Special Feature 1 プログラミング力向上の秘訣
Special Feature 2 新ストレージ活用

Software Design

[ソフトウェア デザイン]
OSとネットワーク、
IT環境を支える
エンジニアの総合誌

2016 April 4

2016年4月18日発行
毎月1回18日発行
通巻372号
（発刊306号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,220円 +税

今すぐ実践できる 良いプログラムの書き方

やればできる！
ワンランク上の
プログラミング

Special Feature 1

きれいなコード
モダンなコード
が書きたい

C
Java
C#
Ruby
JavaScript

Special Feature 2

データはどこに
保存してますか？

オブジェクトストレージの教科書

基礎の基礎と使いどころ完全解説

Extra Feature

LANケーブリングの
次はこれだ！

光ファイバ＋ラック選定の極意

日本で運用と開発は
コラボできるか？

DevOps座談会

2016
April
4
Software Design

第1特集

やればできる!
ワンランク上のプログラミング

今すぐ実践できる 良いプログラムの書き方

きれいなコード
モダンなコード
が書きたい

C / Java / C# / Ruby / JavaScript

contents

2016
April
4
Software Design

contents

第2特集

オブジェクトストレージの教科書

OSSと3つの製品事例から学ぶ、新しいデータ管理のしくみ 063

特別企画

Catch up trend

アラカルト

Column

Development

OS/Network

［ロゴデザイン］
デザイン集合ゼブラ＋坂井 哲也
［表紙デザイン］
藤井 耕志（Re:D）
［表紙写真］
Life On White /gettyimages
［イラスト］
フクモトミホ
［本文デザイン］
＊安達 恵美子
＊石田 昌治（マップス）
＊岩井 栄子
＊ごぼうデザイン事務所
＊SeaGrape
＊轟木 亜紀子、阿保 裕美、佐藤 みどり（トップスタジオデザイン室）
＊伊勢 歩、横山 慎昌（BUCH＋）
＊藤井 耕志（Re:D）
＊森井 一三

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。

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

PDF電子版
Gihyo Digital
Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読（12回）

14,880円（税込み、送料無料）1冊あたり1,240円（6%割引）

PDF電子版の購入については

Software Design ホームページ
http://gihyo.jp/magazine/SD

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

- 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。

※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。

Fujisan.co.jpからのお申し込み方法

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）

Pocket Reference

小さくても、中身充実！

「あれ何だったっけ？」
「こんなことできないかな？」
というときに、すぐに調べられる
機能引きリファレンス。
軽くてハンディなボディに
密度の濃い内容がギュッと凝縮！
関連項目への参照ページもあって、
検索性もバツグン！

片渕彼富 著／山田祥寛 監修
四六判／512ページ
定価（本体2780円+税）
ISBN978-4-7741-7984-1

宮前竜也　著
四六判／224ページ
定価（本体2180円+税）
ISBN978-4-7741-7740-3

沓名亮典 著
四六判／608ページ
定価（本体2380円+税）
ISBN978-4-7741-7404-4

石田つばさ 著
四六判／448ページ
定価（本体2180円+税）
ISBN978-4-7741-4836-6

土井毅／高江賢／飯島聡／高尾哲朗 著　山田祥寛 監修
四六判／488ページ
定価（本体2580円+税）
ISBN978-4-7741-4948-6

高橋晶／安藤敏彦／一戸優介／植田真矢／道化師／湯朝剛介 著
四六判／544ページ
定価（本体2980円+税）
ISBN978-4-7741-7408-2

朝井淳 著
四六判／640ページ
定価（本体1980円+税）
ISBN978-4-7741-3835-0

大垣靖男 著
四六判／648ページ
定価（本体2580円+税）
ISBN978-4-7741-7229-3

岡本隆史／武田健太郎／相良幸範 著
四六判／272ページ
定価（本体2480円+税）
ISBN978-4-7741-5184-7

鶴長鎮一／馬場俊彰 著
四六判／400ページ
定価（本体2780円+税）
ISBN978-4-7741-7633-8

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

忽ち増刷!!

平成28年度【春期】【秋期】

情報セキュリティマネジメント合格教本

岡嶋裕史　著／A5判／424ページ
定価（本体1,880円＋税）　CD-ROM×1枚
ISBN978-4-7741-7869-1

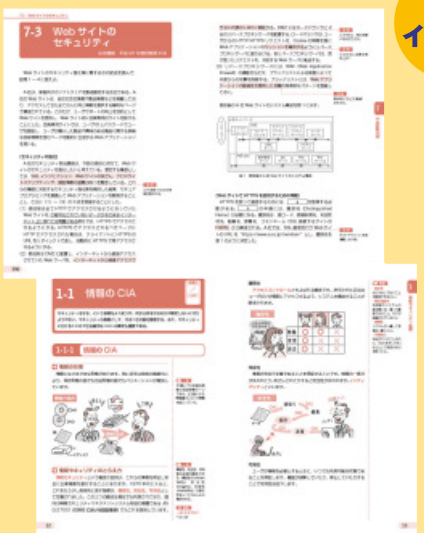
- NHKでおなじみ! 情報セキュリティの専門家が執筆!
- 初心者にもわかりやすい イラスト多用の紙面!
- 長文問題対応／予測問題CD-ROM付き!

2016年新設の注目の試験の参考書が登場!

情報セキュリティの第一人者、 岡嶋裕史先生による試験範囲を網羅した詳細な解説と豊富なイラストで、最新のセキュリティの技術と考え方をバランスよく学べます。午前問題対策はもちろん、午後問題対策の項を設け、長文問題にも対応しました。付録のCD-ROMには午前問題の4回分の予想問題を収録しました。これ1冊で試験対策は完璧です！

本誌の目次

本文イメージ



問題演習ソフトDEKIDAS-2収録

- 学習進度に合わせ、過去問題モード、ジャンル別モード、模擬試験モードが選択でき、ボタン1つで自動採点可能。
- 得意／不得意分野がわかるレーダーチャート表示。
- 「過去に間違えた問題」のみの出題も可能。

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

第88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。

TensorFlow

Google 製の機械学習ライブラリ「TensorFlow」

「TensorFlow」はGoogleが開発した機械学習ライブラリです。もともとは自社サービスのために開発が進められてきたものですが、2015年11月にオープンソースソフトウェアとして一般に公開されたことで大きな話題を呼びました。

近年、機械学習や深層学習（ディープラーニング）といった技術が注目を集めるようになってきました。機械学習とは、簡単に言ってしまえば人間が自然に行っているような学習能力をコンピュータで実現しようとする技術です。深層学習は機械学習の手法の1つで、ニューラルネットワークと呼ばれる人間の神経を模した多層化されたネットワーク構造に膨大なデータを与えて“学習”させることによって、より人間の思考に近い解析結果を導けるようにするものです。

機械学習の具体的な応用分野としては、たとえば写真の自動認識機能が挙げられます。機械学習エンジンに大量の写真データを読み込ませて学習させることで、写真内から数字や文字を読み取ったり、被写体となっている人物や物の種類などを人間と同じように認識できるようになります。

Googleでは近年この技術の開発に力を入れており、すでに写真認識だけでなく、音声認識や機械翻訳、Web検索結果の最適化、メールの分別、広告事業など、あらゆるプロダクトを支える基盤技術として導入を進めているとのことです。それらの実績をもとに洗練されてきたソフトウェア群一式を、一般の開発者が利用できる形にライブラリ化したものがTensorFlowです。したがって、これを利用すればGoogleの各種サービスが持つのと同等の学習機能が、自前のソフトウェアでも実現できるということになります。

TensorFlow の特徴

TensorFlowはApache 2.0ライセンスで公開されています。おもな特徴としては次のような項目が挙げられます。

- **フレキシビリティ**
  計算する内容をデータフローグラフとして表すことができれば、どのようなデータでも処理できる。簡易かつ柔軟な記法によって複雑なモデルを表現できるほか、自前のハイレベルなライブラリを TensorFlow の上に構築することも可能
- **ポータビリティ**
  CPU でも GPU でも動作するように設計されており、モバイルデバイスからデスクトップ、データセンター上のサーバまで幅広い環境にデプロイすることが可能。Docker コンテナ上での動作もサポートされている
- **多様な応用分野**
  科学技術の研究からプロダクションレベルのサービスまで、幅広い分野で利用することができる。これは、研究レベルのアイデアを素早く実際のプロダクトに取り込めるようになることを意味する
- **プログラミング言語との親和性**
  TensorFlow のコア部分は C++ で実装されており、ユーザ向けには C++ と Python のインターフェースが用意されている。そのほかの言語からも、SWIG などのブリッジツールを利用することで TensorFlow のライブラリにアクセスできる。また、すぐに使い始められるようにドキュメントやサンプルがそろっている
- **可視性**
  TensorBoard という強力な可視化ツールが付属する。これによって計算フローや計算結果を簡単に確認することができる

Google の狙い

機械学習はこれからの科学技術研究や革新的なプロダクトにとって不可欠な存在になるでしょう。しかし、実際に関連する研究論文を紐解いて自前で実装するのは決して容易なことではありません。GoogleがTensorFlowをオープンソース化した狙いは、そのような障壁を取り払い、多くの人が手軽に機械学習を活用して自分のアイデアを実現できるようにすることです。そして、多くの人々に利用され応用分野が拡がることによって、機械学習そのものの研究や普及を促進したいとのことです。

今後の方針としては、メモリ使用量の削減や処理速度の向上、対応するOSやデバイスの拡大などを進めることが挙げられています。また、本稿執筆時点のオープンソース版ではマルチマシンによる分散処理に対応していないため、この問題にも対処していくとのことです。SD

TensorFlow
https:////www.tensorflow.org/

DIGITAL GADGET vol.208

安藤 幸央
EXA Corporation
[Twitter] »@yukio_andoh
[Web Site] »http://www.andoh.org/

デジタル楽器のガジェット視点

デジタルな楽器、楽器のデジタル化

単なる懐古主義ではなく、新しい楽器としてシンセサイザーの新機種が続々と登場しています。ある時期、デジタル技術で既存の楽器の音に近い表現が競われ、ありがたがられたことがありました。しかし現在は楽器としての表現は当然のこととして、卓越したデジタル技術によって、さらに魅力のある音の表現力の幅が広がってきています。さらに、1980年代頃に活躍したシンセサイザーの復刻版や、音源を再現したソフトウェアシンセサイザーもその当時の価格と比べると安価に出そろってきています。

楽器の製造過程に関する技術的進化はあれど、バイオリンやピアノなど、演奏する楽器そのものは完成の域に達しています。一方で、その楽器で奏でられる音楽の流行は日々変化しています。楽器や音楽の進化を、コンピュータやソフトウェア、アプリの世界に当てはめて考えてみるとどうなるでしょう?

映画やドラマの音楽で、奇をてらった新しい音、流行の音楽が使われている場合、その時期に聴いたり観たりしたときは、たいそうな目新しさをもたらします。けれど、数年程度、少し時間が経ってから聴いたり観たりすると、とても古臭いものに感じてしまうのではないでしょうか? ハリウッドの大作映画を思い出してみてください。どれもフルオーケストラの古典的で重厚なサウンドトラックで、物語を表現してはいないでしょうか?

iPhone登場当時に大流行りしたスキュアモーフィズムと呼ばれる現実世界にある物を模倣したデザインは、いまではすっかりフラットデザインやマテリアルデザインに置き換えられており、実物風のデザインは古臭いものだと感じられています。

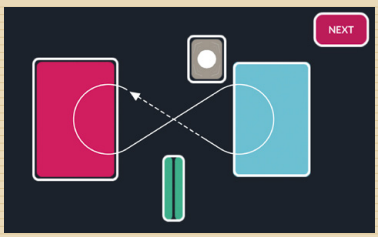
それと同様に文字フォントのデザインやグラフィックデザインも、長く使われてきたオーソドックスなものこそ、斬新な目新しさはないながらも、今後も長く見慣れたものとして変わらぬ印象を持ってもらえるのです。

ユーザインターフェースとして考えるデジタル楽器の存在

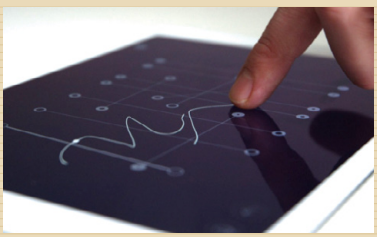
さて、楽器に話しを戻しましょう。楽器の演奏には、指先で感覚として感じられ、物としてのフィードバックがあることが重要です。タッチパネルやジェスチャーによって音を操作するアプリも存在しますが、微妙な演奏や超絶技巧の演奏などにはなかなか向きません。

微妙な調整が必要なもの、反発や、動きといったフィードバックを感じながら、その感覚によってさらに微調整をしていくようなものが楽器です。操作したらすぐに素早く反応するもの、単に音が出る／出ないだけではない「演奏する物」としての楽器の要

フランスExpressive E社の「Touché」。表面を押したり、指を滑らしたりして操作するコントローラ



タッチ操作で音楽を奏でる「PlayGround」。誰にでもノリの良い音楽が奏でられるアプリ



「SoundBow」は指の動きに合わせて音色がループするアプリ

※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います。

素が多々あります。

ボリュームつまみやフェーダー、スイッチなどといった物理的操作方法が、デジタル技術が駆使されたシンセサイザー楽器においても、身体の延長として弾きこなすための楽器には重要な要素となっています。

よく楽器やプログラミングの習得には、1万時間が必要だと言われます。1万時間というと毎日4時間練習したとして約7年です。楽器もプログラミングも、簡単な音を出したり簡単なプログラムであればすぐに実現できますが、人を感動させるような音楽を奏でたり、誰もが唸るようなプログラムを作るのは、そう簡単なことではありません。もちろん持って生まれた才能や身体的能力もあるかもしれませんが、やはり膨大な時間をかけないと、一定の水準に達するのは難しいことでしょう。

その一方、数十時間あれば、そこそこ楽器を弾けるようになるという考えもあり、その際に大切なことは「必要なスキルを細かく分解して把握できること」「間違いを自分で正せるようになること」「集中できること」が重要だと言われています。

そう考えると、スマートフォンのアプリやWebなどの操作にも、単に取っ付きやすい、単に使いやすいということばかりでなく、最初は少し使いづらく習熟までに時間がかかるが、習熟しきった際には間違うことなく、自分の身体の延長線として自由自在に使える、楽器的インターフェースというのを考慮しても良いのかもしれません。

未来の楽器、奏で続けられる楽器

1983年に登場したシンセサイザーの相互制御規格MIDI（Musical Instrument Digital Interface）は30年以上経った今も使われ続けています。登場当時の太くてごわごわしたケーブルから、現在は無線でやりとりできるようになったり、さまざまな機能が拡張されていたりしますが、楽器の音データを送受信するというその基本は変わりません。発案された当時は、限界性能ぎりぎり、かつ必要充分な仕様でありましたが、レガシーと言われていることこそ多くなれど現在でも使い続けられ、30年前のデジタル楽器と現在のデジタル楽器をつなげて演奏できることは素晴らしいことです。

3kgもある肩掛けタイプのショルダーホンが登場したのが1985年でした。今ではファイル保存のアイコンとしか認識してもらえない、3.5インチフロッピーディスクの登場は1980年でした。2DDと呼ばれる容量のディスクで720KBしかなく、今ではスマートフォンで撮影した写真1枚にも足りないくらいです。

今主流となっているものは、たいてい30年ほど前に発明されたり、研究されていたものだと言われています。数年で消えてしまうWebサービスやアプリが多い中で、今後生き残るデジタ

時代に逆行するような、
巨大マトリックススイッチを搭載した「MatrixBrute」
https://www.arturia.com/matrixbrute/overview

どんなギターにもアーム機能を
付け加えてしまう「Virtual Jeff」
http://www.fomofx.com/

「OP-Z」。まるでコンピュータ画面のようなシンセサイザー
https://www.teenageengineering.com/products/op-z

ワイヤレスMIDI/USB
MIDIインターフェース「MD-BT01」
http://jp.yamaha.com/products/music-production/accessories/interfaces/md-bt01/?mode=model

ル技術とはどのようなものでしょうか?

OSやプログラミング言語、開発ツールの寿命もそれほど長くありません。スマートフォンやタブレット端末、アプリの継続性もこの先どうなるのかわかりません。そこで楽器の存在から学べることは何でしょうか?

- ユーザが多いこと、知見や経験が受け継がれていくこと
- 教本や教室などで、教える人、教えられる人がいること
- そもそも奏でることが楽しみであること
- プロフェッショナルだけでなく、たくさんのアマチュアが存在すること

歴史の長い楽器や音楽の素晴らしさからさまざまな事柄を学び取り、デジタル技術やネット上のサービス展開、プログラミング技術、ユーザインターフェースの考え方などに生かせるのではないかと考えています。SD

1970年代に活躍したシンセの復刻版「SYSTEM-500」
http://www.roland.co.jp/products/system-500_complete_set/

譜めくりコントローラ「iRig BlueTurn」
http://www.ikmultimedia.com/products/irigblueturn/

MIDI over Bluetooth LE対応のキーボード「microKEY Air(25鍵、37鍵、49鍵、61鍵)」
http://www.korg.com/jp/products/controllers/microkey2_air/

Gadget 1

ARQ Aero RhythmTrak

https://www.zoom.co.jp/ja/products/production-recording/digital-instruments/arq-aero-rhythmtrak

円盤形ドラムマシン

Zoom ARQは加速度センサーを搭載した円盤形ドラムマシン。ドラム音を奏でるだけでなく、さまざまな音色を再生するシーケンサーとしても機能します。Bluetooth MIDIに対応しており、円形の輪の部分だけが分離してワイヤレスのタンバリンのような楽器としても活用できます。円という形状を活かし、ループフレーズを組み立てるのにも向いてます。468種類の生楽器系の波形、70種類のシンセ波形を内蔵。音色の元となる音源を加工し、新たな音色の加工も自在。2016年春に599.99ドルでリリース予定です。

Gadget 2

C.24

http://miselu.com/ja/c-24/

iPadケース型キーボード

miselu C.24はBluetoothでiPhoneやiPadの音楽アプリと接続することのできる、ケース型のキーボードです。2オクターブ分のミニしかありませんが、楽器として必要充分な操作と機能を持っています。CoreMIDIに対応したiOS、Mac OS Xの350以上の音楽アプリケーションに対応しています。演奏時はiPad/iPhoneスタンドにもなり、持ち運び時にはキーボードの部分が畳み込まれ、薄いケースとして扱うことができます。日本でも、ソフトバンクセレクションなどで24,800円(税込)で販売されています。

Gadget 3

Pocket Operator

https://www.teenageengineering.com/products/po

ゲームウオッチ風ミニシンセ

1980年代に流行した、任天堂のゲーム&ウオッチを覚えているでしょうか? 読者の中にはまだ生まれていない人もいるかもしれません。液晶画面を搭載したシンプルな携帯型ゲームで、任天堂のゲームの原点とも言われるものです。そのゲーム&ウオッチを彷彿させる画面と、無骨な部品むき出しの、電池で動くシンセサイザーがTeenage Engineering Pocket Operatorです。リズム、サブマリン、プラント、アーケード、オフィス、ロボの6機種がそろっており、新製品も予定されています。

Gadget 4

Mixfader

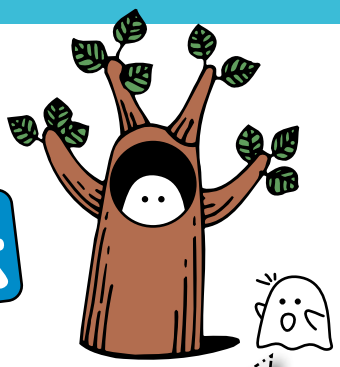
http://www.themixfader.com/

単体ミックスフェーダー

DJが2つの音源を混ぜ合わせながらプレイする際に、ミックスフェーダーと呼ばれるボリュームを操作します。たいていは専用のDJミキサーの中央部に設置されていますが、このMixfaderはBluetoothで接続された、単体の機器として扱うことができます。iPhoneやiPad、PC上のDJアプリのミックス操作を、このMixfaderで行うことが可能です。本体バッテリーは10時間持ち、たいていのDJプレイには充分な時間でしょう。一番心配な反応の遅延(レイテンシー)も極少まで減らされているそうです。

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

結城浩の再発見の発想法 35



Scope

Scope——スコープ

スコープとは

名前の**スコープ**(Scope)とは、名前の**有効範囲**のことです。スコープはもともと広い意味を持つ単語ですが、以下ではプログラミング言語での意味に限定して使います。1つのプログラムの中には、たくさんの名前が登場します。プログラマは、変数・関数・クラス・メソッドのような多数の概念に名前を付け、それらを組み合わせてプログラムを作ります。

スコープの記述方法は、プログラミング言語の文法規則によって定められています。たとえばJavaの場合、“{”と“}”で挟まれた範囲で定義された局所変数が使えるのは、その範囲のみです(**図1**)。言い換えれば、スコープの外からは使うことができません。どのようなスコープが存在するか、またそれをどのように記述するかは、プログラミング言語ごとに異なります。ですから、プログラミング言語を学ぶ人は、スコープについて学ぶ必要があります。

▼図1　名前のスコープ

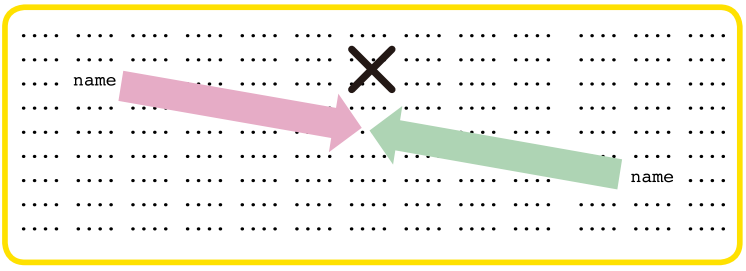
```
class Class1 {

    void method1() {
        int i;
    }

    void method2() {
        int i;
    }

}
```

名前のコンフリクトと人間の能力

プログラミングというものに慣れていない人は「どうしてスコープなんて面倒なものを使うのだろう。変数の名前が見えなかったら使えないのに。わざわざ不便なルールを作るなんておかしい」と考えるかもしれません。でも、もちろん、わざわざ不便なルールを作っているわけではありません。スコープを作る目的の1つは、名前の有効範囲を制限して**名前のコンフリクト(衝突)**を少なくすることにあります。

たとえば、プログラムのある個所で“name”という変数を使っていたとします。“name”という名前はとても一般的なので、別の個所で誰かが別の変数に同じ名前を使う可能性は高いでしょう。これが名前のコンフリクトです(**図2**)。ある個所で処理をしているのに、別の個所でその変数の内容を変更してしまったら、正しい処理

▼図2　名前のコンフリクト



が行われなくなってしまいます。

プログラミング言語は、スコープを使って名前の有効範囲を制限し、プログラムのある個所の処理が、ほかの個所に影響を与えないようにしているのです。

名前の衝突を防ぐというのはスコープの目的の1つですが、あらためて考えてみると、その背後には「**人間の能力は限られている**」という事実があることに気づきます。

プログラムは複雑な構造物ですから、バグを出さないように組み立てるのはたいへんなものです。バグの修正を行うときも、ほかの個所に悪影響を与えていないか注意しなければなりません。しかし、人間の能力は無限ではありませんから、プログラム全体に対して常に注意を払うことは不可能です。スコープ内の変数は、スコープ外から参照されていないことが保証されていますので、修正の影響が広範囲に及ばないように抑えられるのです。スコープは影響を局所化させ、人間への負担を減らしていると言えるでしょう。

1つの関数が何百行にも及ぶようなプログラムを書くと、周りの開発者から非難されます。それは、不用意にスコープを広げることになり、プログラムのメンテナンスがたいへんになるからです。意外と陥りやすいのは、プログラムをちょっとずつ修正しているうちにいつの間にか巨大な関数になってしまうケースです。スコープが大きくなりかけた段階で対処しないと混乱を生んでしまうでしょう。

言葉のスコープ

私たちの日常生活にもスコープはかかわっています。日常生活でのスコープとは、**ある言葉が「通じる範囲」**に相当します。

営業部に田中さんが1人しかいないなら、部内では「田中さん」と言うだけで不都合はありません。しかし、会社に田中さんがたくさんいたら、営業部の外では「田中さん」だけでは駄目ですね。名前のコンフリクトが起きるからです。その場合には「営業部の田中さん」のように補足情報を付けたり、「田中太郎さん」のようにフルネームを使ったりすることになるでしょう。それは、スコープが大きくなって発生した衝突を回避していると言えます。

私たちがプレゼンテーションを行うとき、自分が使う言葉がどの範囲まで通じるかを意識するのは大切です。聴衆が同じ部署だけか、会社全体か、あるいは社外の人も含むのかを意識して言葉を選ばないと、意味が通じない結果になってしまうでしょう。スコープが大きくなればなるほど、言葉の選択には注意が必要になります。

スコープが大きくなる名前ほど、注意深く作る必要があります。社内で使うコードネームならいいのですが、広く浸透させたい商品名やサービス名を生み出すときのことを想像すればよく理解できます。商品名でユーザが誤解したり、サービス名で名前のコンフリクトが起こったりしては困りますね。登録商標などの法的なしくみは名前のコンフリクトを解決するために用意されていると言えるでしょう。

最近私は、「マイナンバー」と「マイナンバーカード」という2つの名前が気になっています。この2つの名前は指すものが異なります。ですから、「マイナンバーを利用する」と「マイナンバーカードを利用する」とでは違う意味を持ちます。マイナンバーカードに記録されている情報はマイナンバーだけではないので、「マイナンバーカードを利用する」からといって「マイナンバーを利用する」とは限らないのです。この2つの名前は、国民全体という非常に大きなスコープに浸透させる名前ですので、もっと整理してほしかったと思います。

◆ ◆ ◆

あなたの周りを見回して、使っている言葉のスコープを考えてみましょう。あなたがよく使う言葉は、自分の周囲のどの範囲まで有効に通じるでしょうか。どこまでスコープを広げたら誤解が生まれるでしょうか。

ぜひ、考えてみてください。SD

増井ラボノート

コロンブス日和

第6回　Dynamic Macro

エンジニアというものは「楽をするためならどんな苦労も厭わない[注1]」ものだと言われていますが、コロンブスの卵のようなゴキゲンな発明によって頑張って楽できるなら、それに越したことはないでしょう。私はコンピュータ上の簡単な工夫で楽をする方法を考えるのが好きで、長年にわたっていろんなシステムを開発してきています。今回の連載では、私がこれまで開発して長年実際に利用しているような単純かつ便利なシステムをたくさん紹介していきます。

Author　増井 俊之（ますい としゆき）　慶應義塾大学

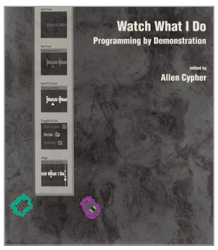
便利な予測インターフェース

計算機の上で同じ作業を何度も繰り返さなければならないことがよくあります。計算機は単純な繰り返し作業が得意なはずですが、つまらない作業を人間が繰り返さなければならないことは意外と多いものです。たとえばExcelの表の中の負の数字だけアンダーラインを付けたいときはどうすれば良いでしょうか？　そういう機能はExcelには用意されているかもしれませんが、知らなければ使えませんし、同じような処理であってもシステムに用意されていなければどうしようもありませんから、この手の仕事があったときは泣きながら手作業で処理したり、頑張ってスクリプトを書いたりしている人が多いのではないでしょうか。

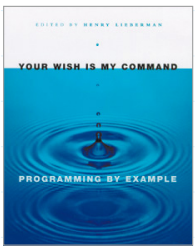
計算機上の操作を効率化するために「**予測インターフェース**」と呼ばれるシステムが広く使われています。スマホのテキスト入力を効率化する「予測入力システム」やブラウザのURL補完機能、エディタの補完機能のような簡単な予測インターフェースは最近よく使われていますし、これまでの購入履歴を基にして商品を推薦するシステムなども一種の予測システムと言えるでしょう。予測インターフェース研究の歴史はけっこう古く、1993年には予測インターフェースの研究をまとめた『Watch What I Do』という本が出版されていますし、最近は例示や予測だけでプログラムを作ってしまおうという研究も行われています（**図1**）。

予測インターフェースシステムでは、アプリケーションに関連したデータベースやユーザの操作履歴などを基にして、ユーザの次の操作を予測することによってユーザの仕事を減らす工夫をしています。URL補完の場合はよく使われるURLのデータベースが利用できますし、プログラミング言語に関する情報を持っていれば、ユーザが次に入力する言語キーワードを予測できます。このような固定的なデータベースを用意しておくことも重要ですが、ユーザの操作履歴を予測のためのデータベースとして利用すると便利です。ユーザが一度訪れたサイトのURLを覚えておけば補完に利用できますし、予測入力システムではユーザが利用した単語やフレーズが次の予測に利用されます。前述のExcelの例のような場合、負の数字にアンダーラインを付ける操作が繰り返されていることをシステムが検出できればユーザ

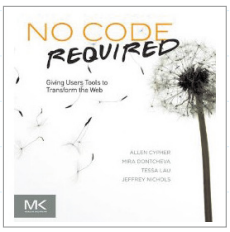
▼**図1**　予測インターフェースに関する書籍



Watch What I Do: Programming by Demonstration（Allen Cypher、The MIT Press、1993年）



Your Wish is My Command: Programming By Example（Henry Lieberman、Morgan Kaufmann、2001年）



