく制御するのが良いでしょう。

またサイトのデザインリニューアルなどのために、あるタイミングからキャッシュをすべて破棄したい場合には無理に破棄しようとせず、別のdistributionを作りDNSで切り替えるなど柔軟に対応してください。

①CloudFrontを前段に配置する

まずはCloudFrontを前段に配置する方法です(**図4**)。

・既存サーバ側(オリジン)にオリジン用のVirtual Hostを定義する
・CloudFrontでdistributionを作成し、オリジンを作成したVirtualHostに向ける
・DNSを書き換え、ユーザのアクセスがCloud Frontに流れるようにする

この方法は3つの中で一番配信効率が良いのですが、プロキシを経由することを考慮していないアプリケーションがないか注意が必要です。万が一キャッシュ制御命令が適切に設定されていなかった場合に、ユーザ個別であるはずのデータが混ざるなど大惨事になる可能性があるため、事前に十分に動作検証・技術検証できる状況で利用することをお勧めします。

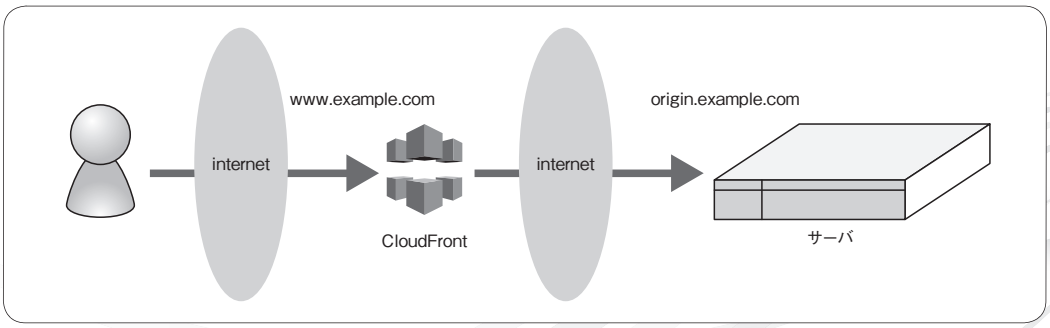
またアクセスされる際のFQDNを意識するアプリケーションの場合には、サーバ着信時のHostヘッダの値が変わるため動作不具合を起こす可能性があります。

②コンテンツを書き換えて特定の静的ファイルのみCloudFrontで配信する

次にコンテンツを書き換えて特定の静的ファイルのみCloudFrontで配信する方法です。これはいたって簡単で、CloudFrontでオリジンをサイトにしたdistributionを作成し、サイトの中の静的ファイルのURLをCloudFrontに向けてしまえばOKです(**図5**)。

静的ファイルとはいえ.js(JavaScript)ファイ

▼図4　CloudFrontを前段に配置する場合



ルはSame-Origin Policyの影響で配信元FQDNにより動作が変わる可能性があるため、適用するサイトで動作問題が出ないか十分に注意して検証してください。

よくあるミスとして同一URLでHTTPとHTTPSを両方提供している場合に、URLをHTTPで記載してしまうことがありますので注意してください。

この方法で、

・上流が10Mbps共有回線
・Wordpressをベースにカスタマイズ
・専用サーバホスティングサービスを利用

という環境で3日後のTV放映を乗り切ったことがあります。期間的・費用的にサーバのスペックアップやネットワーク増強などは難しかったため、CloudFrontを活用することで普段の3倍以上のアクセスにも対処できました。また短期間のアクセスのために長い契約期間のサービスを利用する必要もなく、クラウドの良さであるオンデマンドのリソース割り当てにより、チャンスを逃さず対応できました。

③リダイレクトして特定の静的ファイルのみCloudFrontで配信する

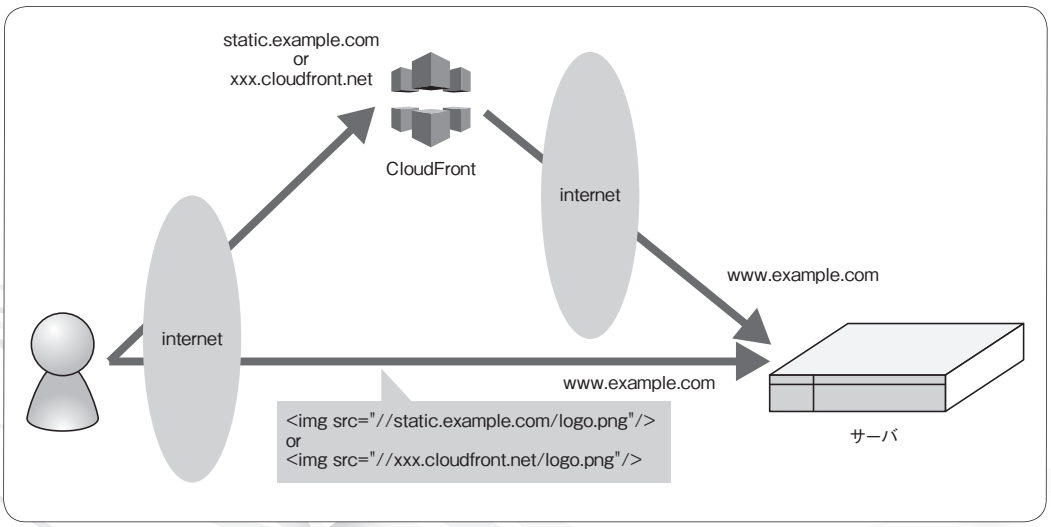
最後にリダイレクトして特定の静的ファイルのみCloudFrontで配信する方法です。

①か②の方法で対応できれば良いのですが、①だとピーク以外のときも常時トラフィックがCloudFrontを流れるため課金が心配、②だとコンテンツ書き換えの手間があり対応しきれないということは往々にしてありますよね、そんなときに利用できる方法です。

一度オリジンサーバでリクエストを受け、CloudFrontへリダイレクトする方法です。オリジンサーバが受け付けるリクエスト数は変わりませんが、処理が軽くなり転送量も格段に減るためマシンリソースやネットワーク帯域の節約ができます(**図6**)。①、②の方法と比較すると節約効果は落ちますが、既存サイトへの影響が小さい・切り戻しが容易などのメリットがあるので、導入までに時間がなく最大効率よりも安全性をとりたい場合にお勧めの方法です。

オリジンサーバでApacheを利用している場合、下記のように「mod_rewrite」で特定コンテンツのみ転送できます。「mod_rewrite」では日時を条件にすることも可能ですので、ピークタ

▼図5　コンテンツを書き換えて特定の静的ファイルのみCloudFrontで配信する



イムのみCloudFrontを利用するなどの仕込みもできます(**リスト1**)。

なお、あらかじめ②の準備ができているのであれば、③の方法を使わずに必要なときのみDNSを切り替える方法のほうが良いと思います。DNSとして普段からAmazon Route 53を利用しておけば、APIベースで動作をプログラミングできるため、このようなアジリティの高い運用を容易に実現できます。こちらの場合も前項の方法同様にHTTP/HTTPSの混在に注意してください。

この方法を使って、通常時ピーク700Mbps程度の物理サーバ構成のメディアサイトで、ピーク時のみ2Gbps相当のトラフィックを処理できるようになりました。

クラウド型セキュリティプロキシ

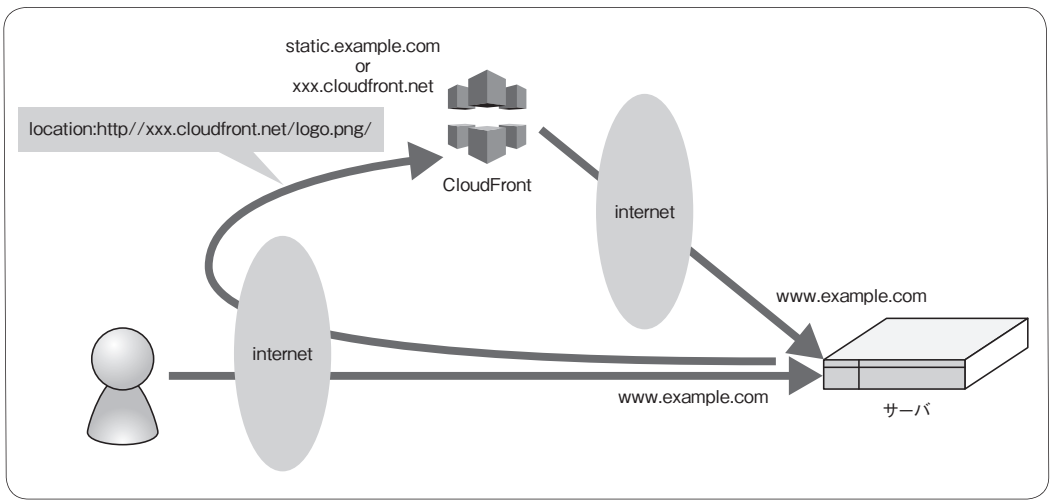
リバースプロキシのテクノロジのおもしろい活用方法を紹介します。Web Application Firewall(WAFと略されたりします)をご存じでしょうか？　昨今騒がれてきたSQL InjectionやOS Command Injection、Directory Traversalなど、Webアプリケーションに対する攻撃に対処するためのテクノロジです。リバースプロキシがリクエストの内容を精査し攻撃をreject(拒絶)することで、バックエンドのWebサーバに対する攻撃をブロックします(**図7**)。

クラウド型でSaaSとして利用可能な製品として、セキュアスカイ・テクノロジ社が提供するScutum[注2]というサービスがあります。

リバースプロキシ型でサービスを提供することで、オリジンサーバ側へのリクエストの精査だけでなく、オリジンサーバからのレスポンスも精査し情報流出をブロックするしくみになっています。

注2) http://www.scutum.jp/outline/saas_waf.html

▼図6　リダイレクトして特定の静的ファイルのみCloudFrontで配信する



▼リスト1　.htaccess記載例

```
RewriteEngine On
RewriteCond %{HTTP_USER_AGENT} !CloudFront
RewriteCond %{HTTP_REFERER} !.swf$
RewriteCond %{HTTP_REFERER} !.xml$
RewriteCond %{REQUEST_URI} ^/resize_image.php$ [OR]
RewriteCond %{REQUEST_URI} .(jpe?g|gif|png|bmp|ico)$ [NC]
RewriteRule ^/?(.*)$ http://xxx.cloudfront.net/$1 [QSA,NE,R=302,L]
```

メールもリバースプロキシでセキュリティ対策

前項ではHTTP、HTTPSでのセキュリティゲートウェイサービスを紹介しましたが、メールでも同じような実装ができます(**図8**)。

送受信時のウィルスチェック・スパムチェックをゲートウェイ側で提供することで、オリジンサーバ(この場合はPOP/SMTPサーバ)での負荷・運用の手間などを回避できます。

受信メールについてはDNSのMXレコードをセキュリティゲートウェイサービスに向け、送信メールについてはメールサーバでスマートホスト機能を利用し、全メールがセキュリティゲートウェイを経由するよう設定します。

ウィルスメール・スパムメールはいつまでたっても悩みの種ですので、アウトソースできるとたいへんすばらしいですね。

クラウド型メールプロキシ

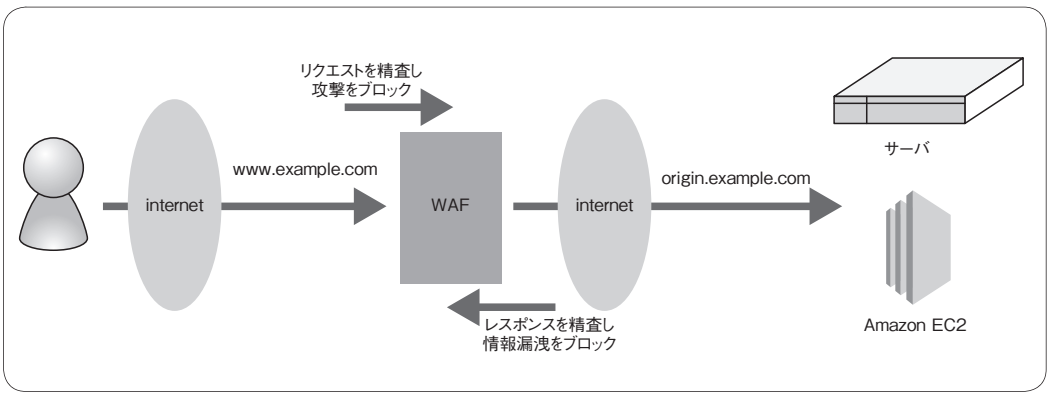
メールについてはすでにセキュリティサービスを紹介しましたが、もう1つたいへん助かるサービスを紹介します。

ご存じの方も多いと思いますが、メールの大量配信はとてもたいへんです。そこでたいへんな部分を肩代わりしてくれるクラウドサービスとしてAWSのAmazon Simple Email Service(Amazon SES)があります(**図9**)。

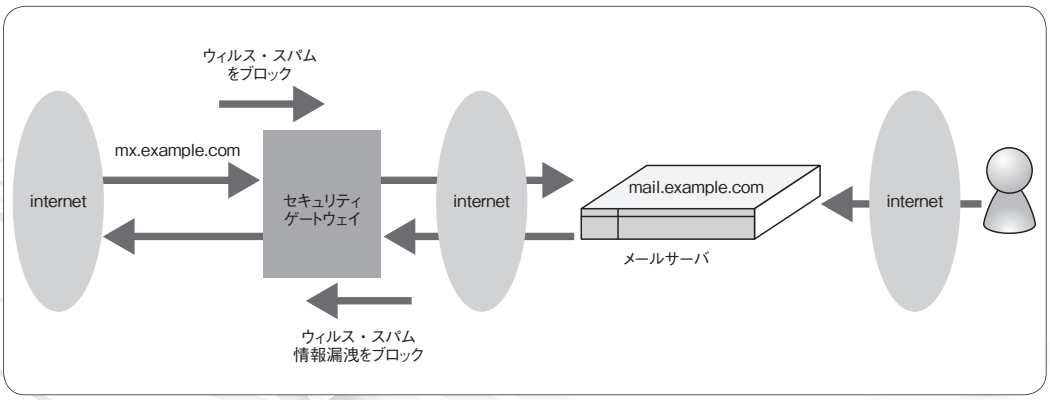
テクノロジ的にフォワードプロキシと言えるか微妙なところではありますが、スマートホスト機能でオリジンサーバからのアウトバウンド

注3) http://sendgrid.com/

▼図7 WAFの動作



▼図8 メールでのセキュリティゲートウェイ



をサービスへ流し込むことで、実際の配信時の諸問題を肩代わりしてくれます。

最近は同種のサービスであるSendGrid[注3]も日本に上陸しました。配信のみではなく解析などの機能も充実していて、今後国内での利用も増えていきそうです。

クラウド型ストレージプロキシ

最後に少し変わったプロキシを紹介します。AWS Storage Gatwayというサービスです。

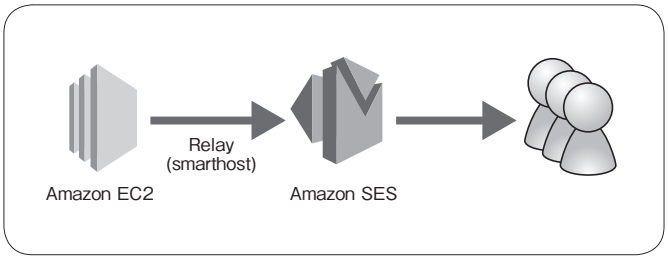
用意された仮想マシンイメージをローカルネットワークの仮想化基盤(VMware ESXi またはMicrosoft Hyper-V)で起動することで、そのインスタンスがAWSのストレージサービスであるAmazon Simple Storage Service(Amazon S3)やAmazon Glacierのゲートウェイになります(**図10**)。

ユーザから見るとStorage Gatewayの仮想マシンがS3やGlacierへのフォワードプロキシとして機能し、データのキャッシュなどを実現してくれます。

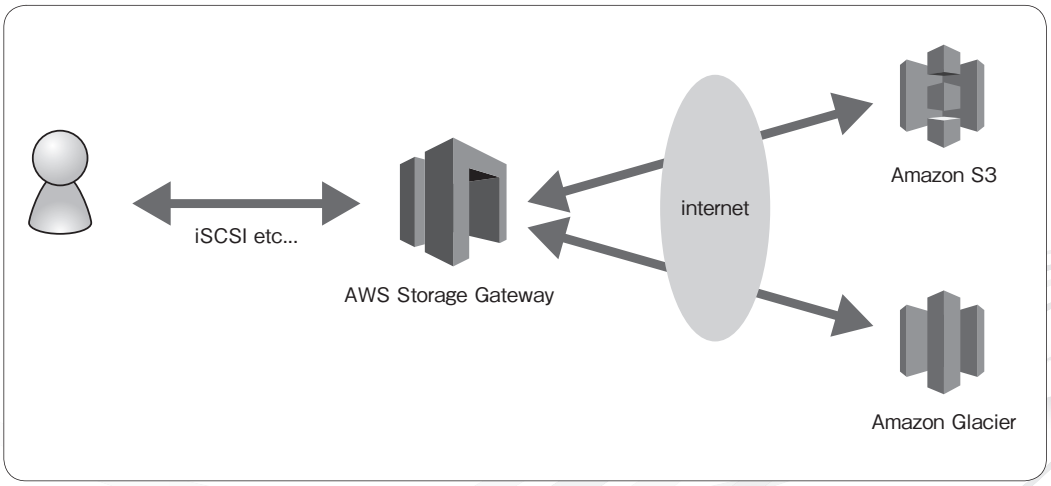
最後に

本章ではプロキシのテクノロジを利用したクラウドサービスと、その利用方法をいくつか紹介しました。プロキシは処理の委譲の典型的な構成でもあり、多くのクラウドサービスで利用されています。とても応用範囲の広いテクノロジですので、うまく活用できるようになりましょう。クラウドサービスを企画される方の参考にもなりましたら幸いです。SD

▼図9　Amazon SESの概念



▼図10　AWS Storage Gatwayの概念



短期集中連載

Mac as a desktop Service -MaaS- !?

#1

さらに踏み込む、Mac OS Xと仮想デスクトップ

複数のOS環境を必要とするMac使いのエンジニアにとって、仮想デスクトップ環境をMacに構築することはもはやあたりまえのことのようです。筆者もその一人ですが、日常的に使っているうちにOS間を行き来するオペレーションに煩わしさを感じるようになりました。この短期連載では、筆者がこの煩わしさから解放されるために行った、普通とはちょっと違ったアプローチをご提案したいと思います。

後藤 大地（ごとう だいち）（有）オングス 代表取締役

合理的な選択肢？「仮想デスクトップ」

Software Design 2013年10月号第2特集「開発するならやっぱりMacですよね？」では、さまざまなエキスパートの方々のMacの使い方が紹介されました。誠に恐縮ながら自分の開発環境も紹介させていただきましたが、Mac OS X以外のオペレーティングシステム（以下、OS）を使うために仮想化ソフトウェアを導入している話もいくつか掲載されていました。

仮想化ソフトウェアを導入して複数のOSを使う理由はさまざまだと思いますが、そういったことが実用的に実施できるパワーが現在のマシンにあるというのは間違いのないところです。実際、自分はそのように活用していますし、十分に機能しています（**図1～3**）。

▼図1　Mac OS X Mavericksスクリーンショット

▼図2　Windows 7 on Mac OS X ～仮想デスクトップとして動作するWindows

▼図3　Linux Mint on Mac OS X ～仮想デスクトップとして動作するLinux Mint

便利だけれどストレス感じてませんか？

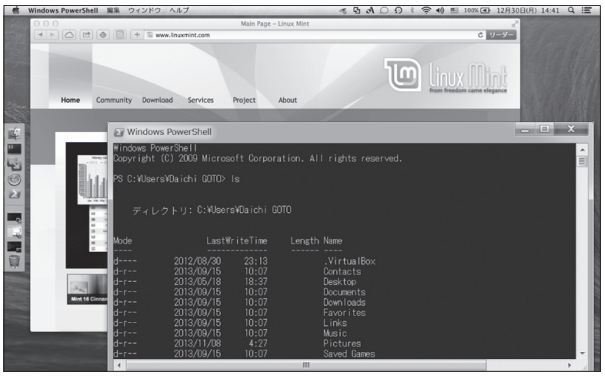
プロセッサのマルチコア／メニーコア化が進んだこと、仮想化技術が実用的なレベルで利用できること、リーズナブルな価格で豊富にメモリを積んだマシンが利用できることなどの要因があって、ノートPCで複数のOSを活用することが日常的に行えるようになりました。現在

のハードウェアスペックと仮想化ソフトウェアの性能を考えると、仮想環境で動作しているOSの動きに日常的な鈍重さを感じることはありません。

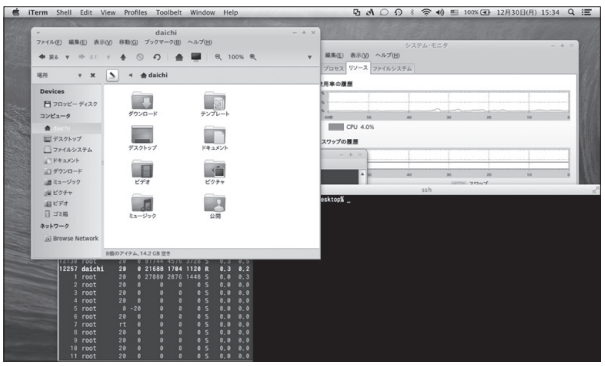
ただし、こうした仮想デスクトップに慣れてくると、いくらか不便な点がでてきます。たとえばUNIX系OSを使って開発をしながら、iMessageでメッセージングもしたいし、PowerShellも目の届く範囲においておきたい。Mac OS Xですべてやろうとすれば、あのコマンドがない、あの設定がないといった苛立ちを感じます。UNIX系OSでやろうとすれば、Flashプレーヤのプラグインがないとか、Javaプラグインが動かないとか、iMessageが使えないなど、やはり不便です。

仮想化ソフトウェアはそういった用途に応える機能も提供しています。たとえばVMware Fusionの「ユニティ」、Parallels Desktop for Macの「Coherence」、VirtualBoxの「シームレス」といった機能です。これらの機能を使うとゲストで動作しているアプリケーションを、ホストで動作しているアプリケーションのようにホスト側に表示して使用することができます(**図4、5**)。

▼図4　VMware Fusionのユニティ表示でネイティブアプリのように振る舞うWindows PowerShell



▼図5　VMware Fusionのユニティ表示でネイティブアプリのように振る舞うXアプリケーション



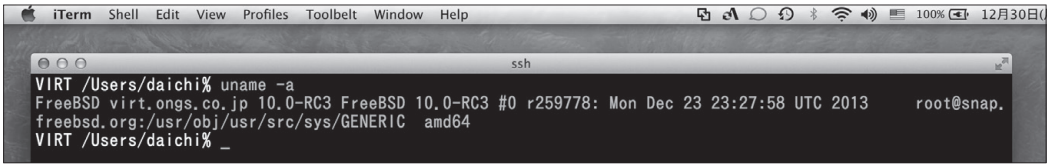
こうした機能も便利なのですが、ちょっとした操作、たとえば「通常の仮想デスクトップであればこの操作ができるのに、ユニティで使うとこの操作ができなくなる」程度のことが大きなストレスになります。コピー&ペーストができるところとできないところがある、ショートカットキースキーマの異なるウィンドウが同じ画面に混在している、これらは考えている以上にストレスです。

ある程度は慣れの問題もあるのですが、利用するアプリケーションごとに操作を変える意識を持つというのは本当の意味でのシームレスにはなりません。また、仮想化ソフトウェアが提供するこれら機能はRetinaディスプレイとの連動がうまくいかないケースもあり、これもストレスです。動くには動くが、できればさらになんとかしたいわけです。

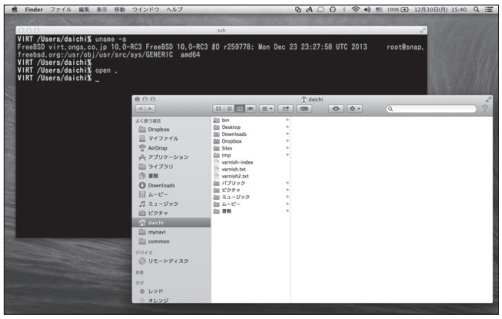
もっとOS間の使い勝手を良くしたい！

本短期連載の目的はそこにあります。こうした仮想化ソフトウェアの機能と、Mac OS Xが従来から提供している基本的な機能を組み合わせて、自分の扱いやすい融合環境を構築しよう、という内容です。最終的にはやっぱり仮想デスクトップをフルスクリーンで使ったほうが使いやすいという結論を得るかもしれませんし、Mac OS XのGUIをベースに融合させたほうが扱いやすくなったという結論が得られるかもしれません。このあたりは好みや使い方の問題もあるので、作る人それぞれ異なる結果が得ら

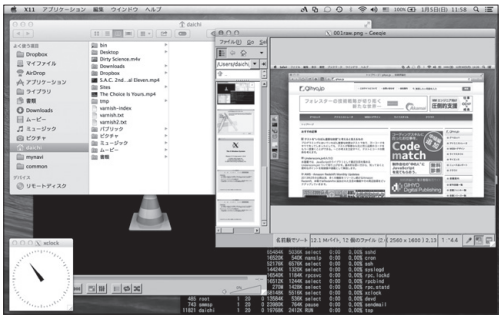
▼図6　ターミナルから仮想環境で動作するゲストOSにリモートログインしている様子



▼図7　ゲストOSでコマンドを実行すると
ホストOSのFinderが連動して動作



▼図8　XアプリケーションとMac OS X
アプリケーションを混在させた使用例



れると思います。まず、そのどちらも体験できるようにしよう、というのがこの短期連載の目的です。

組み上げに利用する基礎技術はどれも従来からよく知られたものです。しかし、Webアプリケーション開発からPCを使い始めた世代や、MacBook AirやMacBook Proのようによくまとまったデバイスから使い始めた世代では知らないことが多いでしょう。これら技術の基本的な機能や設定方法を紹介しながら、自分の手に合うように環境を融合していく方法を紹介します。

記事は3回に分けて掲載します。1回目と2回目は基礎技術の紹介、3回目は実際にどういった設定をすればよいのかの紹介です。実際に試しながら読めるようにしていきますので、作業環境をさらに洗練させるテクニックとして活用していただければと思います。

仮想デスクトップとMacのタイトな融合

Mac OS XにはMacPortsやHomebrewといったサードパーティ製のパッケージ管理システムが存在します。しかし、パッケージ管理システムの総合評価を考えると、やはりLinuxや*BSDのほうが登録されているソフトウェアの数も多く必要に応じて柔軟にカスタマイズできることから、Mac OS XではなくLinuxや*BSDを使いたくなります。GUI部分はMac OS Xを使い、ターミナルの中身がLinuxや*BSD、そしてネイティブのターミナルのようにLinux/*BSDターミナルとMac OS XのGUIがシームレスに連動する、そういった両者のいいところ取りの環境を組み上げるのが、今回構築を目指す環境です(**図6、7**)。

GUIはおもにMac OS X、CUIはおもにUNIX系OSといった使い方に割り切ると、商用アプリケーションを活用しやすくなるというメリットがあります。仮想デスクトップという形で独立して使用した場合、Mac OS XとUNIX系OSとで別々の日本語入力システムを使う必要があります。UNIX系OSの利用はMac OS Xのターミナルアプリケーション経由といった形に絞っておくと、Mac OS Xの日本語入力システムに統一して利用できます。辞書データを単一管理できますし、ATOKのように商用で提

供されている変換効率の良い日本語入力システムが使えます。

また、「Mac OS Xのアプリケーションでもよいのだけれど、どうしてもUNIX系OSでしか提供されていないGUIアプリケーションを使いたい」ということがあります。いままで使い慣れてきたアプリケーションは、なかなかほかのアプリケーションに置き換えられないものです。そうした場合に、UNIX系OSで動作するGUIアプリケーションをMac OS X側に表示して扱いたいわけです(**図8**)。

Mac OS XにもLinux/*BSDにも、こうしたしくみを実現するための機能が標準で備わっています。基礎技術を組み合わせるだけでそういった環境を作ることができます。このしくみは、ゲストOS向けのドライバが提供されていないOSをゲストとして動作させる場合にも有益です。ユニティやシームレス、Coherenceといった機能が利用できないOSでも、似たようなことを実現できます。

知るべき3つの基礎技術

ここでは次の3つの技術を使います。どの技術も現在のUNIX系OSでは標準で機能が提供されています。Mac OS X MavericksにはX Window Systemの実装系は含まれていませんが、「XQuartz」という形で別途提供されていますのでそちらを利用します。

- **ファイルシステム共有「NFS」**
- **リモートログイン「ssh」**
- **ウィンドウシステム「X Window System」**

デスクトップ(ワークステーション)向けのたいていのLinuxディストリビューションはウィンドウシステムにX.Orgを採用しています。X.OrgはX Window Systemの参照実装系ですので、デスクトップ向けのLinuxディストリビューションを扱ったことのある大半の方がX Window Systemを使ったことがある、ということになります。

X Window Systemのしくみを利用するだけなら、ディストリビューションに含まれているような大量のソフトウェアは必要なく、必要最低限のライブラリとxauth(1)[注1]だけで十分です。ssh(1)のX11フォワーディング機能を使うならxauth(1)も不要です。ただ、そもそもXサーバへの接続の認証処理がどういったものであるか理解しておいたほうがいろいろ細かいことができるようになりますし、パフォーマンス上の利点もありますので、この短期連載ではどちらのやり方も説明します。

目指すモデルはコレ

技術の組み上げ方次第でさまざまなことができます。ここでは内容を理解しやすくするために、次の2つのモデルを想定してシステムを組み上げていきます。

①MacBook AirやMacBook Proに仮想化ソフトウェアを導入。UNIX系OSを仮想環境で動作させ、ホスト(Mac OS X)とゲスト(UNIX系OS)を融合させる(図9)

②iMacやMac Pro、Mac miniなどの据え置きタイプのMacと、既存のLinux/*BSDマシンを融合させる(図10)

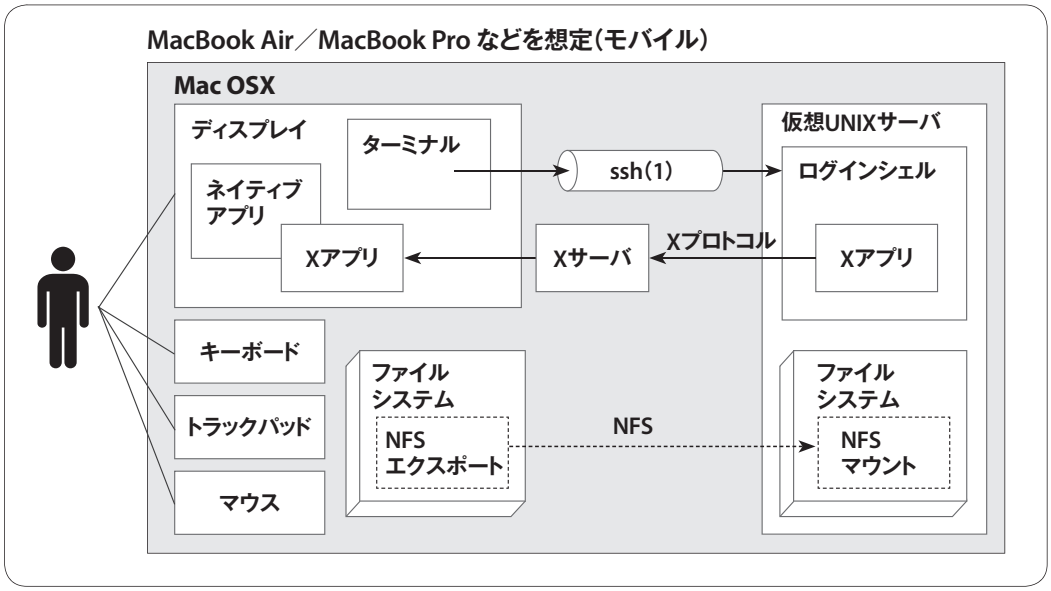
モバイルで移動するマシンと、据え置きで動かすことがないような2つのタイプを作ります。これらの組み合わせをさらに組み合わせると、据え置きとしてもモバイルとしても使える環境を用意できたり、さらに仮想化ソフトウェアが提供しているユニティ／シームレス／Coherenceなどの機能を組み合わせるなど、よりさまざまな面から融合を進めることができます。

説明に使用するモバイル環境は次のとおりです。

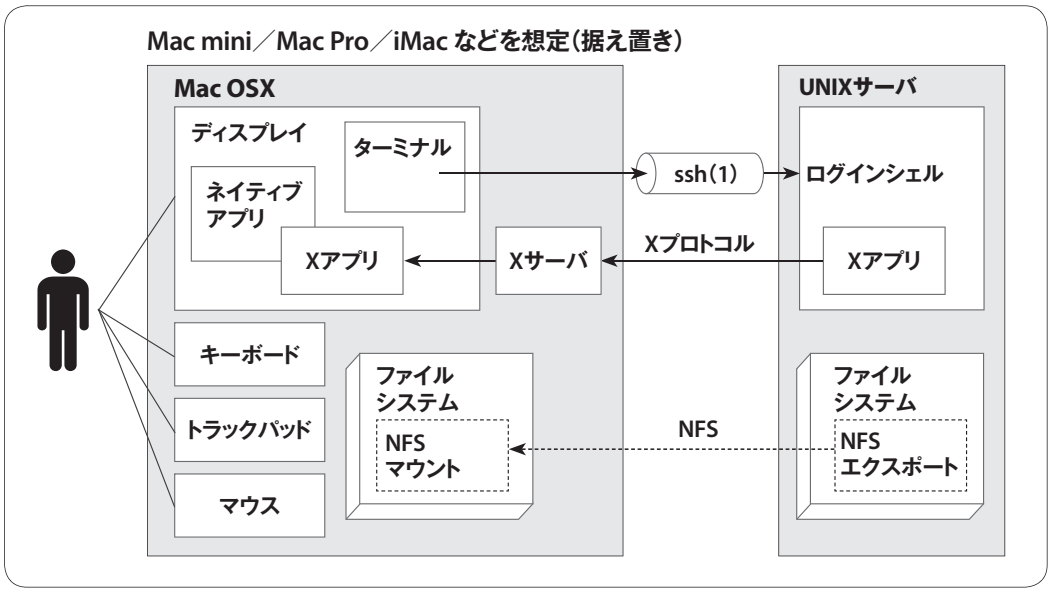
注1) コマンドのうしろにある括弧付きの数字は、manコマンドで見られるマニュアルに掲載されている章番号を表しています。

さらに踏み込む、Mac OS Xと仮想デスクトップ

▼図9　MacBook AirやMacBook Proのようなモバイル環境でのモデル



▼図10　Mac ProやiMac、Mac miniのような据え置き環境でのモデル



- ホストマシン：MacBook Pro Retina 13-inch Late 2013
- ホストOS：Mac OS X Mavericks
- 仮想化ソフトウェア：VMware Fusion 6
- ゲストOS：Ubuntu 12.04 LTS(64bit)
- ゲストOS：FreeBSD 10.0-RELEASE amd 64(64bit)
- ゲストーホスト間ネットワーク：Macを共有(NAT)

モバイル用途では接続するネットワークが移動しながら変わりますので、ゲストのネットワーク接続をブリッジにすることは扱いやすいとはいえません。ここではNATを経由する方法に

します。つまり、ホストとゲストを接続するネットワークのIPアドレスは常に固定です。

ホストの外側のネットワークアドレスは接続する場所によって変わります。その間のアドレス変換はVMware Fusionが提供するNAT機能が実施します。NATの処理が挟まるので通信性能は低下します。性能と利便性のバランスに関しては次回で解説します。

一方、説明に使用する据え置き環境は次のとおりです。

- **Macマシン：MacBook Pro Retina 13-inch Late 2013**
- **Mac OS：Mac OS X Mavericks**
- **UNIXサーバ：FreeBSD 10.0-RELEASE amd64（64bit）**
- **UNIXサーバーMac間ネットワーク：1000base-T（全二重）**

据え置きの説明にもかかわらずノートPCをテスト環境にしていますが、先に述べた据え置き型のiMac/Mac Pro/Mac miniでもやり方は同じですのでご容赦ください。

理由は実際自分がそうしているから、という面も大きいのですがほかにもあります。MacBook AirやMacBook Proを購入するユーザは繰り返し同じモデルの最新版を購入する傾向が見られます。たとえば2年ごとに最新のモデルを購入した場合、まだまだ使えるマシンが余るという状況になります。こうしたマシンを据え置き用に使い、これまで使ってきたUNIXサーバと組み合わせて動作させるといった用途を想定してみました。MacBookの出力をデュアルディスプレイやトリプルディスプレイに拡張して利用すると、作業スペースが広がります。1920×1080対応の液晶は廉価に購入できますので、費用の割に効果が見込める投資です。

仮想化ソフトウェアとしては、グラフィックのレンダリング性能を求める場合にはParallels Desktop for Macのほうが性能を発揮できると言われています。本稿でVMware Fusionを使っているのは、VMware Fusionは対応しているゲストOSの幅が広いこと、UNIX系OSがよく動作すること、USBシリアルケーブルをゲストから扱いやすいこと、ゲストからホストマシンのCPU仮想化機能を利用できることなどの理由があります。多種多様な環境を構築するには都合がよいソフトウェアです。オープンソースの仮想化ソフトウェアを使いたいという場合にはVirtualBoxを使えばよいと思います。

環境構築[1]ファイルシステムの共有「NFS」

まず、ファイルシステムを共有するためにNFS（Network File System）を使います。仮想化ソフトウェアを使用すると、何も設定しなければホストとゲストのファイルシステムはまったく別の分離したものになります。一部の領域を共有する機能は提供されていますが、ホストとゲストのファイルシステムの双方を共有するといったようなことはしません。

しかし、仮想デスクトップを使い続けると、こうしたファイルシステムの分断が面倒なものに思えてきます。いちいちデータをコピーする作業がストレスになってくるのです。どちらからもラクに使いたいので、ホストもゲストも、ユーザのホームディレクトリとデータディレクトリはいっそ同じファイルシステムを扱ってくれないかと感じるようになります。

NFSはネットワーク経由でファイルシステムを共有できるクライアントサーバモデルの分散ファイルシステムです。UNIX系OSの多くがデフォルトで対応しており、システムのデフォルトのファイルシステムとよく連動します。Mac OS X Mavericksは NFSv2、NFSv3、NFSv4に対応し、NFSサーバとしてもNFSクライアントとしても機能します。この機能を使えば先ほどの要求は実現できます。UNIX系OSを使う場合はNFSでファイルシステムを共有する設定を知っておけば、ファイルシステム共有でやりたいようなことはだいたいできると

思います。

NFSの使い方

NFSクライアントにマウントを許可するファイルシステム(またはZFSであればデータセット)に関する設定は、NFSサーバ側(例ではMac OS X)の/etc/exportsファイルに書きます。たとえば**リスト1**のように設定ファイルを書きます。この場合、Mac OS Xユーザ"daichi"のホームディレクトリ以下(/Users/daichi)は、192.168.185.0/24のネットワークに所属しているホストからマウントできる、という指定になります。1002はユーザdaichiのユーザIDで、このファイルシステムをマウントするNFSクライアンからのアクセスはすべてのこのユーザからのアクセスという扱いになります。

設定に必要な要素の具体的な調べ方は第3回で説明しますので、ここでは簡単にこれらの情報を調べる方法を紹介しておきます。まず、ユーザIDやグループIDなど、使っているユーザに設定されている値はid(1)コマンドで確認できます。また、ここでのネットワークアドレスはVMware Fusionが自動的に設定するものをそのまま使っています。Mac OS Xでifconfig(8)コマンドを実行するとvmnet8というインターフェースが見つかると思いますが、このインターフェースに192.168.185.1/24が割り当てられています。これに合わせてゲスト側に192.168.185.*/24のアドレスを割り当てて使うことになります。

Ubuntu(NFSクライアント)から、設定した領域をNFSマウントすると**図11**のようになります。マウントしたディレクトリ以下は、Mac OS Xの/Users/daichi/以下と同じディレクトリやファイルが表示されます。FreeBSDでも同じプロトコルで通信していますので、できることもほとんど同じです。

逆にMac OS XをNFSクライアントとして、

▼リスト1　NFSサーバ側の/etc/exportsファイルの設定例

```
/Users/daichi -mapall=1002 -network 192.168.185.0 -mask 255.255.255.0
```

▼図11　NFSクライアント(Ubuntuの場合)

```
UBUNTU /Users/daichi$ df
Filesystem                    Size  Used Avail Use% Mounted on
/dev/sda1                      28G  3.3G   23G  13% /
udev                          989M  4.0K  989M   1% /dev
tmpfs                         400M  800K  399M   1% /run
none                          5.0M     0  5.0M   0% /run/lock
none                          998M  152K  998M   1% /run/shm
192.168.185.1:/Users/daichi   466G  143G  322G  31% /Users/daichi
UBUNTU /Users/daichi$ ls
Desktop    Downloads  Library  Music     Public  Templates  bin
Documents  Dropbox    Movies   Pictures  Sites   Videos     tmp
UBUNTU /Users/daichi$
```

▼図12　NFSクライアント(Mac OS Xの場合)

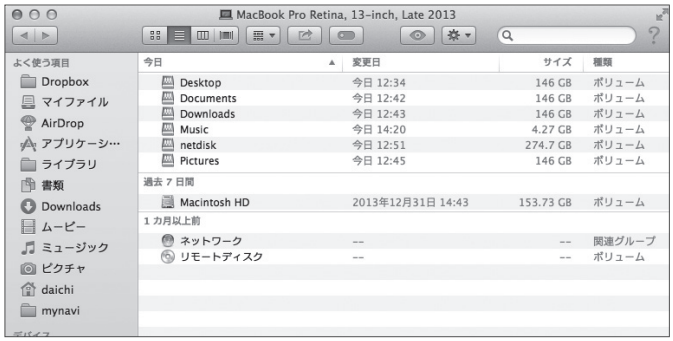
```
/Users/daichi% mount -t nfs
192.168.1.101:/z/daichi/Desktop on /Users/daichi/z/Desktop (nfs)
192.168.1.101:/z/daichi/Documents on /Users/daichi/z/Documents (nfs)
192.168.1.101:/z/daichi/Downloads on /Users/daichi/z/Downloads (nfs)
192.168.1.101:/z/daichi/Pictures on /Users/daichi/z/Pictures (nfs)
192.168.1.101:/z/daichi/Music on /Users/daichi/z/Music (nfs)
192.168.1.100:/n/Netdisk on /Users/daichi/n/Netdisk (nfs)
/Users/daichi%
```

ほかのUNIXサーバやNASストレージが提供しているNFS領域をマウントすることもできます。たとえばMac OS Xから NFSマウントした例を**図12**に示します。

NFSマウントした領域は指定したディレクトリにマウントされ、その領域はボリュームとしてFinderからもチェックできるようになります(**図13**)。Mac OS Xのユーザランドは FreeBSDのユーザランドを移植していますので、このあたりのしくみはFreeBSDと似ています。

NFSはユーザIDとグループIDをそのまま使いますので、基本的にはマウントする側とされる側でユーザ名、ユーザID、グループ名、グループIDなどを合わせておく必要があります。Mac OS X側のユーザIDを変更したりグループIDを変更するには dscl(1)ユーティリティを使うのですが、ちょっとばかり面倒なのと、操作を誤るとログインできなくなるなど弊害も出るので、本稿ではその方法は説明しないでおきます。これも具体的には第3回で説明しますが、Linuxや*BSD側でユーザIDやグループIDを変更するのには、vipw(8)コマンドや/etc/groupファイルを編集することで実施します。また、管理対象が多い場合にはNISという別の技術を合わせますが、本稿では個人が使う内容ということでこちらも割愛します。

▼図13　Mac OS X：NFSマウントした領域はボリュームとして扱われる



column 1　NFSv3とNFSv4のどっちがよい?

現在おもに使われているNFSのバージョンは3または4です。NFSv4はこれまでNFSv3で問題視されてきた部分の機能を整理してプロトコルに取り込まれています。NFSv4ではNISを使わなくてもユーザIDのマッピングが可能であるほか、ファイルロック関連の設定もNFSv3のときよりもすっきりしています。

ただし、NFSv4で提供された新しい機能をとくに必要としないのであれば、NFSv3のほうが安定しているところがあるので、NFSv3のほうがお勧めです。また、相互接続性に関しても、NFSv4を使うときよりもNFSv3を使うときのほうが優れているように感じます。

本稿では安定して動作するファイルシステム共有を求めていますので、とくに理由がなければNFSv3でシステムを組み上げ、試してみて問題がないようであればNFSv4も使ってみる、といったスタンスで取り組んでおくとよいと思います。

第1回のまとめ

短期連載第1回目の今回は、仮想デスクトップとMacデスクトップをもっと融合するために使う基本的な機能と構築方針を紹介し、NFSを利用する方法まで解説しました。次回はssh(1)とX Window Systemを解説し、第3回で具体的な設定方法や、違和感のない融合を実現するためのシステムの設定方法、コマンドの組み立て方などを解説する予定です。

ノートPCは開発者にとって仕事をするための重要な道具です。仕事道具として長い時間を共にする相棒ですから、自分にとって扱いやすいものへカスタマイズして、自分の仕事に磨きをかけたいところです。既製品を使うだけではなく自分だけのツールに仕上げていくことで、愛着のある仕事もできるようになるというものです :)

column 2 ファイルロックはNFSv3プロトコルには含まれていない?

NFSクライアントとして動作する場合、NFSサーバと通信するためのnfsd(8)デーモンまたはNFSカーネルスレッドが動作していればよいと考えそうになりますが、それだけでは都合が悪いことがあります。NFSv2やNFSv3のプロトコルにはファイルロックに関するものが含まれていません。ファイルロックを利用するソフトウェアをNFSクライアント側で実行しようとした場合、その機能は別のデーモンが提供する必要があります。

Ubuntu 12.04であれば[lockd]がそれに該当します(**図A**)。Mac OS XとFreeBSDならrpc.lockd(8)です(**図B、C**)。これらデーモンがファイルロック処理のやりとりをすることで、ファイルロック機能を使用するソフトウェア(たとえばバージョン管理システムなど)を適切に動作させることができます。

ほかにもステータスモニタリングを実施するデーモンが必要です。rpc.statd(8)が担当します。このデーモンが動作していないと、たとえばMacBook AirやMacBook Proであればサスペンド／レジューム実施後にNFSの動作がおかしくなります。rpc.lockd(8)もrpc.statd(8)も動作にrpcbind(8)が必要です。つまり必要最低限、nfsd(8)、rpcbind(8)、rpc.lockd(8)、rpc.statd(8)またはこれに類するソフトウェアを動作させる必要があります。

マウントしたファイルシステムで動作するソフトウェアと動作しないソフトウェアがある場合には、これらデーモンが動作しているか確認してください(NFSサーバ側にはさらにマウントを受け付けるデーモンなども必要になります。NFSv4になるとさらに動作するデーモンは増えます)。SD

▼図A　NFSクライアントに必要なプロセス(Ubuntu 12.04)

```
UBUNTU /Users/daichi$ ps axww | grep -E '(nfs)|(rpc)|(lock)' | grep -v grep
   18 ?        S<     0:00 [kblockd]
  571 ?        S<     0:00 [rpciod]
  572 ?        Ss     0:00 rpcbind -w
  584 ?        S<     0:00 [nfsiod]
  613 ?        Ss     0:00 rpc.statd -L
  617 ?        Ss     0:00 rpc.idmapd
 2620 ?        S      0:00 [lockd]
UBUNTU /Users/daichi$
```

▼図B　NFSクライアントに必要なプロセス(FreeBSD 10.0)

```
FREEBSD /Users/daichi% ps axww | grep -E '(nfs)|(rpc)' | grep -v grep
 621  -  Ss     0:00.02 /usr/sbin/rpcbind
 659  -  Ss     0:00.02 /usr/sbin/rpc.statd
 663  -  Ss     0:00.02 /usr/sbin/rpc.lockd
7668  -  SL     0:00.00 [newnfs 0]
FREEBSD /Users/daichi%
```

▼図C　NFSクライアントに必要なプロセス(Mac OS X Mavericks)

```
MAC /Users/daichi% ps axww | grep -E '(nfs)|(rpc)' | grep -v grep
   65   ??  Ss     0:00.03 /usr/sbin/rpc.statd -n
  111   ??  Ss     0:00.01 /usr/sbin/rpc.lockd
  113   ??  Ss     0:00.01 /usr/sbin/rpc.statd
  114   ??  Ss     0:00.08 /usr/sbin/rpcbind
  119   ??  S      0:00.48 /usr/sbin/rpc.lockd
 2229   ??  Ss     0:03.40 /sbin/nfsd
 2230   ??  Ss     0:00.01 /usr/libexec/rpc.rquotad
MAC /Users/daichi%
```

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2014年2月号

第1特集
λ式からはじめませんか?
関数型プログラミング再入門

第2特集
目利きによるトレンド予測
2014年IT業界はどうなるのか?

一般記事
・会社組織を活性化するスパイス「コンパ」

1,280円

2014年1月号

第1特集
シェルがわかればシステムがわかる
あなたの好きなシェルは何ですか?

第2特集
未来を作るITインフラ
10ギガビットを実現するケーブリング技術

特別付録 & 一般記事
・法輪寺鎮守社電電宮 情報安全護符シール Ver.2
・ソーシャルゲームのDevOpsを支える技術（後編）

1,380円

2013年12月号

第1特集
SDN/OpenFlowの流れを総まとめ!
SDN/OpenFlowで幸せになれますか?

第2特集
下手でも好印象で効果絶大
エンジニアの伝わる図解術

一般記事
・LinuxとFreeBSDのファイルシステムの良い・悪いところをご存じですか?
・「Mirama」a.k.a. VIKING ほか

1,280円

2013年11月号

第1特集
思考をコード化する道具
我が友 Emacs

第2特集
コンピュータの加速装置
サーバサイドフラッシュ Fusion-io全方位解説

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（5）
・ソーシャルゲームのDevOpsを支える技術（前編）
・Ubuntu 13.10 "Saucy Salamander"

1,280円

2013年10月号

第1特集
Vimを使いこなしてますか?
Vim至上主義

第2特集
ネットワーク技術力のパワーアップ
ルータの教科書

一般記事
・Key Value Storeをゼロから創る
・小規模プロジェクト現場から学ぶJenkins活用（4）

1,280円

2013年9月号

第1特集
今からはじめる
sed/AWK再入門

第2特集
開発するならやっぱりMacですよね?
9人9色のデスクトップ拝見＋新OS傾向と対策

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（3）
・最終段階に入ったFedora 19

1,280円

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。

より速く、より莫大に、より高みへ！

サーバマシンの測り方

ベンチマークを極める実践テクニック

第4回　Xeon E5-2600 v2を測る

Ivy Bridgeマイクロアーキテクチャの Xeon E5-2600 v2搭載のサーバが出始め、前世代の Sandy Bridgeマイクロアーキテクチャ（Xeon E5-2600）から世代交代が進んでいるころかと思います。今回は Sandy Bridgeから Ivy Bridgeへ CPUモデルを変更した弊社での事例を紹介します。

Writer ㈱IDCフロンティア　ソリューションアーキテクト　藤城拓哉（ふじしろたくや）／Twitter@tafujish

新CPU導入時に気をつけるポイントとは

当社のサービスにて、これまでSandy Bridgeマイクロアーキテクチャ世代のXeon E5-2670を利用していたところを、今後のサーバ増設からは新しくリリースされたIvy Bridgeマイクロアーキテクチャ世代のXeon E5-2600 v2シリーズを利用することになりました。そこで、新しいCPUのモデルを何にするかという話になります。モデルを決めるための要件は、費用が上がらないこと、消費電力が上がらないこと、同等以上の性能が出ることの3つでした。また、同等以上の性能とは、シングルスレッドとマルチスレッド両方の性能において達成する必要がありました。

昨今のCPUは、ターボブーストや省電力の機能が入り動作周波数が動的に変化します。モデルに応じた動作周波数やコア数だけで比較することが難しくなってきました。そこで、前半ではCPU性能にかかわってくるポイントについて調査し、新しいCPUの特性を把握します。後半では2つの世代のCPUについて性能を比較します。

現行のサーバは、Xeon E5-2670（8コア、2.6GHz）のCPUを2基搭載しています。新しいCPUの評価対象としてXeon E5-2697 v2（12コア、2.7GHz）のCPUを2基搭載したサーバを用意しました。

本来であれば、動作周波数やコア数が同じCPUモデルで比較したほうがわかりやすいのですが、残念ながら今回は用意できませんでした。UnixBench v5.1.3を、

```
# ./Run
```

と実行した結果で比較します。また、CPUのコア数が16より多いため、UnixBenchのサイトのIssuesにある「Can't do default run completely with > 16 CPUs[注1]」のパッチを適用しています。

ハードウェアパワーマネージメント

昨今のサーバには、そのハードウェアで消費電力と処理性能のバランスをとる省電力機能が付いています。つまり、消費電力あたりの処理性能が最大となるので、CPU本来の最高性能が出るわけではありません。

BIOSやUEFIの電力関連の設定から、「最大性能」と「電力と性能のバランス」のそれぞれに設定したときのUnixBenchのスコアを比較します。結果は**表1**のとおりで、1並列、複数並列ともに「最大性能」のほうが高速で、1並列では5割近くの性能比が出ました。

単純にCPUの処理性能を測定し比較するのであれば、「最大性能」でCPU本来の性能で比較するのが良いでしょう。また、本番環境にお

注1）https://code.google.com/p/byte-unixbench/issues/detail?id=4

▼表1　ハードウェアパワーマネージメントのUnixBenchスコア比較（E5-2697 v2）

	1並列	48並列
最大性能	1572.3	11130.8
電力と性能のバランス	1067.2	8654.9

いても電力に問題や制限がなければ「最大性能」を選ぶと有効でしょう。なお、この設定のデフォルト値は、サーバメーカーによって異なるので必ず確認しましょう。

OSパワーマネージメント

OS上からCPUの電圧や動作周波数を変化させる省電力機能もあります。**/proc/cpuinfo**や**lscpu**で動作周波数を見たときに、そのCPUモデルの動作周波数より低い値になっているようであればこの機能が有効になっています。cpufrequtilsやcpupowerutilsでこの設定を操作できます。cpufrequtilsの置き換えとして、cpupowerutilsが登場していますので、ここでは新しいcpupowerutilsを紹介します。**図1**のように利用します。デフォルトのガバナー（調整器）は[ondemand]に設定されています。[ondemand]だとCPU使用率が95%を越えると動作周波数が上がりますがそれまでは低い動作周波数での動作となります。常に100%の負荷がかかり続けるようなベンチマークツールであれば、この機能の影響は出ませんが、一定でない負荷や負荷がかかり切らないベンチマークツールでは結果が低く出てしまう可能性があります。この場合は[performance]に設定すると常に最大の動作周波数で動きます。なお、[ondemand]を選択したとき、後述のターボブーストが有効に働くとその動作周波数で動作するので、それほど性能への影響を気にする必要はなさそうです。

ちなみに執筆時点（2013年12月）でのCentOS 6.5のcpupowerutilsではドライバ未対応のため[no or unknown cpufreq driver is active on this CPU]となります。Xeon E5-2600 v2シリーズにまだ対応しておらず、最大の動作周波数で動作となります。

ターボブースト

Nehalemマイクロアーキテクチャ以降のCPUには、ターボブーストテクノロジの機能が付いています。電源や温度に余裕があるときに、負荷の高いコアの動作周波数を自動でクロックアップする機能です。たとえばXeon E5-2697 v2は、定格で2.7GHz動作ですが、1コアのみに負荷がかかっている場合は3.5GHzまで上がります。この機能はハードウェア側で自動に動作するためOS側には依存しません。BIOSやUEFIからこの機能を有効・無効に選択可能です。

OS上からターボブーストに対応しているかの確認やクロックアップされた動作周波数を確認するときにも

```
# cpupower frequency-info
……（略）……
  boost state support:
    Supported: yes
    Active: yes
```

を利用します。また、ターボブーストが効いているときの動作周波数を見るには、

```
# cpupower monitor
```

を実行したときのMperfのFreqの値で確認できます（MHz）。ただし、先述のとおりドライバが対応している必要があります。

▼図1　cpupowerコマンドの利用方法

```
# yum -y install cpupowerutils ←インストール
# cpupower frequency-info ←現在の設定を確認
# cpupower frequency-set -g performance ←常に最高クロックを維持するガバナーに変更
```

ターボブーストを有効・無効で比較した結果が**図2**です。UnixBenchを-cオプションで並列数を変えていきましたが、すべての結果でターボブースト有効のほうが高速な結果となり、ターボブーストによって性能が向上することが確認できました。Xeon E5-2697 v2では12コアすべてクロックアップが可能なため、すべての並列数にてターボブースト有効のほうが高速な結果となりました。

ハイパースレッディング

Ivy Bridgeマイクロアーキテクチャにおいても1つのコアが2つの論理的なコアとして動作するハイパースレッディングテクノロジの機能が実装されています。

BIOSやUEFIからハイパースレッディングの有効・無効が選べます。この結果の比較が**表2**です。物理コアとしては24個、ハイパースレッディングを有効にしたとき論理コアは全部で48個となるため、並列数をこれに合わせました。48並列の比較では、論理CPUの数の差が出て、ハイパースレッディング有効のほうが高速でした。物理コアが同じ24並列の比較においても結果はハイパースレッディング有効のほうが高速でした。これはUnixBenchの一部のテスト項目(たとえばshell8)では、指定した並列数以上に並行動作するためです。

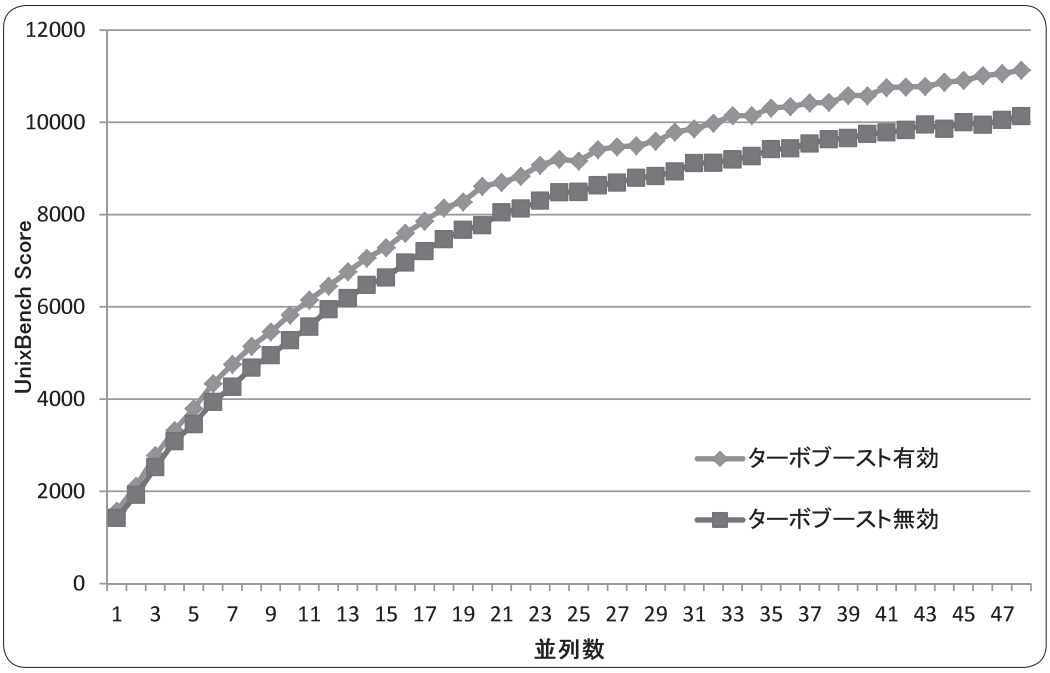
Sandy Bridgeと比較

今回、Sandy BridgeとIvy Bridgeを比較するにあたり用意できたCPUは、コア数も動作周波数も異なるスペックです。そのため、シングルスレッド性能つまり1並列における動作周波数あたりの性能を比較します。この性能指標が得られれば、動作周波数とコア数に比例して予測性能を計算できます。もちろん、キャッシュ

▼表2　ハイパースレッディングのUnixBenchスコア比較(E5-2697 v2)

	24並列	48並列
ハイパースレッディング有効	9197.9	11130.8
ハイパースレッディング無効	8513.0	10568.7

▼図2　ターボブーストのUnixBenchスコア比較(E5-2697 v2)



▼表3　Sandy BridgeとのUnixBench比較

	E5-2697 v2 (Ivy Bridge) 2.7GHz		E5-2670 (Sandy Bridge) 2.6GHz	
	計測スコア	1GHzあたりのスコア	計測スコア	1GHzあたりのスコア
1並列	1422.6	527	1323.8	509
2並列	1921.9	712	1889.8	727
3並列	2521.1	934	2603.4	1001
4並列	3081.3	1141	3017.7	1161

やバスの構成なども変わってくるため正確な情報としては使えませんが、傾向を見るには十分です。

事前に予測をたてる

実際にベンチマークを走らせる前に結果の予測を立てます。CPUの世代が変わるときに、どんなアップグレードがあるかWebから情報を収集するなどします。また、新しいCPUが出るとIntelがベンチマーク情報を公開[注2]しています。ほかにもSPECベンチマークの結果[注3]が公開されていますのでこの情報は役に立ちます。新しい機種でも早いタイミングで情報が公開されています。E5-2600シリーズ(Sandy Bridge)とE5-2600 v2シリーズ(Ivy Bridge)とで動作周波数とコア数が同じサーバで比較すると、ほとんど性能差がないことがわかります。

実際に計測し、比較

動作周波数あたりの性能を比較するにあたり、可能な限り固定された環境下で計測することが望ましく、次の条件下で比較しました。ハードウェアパワーマネージメントは最大性能を選択。OSパワーマネージメントはperformanceガバナーを選択(ドライバが対応しているE5-2670のみ実施)。ターボブーストは無効。ハイパースレッディングは有効にしました。

結果は**表3**です。シングルスレッドとマルチスレッドの性能を測定するため並列数を変え計測し、それぞれの並列数にて計測したスコアを動作周波数で割ったスコア(1GHzあたりのスコア)同士を比較します。たとえば、1並列だとE5-2697 v2は527、E5-2670は509とほぼ同じと言えます。他の並列数で実行した結果を見ても多少の計測誤差はあっても大差はなく、E5-2600シリーズ(Sandy Bridge)とE5-2600 v2シリーズ(Ivy Bridge)との動作周波数あたりの性能は同等と推測できます。今回はそれぞれUnixBenchを1回づつ計測した値ですが、複数回実行しその平均をとると誤差が小さくなりより正確な比較ができます。

次回予告

今回はUnixBenchを用いてE5-2697 v2とE5-2670比較しました。実際はUnixBenchだけを利用するのではなく、前回までに紹介したSysBenchなどでCPUやDBのベンチマークを実施するなどいくつかのツールを用いて比較していきます。結果としてはサーバ用途で考えたときにSandy BridgeとIvy Bridgeでは動作周波数あたりの性能差はないという結論から、E5-2670と動作周波数とコア数が同じであるE5-2650 v2を選択しました。なお、E5-2670とE5-2650 v2が同等性能かという答え合わせは、当社のブログ(http://www.idcf.jp/blog/)で紹介する予定ですので興味あればこちらもどうぞ。

次回はHTTPのベンチマークなどネットワークのベンチマークとチューニングをしていきます。ネットワークをチューニングして負荷をかけ切りましょう。SD

注2) URL http://www.intel.com/performance/
注3) URL http://www.spec.org/cpu2006/results/

分散データベース「未来工房」

第9回 Riakはなぜデータをなくさないのか(1)

Writer 上西 康太(うえにし こうた)　Bashoジャパン株式会社　kota@basho.com

データベースに求められる要件はさまざまだが、永続性(Durability)はあって当然だと思われている場合が多い。トランザクションのないデータベースであれば、永続性がないことを承知で使っている場合も多いが、Riakはトランザクションがないにもかかわらず「絶対にデータをなくさない」アプリケーションの設計や運用が可能になる。今月は「データをなくさない」とはどういうことなのか、それがRiakでどう実現されているかについて解説する。

注)本稿は、筆者の意向により常体で表記している。

データをなくさないとはどういうことか?