No Code Required：Giving Users Tools to Transform the Web（Allen Cypher/Mira Dontchevam/Tessa Lau/Jeffrey Nichols 、Morgan Kaufmann、2010年）

注1）http://thinkit.co.jp/free/article/0709/19/

の次の操作を予測できるでしょう。

編集作業の効率化

文章やプログラムのようなテキストを編集するとき、同じような操作を繰り返すことがよくあります。たとえば、連続する行の先頭に記号を挿入したいような場合はカーソルを1行ずつ動かして記号を入力していくのが普通ですが、たくさんの行に対して同じ操作を繰り返すのはたいへんです。行頭に記号を挿入するスクリプトを書けば良いのかもしれませんが、一度きりかもしれない処理のためにプログラムを作成するのは面倒ですし、プログラミングの知識が必要です。また、CSV(Comma Separated Values：カンマ区切り)データの桁を並び替えたいときはどうでしょうか？CSVデータをExcelなどで読み取ってから並びを変えて出力すれば良いかもしれませんが、方法を考えるのも実際に作業するのも面倒です。

キーボードマクロ

このような作業を簡単にするために「キーボードマクロ」という機能が用意されているエディタがあります。キーボードマクロとは、一連のエディタ操作を1つのキー操作に割り当てることにより、複雑な編集操作を楽に実行しようというものです。たとえば「行頭に記号を挿入してから1つ下の行に移動する」という処理をAというキーに割り当てておけば、Aを連打することによって連続する行の先頭に記号を入力していくことができますし、「カンマで区切られた部分を選択して移動してから次の行に移動する」という処理をBというキーに割り当てておけば、Bを連打することによってCSVの桁を入れ替えていくことができます。

キーボードマクロは便利な機能ですが、キーボードマクロの定義開始と終了のための操作が必要であるうえに、処理を正確に登録するのが難しいという問題があります。たとえば前述の例の場合、「1つ下の行に移動する」処理を定義に含めることを忘れてしまうと正しく動きません。

Dynamic Macro

キーボードマクロの機能をもっと簡単に使えるようにするため、私はキーボードの繰り返し操作から次の操作を予測して、自動的にキーボードマクロとして登録できるDynamic Macroというシステムを作って長年Emacsの上で利用しています。

Dynamic Macroの原理は非常に単純で、

「同じ編集操作を2回繰り返したあとでCTRL+tを押すと繰り返された操作が再実行される」

というものです。「二度あることは三度ある」と言うように、同じことが二度あればもう一度あるのは世の中でごく普通のことです。二度実行した操作をもう一度実行することもよくあることですので、この方法はたいへん効果的です。

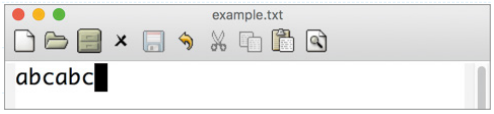
Dynamic Macroの利用例

Emacs上に実装したDynamic Macroの利用例を示します。**図2**はEmacsでabcabcと入力したところです。

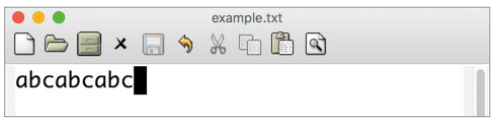
ここでCTRL+tを押すと、Emacsのキー操作履歴から「abcの入力」の繰り返しが検出され、キーボードマクロとして登録されて実行され、**図3**のようにもう1つabcが挿入されます。

再度CTRL+tキーを押すと、**図4**のようにまたabcが入力されます。

▼図2 Emacsでabcabcと入力



▼図3 CTRL+tキーでもう1つabcが入力される



これは単純な例でしたが、Dynamic Macroはもっと複雑な編集操作でも使うことができます。

図5のようなテキストの上から2行を図6のように編集したとします。

図6は、行頭に「puts "」を入力してから行末に移動して「"」を入力して 次の行に移動するという操作を2回繰り返した結果ですが、ここでCTRL+tキーを押すと、この操作の繰り返しが検出されてマクロ登録されて実行されるので、画面は図7のように変化します。

さらにCTRL+tキーを何度か押すと画面は図8のように変化します。

このように、複雑な操作であっても、同じ操作を2回繰り返したあとではDynamic Macroで何度でも連続実行できることになります。

Dynamic Macroの特徴

Dynamic Macroは繰り返し操作を効率化するシステムですが、予測インターフェースの考え方をキーボードマクロに応用したものだともいえます。ユーザの繰り返し操作を基にして次の操作を「予測」し、それをキーボードマクロのように利用できるからです。予測と言っても繰り返し操作からの「予測」ですので、誤った予測をしてしまう可能性はほとんどありません。

キーボードマクロと比較すると、Dynamic Macroには次のような利点があります。

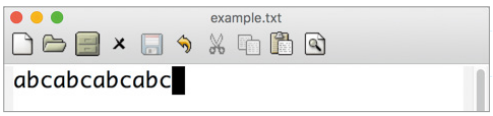
- 使うキーがCTRL+t1つだけである
- 定義の開始と終了を正確に指定する必要がない
  →繰り返し操作中のどこでCTRL+tを押しても操作が再実行される
- 操作を行ったあとで繰り返し実行を指示できる
  →普通のキーボードマクロを利用する場合、これから繰り返し操作を行うぞ、と意識して登録を開始する必要があるが、Dynamic Macroの場合は操作のあとで繰り返しに気づいて再実行させることができる

Emacsでの実装

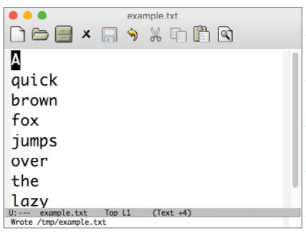
Dynamic Macroは最初はEmacsの上で実装されました。Emacsでは(recent-keys)という関数を使って、最近のキー操作履歴を知ることができるので、CTRL+tが押されたときに、この機能を使ってキー操作履歴を取得し、繰り返し操作が見つかれば、それをキーボードマクロとして登録して実行すれば良いことになります。

dmacro.elを改良したndmacro.el[注2]というシステムもsnj14さんによって公開されています。ndmacro.elでは 1, 2, 3 と入力したあとでCTRL+tを押すと4, 5, 6, ……のように、連続する文字列を入力していくことができます。

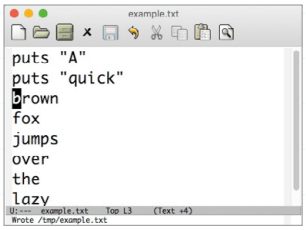
▼図4 CTRL+tキーでさらにもう1つabcが入力される



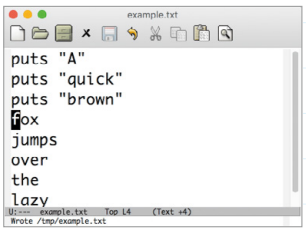
▼図5 Dynamic Macroでもっと複雑な編集操作



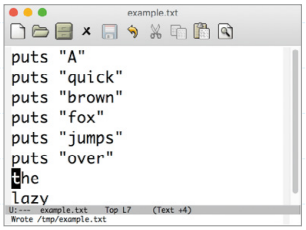
▼図6 テキストを編集してみる



▼図7 CTRL+tキーで入力補完



▼図8 CTRL+tキーを何度か押してみる



注2) https://github.com/snj14/ndmacro.el

Atomでの実装

GitHubが開発しているAtomというエディタが最近プログラマの間で人気が出ています。AtomはJavaScriptとブラウザ技術をベースに作成されたモダンで高機能なエディタで、ユーザが自由にJavaScriptやCoffeeScriptで拡張機能を作成できます。Atomの拡張機能を利用することによって、Atom上でもDynamic Macro[注3]を利用できます(図9)。

Atomには(recent-keys)のような履歴保存機能は用意されていないので、addEventListener()のような機能を使って、操作履歴を自力で覚えておくようにしています。

予測インターフェースの難しいところ

予測インターフェースは便利なものですが、あくまで「予測」ですので、システムの予測がユーザの意図と異なる可能性が常に存在します。ユーザが1, 2, 3, 1, 2, 3と入力したとき、ユーザが次に入力したいのが1なのか4なのかはユーザ本人にしかわかりません。高度な予測を行おうとすると、予測を間違える可能性が高くなりますし、複数の予測結果から希望するものを選択する必要が出てくるかもしれませんから、機能と使い勝手のバランスが重要になってきます。

予測入力システムの場合、ユーザが入力した い単語が常に第一候補として提示されるわけではありませんが、欲しい単語が候補リストに含まれている可能性が高ければ、それほどユーザは不満を感じません。一方、正しく予測されることへの期待が大きいのにもかかわらず、頻繁に間違った予測が実行されてしまうようであれば、ユーザの失望が大きいため、予測システムの利用をあきらめてしまうかもしれません。予測インターフェース研究の歴史は長いにもかかわらず、最近まであまり利用されていなかったのは、こういう理由が大きいと思われます。Dynamic Macroの場合、まったく同じ操作を繰り返すだけですので間違った予測が実行されることはほとんどないのですが、それでも間違えることが皆無ではありません。もっと微妙な予測の場合は、システムが間違った予測をしてしまうことは多いでしょう。

Dynamic Macroのような予測機能は、実世界のさまざまな場所で使える可能性があります。同じ設定で2回動かしたらその設定を繰り返せる洗濯機が売られていたことがありますし、同じフレーズを2回弾いたら何度も繰り返してくれるピアノがあれば便利かもしれません。いろいろな予測機能を有効に使うことによって、世の中の単調作業を何でも効率化していきたいものです。SD

▼図9 Atomエディタのマクロ「atom-dynamic-macro」

コラム「エディタと私」

私は1980年代からEmacsを利用しており、その上で20年以上Dynamic Macroを使い続けています。Emacs以外のエディタも使ってみたい気持ちはあったのですが、Dynamic Macroが使えないエディタの利用は私には考えられないので、ずっとEmacsを使い続けてきていました。最近AtomでDynamic Macroが動くようになったので、これからはAtomを利用するようにしようかと考えています(この原稿もAtomで書いています)。

EmacsもAtomももともとプログラマ向けに開発されたエディタですが、Atomは30年(!)後発なだけに、Emacsに比べると利用のハードルは低いですし、GUI的な機能もしっかりしており、万人に勧められるものに進化しつつあります。これからはAtom上で誰もが便利に使える他のツールの作成にも挑戦したいと思っています。

注3) 今回のソフトウェアの公開場所
・Emacs版Dynamic Macro(https://github.com/masui/DynamicMacro)
・Atom版Dynamic Macro(https://github.com/masui/atom-dynamic-macro)

第2回 一般向け／ビジネス向けのOSC

宮原 徹（みやはら とおる） Twitter @tmiyahar 株式会社びぎねっと

OSCは土曜日開催が基本です

オープンソースカンファレンス（以下OSC）は全国各地で開催していますが、通常の開催ではオープンソースのビジネスを行っている企業と、ボランタリーに活動しているいわゆるコミュニティが一緒になって展示やセミナーを行っています。

東京や京都といった規模の大きな開催では金曜土曜と2日間の開催ですが、ほとんどは1日開催です。これは、参加対象となるエンジニアのみなさんが平日に仕事を抜け出して、あるいは有休を取得して参加するのが難しいという事情を鑑みてのことです。

そのため、出展している企業のみなさんには休日出勤をしていただいていることになり、人員のアサインなどにご苦労をおかけしていることになり申しわけないです。

また、「子供の行事と重なって……」という理由で参加いただけないこともあり、なかなか開催日程を決めるのも難しいなあといつも感じています。今のところは最大公約数としての土曜日開催（あるいは祝日開催）というのが定着しています。

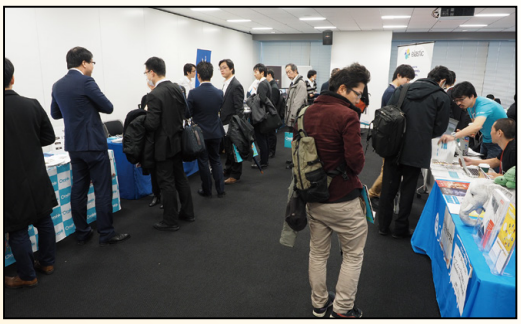
ビジネスのためのOSC.Enterprise

一方で、「ちゃんとビジネスとして平日参加できるよ、むしろ土曜日に参加するのは難しいよ」という声も一定数あります。このようなご要望にお応えするため、ビジネス中心、企業中心での開催となる「OSC.Enterprise」を別シリーズとして東京と大阪で開催しています。東京は2015年12月9日（水）、大阪は2016年1月29日（金）に開催されました。

ビジネス向け開催というだけあって出展するのはほとんどが企業ですが、LibreOfficeやCMSなど、多少ビジネス色があるコミュニティも出展しています。それでも、全体的な雰囲気は文化祭のような普段のOSCに比べると、スーツ着用率も上がりますし、いわゆるビジネスイベントに近いものになります（**写真1、2**）。これはこれで、オープンソースが着実にビジネスで利用されるようになっているということを見える化するためには大事なことだと思っています。

平日に開催するだけでなく、会場選びも少しビジネス寄りにしています。東京は渋谷駅から徒歩数分の貸しホール会議室、大阪もJR大阪駅と空中歩道でつながったビル内の貸し会議室と、便のよい場所を選んでいます。普段の開催が大学や専門学校、公共施設などを利用しているのに比べると、来場

▼写真1 OSC.Enterprise大阪。ビジネス向けなのでスーツ多めです



▼写真2 懇親会はなかったので打ち上げに串揚げです。ソース二度付け禁止

しやすさは格段に上がります。

普段の開催も交通の便のよいところにしたいのはやまやまですが、OSCは広い展示スペースとたくさんのセミナー会場が必要になるため、そのような場所は学校の校舎などでないと見つけられないのと、もし条件に合う会場があったとしても利用料が高過ぎるという悩ましい問題があります。

来場者アンケートに「会場が遠い」と書かれることも多いのですが、そういう事情があるのです。とくに都心部での開催は一度早稲田大学の校舎をお借りして開催したのですが、やはりスペースが不足して廊下を歩くことすらできないという状態でした。どこかよい会場があれば、教えてもらえるとうれしいです。

OSC浜名湖は会場の一体感が魅力

一方で、コミュニティ中心の開催となる小規模な開催も独自の魅力があります。2016年1月23日(土)に開催されたOSC浜名湖は、浜松市市民協働センターという公共施設のギャラリースペースを借りて開催されました。ギャラリーというだけあって、仕切りのない広いスペースです。そこにスクリーンとプロジェクター、客席をセミナースペースとして配置し、それをL字に囲むように展示スペースとして机を配置します。間に遮るものがありませんので、展示スペースにいる人もセミナーの様子を見聞きできます。通常の開催ではセミナー会場は別ですから、セミナー開催中は展示スペースがガラーンとしてしまうことがほとんどですが、このような一体化した会場だと常に賑やかな感じになります(**写真3**)。

▲写真3　2015年ゆるキャラグランプリで優勝した「出世大名家康くん」も来場

もちろん、セミナー中に展示スペースでの話し声や、ときには笑い声なども聞こえてきてしまうのは難点ですが、セミナーも1つだけ、参加者も100名程度の開催規模であれば深刻な問題にはならないで済んでいるようです。

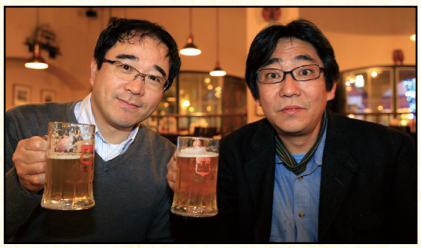
同じようなスタイルで昨年は新潟でも開催しましたし、今後未開催地域での開催はこの展示セミナー一体型のスタイルか、OSunCとしてもっとカジュアルに開催するかのいずれかの方法で全国展開していけそうです。SD

Report

前夜祭から懇親会まで大盛り上がりの浜松

「浜松といえばマインシュロス」というぐらい、地ビールが美味しい店があります。今回はOSC当日が貸し切り営業だったため、有志で前夜祭を開催しました。ドイツスタイルのビールや店内で大盛り上がりでしたが、本番は翌日なのでほどほどでお開き。

OSC浜名湖開催後の懇親会は、30名席に38名を詰め込み身動きが取れないぐらい盛況でしたが、全国の地酒が飲み放題という素敵な店でした。静岡の地酒を飲み比べたり、東北地方の地酒を飲み比べたり。その後、2次会、3次会と、浜松の夜は更けていくのでした。



▲OSC浜名湖前夜祭。マインシュロスのビールで乾杯！　浜松の杉本さんと

▲OSC浜名湖懇親会。参加者が多過ぎてギューギュー詰めです

ツボイのなんでもネットにつなげちまえ道場

第10回

Ethernetにつないでみる

Author 坪井 義浩（つぼい よしひろ） Mail ytsuboi@gmail.com @ytsuboi
協力：スイッチサイエンス

Ethernet!!!

この連載のタイトルは、「なんでもネットにつなげちまえ道場」なのに、まだネットにつなげる話がまったく出てきていませんでしたね。今回は、mbed LPC1768についているEthernetインターフェースの話をしてみたいと思います。

RMIIってなに?

LPC1768というマイコンには、Ethernet MACが付いています。しかし、マイコンにそのままRJ-45のコネクタを接続できるわけではありません。LPC1768に内蔵されているのは、Ethernetのうち、MAC（Media Access Control）、メディアアクセス制御を行う部分までです。EthernetのRJ-45コネクタとMACとの間には、パルストランスという部品と、PHYと呼ばれるチップが必要です。

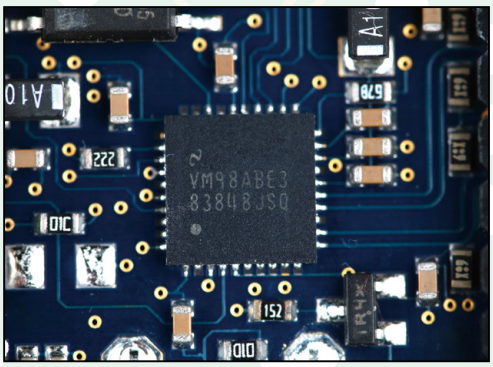
PHYは、OSI参照モデルでも登場する名前です。Physical layer（物理層）の頭文字を取って、PHYと呼ばれています。PHYは物理層を担当し、EthernetのMACアドレスもPHYのチップに記録されています。mbed LPC1768には、DP83848JというPHYのチップ（**写真1**）が搭載されています。

MACとPHYの接続インターフェースには、よく使われるMII（Media Independent Interface）という規格があります。LPC1768のMACには、RMII（Reduced Media Independent Inter face）というMIIの信号線を減らした規格のインターフェースが付いていますので、このRMIIでLPC1768とDP83848Jが接続されています。

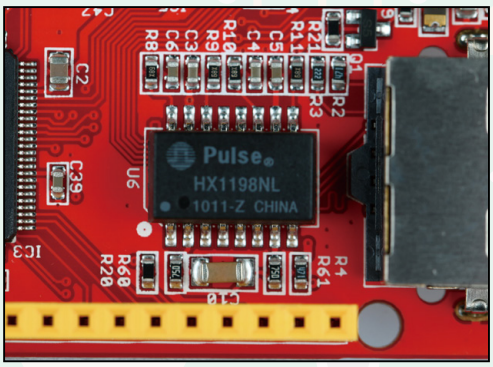
パルストランス（**写真2**）は、LANケーブルを通じてパルス信号を伝送する役割と、LANコネクタとマイコンボードの絶縁をして機器内部の回路を守るという役割をしています。RJ-45コネクタにパルストランスを内蔵した製品もあります。先述のように、mbed LPC1768にはPHYまでが搭載されていますので、あとはパルストランスを用意すればEthernetに接続できます。

mbedアプリケーションボードには、パルスト

▼写真1　PHYのチップ



▼写真2　パルストランス



ランスを内蔵したRJ-45コネクタが付いています。mbedアプリケーションボードをお持ちの方はこれを使うのが手っ取り早い方法でしょう。あるいは、スイッチサイエンスの「mbed用Ethernet接続キット[注1]」を使うと、**写真3、4**のようにブレッドボードの上で手軽にmbedをEthernetにつなげられます。

lwIP

mbedのTCP/IPスタックには、lwIP(A Light weight TCP/IP stack)[注2]が採用されています。lwIPは、メモリが数十KBといった組み込みシステム向けに設計された、オープンソースのTCP/IPスタックです。mbedでは、Ethernet Interfaceというライブラリから参照されている、lwip[注3]というライブラリが使われています。

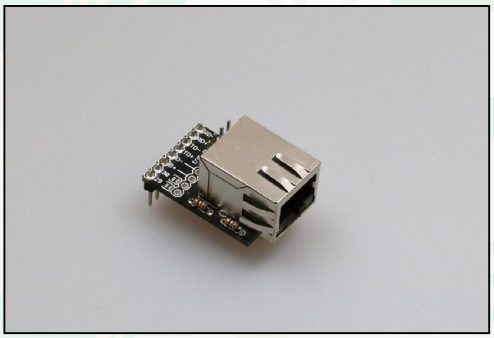
lwIPはトランスポート層までを担っているので、HTTPなどのアプリケーション層を扱うためのライブラリもmbedには存在します。HTTP Client[注4]というものです。アプリケーション層のライブラリには、MQTT[注5]や、NTPClient[注6]などがあります。

なにはともあれ通信してみる

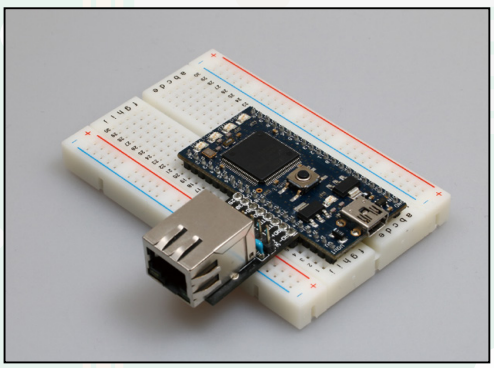
では、mbed LPC1768のEthernetインターフェースを使って、HTTP通信をしてみましょう。サンプルプログラムのHTTPClient_Hello World[注7]を自身のオンラインコンパイラにインポートします(**図1**)。このとき、「Update all libraries to the latest revision」にチェックを入れておいてください。

mbedのライブラリやコードのホスティング機能には、バージョン管理が付いていますが、プログラムやライブラリどうしのバージョン依存性を解決するしくみがありません。このため、サンプルプログラムなどをそのままインポートした状態では動かないことがあります。これは、ユーザがそれぞれ開発をしているmbedにありがちな問題でした。現在開発が進められているmbed OSのパッケージマネージャであるyotta

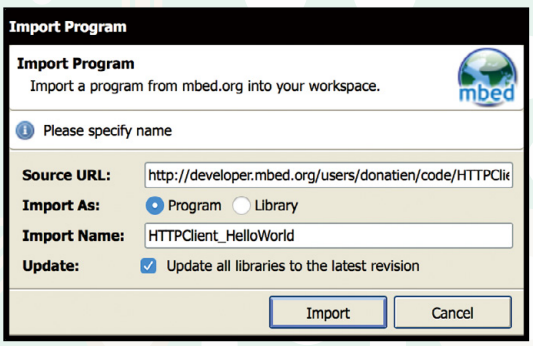
▼写真3　mbed用Ethernet接続キット(組み立て済み)



▼写真4　Ethernet接続キットの接続例



▼図1　サンプルプログラムのインポート



注1) http://ssci.to/555 (514円)
注2) http://savannah.nongnu.org/projects/lwip/
注3) https://developer.mbed.org/users/mbed_official/code/lwip/
注4) https://developer.mbed.org/users/donatien/code/HTTPClient/
注5) https://developer.mbed.org/teams/mqtt/code/MQTT/
注6) https://developer.mbed.org/users/donatien/code/NTPClient/
注7) https://developer.mbed.org/users/donatien/code/HTTPClient_HelloWorld/

では、こういったライブラリどうしのバージョン依存を整理するしくみが取り入れられています。

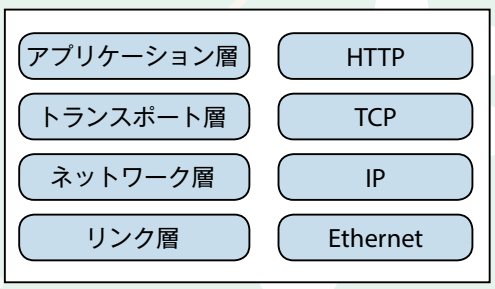
ライブラリを最新版にしてサンプルプログラムをインポートすると、Warningが出るものの、ビルドできました。サンプルプログラムでは、17行目を、

```
int ret = http.get("http://mbed.org/media/uploads/donatien/hello.txt", str, 128);
```

として、HTTPのGETメソッドでアクセスをしています。こうしてmbedでは簡単にHTTPで通信を行うことができます。ちなみに、このサンプルプログラムが作られたのは少し前ですので、アクセスをするように指定しているURLが古くなってしまっています。このURLにアクセスすると「HTTPステータスコード301、Moved Permanently」が返ってくるのですが、ライブラリにはWebブラウザとは異なりステータスコードを処理して再度アクセスをする機能は実装されていません。ここでは、新しいURLである「**http://developer.mbed.org/media/uploads/donatien/hello.txt**」にコードを修正してください。すると、正常にアクセスできます。

mbed LPC1768でとくに指定をせずにprintf()をすると、mbedに搭載しているUSB-UARTブリッジにつながっているUARTから、「9,600bps、8bit、パリティなし、ストップビット1」でテキストメッセージが出力されます。お手持ちのシリアルターミナルをこの設定にして、mbed LPC1768のUSBポートを開くと、上記のメッセージを確認できます。

▼図2　ワークスペースのツリー



RTOS

HTTPClient_HelloWorldをインポートすると、ワークスペースのツリーは**図2**のようになっていて、mbedライブラリやEthernetInterfaceライブラリ、HTTPClientライブラリとともにmbed-rtosというライブラリがインポートされていることが確認できます。

mbed-rtosは、mbedのRTOS(Real-time operating system)です。RTOSは、組み込みシステムの限られたリソースの中で、資源管理を行うソフトウェアです。具体的には、mbed-rtosをインポートすることで、Thread、MutexやSemaphore、Queueなど、並列処理をするのに必要であろう機能を使うことができるようになります。mbedのRTOSについて詳しくは、**https://developer.mbed.org/handbook/RTOS**を参照してください。

mbed OSも同様にRTOSですが、uVisorといったセキュリティ機能が追加されています。uVisorは、Cortex-Mについている MPU(メモリ保護ユニット)を使うなどして、ACL(Access Control Lists)にしたがって入出力やメモリアクセスを制限するといった資源管理を行うものです。今のところ、以前から提供されているmbed SDK 2.xで説明をしている本連載ですが、mbed OSが一般的になりつつあるタイミングでmbed OSベースの記事に移行したいと考えています。

ほかの接続手段

ネットワークにつなげるとなると、Ethernet以外の手段も検討したくなりますよね。mbed LPC1768はUSBホストになることができるので、USBのWi-Fiインターフェースを接続したくなるのですが、筆者の知る限りUSBでWi-Fiを接続するライブラリは公開されていません。

mbedをWi-Fiに接続するには、「ムラタ 無線

LANモジュール Type YD[注8]」や、ESP-WROOM-02[注9](**写真5**)、CC3000[注10]など、Wi-Fiのドライバやサプリカント(Wi-Fiの認証技術をサポートするソフトウェア)が搭載されたWi-Fiモジュールを使います。こうしたモジュールは、3～5千円近くしたのですが、昨年にESP-WROOM-02という安価なモジュールが出て、一気に千円を切る価格で買えるようになりました。

こういったモジュールは、UARTなどのマイコンに一般的についているインターフェースを使って接続するように作られています。たとえばESP-WROOM-02の場合は、モジュールで採用されているマイコンである、ESP8266用のライブラリ[注11]を使って接続します。サンプルコードのESP8266_Testを見ると、mbedのUARTのピンを使って接続をしていることが確認できます。

ほかにも、最近話題のWi-SUNのインターフェースもあります。Wi-SUNは、スマートメーター(通信機能が付いた新しい世代の電力計のこと)の通信を目的として、IEEE802.15.4gという規格を元に作られた無線通信規格です。ROHMのBP35A1というモジュールが手軽に使えるので、筆者も試しに使ってみたことがあります(**写真6**)。BP35A1は、サブギガと呼ばれる1GHzよりも低い周波数帯で通信を行うモジュールです。IPv6のアドレスを使って通信をします。このモジュールもUARTで手軽にマイコンと接続できます。

USBに接続することのできるインターフェースとしては、携帯電話網に接続する3Gモデムがあります。mbedのWebサイトで紹介されているのはイギリスのVodafoneのK3770やK3772-Zという3Gモデムです[注12]。数年前の話ですが、筆者はこのライブラリを改造して、NTTドコモの3GモデムとMVNOのSIMカードを使ってインターネット接続ができるようにしていました。