「データベースはACID特性を持っていなければならない」とはよく聞く言い回しであるが、ここではACIは措くとして、Dこと Durabilityが意味するものを考えてみる。よく直訳して永続性というが、いったん書き込んだデータは、システムが想定しているレベルの障害であれば、どれだけ起ころうとも失われないことを指すのが一般的だ。

つまり「絶対にデータをなくさない」とは、単に障害が起きてもバックアップから復旧できるという単純な理解では意味がない。そのシステムがどのような障害を想定し、どこまで対応しているかを把握し、データベースを利用するアプリケーションが必ず読み込めるようになっていなければならない。

そのうえで結論からいうと、Riak は動作を継続している限り、アプリケーションがデータの更新を知らずに上書きしてしまうということは起きない。世の中に多くあるKVS(Key Value Store)はPUT/GET/DELETEの3種類だけのものであり、memcachedやRedisなどの分散しないタイプはこれに加えてCAS(Compare and Swap)などのデータ操作APIが用意されている。CASを使えば、アトミックな更新を行うことで、書き込みの競合が起きたときに、データの更新を知らずに上書きしてしまうことを防げる(**図1**)。

データの更新を知らずに上書きしてしまうということは、誰にも知られずに捨てられた更新が存在するということである。誰にも読まれないデータは意味がなく、ここではこれも**データをなくした**ものと定義する。いわゆる"Lost udpate"(失われた更新)というものだ。これを防ぐためにはロックやCAS を用いて、並列に起きているデータの更新を排他するのが最も簡単な方法だ。

しかし、排他制御を行うということは、その瞬間にデータにアクセスできるのはたった1つのクライアントということになる。また、**排他ができないケースでは誰もデータを更新できなくなってしまう**。Pthreadプログラミングなどの経験があれば、排他制御そのものはロックを使えばある程度は簡単にできる場合が多い。しかしながら、レプリケーションを行って耐障害性を高めたい場合、問題は簡単ではなくなる。とくにロックを持ったままプロセスが停止した場合には、ほかのプロセスがタイムアウトなどでロックをとれるようにするまでそれなりに時間がかかる。この間の停止時間は、通常の死活監視システムでは数秒〜数分の間設定するものだが、その期間中に書き込みができなくなると、

可用性に影響がある。多くのデータベース製品はそのように割り切っている。

ほかにも、レプリカのないデータベースがディスクの故障などによって永久に読めなくなることもデータをなくしたといえる。

このように、「データをなくさない」という観点で見ると可用性と永続性（整合性）の間にトレードオフがあることがおわかりいただけただろうか。

Riakの話に入る前に、RDBやHBaseなどの整合性重視のシステムで可用性と整合性のトレードオフをどのように取り扱っているか見てみよう。とくにネットワーク障害、書き込みの競合、多重故障のケースなどで、データをなくしてしまういくつかのケース[注1]と、それを防ごうとして可用性を下げざるを得ないケースについて解説していきたい。

ケース1「ネットワーク分断」

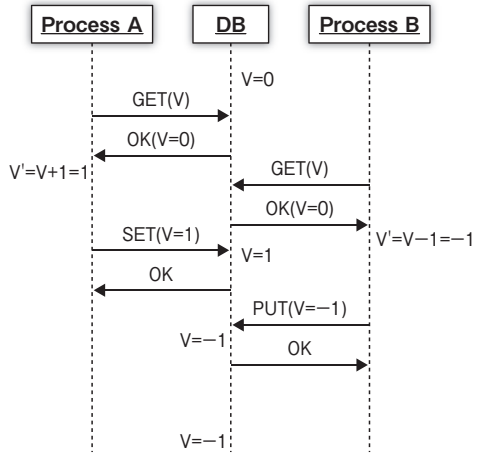
典型的なデータベースの冗長構成としては、更新リクエストを受け付けるマスターのノードと、更新情報をコピーして保存するスレーブのノードを2台用意するものがある。この場合、マスターに故障が起きて再起不能になった場合は、いったんスレーブがマスターに昇格するのが一般的だ。スレーブはマスターの故障を検知してマスターとしての動作を開始する。マスターに接続していたデータベースはスレーブへ接続しなおしてあらためてデータの更新リクエストを送り始める。アクト－スタンバイ構成と呼ぶこともある。

このような構成では、片系の故障についてはロバストに動作するが、マスターとスレーブ間の通信路に異常があった場合は動作しない[注2]。マスターからこの現象を見た場合は、スレーブが応答しないのでスレーブが故障したように見える。一方、スレーブからみるとマスターが応答していないので、マスターが故障したように見える。このときスレーブはマスターへの昇格を行いリクエストを受け付け始めるだろう。こ

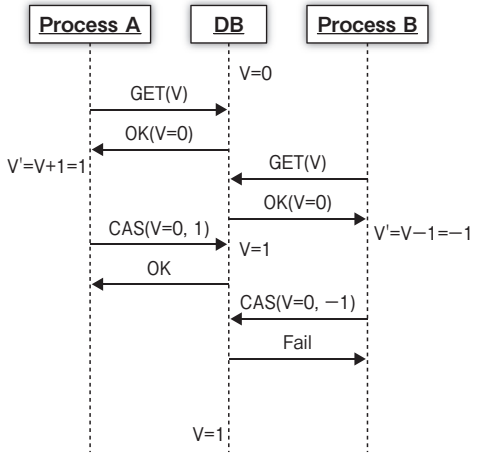
注1）"Lost update"も含む。

注2）スイッチの冗長化などで信頼性を担保するのが一般的かもしれないが、スイッチやネットワークの冗長化は、コモディティ製品では対応してない場合が多くコストがかさみがちになるだろう。

▼図1　プロセスAとプロセスBから1つの排他制御のないデータベースにread-modify-writeを行った結果



(a)複数の更新が来ると、一度データを読んでから変更した結果を再び書き込むことができず、データの更新を知らずに上書きしてしまう。



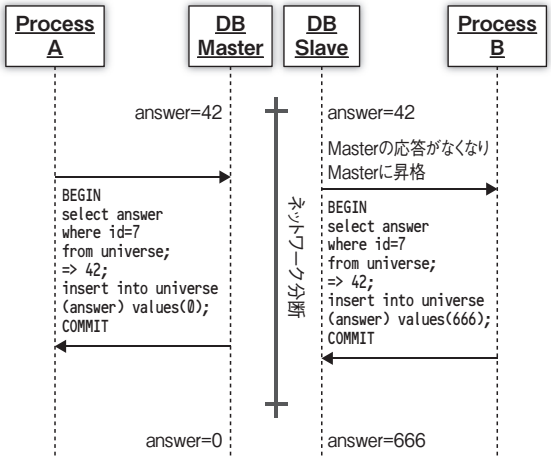
(b)CAS（Compare and Swap）があるだけでも、データの更新を安全に行うことができるようになる。

の状況を、一般的にはネットワーク分断とか、スプリットブレイン(Split Brain)と呼ぶ。これが起きてもシステムが破綻しないことを分断耐性があると呼ぶ。

このとき、同期レプリケーションを行っていた場合は、単にレプリケーションができず更新がいっさいできなくなる。つまり、更新に関してはダウンタイムとなってしまい、可用性が低下する。非同期のレプリケーションを行っていた場合は、マスターとして動作するプロセスが2ヶ所で別々に更新を受け付けてしまい、**図2**のようにデータの矛盾が起きる可能性がある。

これへの運用レベルの対処としては、手動でのマスターフェイルオーバーを行う、データの矛盾が起きたら片系のノードのデータを捨ててレプリケーションをやり直すなどが考えられるが、いずれも大規模なシステムになってくると負担が大きい。

▼図2　AとBの更新がマスターとスレーブでそれぞれ成功してしまった場合、データベースが矛盾を起こしてしまう



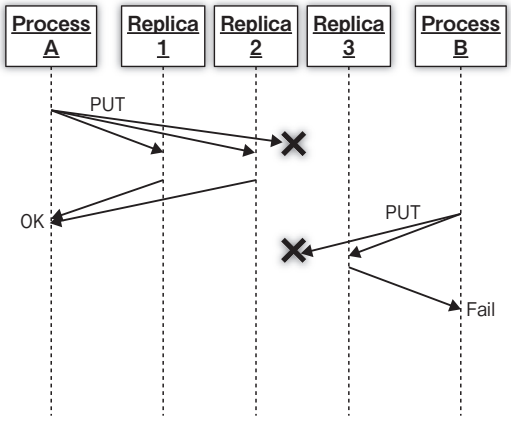
ケース2「多数決と並行書き込み」

このようなネットワーク分断の問題を解決するためには、通信ができない状態でも「どれが正しいデータか」を決める方法が必要になる。このための最も一般的な方法が多数決だ。つまり、ネットワークが分断された際も**過半数を確保したものが正しい**という約束をレプリカ間で合意しておくことによって、分断した際に過半数を確保できていない側のノードは更新が成功したとはみなさない(**図3**)。

これの最も単純なものはクォーラム(定足数)によるレプリケーションという。こうすれば、ネットワーク分断が起きていても、落ちているノードが一部あっても問題なく更新に成功する。

しかしながら、複数の更新が同時にやってきた場合は逆に問題になるケースがある。**図1**や

▼図3　クォーラム(定足数)によるレプリケーション
レプリカ数3の場合、典型的には書き込みと読み出しの定足数をそれぞれ2に設定する。この場合、2個以上のレプリカから応答があるとその操作は成功と定義される。そうでない場合は古いデータが見えてしまうケースがあるため操作を失敗とする。



▼図4　クォーラムによるレプリケーションの例外
クォーラムによるレプリケーションがあっても、複数の書き込みが同時に起こると、値の上書きによって読み出されることのない書き込みがあり得る。

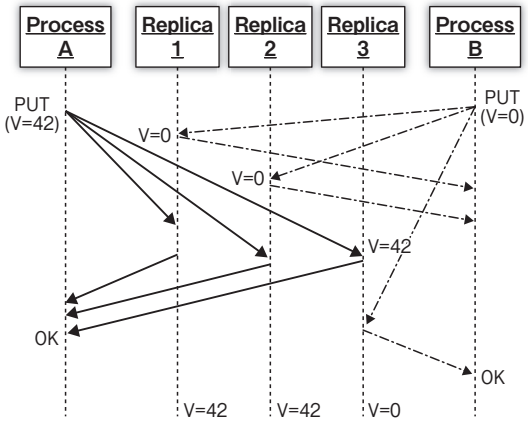


図4のように、成功したにもかかわらず一度も読まれることのないデータがあり得る。

これを可能にするためには、堂々巡りになるようだが、複数のノード間での厳密な排他制御を導入する必要がある。その難しさは先ほど説明したとおりだ。

ケース3「アトミックブロードキャスト」

このような分散環境での排他制御には、ZooKeeperのようなシステムを使った分散ロックを構築して書き込みをシリアライズすれば、確かにうまく動くだろう。しかしながら、ZooKeeperの内部で、上記のようなネットワーク分断やレイテンシ、データ永続化に関する**まったく同じ問題**を解決しているに過ぎない。

システムはコンポーネントが1つ増えると複雑さがそれだけ増える。なるべくコンポーネントを増やさず、データベースのシステムだけでそれを解決すればよいのか?——その答えの1つが、ZooKeeperの中にある。同じ問題を解いているのだから当然である。

ZooKeeperやChubbyをはじめとするこのようなシステムは、内部的にはアトミックブロードキャストという技術で実現されている。アトミックブロードキャストとは、クラスタを構成するノードとノード間の通信路がすべて非同期的で、いつでも故障し得るという前提で、クラスタを構成するノードがすべて、メッセージが欠落することなく順序保証された情報伝達を実現する通信プロトコルのことを指す。これは具体的な特定プロトコルを指すわけではなく、ブロードキャストのプロトコルやアルゴリズムが上記のような性質を満たすときにアトミックと呼ばれる。

実際にZooKeeperが採用しているアトミックブロードキャストプロトコルはZABである。ほかにもPaxosやRaftが著名であるがここで具体的に解説することはしない。重要なのは、レプリケーションされるデータがアトミックなブロードキャストによってレプリカ間で整合した状態で、多少の故障があってもシステムが詰まることなく動作可能になるということだ。

ほかの実装例としては、GoogleのChubbyや、CassandraのCAS、Riak 2.0で提供されるStrong Consistencyの機能ではPaxosが使われている。近年注目されているものとしてはetcd[注3]という類似のシステムがあるが、こちらはRaftというアルゴリズムを採用している。

しかしながら、アトミックブロードキャストの実装の多くでは、レプリカ数や、レプリカを保持するノードは固定されている場合が多い。また、多数決を使った場合に、遅いノードがいた場合でもある程度の性能を得ることはできるが、最悪のケースでは実際にレプリケーションされている数が過半数程度しかなく、あるべきレプリカの数だけデータのコピーがされていない場合がある。この点で、永続性の観点から、レプリカを保持するメンバーが固定されているアトミックブロードキャストでは運用しにくい面がある。

また、アトミックブロードキャストのプロトコルはいずれも、ノード間の通信が一往復で済まないようなものになっていて、性能を得にくい場合がある。

Riakはなぜデータをなくさないのか?

ここまで説明してきたような、データを永続化することの難しさに対してRiakではどういう解決をとっているかということをここから解説する。

Riakでは、アトミックブロードキャストを用いずに、データの永続性を担保する方法としてヒント付きハンドオフとSiblings(シブリング)という方法をとっている。

ヒント付きハンドオフは、おおざっぱにいうと、ノードが故障してデータを書き込めないときには、クラスタの別のノードにとりあえず書

注3) https://github.com/coreos/etcd

き込んでおくという動作である。これによって、データのコピーが必ず規定数だけあることを保証する。

Riakの分散のしくみ

まず、ヒント付きハンドオフを説明する前に、Riakにおけるデータの分散の方法と、書き込みが完了するまでの大まかな流れを正常系と故障時それぞれについて解説しよう。

2013年の8月号でも解説したが、Riakでは“ring”と呼ばれるハッシュ空間を等分割し、それぞれをパーティションと呼ぶ。パーティションは等分割された空間の左端のハッシュ値で表現し、そのパーティションをプライマリで担当するデータベースのフラグメントを“vnode”(ブイノード)と呼ぶ。このvnodeが、Riakでのデータ分散の基本的な単位になる。n番目のvnodeを表すハッシュ値P(n)は、分割数を64とすると、

$$P(n) = 2^{160}\frac{n}{64}$$

となる。このvnodeを各ノードに均等に配置することで全体の負荷分散となる。また、どのノードがどのvnodeを持っているかはゴシッププロコトルでに共有されている。アクセスしたいデータのキーがわかっていれば、そのデータが所属するvnodeは、

$$P(n) \leq Hash(key) < P(n+1)$$

で決まるn番目のものであることがわかる。**n_val=3**のとき[注4]、データのコピーは3つ必要なため、このキーを保存・処理する責任があるのはn、n + 1、n + 2番目のvnodeを保持しているノードである[注5]。実際のレプリケーションはこの3ノードの間で行われる。Riakは、この中からランダムに1つコーディネータを決定して、まずはそこにリクエストを転送する(**図5**)。

コーディネータがリクエストを受け取ると、そこから残りのレプリカへリクエストを転送する[注6]。転送されたリクエストを受け取った各ノードはローカルのディスクへPUTを行い、結果をコーディネータへ返送する。コーディネータは自分も含めAckが既定のW以上になったらリクエスト元へ成功を返す[注7]。ここまでの正常系の動作では、先ほど説明したクォーラムのレプリケーションとほぼ同じだ。

PUTの場合、ここでの書き込みの方法に

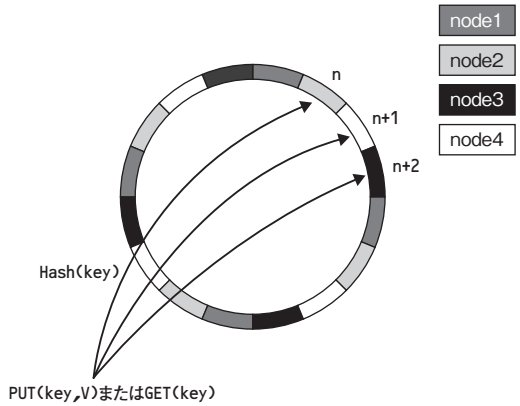
注4) これは、1つのキーと値のペアにいくつのコピーをRiakが用意するかという値である。

注5) ほとんどのケースでコピー数は3であり、Riakはそこに最適化されているが、変更することも可能。

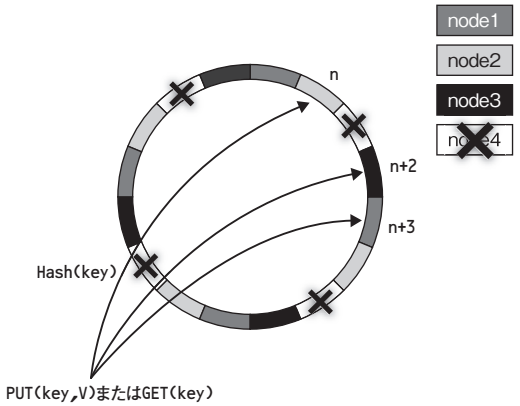
注6) GET/PUTの違いはデータをvnodeへ書き込むか読みだすかの違いであり、このレイヤーではほとんど違いがない。

注7) デフォルトではWは2である。

▼図5　Riakにおける正常系の書き込み
いずれのノードも故障していない。



▼図6　障害時の書き込み
node 4はすでに故障とマークされているため、node 1が持っているvnodeへの書き込みが行われる。



「絶対にデータがなくならない」ことを保証する若干の工夫がされているが、これは後で解説する。

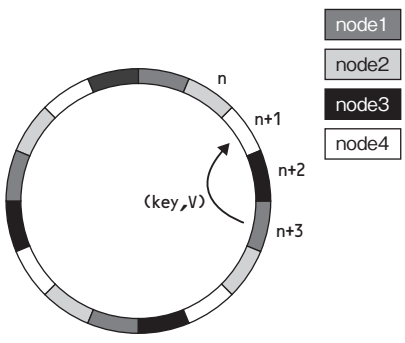
故障時の挙動

それでは、3つのレプリカのうち1つが故障していて、アクセスできない状態であった場合にはレプリケーションはどういう挙動を示すだろうか。

Riakのクラスタを構成するノードはお互いの死活監視を行っているため、アクセスできないノードはあらかじめわかっている。PUTやGETのリクエストが来た時、そのデータを保存すべきvnodeを持っているノードが故障している場合は**図6**のように、リストの次にいるノードへリクエストをフォールバックする。たとえばn + 1のvnodeを保持しているノードが故障していてアクセスできないことがわかっている場合には、このリクエストを処理する責任があるvnodeはn、n + 2、n + 3番目のvnodeとなり、正常系と同様に実際にリクエストが転送される。このように、リストの次にいるノードがリクエストを処理することは、“Hinted Handoff”(ヒント付きハンドオフ[注8])と呼ばれる。

注8) Hinted Handoff:ハンドオフとは「手渡しする」という意味。翻訳するとヒントに従って手渡しするしくみ、といったところだろう。

▼図7 障害からの自動復旧
node 4の復旧とともに、node 1へ一時的に書き込まれたデータがnode 4へ返却される。



ヒント付きハンドオフでは、n + 3番目のvnodeを保持しているノードに渡す。そのときテンポラリにvnodeのファイルを作成して、そこに書き込みを行う。

復旧後の挙動

故障から復帰したときも、ノードがふたたびクラスタに登場したことをフックして、復帰したノードが持っているvnodeの一覧を、自分が持っているものと比較する。もしもヒント付きハンドオフにより、復帰したノードに属するvnodeが持つべきデータを持っている場合は、vnodeを所有しているノードへ**図7**のようにデータを返却する。この様子はriak-admin transfersというコマンドで監視することができる。データの返却が終わると、ヒント付きハンドオフで一時的に保存していたデータを自動的に削除するようになっている。

このように、本来あるべきノードにアクセスできない場合にはヒント付きハンドオフで代替ノードを用意し、そこに一時的に書き込むことによって、正常系、故障時、異常時のいずれにおいてもデータのコピーが3つあることを保証する。

ネットワーク分断が起きても同様に、アクセス可能なノードの中でnの次にあるvnodeを持っているノードを2つ探して、代替ノードとしてそこにPUTやGETのリクエストを転送する。Riakではアクセスできないノードに対して必ず代替ノードを用意しておいて、復旧後に本来持つべきノードへデータを返却することで、あるキーに関するデータを持っているノードが必ず3台いる状態を維持する。

もっと極端な例を解説しよう。**図6**のケースをさらに発展させて、node 1、2、3がすべて停止しているケースを考えるとRiakの可用性がわかりやすい。このようなケースでは、vnodeのn、n + 1、n + 2が使えず、n + 3だけが利用可能だ。しかしながら、もう2ヵ所書きこむべきで、そのvnodeはn + 4以降で生きて

いるノードから順番に選ばれる。このケースではおそらくn＋4からn＋6も同様に生きていないだろうから、n＋7とn＋11と、さらに後ろから選ばれる。この例では4台しかないが、10台などノード数がもっと多いケースであれば別のノードにコピーされることになるだろう。このように、システムが部分的に大きく欠けていたとしても、残っているノードでなんとか書き込みを受け付けるようになっており、これはメンバーが固定されたアトミックブロードキャストでは難しいところである。

だからこそ、Riakは常にデータをなくさないし、いつでも書き込むことができるのである。

「とりあえず書き込む」ことの代償

Riakの、いわば「とりあえず誰でもいいからアクセスする」というしくみやハンドオフによって、同じキーに属するデータのコピー数が3つあることが保証されていることは説明したが、上記で説明した、いったん書き込みに成功したデータが読まれることなく上書きされるケースについては、まだ防げていない。このままでは書き込まれたはずのデータが失われてしまう。

また、ヒント付きハンドオフによっていったんほかのノードに書き込まれたデータが返されるときに、もともとの古いデータが残っていた場合に、タイムスタンプ等で上書きしてしまっては同様にデータが失われてしまう可能性がある。

これを防ぐしくみが、Siblingsとベクタークロックである。この2つのしくみとPUT時の“read-modify-write”を徹底することによって、書き込みに成功したデータをアプリケーションが無視することのないしくみを作ることができる。

まとめ

ここでは、データベースの永続性とは何か、データがなくならないとはどういうことかを、読まれることのない更新が起きないことと定義した。単なる上書きでは失われてしまう更新があること、それを避けるためにトランザクションなどを用いた排他制御のしくみがあるとよいが、システムを冗長化するとネットワーク分断の耐性を得ることが難しくなる。

それを解決するためにアトミックブロードキャストのプロトコルを実装するのが1つの解ではある。しかしRiakではヒント付きハンドオフとそのデータの返却をすることで、どのような故障があっても「とりあえず既定の数のコピー数を確保する」しくみになっていることを説明した。

しかし、とりあえず書きこむというだけでは、データの上書きや整合性がまだ担保されているとは言いがたく、どのように「絶対にデータをなくさない」ことが保証されているかがわからないことと思う。RiakではSiblingsとベクタークロックを用いてそれを解決するが、長くなったので続きは次号に譲ることとしたい。SD

コラム　連続する番号を持つvnodeは必ず別のノードに保存されるのか?

そこがRiakの賢いところで、すべての隣り合うvnodeおよび2つ隣のvnodeは別のノードに保存されるようにノード配置を計算している。単純に並べるだけだと、ノード追加時の再配置のトラフィックが非常に大きいので、ヒューリスティックに並べる順番を工夫しており直感的な順番ではないが、**n_val=3**のときにこの条件を満たすためには必ず5ノード以上必要になる。このようにハンドオフを正常に動作させるために、Riakの最低稼働台数は5台からということになっている。もちろん、データのコピーを2ヵ所に分けて保存したい(2ノードで動かしたい)ということであれば2台でも動作するが、このようなヒント付きハンドオフのしくみがうまく動作しないため、5台以上でのクラスタ構成を推奨している。

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

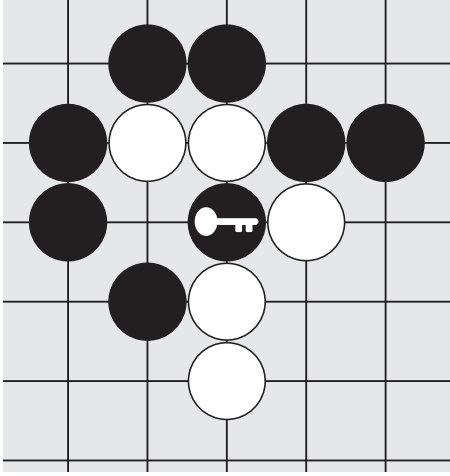
第3回 幸せを運ぶコマンド

/から消したらどうなるんだろう、という好奇心は大切だと思います。他人に迷惑かけない程度でその探求心を満たしてほしいものです。仮想環境も手軽に用意できる昨今はこのような行為も手軽に試せていいのですが、それが重要な開発環境だったり、ファイル共有マウントしてると大惨事になります。「素振り」はしっかりしましょう。

『Linuxコマンドポケットリファレンス』改訂版の出版も決まり、さらに絶好調のくつな先生に愛の手紙を！

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第九回】SSHが危険に晒されるとき

昨年、日本のスーパーコンピュータを保有する研究所や大学の施設に対し、SSH経由で不正アクセスが行われたという報道がありました。SSH(Secure Shell)は安全に遠隔でシェルを使うためのプログラムです。しかし、どんなものでも当たり前ですが、正しい使い方をしなければ安全ではありません。そこで今回は、SSHの安全な使い方について考察します。

SSHについて

ネットワーク端末telnet

今のインターネットと呼ばれるネットワークの最初の実験は1969年に行われました。実験内容はカルフォルニア大学ロサンゼルス校(UCLA)、スタンフォード研究所(SRI)、カリフォルニア大学サンタバーバラ校(UCSB)、ユタ大学(The U)の4つのサイトで通信をすることでした。そのネットワークを経由してリモートのコンピュータにログインするために作られたネットワーク端末ソフトウェアがtelnetです[注1]。

telnetはTCP/IPで通信し、リモートのコンピュータにログインするために古くから使われているプログラムで、ネットワークで接続されたコンピュータをまるでシリアルラインで接続した端末のように使えます。ポートを指定してネットワークからサーバに対してキーボード入力することもできます。

```
% telnet www.sample.com http
GET / ←キーボードから入力
ここに Web サーバからのレスポンスが表示される
```

また、rloginは1983年に4.2BSDの上に作られます。これはBSDにTCP/IPが実装される際に作られたUNIXのためのリモートログイン機能で遠隔地のUNIXにログインします。rloginの最初のRFCはRFC1282として確認できます。

rshはリモートシェルで遠隔地のUNIX上でシェルを実行します。

```
% rsh example.com date
Fri Jan 17 03:36:22 JST 2014
```

これらは非常に便利なのですが、1つ大きな問題があります。それは、暗号化などの対策は何もせずに通信をしているため、そこで送られたデータがすべて見えてしまうことです。たとえばWireshark[注2]のようなネットワークをモニタリングするツールを使い、telnetの通信をモニタリングすればログインのときに入力するユーザIDもパスワードも全部見えてしまいます。

rloginが作られた頃は牧歌的な時代だったのですが、90年代においては極めて「危険」なプログラムと指摘されるまでになってしまいました[注3]。

注1) telnetの一番最初のドキュメントはRFC25で、これは1969年9月25日にユタ大学のC.Stephen Carr氏によって書かれています。最初の通信は1969年10月29日にUCLAからシリコンバレーにあるSRIまでを結んだものでした。最初に行ったテストはtelnetを使ってインターネット経由でコンピュータにログインすることでした。telnetはインターネット最古のプログラムの1つと言えるのです。

注2) **http://www.wireshark.org/**

注3) Lawrence R.Rogers, "rlogin(1): The Untold Story" ,1998 **http://www.cert.org/archive/pdf/98tr017.pdf**

SSH

SSHはSecure Shellの名前から想像するとおりにtelnet、rlogin、rshを代替するために作られたプログラムです。1995年、ヘルシンキ工科大学のTatu Ylonen氏が開発したものです。学内のネットワークにおいて通信している最中に、パスワードを盗聴されないようにするのが最初の目的でした。無料で配布したので[注4]、瞬く間に広がり使われるようになりました。氏は後にSSH Communications Securityという会社を設立します。

長い間、デファクトスタンダートとしてオリジナルのSSHを実装の基準にしていたのですが、やっと2006年にRFC4253が発行されます。初期のプロトコル設計には問題があり、それを回避するためにRFC4253では以前のプロトコルとは互換性のないものとして作られます。RFC4253からのプロトコルはSSH-2と呼ばれ、それまでのプロトコルはSSH-1と呼ばれ区別されるようになりました。

現在は、いくつかのサーバ実装があります。一番多く使われているのは、おそらくOpenSSHでしょう。OpenSSHはセキュリティには定評のあるOpenBSDプロジェクトから出てきた実装です。フリーソフトウェアではGNUプロジェクトでもlshというSSH-2実装があります。しかし、開発はすでに不活発でDebianプロジェクトでメンテナンスしているだけになっていますので、利用はお勧めしません。また、これら以外に商用のサーバもいくつかあります。

一方で、クライアントはたくさんあります。Windows上、Mac上といろいろと存在しています。

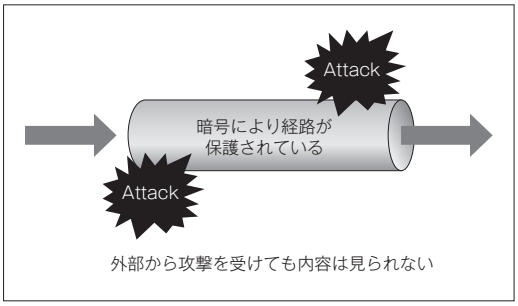
SSHの保護モデル

SSHのモデルは通信路を暗号化し、通信している内容を保護するというものです。データは暗号化されているため、外部からは内容を見られません（**図1**）。よってパスワードなどの情報を入力したとしても外部から覗き見ることはできません。

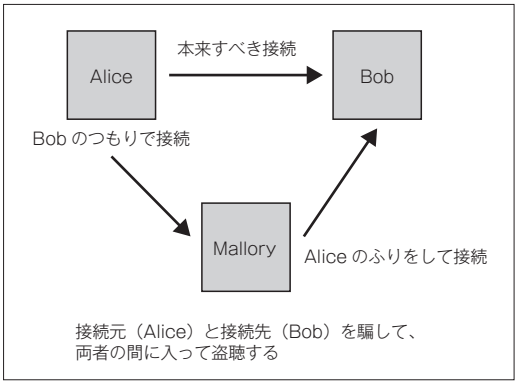
接続先とはディフィー・ヘルマン鍵共有アルゴリズム、またはRSA暗号アルゴリズムを使ってデータ通信時に使う共通鍵暗号の鍵を交換します。ディフィー・ヘルマン鍵共有アルゴリズムも、RSA暗号アルゴリズムの鍵共有も、相手を確認しない、つまり電子署名にあたるような相手の確認ができないので、そのままではMAN-IN-THE-MIDDLE攻撃（**図2**）をされた場合、内容を盗聴されてしまいます。

SSHのサーバ認証は、通常、最初に接続した際にサーバから署名を検証するための検証鍵をもらい、それをユーザが確認しクライアント側に登録しておきます。次回以降は、その検証鍵を使って接続先サーバが以前と同じものであるかどうかを検証します。同じものでない場合、ユーザに警告メッセージを出したうえで接続を中断します。このようにして騙されることを阻止します。

◆図1　通信路保護のモデル



◆図2　MAN-IN-THE-MIDDLE攻撃（中間者攻撃）



注4）筆者の記憶に頼るのですが、このときのライセンスは独自のライセンスで、今で言うオープンソースにも当てはまらないし、もちろんGPLとも互換ではありませんでした。

◉OpenSSHサーバについて

手元にあるDebianとCentOSでOpenSSHのバージョンの違いをチェックしてみました。Debian 7.1.0ではOpenSSH_6.0を採用し、CentOS 6.5ではOpenSSH_5.3を採用していました。鍵交換のアルゴリズムの組み合わせ（Cipher Suite）は次のとおりです。

・楕円曲線ディフィー・ヘルマン鍵共有 ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521
・ディフィー・ヘルマン鍵共有 diffie-hellman-group-exchange-sha256,diffie-hellman-group-exchange-sha1, diffie-hellman-group14-sha1,diffie-hellman-group1-sha1

ユーザの接続認証

SSHはサーバ側のシェルを動かします。その際に、ユーザ認証をする方法は2つあります。1つはパスワード認証、もう1つは公開鍵認証です。

パスワード認証は通常のUNIXのパスワード認証と基本的に同じです。UNIXのシステムが持っているメカニズムを呼び出しています。ですからGNU/LinuxでLinux-PAM[注5]を使っている場合、Linux-PAMの機能と連動できます。sshdの設定に関しては/etc/pam.d/sshdで行います。

最近のLinux-PAMはSELinuxの機能と連動することも可能で、SELinuxがenforcing（有効）のときのみユーザがシステムにログインできるなど、これまでのUNIXのセキュリティよりもレベルを高くすることが可能です。

もう1つは公開鍵認証です。それまでのリモート端末のソフトウェアにはなかった機能で、この機能のおかげで、パスワード方式よりも一歩進んだ安全性を確保できます。

電子署名を使うユーザ認証（公開鍵認証）

まず、ssh-keygenコマンドを使って署名鍵／検証鍵のペアを作ります。次に署名鍵（秘匿鍵）を手元のクライアント側におき、検証鍵（公開鍵）をサーバ側におきます。sshで接続する際は手元のクライアントにおいてある署名鍵を使い電子署名し、それをサーバ側に送ります。サーバ側は検証鍵を使い正しい相手であるかどうかを確認します。

図3では2,048ビット長のRSA暗号の鍵ペアが作られます。sample-id-rsaに署名鍵が入っておりsample-id-rsa.pubに検証鍵が入っています。電子署名アルゴリズムはDSA-1024（現在では鍵長が十分ではありません）やECDSA（楕円曲線DSAアルゴリズム、ECDSA 256）も使えます。sample-id-rsaはパスワードで暗号化されており、第三者がsample-id-rsaを盗んで使おうとしてもパスワードを突破する必要があります。

検証鍵の扱い

サーバにログインするために先ほど作った検証鍵（拡張子が.pubのファイル）の内容をログイン先の$HOME/.ssh/authorized_keysに入れることによって使えるようになります。

また大規模なサイトではユーザ管理をLDAPで実施しているようなところもあるでしょう。そのようなサイトでは、SSHの検証鍵をLDAPで管理することも可能です[注6]。

SSHから侵入されるケース

さて、ここまではSSHの全体像を理解するための説明でしたが、ここからは最初のテーマ、SSHから侵入されてしまったケースについて考えていきます。特定のケースのみを詳細にレポートするわけではなく、SSHならばこのような脅威が考えられるというように、広く考えていきたいと思います。

注5）ほとんどのディストリビューションが採用しているパスワード認証のフレームワークです。
注6）「そろそろLDAPにしてみないか？ 第6回 OpenSSHの公開鍵をLDAPで管理」http://gihyo.jp/admin/serial/01/ldap/0006?ard=1389964348

◆図3　署名鍵／検証鍵のペアファイルの作り方

```
% ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hironobu/.ssh/id_rsa):
sample-id-rsa   ←署名鍵／検証鍵のファイル名を入力
Enter passphrase (empty for no passphrase):   ←パスワードを入力（エコーバックなし）
Enter same passphrase again:   ←再度パスワードを入力
The key fingerprint is:
94:df:19:1a:1c:19:02:09:01:38:8e:aa:47:ff:a9:ba hironobu@sample.com
The key's randomart image is:
+--[ RSA 2048]----+
| ...oo.o. oo     |
|o     .  +..     |
|o.      o o .    |
|..     . . + o   |
|.       S o o    |
|. .              |
|.. .             |
|. . .  .         |
| .Eo.oo          |
+-----------------+

% ls -l sample-id-rsa*
-rw------- 1 hironobu hironobu 1766 Jan 17 21:37 sample-id-rsa   ←署名鍵
-rw-r--r-- 1 hironobu hironobu  394 Jan 17 21:37 sample-id-rsa.pub   ←検証鍵
```

パスワードクラック

電子署名によるユーザ認証を使わず、古典的なパスワードを使っている場合です。この連載で何度も繰り返しているように、ユーザの一定数は弱いパスワードを使っています。よって利用ユーザのリストが手に入るならば、SSHに接続し、可能性の高いパスワードを次々に試していけばいいだけです。一定の確率でパスワードの弱いユーザのアカウントを見つけられます[注7]。

では、なぜ、ユーザ名がわかるのでしょうか。それは組織内で共通のアカウント名を使う運用になっているところがほとんどだからです。たとえばlabo.example.orgという大きな研究組織があったとします。そこでhironobu@labo.example.orgというメールアドレスを使っていると、いつしか大量のスパムが届くようになるのは、私たちが日頃、経験していることです。スパムリスト業者にはどんどんlabo.example.orgのユーザ名が蓄積されていきます。

次に、ユーザhironobu@labo.example.orgは、この組織のどのシステムでも共通認証基盤を使ってログインできるようになっている可能性が高いです。とくに大学などはどこでも同じ環境を使えるように認証基盤が便利にできています。そのような環境では学内Webサーバへログインするのも、学事カレンダーにアクセスするのも、そしてスーパーコンピュータを使うためのゲートウェイも同じパスワードが連動していたりします。

パスワードが漏れるリスクはたくさんあります。使っているクライアントにマルウェアが感染してパスワードを盗まれる場合、ほかのWebサービスなどにも使いまわしているパスワードがWebサービス側の不手際で漏れる場合、最近ではスマートフォン経由で漏れる可能性も考えられます。

漏れなくともパスワードの辞書攻撃で使われるようなキーワード／パスワードを使っている場合は、簡単にわかってしまいます。このようなブルートフォース系の攻撃を回避する定番はiptablesを使うことです。一定時間に接続してくる回数を見て、その接続を無効にすることです。完全なコードではあり

注7）デフォルトではrootはログインできない設定になっていますし、これを意図的にONにしてセキュリティのリスクを負ってまで運用する特殊なケースはまれですから、ちょっとrootのケースは横に置きます。

ませんが、リスト1のようなiptablesの使い方がヒントになるのではないかと思います。

電子署名で使う鍵が盗まれる

パスワードには問題があるので、電子署名方式（公開鍵認証）でのみSSHでの接続を許す運用をしている環境も多くあります。パスワード方式よりは安全に運用できます。ただし、鍵の管理をきちんとしているという前提が必要です。

ssh-keygenでパスワードを入力しなければ、署名鍵にパスワードはかかりません。つまり「裸」の署名鍵を作れます。ネットで検索するとその説明が大量に見つかります。パスワードがなぜ必要なのかということを無視している文章は多いです。「パスワード入力の手間が省けるので作業がはかどる」といった趣旨の内容を書いているサイトもあります。

とくにPC上でのSSHクライアントの場合「クリックだけで接続できるほうがユーザフレンドリー」のような書き方をしているサイトもあり、それを読むと「PCユーザの感覚ではそれが当たり前なのかもしれない」と問題の根深さを感じざるを得ません。

そうでなくとも、バッチ的なスクリプト処理を書くときに、どうしても処理上、パスワードなしにしたくなることがあります。パスワードなしの署名鍵の危険性を理解したうえで使うという場面があるのは否定しません。しかし、リスクを理解せずに使うのは極めて危険です。けっしてお勧めできません。

盗む方法は多くの場合、マルウェア感染でしょう。偶然にマルウェアに感染してしまうか、あるいはターゲット型攻撃で狙われるかはわかりません。唯一言えることは、そのマルウェアはSSHクライアントアプリケーションの設定情報（プロファイル）や署名鍵があれば、真っ先に外部に流出させるでしょう。そして、どこのサイトのどのアカウントに使う署名鍵なのかわかっていますから、それを使わ れたらいともやすやすと侵入されてしまいます。

一度流出してしまえば、その署名鍵でさえ守られているのはパスワードです。そして、パスワードである以上、ブルートフォースをかけてしまえば破られる確率は高いのです。ただし、マルウェアに感染して、署名鍵を盗まれ、ブルートフォースされて署名鍵を使われ始めるまでには少しだけ時間を稼ぐことができます。その間に、署名鍵が盗まれたことに気づき、対処できればどうにかできるかもしれません。このように、どんな場合にもリスクはあるのです。

ホームディレクトリのネットワーク共有

大量のハードウェアがあり、ユーザがどこにログインしてもかまわないような環境を構築しているサイトでは、ユーザのホームディレクトリはネットワークファイルシステムの環境で提供しているケースを多く見かけます。

この場合、自分のホームディレクトリ$HOME/.sshの下にあるファイルは自分の使っているマシン以外にファイルサーバのroot管理者も同様にアクセスできます。ですからネットワークファイルシステムでホームディレクトリを共有するときは、$HOME/.sshにアクセスできるユーザ範囲（権限範囲）が自分が思うよりも大きい場合がありますので十分に注意してください。署名鍵や検証鍵を書き換えられたり、あるいは意図しない検証鍵が追加され外部から侵入されたりしないよう注意が必要です。

LDAP管理

SSHの検証鍵をLDAPで管理して、LDAPで認証管理しているどのマシンからでも使えるようにしている場合、LDAPサーバに不正侵入があったり、あるいはLDAPのほうの情報が不正書き換えされたりすると、何でもできてしまいます。たとえばスーパーコンピュータへのアクセスゲートとなっているマシンへのリモートログインを守るには、スー

◆リスト1　ブルートフォース攻撃を回避するiptablesの設定

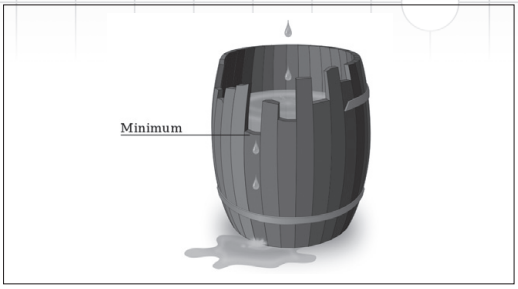
```
iptables -A INPUT -p tcp --syn --dport 22 -m state --state NEW -m recent --set
iptables -A INPUT -p tcp --syn --dport 22 -m state --state NEW -m ↵
recent --update --seconds 60  --hitcount 8 -j DROP    ← 60秒以内に8回以上の接続があれば制限する
```

パーコンピュータへのアクセスゲートを集中的に守るだけでは不十分で、認証サーバであるLDAPも同じレベルで守らなければ意味がありません。

このようにどこか一部だけでも弱い部分があると全体のセキュリティレベルが、そのセキュリティの弱い部分と同じになるということを弱い輪から切れるチェーンにたとえて“The Weakest Link”と言ったり、桶の板が一番低い部分以上に水がたまらない“Barrel Theory”（**図4**）と言ったりします[注8]。

どんなにお金をかけて一部分のセキュリティを高くしてもダメで、全体のセキュリティをバランスよく向上していかなければ意味をなさないのです。

◆**図4** Barrel Theory
（出典：http://en.wikipedia.org/wiki/Liebig's_law_of_the_minimum）



SSHの未知の脆弱性への攻撃

SSHには頻繁にいろいろな地域から攻撃がきます。それは攻撃が成功したならば、システムにログインすることに直結するからです。そして未知の脆弱性に対しゼロデイ攻撃がしかけられた場合、直接的な対応はたいへん難しい、むしろないと言ったほうが良いと思います。

そこで少しでもリスク軽減をするために、関連する環境を整備することを検討してください。まず接続先が決まっているならばiptablesなどを使い、厳密にアクセスできるIPアドレス空間を絞ってください。また限定的なユーザだけが使うように明示的に設定しましょう。そして、標準ポート番号22から別のポート番号に変更しても問題とならないときは、ポート番号を変更することを検討してみてください。関係者しか知らない番号に移行するだけで、そのようなランダムで無用な攻撃を受けなくなります。

SSHをもう一度確認してみよう

ここまでの議論を整理します。

パスワード認証はなるべく使わないようにし、デフォルトは電子署名認証方式にしましょう。ただし、どうしても検証鍵をユーザ自身で設定する必要があるなら、パスワード認証を残しておく必要があるかもしれません。その場合は、Linux-PAMと連動させてパスワード失敗時の再チャレンジ回数を極端に小さくし、失敗時の復帰時間間隔を極端に長くするといった方法も検討してください。

パスワードを設定していない署名鍵を使うのはご法度です。これを必要とする場合は、極めて特殊な運用です。

ネットワークファイルシステムでのホームディレクトリの共有は注意しなければいけません。当初予定されていなかった自分の管理の届かないところで鍵情報の流出や書き換えが起こる可能性を考慮しなければなりません。

SSHが安全なのではない

SSHはオリジナルのコードからRFCのSSHに移行する際に、多くの時間をかけて議論をしました。安全性を確保する／検討するというのは、非常にたいへんです。時間のかかる作業を行い、ついにはSSH-1から互換性の取れないレベルのSSH-2へと変貌しました。完璧ではありませんが、信頼のおけるレベルであり、現在も改良／改善が続けられています。

結局、最後は使う側の問題で、安全か否かが変わってきます。SSHを使うから安全なのではありません、正しくSSHを使えて初めて安全になるのです。SD

注8）日本では「リービッヒの最小律」「ドベネックの桶」と呼ばれる場合が多いです。

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11回

多言語ローカライズで世界に通用するアプリを目指そう!

GimmiQ(ギミック;いたのくまんぼう&リオ・リーバス)
URL http://ninebonz.net/
URL http://www.studioloupe.com/
イラスト●中川 悠京

プログラミングをしたことのない方にもアプリを作る楽しさを味わってもらいたい本連載。今回はこれまで作ってきたブックアプリを多言語対応にする方法を解説します。

電子書籍コンテンツもアプリにするわけ

これまでの連載でも何度か言ってきましたが、アプリと電子書籍とではできることの幅が大きく異なります。アプリとして作ればかなり自由なことができますが、電子書籍では多くの制限が現在ではついてしまいます。中でもアプリで作る大きなメリットの1つは、簡単に「多言語ローカライズ」ができることでしょう。

電子書籍では言語の自動切り替えが簡単にできないため、1つ1つの言語用に別々のパッケージを用意するケースが多いです。たとえば現在iBooks Storeで無料配信されているビートルズのインタラクティブ絵本『Yellow Submarine』も言語ごとに別々の本として用意されています(**図1**)。

しかし、これでは対応言語が多ければ多いほどパッケージが増え、作業工数や管理の手間、審査の回数などが増えることになります。アプ

▼図1　言語ごとに販売される電子書籍の例

リであれば1つのパッケージの中に対応したい言語をいくらでも詰め込むことができ、端末の言語設定に応じて自動的に適切な言語に切り替わるようにできるというメリットがあります。

一応、電子書籍でも1つの本の中にいくつもの言語のバージョンを入れることは不可能ではありません。たとえばページの左側は英語、右側は日本語、といった形式や、最初の50ページが英語、51ページ目から日本語版、101ページからフランス語、などといった方法で無理矢理詰め込むことは可能です。しかしそれはあまりスマートな方法とは言えませんし、この手法で多言語対応している電子書籍は一般的ではありません。

アプリの多言語ローカライズのメリット

今やアプリは世界中で配信することが可能です。しかし、日本語のみに対応したアプリが他国でダウンロードされるかと言えばそう簡単にはいきません。最低限、英語には対応しなければ、アジア圏外の国々で注目されることは難しいでしょう。ローカライズはアプリをより多くの人に届けるためにはとても重要と言えます。

そこで今回はXcodeを使って、アプリを多言語ローカライズ化する方法を紹介します。ローカライズの方法はいくつかありますが、今回はStoryboard上の文字列だけを手っ取り早くローカライズするシンプルな方法を紹介したいと思います。

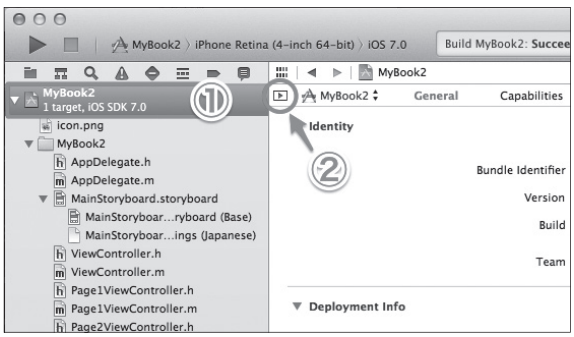
多言語ローカライズ

ステップ1

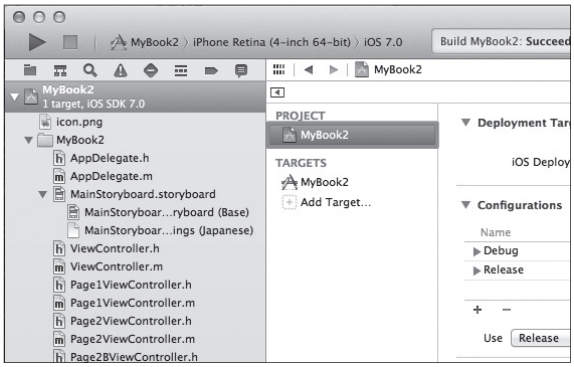
まずはXcodeでプロジェクトを開きましょう。Xcode左側のProject Navigatorからプロジェクトファイル(作例では「MyBook2」)を選択し(**図step1-1の1**)、右側に現れる「四角の中に三角が入ったボタン」をクリックします(**図step1-1の2**)。

図step1-2のように「PROJECT」と「TARGETS」のリストが現れるので、「PROJECT」の下のプロジェクト名(作例では「MyBook2」)の部分を選択してください。

step1-1



step1-2

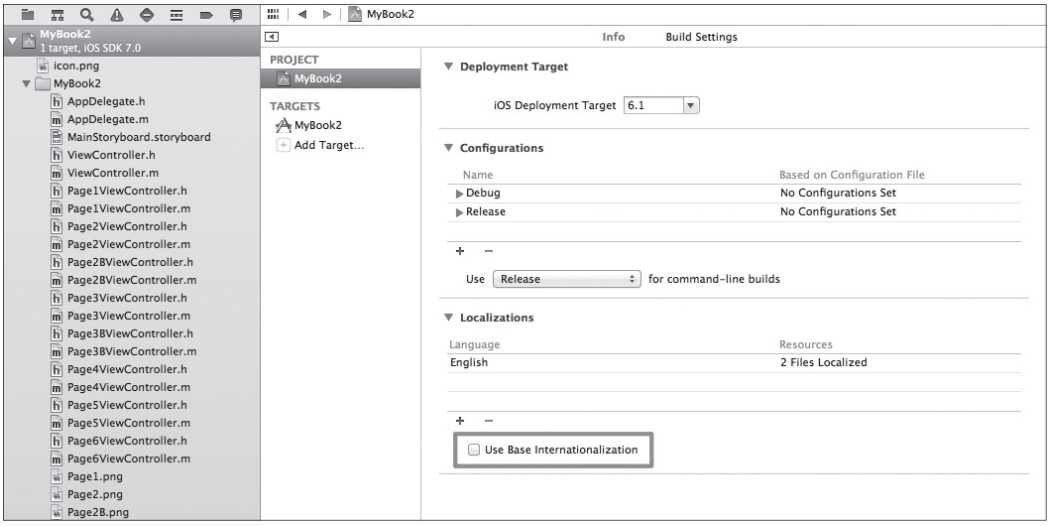


ステップ2

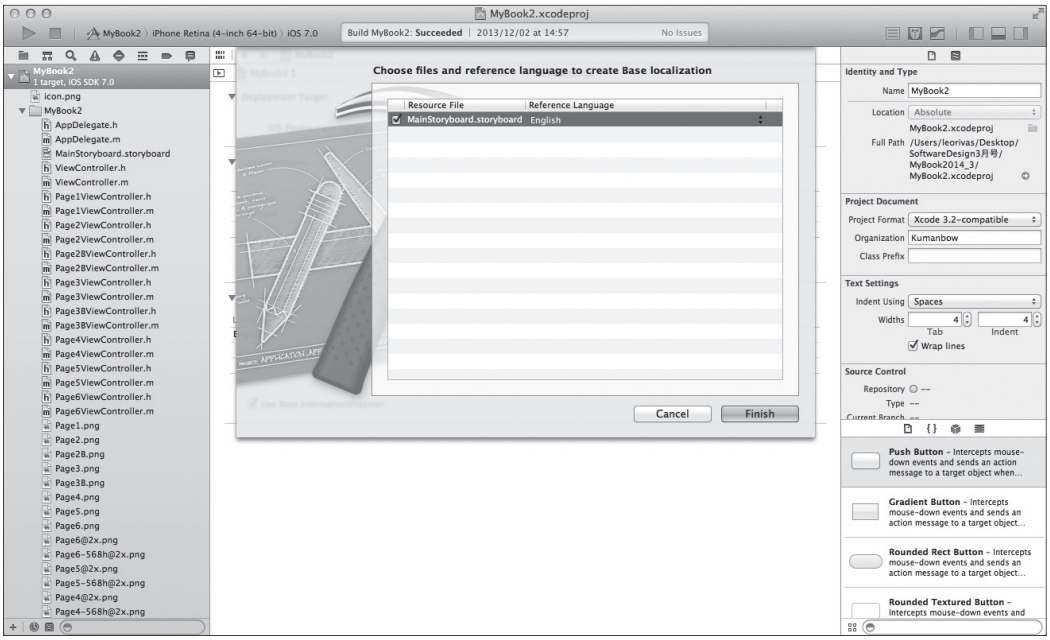
PROJECT情報の下のほうに、「Localizations」という項目があるので、その中の一番最後にある「Use Base Internationalization」と書かれたチェックボックスにチェックを入れます（**図step2-1**）。

ダイアログが現れるので、そのまま[Finish]を選択しましょう（**図step2-2**）。

step2-1



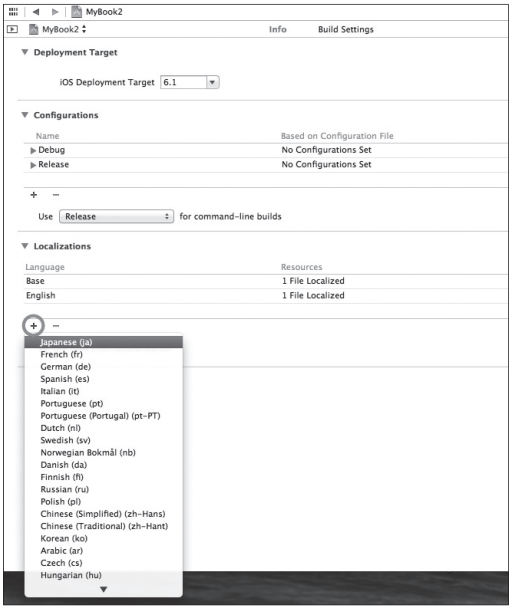
step2-2



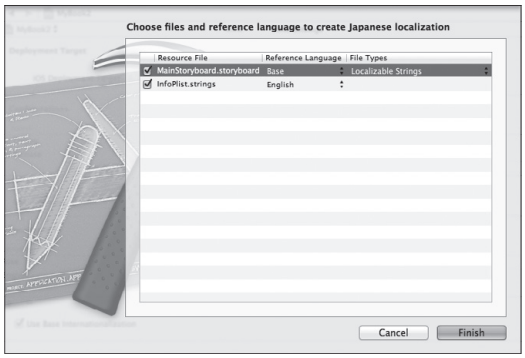
ステップ3

次に、**図step3-1**のようにチェックボックスの上にある「+」マークを押し、選択肢の中から「Japanese (ja)」を選んでください。するとダイアログが現れるので、そのまま[Finish]を押してください(**図step3-2**)。

step3-1



step3-2



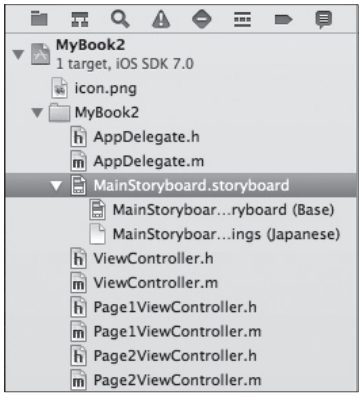
ステップ4

Localizationsに新しく「日本語」を追加したことで、左ペインの「MainStoryboard.storyboard」に新しく「▼」マークが追加されます。それをクリックすると、下に「MainStoryboard.storyboard (Base)」と「Main Storyboard.strings (Japanese)」が現れます(**図step4-1**)。

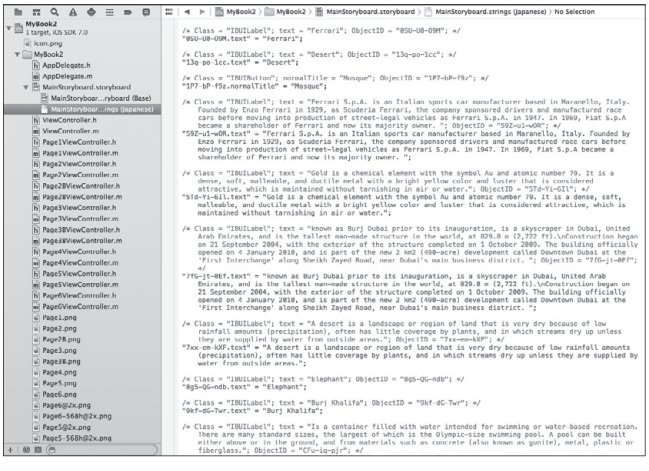
「MainStoryboard.strings (Japanese)」を選択してみてください。一見複雑そうな文字列がずらっと現れるはずです(作例では**図step4-2**のような感じです)。これはStoryboard上のあらゆる文字列を自動で摘出したものです。ここで行わなければいけないことは至って単純な作業で、「=」の後につづく「"」と「"」の間の文字をすべて日本語に置き換えるだけです(手間はかかりますが……)。作例では**図step4-3**のようになりました。

すべての文字列の変換作業が終われば、日本語のローカライズが完成したことになります。ステップ3とステップ4を繰り返せば、他の言語にも対応させることが可能です。

step4-1



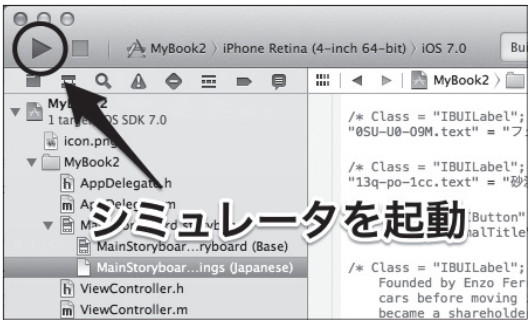
step4-2



step4-3

ステップ5

step5-1

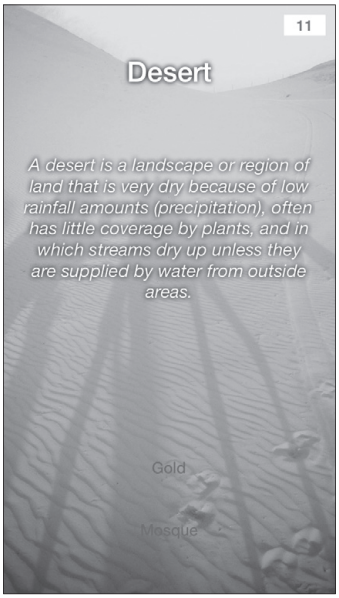


では、実際にシミュレータで日本語と英語が切り替わるかどうかの確認をしてみましょう。Xcode左上の(再生ボタンのような)ボタンをクリックしてください(**図step5-1**)。もしもシミュレータが初期設定のままであれば、恐らく英語設定として起動されるはずです(**図step5-2**)。

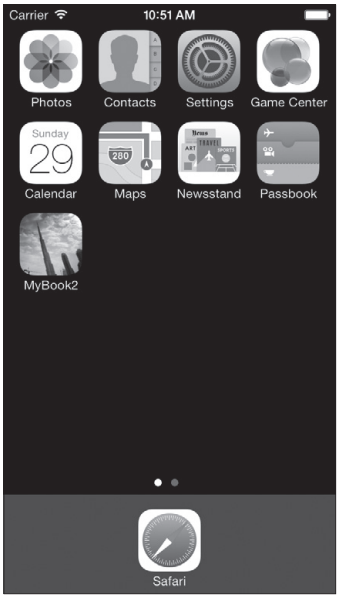
言語を変更するためには、キーボードで shift ＋ command ＋ H を入力しましょう。これでシミュレータ上のホーム画面に戻れます(**図step5-3**)。「Settings(設定)」アプリを開き「General」→「International」→「Language」と選択していき(**図step5-4〜6**)、「日本語」を選び、右上の「Done」を押しましょう(**図step5-7**)。

シミュレータが再起動されたら、自分のアプリのアイコンをクリックしてみましょう。今度は日本語設定の状態で開くので、**図step5-8**のように日本語の文字に切り替わっていることが確認できるはずです。

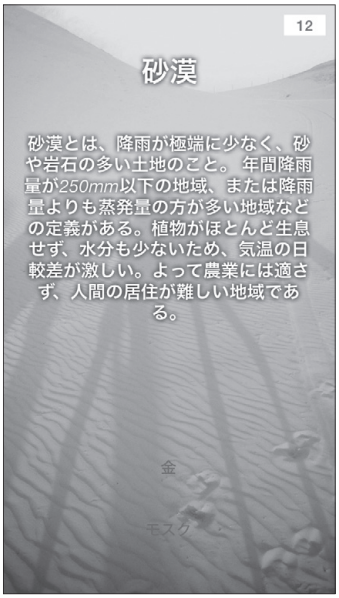
step5-2

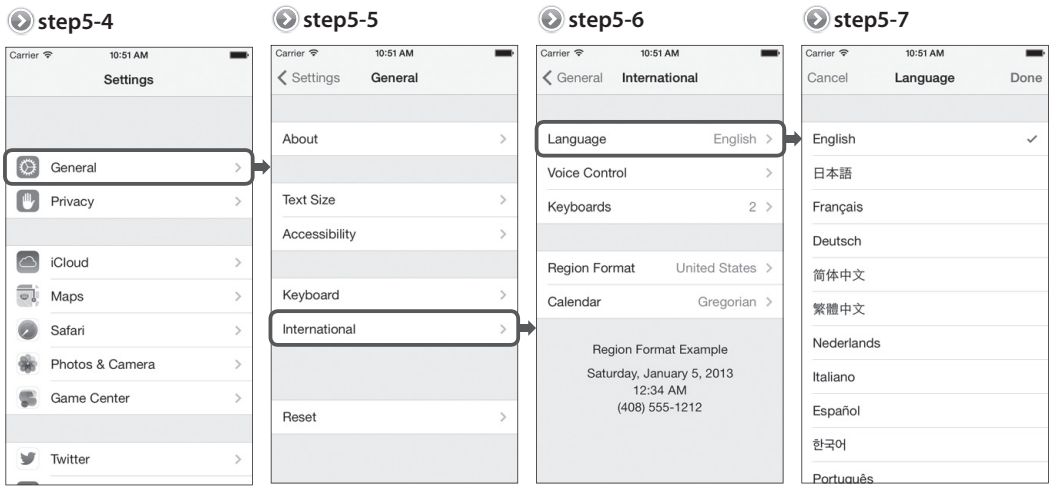


step5-3



step5-8





まとめ

以上がStoryboardを簡単にローカライズする方法でした。今回は紹介しきれませんでしたが、このほかにも文字だけでなく画像を言語ごとに変更する方法や、本誌連載第9回で覚えたUIActionSheetやUIAlertViewなども言語別に対応させる方法などがあります。

ローカライズは対応する言語の数だけ作業量が増えるため、最初からどこまでローカライズするべきかの判断はなかなか難しいところです。最初は最小限の言語に対応した状態でリリースし、ほかに需要が高そうな国があれば、後からアップデートで追加していくという手もあります。ただ、日本語のほかに英語くらいは対応しておくと世界に広まるチャンスが上がることは間違いないので、できるのであれば対応することをお勧めします。手間をかけた分だけアプリを使ってもらえる可能性が高まるので、ぜひがんばってみてください！SD

本連載のサポートページで記事の補足説明をしています。あわせてご活用ください！
› http://www.gimmiq.net/p/sd.html

リオ・リーバス／Leo Rivas
Twitter @StudioLoupe

iOSアプリ開発を中心に電子絵本作家・漫画家として活動中。代表作は、KDDI株式会社に社内導入され、世界で40万ダウンロードを記録する革新的な電卓アプリ「FusionCalc」と、国連主催のWSA Mobile 2013を受賞した、顔の動きで電子書籍が読める「MagicReader」。電子絵本はiBookstore/Kindleストア共に児童書カテゴリ総合1位を獲得。

いたのくまんぼう／Itano Kumanbow
Twitter @Kumanbow

神奈川工科大学非常勤講師。リオさんとはGimmiQ名義で「MagicReader」（手を使わずにページがめくれる電子書籍ビューワ）をリリース。個人ではNinebonz名義で「江頭ジャマだカメラ」（無料総合1位獲得）、「i列車の車窓から―そうだ! 京都行こう!―」（バーチャル旅行アプリ）などをリリース。アプリ紹介サイト「あぷまがどっとねっと（http://appmaga.net/）」の技術サポータ。

第46回 HTML5でLチカに挑戦

モバイルデバイス初のオープンソースプラットフォームとして、エンジニアから高い関心を集めるGoogle Android。いち早くそのノウハウを蓄積したAndroidエンジニアたちが展開するテクニックや情報を参考にして、大きく開かれたAndroidの世界へ踏みだそう！

今岡 通博　IMAOKA Michihiro
日本Androidの会 コミュニティ運営委員
Facebook https://www.facebook.com/imaoka.micihihiro

はじめに

今回はHTML5とAndroidを使って外部器機を制御する、もっとも簡単な方法の1つを紹介します。ダイヤルトーン(DTMF)とAndroidスマートフォンを用いたRGB LED(発光ダイオード)の制御です。HTML5を用いるメリットはプラットフォームに依存しないところです。しかしハードウェアに直接アクセスすると、プラットフォーム依存になってしまいます。この矛盾を解決する1つの手段がDTMF (Dual-Tone Multi-Frequency)を使うことなのです。

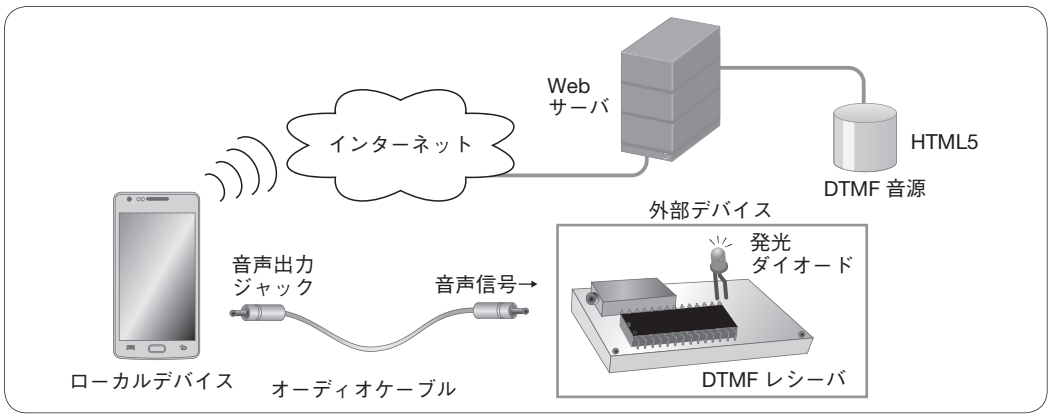
しくみはいたって単純で、DTMFの音源をサウンドファイルとしてWebサーバ側に用意しておき、HTML5でこの音源を再生します。外部機器側ではDTMFレシーバICで解釈(デコード)し、4bitの信号として取り出します。このうちの3bitを用いてRGB LEDを8色に変化させています。

今回はコードを書くだけでなく、電子工作的な要素も盛り込んでいます。ブレッドボードを用いるので半田付けは必要ありませんが、ラジオペンチやニッパをご用意ください。動作確認にはテスタもあったほうがよいかもしれません。

HTML5でLEDを点灯させるしくみ(システム概要)

図1がシステムの概要図です。ローカルデバイスであるスマートフォンの音声出力ジャックと外部デバイスであるブレッドボード上の回路とは、オーディオケーブルで接続されています。

▼図1　システム概要図



スマートフォンが発する音声信号が外部デバイスに伝わるしくみです。ブレッドボード上にはDTMFレシーバICとそれを動作させるために必要な周辺回路で構成されています。スマートフォンから発せられたDTMFダイヤルトーンを解釈(デコード)した結果は、RGB LEDが点灯する色として反映されます。

サンプルコード[注1]で利用するサーバのURLは、

URL http://imaoca.webcrow.jp/dtmf/

です。また、その下のディレクトリaudio-dtmf/以下にDTMF音源ファイルが格納されています。なお動作の様子はYouTubeでご覧いただけます[注2]。

HTML5でDTMFダイヤルトーンを発声させてみよう(SOFT編)

全ソースコードは**リスト1**に掲載しています。HTMLファイルの構成は<head></head>で囲まれたヘッダ部分と<body></body>で囲まれたボディ部分に分けられます。ヘッダ部分は各種属性や、後ほど説明するスタイルシートやスクリプトを記述します。ボディ部分はHTML本体を記述します。

ヘッダ部の中の、<style></style>に囲まれた部分はスタイルシートを記述します。HTML5ではスタイルシートはCSS(Cascading Style Sheets)という別ファイルにする場合も多いのですが、当サンプルコードではヘッダの中に記述しています。