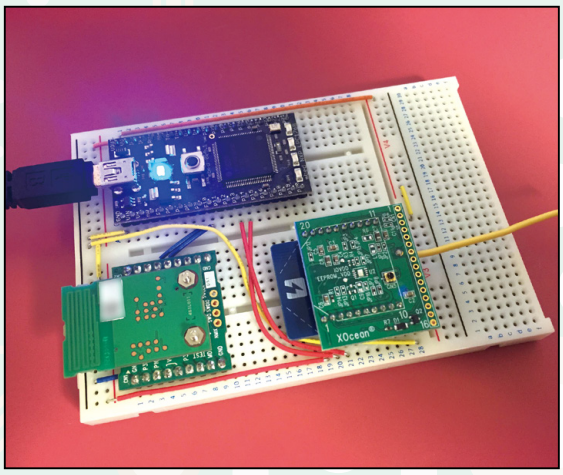
まとめ

今回は、ネットワーク接続の手始めとしてHTTPを使ってみました。次は、IoTで使われるMQTT(Message Queue Telemetry Transport)を使って「なんでもネットにつなげちまう」話をしていきたいと思います。SD

注8) http://ssci.to/1919
注9) http://ssci.to/2341
注10) http://ssci.to/1695
注11) https://developer.mbed.org/components/ESP8266-01/
注12) https://developer.mbed.org/cookbook/VodafoneUSBModem

▼写真5 ESP-WROOM-02ピッチ変換済みモジュール《シンプル版》

▼写真6 BP35A1を試用してみた



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」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2016年4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、ご容赦ください。

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。アンケートの回答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。記入いただいた個人情報は、作業終了後に責任を持って破棄いたします。

01 Wi-Fiホームルータ「Aterm WF1200HP2」 1名

IEEE802.11ac規格のWi-Fiホームルータです。5GHz帯、2.4GHz帯ともに2本のアンテナを利用する2ストリームへ対応し、5GHz帯で最大867Mbps、2.4GHz帯で最大300Mbpsの高速通信ができます。また、「こども安心ネットタイマー」機能によって、保護者がスマホなどから、Wi-Fi接続するスマホやゲーム機の端末ごとに、接続スケジュールを設定できます。

提供元 NECプラットフォームズ http://121ware.com/aterm

02 ヘッドマウントディスプレイ「DN-13539」 3名

スマホに表示させた、左右に2分割された「ステレオペア」仕様の画像・動画を、立体映像として観られるヘッドマウントディスプレイです。軽い素材で、頭に負担がかかりにくい仕様です。対応のスマホはディスプレイサイズが3.5～6インチのもの。

提供元 ドスパラ上海問屋 http://www.donya.jp

03 Paragon Camptune X 2名

MacとWindowsが共存しているBoot Camp環境（HFS/NTFS）で、空き領域を相互に移動することで、両OS間のパーティション比率を変更できるユーティリティソフトです。対応OSは、OS X El Capitan/Yosemite/Mavericks/Mountain Lion/Lion、Windows 10/8.1/7。

提供元 パラゴンソフトウェア http://www.paragon-software.com/jp

04 エンジニアのためのGitの教科書

河村聖悟 ほか 著

Gitは「どういうものか」から具体的に「どう使うか」までしっかり学べる1冊。分散型バージョン管理システムのしくみから基本操作、ブランチの設計・運用、といったチーム開発手法まで解説しています。

提供元 翔泳社 http://www.shoeisha.co.jp 2名

05 Effective Python

Brett Slatkin 著

Pythonを知り尽くした著者が、より良いPythonコードを書くために何をすべきか、すべきでないか、なぜこれが良い方法なのかをPythonの流儀に従って教えてくれます。

提供元 オライリー・ジャパン http://www.oreilly.co.jp 2名

06 ネットワーク運用管理の教科書

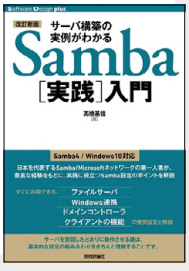
のびきよ 著

ネットワーク運用管理業務を「定常業務」「非定常業務」「Q&A対応」「トラブル対応」の4つに分け、それぞれの作業の進め方や技術ポイントを、図解を使った具体例を挙げて説明しています。

提供元 マイナビ出版 http://pub.mynavi.jp 2名

07 【改訂新版】サーバ構築の実例がわかるSamba[実践]入門

高橋 基信 著



Sambaの基礎的な知識から、具体的なシーン別のサーバ構築の実例を挙げて解説した1冊。前書からは、最新Samba4に対応し、Ubuntu環境、Windows10の対応などの解説が追加されました。

提供元 技術評論社 http://gihyo.jp 2名

やればできる！ ワンランク上のプログラミング

今すぐ実践できる良いプログラムの書き方

きれいなコード／モダンなコードが書きたい［C、Java、C#、Ruby、JavaScript］

書籍やマニュアルを読めば、文法や基礎的なアルゴリズムは学べますし、一応、動くプログラムは書けるようになります。しかし、プロが書くレベルのプログラムには遠く及びません。

プロはコードを書くとき、どんなところに気を配っているのでしょうか。「読みやすいか」「処理効率はいいか」「その言語の慣習に沿っているか」「セキュリティのことを意識しているか」など、その視点は書く人や使う言語によってさまざまです。

そこで本特集では、5つの言語のそれぞれのスペシャリストに良いコードを書くためのポイントを伝授してもらいます。彼らの視点を取り入れて、脱初心者、脱レガシープログラマをめざしましょう。

第1章　C言語編

Best Practices for Improving the Quality of Your Code.

enum、配列、浮動小数点を駆使して差をつけよう「より良いプログラム書きのヒント」

Author 星野 香保子（ほしの かほこ）

C言語は、プログラミングの学習者が最初に覚えることの多い言語です。学び始めのころからよいコードを書く習慣は、上達への早道です。本章では、enum、配列、浮動小数点を題材に、初級者向けによいコードを書くコツをお伝えします。新人さんの教育や、中級者の復習用として、気楽にご一読ください。

プログラミング上達のために

春は新人教育の季節でもあり、プログラミング言語の学習者が増えるそうです。学び始めのころからよいコードを書く習慣を身につけると、プログラミングの上達も早いのではないでしょうか。

本章では、C言語のよいコードを書くためのちょっとしたコツをいくつか紹介します。内容はおもに初級者を対象とした基本知識が中心になります。C言語の基本はだいたい知っている、という方は復習用または新人教育用として目を通していただければ幸いです。

▼リスト1　#defineで定数を定義

```
#include <stdio.h>

#define SPRING  0  /* 春 */ ┐
#define SUMMER  1  /* 夏 */ │
#define AUTUMN  2  /* 秋 */ │──①
#define WINTER  3  /* 冬 */ ┘

int main(void) {
    int    season = SPRING;

    if (season == SPRING) {
        puts("春が来た！");
    }
    return 0;
}
```

enumを活用しよう

プログラムで定数を扱うとき、列挙型（enum）を使うとすっきりと記述できる場合があります。

enumでラクラク定義

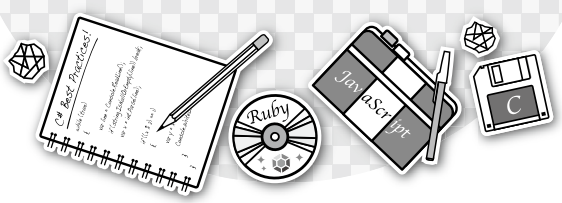
リスト1は、四季の区別を定義した例です。実行すると「春が来た！」と表示されます。

リスト1の①の部分では#defineを使って、SPRING、SUMMER、AUTUMN、WINTERのそれぞれに0から3の数値を割り当てていますが、この数値自体に特別な意味はありません。このような場合、列挙型（enum）を使うとより簡潔に書けます。

enumを使うように書き換えたのが**リスト2**の①の部分です。実行結果は、**リスト1**と同じです。enumによる定義では、名前を列挙するだけで、最初の名前から順に0,1,2……と整数が自動的に割り当てられます。

enumで数値を指定

列挙型は、「=」記号を使うことにより、明示的に値を指定することもできます。**図1**のように書くと、SPRINGに1が定義され、以降は1ずつ加算した数値が割り当てられます。指定できる数値はint型整数の範囲であり、**図2**のよう



に負数を指定することもできます。

配列を使った処理

同じ性質のデータを連続してまとめて扱うときは、配列を使うと便利です。以降では、配列に初期値のデータを入れたり、要素の数を求めたりするための記述方法について説明します。

試験結果を集計する例題

説明のために例題を考えてみます。ここに5人の生徒が受けた試験の結果の点数データがあるとします。**図3**のように点数の範囲により3段階のランク分けを行い、ランクごとの人数を求めるプログラムを作ります。まずは、**リスト3**のようになりました。実行結果を**図4**に示します。

配列を宣言して初期値を入れる

リスト3の②と③では、配列を宣言すると同時にデータ(初期値)を入れています。「=」記号に続けて中括弧「{ }」を書き、初期値として入れる値をカンマで区切って記述します。**tensu**と**shukei**はどちらもint型の配列で、配列**tensu**の要素数は5、配列**shukei**の要素数は3です。

配列の初期化をもっとスマートに

リスト3の①で、NINZU(人数)は5と定義しているので、②は次のような記述と同じです。

```
int  tensu[5] = { 63, 75, 48, 92, 66 };
```

もし生徒の数が7人に増えたら、次のようになります。

```
int  tensu[7] = { 63, 75, 48, 92, 66, ↵
70, 59 };
```

▼リスト2　列挙型(enum)で定数を定義

```
#include <stdio.h>

enum Season {
    SPRING,         /* 春 */
    SUMMER,         /* 夏 */
    AUTUMN,         /* 秋 */
    WINTER          /* 冬 */
};                                    ①

int main(void) {
    int    season = SPRING;

    if (season == SPRING) {
        puts("春が来た！");
    }
    return 0;
}
```

▼図1　列挙型(enum)で数値を指定

```
enum Season {
    SPRING = 1,
    SUMMER,
    AUTUMN,
    WINTER
};
```

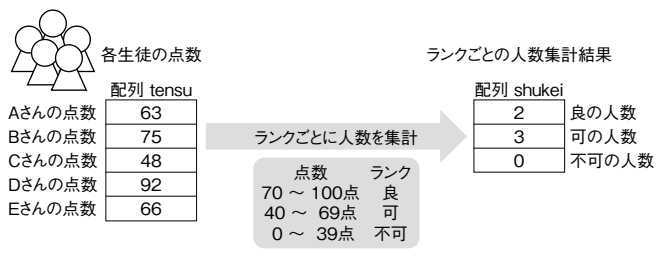
数値を明示的に指定しない場合「手前の値+1」の値が設定される
この例では、SUMMERは2、AUTUMNは3、WINTERは4になる

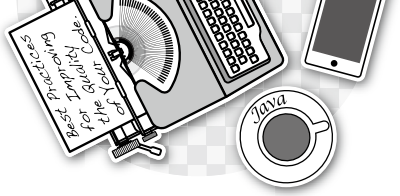
▼図2　enumは負数も指定可

```
enum Season {
    SPRING = -10,
    SUMMER,
    AUTUMN,
    WINTER
};
```

SUMMERは-9、AUTUMNは-8、WINTERは-7になる

▼図3　点数のランクごとに人数を集計





もちろん上記のように記述してもよいのですが、次のように書くこともできます。

```
↓5人のとき
int  tensu[] = { 63, 75, 48, 92, 66 };
↓7人のとき
int  tensu[] = { 63, 75, 48, 92, 66, ➤
70, 59 };
```

このように、「[ ]」の中に要素の数を指定しなければ、コンパイラが自動的に「{ }」の中のデータ数分の配列を用意してくれます。

配列を0で初期化

リスト3の③では配列のすべての要素を0で初期化しています。この例のように配列の要素の数が決まっていて、すべての要素を0で初期化するときは、より簡単な書き方ができます。

▼図4　実行結果（リスト3とリスト4）

```
結果  良: 2人  可: 3人  不可: 0人
```

▼リスト3　配列を使う処理（改良前）

```
#include <stdio.h>
#define NINZU   5   /* 生徒数   5人 */  ―――――――①

int main(void) {
    /* 各生徒の点数 */
    int tensu[NINZU] = { 63, 75, 48, 92, 66 }; ―②

    /* ランク毎の人数集計結果 */
    int shukei[3] = { 0, 0, 0 }; ―――――――――③

    int i;
    /* 生徒数分繰り返す */
    for (i = 0; i < NINZU; i++) { ―――――――――④
        if (tensu[i] >= 70) {      /* 70点以上？ */
            shukei[0]++;
        }
        else if (tensu[i] >= 40) { /* 40点以上？ */
            shukei[1]++;
        }
        else{                      /* 40点未満   */
            shukei[2]++;
        }
    }
    /* 集計結果を表示する */
    printf("結果  良: %d人  可: %d人  不可: %d人¥n",
                shukei[0], shukei[1], shukei[2]);
    return 0;
}
```

```
int  shukei[3] = { 0 };
```

C言語の配列の初期化では、「{ }」の中に書いた初期値の数が「[ ]」で指定した数に満たない場合、残りの要素は0で初期化されます。上記の例では**shukei[3]**と書いているのに「{ }」の中身が1つですので、残りの2つ分は自動的に0の値になるのです。

配列の要素数をsizeofで求める

リスト3の④以降では、生徒の人数分繰り返して集計処理を行います。④では、forループの繰り返し条件を指定するために**NINZU**、つまり5という数値を指定しています。しかし、このように固定的な数値を使わなくても、配列**tensu**の要素数さえわかれば済むはずです。

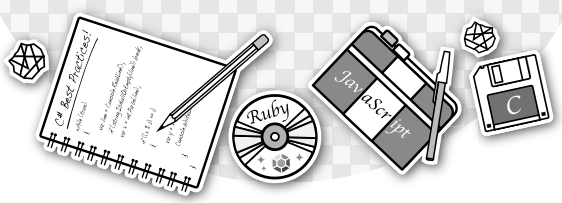
sizeof演算子を使って次のように記述すると、配列の要素数を求めることができます。

```
sizeof(tensu) / sizeof(tensu[0])
```

sizeof(tensu)は配列全体のバイト数を、**sizeof(tensu[0])**は1つの要素のバイト数を取得できます。したがって「全体のバイト数÷1要素のバイト数」を計算することで、配列の要素数が求められるのです。このように書いておくと、のちに配列の要素数が変わっても手直ししないで済みます。それでは、ここまでの改良点を反映したプログラムを**リスト4**に示します。なお、sizeof演算子で求めたサイズは、size_t型（符号なし整数型の1つ）で返ります。そのため、カウンタ変数**i**もsize_t型とすると、より適切なコードになります（**リスト4**の①）。

浮動小数点はちょっとクセモノ？

double型などの浮動小数点の数値を



使って計算を行うと、場合によって期待した結果が得られないことがあります。例を挙げて、その対策方法について説明します。

計算がうまくいかない!

リスト5は、0.0から2.0まで0.1ずつ増やした数を表示するプログラムです。①で、変数dが2.0を超えたらループを終了させているので、**図5**のような実行結果を期待しました。しかし、実際には**図6**の結果となり、2.0が表示されません。なぜこのような結果になるのでしょうか?

調べるために、**リスト6**のように変更してみました。①で小数点以下を20桁表示するようにして、②でループ終了後の変数dの値も見てみることにしました。実行すると、**図7**のような表示結果になりました。

浮動小数点の誤差に注意

図7の結果を見ると、きれいに0.1単位ずつ増えていないことがわかります。20回目の加算後の値が2.0を超えたため、繰り返し処理がそこで終了したのです。

このような動作を理解するには、浮動小数点の特性を知る必要があります。浮動小数点の内部構造は環境により異なりますが、すべての実数を正確に格納できる方式ではありません。0.1という数値も内部的には誤差が発生していて、**図7**の2行目を見るときっかり0.1でないことがわかります。

浮動小数点をカウンタとして使わない

浮動小数点の演算は誤差が発生する場合があるため、意図した結果が得られないことがあります。回数を数える目的としては、浮動小数点ではなく整数を使うようにしてください。整数を使うように書き換えたのが**リスト7**です。実行結果は**図5**のようになります。

▼リスト4　配列を使う処理(改良後)

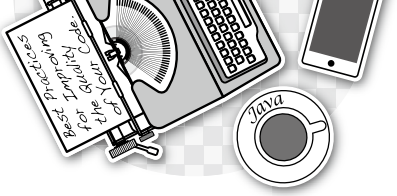
```
#include <stdio.h>

int main(void) {
    /* 各生徒の点数 */
    int tensu[] = { 63, 75, 48, 92, 66 };

    /* ランク毎の人数集計結果 */
    int shukei[3] = { 0 };

    size_t i; ―――――――――――――――――――――――――――――――――――①
    /* 生徒数分繰り返す */
    for (i = 0; i < (sizeof(tensu) / sizeof(tensu[0])); i++) {
        if (tensu[i] >= 70) {      /* 70点以上?   */
            shukei[0]++;
        }
        else if (tensu[i] >= 40) { /* 40点以上?   */
            shukei[1]++;
        }
        else{                      /* 40点未満    */
            shukei[2]++;
        }
    }
    /* 集計結果を表示する */
    printf("結果  良: %d人  可: %d人  不可: %d人¥n",
                shukei[0], shukei[1], shukei[2]);
    return 0;
}
```

▼リスト5　浮動小数点のカウンタ(正しく動かない例)

```
#include <stdio.h>

int main(void) {
    double  d;

    /* カウンタが浮動小数点(正しく動かない例) */
    for (d = 0.0; d <= 2.0; d += 0.1) { ――①
        printf("%.1f¥n", d);
    }
    return 0;
}
```

▼リスト6　浮動小数点のカウンタ(正しく動かない例の調査)

```
#include <stdio.h>

int main(void) {
    double  d;

    /* カウンタが浮動小数点(正しく動かない例の調査) */
    for (d = 0.0; d <= 2.0; d += 0.1) {
        printf("%.20f¥n", d); ―――――――――――①
    }
    printf("ループを抜けた後¥n%.20f¥n", d); ―②
    return 0;
}
```



C言語は進化してるの？

C言語はバージョンアップが頻繁に行われる言語ではありません。C言語の主な規格を**表1**に示します。最新の規格は通称C11と呼ばれるものですが、この規格に完全に対応するコンパイラは現時点で多くありません。

やっとC99が広まってきた

GNUコンパイラ(GCC)やMicrosft Visual C++などが数年前からC99の仕様を搭載し始めたことにより、C99を使う機会は広がってきました。しかし、組み込みシステムなどの開発では長年実績のあるANSI Cもよく使われています。以降では、C99以降の機能について一部を紹介します。

C++形式のコメント

C99以降では、「**/***」と「***/**」で囲む形式のコメントに加え、「**//**」形式のコメントが使えます(**リスト8**の①)。「**//**」以降行末までがコメントとして扱われます。

変数の宣言位置

ANSI Cでは、関数の中で使う変数はブロックの先頭で宣言する必要がありました。これはC言語の常識とも言えるルールでした。しかし、C99以降では、ブロックの途中で変数を宣言できます(**リスト8**の④)。また、forループ本体で使う変数をfor文の記述時に宣言できます(**リスト8**の③)。for文で宣言した変数はforループ内でのみ有効となります。

gets関数は使わないで

C言語の規格の話からは外れますが、gets関数の非推奨について説明します。gets関数は標準入力から1行分の文字列を取得してメモリに格納する関数ですが、格納先メモリのサイズを指定できないため簡単にバッファのオーバーフローを引き起こしてしまうという問題があります。したがってgets関数は使わずに、fgets関数を代用することが推奨されています(**リスト8**の②)。C11規格ではgets関数は廃止されました。代わりにgets_s関数が追加されています。ただし、gets_s関数はオプション扱いでコンパ

▼図5　リスト5で期待した実行結果(リスト7の実行結果)

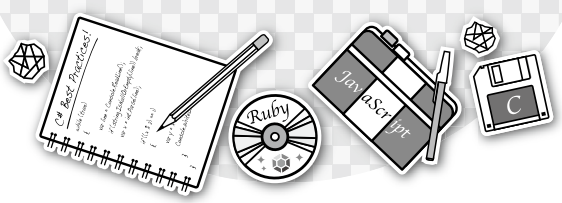
```
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2.0
```

▼図6　実際のリスト5の実行結果

```
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
```

▼図7　リスト6の実行結果

```
0.00000000000000000000
0.10000000000000000555
0.20000000000000001110
0.30000000000000004441
0.40000000000000002220
0.50000000000000000000
0.59999999999999997780
0.69999999999999995559
0.79999999999999993339
0.89999999999999991118
0.99999999999999988898
1.09999999999999986677
1.19999999999999995559
1.30000000000000004441
1.40000000000000013323
1.50000000000000022204
1.60000000000000031086
1.70000000000000039968
1.80000000000000048850
1.90000000000000057732
ループを抜けた後
2.00000000000000044409
```



イラによっては実装されない場合もあります。

C99で追加されたそのほかの機能

そのほかにも多くの便利な機能が追加されています。主なものを挙げると、新しい型(論理型、複素数型など)の追加、可変長配列、数学ライブラリ(math.h)の大幅な機能向上などです。

終わりに

本章では、C言語のプログラミングにおけるよい書き方や注意点などをいくつか紹介しました。コードを書くときから、読みやすさを考慮したり、不具合を作り込まないように意識したりすることは大切です。そうすることで、デバッグ、テスト、保守などの作業も、より効率よく進められるでしょう。SD

▼リスト7　整数のカウンタ(良い例)

```
#include <stdio.h>

int main(void) {
    int     i;

    /* カウンタが整数(良い例)*/
    for (i = 0; i <= 20; i++) {
        printf("%.1f¥n", i / 10.0);
    }
    return 0;
}
```

▼表1　C言語の主な規格

規格	制定年
ANSI C	1989年
ISO C99	1999年
ISO C11	2011年

▼リスト8　C99の仕様を取り入れたコード

```
#include <stdio.h>

#define BUFSIZE  128  // これはコメントです  ————①

int main(void) {
    char    buf[BUFSIZE];

    printf("入力してください -> ");

    if (fgets(buf, sizeof(buf), stdin) != NULL) { —②
        for (int i = 0; i < 3; i++) { ————————③
            printf("%s", buf);
        }
    }

    int     a; ——————————————————————————④
    a = 2;

    printf("a * a = %d¥n", a * a);

    return 0;
}
```

column フリーのコンパイラでC言語を始めよう

C言語のプログラムを動かすにはCコンパイラが必要です。以下に、無償で入手可能なCコンパイラをいくつか紹介します。なお、コンパイラの動作環境、ダウンロード方法、インストール方法の詳細については、各Webサイトを参照してください。

- Microsoft Visual Studio Express for Desktop
  http://www.visualstudio.com/products/mt238358
- MinGW - Minimalist GNU for Windows
  http://sourceforge.net/projects/mingw/
- Borland C++ Compiler 5.5
  http://www.embarcadero.com/jp/products/cbuilder/free-compiler
- Xcode(Mac App Storeから統合開発環境のXcodeを入手可能)
  http://www.apple.com/jp/osx/apps/app-store/

上記のリンク先は2016年2月時点の内容であり、変更される場合があります。コンパイラのインストール、プログラムのコンパイル／実行については、自己の責任に基づいて行ってください。

第2章 Java編

Best Practices for Improving the Quality of Your Code.

良いコーディングのさいしょの一歩

Author 石田 真彩（いしだ まあや）、長澤 太郎（ながさわ たろう）

動くコードが書けるようになったら、次はコードの読みやすさを意識してみましょう。本章では、変数や構造をほんの少し工夫することで、処理内容を明確にできる指針をいくつか紹介します。また、ラムダ式やOptionalも適所に活用できればワンランクアップです！

はじめに

本章ではJavaにおける「良いコード」について説明をしていきます。ここでは「良いコード＝保守性の高いコード」と定義することにします。保守性の高いコードとはどのようなコードのことを言うのでしょうか。本章では、時間をあけてコードを読んだときに「何を行っているのか理解しやすいコード」「不具合を発見しやすいコード」と定義し、進めていきます。

「良いコード」を書くためには、コード1つ1つの意味と意図をできる限り明確に表すことが重要です。意味や意図が伝わらないまま不具合改修や機能追加を行うと、新たな不具合が生まれる可能性が高くなってしまいます。しかしながら、システムが「良いコード」で書かれていれば、たとえば運用中に不具合が発生した場合も、スピーディな原因の解明と改修に貢献することができるでしょう。

「良いコード」のアプローチはさまざまありますが、今回はその中からいくつかピックアップしてご紹介します。また、「良いコードの“書き方”“考え方”」にフォーカスして解説を行うため、全体を通して文法についての説明は最小限になっています。もし出てきた文法がわからない方は、別途文法について詳細が書かれた本などを参照してみてくださいね。

変数の初期化と変数のスコープ

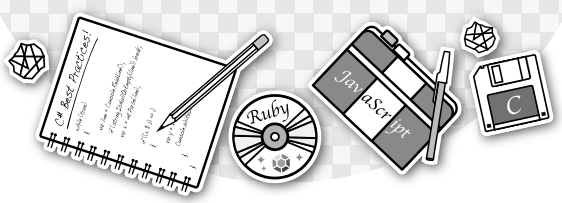
適切なタイミングで必要な処理だけを書くことは「良いコード」の重要なアプローチの1つです。この節では、「不要な処理をなくしていくこと」「必要最小限のスコープで処理を実行していくこと」の2つの視点から、変数について考えていきます。

その初期化、必要ですか

まずは「不要な処理をなくしていくこと」という観点から変数の初期化について考えていきましょう。

リスト1が最初のサンプルです。よく見てみるとevaluationは、**必ず何かしらの値が入力されます**。一番最初に設定した空文字が使用されることはありません。つまり、このコードではevaluationの初期化は不要な処理であるといえます。使われていない初期化処理は記述しないようにしましょう（**リスト2**）。

今回はStringを例に挙げましたが、ListやMapなども同様で、使っていないインスタンスの生成をしてしまっているコードを見かけることがあります。とくに、実際に変数の宣言をするコードと代入するコードが離れた場所にあ



る場合は不要な初期化を書きがちです。一度書いたコードを見直すときには注意して確認すると良いでしょう。

最小限のスコープ

次は「必要最小限のスコープで処理を実行していくこと」について考えてみましょう。先ほど、変数の宣言とその変数へ代入をするコードが離れているときに、不要な初期化が起こりやすいと書きました。そもそもこの宣言と代入が離れた状態を改善することはできないか、**リスト3**で検討してみましょう。

countriesは①で変数の宣言をした後、②まで使用されません。使用されないのであれば使用する②の直前で変数の宣言をしましょう(**リスト4**)。直前で宣言をすることで、不要な処理の介入を防ぐことができます。保守時のコードの理解のしやすさにもつながりますね。

このように変数のスコープを最小限にとどめられるように考えることは、「良いコード」を書くにあたりとても重要です。スコープが広くなってしまうと、その分考えなければいけないことが増えてしまいます。もちろんすべてのコードが今回の例のようにまとめられるわけではありませんが、できる限り最小のスコープで変数を利用するよう心がけましょう。

メソッドを短く保とう

この節は「コードを理解しやすくする」アプローチの1つである「処理単位を短くする」ことについて考えていきます。コードが理解しやすければ後で保守するときも楽になりますね。

処理単位を小さく

たとえば、500行のコードからなるクラスがあったとします。そのクラスの中に1メソッドしかなかったらどうでしょう。想像してみてください。メソッドの先頭から読み始めたものの、なかなかメソッドは終わりません。やっと終わりにたどり着いたころには、メソッドの始めに書かれていた内容なんて忘れてしまっています。メソッドの始めに使用した変数の値をメソッドの終わりで使用していたら、「この変数には何が入っているんだっけ……？」となっても不思議ではありません。

では、この500行のコードが20個のメソッドに分かれていたらどうでしょうか。各メソッドに長短はありますが、おおむね20〜30行くらいになっているはずです。この程度のメソッドの長さであれば、読む意欲も湧きますし、変数の値を忘れることもないでしょう。

長いメソッドを読むと心が折れそうになる原

▼リスト1　不要な初期化があるコード

```
String evaluation = "";
if (score > border) {
  evaluation = Judgment.getSuccess();
} else {
  evaluation = Judgment.getFailure();
}
```

▼リスト2　不要な初期化のないコード

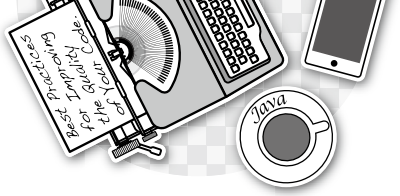
```
String evaluation;
if (score > border) {
  evaluation = Judgment.getSuccess();
} else {
  evaluation = Judgment.getFailure();
}
```

▼リスト3　広いスコープ

```
List<Country> countries;  ……①

String continent = getContinent(country.getName());
dialog(continent);

countries = getCountries(continent);  ……②
```

▼リスト4　リスト3のスコープを最小限にしたコード

```
String continent = getContinent(country.getName());
dialog(continent);

List<Country> countries = getCountries(continent);
  ↑使う個所で宣言する
```



因として、「その処理のゴールが見えない」ということが挙げられます。長いメソッドは、どこまででひとまとまりなのかを再考しましょう。ひとまとまりごとにメソッドにすることで、その処理のゴールが見えやすくなります。たとえば、for文の中でifを使った分岐をしている場合や、if文の中で長く複雑な処理をしている場合はメソッド化を検討してみるべきです。

本来ならば、長いメソッドの例を使いたいのですが、記事の分量制限もあるため今回は短いコードを例にします(**リスト5**)。しかし、ここで示した手法は、長いメソッドを分割する場合でも活用できるはずです。一緒に「ひとまとまり」を探す練習をしてみましょう。

リスト5のメソッドの中は次の処理に分けることができます。

・名前を出力する処理
・今住んでいる国を判断し、文言を出力する処理
　- 今住んでいる国を判断する処理
　- 判断結果を出力する処理

「自己紹介を出力する処理」として1つにまとまっていたコードを、4つの処理に分けることができましたね。ではこれをそれぞれメソッドにしてみましょう(**リスト6**)。