HTMLコードが表示するページの内容であるのに対し、スタイルシートはその見え方やレイアウトを効率的かつ統一的に記述します。また、ブラウザによる差異を限定的な個所に集めて記述するためのものでもあります。

図2はローカルデバイスのブラウザでサンプルコードindex.htmを実行したイメージです。残念ながら本誌面はカラーページではないので確認しづらいですが、左から緑、青、シアン、赤、黄、マゼンタ、白、黒とボタンが並んでします。ボタンを押せば、その色に発光ダイオードが点灯する画面デザインになっています。

#contents{……}の節(**リスト1-❶**)ですが、これは後で説明するbtnの配置を横並びにするための記述です。**display**項にはさまざまなオプションがあります。それぞれ-webit-boxがSafari/Google Chrome用、-moz-boxがFirefoxに対応した記述となります。

.btn(**リスト1-❷**)はクラス名btnとして正方形の図形を定義しています。widthとheightで縦横とも50ピクセルの長さに設定します。marginで隣り合う図形との間隔を4ピクセルに設定します。border-radiusで角に丸みを持たせています。

.btn:mth-of-type(番号)の節(**リスト1-❸**)は、btnの番号順の要素にスタイルを適応しています。このサンプルでは1番目の要素から8番目の要素に色を設定していきます。

<script></script>の間(**リスト1-❹**)に、JavaScriptでindex.htmの動作を記述します。**window.onload**以下の項はブラウザ上でindex.htmを開いたときに一度だけ実行する処理を記述します。まずクラス名btnのオブジェクトをすべて取得し、eBtnListへ格納します。次にそのオブジェクトリストをもとに、すべてのbtnオブジェクトにクリックイベントを登録していきます。クリックイベントが発生したときにそれぞれのbtnに対応した音声ファイルがAudio("ファイル名").play()により再生されます。たとえば1番目のbtn(**図2**の1番左のオブジェクト)が押されると、audio-dtmf/dtmf_1.wavが再生されるしくみです。

HTMLの本体部分(**リスト1-❺**)には表示されるボタンを記述します。またボタンが押されたとき、イベントリスナーのfunction(**リスト1**の下線部分)から、ボタンの属性としてDTMF音源ファイル名の一部を"val"という属性名で取り

注1) 本誌サポートページからダウンロードできます。
URL http://gihyo.jp/magazine/SD/archive/2014/201403/support

注2) http://www.youtube.com/watch?v=6WwrrZjSuAA

▼リスト1　サンプルソースコード（index.htm）

```
<!DOCTYPE html>
<html><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <meta charset="UTF-8">
        <title>HTML5でLチカ</title>
        <style>
        #contents {                                    ┐
            background-color: aliceblue;               │
            display: -webkit-box;                      ├─❶
            display: -moz-box;                         │
            display: -o-box;                           │
            display: box;                              ┘
        }
        .btn {margin: 4px; border-radius: 8px;width: 50px; height: 50px;} ──❷
        .btn:nth-of-type(1) { background-color: green; }   ┐
        .btn:nth-of-type(2) { background-color: blue; }    │
        .btn:nth-of-type(3) { background-color: cyan; }    │
        .btn:nth-of-type(4) { background-color: red; }     ├─❸
        .btn:nth-of-type(5) { background-color: yellow; }  │
        .btn:nth-of-type(6) { background-color: magenta; } │
        .btn:nth-of-type(7) { background-color: white; }   │
        .btn:nth-of-type(8) { background-color: black; }   ┘
        </style>
        <script>
            window.onload = function() {                                                    ┐
                var eBtnList = document.getElementsByClassName("btn");                      │
                   for (var i=0,len=eBtnList.length; i<len; ++i) {                          │
                      eBtnList[i].addEventListener("click", function(){                     ├─❹
                      (new Audio("audio-dtmf/" + this.getAttribute("val") + ".wav")).play();│
                   }, false);                                                               │
                }                                                                           │
            };                                                                              ┘
        </script>
    </head>
    <body>
        <header id="header"><h1>HTML5でRGB行灯の色を変えてみよう.</h1></header>
        <div id="contents">
            <div class="btn" val="dtmf_1">1</div>      ┐
            <div class="btn" val="dtmf_2">2</div>      │
            <div class="btn" val="dtmf_3">3</div>      │
            <div class="btn" val="dtmf_4">4</div>      ├─❺
            <div class="btn" val="dtmf_5">5</div>      │
            <div class="btn" val="dtmf_6">6</div>      │
            <div class="btn" val="dtmf_7">7</div>      │
            <div class="btn" val="dtmf_8">8</div>      ┘
        </div>
</body></html>
```

▼図2　サンプルコード動作画面

出せるようにそれぞれのボタンと関連付けています。

DTMF音声ファイルの生成

ここでお断りしておくと、HTML5ではブラウザにより再生できる音声ファイル形式がことなるので、ブラウザに実装されている音声ファイル形式を取得して、それに応じて適切な音声ファイルを提供するしくみにするのが本来の作法です。しかし今回は本稿の趣旨にかかわらないコードは極力省くという方針ため、音声ファ

▼表1　ブラウザ上に表示されるオブジェクトとDTMF音源のファイル、LEDの色の対応表

ブラウザに表示されるbtn番号	DTMF番号（ダイヤルパッドの番号）	ファイル名 audio-dtmf/	周波数 低域(Hz)	周波数 高域(Hz)	D2 (Red)	D1 (Blue)	D0 (Green)	LEDの色
1	1	dtmf_1.wav	697	1209	0	0	1	Green
2	2	dtmf_2.wav	697	1336	0	1	0	Blue
3	3	dtmf_3.wav	697	1477	0	1	1	Cyan
4	4	dtmf_4.wav	770	1209	1	0	0	Red
5	5	dtmf_5.wav	770	1336	1	0	1	Yellow
6	6	dtmf_6.wav	770	1477	1	1	0	Magenta
7	7	dtmf_7.wav	852	1209	1	1	1	White
8	8	dtmf_8.wav	852	1336	0	0	0	Black

イル形式はwavのみとさせてもらいました。wavフォーマットに対応していないブラウザ[注3]で試したい場合は、フォーマットの変換を行ってください。

さて、DTMFのダイヤルトーン音源はあらかじめ生成したwavファイルをaudio-dtmf/ディレクトリの下に格納します。**図2**のように、ブラウザのページ上では左からDTMFトーンの1番から8番に対応するように並んでいます。btn番号とDTMF番号とファイル名の対応は**表1**を参照してください。

DTMFの音声ファイルを生成する方法はいろいろあるとは思いますが、Web上で任意のDTMF音声ファイルをダウンロードできる「ON LINE DTMF Tone Generator[注4]」というサイトを見つけましたので、ここではそれを紹介します。

このサイトのページ中段にある「Type your keypad sequence here」の下のテキストボックスと「Download.wav file」と書かかれたプッシュボタンがあるので、これらを操作するだけです。

テキストボックスには「112163 11219611 #9632 ##9696」とあらかじめ表示されていますが、いったんこれをクリアします。そして、たとえばDTMF番号1の音声ファイル（wavファイル）を生成したいのであれば、テキストボックスに「1」と入力し、プッシュボタンを押します。するとローカルPCのダウンロードフォルダに適当なファイル名でwavファイルが生成されます。これを**表1**に従ってdtmf_1.wavにリネームし、audio-dtmf/ディレクトリ内にコピーします。この操作を同様に8回繰り返せばDTMF番号に相当する8個のwavファイルが用意できます。もっとも、この操作が面倒という方は筆者がアップした音源ファイルをそのままコピーして使ってくださっても結構です。

今回のサンプルでは1つのボタンにDTMF1音しか割り当てていませんが、1つのファイルに複数のダイヤルトーンを連続して生成することも可能です。それを使うとLEDを特定のシーケンスで点滅させることもできます。

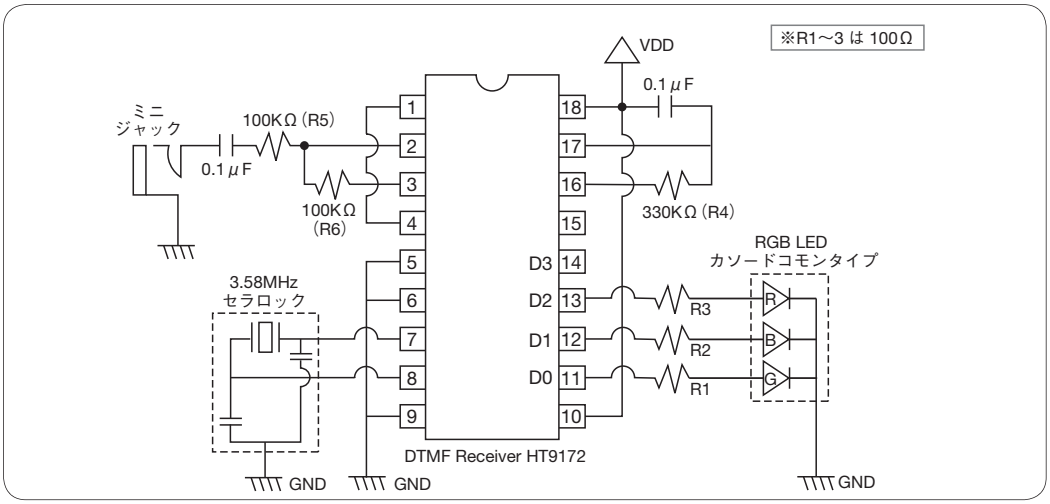
DTMFで発光ダイオードを点灯させてみよう（HARD編）

DTMFダイヤルトーンの番号と点灯するLEDの色の関係について説明します。 DTMFは数字の0から9までとA、B、C、Dと＊、#の16種類の記号を2種類の音声帯域の周波数の組み合わせで送る規格です。今回の用途では数字の1から8までの符号を用います。

DTMFレシーバICにどのDTMFダイヤルトーンが入ってくると、LEDが何色に光るかを**表1**に示しました。RGB LEDは赤、緑、青に光るLEDが同一のパッケージに封入されており、それぞれリード線につながっています。これらのリード線をON/OFFすることで光の3原色が交じり合って8通りの色を表現します。今回は色がうまく交じり合うようLED光拡散

注3）https://developer.mozilla.org/ja/docs/Web/HTML/Supported_media_formats
注4）http://www.audiocheck.net/audiocheck_dtmf.php

▼図3　回路図



キャップを使いました。

DTMFの音を解釈して4bitの出力信号として取り出すのがDTMFレシーバ(HT9172)です。**表1**をご覧ください。DTMFレシーバの出力、D0、D1、D2はRGB LEDのそれぞれGreen、Blue、Redに接続されています。すなわち、DTMFダイヤルトーン1の音を入力するとD0の出力が1となり、Greenが点灯するという具合です。ダイヤルトーン2の音の場合はD1の出力のみが1となり、Blueが点灯します。ダイヤルトーン3の場合はD0とD1が1となり、GreenとBlueが点灯して2つの光が交じり合った水色(Cyan)となります。このようにDTMFダイヤルトーンに1から8を入力すると、それに対応した色にLEDが点灯するというわけです。

図3が外部デバイスの回路図です[注5]。オーディオケーブル経由で流れてきた音声信号をミニジャックで受け、コンデンサと抵抗を経てDTMFレシーバの2番ピンに入ります。推奨回路[注6]ではDTMFレシーバの7番ピンと8番ピンには3.579545MHzの水晶発振子を接続していますが、筆者は3.58MHzのセラロック発振子(コンデンサ内蔵タイプ)を用いました。セラロックは水晶と比べ多少精度は劣るものの、低価格であることと入手性の良さから採用しました。

入力されたDTMFのデコード結果は11番ピン、12番ピン、13番ピン、14番ピンにそれぞれD0、D1、D2、D3が出力されます。このうちの3bitの出力信号は抵抗経由で、RGB LEDのそれぞれ対応する色のアノードに接続されています。これらの抵抗値は、RGBの色のバランスをとるために調整された値を用いますが、今回はすべて100Ωとします。

これ以外にDTMFレシーバに接続されている部品および値は、データシートの推奨回路と同じ値を採用しています。VDDには供給電源のプラス側を、GNDにはマイナス側を接続します。供給電源の電圧は5Vが標準です。だいたい3.6Vから動作しますが、5.5Vは超えないようにしてください。

ブレッドボード上に回路を構成する(MAKE編)

それではいよいよ**図3**の回路をブレッドボード上に構成していきます。ブレッドボードの部品を装てんするホールの行には数字、列にはアルファベットのアドレスがふられています。X列とA列の間には1ホール分のスペースが空いています。E列とF列の間には2ホール分のス

注5) 回路図とブレッドボード配線図は本誌サポートサイトにもアップしておきます。

注6) http://www.holtek.com/pdf/comm/9170v111.pdf

ペース(溝)が空いています。

今回利用するブレッドボードはX列、Y列がそれぞれ内部で電気的に接続されています。また中央の溝を挟んで、それぞれの行も内部で接続されています。

図4がこれからブレッドボードに装てんする部品群です。各部品の入手先は**表2**の部品表を参考にしてください。これらをブレッドボードに2段階に分けて装てんしていきます。

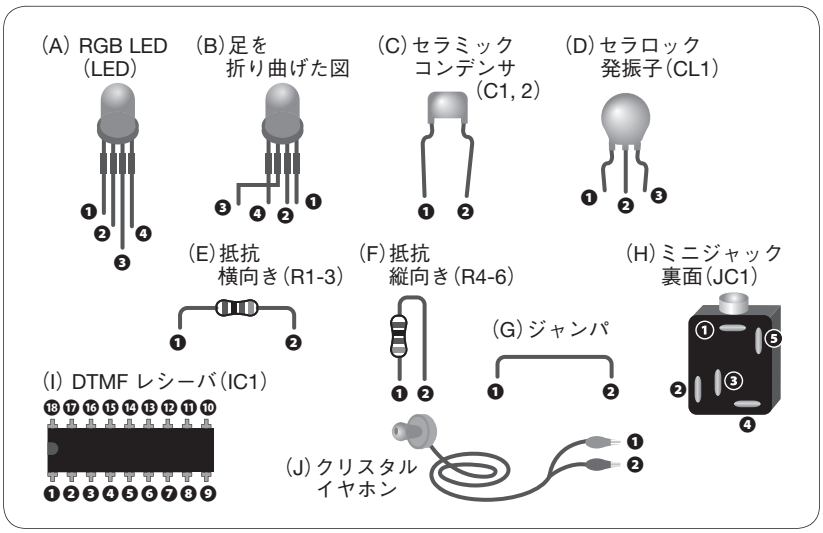
部品装てん1段階目

図5が最初のステップです。ジャンパ線と背の低い部品を先に装てんします。**図4(G)**のようなジャンパを用意します。直径0.6mmのスズめっき線を適当な長さに切り、両端を1cm程度折り曲げます。今回の製作ではブレッドボードの1ホールまたぐジャンパを6本、2ホールまたぐジャンパを2本、3ホールまたぐジャンパを1本用意します。ブレッドボードのホールの間隔は2.54mm(0.1インチ)ですので、それに応じてスズめっき線をニッパなどでカットします。

抵抗は**図4(E)**のような横向きにブレッドボードに装てんする抵抗を作製します。これらはDTMFレシーバとRGB LEDの各アノード間に入る抵抗です(R1〜3)。DTMFレシーバのD0とRGB LEDの緑のアノードをつなぐ抵抗は5ホール、D1と青をつなぐ抵抗は5ホール、D2と赤をつなぐ抵抗は4ホールまたぎますので、それに合わせてリード線をカットし両端をジャンパ同様1cm程度折り曲げます。

用意した部品を**図5**のように装てんしていきます。これらの部品は極性などありませんから

▼図4　部品図鑑

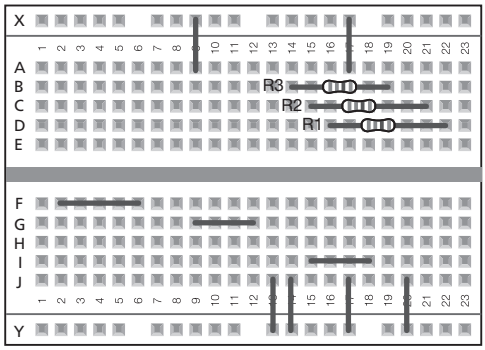


▼表2　部品表

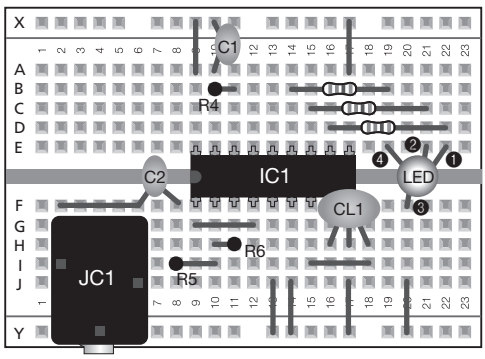
部品名	数量	参考価格	参考購入先
HT9172 DTMF Receiver	1	105	マルツパーツ館
セラロック3.58MHz	1	20	秋月電子通商
抵抗100KΩ	2	1	秋月電子通商(100個購入時の単価)
抵抗300KΩ	1	1	秋月電子通商(100個購入時の単価)
抵抗100Ω	3	1	秋月電子通商(100個購入時の単価)
セラミックコンデンサ0.1μF	3	10	秋月電子通商(10個購入時の単価)
3.5mmステレオピンジャック基板取付用AJ-1780	1	50	秋月電子通商
ブレッドボードEIC-301	1	150	秋月電子通商
RGB フルカラーLED　5mm　OSTA5131Aカソードコモン	1	60	秋月電子通商
LED光拡散キャップ	1	4	秋月電子通商(50個購入時の単価)
クリスタルイヤホン　みのむし端子付(動作検証用)	1	380	若松通商

備考：2013/12/27現在のものです。価格および在庫を保障するものではありません。

▼図5　ブレッドボードジャンパ配線



▼図6　ブレッドボード完成



向きを気にする必要はありません。

部品装てん2段階目

次のステップでは残りの部品を**図6**のように装てんしていきます。比較的背の低い部品から装てんしていくと楽に作業ができます。まず、DTMFレシーバ(**図4(I)**)を装てんします。この部品は向きを逆に装てんすると、動作しないばかりか壊れてしまう可能性がありますので注意してください。部品の表面のどちらか一端には半月状のくぼみがあります。これを左に向けて下に来るピンが左から1番から9番まで並んでいます。上のピンに移って9番の対面が10番ピン。右から順に18番ピンまで並んでいます。1番ピンがブレッドボードのホールアドレスF列の9行になるように装てんします(IC1)。

セラロックを装てんします。これは3本足ですが極性がありません(CL1)。次は縦向きに装てんするよう抵抗を**図4(F)**のように折り曲げます。330KΩ1本(R4)と100KΩ2本(R5、R6)を用意し、**図6**に従って装てんします。次にセラミックコンデンサ(**図4(C)**)を装てんします(C1、C2)。極性はありません。F列の6行ですが、ここは1つのホールにジャンパとセラミックコンデンサのリード線を入れますので注意してください。

ミニジャックを装てんします。**図4(H)**はミニジャックの裏面です。1番ピンがブレッドボードのY列の4行に来るように位置決めしてください(JC1)。ミニジャックのピンは少しホールの大きさに合わなくてきついのですがしっかり押し込んでください。今のところ実用的には問題ないようです。ただこのブレッドボードを再利用して、他の部品を装てんすると穴が広がりすぎていて多少接触が悪くなるかもしれません。その点はご了承ください。

最後にLEDを装てんして作業完了です。**図4(A)**はRGB LEDのピン配列を示しています。❶が緑のアノード、❷が青のアノード、❸が3つのLEDの共通カソード、❹が赤のアノードとなっています。**図4(B)**のように3番ピンを折り曲げます。長さは2ホールまたぐ長さにして、先を1cmほど折り曲げます。ほかのピンも1cmくらい残してカットします。ただ、仮装てんであればカットしないで直接ブレッドボードに差し込んでも問題ないでしょう。

これで、すべての部品の装てんが完了しました。早速電源をつないでみたいところですが、はやる気持ちを抑えてもう一度ブレッドボードの配線を確認してみてください。テスタがあればX列とY列間の抵抗値を測ってみてください。両者がショートしてないことが確認できたら、次に電源を接続します。X列にプラス、Y列にマイナスを接続します。くれぐれも電源電圧が5.5Vを越えないようにしてください。電源を入れた直後はDTMFレシーバの出力の状態は不定なので、電源を入れるたびにLEDが何色になる

かはわかりません。筆者の場合は3.6Vの電源を用いましたが、テスタがある方は、電源とブレッドボードのX列あるいはY列の間の電流を測ってみてください。LEDが消灯あるいは未装てんの状態で1.5mA程度であれば回路はほぼ問題ないでしょう。

動作確認

ローカルデバイスと外部デバイスをつないで結合テストです。まずローカルデバイスでブラウザを立ち上げて、DTMF音が再生できるか確認してください。確認できたらオーディオケーブルでローカルデバイスの音声出力ジャックと外部デバイスのミニジャックを接続します。外部デバイスの電源を入れてください。LEDは何かしらの色で点灯していると思います。いよいよブラウザの操作でDTMF音を外部デバイスに送り込んでみます。LEDは意図した色に点灯したでしょうか。

うまくいかなかった場合は1つ1つ確認していきましょう。原因としてローカルデバイスの音量不足が考えられます。音量を上げて試してみてください。それでもうまくいかない場合は、外部デバイスのDTMFレシーバに音声信号が伝わっていない可能性があります。ブレッドボードのY列の任意のホールとJ列10行のホール間に、クリスタルイヤホン(**図4(J)**)を接続すると小さな音ですがDTMF音を確認できます。音が確認できても正常動作しない場合は、ブレッドボード上の配線をもう一度確認してみてください。

音が確認できない場合は、オーディオケーブルとミニジャックの接触あるいは相性の問題かミニジャックがブレッドボードに適切に装てんされてない可能性があります。これらを確認してみてください。

ブラウザの操作でLEDの色は変わるが、意図した色にならない場合は、DTMFレシーバの出力とLEDのアノードまでの経路を確認してみてください。

ブレッドボードに部品やジャンパを装てんする際、ホールの中の接点までリード線が届いてないことがよくあります。この場合ブレッドボードの表面を見ただけでは、なかなか問題個所を発見することができません。必ず1cm以上はホールにリード線を差し込むようにしてください。

完成が確認できたら、筆者はホットボンドで装てん部品を固定しています。しばらく使っているうちに部品が取れたりするトラブルを避けることができます。部品やブレッドボードを再利用する際も、あとを残さず比較的簡単にはがすことができるので重宝しています。

まとめ

いかがでしたか。ブラウザの操作でLEDの色が変わるだけなのですが、なぜか楽しくなりませんか。DTMFレシーバの出力は4bitあるので、もっといろんなものをつなげれば楽しいかもしれませんね。YouTubeの筆者のチャンネル[注7]には、同じしくみでモータを制御したりする動画もアップしています。HTML5とDTMFレシーバの組み合わせはとても単純ですが、さまざまな応用が可能かと思います。ぜひチャレンジしてみてください。SD

注7) http://www.youtube.com/user/imaoca

今岡 通博(いまおか みちひろ) 日本Androidの会 コミュニティ運営委員

松山市在住。今岡工学事務所(個人事業主)として組込み系、FPGAがらみの開発を生業とするかたわら、日本Androidの会、SAKURAボードユーザ会などのオープンソース系コミュニティの運営に携わる。
Mail imaoca@gmail.com Facebook https://www.facebook.com/imaoka.micihihiro
YouTube http://www.youtube.com/user/imaoca

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第5回 ◇ 次世代パッケージ管理システム pkg(8)

◉著者プロフィール
後藤 大地（ごとう だいち）
BSDコンサルティング（株） 取締役／（有）オングス 代表取締役／FreeBSD committer
エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

FreeBSD 10.0新機能、次世代パッケージ管理システムpkg(8)

FreeBSDはサードパーティ製ソフトウェアの導入や管理に「Ports Collection」を使用しています。Ports Collectionは**make**(1)をベースメカニズムに採用したソフトウェアビルドやインストール方法をまとめたもので、アプリケーションの情報、ビルド方法、パッチ、インストール方法、依存関係などのデータがまとまったソフトウェアのカタログ集のようなものになっています。ユーザはPorts Collectionを使うことで、2万5千個ほどのサードパーティ製ソフトウェアをインストールして利用できます。

FreeBSDではPorts Collectionを使って、事前にコンパイルしたソフトウェアからサードパーティ製ソフトウェアを導入したり管理したりする方法も提供しています。こちらは「パッケージ」と呼ばれています。パッケージはFreeBSD 9までとFreeBSD 10以降で別のものに変わります。ここではこの2つのパッケージ管理システムを、それぞれ「従来のパッケージ」「pkg(8)」という言葉で区別します。**pkg**(8)がFreeBSD 10以降の新しいパッケージ管理システムです。

従来のパッケージにはいくつかの課題がありました。主な課題をまとめると次のようになります。

- 一貫性のあるアップグレード方法が提供されていない
- 依存関係のトラッキングが不完全

従来のパッケージはリリースバージョンに対して、その時点のPorts Collectionを使ってビルドされたソフトウェアをまとめたものです。リリース時のインストールには便利ですが、その後のバージョンアップには向いていません。通常、インストール後のアップグレードには**portupgrade**(1)や**portmaster**(8)などの管理ツールを使い、Ports Collectionからビルドする方法を利用するか、Ports Collectionからパッケージを作成し、そのパッケージからインストールするといった手段を取ります。

pkg(8)はこうした課題に対する1つの答えを提供するものです。パッケージ管理システムとしては最後発となるもので、**yum**(8)や**apt-get**(8)と似ています。yum(8)やapt-get(8)との大きな違いは、常に最新版にアップデートされ続けるPorts Collectionとシームレスに統合していることと、週に1回といったペースで最新のパッケージが提供される点にあります。

pkg(8)の提供する機能は強力です。FreeBSD 10以降はFreeBSDをベースに構築されたシステムのパッケージ管理の手法が一変すると見られます。本稿ではpkg(8)の基本的な動作を説明するとともに、セキュリティ脆弱性への対応方法やPorts Collectionと混在して運用する方法を紹介します。

pkg(8)の基本的な使い方

pkg(8)は「**pkg　サブコマンド**」または「**pkg　オプション　サブコマンド　オプション　引数……**」と

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

いった形で使用します(**図1**)。サブコマンドで動作を切り替えるしくみを採用していて、pkg(8)コマンドだけでパッケージ管理できるようになっています。提供されている主なサブコマンドやオプションは**表1、2**のとおりです。

パッケージをインストールするには「**pkg install パッケージ名**」で実施します(**図2**)。アンインストールは「**pkg remove パッケージ名**」です(**図3**)。インストールできるパッケージの検索は「**pkg search キーワード**」で実施します(**図4**)。インストールされているパッケージの情報は「**pkg info パッケージ名**」で得られます(**図5**)。インストールされているパッケージの一覧を見るなら「**pkg info -a**」と実行します。

パッケージのビルドは毎週水曜日に実施され、数日後にリポジトリに反映される予定になっています。週に1回のペースで最新版に更新していくといった運用方法になります。アップグレードはインストール済みソフトウェアすべてに対して実施されます。「**pkg upgrade**」でその作業を行えます。

個別にアップグレードしたり、アップグレードを停止する場合はPorts Collectionを使ったり「**pkg lock**」を使います。

pkg(8)のしくみを知ろう

pkg(8)コマンドは2つ存在します。/usr/sbin/pkgと/usr/local/sbin/pkgです(**図6**)。

▼表1　pkg(8)主要サブコマンド

サブコマンド	意味
add	パッケージの登録とインストール
annotate	パッケージのアノテーション編集(追加、削除、変更)
audit	セキュリティ脆弱性を抱えたパッケージの報告
autoremove	オルファンパッケージの削除
backup	ローカルパッケージデータベースのバックアップとリストア
check	データベース一貫性のチェックと依存関係エラーのチェック
clean	キャッシュから古いパッケージを削除
create	ソフトウェアパッケージディストリビューションの作成
delete	パッケージのアンインストール
fetch	リモートリポジトリからパッケージのダウンロード
help	サブコマンドの使い方などの情報を表示
info	インストール済みパッケージの情報を表示
install	パッケージのインストール
lock	パッケージのロック
register	パッケージをローカルデータベースへ登録
remove	パッケージのアンインストール
search	パッケージの検索
set	ローカルデータベースにおけるパッケージ情報を編集
shlib	対象のライブラリにリンクしているパッケージの表示
stats	パッケージ統計情報の表示
unlock	パッケージのロックを解除
update	パッケージリポジトリカタログのアップデート
updating	UPDATING情報の表示
upgrade	パッケージの一斉アップグレード
version	インストール済みパッケージのバージョン情報を表示
which	指定したファイルがどのパッケージ経由でインストールされたものかの表示

▼表2　pkg(8)主要オプション

pkg(8)オプション	意味
-j	指定したjail(8)環境下でpkg(8)を実行せよというオプション
-c	指定したchroot(8)環境下でpkg(8)を実行せよというオプション
-l	サブコマンド一覧の表示
-v	pkg(8)バージョン番号の表示

▼図1　pkg(8)コマンド実行例

```
% pkg
Usage: pkg [-v] [-d] [-l] [-N] [-j <jail name or id>|-c <chroot path>] [-C <configuration file>] [-R
<repo config dir>] <command> [<args>]

Global options supported:
        -d              Increment debug level
        -j              Execute pkg(8) inside a jail(8)
        -c              Execute pkg(8) inside a chroot(8)
 (…略…)

For more information on the different commands see 'pkg help <command>'.
%
```

チャーリー・ルートからの手紙

i-node番号やファイルのサイズを確認すると、それぞれ別のソフトウェアであることがわかります。

/usr/sbin/pkgはFreeBSDのベースシステムとして提供されているコマンド、/usr/local/sbin/pkgはports-mgmt/pkgからインストールされるソフトウェアです（図7、8）。

pkg(8)コマンドはその性質上、pkg(8)コマンドそのものをかなり頻繁にアップグレードする必要があります。ベースシステムに統合したコマンドではそういったことを実施できません。このため、ベースシステムにマージされたpkg(8)は/usr/local/sbin/pkgをインストールして、そちらへ処理を移すブー

▼図2　パッケージのインストール

```
# pkg install dash
Updating repository catalogue
The following 1 packages will be installed:

        Installing dash: 0.5.7

The installation will require 121 KB more space

68 KB to be downloaded

Proceed with installing packages [y/N]: y   ← yと入力
dash-0.5.7.txz                                    100%   68KB  68.3KB/s  68.3KB/s   00:00
Checking integrity... done
[1/1] Installing dash-0.5.7... done
#
```

▼図3　パッケージのアンインストール

```
# pkg remove dash
Deinstallation has been requested for the following 1 packages:

        dash-0.5.7

The deinstallation will free 121 KB

Proceed with deinstalling packages [y/N]: y   ← yと入力
[1/1] Deleting dash-0.5.7... done
#
```

▼図4　パッケージの検索

```
# pkg search subversion
git-subversion-1.8.4.3
p5-subversion-1.8.5
py27-hgsubversion-1.5.1
py27-subversion-1.8.5
ruby-subversion-1.8.5
subversion-1.6.23_2
subversion-1.7.14
subversion-1.8.5
subversion-book-4515
subversion-static-1.8.5
#
```

▼図5　インストール済みパッケージの情報表示

```
# pkg info dash
dash-0.5.7
Name           : dash
Version        : 0.5.7
Origin         : shells/dash
Architecture   : freebsd:10:x86:64
Prefix         : /usr/local
Categories     : shells
Maintainer     : eadler@FreeBSD.org
WWW            : http://gondor.apana.org.au/~herbert/dash/
Comment        : POSIX-compliant implementation of /bin/sh
Flat size      : 121KiB
Description    :
DASH is a POSIX-compliant implementation of /bin/sh that aims to be as small as
possible. It does this without sacrificing speed where possible. In fact, it
is significantly faster than bash (the GNU Bourne-Again SHell) for most tasks.

WWW: http://gondor.apana.org.au/~herbert/dash/
#
```

▼図6　pkg(8)は2つ存在する

```
# which -a pkg
/usr/sbin/pkg
/usr/local/sbin/pkg
#
```

▼図7　/usr/local/sbin/pkgはports-mgmt/pkgで作成されたパッケージ

```
# ls -il /usr/sbin/pkg /usr/local/sbin/pkg
45434 -r-xr-xr-x  1 root  wheel  133568 Dec 20 06:03 /usr/local/sbin/pkg
19824 -r-xr-xr-x  1 root  wheel   33400 Dec 17 10:49 /usr/sbin/pkg
#
```

トストラップコマンドになっています。/usr/local/sbin/pkgはベースシステムに統合されたコマンドではありませんので、pkg(8)経由で頻繁にアップグレードできます。

従来のパッケージではインストールしたソフトウェアの情報は/var/db/pkg/以下にテキストファイルの状態で保持されていました。pkg(8)ではそれぞれSQLiteのデータベースとして保持されるようになります(**図9**)。これらデータベースを直接操作することは推奨されておらず、pkg(8)コマンド経由で操作するようにとされています。

自動アップグレードとカスタマイズのバランスのよい運用へ

FreeBSDカーネルおよびユーザランドはFreeBSD Updateで自動アップグレードが可能です。pkg(8)でサードパーティ製ソフトウェアの自動アップグレードも可能になります。FreeBSD 10からはBHyVeハイパーバイザも利用できるようになりますので、たとえばホスト環境はFreeBSD Update + pkg(8)でアップデートを完全に自動化しておき、カスタマイズしたカーネルの機能やオプションを指定してビルドしたソフトウェアについてはBHyVe上で実行する、といったように切り分ける運用もできるようになります。

FreeBSD 10以降で登場する新機能はFreeBSDの運用やシステム構築のスタイルを大きく変えるものがあります。しばらくはこうした新機能を紹介していこうと思います :) SD

p.s. PBIとpkg(8)という両極端のアプローチ

FreeBSDに限らずほかのUNIXやLinuxでも同様ですが、サードパーティ製ソフトウェアの管理には必ず「依存関係」という課題が発生します。一部をアップデートすればほかのソフトウェアの依存関係に影響が出る可能性があります。ここを解決しないと、安定的に最新のソフトウェアを使用することができません。

1つの解決方法はWindowsやMac OS Xのアプリケーションのように、そのアプリケーションの動作に必要になるライブラリなどを単一のパッケージのなかに全部含めてしまうというアプローチです。ほかのソフトウェアへの依存度が低くなり、そのアプリケーションのみでバージョンアップやロールバックを実施できます。UNIXではPC-BSDの採用しているPBIがこのアプローチを採用しています。

pkg(8)はこれとは真逆のアプローチを取ったところが興味深いところです。全体としての依存関係があるのだから、なるべく短周期で登録されているサードパーティ製ソフトウェアを全部ビルドして最新版を提供すればよい、という発想です。FreeBSDプロジェクトでは、この発想を実現するためにネットワークインフラとビルドインフラのハードウェアやサービスを含めて強化に取り組みました。

▼図9 /var/db/pkg/以下はSQLiteデータベースと脆弱性データファイル

```
# tree /var/db/pkg/
/var/db/pkg/
|-- auditfile
|-- local.sqlite
|-- repo-FreeBSD.sqlite
|-- repo-packagesite.sqlite
\`-- vuln.xml

0 directories, 5 files
#
```

▼図8 periodic(8)も/etc/periodic/ではなく/usr/local/etc/periodic/が使われる

```
# pkg info -l pkg
pkg-1.2.4_1:
        /usr/local/etc/bash_completion.d/_pkg.bash
        /usr/local/etc/periodic/daily/411.pkg-backup
        /usr/local/etc/periodic/daily/490.status-pkg-changes
        /usr/local/etc/periodic/security/410.pkg-audit
        /usr/local/etc/periodic/security/460.pkg-checksum
        /usr/local/etc/periodic/weekly/400.status-pkg
        (…略…)
        /usr/local/sbin/pkg
        /usr/local/sbin/pkg-static
        /usr/local/sbin/pkg2ng
        (…略…)
#
```

第18回

そのソースコード forkしてませんか?

綴綴 昌嗣
Koketsu MASATSUGU
レッドハット㈱
常務執行役員
製品・ソリューション事業統括本部長

はじめに

筆者は、恵比寿の会社で営業の管理職をしています。オープンソースの会社であるにもかかわらず、自分でLinuxのソースコードをあまり見たことはありません。恵比寿の社員はみんながソースコードを読めるということはなく、筆者のように「残念な」ひともいます。

きっかけ

そんな残念なひとですが、恵比寿通信に寄稿することを、何気なく引き受けてしまいました。10年以上Software Designを読んだこともないため、年末年始の休みに、本屋で手に取った表紙に踊る言葉は「あなたの好きなシェルは何ですか？」。筆者の世代(50歳を越えてます)だとcshですか。筆者は、営業が使うワークステーションでさえSolarisという会社にいましたのでcshを使っていました。ただ、親しみがあるのはbshとその互換シェルのkshです。社会人になって5年目に、当時勤めていたメーカーでUNIX System V Release 4(SVR4)をベースにしたOS開発プロジェクトのプロジェクト管理とビルドチームを担当することになりました。2年にわたりグループメンバと毎日のビルドとエラー対応に悪戦苦闘。bshスクリプトを大量に書き、そして、その数十倍を読みました。ソースファイルツリーにある膨大な数のMakefileでは、UNIXのデフォルトシェルであるbshで手続きが記載されていたためです。

それなりにオープンだった時代

技術者として、営業として、UNIXとLinuxの両方に関連する仕事をしてきました。いうまでもありませんが、UNIXとLinuxは似ています。LinuxはPOSIXにだいたい準拠しています。プログラムを書いて、ビルドする基本的な方法も似ています。

最近では、レガシィと悪口を言われるUNIXですが、開発現場はオープンな雰囲気でした。『伽藍(がらん)とバザール』で言われる「伽藍」の印象は、内側でバタバタしていた技術者としては、まったく感じませんでした。インターネットで相互接続され緊密に連携した開発コミュニティこそUNIXにはありませんでしたが、AT＆TやカリフォルニアC大学バークレイ校を中心として組織された開発者集団が作りあげたソースコードが、有償／無償のライセンスで頒布され、世界中の技術者がUNIXの進化を実現するために力を尽くしていました。ライセンスされたソースファイルを展開して、初めてエディタで開いたとき、「これが世界への扉だ」と感動したのを思い出します。

当時の筆者の職場では、AT＆Tから独立したUnix System Laboratories(USL)が開発したソースコードを基にAT＆Tのハードウェアである3b2からMIPS CPUのワークステーションに移植し、独自にマルチプロセッサ化したり、新機能を追加していました。移植というと簡単に思われるかもしれませんが、アーキテクチャがまるで異なるCPUへのkernelの移植はとて

もたいへんでした。プロセス、メモリ管理の中核はアセンブラ。実際には、機能不足があるので、自分たちで造り込みもする。品質への要求がUSLと日本のメーカーで異なるので、バグの洗い出しと、徹底した性能改善。TPCなどという、世界標準のベンチマークで他社と性能を比べられるので必死です。パニック文なんてけしからんものはなくして、何があっても必要な情報はとったうえで問題なくシステムを停止できるようにする。lint、kdb、cvsだけの開発環境でUNIX kernelの開発をするのは、それはたいへんでした。アセンブラと16進数のデータを読んでCのソースコードが即座に頭に浮かぶくらいプログラミングとデバッグに明け暮れました。おかげで、体力だけはつきました。

UNIXのGUIやファイルシステムは発展途上でした。OPENLOOKやMotifなどのウィンドウ管理システムも何が主流になるかわからず、ネットワークファイルシステムとしてAFS(今ではOpenAFSとしてオープンソースになっています)でなくてサン・マイクロシステムズのNFSを選択するのにさえ勇気が必要でした。メインフレーム出身の上司たちからは、リソース管理の機能が必要だと言われて独自に作り込みもしました。RHEL6のcgroupと同じ機能を20年以上前に作り込みました。

ちなみに、NFSはサン・マイクロシステムズが開発したものだということはご存じでしたか。当時、サンは自社の開発した技術を低価格でライセンスをしてその技術の普及に努めていました。後に、オープンソース「的」なアプローチでJavaが大きく普及することになる片鱗を見ることができます。

筆者のいた会社だけでなく、世界中のコンピュータメーカーが、同じように、独自のUNIXを作っていたと思います。自社の作った技術を他社にライセンスすることもありました。実際に、筆者も自社の開発したUNIXを国内外のメーカーにライセンスする営業活動をしました(筆者の営業としての原点です)。日本語マニュアルを作って、USLに逆ライセンスもしました。それまでのメインフレームやオフコンとは異なり、UNIXの世界では、自社だけで開発した技術には価値がなくなり、外から飛びこんでくる新しい技術を自分たちの技術に昇華させ、自分たちの開発した技術を世界に普及させるということが最重要課題でした。

互換性の維持は難しい

UNIXの普及により、OS技術の共通化は進みました。しかし、SVR4という共通のソースコードであったにもかかわらず、互換性は高くありませんでした。POSIX仕様で縛ってもすべての動作が同じになるわけではなく、結局は別のOSになってしまいました。同じソースコードを同じCPU向けにビルドしたはずのUNIXも、実際のビルド環境によって違ったものになります。ライセンス先の外国企業がソースコードから日本語のライブラリを外してしまったので、static linkされていたviのバイナリからエラーが出ていたのを思い出します。加えて、それぞれの企業が独自に機能追加やバグ修正をしていくので、互換性がドンドンなくなっていきました。

オープンに世界を相手に仕事をしていたつもりだったのですが、出てきたものは結局「囲い込み」の道具になってしまいました。

悪者はfork

Linuxでも同じことが起きてもおかしくありません。オープンソースではすべてのソースコードが共有され、公開されます。しかしコンパイラや、そのオプション、ヘッダやソースファイルのレイアウトはビルド環境で異なりますから、ビルドしたものがバイナリレベルでの互換性があるとは言えません。ましてソースコードが変わってしまえば、UNIXの二の舞です。

POSIX仕様が互換性の拠り所だったUNIXからは大きく進歩しました。しかし、まだUNIX

と同じ非互換の罠にはまるリスクがあります。ソースコードの「fork」です。forkさえしなければ、ソースコードレベルでの互換性は維持できます。恵比寿の会社では、forkは「悪者」です。「悪者」の誘惑はカスタマイズ文化の日本ではとくに強いのですが、筆者達は頑固なまでに拒絶しています。このforkを許さない厳しい姿勢がRed Hat Enterprise Linuxが生き残ってきた最大の理由だと思います。

ドアの影から

オープンソースは新しい時代に入っています。OpenStackやHadoopがITの未来を築きつつあることは、もはや否定できないでしょう。新しく生まれたテクノロジも、しっかりしたコミュニティのガバナンスの中で、各メーカーの「囲い込み」のための独自色を捨てて普及を図っていかなればなりません。そうでなければ、forkのために失速したUNIXの時代に逆戻りしかねません。

オープンソースの世界では、エンジニアの皆さんは技術革新を実現していくグローバルなコミュニティの一員です。オープンソースは人類共通のソフトウェア資産を作り上げるうえでの最良の方法だと思います。が、forkにはご注意を。ドアの影からあなたを覗いているかもしれません。最近また、forkを世の中で見かけますから……。SD

第47回 Ubuntu Monthly Report

LibreOffice 4.2の新機能

Ubuntu Japanese Team
あわしろいくや　AWASHIRO Ikuya
Mail ikuya@fruitsbasket.info

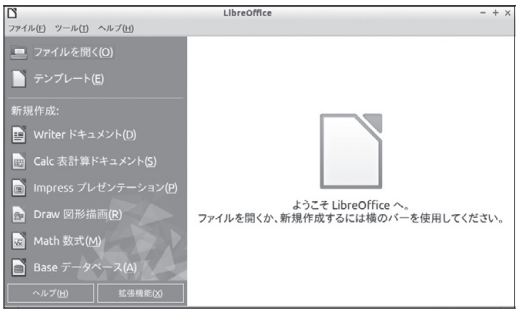
今回は無事に1月30日にリリースされた、LibreOffice 4.2の新機能を紹介します。

概要

LibreOffice（以下LibO）は1年に2回のメジャーバージョンアップが行われていますが、4.2はそのスケジュールに沿った本年最初のリリースです。4.1ではApache OpenOffice 4.0からの機能の取り込みが多かったのですが、4.2ではあまり多くなく、LibreOffice独自の新機能が目白押しで、それだけ開発が活発に行われているということです。

新機能に一貫性がないのはいつものとおりなのですが、今回は昨年のGoogle Summer of Code（GSoC）での成果がいち早く取り込まれているのが目立ちます。これまでは別ブランチで途中まで開発して放置ということもありました。今回もそういうのもありますがあまり多くなく、GSoCでの開発がおおむねスムーズに進んだことが伺えます。

図1　刷新されたスタートスクリーン



全般

■スタートスクリーン

Ubuntuで使っている分には気づかないかもしれませんが、LibOを引数なしで起動すると表示されるスタートスクリーンが一新されました（**図1**）。これはGSoCでの成果です。WriterやCalcなどが起動できるのはこれまでと同じですが、過去に作成／編集したファイルも表示されるようになりました。

■Google Drive

ついにLibOから直接Google Driveにアクセスしてファイルの編集ができるようになりました。これもGSoCでの成果です。ただし、残念ながらLinux版の4.2.0では動作しませんでした。また、特殊なconfigureオプションが必要ですので、パッケージ版、すなわちUbuntuに最初からインストールされているLibOではこの機能が有効になっていない可能性があります。

この機能が有効かどうかは、次の方法で確認してください。まず［ツール］-［オプション］-［全般］の［LibreOfficeダイアログを使用する］にチェックを入れてください。次にファイルダイアログを開き、右上にある［...］をクリックします。するとリモートアクセスに必要な設定を行うダイアログが表示されるので、［種類］を［CMIS］にしてください。［サーバーの

種類]に[Google Drive]があればこの機能が使用できます。コツとしては、[ユーザー名]はGmailのアドレスであることくらいです(**図2**)。

■すべて検索

検索機能に[すべて検索]ボタンが追加されました。これまでだと上下の矢印で1個ずつ検索する必要がありましたが、[すべて検索]ボタンを使うと上下の区別がなく一気に全部検索できます。

Writer

■囲み線

Wordでいうところの囲み線機能が実装されました。これもGSoCの成果です。Writerでも同じ翻訳にしました。囲みたい文字を選択し、[書式]-[文字]-[囲み線]で設定できます。日本語的には[囲い文字]のほうがよく使う気もしますが、残念ながらこの機能は実装されていません。

Calc

■コアの書き換え

Calcは平均してどのバージョンでもかなりの改善があるのですが、4.2ではついにコアを書き換えるところまで来てしまいました。コアの書き換えによって処理速度が向上し、メモリの使用量も減少している、とのことです。そればかりか、OpenCLを使用してGPUでの計算もできるようになりました。

AMDの発表によると、AMDの新アーキテクチャのAPUを使用する[注1]と、従来よりも7倍の速度で処理できる(こともある)とのことです。もちろんそこまで高速化はされなくてもOpenCLなのでAMDのAPUだけではなくIntelやNVIDIAでもOpenCLを有効にすればこの恩恵にあずかることができます。WindowsだとOpenCLが有効になっているドライバをインストールするとおしまいという感じなのですが[注2]、Ubuntuだとそうもいかなくてやや困難な場合もあります。具体的には、AMDのAPUでOpenCLを有効にするにはUbuntu 14.04にあるドライバが必要です。Calcを起動して[ツール]-[オプション]-[LibreOffice Calc]-[数式]-[ユーザー定義]-[詳細]-[一部の数式の演算にOpenCLを有効にする]を[真]にしてください(**図3**)。

■関数の追加

関数はExcelとの相互運用性を向上させるために追加されることが多いのですが、4.2ではいつもより多くの関数が追加されています。詳しくは**表1**を参照してください。ほとんどがExcel 2010の関数ですが、一部Excel 2013で追加された関数にも対応しています。しかもWEBSERVICE関数とFILTERXML

図2　このように設定できれば、Google Driveにアクセスできます

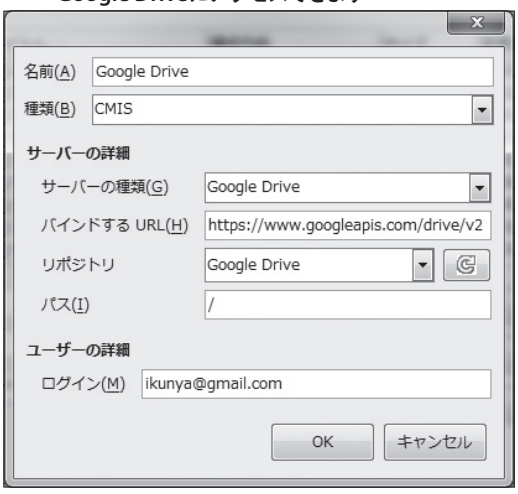
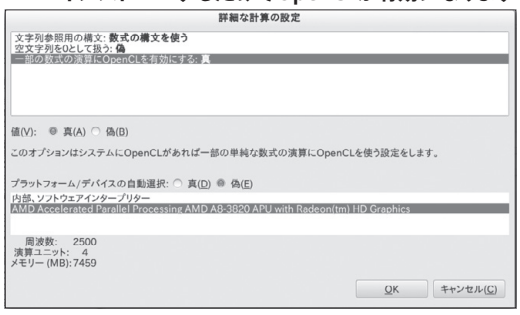


図3　Ubuntu 14.04であれば、プロプライエタリなドライバをインストールするだけでOpenCLが有効になります



注1) コードネーム "Kaveri" のAPUです。すでに販売は開始しています。

注2) 筆者が所有するWindows PCで確認してみたところ、IntelとAMDはドライバが新しければOpenCL対応になっているという感じでした。NVIDIAは確認していないものの同様だと思われます。

関数ですので、外部で取得したデータを加工してCalcにインポートできます。

■乱数生成

筆者にはどのようなニーズがあるのかとんと見当がつかないのですが[注3]、[編集]-[連続データ]-[乱数]で簡単に乱数が生成できるようになりました。

■統計

[データ]-[統計]に、

- サンプリング
- 基本統計量
- 分散分析（ANOVA）
- 相関
- 共分散
- 指数平滑
- 移動平均

が追加され、これらの計算ができるようになりました。Excelの分析ツールと比較するとまだ足りない部分もありますが、統計関係で確実にCalcを使う機会が増えるであろう新機能です。

Impress

■サイドバーがデフォルトに

これまでのタスクペインに代わり、サイドバーがデフォルトになりました。もちろん実験的な機能ではなくなっているのですが、デフォルトで表示されるのはImpressだけで、あとは必要な場合[表示]-[サイドバー]のチェックを入れて表示させる必要があります。タブが横にまとまり、クリップアートの挿入もしやすくなって使い勝手が向上したのではないでしょうか。

■アニメーションの設定

[アニメーションの設定]が刷新されました。機能がボタンにまとまったので、ページ内に設定したアニメーションの一覧が広く表示されるようになりました。これまではアニメーションの一覧の枠があまり広くなく、1ページにたくさんのアニメーションを設定するとスクロールして確認する必要がありましたが、その問題を解消して視認性が大幅に向上しました。

■Impressリモート

Impressリモートはプレゼンの最中にImpressを操作する機能で、今まではAndroid用しかありませんでしたが、GSoCでiOS用も開発されました。しかし、1月中旬現在ではまだApp Storeでは配信されていません。また、Android用Impressリモートも改良されています。こちらもやはりGSoCの成果です。

Base

■Firebird

今まで内蔵データベース（組み込みデータベース）

注3）適当な数字でサンプルデータを作るのに便利かなと思いましたが、そんなの原稿書く人でないと使わないですよね……。

表1　4.2で追加された関数

関数	Excelの対応バージョン	ジャンル
WEBSERVICE, FILTERXML	2013	文字列
LEFTB, LENB, MIDB, RIGHTB	2000	文字列
COVARIANCE.P, COVARIANCE.S	2010	統計
STDEV.P, STDEV.S	2010	統計
VAR.P, VAR.S	2010	統計
BETA.DIST, BETA.INV	2010	統計
BINOM.DIST, BINOM.INV	2010	統計
CONFIDENCE.NORM, CONFIDENCE.T	2010	統計
F.DIST, F.DIST.RT, F.INV, F.INV.RT, F.TEST	2010	統計
EXPON.DIST, HYPGEOM.DIST, POISSON.DIST, WEIBULL.DIST	2010	統計

はHSQLDBだけだったのですが、4.2からはFirebirdも選択できるようになりました。ただしRC2で確認する限りWindowsでは両者が選択できるのですが、Linux版はHSQLDBだけでした。もちろんUbuntuパッケージ版だとどうなるのかもわかりません。

■Basic IDEのコード補完

Baseに限ったことではありませんが、Basicでコードの補完ができるようになりました。詳細な設定は［ツール］-［オプション］-［LibreOffice］-［Basic IDEオプション］から変更できます。

インポート

■フィルタ

AppleのKeynoteインポートフィルタが実装されましたが、ちょっと試してみたところではあまり精度はよくなかったです。そもそもKeynoteのファイルをImpressで開くことはあまり多くないのかなと思います。ほかにもいくつかインポートフィルタが実装されていますが、古いMac（Macintosh）用だったりして、あまりお世話になることはなさそうです。

既存のフィルタでは、Microsoft Office形式[注4]の相互運用性は向上しています。とくにWord（DOCX）はかなり手が入っています。このあたりを重点的に開発している企業（CloudOn）が昨年Advisory Boardになったため、今後も継続的な改善が期待できます。また、パスワード付きのMicrosoft Officeファイルを扱えるようになりました。聞いたところによると今までは2007までの対応だったのですが、今回2013まで対応したとのことです。Microsoftが暗号化の形式をいろいろと変えているので、対応できなくなっていたらしいです。

Apache OpenOfficeからのソース取り込み

Apache OpenOfficeからのソースコードの取り込みも継続的に行われています。大きな変更点だとIAccessible2サポートがありますが、4.2では実験的な機能であり、デフォルトでは有効になっていません。IAccessible2はその名のとおりアクセシビリティツールのためのAPIです。もともとIBM Lotus Symphonyにあった機能でApache OpenOffice 4.0に盛り込まれる予定でしたが、4.1に延期になりました。その4.1も春にはリリース予定です。リリース前に実験的機能として盛り込んでしまうのはサイドバーでも同じでした。小さな変更だと各種バグフィックスもあります。ここでいうところの大きな変更／小さな変更というのはソースコードの修正量のことであり、重要かそうでないかということではないことにご注意ください。

ライセンス

バイナリを使っているぶんにはあまり関係ありませんし、開発者的にもあまり関係はないのですが、バイナリのライセンスがLGPL3からMPLv2に変更になりました（**図4**）。確かにAL2とMPLv2のソースコードが増えているにもかかわらず、バイナリのライセンスがLGPL3というのは実体に即していなかったので、落ち着くところに落ち着いたという感じです。また1つOpenOffice.orgの呪縛から解き放たれたともいえます。

その他

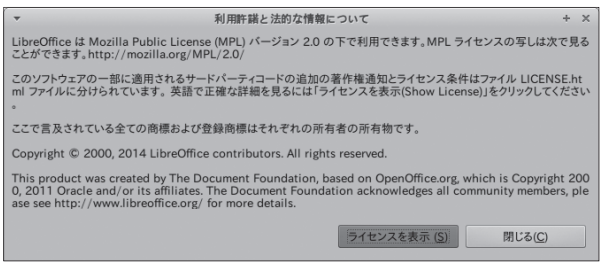
■Gladeへの置き換え

LibOは各種ダイアログ／ウィンドウをGlade[注5]で

注4） 古いバイナリ形式も、新しいXML形式もです。

注5） ざっくり解説すると、GTK+アプリケーションで使われているXMLで構成されたUIです。

図4　ライセンスがMPLv2になりました



書き換えるという作業を継続的に行っており、4.2でもかなり進行しています[注6]。このペースだと4.3にはだいたい終わるのではないかと思っていたりもしますが、既存の翻訳がリセットされるので翻訳者としてはとてもたいへんです。ちなみにGladeで書き換えたダイアログ／ウィンドウとそうでないのは、ウィンドウサイズを変更できるかどうかで区別できます。わかりやすいところでは［ファイル］-［PDFとしてエクスポート］で、4.2だと確かにウィンドウのサイズが変更できるようになっています。

■上級者向き設定

［書式］-［ページ］-［LibreOffice］-［詳細］に［上級者向き設定］というボタンが追加されました。これはFirefoxでいうところの“about:config”に相当するもので、生の設定を直接変更できます。すなわち本来はあまり触るべきところではないのですが、今まではどういじっていいかもよくわからなかったので設定項目が増えたことはいいことではないでしょうか。

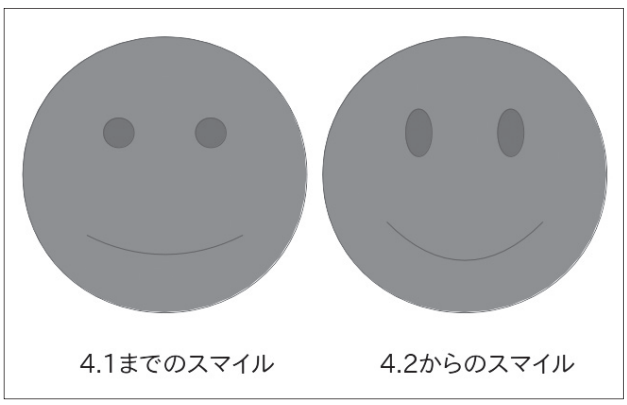
■スマイル

主としてDraw/Impressの［記号シェイプ］にある［スマイル］が、より笑顔になりました。ある意味4.2の一番重要な変更点といえます[注7]（**図5**）。

注6） 1月現在で70%ほどが完了しているとのことです。

注7） ネタばらしするのもあれですが、リリースノートでもオチっぽく紹介されているのでここでもそうしてみました。

図5　4.1と4.2のスマイルを比較



翻訳

前述のとおり新機能がいろいろとあり、UIの書き換えも進んで8000語前後という未訳語がありましたが、なんとか翻訳が進んで4.2.0リリース時点での未訳語は700語前後になりそうです。残っている部分は、おおむねその分野の専門知識がないと翻訳できない部分です。また、同じダイアログでも前のバージョンと翻訳が変わってしまうことも容易に想像でき、それはすべてにおいていいことではないのでWindows向けの翻訳確認バイナリを用意し、それが窓の杜[注8]で紹介されたりもしました。Windows版のバイナリビルドは根気と時間とWindowsの知識が必要でUbuntuでビルドするのと比較してすごくたいへんなのですが、やはりユーザベースが多いので確認してくれる人の見込みも多いのです[注9]。

Ubuntuで4.2を使用する方法

Ubuntu 14.04にはデフォルトで搭載されます。通常リリースのサポート期間が9ヵ月になったということは13.10ユーザはアップグレード必須ですし、12.04ユーザも順次アップグレードしていくでしょうし、LibO 4.2だけを単体でインストールすることはあまりないのではないかと思われます。

とはいえ、今回もPPA[注10]での提供はされるでしょう。ただし問題があり、12.04にPPAのLibreOfficeをインストールするとファイルサーバに置いてあるファイルを開けないという不具合があります。それを考えても、今回ばかりはPPAを使用するのではなく、Ubuntuそのものをアップグレードしてしまうのがいいでしょう。SD

注8） http://www.forest.impress.co.jp/docs/news/20140110_630372.html

注9） とはいえ期待したほどではないので、やはりユーザが少ないということを意識せざるを得ません。

注10） https://launchpad.net/~libreoffice

第24回

Linux 3.13の新機能
パケットフィルタリングエンジン nftables

Text：青田 直大　AOTA Naohiro

Linusの旅行の影響で一週間遅れることとなりましたが、Linux 3.13も無事にリリースされました。今回は3.13に追加された新しいパケットフィルタリングエンジンnftablesについて解説します。

nftablesとiptables

新しいパケットフィルタリングエンジンと聞くと「なぜ今さら?」と思われるかもしれません。確かにLinuxカーネルにはすでにiptablesなどのパケットフィルタリング機能が入っていますが、IPv4やIPv6などで同じようなコードが使われている、カーネル内部にパケットにマッチするルールを記述しているため拡張性が悪いといった問題があります。nftablesはこういった問題を解決し、カーネル内のコードがシンプルになっているという利点があります。それではiptablesの現状と、nftablesではそれがどのように変わるのかを見ていきましょう。

iptablesの問題
コードの重複

今のLinuxカーネルにはパケットを扱うフレームワークが4つも存在しています。まずはIPv4、IPv6それぞれのパケットを扱う「iptables」と「ip6tables」、ARP（アドレス解決プロトコル）パケットを扱う「arptables」、そしてbridgeインターフェースを通過するパケットを扱う「ebtables」の4つです。この4つのフレームワークの中にそれぞれ同じようなコードが書かれています。

たとえばどちらもパケットフィルタリングを行っている部分であるiptablesの関数ipt_do_table（**リスト1**）とip6tableのip6t_do_table（**リスト2**）とを見てみましょう。IPv4とIPv6の違いということで“ip”が“ip6”になっていたりと細かな違いはいくつか見うけられますが、コメントも含めてほとんど似たものになっていることが確認できます。「IPv4とIPv6ということでとくに似ている」というのもありますが、このようにiptablesではコードが重複して書かれています。

この部分に対応するコードがnftablesでは**リスト3、4**のように変わっています。それぞれの関数ではIPv4、IPv6のヘッダ位置を示すオフセットなどの情報を設定する関数が呼び出され、メインのルールを適用してパケットを落とすかどうか決定するような部分は共通のnft_do_chain_pktinfo関数によって行われるようになっています。

このようなフィルタ実行部分のコード重複だけではなく、ほかのマッチやターゲットといっ

▼リスト1　net/ipv4/netfilter/ip_tables.c

```
/* Returns one of the generic firewall policies, like NF_ACCEPT. */
unsigned int
ipt_do_table(struct sk_buff *skb,
         unsigned int hook,
         const struct net_device *in,
         const struct net_device *out,
         struct xt_table *table)
{
    static const char nulldevname[IFNAMSIZ] __attribute__((aligned(sizeof(long))));
    const struct iphdr *ip;
    ⋮
    /*
     * Ensure we load private-> members after we've fetched the base
     * pointer.
     */
    smp_read_barrier_depends();
    table_base = private->entries[cpu];
    jumpstack  = (struct ipt_entry **)private->jumpstack[cpu];
    stackptr   = per_cpu_ptr(private->stackptr, cpu);
    origptr    = *stackptr;
    ⋮
        t = ipt_get_target(e);
        IP_NF_ASSERT(t->u.kernel.target);

#if IS_ENABLED(CONFIG_NETFILTER_XT_TARGET_TRACE)
        /* The packet is traced: log it */
        if (unlikely(skb->nf_trace))
            trace_packet(skb, hook, in, out,
                     table->name, private, e);
#endif
    ⋮
}
```

▼リスト2　net/ipv6/netfilter/ip6_tables.c

```
/* Returns one of the generic firewall policies, like NF_ACCEPT. */
unsigned int
ip6t_do_table(struct sk_buff *skb,
          unsigned int hook,
          const struct net_device *in,
          const struct net_device *out,
          struct xt_table *table)
{
    static const char nulldevname[IFNAMSIZ] __attribute__((aligned(sizeof(long))));
    ⋮
    /*
     * Ensure we load private-> members after we've fetched the base
     * pointer.
     */
    smp_read_barrier_depends();
    cpu        = smp_processor_id();
    table_base = private->entries[cpu];
    jumpstack  = (struct ip6t_entry **)private->jumpstack[cpu];
    stackptr   = per_cpu_ptr(private->stackptr, cpu);
    origptr    = *stackptr;
    ⋮
        t = ip6t_get_target_c(e);
        IP_NF_ASSERT(t->u.kernel.target);

#if IS_ENABLED(CONFIG_NETFILTER_XT_TARGET_TRACE)
        /* The packet is traced: log it */
        if (unlikely(skb->nf_trace))
            trace_packet(skb, hook, in, out,
                     table->name, private, e);
#endif
    ⋮
}
```

た部分にもコードの重複があります。たとえばah matchのコードも**リスト5、6**のように似ています。

iptablesの問題 カーネル内のプロトコル解析コード

前項でも見たようにiptablesにはさまざまな部分にコードの重複がありました。それだけではなくiptablesではカーネル内部にプロトコルの解析コードを持っています。もし新しくフィルタの条件にしたい部分ができた場合は、カーネルモジュールを新しく書く必要があります。

nftablesでは**リスト7**のようなプログラムをバイトコードに変換してカーネル側に送ります。

▼リスト3　IPv4のnftables

```
static unsigned int
nft_do_chain_ipv4(const struct nf_hook_ops *ops,
          struct sk_buff *skb,
          const struct net_device *in,
          const struct net_device *out,
          int (*okfn)(struct sk_buff *))
{
    struct nft_pktinfo pkt;

    nft_set_pktinfo_ipv4(&pkt, ops, skb, in, out);

    return nft_do_chain_pktinfo(&pkt, ops);
}
```

▼リスト4　IPv6のnftables

```
static unsigned int
nft_do_chain_ipv6(const struct nf_hook_ops *ops,
          struct sk_buff *skb,
          const struct net_device *in,
          const struct net_device *out,
          int (*okfn)(struct sk_buff *))
{
    struct nft_pktinfo pkt;

    /* malformed packet, drop it */
    if (nft_set_pktinfo_ipv6(&pkt, ops, skb, in, out) < 0)
        return NF_DROP;

    return nft_do_chain_pktinfo(&pkt, ops);
}
```

▼リスト5　IPv4のah match

```
net/ipv4/netfilter/ipt_ah.c
/* Returns 1 if the spi is matched by the range, 0 otherwise */
static inline bool
spi_match(u_int32_t min, u_int32_t max, u_int32_t spi, bool invert)
{
    bool r;
    pr_debug("spi_match:%c 0x%x <= 0x%x <= 0x%x\n",
         invert ? '!' : ' ', min, spi, max);
    r = (spi >= min && spi <= max) ^ invert;
    pr_debug(" result %s\n", r ? "PASS" : "FAILED");
    return r;
}

static bool ah_mt(const struct sk_buff *skb, struct xt_action_param *par)
{
    struct ip_auth_hdr _ahdr;
    const struct ip_auth_hdr *ah;
    const struct ipt_ah *ahinfo = par->matchinfo;

    /* Must not be a fragment. */
    if (par->fragoff != 0)
        return false;

    ah = skb_header_pointer(skb, par->thoff, sizeof(_ahdr), &_ahdr);
    if (ah == NULL) {
        /* We've been asked to examine this packet, and we
         * can't.  Hence, no choice but to drop.
         */
        pr_debug("Dropping evil AH tinygram.¥n");
        par->hotdrop = true;
        return 0;
    }
⋮
}
```

リスト7のバイトコードでは、

- ネットワークヘッダから16byte目から4byteをレジスタ1に読み込む
- レジスタ1と0x00ffffffの論理積をとる
- レジスタ1 == 0x005000fd9 の比較を行う
- レジスタ1 == 0x005000fd9 であればカウンタを増やす