今回は短いコードを分割しただけなので、ただコードが長くなっただけのように見えます。しかし、処理のまとまりを見つけ、メソッド化することでコードの理解しやすさは向上します。処理のまとまりによっては、クラス化することも検討してみるとよいでしょう。

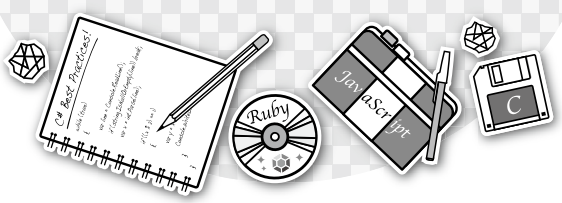
ちなみに筆者はだいたい10～20行、長くても30行に収まるよう実装を行っています。あくまで個人の例になりますが、参考にしてみてください。「この処理は分けられるかな？」という視点を持ち続けながらコードを書くことが重要なのです。

▼リスト5　メソッド分割前

```
public void printSelfIntroductions(Profile profile) {
    System.out.println("私は" + profile.getName() + "です。");

    if (profile.getStay() == Country.JAPAN) {
        System.out.println("日本に住んでいます。");
    } else {
        System.out.println("日本以外の国に住んでいます。");
    }
}
```

▼リスト6　メソッド分割後

```
public void printSelfIntroductions(Profile profile) {
    //名前を出力
    printName(profile);
    //今住んでいる国を出力
    printStay(profile.getStay());
}

private void printName(Profile profile) {
    System.out.println("私は" + profile.getName() + "です。");
}

private void printStay(Country country) {
    System.out.println(getStayMessage(country));
}

private String getStayMessage(Country country) {
    return (country == Country.JAPAN)
        ? "日本に住んでいます。"
        : "日本以外の国に住んでいます。";
}
```



finalな変数を検討する

変数の値が変わらないことは良いことです。なぜなら、値が変更されないことを知っていれば、その変数を注意深く観察する労力が軽減できるからです。逆に言うと、変数が変更される可能性があると、注意深く読み進める必要があるということです。メソッドを読み進める際に、あるローカル変数が複数個所で参照されることはよくありますが、それぞれの場所で異なる値になっていたら、理解するために時間がかかりますし、誤りの元になります。

ここで主張したいことは、ローカル変数にfinalを付けて変更を許可しないようにすることです。これは変数への再代入をしないということと同じ意味です。finalが付いたローカル変数は再代入される可能性がないので、複数個所で参照されていても、どこでも常に同じ値です。

リスト7はログイン処理を行うメソッドです。ログインIDを取得し、アカウント情報を取得します。その後パスワードを取得し、該当アカウントのパスワードと照合します。最後に、自動ログインを有効にするかどうかのフラグを文字列として取得し、アカウント情報に結びつけています。

このコードの中で、変数sは3度代入されています。**リスト7の**①のログインIDを取得するために参照される個所と、**リスト7の**②のパスワード照合で参照される個所では、同じ変数sでも値が異なります。コードを読む際には、都度「s =」のように代入している部分を確認する必要があることを意味します。

このコードから再代入を排除したバージョン(**リスト8**)を見てみましょう。変数loginId、password、autoLoginEnabledを導入し、それぞれ一度だけ代入を行っています。また、再代入がされ得ないことを表明するためにfinalを付けています。loginIdは最初に代入された値から変更されません。このメソッド内のどこでloginIdを参照しても、常に同じ値です。

原則としてfinalなローカル変数を定義するようにしましょう。しかし、リストの各要素を走査して1つの値にまとめあげるような処理(たとえば整数リストから合計を計算するような処理)では、変数への再代入を受け入れるべきです。メソッドの再帰呼び出しをうまく使えば、再代入を排除することは可能な場合もあります。しかしこの場合はスタックオーバフローの危険性があるため注意が必要です。

コードを読み進めるうえで、変数の値が何回も変更されることは、考慮すべきことが増えて可読性低下につながります。再代入を避けて、シンプルなコードに保ちましょう。finalを変数に付けることで、再代入が不可能な変数であることを保証できます。

▼リスト7　ログイン処理を行うメソッド

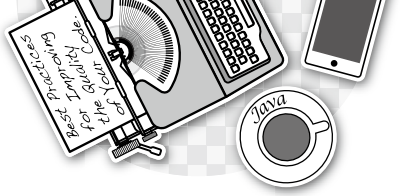
```
private LoginResult login() {
  String s = getLoginId();
  Account account = accountRepository.➥
findByLoginId(s);  ①
  if(account == null)
    return LOGIN_ID_NOT_FOUND;
  s = getPassword();
  if(notEquals(s, account.getPassword()))  ②
    return WRONG_PASSWORD;
  s = getAutoLoginEnabled();
  account.setAutoLoginEnabled(s.equals(AUTO_➥
LOGIN_ENABLED));
  return SUCCESS;
}
```

▼リスト8　再代入を排除しfinalな変数に

```
private LoginResult login() {
  final String loginId = getLoginId();
  Account account = accountRepository.➥
findByLoginId(loginId);
  if(account == null)
    return LOGIN_ID_NOT_FOUND;
  final String password = getPassword();
  if(notEquals(password, account.➥
getPassword()))
    return WRONG_PASSWORD;
  final String autoLoginEnabled = ➥
getAutoLoginEnabled();
  account.setAutoLoginEnabled(autoLogin➥
Enabled.equals(AUTO_LOGIN_ENABLED));
  return SUCCESS;
}
```

今すぐ実践できる良いプログラムの書き方

［C、Java、C#、Ruby、JavaScript］



ラムダ式を使ってみよう

Java SE 8の大きな変更点の1つにラムダ式があります。「->」という記法や今までのJavaと異なる考え方にとまどっている人も多いのではないでしょうか。最初はとっつきにくいかもしれませんが、慣れればコード作成の強い味方になります。ラムダ式を使って、すっきりとしたコードを目指しましょう。

ラムダ式

ラムダ式は関数型インターフェースを関数として扱える記法です。関数型インターフェースとは、実装が必要なメソッドが1つだけであるインターフェースです。既存のライブラリにも、すでに関数型インターフェースは存在しています。たとえばjava.lang.Runnableインターフェースやjava.util.Comparatorインターフェースがこれに該当します。

ラムダ式の記述は次のとおりです。

［様式例1　ラムダの式］

```
（引数）-> {実行したい処理}
```

引数は2つでもいいですし、引数なしでも構いません。引数が1つの場合は小かっこ（ ）は省略しても構いません。同様に、実行したい処理が1つなのであれば中かっこ{ }は省略してもよいです。

［様式例2　引数が2個の場合］

```
(x, y) -> {実行したい処理}
```

［様式例3　省略できるかっこ（引数が1個の場合・実行したい処理が1つの場合はそれぞれかっこを省略できる）］

```
引数 -> 実行したい処理
```

ラムダ式に書き換えてみよう

まずはJava SE 7までのRunnableインターフェースのコード（**リスト9**）を見てみましょう。匿名クラスで実行処理が記述されており、冗長な印象を与えますね。

では、**リスト9**をラムダ式にしてみましょう。①にあるRunnableインターフェースに渡している引数と②にある実行したいメソッドを使ってラムダ式を作ります。Runnableインターフェースの場合はインターフェースに渡す引数が存在しないため、空の小かっこ（ ）を記述します（**リスト10**）。

newや@Overrideなどがなくなり、だいぶすっきりとし、かつ本当にやりたい処理だけが残りましたね。

for文を書き換えてみよう

Java SE 7まではインターフェースにメソッドを追加することはできませんでしたが、Java SE 8から追加できるようになりました。この処理を利用して、コレクションや配列に使

▼リスト9　今までの実装

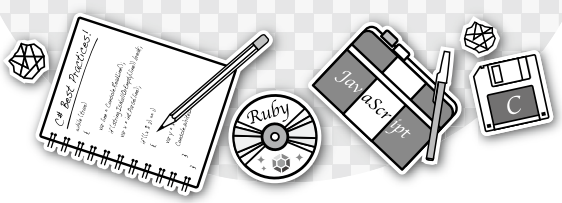
```
Runnable threadTask = new Runnable() {
  @Override
  public void run() {
    System.out.println("スレッドを実行します");
  }
};
```

①（下線部）
②（太字部）

▼リスト10　ラムダ式を使った実装

```
Runnable threadTask = () -> {System.out.println("スレッド実行します");};
```

①' ……リスト9の①に相当
②' ……リスト9の②に相当



用されているIterableインターフェースには、forEachメソッドが追加されています。forEachメソッドはラムダ式を使うことでfor文処理を記述することができます。

リスト11のラムダ式は、引数が1つ・実行したい処理が1つのパターンです。様式例3のパターンで記述されています。複雑な処理条件が発生する場合などはほかの実装方法をするべきです。しかし、たとえば今回のようにList全体に同じ処理を1つ行う程度の内容であれば、forEachメソッドの利用を検討できます。

このほかにもStream APIなどでラムダ式は多用されます。覚えておいて損はない文法です。今までJavaをやってきた方で、ラムダ式の考え方にとまどっているのであれば、まずは今までの書き方をラムダ式に書き換える練習から始めるととりかかりやすいです。ラムダ式を使い、よりシンプルなコードを保てるようにしましょう。

Optionalを使った nullとの付き合い方

Optional

StringやIntegerなど、Javaで使える「型」にはnullを設定することができます。しかし、場合によってはNullPointerExceptionなどのシステムエラーを引き起こしてしまいます。そのため、今までは@NotNullを使用したり、値のチェック処理を都度実装したりしていました。

Java SE 8で導入されたOptionalは、nullを取り扱わないようにしようという考え方を持つクラスです。オブジェクトを1つだけ保持できるコンテナであり、保持したオブジェクトを別のオブジェクトに置き換えることはできません。今までは、メソッドに何も返すものがない、つまり値が存在しないということを表現するためにnullを返していました。しかしOptionalを使用することで、nullを一切使用せずに値が存在しないことを表現できます。たとえば、そのメソッドの返り値がOptionalであったならば、その返り値は「nullではない」ということを明示できるのです。

想像してみてください。値が存在しないことがあるメソッドの返り値がすべてOptionalで返されます。また、必ず値が返されるメソッドの返り値はOptional以外の型（StringやInteger）が返されます。そんなシステムであれば、メソッドを呼び出す側でnullが入ってくるか否かを気にする必要はありません。気を付けなければいけないことが減ることは、コードの読みやすさや理解のしやすさにつながります。

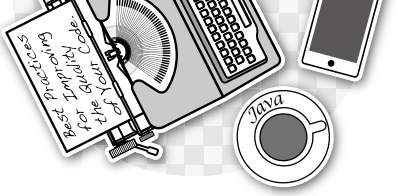
実際にOptionalを使う例を見てみましょう（**リスト12**）。

▼リスト12　Optionalを使用したコード例

```
interface User {
  Optional<String> getAddress();
}

private void showUserAddress(final User user) {
  Optional<String> address = user.getAddress();

  showDialog(address);
}
```

▼リスト11　拡張for文の書き換え

```
今までの書き方
for (Profile profile : profiles) {
                ③       ④
    System.out.println("名前は" + profile.getName());
                      ⑤
}

forEachメソッドを使った書き方
  profiles.forEach(profile -> System.out.println("名前は" + profile.getName()));
    ④'              ③'                          ⑤'
```



showUserAddressメソッドから呼び出したとき、getAddressメソッドからは必ず値が返るということをコード上に明示できたことになります。nullが返えら「ない」ことをコードとして示しておくことは、システムの仕様をよく知らない人がコードリーディングするにあたり、貴重な情報となります。

リスト12では、値が存在しない可能性がある値をOptionalで表しました。必ず何かしらの値が返却されると決まっている（わかっている）値についてはOptionalにする必要はありません。すべての値をOptionalにすればいいというわけではないのです。

リスト13では、addressのみOptionalになっています。これはすなわち、firstNameとlastNameは必ず値が入るものであり、addressは値が設定されない可能性があるということを示しています。値が設定されない可能性がある値のみOptionalを使用することで、想定される値の形（nullが設定されることがあるのか否か）を伝えることができるのです。

Optionalの生成

Optionalには3つのファクトリメソッドがあります。値が存在しないことを表現するOptionalを取得する場合はemptyメソッドを使用します。値を保持したOptionalを生成したい場合はofメソッドを使用します。ofメソッドの引数にnullが指定されてしまった場合はNullPointerExceptionが発生します。そして3つめのofNullableメソッドは、ofメソッドと似ていますが、nullを引数に取る点で異なります。ofNullableメソッドの引数にnullが渡された場合は、空のOptionalオブジェクトを返します。すなわちemptyメソッドと同じ結果となります。

たとえば**リスト14**のように、nullである可能性があるuserOrNullをOptionalに包むことができます。

▼リスト13　Optionalの使い分け

```
public class User {
    String firstName;
    String lastName;
    Optional<String> address;
}
```

▼リスト14　生成

```
User userOrNull = findUser();
Optional<User> optionalUser = (userOrNull != null)
  ? Optional.of(userOrNull)
  : Optional.empty();
// ofNullableを用いて同じことができる
// Optional<User> optionalUser = Optional.ofNullable(userOrNull);
```

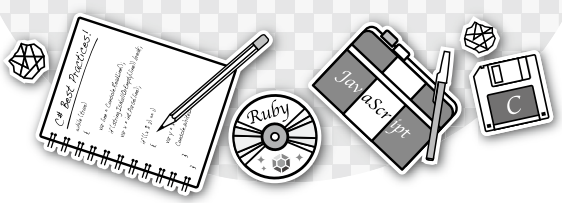
Optionalの利用

Optionalの使い方について見ていきましょう。Optionalに用意されているisPresentメソッドは、Optionalの値の有無を確認できます。またgetメソッドでは、Optionalの値を取得することができます。この2つのメソッドを使うと、**リスト15**のように「値が存在した場合にのみ処理を実行する」という実装をすることができます。

しかしgetメソッドは、値が存在しない場合にNoSuchElementExceptionを投げるので注意が必要です。値がないことによる例外の発生を防ぎたいためにOptionalを利用するにもかかわらず、例外が発生してしまう可能性のあるgetメソッドを使うことは、今まで説明してきたOptionalを使った良いコードの概念を覆す恐れがあります。そのため、できれば利用を避けたいメソッドです。

そこで利用をお勧めしたいのが、ifPresentメソッドです。ifPresentメソッドを使うと、**リスト15**の処理と同じ処理をgetメソッドを使わずに実装できます（**リスト16**）。

getメソッドも使用せず、if文を使うことも



ないのでリスト15より良いコードと言えます。Optionalに値がある場合にのみ処理を実行したい場合は利用を検討してください。

このほかにもOptionalにはmapメソッドやflatMapメソッドなどが用意されています。用途に応じて使い分けてみてくださいね。今までOptionalを知らなかった方も、ぜひ用途に応じてOptionalを使って「良いコード」を作っていきましょう。

enumのすすめ

Java SE 5で導入された列挙型(enum)は、同じカテゴリに属した複数の定数を1つのまとまりとして扱うことができるしくみです。この列挙型を適切に使うことにより、後でコードを読む人にそのコードが持っている意味が伝わりやすくなります。

列挙型に置き換えてみよう

「同じカテゴリに属した複数の定数」とはどんな定数を指すのでしょうか。たとえば、リスト17のようなコードを挙げることができます。

この定数は、3つとも天気というカテゴリを表す定数です。リスト17のような定数が定義されていた場合、利用方法としてはリスト18のようなコードが想定されます。

リスト18の良くない点は、処理の判定が数値で行われていることです。元々はただの数字であり、意味を持っていません。数字に定数名で半ば無理やり意味を持たせて判定を行っています。本来ならば天気が晴れなのか曇りなのかを判定したいはずなのに、それ自体には意味のない数値が判定に介入してしまっているのです。意味のない数値とはつまり、「WEATHER_RAINY = 99」に変更したとしても判定処理そのものの意味や内容が変わらないということです。

しかも、この定数が同じカテゴリとして扱われているのだということを示せるものは定数名だけです。残念ながら、文法的な制約はありません。この例では「WEATHER_」という文字列を付与してカテゴリをあらわそうとしています。しかし、たとえば後からこの判定に「雪」という条件を追加する場合に「SNOW」という定数名で判定条件が追加されてしまっても、コード上は警告やエラーにはなりません。

このような意味のない値を取り除くことは「良いコード」にするための1つの手段となります。

上記例のように複数の定数を1つのカテゴリとして明示したい場合に使えるのが列挙型です。リスト17、18を列挙型に置き換えると、次の問題点を改善することができます。

・意味のない値が介入すること
・複数の定数が同じカテゴリの定数であること
　を文法的に示せないこと

▼リスト15　値が存在した場合にのみ処理を実行する

```
if (optionalUser.isPresent()) {
  User user = optionalUser.get();
  System.out.println(user);
}
```

▼リスト16　ifPresentメソッドを利用する

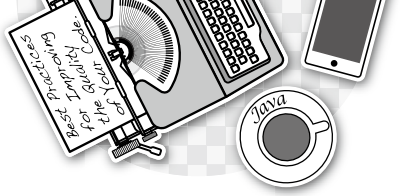
```
optionalUser.ifPresent(user -> System.out.println(user));
```

▼リスト17　intを使った定数

```
public static final int WEATHER_SUNNY = 1;
public static final int WEATHER_CLOUDY = 2;
public static final int WEATHER_RAINY = 3;
```

▼リスト18　int定数の利用例

```
int weather = getWeather();

if (weather == WEATHER_SUNNY) {
    return "晴れ";
} else if (weather == WEATHER_CLOUDY) {
    return "曇り";
} else if (weather == WEATHER_RAINY) {
    return "雨";
} else {
    return "観測不能";
};
```



では列挙型に置き換えた例を見てみましょう（**リスト19**）。

列挙型は任意の名前をつけることができます。今回の例でいうとWeatherです。このため、intの定数を使っていたときのような、都度名前に「WEATHER_」とつけるといった規約を設ける必要がありません。一度列挙型に名前を設定してしまえば、命名に気をとられることがなくなります。そのうえ「同じカテゴリの定数である」ことも明示し続けることができるのです。

また、最大の難点であった「意味のない値」を持つ必要がありません。SUNNYはSUNNY、と余計な値を持つことなく判定処理を行うことができます。

列挙型はswitch文の分岐にも使用できます（**リスト20**）。

終わりに

いかがでしたでしょうか。「良いコード」といってもさまざまなアプローチがあります。小さな改善も「良いコード」の大事な1アクションです。改善を積み重ねて、より良いコードを作っていきましょう。SD

▼リスト19　列挙型に修正する

```
public enum Weather {
    SUNNY,
    CLOUDY,
    RAINY;
}

private String getWeatherName(Weather weather) {
        if (weather == Weather.SUNNY) {
            return "晴れ";
        } else if (weather == Weather.CLOUDY) {
            return "曇り";
        } else if (weather == Weather.RAINY) {
            return "雨";
        } else {
            return "観測不能";
        }
}
```

▼リスト20　switch文利用例

```
public enum Weather {
    SUNNY,
    CLOUDY,
    RAINY;
}

private String getWeatherName(Weather weather) {
    switch (weather){
            case SUNNY:
                return "晴れ";
            case CLOUDY:
                return "曇り";
            case RAINY:
                return "雨";
            default:
                return "観測不能";
        }
}
```

第3章　C#編

Best Practices for Improving the Quality of Your Code.

言語機能の進化から学ぶ「良いコードの書き方」

Author 岩永 信之（いわなが のぶゆき）

C#は時代とともに新たな機能を積極的に取り入れてきました。その中には、一般的に良いとされるコードの書き方のパターンが、言語機能として取り入れられたものも、たくさんあります。本章ではC#の言語機能の進化を見ていきながら、良いコードの書き方とは何かをひも解いていきます。

はじめに

プログラミング言語は日々進化しています。1つのプログラミング言語であってもバージョンアップを重ねていますし、新しいプログラミング言語も登場してきています。

進化によってより良いコードが書けるようになっているわけですが、では、そもそもより良いコードとはどういうことなのでしょう。いくつか指標はありますが、3点紹介しましょう。

債務分割

1人の人間が一度に取り組める作業には限界があります。複雑な問題に対しては、問題を小さく分割して1つずつ解決していくしかありません。複数人での分業になるかもしれません。あるいは、1人での開発であっても過去の作業は忘れていくものなので、他人が読み書きする可能性のあるものと考えて作るべきです。そこで重要になるのは、（今の）自分のコード変更が他人（あるいは過去の自分）のコードに影響しないようにすることです。

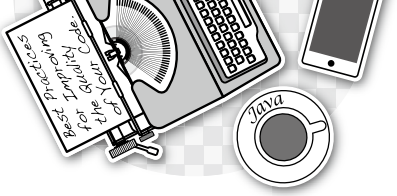
意図の記述

何かの課題に取り組む際には、why（なぜ必要かという理由）、what（何をしたいかという意図）、how（どう実現するかという手順）という3段階のものを考える必要があります。低水準（コンピュータのハードウェアに近いという意味）なプログラミング言語ほどhow（手順）を事細かに書く必要があります。しかし、重要なのはむしろwhat（意図）で、それを伝えるために、別途、自然言語での注釈（コード中のコメントや外部ドキュメント）が必要になります。理想的には、コードを見ただけでwhatがわかるべきです。whatを直接コードとして書けるべきで、コメントなどにはwhyだけ残るのが理想形となります。

性能

現在では、前述の債務分割や意図の記述が重要で、それを優先して性能が犠牲にされる場合も多いです。しかし、ものには限度があります。債務分割するうえでは良くても、数メガバイト単位でメモリを余計に使ったり、プログラムが数秒間フリーズしたりするようではさすがにつらいです。また、債務分割しやすさなどを犠牲にせずに単純に性能を向上できるのなら、積極的にそういう書き方をすべきでしょう。

これらの重要性は今に始まったわけではなく、大昔からずっと変わっていません。古いプログラミング言語には、当然、言語機能が欠けてい



たわけですが、だからと言って何もしていなかったわけではありません。良いとされるパターンが議論され、定着してきました。構造化プログラミングやオブジェクト指向プログラミングというような、現代的なプログラミングでは当たり前の概念も、かつてはパターンから始まり、やがてプログラミング言語の機能として取り入れられてきたものです。

かつては、プログラミング言語の文法を覚えたら、続いてパターンを学ぶ必要がありました。「デザインパターン」と付く書籍は必読書と言われていました。今は、その多くのものがプログラミング言語自体の機能になっています。逆に言うと、プログラミング言語を学ぶ中から、良いとされるパターンを学ぶことができます。

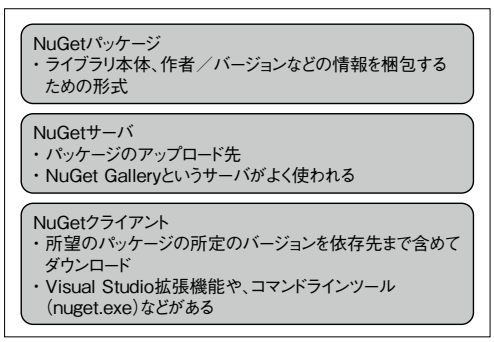
本章は、C#について次のものを取り上げます。

・パッケージ管理
・プロパティ
・イテレータ
・データ処理
・非同期メソッド
・nullの撤廃

パッケージ管理は、コードを書く前に、まず、そもそもコードを書かないことを考えるべきという話です。プロパティは、C#の歴史とともに何度かの機能追加があったものです。パターンができ、それが言語構文として取り入れられる過程の繰り返しがよくわかると思い、ここで紹介しています。残りは、2000年代後半からの数年で急激に進歩した分野です。「モダンな書き方」を目指すうえで外せないポイントでしょう。

C#の現在の最新バージョンは6です。また、C# 7に向けて、新機能の提案やディスカッションがGitHub上で行われています。本稿では一部、このC# 7の機能を先取りして紹介していますが、あくまで提案です。取りやめや延期(7ではなく8や9)になる可能性もあるのでご注意ください。

▼図1　NuGetパッケージ管理のしくみ



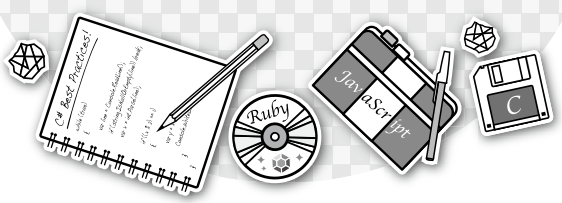
パッケージの利用

本特集のテーマは「良いコードの書き方」ですが、コードを書き始める前にまず考えるべきことは、「いかにコードを書かないか」です。同じような問題に直面し、同じようなコードを過去に書いたことがある人がいて、そのコードをライブラリ化して公開してくれているかもしれません。

多くのプログラミング言語で、他人が書いたコードを簡単に使えるように、パッケージ管理システムが整備されています。ここで言うパッケージとはライブラリ自体やその付属データ、作者やバージョンなどに関する情報を梱包したものです。そして、パッケージ管理に必要な一連のシステム、すなわち、作ったパッケージをアップロードするサーバ、サーバ上のパッケージを検索する機能、そのパッケージの作者やライセンスを確認する機能、所望のパッケージの所定のバージョンを依存先まで含めてダウンロードする機能などが提供されています。

.NETでは、NuGetと呼ばれるパッケージ管理システムを使います。パッケージをアップロードする先のサーバは設定変更もできますが、最もよく使われるのはNuGet Gallery[注1]というサーバです。まとめると、**図1**のようになります。

注1） https://www.nuget.org/



誌面の都合で詳細は省きますが、現在では、NuGet Gallery上のパッケージも充実し、NuGetクライアントもこなれて、Web上でのNuGetに関する情報も増えていて、とくに苦労することもなくNuGetを使えるでしょう。

Visual Studioが標準で対応しており、また.NET Core(クロスプラットフォーム向けのオープンソースな.NET実装)では実行環境自体や標準ライブラリも含めてNuGetでパッケージ管理しています。その結果、NuGetは.NETの根幹をなす重要な技術となっています。

プロパティ

プロパティは、クラスの実装側にとってはメソッドのような振る舞いを書け、クラスの利用側にとってはフィールド(変数)の読み書きのように書けるメンバ(構成要素)です。

C#のプロパティ構文は、C#のバージョンアップに伴い何度か機能追加されています。時とともに徐々に、より良い書き方や、典型的な書き方が明らかになった結果です。

そこでここでは、C#のプロパティ構文の変遷と、それがどういう要件に基づいているのかを説明していきます。

getter/setter

C#以前からある良いとされる習慣の1つに、「クラスの持つデータは必ずメソッドを通して返せ、フィールドを公開するな」というものがあります。メソッドを通すことで次のような利点があるためです。

- 実装方法を選べる
- 単純なデータの読み書きだけではなく、追加の処理を挟める
- 実装方法を後から変えても、クラス利用側への影響が出ない
- virtualにすることで、派生クラスで挙動を変更できる

そこで、たとえばXというデータを読み書きする際には、GetX、SetXというメソッドを介する習慣ができました。これらのメソッドをそれぞれgetter/setterと言い、2つ合わせてアクセサー(accessor)と呼びます。単純なデータの読み書きであっても、次のようにGet/Setメソッドを書くべきということです。

例1 Get/Setメソッド
```
class Sample
{
    private int _x;
    public int GetX() { return _x; }
    public void SetX(int x) { _x = x; }
}
```

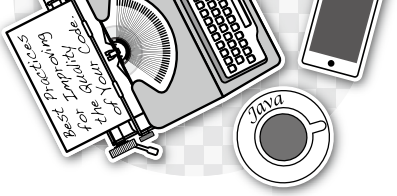
このような書き方には前述のようなメリットがある一方で、クラス利用側のコードが煩雑になるという問題があります。たとえば、Xの値に1を加えるだけでも、次のような書き方が必要になります。

例2 クラス利用側のGet/Setメソッド
```
var s = new Sample();
s.SetX(s.GetX() + 1);
```

そこで、C#では、プロパティという構文を用意しました。次のように書きます。

例3 C# 1.0のプロパティ
```
class Sample
{
    private int _x;
    public int X
    {
        get { return _x; }
        set { _x = value; }
    }
}
```

get、setに続けて、メソッド的に振る舞いを書けます。一方で、利用側のコードは例4のように、フィールドの読み書きと同じように書けます。



例4 クラス利用側のプロパティ参照
```
var s = new Sample();
s.X += 1;
```

アクセシビリティの変更（C# 2.0）

データに対して、読む（get）のと書く（set）のとではだいぶ重みが違います。一般に、書き込みのほうが慎重に行う必要があります。多くの場合、読み取り（get）だけを公開（public）し、書き込み（set）はクラス内にとどめる（private）ことになるでしょう。

そこでC# 2.0で、プロパティのgetとsetのアクセシビリティを別々に設定できるようになりました。たとえばgetだけpublicにして、setをprivateにするには例5のように書きます。

例5 getとsetで異なるアクセシビリティを指定
```
class Sample
{
    private int _x;
    public int X
    {
        get { return _x; }
        private set { _x = value; }
    }
    public Sample(int x) { _x = x; }
}
```

自動プロパティ（C# 3.0）

あとから実装方法を変更するかもしれないのでメソッド（のように振る舞えるプロパティ）を使うべきといっても、実際のところ、例3や例5がそうですが、ほとんどのプロパティは単純なフィールドの読み書きです。このような書き方は、「あとから変えるかもしれない」という心配のためだけに書くには少し煩雑過ぎます。