といった処理が実行されます。すなわち、宛先アドレスが"217.15.8.0/24"であるようなIPv4パケットをカウントする処理が行われます。

このようにnftablesではプログラムをカーネル側に送ることができるので、新しくマッチしたい部分ができても対応するバイトコードをユーザ空間で書いて送信すれば良いので、カーネル内部に新しいコードを書く必要がありません。実際にカーネル内のコードがiptablesの70,000行ほどから、nftablesでは7,000行まで減ってシンプルになっています。また、nftablesではカーネルの更新なしに新しいマッチを行うことが可能になっているわけです。

iptablesのその他の問題

iptablesには、ほかにもルールの数が増えてくるとルールの追加に時間がかかるという問題がありました。nftablesでは1回のカーネル — ユーザ間の通信で複数のルールを送信できるようになったのでこの問題が解決されています。また、ユーザプログラムの観点では、iptablesはほかのプログラムから呼び出すことのできるラ

▼リスト7　カーネル側に送られるプログラム

```
payload load 4b @ network header + 16 => reg 1
bitwise reg 1 = (reg=1 & 0x00ffffff) ^ 0x00000000
cmp eq reg 1 0x005000fd9
counter pkts 0 bytes 0
```

▼リスト6　IPv6のah match

```
net/ipv6/netfilter/ip6t_ah.c
/* Returns 1 if the spi is matched by the range, 0 otherwise */
static inline bool
spi_match(u_int32_t min, u_int32_t max, u_int32_t spi, bool invert)
{
    bool r;

    pr_debug("spi_match:%c 0x%x <= 0x%x <= 0x%x\n",
         invert ? '!' : ' ', min, spi, max);
    r = (spi >= min && spi <= max) ^ invert;
    pr_debug(" result %s\n", r ? "PASS" : "FAILED");
    return r;
}

static bool ah_mt6(const struct sk_buff *skb, struct xt_action_param *par)
{
    struct ip_auth_hdr _ah;
    const struct ip_auth_hdr *ah;
    const struct ip6t_ah *ahinfo = par->matchinfo;
    unsigned int ptr = 0;
    unsigned int hdrlen = 0;
    int err;

    err = ipv6_find_hdr(skb, &ptr, NEXTHDR_AUTH, NULL, NULL);
    if (err < 0) {
        if (err != -ENOENT)
            par->hotdrop = true;
        return false;
    }

    ah = skb_header_pointer(skb, ptr, sizeof(_ah), &_ah);
    if (ah == NULL) {
        par->hotdrop = true;
        return false;
    }
⋮
}
```

イブラリを実装していないという問題があります。つまり、ほかのプログラムからiptablesの機能を使おうと思えばコマンドを叩くしかありませんでした。

nftablesではlibnftnlというnftablesの機能へのライブラリが開発されています。このライブラリを使うことで、nftコマンド(nftables用のコマンド)を使わずに自分でnftablesのルールを参照・追加・削除するようなプログラムを書くことができるようになります[注1]。

nftの使い方

ここからはnftablesのコマンドnftの使い方について見ていきます。nftでルールを設定するには3つの方法があります。1つはコマンド引数として指定する方法、2つめはファイルに書いて"-f"の引数として指定する方法、そして3つめは"-i"引数を用いてnftコマンドをinteractiveモードで起動する方法です。

iptablesでは初めからtableとchainが設定されていました[注2]が、nftablesでは初めから設定されているtableとchainはありません。ですので、まずはtableとchainを設定する必要があります。nftablesのソースに同梱されている"ipv4-filter"というファイルがiptablesに相当するchainを追加するためのルールを記述したファイルですのでこれを使います。"nft -f"でファイルを読み込み、"nft list table ip filter"を使うとchainが正しく追加されていることが確認できます(**図1**)。

では次に"8.8.4.4"へのパケットをカウントするルールを追加してみましょう。ルールの追加には"add rule <ファミリー> <テーブル> <chain> <条件> <ターゲット>"といった記法を使います。ターゲットに"counter"と"log"という2つが一度に指定できていることに注目してください(**図2**)。"list table"してみるとcounterの値が0で表示されています。8.8.4.4に1回pingしてから再度"listtable"するとcounterの値が正しく増えていることを見ることができます。

注1) たとえばnftablesのルールをjsonの記述から追加するCのコードが公開されています。https://git.netfilter.org/libnftnl/tree/examples/nft-rule-json-add.c

注2) そのため、何もルールを設定していなくてもパフォーマンスが落ちる問題がありました。

▼図1　tableとchainの設定

```
# cat /etc/nftables/ipv4-filter
#!/sbin/nft -f

table filter {
        chain input             { type filter hook input priority 0; }
        chain forward           { type filter hook forward priority 0; }
        chain output            { type filter hook output priority 0; }
}
# nft -f /etc/nftables/ipv4-filter
# nft list table ip filter
table ip filter {
        chain input {
                type filter hook input priority 0;
        }

        chain forward {
                type filter hook forward priority 0;
        }

        chain output {
                type filter hook output priority 0;
        }
}
```

nftables：set

iptablesでもIP setを使うことでIPアドレスの集合を扱い、その集合についてのルールを記述できます。nftablesではより統合された形でIPアドレスやポート番号の集合を定義し、使用できます。集合には無名の集合と、名前付きの集合とがあります。無名の集合は**図3**の❶のように“add rule”で“{”と“}”とでくくって使用できます。❷名前付きの集合はまず“add set”を使って名前の型を指定して定義し、❸その集合に要

▼図2　パケットをカウントするルールを追加する

```
# nft add rule ip filter output ip daddr 8.8.4.4 counter log
# nft list table ip filter
table ip filter {
…
        chain output {
                type filter hook output priority 0;
                ip daddr google-public-dns-b.google.com counter packets 0 bytes 0 log
        }
}
# ping -c 1 -q 8.8.4.4
PING 8.8.4.4 (8.8.4.4) 56(84) bytes of data.

--- 8.8.4.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 52.542/52.542/52.542/0.000 ms
# nft -n list table ip filter
table ip filter {
…
        chain output {
                type filter hook output priority 0;
                ip daddr 8.8.4.4 counter packets 1 bytes 84 log
        }
}
```

▼図3　IP setでIPアドレスの集合を扱う

```
# nft -i
nft> flush table ip filter
nft> add rule ip filter output ip daddr { 8.8.8.8, 8.8.4.4 } counter ……❶
nft> list table ip filter
table ip filter {
…
        chain output {
                type filter hook output priority 0;
                ip daddr { 8.8.8.8, 8.8.4.4} counter packets 5 bytes 331
        }
}
nft> add set ip filter google_dns {type ipv4_address;} ……❷
nft> add element ip filter google_dns {8.8.8.8, 8.8.4.4} ……❸
nft> flush table ip filter
nft> add rule ip filter output ip daddr @google_dns counter ……❹
nft> list table ip filter ……❺
table ip filter {
        set google_dns {
                type ipv4_address
                elements = { 8.8.8.8, 8.8.4.4}
        }
…
        chain output {
                type filter hook output priority 0;
                ip daddr @google_dns counter packets 0 bytes 0
        }
}
nft> delete set ip filter google_dns
```

素を追加します。❹こうして、名前を付けた集合はルールの定義の中で“@”を先頭に付けて参照できます。❺“list table”を使って“table ip filter”の中に定義されている集合を見ることができます。

verdictsとmapping

最後にverdictsとmappingについて見てみましょう。これは多くのプログラミング言語で使われている「辞書」のようなデータ構造を使ってルールのマッチングをできるしくみとなります。たとえば、“192.168.0.0/24”へのパケットを落とすが、“192.168.0.5”へのパケットだけは通すといった処理をverdictsを使って**図4**のように書くことができます。verdictsも集合の場合と同様に名前を付けることができます。

まだ多くのドキュメントでverdictsの“キー”と“値”の区切りに“=>”が使われていますが、これではシェルで使いにくいこともあり“:”を使うように変更されていることに注意してください[注3]。

まとめ

今回はiptablesなどを置き換えることを目的とした、新しいパケットフィルタリングエンジンnftablesについて解説しました。nftablesによってカーネル内にコードを書かなくても新しいマッチを行えるようになったり、ほかのプログラムから使えるライブラリが提供されるなど、パケットフィルタの開発が容易になります。

Linux 3.13がリリースされたことで3.14のマージウィンドウが開始されています。次回は3.14にどんな機能が入るのかを紹介できるかと思います。SD

注3）https://git.netfilter.org/nftables/commit/src/parser.y?id=21cfa9a7405f78f424c869e592d21ebdaf379803

▼図4　verdictsとmapping

```
# nft -ni
nft> flush table ip filter
nft> add rule ip filter output ip daddr vmap {192.168.0.0/24 : drop, 192.168.0.5 : accept}
nft> list table ip filter
table ip filter {
…
        chain output {
                type filter hook output priority 0;
                ip daddr vmap { 192.168.0.0-192.168.0.4 : drop, 192.168.0.5 : accept, ➥
192.168.0.6-192.168.0.255 : drop}
        }
}
nft> add map ip filter verdict_map {type ipv4_address : verdict;}
nft> add element ip filter verdict_map {8.8.8.8:drop}
nft> add rule ip filter output ip daddr vmap @verdict_map
nft> list table ip filter
table ip filter {
        map verdict_map {
                type ipv4_address : verdict
                elements = { 8.8.8.8 : drop}
        }
…
        chain output {
                type filter hook output priority 0;
                ip daddr vmap { 192.168.0.0-192.168.0.4 : drop, 192.168.0.5 : accept, ➥
192.168.0.6-192.168.0.255 : drop}
                ip daddr vmap @verdict_map
        }
}
nft> flush chain ip filter output
nft> delete map ip filter verdict_map
```

SDでSF 小飼弾

第3回

『声の網』
（星新一／角川文庫）

「SDでSF」といったら、やはり避けられないのが「人工頭脳の支配と反乱」というテーマ。ロボット工学三原則の生みの親であるアイザック・アシモフも好んで取り上げたテーマですし、その「ロボット」という言葉の生みの親であるカレル・チャペックの『R.U.R』自体、このテーマの作品になっています。ここであえて「電子頭脳」ではなく「人工頭脳」と書いたのは、電子計算機よりも古いテーマだから。実際アシモフの人工頭脳は、電子回路ではなく陽電子回路という謎技術で作られたという設定になっています。

しかし、この頃の人工頭脳には、人格と意志がありました。ある意味ふつうの「寛大なる（Benevolent）独裁者（Dictator）」。しかし人格を持った人工頭脳はいまだ実現されていないのは、みなさんご存じのとおり。HAL9000もアトムもどう実現したものやらいまだに見当もつきません。え？　北の独裁者三代目はドクター・ゲロ作ですって？　それはマンガの読みすぎですっ。

にもかかわらず、「人格なき人工頭脳による人類支配」は、見事に予言されています。星新一その人によって。『声の網』が上梓されたのは1973年。さすがにスマホどころかパソコンもない時代に書かれただけあって、人工頭脳とのやりとりは入出力とも電話による音声というところにこそ時代を感じさせますが、コンピュータではなくコンピュータネットワークが「支配者」であること、そこには「独裁者」が存在しないこと、そしてなによりそのネットワークが人類の福祉と治安を向上させるために生まれ、育てられたことは「完全に一致」しています。

本書を21世紀においてリアルにしているのは、「支配者」の手足が生身のヒトであること。ネットの通報で現場に向かう警官もヒトなら、データセンターで壊れた部品を交換するのもヒトなら、ポチられた商品を玄関まで届けてくれるのもヒト。「支配網」を実現するのにあたって、人格をもった支配者も必要なければ支配を実行する手足も必要なかったのですね。我々自身という部品がすでにあるのですから。

圧巻なのは、それがクラッキングやフィッシングといった悪意ではなく、それを防ごうという善意の積み重ねによって実現したこと。その善意の一つ一つを短編とし、それを十二編集めて長編としたことは、ショートショートの第一人者の面目躍如。読者の皆さんは、その善意の一片を担う立場にいるはずです。一時絶版だったのですが、復刊されたうえにKindle化されていることは、電網時代におけるせめてもの救いかもしれません。SD

題字／イラスト　by aico

March 2014

NO.29

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業界の2つのムーブメントを追う（情報銀行&AWS）

今回は2013年11月に大阪と大分で行った研究会の模様を報告します。

大阪大会

■Thinking RealSpace：みんなのパーソナル情報を集め気持ちよく提供してもらい活用しよう!!!　―情報銀行：あなたの「情報」を預けて利活用してみませんか？―

【講師】砂原 秀樹
（慶応義塾大学／日本UNIXユーザ会）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2013年11月8日（金）15:00～15:50
【会場】大阪南港ATC ITM棟 10階サロン

大阪大会は前回報告した関西オープンフォーラム（KOF）の中で行いました。2013年8月の島根大会と同じテーマで砂原さんに発表していただきましたが、今回は持ち時間が50分あったのでじっくりとお話をお聞きしました。参加者は38人でした。

最初に、これまでの砂原さんの研究テーマの中から「個人から情報を集める」例がいくつか紹介されました。たとえばインターネットカーのプロジェクトでは、個々の車両から集めた情報を集積し地図にマッピングすることで、渋滞状況やタクシーの巡回状況などがわかるようになりました。これらの成果は今では実サービスとして利用されています。

■「勝手に取られる」から「意識的に預ける」しくみへ

前述のように個人から出される情報には高い価値があるのですが、砂原さんは「勝手に取られている情報もたくさんある」と指摘します。これからはそのような個人情報を各自が意識できる形で預け、何かに役立てるしくみを作りたいという考えを示しました。その実現を目指すのが「情報銀行」プロジェクトです。このようなしくみができれば、たとえば医療においても個人の体質などのデータを利用した効果的な健康管理や予防治療が可能になりますし、ほかにも街づくりや災害対策などさまざまな方面への適用が考えられます。

情報銀行のしくみは、個人は総合的な個人情報の管理／利用を情報銀行に「信託」し、企業は情報銀行から情報を受け取り自社のサービスに利用する、というものです。実現のポイントとして、情報を保護するのは当然ですが、統合化された個人情報から新しい価値を生み出せることや、社会から信頼され、ここに預けることで良いことがある、というブランドを確立することが重要です。活動としては、まず母体となる組織として情報銀行コンソーシアムを設立し、技術開発や実証実験、社会に受け入れられるためのしくみ作り、利活用についての検討などを行っていきます。当初の活動期間は3年を予定しています。詳しくは情報銀行コンソーシアムのWebサイト[注1]をご覧ください。

質疑応答の時間も十分にあり、とくにKOFの基調講演のため来場していた高木浩光さんとの議論は貴重な1コマでした。これからどのような活動成果が出てくるか楽しみです。

注1）URL http://www.information-bank.org/

大分大会

■JAWS-UG熊本の活動について
【講師】松岡 祥仁（JAWS-UG熊本支部 副会長）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2013年11月23日（土）15:15～16:00
【会場】大分県消費生活・男女共同参画プラザ「アイネス」小会議室2

■国内シェア1位のAWS、その理由は？

大分大会は2年連続の開催となりました。今回はAmazon Web Services（以下、AWS）のユーザグループ「JAWS-UG」熊本支部の松岡さんにご講演いただきました。参加者は7名でした。

はじめに、同支部の主要メンバーが支部設立後続々と独立した（松岡さんもその1人）というエピソードを紹介し、このようなことが可能なのもAWSのおかげであるという話がなされました。続いて同支部の活動紹介として、まず2013年6月に行われたAWS Summit Tokyoに主要メンバー全員で参加した話が、多数の写真を交えて紹介されました。

同イベントには約9,000人が参加し、場内の混雑も相当なものでした。このときに日本のAWSユーザ数が初めて発表されたのですが、約2万社が採用しているとのこと。国内クラウドサービスでもAWSがシェア1位であることが明らかになりました。

イベントのスポンサー一覧を見ると、小さな会社がエバンジェリストなどの人的貢献を評価されて上位クラスのスポンサーに名を連ねています。これはAWSを使うことで小企業も大企業と対等に勝負できることの表れであり、そんな夢のある世界であることが、これだけ多くの人をイベントに引き寄せているのではないかというのが松岡さんの分析です。

■とはいえ、まだ課題も

熊本支部ではこれまでに4回のセミナーを開催していますが、上記の盛り上がりとは裏腹に、回を追うごとに参加者数が減っているというデータが紹介されました。3回目が18人にまで落ち込んだことに危機感を持ち、4回目は懇親会を熊本の路面電車を借り切って開催したら参加者が増えました。参加者数の伸び悩みの理由は、「AWSは基本的に英語で提供されている」「サービスが多岐に渡るため理解しにくい」「ハンズオンセッションでインスタンスを作成するのにも費用がかかる」「クラウドにデータを預けることへの抵抗感がまだ根強い」などが挙げられます。そして、やはり金の匂いがしないと人が集まらないので、AWSを基盤とする新たなビジネスを考えていきたい、具体的にはAndroid搭載のSmart TV製品とAWSを組み合わせたサービスを検討中であるという話をして講演を締めくくりました。

時間に余裕があったので質疑応答もいくつか行いました。熊本支部で行っているセミナーの内容は、AWSのサービスから1つを選んでエバンジェリストが解説、ハンズオンセッション、導入事例のライトニングトーク、セミナーが終わってからの懇親会といったところです。九州のJAWS-UGで一番活発なのは福岡で、ほぼ毎週会合を行っています。

■価格面も魅力の1つ

AWSの最大の魅力は価格で、とくにアップロード時は課金されずダウンロード時のみ課金されるしくみのため、大きなデータを長期保存するのに適しています。また、冗長化のしくみが標準で装備されているのも利点です。AWSは稼働時間で課金されるので、使わないインスタンスは停止しておくと課金されないといったノウハウも紹介されました。熊本支部の今後の活動としては、2014年3月頃に次回のセミナーを実施すべく企画しています。

◆ ◆ ◆

JAWS-UGは全国で草の根的に支部が立ち上がっており、九州内にもいくつかの支部があるそうです。このような動きの背景には技術的なおもしろさだけでなくビジネス的な要素もあることが今回の講演でわかり、これはもしかしたら1990年頃のUNIXブームと同じようなものなのかもしれないと思いました。SD

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
及川 卓也　OIKAWA Takuya
Twitter @takoratta
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka

第27回「ITx災害」会議（後編）

社会的課題をテクノロジで解決するためのコミュニティHack For Japanの活動をレポートする本連載。今号も前号に引き続き、昨年10月6日に開催された「ITx災害」会議の後半の模様をお届けします。

前号では、ITを用いた復旧・復興支援を行っている団体やコミュニティ、個人が集まり、今後の活動を議論するための「ITx災害」会議の背景や準備、午前中の会議の様子をお伝えしました。今号では、参加者間のネットワーキングを広げる試みとして企画した芋煮会と、午後のアンカンファレンスについてご紹介します。

芋煮会

前号でお伝えした参加証のほかに、もう1つの参加者の交流を促すしかけが「芋煮会」です（**写真1**）。皆さんは芋煮という料理をご存じでしょうか。芋、野菜、肉など具材を持ち寄って屋外で煮込んで皆で食べるという芋煮会は、東北でもとくに山形と宮城の秋の風物詩でもあります（**写真2**）。

今回のイベントのランチタイムには「芋煮はコミュニケーションプラットフォーム」と掲げて活動されている全日本芋煮同好会[注1]の皆さんにより、山形風の醤油ベースの芋煮とカレーうどんが振る舞われました。当日は天候が良かったこともあって、屋外で正に芋煮会さながらに、参加いただいた皆さんが和やかな場の中で会話を楽しんでいました。アンカンファレンスの場とはまた違った雰囲気で、新しいつながりが生まれていたようです。

◆写真1　芋煮会会場の様子

◆写真2　芋煮

午後の部：アンカンファレンス

参加者の皆さんには大型の付箋紙に自分が考えているトピックを書いて、それをホワイトボードにどんどん貼り付けてもらいました（**写真3**）。午後は皆さんから出していただいたそれらのトピックを14に絞り込み、AからEの5つのトラックに分かれてアンカファレンス方式でディスカッションが行われました。各トラックにはファシリテーターが付き、トピックごとに参加者からトピックリーダーが選出されて議論が進められていき、最後に各トラックからの発表[注2]が行われました。

Aトラック

Aトラックでは環境というキーワードを軸に、被災者自身による情報、ツールをどうするかについて

◆写真3　続々と集まるトピック

注1　https://www.facebook.com/imonikai
注2　個別セッション発表の動画 URL http://youtu.be/1YekN9CwH7k

議論が行われました。

●被災者自身による情報発信

“マスメディアは命を救うことはできない”という話が出ました。マスメディアのようなトップダウンではなく、ボトムアップによる市民メディアが必要で、その例として女川災害FMの事例が挙がりました。

女川災害FMはローカル発信のメディアとしてTwitterで発信している人たちを集めて発信し、震災時に役立ちました。その反面、個人が発信する情報の信頼性も問題となり、女川災害FMにおいても信頼性に対するリテラシーを高めていくことにチャレンジしていました。

また、女川災害FMからの情報を東北放送が放送するといったこともあり、情報発信のプラットフォームについての議論もありました。

●ツールをどうするか

災害時に人の命を救うようなアプリケーションはどのようなものがあるかを考えました。状況によって、ネットにつながっている場合と、つながらない場合の2つの前提があります。

つながっている場合でも、災害用サービスを使える状況になっているのは2割というデータがあります。緊急地震速報を受信したら自動的に立ち上がるアプリなどを最初から組み込んでおくことが必要とされるのではないかと思われます。

一方つながっていない場合には、まず何とか被災地でのネットワークを構築できないかという話になり、携帯をトランシーバーのように使えるような、Wi-Fiのアドホック通信などの話が出ました。長い距離での通信ができるようにして、避難所間で支援の情報などが共有できると良いのではないかと思います。

●プロジェクト継続

お金の問題、人の問題、体制の問題の3つの軸で議論しました。

お金については、災害といっても普通のビジネスと同じような問題があります。ボランティアにエンジニアはいても、企画や営業の人があまりいなかったのではないでしょうか。そういう人にお金を回すしくみを考えてもらえると良いのではないかという話がありました。

人については、気持ち、情熱、やりがいがどこまで続くかという問題が出てきます。ボランティアでやっていただいたことに対してお金を払うようにした後、そのお金の範囲でしか成果が出てこなくなったという事例もあり、ビジネスとボランティアという相反する中でどう自分の立ち位置を決めるか難しいのではないかと思います。

体制については、プロジェクトの立ち上げ時に合意形成をしないといけない、ということが挙げられました。

Bトラック

Bトラックでは連携というキーワードを軸に、地図情報、プライバシー、団体間連携について議論が行われました。

●地図・地理情報

地図をどう使ったかの振り返りを行いました。災害時、既存の地図は必ずしも使いやすいものとはならず、地図と何をひもづけるか、たとえば物資をどこに送るかなどに活用できるよう、自転車に何かセンサーを付けることで地図作りができるのではないかというアイデアが出ました。

●プライバシー

アーカイブとしていろいろ残っているものの中に、忘れる権利、忘れてほしい、というものもあるのではないかという議論がありました。「プライバシー」という言葉が片仮名であるように、日本ではそもそも対応する概念がないのではないでしょうか。たとえばTwitterのアーカイブを削除する権利もあるのではないかと思います。

客観的なプライバシーと主観的なプライバシーの違いもあり、不安を持たずに個人情報を登録する環境はないかという話になり、個人情報保護法など法整備の問題があるが民間主導型でやれると良いのではという話も出ました。

●団体間連携

「連携したほうがいいよね」と言っているだけではなく、想いがあって動ける人が集まる今日のような場を継続してやっていくことが大切です。顔を突き合わせた関係はオンラインより強いので、ネットだ

けでなくリアルでの飲み会などをやっていきたいということになりました。

Cトラック

Cトラックでは行政と民間の連携、オープンデータ、記録・アーカイブについて話し合われました。

●行政と民間連携

緊急時、復興時の話がありますが今回おもに話したのは緊急時のことで、行政のほうに根本的な思想がないので統一した動きが取れていないのではないかという話がありました。民間のほうもバラバラではなくまとまるべきで、最終的には人間関係が大切だという話になりました。

●オープンデータ

オープンデータと言ってしまうと余計な調整が入ってしまう恐れがあります。災害時公開情報というような言い方にして災害時公開協定というようなものを作り、具体的な事例とセットで設けておくと良いのではないかという話が出ました。

また、行政との間で共同でアイデアソンを行って、お互いの合意を取っていくことができると良いのではないでしょうか。

●記録アーカイブ

基本的に"残すことは良し"という方針で議論しました。利用する際には、いかにしてアーカイブしたものを見やすくするかを考える必要があります。アーカイブの可能性としては、個人が撮ったものも残していけないかと思っているが、個人の写真や顔が入ってしまったものをどう扱うかというスクリーニングが課題となります。

また、ライセンスの問題もあります。例として、気仙沼のリアスアーク美術館ではスクリーニングがされており、個人撮影のものが展示されていて参考になります。

Dトラック

Dトラックでは人材について、ITリテラシーが高くない人への対応をどうするかなどの話をしました。

●IT弱者

おもに高齢のIT機器を使えない人に、どうやって情報を届けるかということを議論しました。通話とメールしかできない人にどうやって届けるか、若者と高齢者がコミュニケーションを取れるようにして、そこから伝えてもらうようにするのが良いのではないでしょうか。その一方で、タブレット機器は高齢の方でも使ってみたいということがあるので、UIを工夫して使ってもらえるようなものを考えていく必要もあると思います。

このテーマはなかなか成功例が見えてこない分野でもありますので、引き続きアイデア出しと実践をやっていかなければいけないと感じています。

●ITベテランの役割

ITベテランとは、ここでは災害を経験し、そこで何らかの活動をした人と定義しています。ハッカソンをどんどん開催し、いざというときに役立てられるよう、その成果をストックしていつでも見られるようにしておくのが良いと思います。実際の災害時にはニーズが合わないことも想定されますが、短時間で何かを仕上げるハッカソンではスピードと忍耐力が鍛えられるため、緊急時にはその忍耐力を活かして物を作っていくことができたらと思います。

そして、そういったイベントに若者をどんどん巻き込んでいくべきです。なぜなら今ここにいるベテランの人は、次に何か起こったときには高齢化してるかもしれないからです。

●復興・IT・若者

ITの教育といっても上からではなく「自走するためのエンジンを身につける」ことが大切です。ほかの人にも教える、伝える、クローズドにせず成果を公開して誰でも使えるものにすることが広がりを産んでいきます。

Eトラック

Eトラックでは復興事業の立ち上げについて、過疎の加速、何を魅力に人は集まるのかなどについて話し合いが行われました。

●プロジェクト立ち上げ

災害発生後10から100時間の間にやっていくことを話しました。避難所のガバナンスの話で、地域にリーダー的な人がいる場合は良いのですが、そう

いう人がいない場合は治安の問題なども発生する確率が高かったようです。

また、避難訓練の際に避難所を立ち上げてITでやることも同時に訓練すると良いのではないかという話もありました。

●復興事業の立ち上げ

被災地の過疎化は実は震災が起こる前から進んでいて、そこに震災が起きてスピードアップしてしまった状況です。そこでどうやってコミュニティを活性化するかということがポイントになります。人を集めるしかけ、しくみづくりが必要ということでいろいろな提案があり、人口を増やすことに成功した徳島の事例を東北と共有すると良いのではないかという話が出ました。

また、コミュニティを再生するにはマネジメントスキルを持つ人が必要で、そういう人をいかにして地元で育て、活躍してもらうかということが必要です（**写真4**）。

その後

2013年10月6日の会議を受けて、10月26日には「減災ソフトウェア開発に関わる一日会議[注3]」が行われました。これは、10月6日の議論を深め、さらに新たな知見を集めることで、災害発生時にどのような対応をとることができるのか、実践的かつ理論的に討議することを目的としたものです。

ここでは発災直後のITの役割やソーシャルメディアの活用ポリシー、被災地で早急にインターネットアクセスを可能にするための技術などについて話されました。たとえば、東日本大震災の際、ソーシャルメディアで情報の拡散を依頼されることがありました。情報によって拡散すべきものとそうでないものがありますし、またより精度の高い情報が後に提供された場合に、拡散した情報をどのように扱うかは検討が必要です。デマが拡散されてしまった事例などを思い出す人もいるでしょう。このように、発災後にどのようにソーシャルメディアを活用するかを事前に明文化しておき、発災とともに、そのポリシーを明示することなどが議論されました。

また、東日本大震災でもITが必ずしも有効に活用されていなかった事実を冷静に見つめなおし、発災後のある段階からより効率的にITを活用するために、もっと組織的に取り組むための話し合い（情報支援レスキュー隊構想）も行われました。

この「減災ソフトウェア開発に関わる一日会議」に見られるように、10月26日の会議は、会議を行うことが目的ではなく、これをきっかけとしてITによる災害への対応を具体的に進めていくことが目的です。そのため、アンカンファレンスで出たアイデアなどを実現していくためのしくみをスタッフ側では用意しようと考えています。具体的には、Wikiを立ちあげ、そこに派生したプロジェクトの進捗や作業メモなどを保管できるようにします。このWikiは発災後の情報ポータルとしても活用可能です。現在、「ITx災害」サイト[注4]はイベントの報告が中心となっていますが、今後、Wiki上で派生したプロジェクトの情報が蓄積されていったならば、それらの紹介を掲載したりすることで、より汎用的にITと災害を考えられるサイトに変更していこうと思います。SD

◆写真4　アンカファレンスの様子

注3　http://gensai.itxsaigai.org/
注4　http://www.itxsaigai.org/

温故知新
IT むかしばなし

第31回

ぴゅう太

LINE株式会社　佐野 裕　SANO Yutaka　Twitter：@sanonosa

はじめに

今回は1980年代に発売されたぴゅう太についてお話をしたいと思います。

ぴゅう太はトミー社（現在タカラトミー社）より発売された16ビットゲームパソコン[注1]です。初代ぴゅう太は1982年に59,800円で発売されました（**表1**）。

トミー社といえばプラレールや黒ひげ危機一髪などが有名な玩具メーカーです。玩具メーカーがゲームパソコンを発売するだけに、完全にホビーユースが想定されていました。他社のパソコンは通常パソコンショップで発売されていたのに対して、ぴゅう太は玩具売り場で発売されていたという点で、他社とは一線を画していました。筆者も当時、玩具売り場の店頭に展示されていたぴゅう太でさまざまなゲームを楽しんだ記憶があります。

8ビットが主流だったこの時代のパソコンの中で、16ビットCPU[注2]を採用することになった経緯は謎です。16ビットだからといって演算能力が特段優れていたわけでもなく、搭載メモリもわずか16KB（16MBや16GBではないです!）程度でしたので、CPUが16ビットである必要があったのか謎は深まるばかりです。

注1）当時はマイコンと呼ばれていました。

注2）Texas Instruments社製。レジスタが3個（プログラムカウンタ、ワークスペースポインタ、ステータスレジスタ）で16ビットでした。それ以外のレジスタは当時はCPUより高速だったメインメモリ上にありました。

▼表1　初代ぴゅう太の仕様

CPU	TMS9995（16bit）
VDP	TMS9918
メモリ	ROM20KB、RAM16KB
キーボード	56キー（ゴムキー）
グラフィック機能	256×192ドット
表示能力	16色
サウンド	擬音4種類/3和音
メディア	ROMカートリッジ
大きさ	370×255×63（mm）
重さ	1.7kg

ROMカートリッジ

ぴゅう太にはさまざまなゲームがROMカートリッジで提供されました。いずれも4,800円で、その中にはフロッガー、スクランブル、プーヤン、ガッタンゴットン、Mr. Do!など、当時ゲームセンターで人気があったゲームの移植作品も含まれます。

ぴゅう太に限らず1980年代のホビー向けパソコンでは市販ソフトをROMカートリッジで提供することが一般的でした。当時主流だった外部記憶媒体はカセットテープでしたが、データレコーダは別売りであることに加え、カセットテープでプログラムを1本読み書きするには数分から数十分かかるため、ゲームマシンとして考えた場合、電源投入後すぐにゲームが楽しめるROMカートリッジが主流になったものと予想されます。

一応カセットテープ媒体のゲームも数種類発売されました。カセットテープは1本のテープにA面とB面があり、それぞれに別のゲームが入って1,000円とROMカートリッジと比べてとても安価な価格

設定でしたが、主流となりませんでした。

日本語BASIC言語搭載

初代ぴゅう太が異色な存在だったのは、日本語BASIC言語「G-BASIC」を搭載していたことです。日本語BASICと言っても本格的な日本語処理に長けていたということではなくて、扱われるコマンドがすべてカタカナになっていました。

プログラミングを学習したことがある人の中には「何でプログラミング言語は英語なんだ。日本語だったら良いのに」と思ったことがある人がきっと大勢いることでしょう。筆者もその1人でした。

しかしぴゅう太の日本語BASICプログラムを見たらその考えが180度変わった記憶があります。率直に言うとプログラムがたいへん見づらいです(**リスト1**)。同様の意見が多かったのか、その後に発売されることになる「ぴゅう太mk2」では英語版G-BASICが採用されることになります。

スプライト機能の搭載

ぴゅう太ではゲームを作りやすくするためにスプライト機能が搭載されていました。スプライト機能とは、背景画面と特定サイズのキャラクターが自然に重なって見えるようにハードウェアが自動的に計算して合成してくれる機能のことです[注3]。アクションゲームを作るのに大変便利な機能ですので、ゲームパソコンとしてこの機能はあって当然の機能だったと推測できます。もしスプライト機能がなかったら、自分で合成処理機能を開発するか、もしくは背景画像とキャラクターが重ならないようなゲームデザインを行う必要があります。