そこで、C# 3.0では、自動的に上記のようなフィールドとそれに対する読み書きを生成する「自動プロパティ」（auto property）という機能が追加されました。例6のように、getやsetの後ろのブロックを省略することで自動プロパティになります。

例6 自動プロパティ
```
class Sample
{
    public int X { get; private set; }
    public Sample(int x) { X = x; }
}
```

単純なフィールドの読み書きでいい間はこの書き方をし、追加の処理が必要になった際には自動実装ではない以前どおりのプロパティに変更します。

get-onlyプロパティ（C# 6）

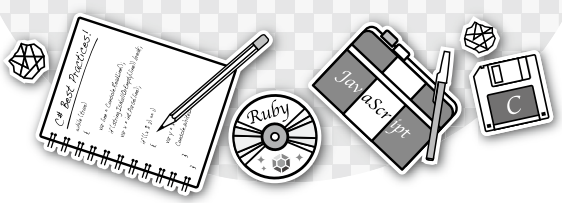
繰り返しになりますが、一般に、データの書き込みには慎重になるべきです。極端に言うと、コンストラクタでだけ値を代入して、ほかの場所では書き換えないほうがいいことが多いです。こういう書き換え不能なデータのことをimmutable（変更不能）なデータと呼びます。

C# 5.0までは、immutableであることを確実に保証するには、例7のような書き方でプロパティを作る必要がありました。

例7 C# 5.0までのimmutableプロパティの書き方
```
class Sample
{
    private readonly int _x;
    public int X { get { return _x; } }
    public Sample(int x) { _x = x; }
}
```

時代とともに、データをimmutableにしておくことの良さが一般に広まってきました。C#でも、プロパティをimmutableに書くことが増えています。

そこで、C# 6ではimmutableなプロパティを書きやすくするため、get-onlyプロパティという構文が追加されました。例8のように、getだけを書くことで、前述のようなreadonlyフィールドを自動生成してくれます。書き込みはコンストラクタ内でだけ行えます。



例8 get-onlyプロパティ
```
class Sample
{
    public int X { get; }
    public Sample(int x) { X = x; }
}
```

レコード型(C# 7)

immutableなプロパティは、値の初期化をコンストラクタで行う都合上、プロパティに対応するコンストラクタ引数が必ず必要になります。前項の例で言うと、プロパティXに対して引数xがあります。これは、大文字／小文字が違うくらいで、ほぼ同じものを何度も書かされている状態です。

そこで、C# 7では例9のような書き方で、コンストラクタとimmutableなプロパティを自動生成する構文が追加される予定です。

例9 レコード型(予定)
```
class Sample(int X);
```

X以外のメンバを書きたいときだけ追加でクラス本体を書きます。immutableなプロパティだけでいい場合には、この例のように;だけを書いて、本体を省略します。

この構文をレコード型(record type)と呼びます。ここでのレコード(record：記録、1件のデータ)という言葉は、単純なプロパティしか持たず、純粋にデータを表現するための型という意味です。

イテレータ

プログラムを書く際に頻出するものの1つが、配列やリストなどのデータ列(sequence)に対する操作です。C#の場合、データ列を表す型(コレクション)にはIEnumerable<T>インターフェース(System.Collections.Generic名前空間)を実装しておくことが一般的です。IEnumerable<T>を介することで、どんなデータ列に対しても同じコードでデータ処理ができます。**リスト1**に簡単な例を示します。

このように、データ列を処理する側はきれいに書けます。しかし、面倒なのはその逆で、データ列を作る側です。

IEnumerable<T>の実装

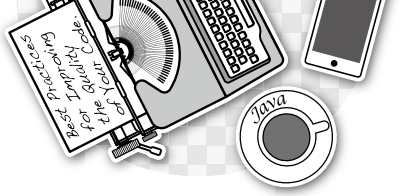
C#では、データ列を作る側を簡単化するために、イテレータと呼ばれる機能が備わっています。しかし、先に、イテレータを使わない場合の説明をしておきます。簡単な例として、同じ数字を指定した回数繰り返すことを考えてみましょう。Repeatと名付けます。

最も単純な実装は、**リスト2**に示すような、指定した長さ分の配列を作ってしまうことです。

コードはシンプルですが、この方法の問題は、

▼リスト1　IEnumerable<T>を介したデータ処理
```
using System;
using System.Collections.Generic;

class Program
{
    public static void Run()
    {
        Write(new[] { 1, 2, 3, 4, 5 });
        Write(new List<int> { 1, 2, 3, 4, 5 });
    }

    static void Write<T>(IEnumerable<T> data)
    {
        foreach (var x in data)
            Console.WriteLine(x);
    }
}
```

▼リスト2　配列を作ってデータ列を返すメソッド
```
public static IEnumerable<int> Repeat(int value, int count)
{
    var data = new int[count];
    for (int i = 0; i < count; i++)
    {
        data[i] = value;
    }
    return data;
}
```



不要な配列が作られることです。データ列を処理する側は、1度に1つのデータしか見ないことが多いです。1つずつ返せばいいものを、全データ分の長さの配列を作ってしまっています。データの個数が少ないことがわかっている間はこういう実装でもいいのですが、不特定多数だとそうはいきません。何百万／何千万という個数になった場合に、これでは使用メモリ的に無駄が大きすぎます。

このような無駄をなくすために、**リスト3**に示すような型を作るというパターンが知られています(Repeatメソッドはこの型を作って返すだけにします)。

これで、無駄な配列は作られず、ちゃんと1つ1つデータが得られます。その一方で、IEnumerable<T>インターフェースなどについての知識が求められ少し難しくなりますし、余計なクラスを1つ作ることになるので面倒も増えます。また、配列を作る場合と比べてかなり煩雑なコードになりました。

ちなみに、一応、**リスト3**のような型は、**リスト2**のコードから機械的なルールで作れます。機械的に作れるのなら、コンパイラに自動生成させればいいというのが、C#のイテレータの基本的なアイデアです。

▼リスト3　データ列生成でよくあるパターン

```
struct Repeater : IEnumerable<int>, IEnumerator<int>
{
    public int Current { get; }
    readonly int _count;
    int _i;

    public Repeater(int value, int count)
    {
        Current = value;
        _count = count;
        _i = 0;
    }

    public bool MoveNext()
    {
        _i++;
        return _i <= _count;
    }
    // 後略
}

public static IEnumerable<int> Repeat(int value, int count)
{
    return new Repeater(value, count);
}
```

▼リスト4　イテレータを使ってデータ列を返すメソッド

```
public static IEnumerable<int> Repeat(int value, int count)
{
    for (int i = 0; i < count; i++)
    {
        yield return value;
    }
}
```

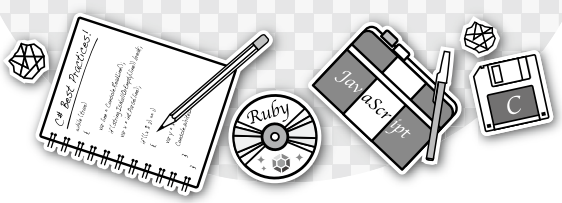
イテレータを使った実装

データ列生成の問題を解消するのが、C# 2.0で追加されたイテレータ(iterator)という機能です。イテレータは、**リスト2**に近い書き方から、**リスト3**に示すような型をコンパイラが生成して、無駄なくデータ列を生成できるようにします。**リスト4**に示すように、1つ1つのデータを返したい場所で**yield return**というものを書きます。

yieldという単語は「譲り渡す」という意味です。**yield return**の行に来るたびに一度このメソッドを抜けて、呼び出し側に移ります(実行権を譲り渡している)。そして、「次の1データがほしい」となった瞬間(**リスト3**で言うとMoveNextメソッドが呼ばれた瞬間)に、処理を再開して次の1データを作って返します。

データ処理の汎用化

イテレータから引き続きデータ処理の話をしましょう。データ処理と言うと、C#にはLINQ(リンク)(Language Integrated Query)と呼ばれる機能があります。LINQは、正確に言うとデータ処理に関連する複



数の構文やライブラリの組み合わせを指す言葉です。

「LINQとは何か」については割愛しますが、重要なのは、「組み合わせ」という部分です。小さな機能を組み合わせて大きな目的を実現したり、汎用的な処理を組み合わせて複雑な処理を作ったり、それぞれ別の担当者が書いた小さな部品を組み合わせてシステム全体を構築したり、さまざまな組み合わせが考えられます。ここでは、C#でデータ処理を行ううえで、「組み合わせ」がどう活きているかという話をしていきましょう。

入力、加工、出力

1つめは、データ列の入力元と出力先の組み合わせです。少し恣意的な例になりますが、「入力した整数列のうち、奇数のものだけを抜き出して、2乗したものを出力する」という処理を考えましょう。入力元／出力先が固定でいいならそう難しい話ではありません。たとえば、コンソールからの入出力で考えると、**リスト5**に示すようになります。

問題は、入力元／出力先はコンソールとは限らないことです。ファイルの読み書きであったり、ネット越しの受け渡しであったり、さまざまな入出力が考えられます。そのたびに、**リスト5**に類するコードを書くのは非効率で、「奇数のものだけ抜き出して、2乗」という加工する部分だけを切り出して、さまざまな入出力と組み合わせて使えるようにすべきです。

これは、IEnumerable<T>を受け取り、IEnumerable<T>を返すメソッドを作れば実現できます。前節で説明したイテレータを使えばそう難しくはありません。**リスト6**に示すような書き方ができます。

これで、**図2**に示すように、さまざまな入出力の組み合わせが使えるようになります。

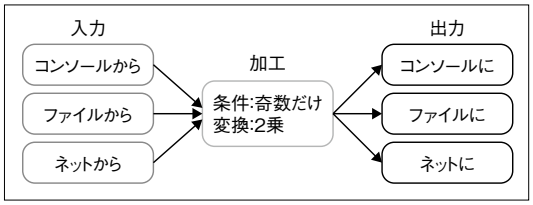
▼リスト5　入力から出力までを1つのメソッドで実装する例

```
while (true)
{
    var line = Console.ReadLine();
    if (string.IsNullOrEmpty(line)) break;
    var x = int.Parse(line);

    if ((x % 2) == 1)
    {
        var y = x * x;
        Console.WriteLine(y);
    }
}
```

汎用処理の組み合わせ

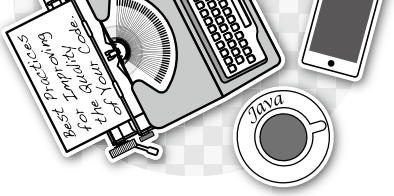
続いては小さな汎用処理の組み合わせで所望の処理を実現することについて考えます。前項の加工処理(**リスト6**のFilterメソッド)をさらに細かく分けると、そこには次の処理が含まれています。

▼図2　入力、加工、出力の組み合わせ



▼リスト6　入力(Read)、加工(Filter)、出力(Write)の分離

```
static IEnumerable<int> Read()
{
    while (true)
    {
        var line = Console.ReadLine();
        if (string.IsNullOrEmpty(line)) break;
        yield return int.Parse(line);
    }
}

static IEnumerable<int> Filter(IEnumerable<int> source)
{
    foreach (var x in source)
        if ((x % 2) == 1)
            yield return x * x;
}

static void Write(IEnumerable<int> source)
{
    foreach (var x in source)
        Console.WriteLine(x);
}
```



・条件選択：奇数だけ取り出す
・変換：2乗を計算する

そして、一般に、多くのデータ処理がこの類型に当てはまります。すなわち、何らかの条件を与えて選択を行い、何らかの式に従って変換を行います。

実は、.NETには標準で、条件選択や変換のためのライブラリが含まれています。WhereメソッドとSelectメソッド(いずれもSystem.Linq名前空間のEnumerableクラスの静的メソッド[注2])です。

・Where：条件を与えてデータを選択する
・Select：式を与えてデータを変換する

これらの名前は、SQLのキーワードに由来します。このほかにも、Enumerableクラスには、データ加工用のさまざまなメソッドが用意されています。

これらを使って**リスト6**と同じ処理を書き直すと、(**リスト6**のRead、Writeに対して)**リスト7**のような書き方ができます。

ちなみに、Where、Selectは、インスタンスメソッドと同じように**x.Where(...)**というような書き方をしていますが、実際に呼ばれるのはEnumerableクラスのWhere静的メソッドです。これは、拡張メソッドと呼ばれる機能を使っています。

これで、**図3**に示すように、汎用処理の組み合わせで所望の処理を実現できます。

注2) クラスに属するメソッドのこと。ちなみに、インスタンスに属するメソッドはインスタンスメソッドと言う。

▼リスト7　標準の汎用的なメソッドでデータ列を加工する例

```
Write(Read()
    .Where(x => (x % 2) == 1)
    .Select(x => x * x)
    );
```

契約、実装、処理

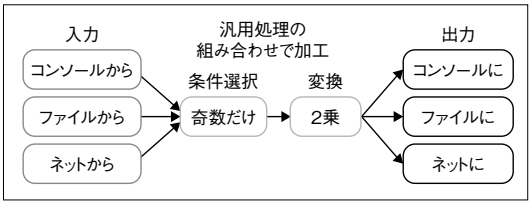
前項で説明したようなIEnumerable<T>を中心とした汎用処理には、**図4**に示すような3つの立場が絡みます。

規約(contract)は、型が持つべきメンバが何かを定めます。IEnumerable<T>の例で言うと、「データ列を得るためにはCurrentプロパティやMoveNextメソッドが必要」というようなものです。これを定めるのがインターフェースです。

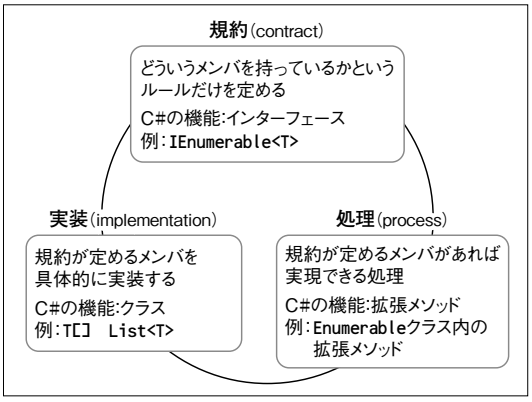
実装(implementation)は、規約が定めるメンバをどう実現するかです。同様の例で言うと、「配列やリストなどのクラスは、IEnumerable<T>を実装しているので、データ列を列挙できる。列挙のしかたはそれぞれのクラスによって異なる」となります。

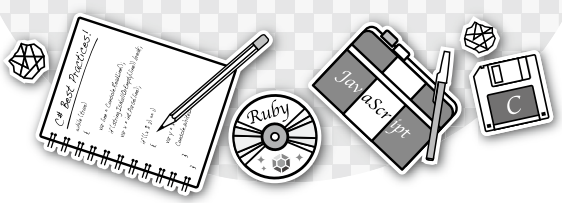
そして最後に、この規約に沿えば実現できる処理(process)があります。今回の例で言うと、「WhereやSelectなど、IEnumerable<T>から

▼図3　汎用処理の組み合わせ



▼図4　規約、実装、処理





得られるデータ列を加工して、別のIEnumerable<T>を返すメソッドを作る」といったものです。

重要なのは、規約、実装、処理の3つは、それぞれ別の担当者が書く(ということがあるし、そうできるべき)ということです。これに対してありがちなミスは、実装クラス(ここで言う配列やリスト)に処理(ここで言うWhereやSelect)を含めてしまうというものです。そうやってしまうと、どんな実装にでも使えそうな汎用的な処理が特定の実装にだけ含まれることになって、組み合わせて使えなくなります。組み合わせを増やすために、規約、実装、処理の分離を意識しましょう。

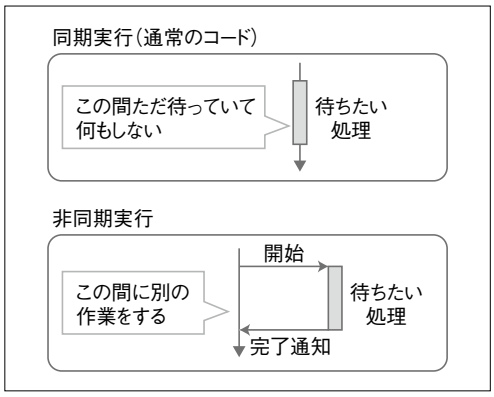
文法の組み合わせ

この節の冒頭で「LINQとはデータ処理に関連する複数の構文の組み合わせ」という話をしました。データ処理はプログラミングにおいて重要なテーマの1つですが、それでも、汎用プログラミング言語へデータ処理専用の構文を導入するのはやり過ぎでしょう(「汎用」でなくなる)。しかし、それぞれ汎用に使える小さな構文の組み合わせで実現できるなら話は別で、汎用プログラミング言語に導入する価値が高くなります。

誌面の都合で詳細は割愛しますが、LINQは次のような構文の組み合わせで実現されています。これらはすべて、C# 3.0で追加され、データ処理以外のことに対しても有用です。

- オブジェクト初期化子
- ラムダ式
- 拡張メソッド
- 変数の型推論(var)
- 匿名型

▼図5　同期処理と非同期処理の違い



非同期メソッド

ネットワーク越しにデータをダウンロードする場合など、完了までに時間がかかるのを待っていないといけない処理があります。この手の処理は、非同期に実行すべきとされています。非同期(asynchronous)というのは、**図5**に示すように、時間がかかる処理を待たず、ほかの作業ができる状態を保つことを言います。

たとえば、Web上のコンテンツをダウンロードする場合を考えましょう。C#ではWebアクセスにHttpClientクラス(System.Net.Http名前空間)を使います。同期実行であれば、**リスト8**に示すようなコードになります[注3]。

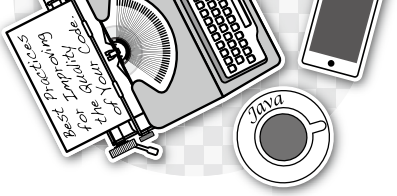
この書き方では、ダウンロードが終わるまで(通信状況やデータ量によっては数秒から数分かかる場合もある)の間、ほかの処理が何もできません。この「ほかの処理」には、たとえばユーザのマウスやキーボード、タッチ操作に対する応答も含まれていて、同期実行してしまうとアプリケーションがフリーズします。

そこで非同期実行が必要になるわけですが、

注3) 実際には、HttpClientクラスに同期実行できるメソッドはなく、このコードはコンパイルできません。本節で説明するとおりWebアクセスの同期実行は非推奨なのですが、推奨できないものは最初からAPI提供しないという方針です。

▼リスト8　同期でWebダウンロード

```
var c = new HttpClient();
var res = c.Get("http://ufcpp.net");
var content = res.Content.ReadAsString();
Console.WriteLine(content);
```



非同期実行できるコードを書くのは骨が折れる作業でした。たとえば、C# 4.0以前の書き方で**リスト8**の処理を非同期化するには**リスト9**のようなコードが必要でした。

やりたいこと自体は**リスト8**と同じなのに、これだけ面倒になります。これでもまだましなほうで、条件分岐や繰り返し、エラー処理などが絡むとさらに大幅に複雑化します。

そこで、C# 5.0で、非同期メソッド（asynchronous method）という機能が追加されました。**リスト10**のようなコードで非同期実行できる機能です。

awaitと書いたところを、コンパイラが**リスト9**のようなコードに展開してくれます（正確に言うと、イテレータと同じようなクラスの生成が行われます）。

これで、同期実行とほとんど同じ書き方で非同期実行できるようになります。一般に、非同期実行にはここで説明したようなもの以外にもさまざまなタイプのものがあり、C#の非同期メソッドでそのすべてがカバーできるわけではありませんが、多くのタイプの処理が大幅に簡素化できます。

nullの撤廃

プログラミング言語におけるnull（無効な参照）の存在はbillion-dollar mistake（10億ドル規模で損失を生んでいる失敗）と呼ばれることがあります。無効であることを表すものがほしいことも時にはありますが、無効なものがあり得ない場合のほうが多いですし、少なくとも次のようにすべきだったと言われています。

・既定動作としては「無効」を認めない
・「無効」があり得るかあり得ないかを一目で区別できるようにする

C#の場合、値型（整数や論理値、構造体など）と参照型（文字列やクラスなど）があり、値型についてはこの条件を満たしています。すなわち、値型にはそのままではnullを代入できず、別途null許容型（nullable type）というものを作って初めてnullを使えます。また、通常の値型とnull許容型は明確に区別されるため、型を見ただけでnullがあり得るかどうかがわかります。**リスト11**に例を示します。

型名の後ろに?がついているものがnull許容型で、この場合だけnullを代入できます。null許容型から普通の値型への変換は暗黙的には行えません。

▼リスト11　値型とnull許容型

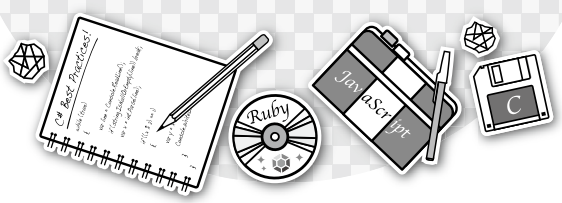
```
int a = 10; // OK
int b = null; // コンパイルエラー

// 型名の後ろに?を付けるとnull許容型になる
int? c = 10; // OK
int? d = null; // これもOK

c = a; // int→int?の変換は常に成功
a = c; // 逆はコンパイルエラー

a = c ?? 0; // nullをなくす処理が必要
```

▼リスト9　C# 4.0以前の非同期実行の書き方

```
var c = new HttpClient();
c.GetAsync("http://ufcpp.net")
    .ContinueWith(res => res.Result.Content.ReadAsStringAsync())
    .ContinueWith(content => Console.WriteLine(content.Result));
```

▼リスト10　C# 5.0以降の非同期実行の書き方

```
var c = new HttpClient();
var res = await c.GetAsync("http://ufcpp.net");
var content = await res.Content.ReadAsStringAsync();
Console.WriteLine(content);
```



nullのない参照型(C# 7)

問題は参照型についてです。C# 6までの範囲では、参照型は常にnullを認めます。値型のように、nullの有無を区別する方法がありません。

そこで、C# 7に向けて、参照型の仕様変更が検討されています。参照型も、既定ではnullを認めず、型名に?を付けたときだけnull許容にしようという話です。

ただし、C#としては過去のバージョンとの互換性を非常に重要視しています。機能追加に際しては、過去に書いたコードを破壊しないように細心の注意を払っています。単純に参照型の仕様変更をしてしまうと、この方針に背くことになるので、opt-in(オプションを変えるなど、明示的に選択しない限り有効にならない)機能として提供されることになりそうです。

nullへの対処

nullに対する対処方法には3パターンあります。

①nullだったらエラーにする
②nullだったら代わりに別の有効な値を返す処理を挟む
③nullを伝搬させる

①は、できればコンパイルエラーになるべきものでしょう。たとえば次の例を見てください。

例10 nullのチェックのタイミング 注4

```
static void Main() => X(null);

static int X(string s) =>
s.Length;
```

このコードは、C# 6以前であれば、コンパイルできてしまい、実行すると例外を発生させます。このような挙動は、ほとんどの場合バグで、バグを見つけにくいという意味で好ましくありません(これがbillion-dollar mistakeと呼ばれる所以(ゆえん)です)。C# 7で提案されているのは、これをそもそもコンパイルエラーとして検出できるようにしようというものです。

②は、nullを認める文脈から、認めない文脈への変換ということになります。「無効な値が来た場合には、代わりにこの値を使う」というような値の指定を行います。C#には、例11に示すように、??演算子(null合体演算子と呼びます)というものがあり、これを使って「代わりの値」を与えられます(C# 2.0からの機能)。

例11 null合体演算子

```
int? n = null;
int x = n ?? 0; // nullの代わりに0を使う
```

③は、nullを許容する文脈内部において、「入力が無効だったら出力も無効にする」というものです。C#には、例12に示すように、?.演算子(null条件演算子と呼びます)というものがあり、入力がnullだったら出力もnullを返すことができます(C# 6からの機能)。

例12 null条件演算子

```
int? X(string s) => s?.Length;
```

◆ ◆ ◆

ここまで、筆者が考えるC#における「良いコードの書き方」を紹介してきました。C#の構文には、「良いコード」に関するノウハウが詰まっています。また、C#のバージョンアップとともにより良いコードが書けるようになっています。本稿や、その他のC#入門文書などを、今後、プログラミングする際に、参考にしていただければ、幸いです。SD

注4) `void Main() => X(null);`は、C# 6で追加されたメソッド定義の構文です。returnステートメント1つだけのメソッドなら、=>を使ってreturnキーワードや{}を省略することができます。
`void Main() { return X(null); }` と `void Main() => X(null);` は同じ意味です。

第4章　Ruby編

Best Practices for Improving the Quality of Your Code.

お作法を意識して可読性や保守性を高めよう

Author 伊藤 淳一（いとう じゅんいち）　㈱ソニックガーデン　Twitter @jnchito

Rubyはコードの書き方にはかなり寛大な言語です。「生産的」でかつ「楽しく」プログラミングできることが第一の目的とされていることからも、その懐の深さがうかがえます。その一方で、他人にも読みやすいコードを書くには、いくつかのポイントを押さえておくことが重要になってきます。

はじめに

Rubyはまつもとゆきひろ氏（通称Matz）によって開発されたオブジェクト指向スクリプト言語です。Rubyは表現力豊かな文法と強力な標準ライブラリを備えています。2004年にはRubyを使ったWebアプリケーションフレームワークであるRuby on Rails[注1]（以下、Rails）が登場し、世界的にRubyが使われるようになりました。現在でもRailsはメジャーなWebアプリケーションフレームワークの1つです。きっとみなさんの中にも「Railsなら使ったことがある」という人はたくさんいると思います。

ところで、「単にRubyを動かすだけ」であればそれほど難しくはありません。むしろRubyは1つのことを実現するのに多彩な書き方を許容しているため、「こんな書き方」でも「あんな書き方」でも動いてしまいます。とはいえ、Rubyプログラマの中ではある程度の「お作法」が存在しています。プログラミング初心者の人や、ほかの言語を使っていた期間が長い人はつい「お作法」から外れたコードを書いてしまいがちです。

そこで本章ではRuby初心者によくありがちな「お作法から外れた書き方」と、それに対応する「お作法に則した書き方」を紹介していきます。また、記事の後半ではお作法に沿ったRubyプログラムを書くための考え方や学習方法を紹介します。お作法の中には「なんか気持ち悪い」と思うものもあるかもしれませんが、「郷に入れば郷に従え」の精神で、まずはRubyらしい書き方をマスターしていってください。

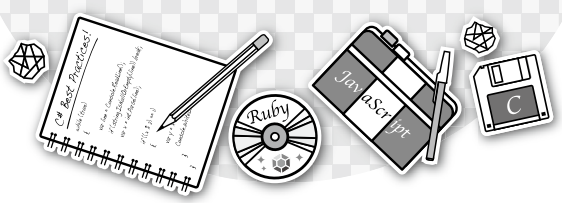
ソースコードのインデントは半角スペース2つ

Rubyのソースコードのインデントは半角スペース2つが一般的です。半角スペース4つにしたり、タブ文字を使ったりするのはNGです。

半角スペース2つでインデントさせる
```
class Person
  def hello
    puts 'hello'
  end
end
```

半角スペース4つや、タブ文字は使わない
```
class Person
    def hello
        puts 'hello'
    end
end
```

注1）http://rubyonrails.org/



メソッド名や変数名はスネークケース

Rubyのメソッド名や引数名、変数名は通常スネークケース(アンダーバー区切り)で書きます。ほかの言語からやってきた人の中には、キャメルケースで書いてしまう人をときどき見かけるので注意してください。

```
class UserProfile
  phoneNumberのような
  キャメルケースの引数名・変数名はNG
  def initialize(name, phone_number)
    @name = name
    @phone_number = phone_number
  end

  firstNameのような
  キャメルケースのメソッド名はNG
  def first_name
    @name.split(' ').first
  end
end
```

ちなみにRubyではインスタンス変数を@nameのように@で始めます。

引数がない場合はメソッド呼び出しに丸括弧を使わない

Rubyではメソッド呼び出し時の丸括弧が省略できます(ただし、文法上省略できないケースもあります)。とくに、引数なしでメソッドを呼び出す場合は、丸括弧を省略することが多いです。また、引数を伴うメソッド呼び出しでも丸括弧が省略されることがよくあります。

```
upcase()と書くことは少ない
'hello'.upcase  => 'HELLO'

puts('hello')でも良いが、
丸括弧は省略されることが多い
puts 'hello'
```

メソッドの戻り値にはreturnを付けない

メソッドから戻り値を返す場合、Rubyではreturnを付ける必要がありません。これは最後に評価された式がメソッドの戻り値になるためです。

たとえば、Fizz Buzz問題[注2]をRubyのメソッドとして実装するとこうなります。

```
def fizz_buzz(n)
  if n % 5 == 0 && n % 3 == 0
    'Fizz Buzz'
  elsif n % 3 == 0
    'Fizz'
  elsif n % 5 == 0
    'Buzz'
  else
    n
  end
end
```

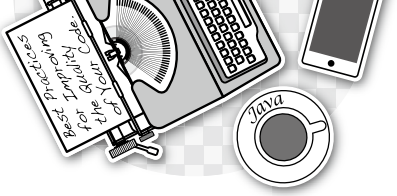
上の例のように、Rubyではreturnを使わずにメソッドの戻り値を書くスタイルが一般的です。ただし、メソッドの途中で処理を抜けたい場合はreturnを使います。次のコードはreturnを使ってメソッドを途中で抜けるコード例です。

```
def go_to_school
  today = Date.today
  if today.saturday? || today.sunday?
    土曜日と日曜日は何もせず
    メソッドを抜ける
    return
  end

  self.get_up
  self.eat_breakfast
  処理が続く...
end
```

if修飾子を使って行数を減らす

Rubyの条件分岐は実行されるコードの後ろに置いて1行で書くことができます(if修飾子)(**リスト1**)。ただし、これは「絶対にこうすべき」というお作法ではありません。コードの読みやすさを考慮して、適宜普通のifと使い分けてく

注2) https://ja.wikipedia.org/wiki/Fizz_Buzz



ください。

if＋否定形の条件はunlessに置き換える

Rubyの条件分岐にはifと逆の意味になるunlessが使えます(つまり、結果が偽となった場合に処理を実行します)(**リスト2**)。if＋否定形で書いた条件分岐はunlessに書き換えると意図がより明確になる場合があります。

真偽値を返すメソッドの名前は“?”で終わらせる

ここまでに挙げたサンプルコードの中にもときどき登場していますが、真偽値を返すメソッド名は慣習として**?**で終わらせます。こういうメソッドはほかの言語だと、is_adminやcan_editのようなメソッド名が付けられることが多いです。

```
is_adminよりもadmin?のように
?で終わらせるほうがベター
def admin?
  self.role == 'admin'
end
```

▼リスト1　if修飾子を使った条件分岐

```
普通のifを使った場合
if user.age < 20
  puts 'お酒は20歳になってから！'
end

if修飾子を使った場合
puts 'お酒は20歳になってから！' if user.age < 20
```

▼リスト2　unlessを使った条件分岐

```
userが管理者でなければ通知を送る
if !user.admin?
  send_notification_to(user)
end

上の条件分岐をunlessで置き換える
unless user.admin?
  send_notification_to(user)
end

ifと同様、後ろに置くこともできる
send_notification_to(user) unless user.admin?
```

“+”ではなく、式展開(#{ })を使って文字列を組み立てる

Rubyでは**+**演算子を使って文字列を連結させることもできますが、変数と組み合わせる場合は式展開(**#{ }**)を使うことが多いです。

```
def hello(name)
  "Hello, " + name + "!"ではなく、
  式展開を使う
  "Hello, #{name}!"
end
hello 'Alice'  => 'Hello, Alice!'
```

なお、文字列内で式展開を使う場合はシングルクオート(**'**)ではなく、ダブルクオート(**"**)を使う必要があります。

column　Rubyの真偽値を理解する

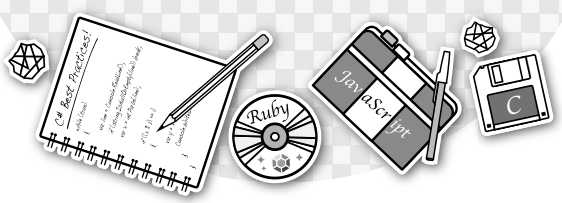
Rubyの真偽値は次のようなルールを持っています。

- falseとnilは偽
- それ以外の値はすべて真

nilはJavaやC#でいうところのnullです。真偽値の評価を具体的なコードで説明すると次のようになります。

```
if false
  puts '実行されません'
end

if nil
  puts '実行されません'
end

if true
  puts '実行されます'
end

if 0
  puts '実行されます'
end
```

ほかの言語の経験が長いと、つい使い慣れた言語のルールでコードを読み書きしがちなので、この点には注意してください。



ハッシュのキーには なるべくシンボルを使う

ハッシュはキーと値を関連づけてデータを格納するオブジェクトです。ほかの言語では連想配列やマップ、辞書(ディクショナリ)と呼ばれることもあります。

リスト3は国の名前と通貨を関連づけたハッシュの使用例です。

ところで、**リスト3**の例ではキーに文字列を使いましたが、特別な理由がなければ**リスト4**のように文字列ではなく、シンボルを使う方がベターです。

文字列のキーと比較した場合、シンボルには次のようなメリットがあります。

- `{ key: value }`のように簡潔なリテラルでハッシュを初期化できる
- 文字列よりも速い
- 文字列よりもメモリの使用効率が良い

繰り返し処理ではforやwhileではなく、eachを使う

ほかの言語だと繰り返し処理はforループやwhileループを使うのが基本、という場合もありますが、Rubyではeachや後述するmapといった、繰り返し処理用のメソッドを使うことがほとんどです。forループやwhileループ使うロジックを書いてしまったときは、eachやmapで書き換えられないか確認してみましょう。

次のコードは配列の中身を順番にコンソールへ出力するコード例です。eachメソッドを使った場合と、forループを使った場合の2パターンを記述しています。

```
fruits = ['apple', 'melon', 'banana']

繰り返し処理は
eachメソッドを使うのが一般的
fruits.each do |fruit|
  puts fruit
end

単純な繰り返し処理で
forループが登場することはまずない
for fruit in fruits
  puts fruit
end
```

Column そもそもシンボルって何?

シンボルは**:japan**や**:red**のように、「コロン+文字列」のリテラルで生成されるオブジェクトです。一見するとシンボルは普通の文字列によく似ています。ですが、内部的には1つの名前に対して1つのユニークな整数が割り当てられており、文字列よりも高速に値の比較ができます。

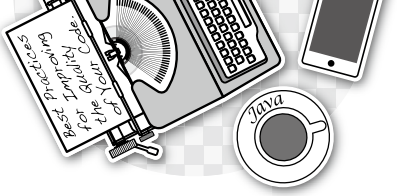
こうした特徴から、シンボルはハッシュのキーのように「ソースコード上では名前を識別できるようにしたいが、その名前が必ずしも文字列である必要はない場合」によく利用されます。

▼リスト3　ハッシュのキーに文字列を使った例

```
currencies = { 'japan' => 'yen', 'america' => 'dollar', 'italy' => 'euro' }
currencies['india'] = 'rupee'
puts currencies['japan']  => 'yen'
```

▼リスト4　ハッシュのキーにシンボルを使った例

```
ハッシュのキーにシンボルを使う
currencies = { japan: 'yen', america: 'dollar', italy: 'euro' }
currencies[:india] = 'rupee'
puts currencies[:japan]  => 'yen'
```



column　Rubyのブロックを理解する

eachメソッドのサンプルコードで、**do ... end**で囲まれている部分をブロックといいます。ブロックは簡単に言うと「メソッドの内部から呼び出せる、手続きのかたまり」です。メソッドの中にはブロックを引数として受け取り、メソッドの実行中にブロックを呼び出すものがあります。

eachもブロックを受け取るメソッドの1つです。eachメソッドの仕事は配列やハッシュの各要素を最初から最後まで順番に取り出すことです。取り出した要素をどう扱うかは、eachメソッドを呼びだした側の仕事になるので、ブロック内で必要な手続きを記述します。

次のコードはeachメソッドの利用例です。この場合は「配列から順に取り出した要素(fruit)をコンソールに出力(puts)すること」をブロックで記述したことになります。

```
fruits.each do |fruit|
  puts fruit
end
```

ほかの言語で類似した機能を使ったことがない人はピンとこないかもしれませんが、ブロックは非常によく使われる機能の1つですので、早く慣れるようにしましょう。

配列の全要素を加工する場合はmapを使う

配列の全要素を加工して別の新しい配列を作る場合はmapを使うと簡潔なコードが書けます。例として、数値で構成された配列から16進数で表現した文字列の配列へ変換するプログラムを考えてみましょう。**リスト5**はmapを使わない場合です。

mapを使うと**リスト5**を**リスト6**のように書き直せます(ブロック内で返却された値を順に集めた、新しい配列が作成されます)。

▼リスト5　mapを使わない場合

```
numbers = [8, 9, 10, 11, 12]
hex_numbers = []
numbers.each do |n|
  hex_numbers << n.to_s(16)
end
puts hex_numbers  => ['8', '9', 'a', 'b', 'c']
```

▼リスト6　mapを使った場合

```
numbers = [8, 9, 10, 11, 12]
hex_numbers = numbers.map do |n|
  n.to_s(16)
end
puts hex_numbers  => ['8', '9', 'a', 'b', 'c']
```

空の配列を準備し、eachブロック内でその配列に値を詰め込んでいくタイプの処理はほとんどmapで書き換えることが可能です。mapで置き換え可能な繰り返し処理はとても多いので、mapを使いこなせていないと思う人は置き換え可能な処理がないかどうか、注意深く自分のコードを見直してみましょう。

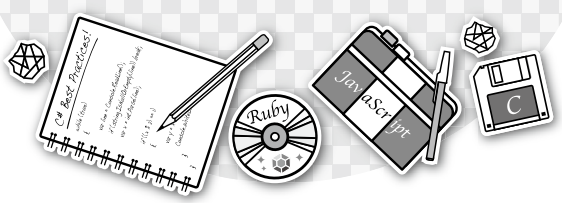
配列から条件に合う要素だけを抜き出すときはselectを使う

配列から条件に合う要素を抜き出す場合はselectを使うと簡潔なコードが書けます。例として、数値が入った配列から奇数だけを抜き出すプログラムを考えてみましょう。**リスト7**はselectを使わない場合です。

selectを使うと**リスト7**を**リスト8**のように書き直せます(ブロック内で返却された値が真になる要素を順に集めた、新しい配列が作成されます)。

配列から条件に合致する最初の1件を取得するときはfindを使う

配列から条件に合致する最初の1件だけを取得する場合はfindを使うと簡潔にコードが書け



ます。例として、ランダムに数字が並んだ配列から100より大きい数を1件だけ抜き出すプログラムを考えてみましょう。**リスト9**はfindを使わない場合です。

findを使うと**リスト9**を**リスト10**のように書き直せます(ブロック内の戻り値が最初に真になった要素を返します)。

なお、**リスト10**では**do ... end**の代わりに中括弧を使ってブロックを書きました。中括弧を使ったブロックの作成はコラム「Rubyの言語機能を活用してコードをもっと短く!」を参照してください。

▼リスト7　selectを使わない場合

```
numbers = [1, 2, 3, 4, 5]
odd_numbers = []
numbers.each do |n|
  if n.odd?
    odd_numbers << n
  end
end
puts odd_numbers  => [1, 3, 5]
```

▼リスト8　selectを使った場合

```
numbers = [1, 2, 3, 4, 5]
odd_numbers = numbers.select do |n|
  n.odd?
end
puts odd_numbers  => [1, 3, 5]
```

▼リスト9　findを使わない場合

```
numbers = [98, 90, 109, 94, 102]
target = nil
numbers.each do |n|
  if n > 100
    target = n
    break
  end
end
puts target  => 109
```

▼リスト10　findを使った場合

```
numbers = [98, 90, 109, 94, 102]
target = numbers.find { |n| n > 100 }
puts target  => 109
```

Rubyの言語機能を活用してコードをもっと短く!

selectの説明で使った**リスト8**のコードはRubyの言語機能を活用することで、もっと短くすることができます。Rubyに慣れてきたら次のようなリファクタリングにもトライしてみましょう。

まず、**do ... end**のブロックは改行せずに1行で書くことができます(**リストA**)。ただし、このままだと読みにくいので、**do ... end**を中括弧(**{ }**)に書き換えます(**リストB**)。

▼リストA　リスト8のdo ... endを1行で書く

```
odd_numbers = numbers.select do |n| n.odd? end
```

▼リストB　リストAのdo ... endを中括弧で置き換え

```
odd_numbers = numbers.select { |n| n.odd? }
```

▼リストC　リストBの引数nを&:を使って書き換え

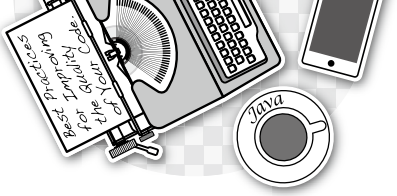
```
odd_numbers = numbers.select(&:odd?)
```

さらに、ブロックの引数として渡されるオブジェクト(ここでは**n**)に対して、引数なしのメソッドを呼び出す場合は、**リストC**のように"**&:メソッド名**"と書くことができます(アンパサンドの意味を説明すると長くなるのでここでは省略します)。

リファクタリングはこれで完了です。selectを使わない元のコードと比較すると、**リストD**のように短くなりました!

▼リストD　リファクタリング結果の比較

```
selectを使わない場合
odd_numbers = []
numbers.each do |n|
  if n.odd?
    odd_numbers << n
  end
end

selectを使う場合(リファクタリング後)
odd_numbers = numbers.select(&:odd?)
```



“&.”演算子でnilでもメソッドを安全に呼び出す（Ruby 2.3以降）

2015年12月にリリースされたRuby 2.3[注3]では、さまざまな便利機能が追加されました。本稿ではその中から利用頻度が高そうな**&.**演算子（safe navigation operator）を紹介します。

Ruby 2.2以前では、オブジェクトがnilの可能性がある場合、そのままメソッドを呼び出すとNoMethodErrorが発生するため、次のように条件分岐させる必要がありました。

```
userはnilの可能性があるので
ガード条件を付ける
unless user.nil?
  user.say 'Hello!'
end
```

Ruby 2.3から導入された**&.**演算子を使うと、オブジェクトがnilでもNoMethodErrorが発生しなくなります（nil以外のオブジェクトであれば普通にメソッドが実行され、nilに対してメソッドを呼びだした場合はnilが返ります）。

```
userがnilでも気にせずに
sayメソッドを呼び出せる
user&.say 'Hello!'
```

さて、「こういう場合はこう書いた方が良い」という事例を個別に挙げていくと、誌面がいくらあっても足りません。個別の事例を挙げるのはここまでにして、ここからあとはもう少し抽象度を上げ、良いRubyプログラマ、いや、「良いRubyist」（Rubyが好きなプログラマのことをRubyistと呼びます）になる方法を考えてみましょう。

車輪の再発明をせず、既製品（標準ライブラリやgem）を使う

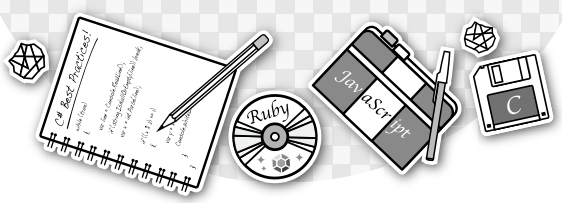
プログラミングの世界では、しばしば「車輪の再発明」という慣用句が登場します。これは「すでに用意されている解決策を使わずに自分でイチからプログラムを書いてしまうこと」を意味し、通常はバッドプラクティスと見なされます。もちろんRubyの世界でも車輪の再発明をやってしまうのは好ましくありません！

Rubyの標準ライブラリには非常に数多くの便利メソッドが用意されています。前述のmapやselectのような基本的なメソッドから、「こんな処理まで標準ライブラリでやってくれるのか！」と驚くようなメソッドまで非常に幅広く用意されています。標準ライブラリを活用すれば、車輪の再発明をせずに簡潔で安全なコードを書くことができます。

とはいえ、標準ライブラリの内容を隅から隅まで頭に入れるのはちょっと無理です。なのでRuby初心者の人はまず、次に挙げる利用頻度の高いクラスやモジュール（モジュールについてはコラム「モジュールって何？」を参照）のAPIドキュメントをひととおりチェックしておきましょう。

・Stringクラス
http://docs.ruby-lang.org/ja/2.3.0/class/String.html
・Arrayクラス
http://docs.ruby-lang.org/ja/2.3.0/class/Array.html
・Hashクラス
http://docs.ruby-lang.org/ja/2.3.0/class/Hash.html
・Enumerableモジュール
http://docs.ruby-lang.org/ja/2.3.0/class/Enumerable.html

注3）https://www.ruby-lang.org/ja/news/2015/12/25/ruby-2-3-0-released/



また、Rubyにはオープンクラスと呼ばれる機能があり、既存のクラスを自由に拡張できるようになっています。Railsではこの特性を積極的に利用し、blank?やpresent?、truncate、end_of_dayといった、さまざまな便利メソッドをRubyの標準クラスに追加しています。量が多いのでこちらもやはり隅から隅まで頭に入れるのは無理だと思いますが、

モジュールって何?

モジュールはクラスの継承を使わずに共通のメソッドを再利用できる、ミックスインという機能を実現するために使われます。たとえば**リストE**のように、Rubyでは配列でもハッシュでも、どちらもfindメソッドが使えるようになっています。

これはArrayクラスやHashクラスがEnumerableモジュールを「ミックスイン」している(厳密にいうとincludeしている)ためです。findメソッドはArrayクラスやHashクラスではなく、Enumerableモジュール内で実装されています。

言葉だけではピンとこないと思うので、コードを使ってモジュールの定義とミックスインの使い方(ただしイメージレベル)を**リストF**に示します。

ほかにもモジュールは名前空間として使われる場合があります。ただし、モジュールを完全に理解しようと思うと誌面が足りなくなるので、今回は割愛します。

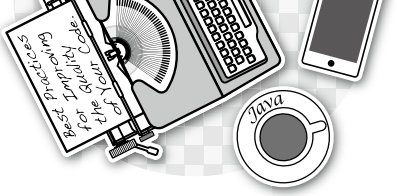
とりあえずここでは、「配列やハッシュで使える便利メソッドを探すときは、Enumerableモジュールも一緒に調べたほうが良い」ということを覚えておいてください。配列やハッシュにはfindメソッド以外にもたくさんの便利メソッドがEnumerableモジュールからミックスインされているからです。

▼リストE　配列とハッシュで使えるfind

```
配列でfindを使う
numbers = [11, 12, 13, 14, 15]
target = numbers.find { |n| n % 3 == 0 }
puts target  => 12

ハッシュでfindを使う
currencies = { japan: 'yen', america: 'dollar', italy: 'euro' }
target = currencies.find { |key, value| value == 'dollar' }
puts target  => [:america, 'dollar']
```

▼リストF　モジュールの定義とミックスインの使い方(イメージ)

```
注:これは説明用に極端に単純化したコードです。
実際のRubyの実装とは異なります。
module Enumerable
  def find
    findの実装コード
  end
end

class Array
  ミックスインにより配列で
  findメソッドが使えるようになる
  include Enumerable
end

class Hash
  ミックスインによりハッシュで
  findメソッドが使えるようになる
  include Enumerable
end
```



Railsを使うなら最低限、Railsガイドの「Active Support コア拡張機能[注4]」に目を通しておくことをオススメします。

さらにいうと、Rubyの世界には標準ライブラリやRailsだけではなく、gemと呼ばれるサードパーティライブラリが無数にあります（厳密にいえばRailsもgemの1つです）。わざわざ自分で実装しなくてもgemを使えば一発で実現できた！ということも多いです。

何か自分でロジックを書こうとしたときは、まず「これをやろうとしているのは世界で自分一人だけか？」と自問してください。答えがNOであれば、ほかにも同じようなことを考えている人がたくさんいて、誰かがすでにgemを作っている可能性があります。

たとえば、Railsで"/users/123"のようなデータベースのIDを使ったURLではなく、"/users/matz"のようにユーザ名でURLを作りたいと考える人は世界であなた一人だけでしょうか？　そうじゃないですよね。だからそういうgemがすでにあります。こういうケースではfriendly_id[注5]というgemを使うと簡単に実現できます。

「完全独学」を避け、「師匠」を見つける

本稿では「Rubyのお作法に従おう」とか「自分でロジックを書かずにgemを探そう」といった内容を書いていますが、そもそもRuby初心者の場合、「お作法に沿っているのか沿っていないのか、自分で判断がつかない」「gemを探したけど、目的に合ったgemを見つけられない」といったように、「それ以前」の段階でつまづいてしまう場合があります。

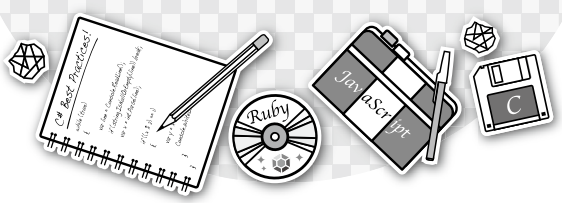
こういうときは気軽に相談したり、コードレビューしてもらったりできる「Rubyの師匠」が身近にいるのが理想的です。RubyやRailsで長年開発をしている職場であれば、おそらくその職場で師匠を見つけられると思います。日常的に相談できる師匠がいない場合でも近くのRuby勉強会に参加したりすれば、師匠的なRubyistと知り合いになれる可能性があります。それすらかなわないという場合は、スタック・オーバーフロー[注6]のようなQ＆Aサイトを利用して、適切なコードの書き方を質問するのも1つの手です。

また、良いコードかどうかはツールを使ってある程度機械的に判定することもできます。Rubyプログラムの静的解析ツールとしては、RuboCop[注7]やrails_best_practices[注8]といったgemがあります。ほかにも、脆弱性チェックをしたい場合はBrakeman[注9]というgemがあります。ただ、ツールを使うと大量の警告が出て圧倒されてしまったり、メッセージが英語で修正方法がぱっと理解できなかったりしがちなので、こういうときもできれば「相談できる師匠」が近くにいるのが理想的です。

おわりに

Rubyはプログラマが「生産的」でかつ、「楽しく」プログラミングできることを第一の目的にしたプログラミング言語です（英語版Wikipedia[注10]を参照）。筆者もかれこれ4～5年Rubyを使っていますが、Rubyのことを知れば知るほどまったくそのとおりだなと感じます。筆者はもともとJavaやC#をメインで使っていたので、丸括弧なしでメソッド呼び出しができる点や、falseだけでなくnilも偽として扱う点などは「なんか気持ち悪いな～」と感じていました。しかし、Rubyを使っているうちに「いや、こっちのほうが便利だし、読み書きしやすいじゃな

注4）http://railsguides.jp/active_support_core_extensions.html
注5）https://github.com/norman/friendly_id
注6）http://ja.stackoverflow.com/
注7）https://github.com/bbatsov/rubocop
注8）https://github.com/railsbp/rails_best_practices
注9）https://github.com/presidentbeef/brakeman
注10）https://en.wikipedia.org/wiki/Ruby_(programming_language)



いか！」と思うようになりました。標準ライブラリの機能も非常に豊富なので、「こんなことまでRuby単体でできるなんてすごい！」と感動するときがよくあります。

また、言語そのものとは直接関係ありませんが、大小のRubyコミュニティが各地に数多く存在し、活発に活動しているのも興味深い点です(筆者自身も兵庫県で西脇.rb[注11]という地域Rubyコミュニティを主催しています)。こうしたコミュニティの勉強会やイベントに参加すると、知り合いのRubyistがどんどん増えていきます。個人的にはこうした体験もRubyを「楽しい！」と思わせる要因の1つになっているんじゃないかと思います。

本稿は「お作法に沿ったRubyプログラムの書き方」というテーマであれこれ書いてきましたが、知識だけ増やして頭でっかちになるのは望ましくありません。それよりもどんどんRubyのコードを書き、どんどんコミュニティに参加して、「Rubyって楽しい！」という気持ちを実際に体験してもらうのが、Rubyのプログラミング力を向上させる一番の秘訣かもしれません。

みなさんもRubyでプログラミングをエンジョイしてください！SD

注11) https://nishiwaki-koberb.doorkeeper.jp/

column 良いRubyのコードを書くための情報源

「Rubyスタイルガイド」
https://github.com/bbatsov/ruby-style-guide

Rubyにはとくに公式のコーディング規約はありません。ネットを検索するといくつかのコーディング規約が見つかると思います。Rubyスタイルガイドは比較的有名なコーディング規約で、前述のRuboCopでも静的解析時のガイドラインとして利用されています(原文は英語ですが日本語訳[注12]もあります)。

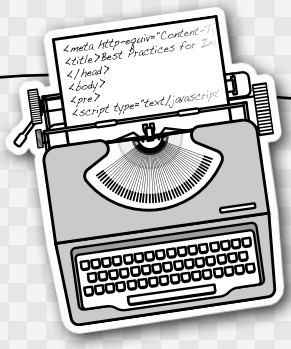
「プログラミング言語 Ruby」(オライリー・ジャパン刊)
https://www.oreilly.co.jp/books/9784873113944/

Rubyに関する技術書は数多く出版されています。筆者はすべての書籍をチェックしたわけではありませんが、個人的にオススメしたいのがこの「プログラミング言語 Ruby」です。Rubyの言語仕様がかなり細かいところまで網羅されているため、「この奇妙な記号の意味がわからない」とか「なぜか文法エラーが発生して動かない」といったときに、この本を開くと答えが載っていることが多いです。対象となるバージョンは1.9までなので、Ruby 2.0以降に登場したキーワード引数やprependなど、一部の言語仕様はカバーされていませんが、大半の内容は今でも有効です。

「[初心者向け]RubyやRailsでリファクタリングに使えそうなイディオムとか便利メソッドとか」
http://qiita.com/jnchito/items/dedb3b889ab226933ccf

筆者がQiitaに投稿した記事です。今回紹介しきれなかった「Rubyのお作法」や「初心者が見落としがちな便利メソッド」などを解説しています。

注12) https://github.com/fortissimo1997/ruby-style-guide/blob/japanese/README.ja.md



第5章　JavaScript+HTML+CSS編

Best Practices for Improving the Quality of Your Code.