当時はゲームセンターにあったゲームをパソコンでも遊べるよう移植することが求めらていた時代でしたので、このスプライト機能は、自作ゲームを作るのに便利だっただけでなく、アーケードゲームを移植しやすくするという意味でも大変有利な機能でした。

▼リスト1　日本語BASICでのプログラム例

```
10 シキ I=1
20 カケ I
30 シキ I=1+1
40 モシ I=10ナラバ60 ニイケ
50 20 ニイケ
60 オワリ
```

ゲームマシンとしてのぴゅう太

その後「ぴゅう太Jr」という機種が定価19,800円で発売されました。ぴゅう太Jrは完全にゲームマシンとして割り切り、大胆にもG-BASICとキーボードが取り払われました。

続いて発売された「ぴゅう太mk2」では日本語G-BASICの代わりに英語G-BASICが採用され、かつキーボードがゴム製[注4]からプラスチック製となり、定価29,800円で発売されました。

しかしこのころ任天堂社から世界的大ヒットゲームマシン「ファミリーコンピューター(ファミコン)」が発売され、ゲームマシンの主役は完全にファミコンとなり、ゲームパソコンとしてのコンセプトで発売されていたぴゅう太が終焉を迎えることとなりました。

終わりに

1980年台はさまざまなメーカーからいろいろなコンセプトのパソコンが登場していた非常におもしろい時代でした。ビジネス向けの硬派なパソコンもあれば、ぴゅう太のようなゲームマシンに近いパソコンやビジネスとホビーのいずれも意識したパソコンなどいろいろな機種があり、各メーカーの試行錯誤が続いていました。価格帯も3万円～20万円くらいと幅が広くてどのパソコンを買うか選ぶのが楽しい時代とも言えました。

「どんなゲームが遊べるのか」というのはパソコン選びの大きな要素のひとつとなっていました。そういう意味でぴゅう太のゲームパソコンとしての割り切りは、理にかなったものだったと言えそうです。SD

▼写真　初代ぴゅう太

注3) ちょうど画面上に別レイヤがあり、そこにキャラクタを定義しておいて、その座標値だけで移動できるようなしくみです。

注4) 当時は消しゴムキーボードと呼ばれ、ちょうど中心部分を押さないと入力されないことがある使いにくいものでした。

Software

レッドハット、「Red Hat Enterprise Linux OpenStack Platform 4.0」などクラウド管理製品の最新版を提供開始

レッドハット㈱は1月28日、「Red Hat Enterprise Linux OpenStack Platform 4.0」（以下、RHELOP）を一般向けに提供を開始した。また同日、「Red Hat Enterprise Virtualization 3.3」（以下、RHEV）と「Red Hat Cloud Infrastructure 4.0」の提供も開始した。

Red Hat Enterprise Linux OpenStack Platform 4.0

同製品は、OpenStackをベースとしたクラウド基盤を構築するためのプラットフォーム。米Red Hat社によって強化されたOpenStack Havanaコードと、OSである「Red Hat Enterprise Linux 6.5」、KVMベースのハイパーバイザ「Red Hat Enterprise Virtualization Hypervisor」で構成されている。

最新版の4.0では、「Foreman」、「OpenStack Orchestration（Heat）」、「OpenStack Networking（Neutron）」などの機能をフルサポートする。また、「Red Hat CloudForms」や「Red Hat Storage Server」といった同社のインフラストラクチャ製品との統合も進められている。各機能や製品の概要は次のとおり。

- Foreman：物理と仮想両方のインフラリソースのプロビジョニングができるライフサイクル管理ツール
- OpenStack Orchestration（Heat）：広範なインフラリソースのプロビジョニングを迅速に行えるオーケストレーションエンジン
- OpenStack Networking（Neutron）：仮想ネットワークインターフェースカードなどのインターフェースデバイス間のSDN（Software Defined Network）を提供する
- Red Hat CloudForms：複数の仮想化ソリューション、およびRHELOPに対して統一された管理インターフェースを提供する
- Red Hat Storage Server：ソフトウェアによる分散ストレージ基盤をRHELOPに提供し、高いスケーラビリティと可用性を備えたストレージプラットフォーム

Red Hat Enterprise Virtualization 3.3

同製品は、ホストOSとゲストOSの管理コンソール「Red Hat Enterprise Virtualizationマネージャ」とハイパーバイザ「Red Hat Enterprise Virtualization Hypervisor」から構成される製品。データセンターなどの大規模な仮想化環境を効率よく構築／管理できる。

最新版で強化された点は次のとおり。

- Red Hat Enterprise Virtualizationマネージャをホスト上の仮想マシンとして配備できるようになり、環境構築に必要なハードウェアを減らせる
- バックアップ&リストア用のAPIが追加され、サードパーティのソフトウェアで仮想マシンのバックアップ／リストアを行えるようになった
- OpenStack GlanceとOpenStack Neutronのサポートにより、プライベートクラウドとデータセンター仮想化の間で共通のインフラストラクチャを使用可能になった。これにより仮想マシンテンプレートの保存や、先進的なネットワークを構成できる

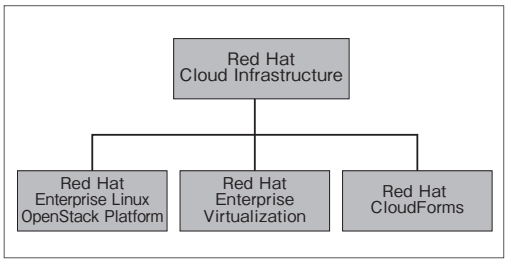
Red Hat Cloud Infrastructure 4.0

同製品は、先に紹介したOpenStackベースの仮想環境構築プラットフォームRHELOPと、データセンター仮想化プラットフォームRHEV、そして複数の異なる仮想環境で統一された管理インターフェースを提供する管理ツール「Red Hat CloudForms」とを合わせて提供するもの。RHELOPとRHEVのバージョンアップに伴い、同製品も4.0へとアップデートされた。

同社は、同製品の利用により、RHEV、VMware vSphere、Microsoft Hyper-Vを含む異種混合のクラウド環境を管理できるほか、SDN技術との統合を通してより高度なネットワーク方式を利用できるとしている。

RHEVとRHELOPは、それぞれNeutronとGlanceを通してネットワークとイメージライブラリのサービスを共有しているため、ユーザは1つのストレージリポジトリからOpenStackイメージの作成、編集、実行を行えるという。

同製品の最新版は、既存のユーザの場合、Red Hat Cloud Infrastructureのサブスクリプションを通して、すでに利用可能。



▲Red Hat Cloud Infrastructureの構成

CONTACT レッドハット㈱
URL http://jp.redhat.com/

Hardware

日本ヒューレット・パッカード、“自働サーバ”「HP ProLiant」の新モデルを５製品を発売

日本ヒューレット・パッカード㈱は１月30日、“自働サーバ”「HP ProLiantサーバGeneration 8」に、最新のインテルXeonプロセッサ「E5-2400 v2」製品ファミリーを搭載した５つの新モデルを発売した。

同製品は、リモート管理機能、電力管理／制御機能に加え、サーバ導入時間を大幅に短縮する「HP Intelligent Provisioning」、OSに依存しないエージェントレス監視機能、すべてのログを記録する「HP Active Health System」など、システム管理者の運用工数削減に寄与する新機能を標準で提供する製品ファミリー。

今回発表されたのは次の５モデル。

▼新製品のラインアップと価格

製品名	希望小売価格（税抜）
HP ProLiant ML350e Gen8 v2	168,000 円～
HP ProLiant DL360e Gen8	237,000 円～
HP ProLiant DL380e Gen8	344,000 円～
HP ProLiant BL420c Gen8	228,000 円～
HP ProLiant SL4540 Gen8	1,044,000 円～

これらの製品は、従来製品と比較して最大25％の性能向上を実現したほか、DL360e Gen8、DL380e Gen8は今回から80 Plus Titanium電源をサポートし、最大96％の交流／直流変換効率による電力削減を実現している。

同時に、期間限定で「HPサーバへのりかえ割キャンペーン」を実施している。詳細は次のとおり。

HPサーバへのりかえ割キャンペーン

期間：2014年１月16日～2014年４月30日

内容：最新のインテルXeonプロセッサ E5-2400 v2/E5-2600 v2製品ファミリーを搭載したHP ProLiant Gen8サーバとオプションを、30％引きの特別価格で提供する。HP OEM版のWindows Server OSを同時に購入すると、追加で5％引きの35％引きで提供する

CONTACT 日本ヒューレット・パッカード㈱
URL http://www.hp.com/jp/

Hardware

日本オラクル、高速データベースマシンの最新版「Oracle Exadata Database Machine X4」を提供開始

日本オラクル㈱は、最新世代の高速データベースマシン「Oracle Exadata Database Machine X4」の国内提供を１月21日より開始した。

前世代の製品からハードウェアおよびソフトウェアが刷新され、性能の向上、容量の拡大、データベース展開の効率性およびサービス品質の向上が図られている。同社調べによると、性能は50～100%向上し、ストレージ容量は33～100%拡大したとのこと。

「Oracle Database 12c」の新機能であるマルチテナントアーキテクチャを利用することで、データベース集約における高いリソース効率と運用工数の削減を可能にした。さらに「Oracle Enterprise Manager 12c」のクラウドサービス機能に含まれるデータベースサービスのプロビジョニングを始めとした管理機能を利用することで、自社内におけるDBaaSを迅速かつ容易に実現できるようになった。

同製品の参考価格（税抜）は2,390万円～。

「Oracle Exadata Database Machine X4」の強化点

- 何百ものデータベースをDBaaSアーキテクチャとして単一のラックに統合でき、コストを抑えながらデータベースサービス作成の俊敏性を向上させる
- 物理フラッシュの大容量化と独自の超高速フラッシュ圧縮技術により、有効フラッシュメモリ容量を４倍に拡大し、OLTP処理が大幅に高速化。単一ラックの「Oracle Exadata」のデータスループットは100GB／秒
- データウェアハウスで一般的なテーブルおよびパーティションのスキャン処理に焦点を当てた新しいFlash Cachingアルゴリズムにより、データウェアハウジング処理の性能が向上

▲Oracle Exadata Database Machine X4

CONTACT 日本オラクル㈱
URL http://www.oracle.com/

Hardware

トランセンドジャパン、デュアルUSBメモリ「JetFlash 380」を発売

トランセンドジャパン㈱は、マイクロUSB端子が付いたデュアルUSBメモリ「JetFlash 380」を発売した。

JetFlash 380は標準USBインターフェースでPCに接続できるだけでなく、マイクロUSBインターフェースでUSB OTG対応のモバイル機器にも接続できる。製品は8GB、16GB、32GBの容量とゴールド、シルバーの2色を用意している。

本製品と一緒に使うと便利な専用アプリ「Transcend Elite App」も提供されている。Google Playから無料ダウンロードでき、アプリをインストールしたAndroid機器に同製品を接続すると自動で起動する。当アプリを利用することで、モバイルデバイスでもファイルの検索やバックアップが簡単に行える。Android 4.0以降の端末で利用可能。

▲JetFlash 380

価格はオープン。実勢想定価格（税込）は、8GBが1,580円、16GBが2,480円、32GBが4,880円。

CONTACT トランセンドジャパン㈱
URL http://jp.transcend-info.com/

Hardware

NEC、NECアクセステクニカ、最大約24時間の連続通信が可能なLTEモバイルルータ「AtermMR03LN」を発売

NECとNECアクセステクニカは、LTEモバイルルータとしてBluetoothテザリング機能により、最大約24時間の長時間連続通信を実現した「AtermMR03LN」を2月1日より発売した。価格はオープン。

㈱NTTドコモが提供するクアッドバンドLTEサービスを利用した通信サービス「BIGLOBE LTE・3G」「@nifty do LTE」「ぷららモバイルLTE」に対応するLTEモバイルルータとして、各ISPより順次販売される。

同製品はWi-Fiの最新規格IEEE802.11acへの対応による高速通信（最大433Mbps）とともに、消費電力の少ないBluetoothを用いたテザリング機能による長時間使用も可能としている。特徴は次のとおり。

- Bluetoothテザリングによる長時間利用を実現
- クアッドバンドLTEと11ac対応による高速通信
- 持ち運びやすい小型／軽量を実現
- 6色のカラータイルから選べるタッチパネルカラーディスプレイ
- ギガビットイーサ対応のクレードル

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

Hardware

東プレ、REALFORCEの英語配列キーボード3機種を発売中

東プレ㈱は、人気キーボード「REALFORCE」シリーズの新製品として、2013年夏に3機種を発売している。

そのうちの1つが「REALFORCE104UG－HiPro」。これは、本誌今月号の読者プレゼントになっている「108UG-HiPro」（写真）の英語配列版。キートップ天面の形状やキートップの高さなどのプロフェッショナル仕様はそのままに、キーレイアウトは英語104配列を採用している。インターフェースはUSB。店頭価格は27,800円。

そのほか2機種は海外では定着し、日本発売のリクエストが多数寄せられた英語87配列キーボードの白と黒の2機種。白が「REALFORCE87UW」で、黒が「REALFORCE87UB」。ともにインターフェースはUSB。店頭価格は21,800円。


▲REALFORCE108UG-HiPro

CONTACT 東プレ㈱
URL http://www.topre.co.jp/

Hardware

トレンドマイクロ、スタンドアロン向けのウィルス検索／駆除ツール新製品「Trend Micro Portable Security 2」を提供開始

トレンドマイクロ㈱は、スタンドアロンやクローズド環境向けのウィルス検索／駆除ツールの新製品「Trend Micro Portable Security 2」を、1月27日より受注開始した。

同製品はUSBメモリ型の製品検索ツールで、インターネットに接続されていなくても、USBインターフェースを使って手軽にウィルスの検索や駆除が行える。

インターネットに接続されていないスタンドアロン／クローズド環境は、安心／安全と考えられがちだが、実際にはUSBメモリや持ち込みPCを通じて、ウィルス感染の脅威に晒されている。そんな場合でも、同製品を使うことでセキュリティ対策を行える。同製品の特徴は次のとおり。

- 管理プログラムの集中管理機能により、各拠点の検索ツールの検索ログを一元的に把握できる。検索ツールのパターンファイルのアップデートや各種設定を一括で行うことも可能
- 検索ツールに搭載されたLEDにより、検索ステータスと結果を3段階（青色：検出なし、黄色：検出／駆除済み、赤色：検出／駆除処理待ち、検索中はLEDが点滅）で通知する
- 管理プログラムを使用しないモードを選択すれば、検索ツールとインターネット接続された汎用端末を用いてパターンファイルのアップデートやログの閲覧が可能

参考標準価格（税別）は28,800円（1年間のスタンダードサポートサービス料金、ハードウェア保証を含む）。新規購入時の最低購入本数は5本。検索ツール1本で複数端末のウィルスチェックが可能。

また、2月17日～5月16日の間、同製品の新規購入者を対象に、20％OFFの価格で提供するキャンペーンを実施している。

▲Trend Micro Portable Security 2

CONTACT トレンドマイクロ㈱
URL http://www.trendmicro.co.jp

Software

グレープシティ、ComponentOne Studioシリーズの新エディション「ComponentOne Studio for WinRT XAML」を発売

グレープシティ㈱は、アプリケーション開発に便利なコンポーネントを多数収録したスイート製品「ComponentOne Studio」シリーズの新エディション「ComponentOne Studio for WinRT XAML」（以下、WinRT XAML）を1月30日に発売した。また、同エディションを加えた「ComponentOne Studio 2013J v3」も合わせて発売した。

ComponentOne Studio

幅広いプラットフォームで業務システムを効率的に開発できるコンポーネントセット製品のシリーズ。各開発プラットフォームに対応した5つのエディション（WinForms、ASP.NET Wijmo、WPF、Silverlightと新発売のWinRT XAML）とそれらすべてを収録した「Studio Enterprise」、「Ultimate」という上位製品がある。どのエディションにもデータグリッドや帳票、チャート、UI部品などをバランスよく収録している。

ComponentOne Studio for WinRT XAML

このたび、同シリーズに加わった新エディション。Visual StudioおよびExpression Blendで、あらゆるジャンルのアプリケーションを開発できるWindowsストアアプリ用コンポーネントセット。FlexGrid、チャート、PDFビューワ、Excel出力などのコンポーネントを24種収録している。収録されているすべてのコントロールは、Windows RTでも実行可能。

価格

ComponentOne Studioシリーズはサブスクリプション方式で販売しており、初回費用（税別）は「Studio Enterprise」が150,000円、WinRT XAML単体が100,000円。1年単位の更新費用はそれぞれ初回費用の40%となる。

サブスクリプション契約期間内の「Studio Enterprise」、「Ultimate」のユーザに対しては、WinRT XAMLは「ComponentOne Studio 2013J」のメジャーリリースとして無償で提供される。

CONTACT グレープシティ㈱
URL http://www.grapecity.com/tools/

Topic

The Linux Foundation、「LinuxCon Japan 2014」など、Linuxイベントの発表者募集を開始

The Linux Foundationは、5月20日～22日に椿山荘会議センターで開催する「LinuxCon Japan 2014」と「CloudOpen Japan 2014」（同時開催）の発表者募集（CFP：Call for Participation）を開始した。また、7月1日～2日に同会場で開催する「Automotive Linux Summit 2014」のCFPも開始した。発表案は同団体のWebサイトにて3月14日まで受け付けている。

Linux Con Japan 2014

日本、アジア地域の最大のLinuxカンファレンス。世界中のトップクラスの開発者が集結する。各種プレゼンテーション、チュートリアル、BOF（分科会）、基調講演、ミニサミットなどが用意されている。

CFP推奨トピック：

Linuxカーネル開発の進化／オープンクラウド、仮想化、および分散サービス／Linuxによる企業基盤の最適化／Linuxシステム管理＆セキュリティほか

CloudOpen Japan 2014

クラウド構築に関わるOSSプロジェクト、テクノロジ、企業などについて支援や検討を行うカンファレンス。昨今のクラウドやビッグデータエコシステムを牽引しているOSSプロジェクト、製品、企業のほか、伝統的なOSS分野のベストプラクティスが集結する。

CFP推奨トピック：

Linux／KVM／Xen／Hadoop／Puppet／Chef／Gluster／Devops／ビッグデータほか

Automotive Linux Summit 2014

自動車システムエンジニア、Linuxエキスパート、R&Dマネージャー、ビジネスエグゼクティブ、OSSライセンスやコンプライアンスのスペシャリスト、Linuxコミュニティの開発者などが集結。世界中のトップクラスの講演者陣が登壇し、数々の革新的なプログラムコンテンツが提供される。

CFP推奨トピック：

Linux車載システム／ミッションクリティカル向けのLinux／グラフィック、暗視スコープ、拡張現実ほか

CONTACT The Linux Foundation
URL www.linuxfoundation.jp

Service

日本マイクロソフト、Windows XP/Office 2003からWindows 8.1/Office 365への移行促進キャンペーンを実施

日本マイクロソフト㈱は、Windows XPおよびOffice 2003を利用中の中小企業、公共機関、医療機関を対象に、Windows 8.1やOffice 365などの最新PC環境へ移行する際のライセンス価格を最大25%割引で提供するキャンペーンを2月1日～4月30日の期間限定で実施している。

Windows XPとOffice 2003は、同社の「サポートライフサイクルポリシー」に基づくサポートの提供を4月9日（日本時間）に終了する。サポート終了後は、セキュリティ上の脅威に対応することが困難になることから、同社はパートナー各社と連携しながら本キャンペーンを活用した最新環境への移行を推進している。

キャンペーン概要

①ビジネスに最適な、良いとこ取りOS Windows 8.1移行促進キャンペーン

対象期間：2014年2月1日～4月30日
対象：PC台数が250台未満の中小企業、公共機関、医療機関
概要：Windows XPなど対象OSからWindows 8.1 Proへのアップグレードライセンス価格を20%割引する
Webサイト：http://aka.ms/win81cp

②Office 365への移行促進キャンペーン

対象期間：2014年2月1日～4月30日
対象：PC台数が250台未満の中小企業、公共機関、医療機関
概要：クラウドサービスOffice 365のライセンス価格を単年購入時に20%、複数年一括購入時に25%割引する
Webサイト：http://aka.ms/365cpn

最新環境への移行費用について、今年度（2014年3月末日まで）の予算化が難しいユーザ向けに「PC購入支援キャンペーン」も3月末まで提供している。これにより金利ゼロでPCの調達と導入を今年度内に終了し、来年度予算で費用を支払うことが可能になる。

CONTACT 日本マイクロソフト㈱
URL http://www.microsoft.com/ja-jp/

Topic

Red Hat、CentOS開発において、CentOSプロジェクトとの協業を発表

米Red Hat社は1月15日、CentOSの開発においてCentOSプロジェクトと協力していくことを発表した。

CentOSはRed Hat Enterprise Linux（RHEL）のコードをベースにして開発されているLinuxディストリビューション。その安定性の高さから多くの企業、サービスで利用されている。今後、同社はCentOSプロジェクトに対して、さまざまなリソースや専門知識を提供していく。こうした協力関係を築くことで、CentOSとRHEL双方の認知度向上させ、ひいてはRHELの需要増大につなげようというのが同社の狙い。

同社は「CentOSはRHELのコードをベースとしているが、ビルド環境が違ったり、一部で異なるカーネルやOSSコンポーネントが含まれていたりすることから、両者は完全に同じというわけではない」と主張する。RHELは主要なメーカーと提携して多くのハードウェアでテストを行い動作認定も提供しているため、「最新のイノベーションを利用、体験、実験したい場合にはCentOSを選択し、本番環境での利用や、プロフェッショナル向けの認定とサポートを望む場合にはRHELを選択するのが適切」としている。

CONTACT レッドハット㈱
URL http://jp.redhat.com/

Software

Parallels、MacとWindowsアプリをiPadから利用可能にする「Parallels Access」の新バージョンを発表

米Parallels社は、iPadからWindowsおよびMacアプリケーションにリモートアクセスし、それらをiPad専用アプリのように使用可能にするiPad向けアプリ「Parallels Access for iPad」（以下、Parallels Access）に日本語対応を含む新機能を追加搭載した、新バージョン1.1を発表した。1月28日より提供を開始している。

新バージョンでは、これまでの英語およびドイツ語に加え、新たに日本語を含む11言語に対応した。

また本バージョンより、Windows PCの正式サポートが開始される（これにより対応OSは、Windows 7/8/8.1、Mac OS X Lion、Mountain Lion、Mavericksとなる）とともに、企業ネットワーク内など特定の環境下でのユーザ作業を円滑化するシングルポート接続も使用可能となった。

さらに、本バージョンから新しい価格体系が導入された。ユーザは1ヵ月間450円、または1年間4,300円のいずれかのサブスクリプションを選び、Parallels Accessを最大10台のWindows PC、またはMac PCで利用できる。

CONTACT パラレルス㈱
URL http://www.parallels.com/jp

Software

アシスト、列指向データベース「InfiniDB 4」をリリース

㈱アシストは、米Calpont社が提供する大量データ分析に特化したデータベース「Calpont InfiniDB」の新バージョン「InfiniDB 4」の提供を開始した。

InfiniDBは、ビッグデータの分析処理をシンプルかつ高速に実現する列指向型データベース。データ量の増加にスケーラブルに対応し、またチューニングレス（インデックス不要）にもかかわらず安定したパフォーマンスが得られる。最新版のInfiniDB 4では、次の点が強化された。

- 分析関数として知られる「ウィンドウ関数」のサポートにより、分析作業の処理効率がさらに向上した
- ネットワークトラフィックを従来バージョンの2分の1以下に圧縮したことで、これまで以上にスムーズで安定したデータ処理環境を提供する
- 物理的に増え続けるデータ量をさまざまな単位（データベース、スキーマ、表、列）で簡単にレポートでき、将来的にどのくらいのストレージが必要であるかを予測できる

CONTACT ㈱アシスト
URL http://www.ashisuto.co.jp/

Letters from Readers

家電も狙われる時代!?

脆弱性の発見やWeb改ざんなど、セキュリティ関連のニュースは日々、途絶えることがありませんが、最近で一番衝撃的だったニュースは「冷蔵庫がスパムメールを大量送信した」というもの（結局は誤報だったようです）。冷蔵庫がメールを送信するなんて、一見、人間的で微笑ましい感じがするものの、いずれ家電にまでウィルス対策ソフトを入れないといけないと思うと、やっぱり嫌ですね。

2014年1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 あなたの好きなシェルは何ですか?

UNIX系OSを使う場合、私たちは常にシェルを通して作業を行います。シェルにもsh、ash、dash、bash、ksh、csh、tcsh、zshといろいろあります。本特集ではこれらのシェルの特徴を整理しました。また、コマンドやパイプ、変数などシェルの基礎となる概念やしくみについて解説しました。

シェルスクリプトは、使いこなせると便利だよなあ、といつも思っていますが、なかなか……。今回の特集を参考に今一度トライしたいと思います。

神奈川県／kumaaさん

縁の下の力持ち的な位置づけで、普段はごく一部の機能しか使っていない。そんなシェルのことを深く掘り下げる、良い企画でした。最も身近なツールなのだから、もっと活用しなければ。

神奈川県／hiroさん

シェルといえば、やっぱりshかな……。だって、自分のキャパシティが小さくて覚えきれないから（笑）

埼玉県／南雲さん

役立つが、最新情報や目新しい情報はなく、雑誌ではなく書籍で読むのが良いと感じる。

東京都／IT企業経営者さん

「好きなシェルは何ですか、と訊かれても……、とくに気にせずに、OSからデフォルトで提供されているシェルを使っています」という人も多いでしょう。しかし、今後は端末を使うときの作業効率や、シェルスクリプトの移植性などを考慮して、最適なシェルを選んでみてはいかがでしょうか?

第2特集 10ギガビットを実現するケーブリング技術

クラウドコンピューティングの普及により、ネットワークの広帯域／高速化が求められています。本特集では10GBASE-Tや光ファイバを使ったネットワーク環境を導入する際に気をつけるべき点、ケーブル敷設方法や、エアフローも考慮したケーブリングについて取り上げました。

10GBASE-Tは、普及期だと思うのでちょうど良かったです。1年ちょっと前は、対応スイッチが少なくて導入を断念しました。

東京都／hiddenさん

1本のケーブル内を流れる情報の量が、増えれば増えるほどノイズなどの影響を受けやすくなるので、確かにこれからは大切だと感じました。

長崎県／romeosheartさん

こういうハードウェアの技術も興味があるけれども、知識があまりなかったので良かったです。

石川県／Keiさん

考えたことはなかったが、これほどまで深い話があるとは驚き。

東京都／藤田さん

聞きなれない用語も多く、やや専門的な内容でしたが、いかがでしたでしょうか?　機器の吸気や排気を考慮したラック内／ラック間のケーブルマネジメントも紹介しましたが、これらは10GBASE-Tや光ファイバに限らず活用できるアイデアです。業務でラックを組むときには、参考にしてみてください。

特別企画 ソーシャルゲームのDevOpsを支える技術（後編）

2013年11月号で掲載した記事の後編です。今回は、開発者サイドからDevOpsの取り組みについて、クラウド事業者サイドから膨大な負荷をさばくためのインフラ作りについて、それぞれのノウハウを紹介しました。

技術よりも、開発者の意識、コンセプト

をぶらさないという姿勢に興味を覚えました。

長野県／清水さん

技術者に「なのは」ファンは多そう。ただ、ハードウェアにも人にも負荷が高そうなシステムだ。

神奈川県／miffさん

半端ない負荷に対応していると知りました。

奈良県／管理されるのは嫌いさん

読者のみなさんの感想には、有名なソーシャルゲームの裏側を知ることができて良かったという声とともに、運用がたいへんそうだという同情も多かったです。

特別付録 情報安全護符シールVer2.0

2013年1月号で好評だった情報安全護符シールを、2014年も付録として付けしました。今回も、京都府嵐山の法輪寺に供養していただいた本格的な護符シールです。

真面目に供養しているのが珍しい。

神奈川県／高畑さん

年賀状に貼って人に送ったつもりが、気がついたら2013年のものを貼って送ってました。

滋賀県／田中さん

こういう神社があること自体知りませんでした。

青森県／神さん

バグの発生を抑止する護符もほしい、というご要望も、けっこう多くいただきました。日ごろのみなさんが、どれだけバグに悩まされているのか……その苦労が垣間見れました。

連載

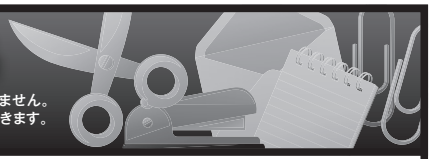
先日、久しぶりにサーバラックを組み直す機会があったので、「自宅ラックのすすめ」が非常に参考になった。

東京都／ひろし@巣鴨さん

1月号の当連載は、第2特集「10ギガビットを実現するケーブリング技術」に呼応するかのように、ケーブリングが主題でした。ぜひ、それぞれの記事を読み返して、ケーブリングの技術を磨いてください。

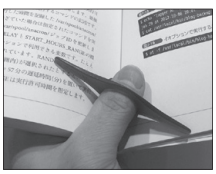
エンジニアの能率を高める一品

仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。



サムシング

350円（税抜）／丸善㈱　https://www.maruzen.co.jp/



▲写真2 斜めにあてがうとラク

「電車通勤時に楽に本を読みたい」そんな願いを叶えてくれるのが、読書補助具「サムシング」。片手だけでしっかり本を広げられる一品です（さすがにページをめくるときは、もう片方の手が必要です）。同製品のWebサイトなどで紹介されている使用例は**写真1**のような感じですが、ここまで深くまっすぐに親指を突っ込むとページがめくりにくいうえに、親指が疲れます。**写真2**のように軽く斜めにあてがう程度で十分です。これならページも比較的めくりやすいです。片手読書で親指がつらくなるのは、本の終盤に差し掛かったとき。終盤になるほど本は自然に閉じようとするので、親指の負担が増します。そのときに本製品があると、最後まで楽に読めます。本の終盤にきたときにだけ使うというのもアリですね。

▲写真1 基本的な使い方

祝 1月号のプレゼント当選者は、次の皆さまです

①Olasonic TW-BT5 ..... 長野県　池上圭二様
②弥生会計14スタンダード ..... 埼玉県　小堀大介様

★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
April 2014

2014年4月号
定価1,280円　176ページ

3月18日発売

［第1特集］Java、JavaScript、PHP言語別で考える

なぜMVCモデルは誤解されるのか?

Web開発においてMVCモデルをベースに開発を行うことが定番となっています。しかしながらMVCモデルを正しく理解して使っている方は多いとは言えず、思い込みのうえで誤った設計・実装がよく見られ、議論の対象になっています。

本特集は各言語別にMVCモデルを復習し、正しい理解ができるようになることを目標としています。開発の効率を向上させ、デバッグやメンテナンスにおいても見通しをよくするMVCモデルをきちんとマスターしてみましょう!

［第2特集］ネットワークエンジニア養成

ロードバランサーの教科書

ネットワークの負荷分散を行うロードバランサーの基礎を解説します。ハードウェア・ソフトウェアの両面から解説をすることで、高い視点からネットワークを理解できるようになります。

［特別企画］今すぐ知りたいSIMのしくみ
［新 連 載］シェルスクリプトではじめるAWS入門

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「ハイパーバイザの作り方」（第18回）、「Debian Hot Topics」（第13回）は都合によりお休みいたします。

SD Staff Room

●1月に2冊を作るという年末進行の怒濤の混乱を終えたのち、感染性胃腸炎にかかり地獄を見た。狂乱の全社忘年会には参加できず、翌日の納会も中座。およそ3Kgの減量になるも、正月の宴会で元にもどる。リバウンドしなかっただけいいじゃないか。次回、「黄金週間進行」まだ原稿は来ない。（本）

●50歳を越えてからというものの、体調が変化してきているのがよくわかる。昔はもっと頑張れたと思うが、集中力が続かない。健康診断も近く、以前は直近にはダイエットやら運動やらしていたが、今年は成り行きでもいいかという感じ。日々楽しくないと長続きしないと実感。それでも2週間くらいは禁酒します。（幕）

●今年も行って参りました次世代ワールドホビーフェア。年末に痛めたヒザを抱え、入場まで約3時間立ちっぱなしで並んだのはつらかった。バンダイブースからヨーヨーがなくなってしまい寂しい限り。短い冬になるといいなぁ。今冬、小学生男子にとっては「バディファイト」が熱いようです（我が家調べ）。（キ）

●高齢者が餅を喉につまらせて病院に運ばれる事件が多いので、妻に「あなたのおばあちゃんにも気をつけてあげないとね」と言ったところ、妻曰く「うちは父親にも気をつけないといけないのよ」とのこと。そうか、自分たちの親も、もう高齢者の年代に入りつつあるのだった。うーむ、笑えない。（よし）

●また健康診断です。例年通り激太りしてます。去年はフレンチ惣菜屋の売れ残りを相当もらって食べたし、内臓脂肪もドエライ事になっていそう。残り1か月、猛ダイエット中ですが、夜を軽くしている為、空腹で4時に目覚め、仕方なくスープを飲んで再度寝てます。これでいいのか?!（ブタえもん）

●平日に風邪をひいて休んだ時のこと。食欲はあったのでお昼どうしようかと思い台所へ行ったが、食材を前日に使い切ってしまっていたのでない。近くにスーパーもあるが、買いに行く気力はなく、保存用に買っていたカップ麺を食べました。実家では食に困ることがなかったのでありがたみを再実感しました。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※ FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2014年3月号

発行日
2014年3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。