再考! 今どきのWebアプリ開発のベストプラクティス

Author 青木 裕一（あおき ゆういち）　㈱ネオジャパン

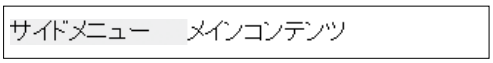
本章では、Webアプリケーション（以下、Webアプリ）を作るときの、HTML、CSS、JavaScriptの書き方を紹介します。Webページの見た目を整えるための書き方、不具合を生まないための書き方のほか、Webアプリ開発では必須の複数ブラウザ対応やセキュリティ対応の観点からも解説を行います。

HTML、CSS編

意味と見た目を使い分ける

Webページを作成するとき、個々の部分を意味で考えるか見た目で考えるかは重要です。たとえば、画面最上部をヘッダと考えるか、背景色付きのバーと考えるかです。基本的に意味を担当するのがHTML、見た目を担当するのがCSSです。しかし、これらの担当がちゃんと分離できていないWebページをたまに見かけます。ここでは、そういう例とその対処方法を説明します。

▼図1　positionの使用例（画面）



▼リスト1　positionの使用例（HTML）

```
div class="app-container"> ←親要素
  <div class="app-main">
    メインコンテンツ
  </div>
  <div class="app-menu"> ←子要素
    サイドメニュー
  </div>
</div>
```

tableレイアウトを使わない工夫

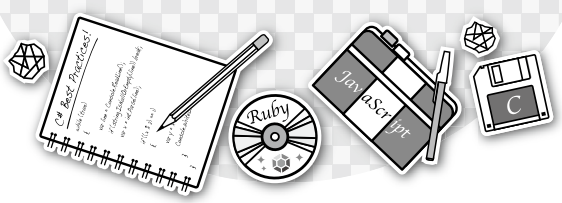
table要素を単なる配置のために使用してはいけないことは、昔から言われてきたことです。それでも、不勉強からか代替案の難しさからか、単なる配置のためにtable要素を使っているのを見かけることがあります。table要素を使わずに思ったとおりに配置するにはどうすればいいのでしょうか？　複数の要素を縦に並べて配置するのは、div要素などを並べて記述するだけでいいので簡単です[注1]。問題は横に並べる場合で、これはCSSをうまく使う必要があります。

最初に説明するのがposition: relativeの中にposition: absoluteを入れる方法です。本来、position: absoluteを指定した

注1）HTMLにて連続して書かれたdiv要素、h1～h6要素、ul要素、li要素など（HTML4.1時代にブロックレベル要素と呼ばれていたもの）は、CSSなどで指定しないかぎり、画面上は縦に並んで配置されるため。

▼リスト2　positionの使用例（CSS）

```
.app-container {
    position: relative; ←親要素にposition: relativeを指定
    padding-left: 120px;
}
.app-menu {
    position: absolute; ←子要素にposition: absoluteを指定（親要素にposition: relativeが指定されているため、以下のtop、leftは上位要素からの相対位置になる）
    top: 0;
    left: 0;
    bottom: 0;
    width: 120px;
    background-color: #EEEEEE;
}
```



要素は画面全体を基準に配置されますが、上位にposition: relativeを指定した要素を配置すると、その要素からの相対位置に配置できます。これを使えば、上位の要素(親要素)内に子要素を好きな位置に配置できます。**図1、リスト1、2**のようにメインコンテンツの横にサイドメニューを配置したい場合に有用です。この例ではメインコンテンツの左に幅120pxのサイドメニューを配置しています。

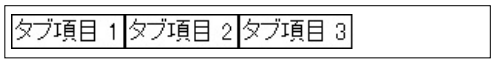
次に説明するのがdisplay: inline-blockを使用する方法です。display: inline-blockを使えば、サイズ指定した要素を横に並べることができます。**図2、リスト3、4**のようにタブメニューなどを横に並べたいときに有用です。この例では線で囲ったタブ項目を横に並べています。display: inline-blockを指定した要素と要素の間に空白文字があると、画面上でも少しの空白が開くので、コメントか何かで埋めましょう。

最後に説明するのがfloatを使用する方法です。要素にfloatを指定することで、親要素の左側か右側に子要素を配置することができます。**図3、リスト5、6**のように左に見出し、右にボタンを配置したいときに有用です。この例では変更ボタンをタイトルと同じ縦位置で右側に配置しています。横に並べようとしていた要素が上下に配置されたり(いわゆるカラム落ち)、floatを指定した要素が親要素からはみ出たり(**リスト6**のoverflow: hiddenはその対策)と、結構クセは強いです。

CSSのクラス名は意味で命名する

ところで、CSSのクラス名は意味と見た目、どちらで記述したほうが良いのでしょうか。どちらでもあまり変わらないように思うかもしれませんが、意味に対する見た目はあとで変更される可能性があります。このとき、クラス名を見た目で命名していると、設定先のHTML(Webアプリの場合はプログラム)を変更することになります。クラス名を意味で命名していると、CSSを変更することになります。両者を比較した場合、CSSの変更のほうがコストが小さい場合が多いので、クラス名は意味で命名したほうがいいでしょう。たとえば、未読の場合はboldではなくunread、重要な注釈の場合はredではなくimportantをクラス名に付けるようにしましょう。

▼図2　inline-blockの使用例(画面)



▼リスト3　inline-blockの使用例(HTML)

```
<ul class="app-items"><!--
--><li>タブ項目 1</li><!--
--><li>タブ項目 2</li><!--
--><li>タブ項目 3</li><!--
--></ul>
```

要素間の空白をコメントで埋める

▼リスト4　inline-blockの使用例(CSS)

```
ul.app-items {
    margin: 0;
    padding: 0;
}
ul.app-items > li {
    display: inline-block;
    padding: 2px;
    border: 1px solid black;
}
```

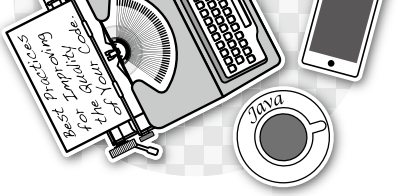
▼図3　floatの使用例(画面)

▼リスト5　floatの使用例(HTML)

```
<div class="app-itembar">
  <div class="app-button"><button>変更</button></div>
  <h4 class="app-title">タイトル</h4>
</div>
```

▼リスト6　floatの使用例(CSS)

```
.app-itembar {
    background-color: #EEEEEE;
    overflow: hidden;
}
.app-itembar > .app-button {
    float: right;
}
```



JavaScript編

外部プログラムとの共存

豪華なWebアプリのコードをすべて自分たちの力だけで製作するのはたいへんです。そんなときに重宝するのが、便利な処理を集めたライブラリや特定の画面機能を実現するウィジェットなどの外部プログラムです。それらを取り込むことにより、開発期間を短縮できるでしょう。また、自分たちがそういったライブラリを作成することもあるでしょう。そうすれば、ほかのプロジェクトでも流用できて便利です。

Webアプリ自体や外部プログラムのコードは同じ画面上で動くため、互いに影響を与え合います。そして、これらの作りが良くないと、ほかに悪影響を及ぼし思わぬ不具合を招きます。ほかに不必要な影響を与えないような作りにするには、いろいろとコツがあります。ここではそのコツを紹介します。

イベントを埋め込むときの注意

要素の「on＋イベント名」のプロパティに関数を指定することで画面にイベントを埋め込むことができます。これはJavaScriptができたころからある方法で、昔からよく使われてきました。

しかし、この方法では、複数のプログラムを組み合わせてWebアプリを作成する場合に、問題が起こる危険性があります。たとえば、Webアプリが**リスト7**のような構成になっていたらどうでしょうか？　このような場合、ライブラリAの初期処理がライブラリBの初期処理で上書きされてしまい、ライブラリAの初期処理が実行されません。これは、「on＋イベント名」でイベントを埋め込んだ場合には、要素とイベントの組み合わせそれぞれにつき、1つのイベントしか設定できないからです。

リスト8のようにaddEventListener()を使えば、同じ要素に同じイベントを複数埋め込んでもそれらすべてのイベント処理が実行されます。

名前空間を意識する

JavaScriptにおいて、関数の外側で`var 変数名`や`function 関数名(){ ... }`と記述した場合、それらはwindow（ブラウザのウィンドウ全体を表すオブジェクト）に定義されます。つまり、どの外部プログラムや自作スクリプトでも、同じ場所に変数や関数が定義されることになります。そして、同じ名前で変数や関数を定義すると、あとから書いたほうで上書きされます。Webアプリで、**リスト9**のように変数や関数を定義していったらどうなるのでしょうか。定義した変数や関数が外部プログラムのそれと重複してしまうと、正しく動作しないことになります。重複する名前の変数や関数を持つ外部プログラムがなかったとしても、将来的にそういうものが導入されないとも限りません。

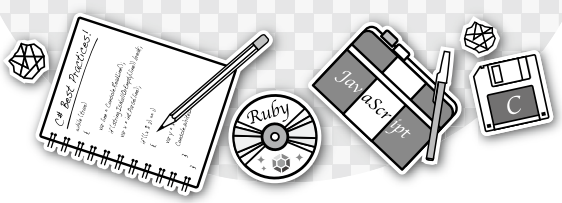
これを回避するため、**リスト10**のようにwindowにオブジェクトを1つ作っておき、変数や関数をそこに作っておくようにしてくださ

▼リスト7　on＋イベントでの埋め込み

```
window.onload = function (){ /* ライブラリAの初期処理 */ }
...中略...
window.onload = function (){ /* ライブラリBの初期処理 */ }
```

▼リスト8　addEventListener()での埋め込み

```
window.addEventListener('load', function (event){ /* ライブラリAの初期処理 */ }, false);
...中略...
window.addEventListener('load', function (event){ /* ライブラリBの初期処理 */ }, false);
```



い。windowに作るオブジェクトの名前は製品名、サービス名にちなんだものが良いでしょう。こうすることにより、変数や関数の上書きの危険性を最小限に抑えることができます。

さらに変数や関数の定義を抑える方法があります。**リスト11**のように**(function(){ ... })()**で囲まれた部分にコード書くと、変数や関数を定義してもローカル変数として扱われるので、windowでの変数や関数の定義を0にできます。ただし、**(function(){ ... })()**で囲まれた部分で定義した変数や関数は外部からは扱えないので、別ファイルのプログラムとのやりとりに制限が出てきます。1ファイルに収まる大きさの処理の場合に限り、こちらのやり方を検討したほうが良いでしょう。

クロスブラウザ対応

WebアプリのHTML、CSS、JavaScriptはブラウザ上で動作します。そして、そのブラウザは各ユーザが用意します。そのため、どのブラウザでWebアプリを動作させるかは、ユーザによって異なることになります。それゆえに、Webアプリはユーザが使うと想定したあらゆるブラウザで正常に動作しなくてはなりません。

基本的にブラウザはW3C(World Wide Web Consortium)や WHATWG(Web Hypertext Application Technology Working Group)で決められた仕様で動作するので、ブラウザ間で動作が大きく違うことはありません。しかし、HTML5やCSSの実装状況や、ブラウザの不具合などにより、同じコードでも動作が違うことがあります。古いバージョンのブラウザの場合、W3CやWHATWGの仕様に準拠していないこともあります。ここでは、そういったブラウザによる動作の違いへの対処方法を紹介します。

まずは、どのブラウザをサポート対象にするかを考える必要があります。ごく一部のブラウザしかサポートしないのは、ユーザにとって不便になります。だからと言って、何でもかんでもサポート対象にすると、開発者の負担になります。その判断基準としてシェアとサポート状況が挙げられます。使っている人がほとんどいないブラウザや、ベンダがサポートしていないブラウザは、サポートしなくていいと思います。

その際に問題になってくるのは、古いバージョンのInternet Explorer(以下、IE)です。2016年1月13日にIE 8のサポートが外れたもののシェアはそれなりにありますし、IE 9とIE 10のサポートは完全に外れてはいません。IE 8〜10をサポートするかどうかはWebアプリの仕様や対象ユーザによって変わってくると思います。

特定ブラウザで機能が実装されていない場合

HTML5の流れにより、さまざまな機能が追加されました。しかし、Webアプリでサポート対象にしているブラウザで、これらの機能が実装されていない場合があります。当然、サポー

▼リスト9　変数／関数定義の例

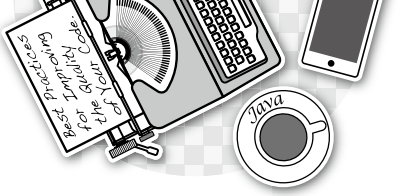
```
var root, item;
function init(){ /* 初期処理 */ }
function getItem(){ /* 処理 */ }
function getList(){ /* 処理 */ }
```

▼リスト10　変数／関数定義の修正案（オブジェクト内に定義）

```
/* ライブラリAの処理 */
var myApp = {
    root: null,
    item: null,
    init: function (){ /* 初期処理 */ },
    getItem: function (){ /* 処理 */ },
    getList: function (){ /* 処理 */ }
}
```

▼リスト11　さらにwindowでの変数や関数の定義を抑える方法

```
/* ライブラリAの処理 */
(function(){
    var root, item;
    function init(){ /* Aの初期処理 */ }
    function getItem(){ /* 処理 */ }
    function getList(){ /* 処理 */ }
})();
```



ト対象の機能が未実装のブラウザでもちゃんと動作しなくてはいけません。しかし、機能が実装されていない以上、その機能をそのまま実装することはできません。そのときの対処方法は場合によりけりですが、一例として**表1**のような実装案があります。

方針として、未実装のブラウザに少しの不便を強いたり、重要でない機能が使えなかったりするのはかまいませんが、重要な機能が使えなかったり、おかしな動きになったりするのは避けたほうがいいでしょう。

機能の実装有無は該当機能のプロパティが実装されているかどうかで判定できます。ファイルドロップの有無を判定する場合、**リスト12**のように実装します。JavaScriptの新機能は何らかのオブジェクト、プロパティが増えていることがほとんどですので、それの存在有無を調べれば、機能の実装有無を判定できます。

機能の実装有無を調べる必要性が多数生じた場合、Modernizr[注2]を使うのも手です。Modernizrを使うとJavaScriptだけでなく、比較的調べにくいHTMLやCSSの実装状況もプロパティ参照のみで調べられます。**リスト12**と同じ判定を、Modernizrを使って実装する場合、**リスト13**のようになります。

特定ブラウザで変な動きをする場合

Webアプリの開発をしていると、特定のブラウザでほかのブラウザと違う動作をする状況に遭遇してしまうことがたまにあります。Webアプリの開発者が悪くない場合も多々ありますが、ユーザにサービスとして提供する以上、対処しないわけにはいきません。こちらの動作の違いの対処は、ブラウザの機能の実装有無の対処に比べて難しいことが多いです。ここでは筆者が実際に遭遇した状況を挙げて、対処方法の例とします。

比較的対処がしやすいのが、標準仕様に準拠していない動作に遭遇した場合です。たとえば、次のようなsplit()を使ったコード

```
'abc123def'.split(/(¥d+)/);
```

注2）ブラウザにHTML5などの機能が実装されているかどうかを調査するためのJavaScriptライブラリ。
https://modernizr.com/

▼表1　HTML5の機能が実装されていない場合の対処例

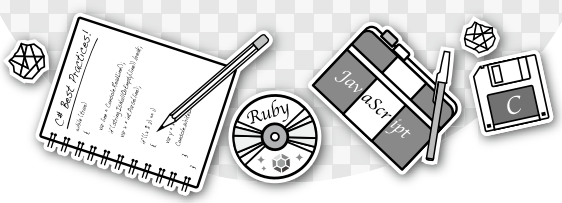
機能	機能がない場合の対処方法
ファイルドロップ	ファイル入力欄を設ける
クライアント側の入力値検証	クライアント側で行わない
デスクトップ通知	別の方法（たとえばタイトル）で通知する

▼リスト12　ファイルドロップ機能の有無を判定

```
if (typeof window.File != 'undefined' && typeof window.FormData != 'undefined') {
    // ファイルドロップの処理
} else {
    // ファイルドロップが実装されていない場合の代わりの処理
}
```

▼リスト13　Modernizrによりファイルドロップ機能の有無を判定

```
if (Modernizr.filereader) {
    // ファイルドロップの処理
} else {
    // ファイルドロップが実装されていない場合の代わりの処理
}
```



をIE 8とモダンブラウザで同じ値で実行してみたところ、次のようになりました。

```
IE 8の実行結果
["abc", "def"]

モダンブラウザの実行結果
["abc", "123", "def"]
```

このようにIE 8のsplit()は独自の仕様で実装されているようでした。そのため、Webアプリでは、split()が仕様どおりの結果になる場合はsplit()を使い、そうでない場合はJavaScriptの自作関数を実行するラッパー関数を作成し、それを使うようにしました(**リスト14**)。

対処しにくいのが、HTMLやCSSを同じように変更しても、変更が反映されなかったり、見た目がおかしくなったりする場合です。決まった対処方法はないのですが、変更処理の方法を変えたり、該当部分のCSSの書き方を変えたりすると直ることがあります。いろいろ試行錯誤してみるしかないのかもしれません。

最後にブラウザの種類自体で判定する方法を紹介します。**リスト15**ではブラウザごとの固有の文字列であるユーザエージェントをもとに判別しています。ブラウザのバージョンアップがあったときに問題が起こりやすいのでお勧めはできませんが、これまでに紹介した方法で対処できない場合は、この方法を使うしかありません。最終手段としてとらえてください。

セキュリティ対応

Webアプリはいろいろな人物からアクセスされます。当然、悪意ある人物からの攻撃を受けることもあります。Webアプリとして公開する以上、こういうものからユーザの情報を守らなくてはなりません。

クロスサイトスクリプティング[注3]対策

サーバで生成されるHTMLが例に出されることが多いクロスサイトスクリプティング(XSS)ですが、JavaScriptでデータを出力する場合でも、これと無縁ではいられません。たとえば、

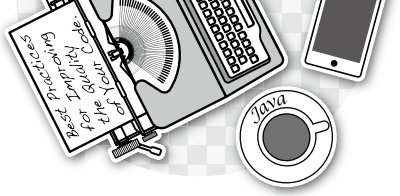
注3) Webサイト上でユーザから受け付けた入力をもとに、動的にHTMLを生成するWebアプリ(掲示板など)で起こりやすい脆弱性の1つ。あるユーザがサイトの入力フォームに何らかのコードを入力すると、それがそのままHTMLに埋め込まれ、そのサイトを閲覧したコンピュータ側でそのコードが実行されてしまうというもの。

▼リスト14　split()の動作の違いの解決方法

```
sample.split = function (text, expression, limit) {
    if (typeof this._testRegExpSplitResult != 'boolean') {
        // split()が仕様どおりか確認
        this._testRegExpSplitResult = 'a'.split(/(a)/).length == 3;
    }
    if (this._testRegExpSplitResult) {
        // split()が仕様どおりならsplit()を実行
        return String(text).split(expression, limit);
    }
    // split()が仕様どおりでないブラウザ用の処理
    this.splitWithJs(text, expression, limit);
}
```

▼リスト15　ブラウザの種類自体による分岐(IEを判定する例)

```
var isIE = navigator.userAgent.indexOf('MSIE') >= 0;
...中略...
if (isIE) {
    // IEの場合のコード
} else {
    // IE以外の場合のコード
}
```



次のようにサーバから取得した値をinner HTMLなどでそのまま埋め込んだ場合、どうなるでしょうか？

```
element.innerHTML = textFromServer;
```

通常は何の問題もありませんが、textFromServerにほかのユーザが入力した値**<script>［任意のコード］</script>**のような値が設定されていると、それがHTMLとして挿入されるため、そのユーザが仕込んだコードが自分のアカウント上で実行されてしまいます。サーバから取得した値に文字列を連結して設定しても、同様の問題があります。

サーバから値に埋め込む際のXSS対策の常套手段はHTMLのエスケープですが、JavaScriptではテンプレートエンジンを除き、その手段が使われることはまずありません。よく使われるのは、**リスト16**のように文字データを要素に追記する方法です。この場合、挿入した文字列はあくまで文字列であるため、**<script>［任意のコード］</script>**のような値が設定されていても、そのまま表示されるだけです。

▼リスト16　文字データを要素に追記する方法

```
var textNode = document.createTextNode(textFromServer);
element.innerHTML = '';
element.appendChild(textNode);
```

▼リスト17　setTimeout()の例

```
①悪い例
setTimeout('action()', 1000);
②良い例
setTimeout(function () { action(); }, 1000);
```

▼リスト18　プロパティ名を文字列変数で指定する例

```
①悪い例（evalを使用）
var page = eval('app.' + actionName);
②良い例（eval未使用）
var page = app[actionName];
```

evalを使用しない

使われているのをたまに見かけるeval、または文字列をスクリプトとして実行する処理ですが、原則として使ってはいけません。**リスト17－①**のようにsetTimeout()やsetInterval()の第1引数を文字列で設定しているサンプルプログラムをたまに見かけますが、これは昔の書き方ですので、やらないようにしてください。**リスト17－②**のように文字列の代わりに関数を設定すれば、同じことができます。

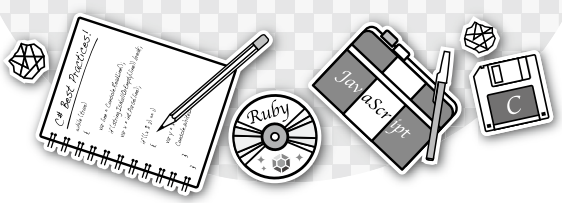
evalを使ってしまいがちなもう1つの例は、**リスト18－①**のようにプロパティ名を文字列変数で指定するときです。この場合、actionNameに**x; ［任意のコード］**のような値が設定されていると、そのコードが実行されてしまいます。**リスト18－②**のように、**[]**で囲まれた変数を使ってそのオブジェクトのプロパティを参照すれば、**リスト18－①**と同じことができます。しかもこの場合、actionNameはappのプロパティでしかないので、任意のコードが実行されることはありません。

クロスドメイン対応

よく知られていることですが、Ajax（Asynchronous JavaScript + XML）では、HTMLの取得元のドメインとは違うドメインへのリクエストに制約（クロスドメイン制約）がかけられています（**図4**）。なぜなら、この制約がなければ、Ajaxを使って罠（わな）サイトにアクセスしたユーザの個人データを盗むことができるからです。ただ、この制約によりマッシュアップ[注4]がやりづらくなっているのも事実で、さまざまな回避策が生まれました。

最初に紹介するのはJSONP（JSON with Padding）で、昔からあった方法です。これは、script要素からの参照にはクロスドメイン制

注4）Web APIにより複数のWebサービスを統合して、新しいサービスを提供すること。



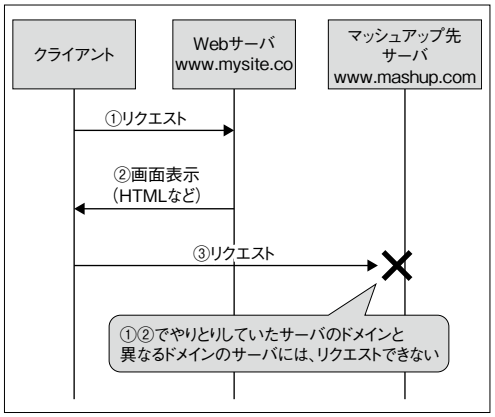
約がかかっていないことを利用します。実際には、**リスト19**のような値を出力するURLを、**リスト20**のようにscript要素で読み込むことによって実装します。ただし、Ajaxと同じように罠サイトを使って外部からデータを盗むことができるので、非公開情報を扱ってはいけません。

次に紹介するのはCORS(Cross-Origin Resource Sharing)です。ドメインを超えてデータのやりとりをする場合、こちらを使うことをお勧めします。マッシュアップ先のサーバが、レスポンスでアクセスを許可するドメインを指定することにより、そのドメインとのクロスドメインのAjax通信ができるしくみです。ちなみに、レスポンスでクロスドメイン通信を許可されなかった場合には、ブラウザが強制的に通信を失敗扱いにします。

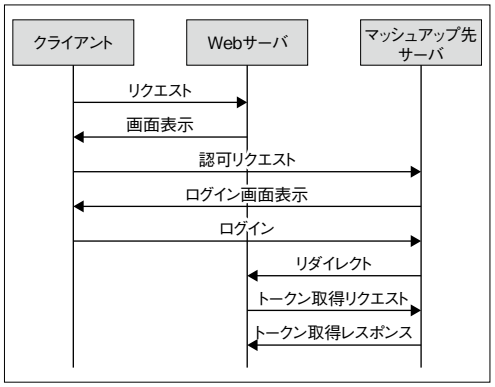
詳細なやり方の説明は省略しますが、マッシュアップ先のサーバがレスポンスの際に、**リスト21**のようなヘッダでアクセス可能なドメインを返す以外、普通のAjaxと同じです。ドメインの代わりに*を返せばどこからでもアクセスできるようになりますが、アクセス可能なドメインはできるだけ少なくしたほうがいいでしょう。

公開情報を扱う場合はこれだけでいいのですが、非公開情報を扱う場合は認証にも気をつけなければなりません。認証の方式としてはとくに理由がない限り、OAuth 2.0を使うべきでしょう。**図5**はWebアプリでOAuth 2.0を使う場合のシーケンスです。OAuth 2.0の認証が成功するとアクセストークンがWebページ側に渡りますので、それを使ってユーザを認証すれば、安全に非公開情報をやりとりできます。少し複雑なしくみですが、非公開情報を扱う以上、必要なコストだと思います。SD

▼図4　クロスドメイン制約



▼図5　OAuth 2.0のシーケンス



▼リスト19　JSONPの出力例

```
callback({
    'key1': 'value1',
    'key2': 'value2'
});
```

▼リスト20　JSONPのアクセス方法

```
var element = document.
createElement('SCRIPT');
element.src = '/path/to/list20';
document.body.appendChild(element);
```

リスト20のURL

▼リスト21　アクセスを許可するドメインを指定するためのHTTPレスポンスヘッダ

```
Access-Control-Allow-Origin: http://external.com
```

BACKNUMBER バックナンバーのお知らせ

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。

2016年3月号

第1特集
チーム開発をまわす現場のアイデア

第2特集
あなたの知らないCOBOLの実力

一般記事
・iPad Proのさきに見えてくるもの
・Webサイトが改ざん！　サイトオーナがとるべき行動と注意点

定価（本体1,220円+税）

2016年2月号

第1特集
［最新］MySQLとPostgreSQL徹底比較

第2特集
1Gbps超ネットワーク高速化時代の適切なLANケーブリングの教科書

一般記事
・Android Studioのスタイルで効率アップ!

定価（本体1,220円+税）

2016年1月号

第1特集
はじまっています。ChatOps
導入を決めた7社の成功パターン

第2特集
手軽さとコード化しやすさが人気!
Ansibleでサーバ管理構成を省力化

新連載
・Androidで広がるエンジニアの愉しみ

定価（本体1,220円+税）

2015年12月号

第1特集
［決定版］Docker自由自在
実用期に入ったLinuxコンテナ技術

第2特集
ネットワーク・システム管理の定石
SNMPの教科書

短期連載
・クラウド時代のWebサービス負荷試験再入門

定価（本体1,220円+税）

2015年11月号

第1特集
すいすいわかるHTTP/2
HTTP/1.1から変わること・変わらないこと

第2特集
攻撃を最前線で防ぐ
ファイアウォールの教科書

特別企画
・SMB実装をめぐる冒険
File System for Windowsの作り方

定価（本体1,220円+税）

2015年10月号

第1特集
多層防御や感染後対策を汎用サーバに実装
攻撃に強いネットワークの作り方

第2特集
Webメールの教科書
クラウドサービス利用か？　自社で構築か？

特別付録
・創刊300号記念　Vim&Emacsチートシート

定価（本体1,220円+税）

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。

DIGITAL デジタル版のお知らせ

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。

第2特集

オブジェクトストレージの教科書

OSSと3つの製品事例から学ぶ、新しいデータ管理のしくみ

大量のファイルをAPIでシンプルに出し入れできる「オブジェクトストレージ」。とくにWebエンジニアからの注目が大きいこの新しいストレージについて、基本から応用までとことん解説します。

Part1では、オブジェクトストレージ全般の話題として、従来のファイルサーバやFTPサーバとは何が違うのかというところから解説をはじめ、オブジェクトストレージの概要と機能、メリットを紹介します。Part2では、オブジェクトストレージの大きな特徴である分散保存のしくみをひも解きながら、2つのOSS「OpenStack Swift」「Ceph」について解説していきます。そしてPart3では、3つのオブジェクトストレージサービスについて、開発にかかわったエンジニアが内部のしくみを公開しつつ、利用者目線でのTipsを紹介します。

contents

part 1

オブジェクトストレージとは何か?

ファイルサーバ／FTPサーバとの違いから考える

Amazon S3に代表される「オブジェクトストレージ」ですが、その特徴はズバリ「大量のファイルを高いスループットでやりとりできること」「インターフェースがシンプルであること」。本章ではオブジェクトストレージの概要と機能について紹介し、利用するうえでどのようなメリットがあるのかをみていきます。

Author 中井 悦司(なかい えつじ) レッドハット株式会社 Twitter @enakai00

オブジェクトストレージって何がすごいの?

みなさんが「オブジェクトストレージ」という言葉を初めて聞いたのは、いつごろだったでしょうか? Amazon Web Services(AWS)によって、オンラインストレージサービス「Amazon S3」の提供が開始されたのが2006年、そして、Amazon S3に類似の環境を自前で構築するオープンソースソフトウェア「OpenStack Swift」が公開されたのが2011年です。これらのサービス／技術をきっかけに、オブジェクトストレージに興味を持った方も多いことでしょう。

そして、初めてこれらの機能を聞いて、どのように感じたでしょうか? さまざまなストレージ技術を知っているインフラエンジニアであれば、「え? それだけ?」と感じた方も少なくないかもしれません。オブジェクトストレージの機能を端的に説明すると、「ファイル単位でデータを出し入れするネットワーク上のストレージサービス」です。NFS(Network File System)サーバのように、ストレージ上のファイルを直接に読み書きするというわけではなく、あくまでファイルを出し入れするだけの機能になります。言ってみれば、FTPサーバにファイルをアップロード／ダウンロードするようなものです(**図1**)。

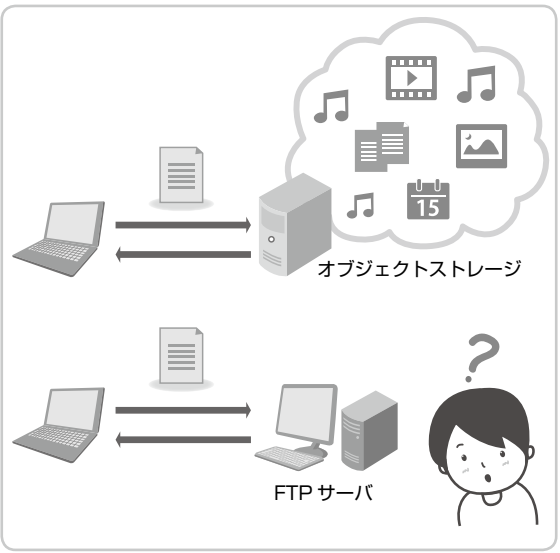
実際のところ、筆者も最初は同じように感じました。「画像や動画など、さまざまなオブジェクトが保存できる未来のストレージ!」というような宣伝文句を耳にすることもありますが、「画像や動画」と言っても、その実体はただのファイルです。ファイルが保存できることを自慢してどうしようというのでしょうか?——この理解は、間違っているというわけではありませんが、もちろんこれでは、オブジェクトストレージがこれほど広く活用されるようになった理由は説明できません。

オブジェクトストレージの真価とは

オブジェクトストレージが単なるFTPサーバと異なる点は、大きく2つあります。1つは、多数のファイルを分散保存するアーキテクチャにより、大量のファイルに対するアクセスを高いスループットで処理するという点です。

Web上で提供されるサービスの多くは、大

▼ 図1 オブジェクトストレージとFTPサーバは何が違う?



量の画像・動画ファイルを取り扱う必要があります。2012年には、AWSから、Amazon S3に保存されたオブジェクト(ファイル)の総数が1兆個を超えたという発表もなされています[注1]。オブジェクトストレージが画像や動画の保存だけに最適化されているわけではありませんが、一般的なファイルサーバやFTPサーバではなく、オブジェクトストレージを利用することで初めて実現できたサービスも多いと想像されます。インフラエンジニアの視点では、このような分散保存のしくみが最も興味を引かれる点となるでしょう。

そして、もうひとつの違いは、アプリケーション開発の視点から見えてきます。

FTPサーバは、UNIX/Linuxのディレクトリにファイルを保存するというしくみが外からも見えており、利用者はUNIX/Linuxのディレクトリ構造を意識して操作する必要があります。たとえばFTPサーバに保存したファイルの総数や総容量を知りたい場合、みなさんはどうするでしょうか？　年季の入ったシステム管理者であれば、FTP接続したあとに、cdコマンドとlsコマンドを駆使して、ファイルの個数や容量を計算していくシェルスクリプト(もしかしたら、Perlやawkのスクリプト)を書いた思い出があることでしょう。

しかしながら、Webアプリケーションを開発するプログラマにとって、そんなことはどうでもよい話です。オブジェクトストレージは「ファイル単位で出し入れする」という機能に特化すると同時に、アプリケーションプログラムにとって必要なインターフェースを“厳選して”提供しています。たとえばOpenStack Swiftの場合、プログラマが意識するのは、「アカウント」「コンテナ」「オブジェクト」の3種類の操作対象です。これらは、REST API[注2]の用語として「リソース」とも呼ばれており、それぞれの役割は**表1**のとおりです。

▼表1　OpenStack Swiftの操作対象リソース

リソース	説明
アカウント	ストレージサービスを利用するテナント
コンテナ	オブジェクトを保存する入れ物
オブジェクト	コンテナに保存するファイルの実体

アプリケーションプログラマは、これらのリソースを操作するコードを書くことで、オブジェクトストレージを利用したアプリケーションを開発していきます。それぞれのオブジェクトストレージは、アプリケーションから利用するためのライブラリをRuby、Python、Java、C言語といった主要なプログラム言語に対して用意しており、コード内ではこれらのライブラリが提供する関数やメソッドを呼び出します。アプリケーションとオブジェクトストレージの間はREST API、すなわちHTTPプロトコルによる通信が行われますが、プログラマ自身がそのようなプロトコルを意識する必要はありません。

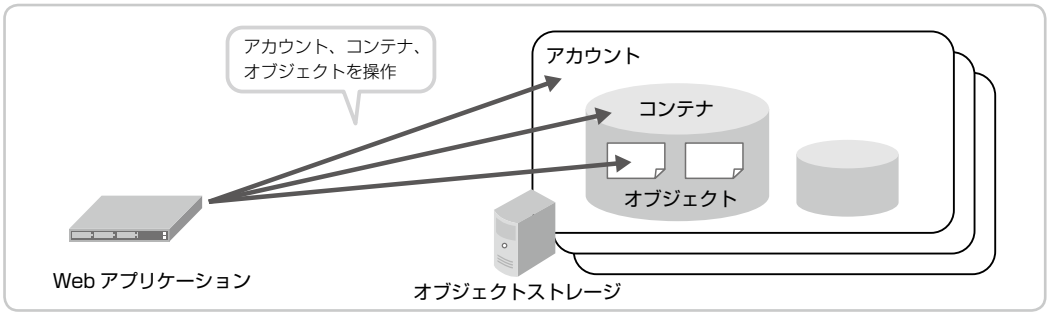
このように、アプリケーションプログラム、とりわけWebアプリケーションと連携して使用する際に、開発の利便性を高めるように配慮されている点が、オブジェクトストレージのもうひとつの特徴になります。

オブジェクトストレージのAPI

オブジェクトストレージの背後にある分散保存のしくみについては、このあとのPart2で解説することにして、ここではAPIの概要、すなわちアプリケーションプログラムから利用できる機能を確認しておきます。代表例として、OpenStack SwiftのAPIを紹介しますが、そのほかのオブジェクトストレージでも基本的な機

注1)「Amazon S3―オブジェクト数が1兆個に！」
URL http://aws.typepad.com/aws_japan/2012/06/amazon-s3-the-first-trillion-objects-.html
注2) REST APIは「REST(Representational State Transfer)」の原則に従って設計されたAPI(アプリケーション間の通信規約)で、HTTPプロトコルを用いて通信を行い、URLで操作対象のリソースを識別します。

▼図2　OpenStack Swiftの操作体系



能はほぼ同じです。

前述のように、OpenStack Swiftを利用するアプリケーションは、**表1**に示した3種類のリソースに対して各種の操作を行います(**図2**)。

アカウントに対する操作

OpenStack Swiftが提供するストレージサービスはマルチテナント型になっており、複数のテナントから使用できます。「アカウント」はストレージサービスを利用するテナントを表すもので、アプリケーションは特定のアカウントを指定して、オブジェクトストレージにアクセスします。それぞれのアカウントは、オブジェクト(ファイル)を保存する入れ物となるコンテナを自由に作成できます。アカウントに対する主要な操作は次のとおりです。

・アカウントが所有するコンテナの一覧を取得
・アカウントが所有するコンテナ数、バイト数、メタデータを取得
・アカウントのメタデータを設定／更新

メタデータというのは、Key-Value形式の任意のテキストデータで、アプリケーションが必要とする情報を自由に設定できます。アカウント、コンテナ、オブジェクトのそれぞれに対して、複数のメタデータを設定できます。また、このあとで説明するオブジェクトの有効期限設定のように、メタデータを用いてリソースの管理設定を行うこともあります。

コンテナに対する操作

コンテナは、オブジェクトを保存する入れ物です。UNIX/Linuxのディレクトリのような階層構造は作れません。階層的にファイルを保存したい場合は「directory01_file01」のように、オブジェクト名に対して擬似的にディレクトリ名を含めて利用します。コンテナに対する主要な操作は次のとおりです。

・コンテナを作成
・空のコンテナを削除
・コンテナに含まれるオブジェクトの一覧を取得
・コンテナに含まれるオブジェクト数、バイト数、メタデータを取得
・コンテナのメタデータを設定／更新

コンテナのメタデータには、コンテナの使用目的やアプリケーションからみたコンテナの状態を設定するといった使い方が考えられます。たとえば画像共有アプリケーションにおいて、あるコンテナに対して「状態＝無効」というメタデータをセットすると、そのコンテナのファイルは利用者からは参照できなくなる、といった機能を実装できるでしょう。REST APIでコンテナを操作する際は、「アカウント／コンテナ」の形式で、どのアカウントのコンテナを対象とするのかを示します。

オブジェクトに対する操作

オブジェクトは、コンテナに保存するファイ

ルの実体です。REST APIでオブジェクトを操作する際は、「アカウント／コンテナ／オブジェクト」の形式で、どのアカウントのどのコンテナに含まれるオブジェクトであるのかを示します。オブジェクトに対する主要な操作は次のとおりです。

・オブジェクトをダウンロード
・オブジェクトをアップロードして、メタデータを設定
・オブジェクトを削除
・オブジェクトを別の名前でコピー
・オブジェクトのメタデータを取得
・オブジェクトのメタデータを設定／更新

OpenStack Swiftではオブジェクトのメタデータを利用して、有効期限も設定できます。メタデータ「X-Delete-At」で指定された日時、あるいは「X-Delete-After」で指定された時間が経過すると、自動的にオブジェクトが削除されます。

オブジェクトストレージの管理機能

ここまでの説明で、オブジェクトストレージとFTPサーバの違いがつかめたのではないでしょうか。「ファイル単位で出し入れする」という点は同じですが、オブジェクトストレージの場合は、アプリケーションから利用しにくいディレクトリ構造の概念を思い切って削除する一方、オブジェクト数／バイト数の取得やメタデータの設定など、アプリケーションの作成に便利な機能が追加されています。またOpenStack Swiftでは、オブジェクトの有効期限設定のような管理機能も用意されています。

このような管理機能は、各種オブジェクトストレージの差別化ポイントにもなります。たとえばOpenStack Swift、あるいはAmazon S3では、オブジェクトのバージョン管理機能が提供されており、同一のオブジェクトを上書きでアップロードした場合に、古いオブジェクトを削除せずに保存したままにできます。オブジェクトのバージョン番号を指定することで、過去のオブジェクトを取り出すことができるようになります。

そのほかには、保存したオブジェクトにURLを割り当てて、Webブラウザからアクセス可能にするような機能もあります。HTMLファイルを保存して静的なWebサイトを作ったり、あるいはほかのWebサイトからリンク可能な形で画像ファイルを公開するといった使い方ができます。

コラム

ミニマム機能に特化したオブジェクトストレージ「Minio」

本文では、追加の管理機能がオブジェクトストレージの差別化ポイントになると説明しましたが、世の中には余計な追加機能を持たないことを差別化ポイントとするおもしろいオブジェクトストレージがあります。分散ファイルシステム「GlusterFS」のオリジナル開発者である、Anand Babu（AB）が開発を進める「Minio」[注A]です。

Minioが提供するAPIは、基本的にはコンテナの作成とオブジェクトのアップロード、ダウンロード、そしてコンテナとオブジェクトの一覧表示、ただそれだけです。メタデータの設定や有効期限の設定などはありません。これは、追加機能を実装していくと、オブジェクトストレージの処理が段々と複雑になっていき、本来の特徴であるスケーラビリティ、すなわち大量のアクセスを高速に処理するという特性が失われてしまう、という考え方に基づきます。オブジェクトストレージは、あくまでもその本来の機能に特化して、それ以外の追加機能はオブジェクトストレージを使用するアプリケーション側で実装したほうが効率的で柔軟性も高くなるはずだ、というのが開発者であるABの主張です。

彼はGlusterFSを開発する中で、分散型のストレージソフトウェアにおいて、処理性能を犠牲にせずに高度な機能を実現することがいかに困難であるかを学んだと言います。その経験を最大限に活かして開発されたMinioがどのようなものになるのか、筆者は興味を持って開発の進捗を見守っています。

注A）URL https://www.minio.io

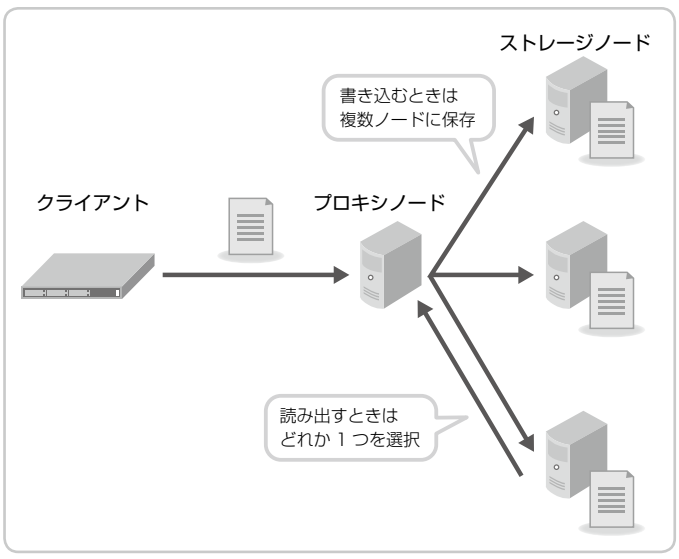
ファイルの冗長保存と注意点

オブジェクトストレージでは一般に、サーバ障害時に保存したファイルを失わないための冗長化もなされています。OpenStack Swiftの場合、ストレージサービスを利用するクライアントは、フロントにある「プロキシノード」を経由して、データのやりとりを行います(**図3**)。プロキシノードは、受け取ったファイルを、背後にあるストレージノードに保存するわけですが、この際複数のストレージノードに対してファイルの複製を保存します。一方ファイルを読みだすときは、複製されたファイルのどれか1つを選んで読み出すことで、負荷分散を行います。また、プロキシノード自体の障害にも対応するため、複数のプロキシノードを並べて、ロードバランサで負荷分散するような構成も可能になります。

そして、このしくみに関連して注意しないといけないことは「Eventual Consistency(結果整合性)」の考え方です。たとえばあるストレージノードが障害停止している間、プロキシノードは代替のストレージノードに複製を保存します。そのあと、停止していたストレージノードが復旧すると、このノードには古いファイルが残ったままになっています。しばらくすると、新しい複製が自動的にコピーされて正しい状態に戻りますが、それまでの間に、クライアントは古いファイルを読みだしてしまう可能性があります。つまりクライアントは、必ずしも最新のファイルが読み出せるとは限らないのです。Eventual Consistencyというのは、このような「しばらくすると正しい状態になる」という動作を表します。

アプリケーションの特性上、確実に最新のファイルを取得する必要がある場合は、APIリクエストに「X-Newestヘッダ」をセットします。この場合プロキシノードは、すべての複製を読み込んでその中で最新のものを返送します。当然ながらアクセス性能は悪くなりますので、本当に必要な場合にのみ使用してください。

▼図3 OpenStack Swiftの複製保存のしくみ



まとめ

本パートではオブジェクトストレージの特徴として、分散保存型のアーキテクチャに加えて、アプリケーションプログラムからの利用に特化したAPI設計があることを説明しました。SNS(ソーシャルネットワーキングサービス)に代表される、大量の画像や動画ファイルを扱うWebアプリケーションの増加と相まって、オブジェクトストレージの需要が高まっていることが理解できたものと思います。

このあとに続くPart2では、オープンソースとして内部仕様が公開されているOpenStack SwiftとCephを例として、オブジェクトストレージの分散保存アーキテクチャを解説していきます。SD

part 2

オブジェクトストレージの分散処理を理解しよう

OpenStack SwiftとCephを支える技術

複数の場所にファイルをどのように分散保存するのか、またそれらへのアクセスをどのように振り分けるのか。本章ではオブジェクトストレージの大きな特徴である分散保存のしくみを扱います。また、OSSのオブジェクトストレージOpenStack SwiftとCephについて、両者の違いに触れながら解説していきます。

Author 中井 悦司(なかい えつじ) レッドハット株式会社 Twitter @enakai00

分散ストレージの歴史

Part1では、オブジェクトストレージの特徴の1つとして、多数のファイルを分散保存するという点を紹介しました。分散保存する目的としては、1台のサーバでは保存しきれない大量のファイルを保存することに加えて、アクセス性能の確保が挙げられます。

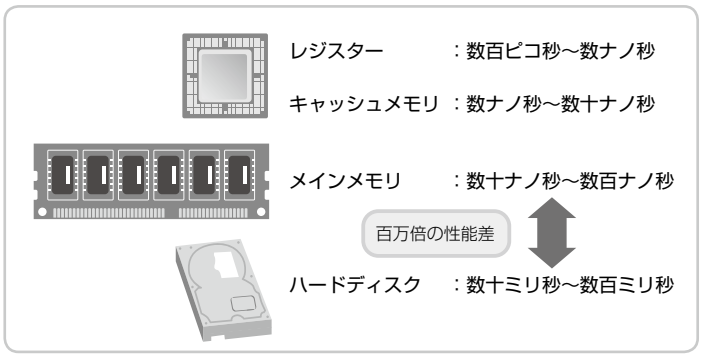
ご存じのように、コンピュータが利用する記憶装置はCPU内部のレジスターから始まり、キャッシュメモリ、メインメモリ、ハードディスクという形でアクセス速度の順番に階層化されています(**図1**)。そして、物理的な可動部分を持つハードディスクのアクセス速度は、メインメモリの100万倍ほども遅くなります。そこで、ハードディスクとのデータのやりとりを高速化するために利用されるのが、多数のハードディスクに同時にアクセスするという手法です。

典型例として挙げられるのが、ハードディスクのストライピング構成です。サーバにRAIDコントローラを搭載すると、コントローラの機能によって複数のハードディスクを束ねたRAIDアレイを構成して、OSに対しては単一のハードディスクであるかのように見せかけられます。OSからのデータアクセスを複数のハードディスクで並列処理することにより、単位時間あたりのデータ転送量(スループット)を向上できるのです。このしくみには、OS、あるいはその上で稼働するアプリケーションからみたときに、背後のしくみを意識せずとも、単体のハードディスクと同じ方法で扱えるというメリットがあります。

ただしこの方法の場合、複数サーバでの共用ができない、あるいは1台のサーバに搭載可能なハードディスクの容量を超えてデータを保存できないといった限界があります。そこで、一般的な企業システムで広く利用されるようになったのが、SAN(Storage Area Network)ストレージシステムです。これは大量のハードディスクを搭載した専用のストレージ装置をファイバーケーブルを介して複数サーバから共用する技術です。

SANストレージシステムは現在でも広く利用されていますが、その一方で専用のストレージ装置を用いるのではなく、一般的なサーバの内蔵ディスクを束ねることで、大容量ストレージを構成するという考え方も生まれてきました。現在オブジェクトストレージとして利用されているものは、その大部分がこちらの考え方になります。サーバ上のソフトウェアでストレージを構築

▼図1 記憶装置の階層構造



するので、「ソフトウェアストレージ」と呼ばれることもあります。

分散ストレージソフトウェアのアーキテクチャ

複数サーバの内蔵ディスクを束ねるタイプの「分散ストレージソフトウェア」では、一般にディスク領域を提供するサーバを「ストレージノード」、そしてこれらのディスクにアクセスして使用する側のサーバを「クライアントノード」と呼びます。このとき、クライアントノードから多数のストレージノードへのアクセスを仲介する役割がどこかに必要となります。この「仲介役」のしくみを理解することが、分散ストレージソフトウェアのアーキテクチャを理解するポイントになります。

分散ストレージソフトウェアには、大きく2つのタイプがあります。それぞれのタイプについて、「仲介役」がどこにあるのか確認しておきましょう。

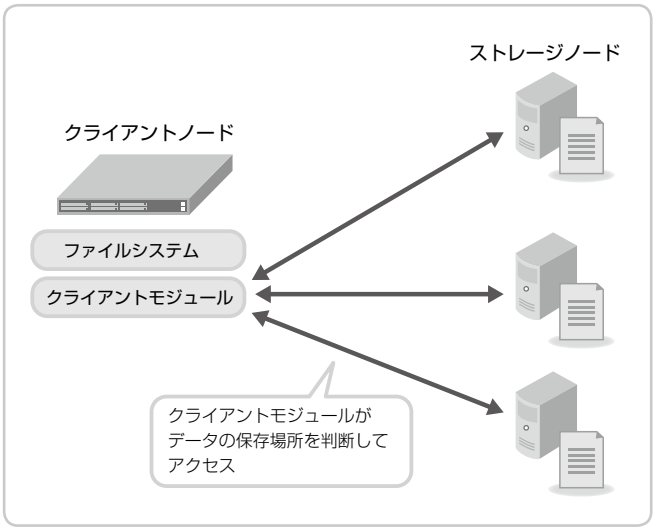
1つは、ネットワーク経由でファイルシステムをマウントして利用する「分散ファイルシステム」と呼ばれるタイプです。オープンソースとして開発されているものには、GlusterFSやCephFSなどがあります。このタイプでは、ファイルシステムをマウントするクライアントノード側に仲介役の機能が存在します(**図2**)。アプリケーションがファイルシステム内のファイルにアクセスすると、OS上で稼働するクライアントモジュールがファイルの実体があるストレージノードを判断して、該当のストレージノードに対するアクセスを行います[注1]。

一方、本特集のテーマであるオブジェクトストレージの場合は、クライアントノードのOSに特別なモジュールは必要ありません。Part1で説明したように、HTTPプロトコルを用いたREST APIでアクセスしますので、クライアントノードからは特定のサービスのURLに対してアクセスリクエストを送信することになります。このリクエストを受け取るサーバは、一般にプロキシノード(もしくは、ゲートウェイノード)と呼ばれており、その背後で複数のストレージノードに分散アクセスを行います。

◆ ◆ ◆

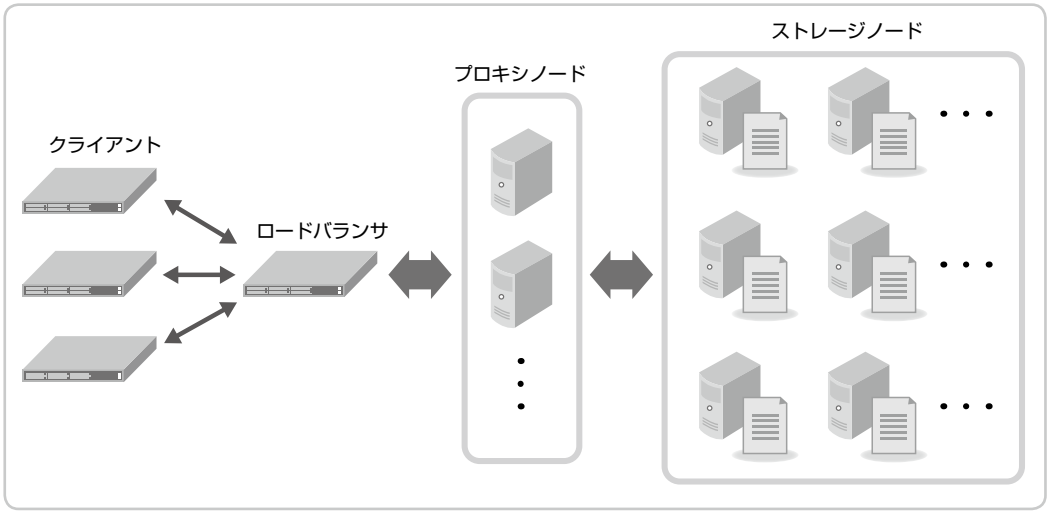
次節からはまず、OpenStack Swiftの場合を例として、プロキシノードが分散アクセスを行うしくみを解説します。またもう1つの興味深い例として、Cephのしくみを解説します。Cephは、RADOSと呼ばれるオブジェクトストレージの機能を持っており、これをベースとして、オブジェクトストレージに加えてブロックデバイス(RBD：Rados Block Device)や分散ファイルシステム(前述のCephFS)の機能を実現しています。ここではとくに、Cephの中核となるRADOSのしくみを解説します。

▼図2 分散ファイルシステムのアーキテクチャ



注1) GlusterFSの場合はユーザランドで稼働するFUSEモジュール、CephFSの場合はカーネルモジュールがこの機能を提供します。

▼図3　OpenStack Swiftを構成するサーバ群



OpenStack Swift

ここからは、オブジェクトストレージの具体例として「OpenStack Swift」を紹介します。なお、OpenStack Swiftの内部構造は、バージョンアップとともにより複雑になっています。ここではその本質を理解するために、まずは少し前のバージョンであるGrizzlyリリースでの実装を基に解説します。最新バージョンにおける構造の違いについては、別途、補足説明を加えてあります。

アーキテクチャについて

OpenStack Swiftの全体像は、**図3**のようになります。REST APIによるリクエストを受け付けるプロキシノードが複数あり、フロントのロードバランサによって負荷分散が行われます。さらに、プロキシノードの背後に多数のストレージノードが存在します。Part1で説明したように、OpenStack Swiftでは「アカウント」「コンテナ」「オブジェクト」の3種類のリソースを扱いますが、これらのリソースは、すべて複数のストレージノードに分散保存されます。

アカウントやコンテナが分散保存されるというと妙な感じがしますが、たとえばコンテナであれば、それぞれのコンテナに保存されたオブジェクトの一覧といった管理情報を保存する必要があります。このような管理情報が分散保存されていると考えてください。

そしてこれらの管理情報、あるいはオブジェクトの実体となるファイルをどのストレージノードに保存するかを決定するルールが必要になります。これは「リング」と呼ばれる静的なハッシュテーブルで実現されます。――と言うと難しそうですが、実際にはとてもシンプルでわかりやすいしくみです。順を追って説明していきましょう。

まずはじめに、論理的な保存場所を示す多数の「パーティション」を用意して、それぞれのパーティションをストレージノードに搭載されたハードディスクに割り当てます[注2]。**図4**のように、1つのハードディスクに対して複数(一般には数百個程度)のパーティションが割り当てられます。そして、保存対象とするリソースの名前(アカウント名、コンテナ名、オブジェクト名)のハッシュ値から保存先のパーティションを決

注2）オブジェクトストレージでは、一般にストレージノードのハードディスクに対してRAIDを構成することはしません。個々のディスクを個別にオブジェクト(ファイル)の保存領域として使用します。

定する「対応表」を用意します。リングというのはこの対応表のことで、アカウント、コンテナ、オブジェクトのそれぞれに対して個別のリングが用意されます。OpenStack Swiftでは、データを冗長保存するために、標準で3つのレプリカ(複製)を異なるストレージノードに保存するようになっており、**図5**のように3つのレプリカに対して、それぞれに保存先のパーティションが決まります。

ここまでの準備ができれば、このあとの動作はそれほど難しくはありません。**図5**に示したリング情報を記録した「リングファイル」をプロキシノード、およびすべてのストレージノードに配布しておき、それぞれのノードは、リングファイルを見てリソースの保存先となるパーティションを判断します。たとえば、新たなオブジェクト(ファイル)の保存リクエストを受け取ったプロキシノードは、オブジェクト名のハッシュ値を計算したあとに、リングファイルを参照してこのオブジェクトを保存する3つのパーティションを決定します。

リング作成のしくみ

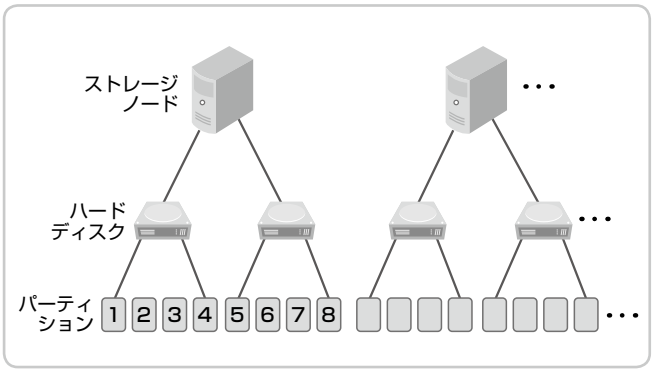
これまでの説明からもわかるように、**図5**に示したリングの構造がOpenStack Swiftを理解するポイントになります。それでは、このリングはどのようにして作成するのでしょうか?——ここにOpenStack Swiftの最大の特徴があります。

リングファイルは、「リングビルダー」と呼ばれるツールで事前に作成しておきますが、その際ストレージノードとそれらに搭載された個々のハードディスクの情報、そして各ハードディスクの「ゾーン」と「重み」を指定します。するとリングビルダーは、それぞれのハードディスクに対して、重みに比例した数のパーティションを割り当てます。各パーティションには均一にデータが割り当てられるので、結果として重みの大きいハードディスクにはより多くのデータが保存されます。一般には、個々のハードディスクの容量に応じて、重みの値を調整します。

さらに3つのレプリカには、異なるゾーンのハードディスクを割り当てます。たとえば、複数のラックにストレージノードを配置する場合、ラックごとにゾーンを分けておけば、3つのレプリカは必ず別々のラックに分配されます。ネットワークスイッチの障害で、特定ラックのストレージノード全体がアクセス不能になった場合でも、プロキシノードがすべてのレプリカにアクセスできなくなる事態は避けることができます。リングビルダーには、このような制約条件を満たすよう“うまいぐあいに”リングを作成するアルゴリズムが組み込まれています。

ストレージノードをあとから追加する場合は、リングビルダーを用いてリングを作成しなおし

▼図4 ハードディスクに対するパーティションの割り当て



▼図5 リング(ハッシュテーブル)の構造

ハッシュ値	xxx	yyy	zzz	…	…	…	…
レプリカ#1	1	4	3	9	5	2	1
レプリカ#2	3	2	5	8	7	5	2
レプリカ#3	7	6	9	4	6	1	4

それぞれのレプリカを保存するパーティションの番号

ます。その際、既存のオブジェクトについて、配置先のパーティションが変更される可能性があります。作成しなおしたリングファイルを各ノードに配布すると、新たな配置情報に基づいて既存のオブジェクトを再配置する「リバランス」の処理が行われます。リバランスの実施中であっても、オブジェクトストレージへのアクセスは継続できるように考慮した設計がなされています。

最後に、最新バージョンでの構造の違いを補足しておきます。現在のバージョンでは、地理的に離れた場所のストレージノードを組み合わせられるよう、「ゾーン」に加えて「リージョン」の指定ができるようになっています。これにより、複数地域にまたがってレプリカを配置したり、あるいは異なる地域間でのファイル転送時には、ファイルを圧縮してから転送したりするなどのオプション設定が可能になります。そのほかには、ハードディスク容量の使用効率を上げるために、ファイルを完全に複製するのではなく、冗長性を持たせた形で分割配置する「Erasure Coding」と呼ばれるしくみも利用できるようになっています。

Ceph

つづいて、オブジェクトストレージのもうひとつの具体例として「Ceph」を紹介します。前述のように、CephはRADOSというオブジェクトストレージの機能をベースに、ブロックデバイス(RBD)や分散ファイルシステム(CephFS)の機能も提供するというユニークなアーキテクチャを持っています。ここでは、その基礎となるRADOSのしくみを中心に解説します。なお、Cephをオブジェクトストレージとして使用する際は、プロキシノードに相当する「RADOSゲートウェイ」が稼働するノードを用意します。クライアントからのリクエストを受け取ったRADOSゲートウェイが、その背後にあるストレージノードとのデータのやりとりを行います。

RADOSのしくみ

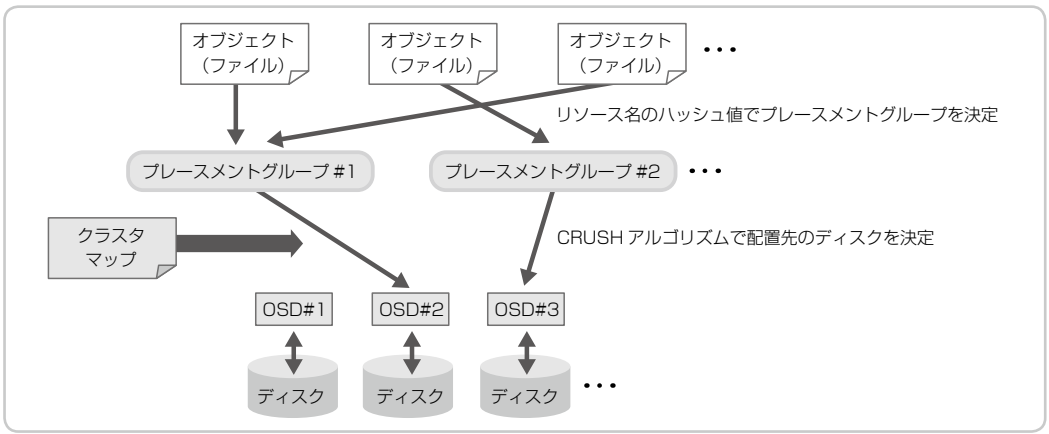
OpenStack Swiftは、1つのディスクに対して複数の論理的な保管場所(パーティション)を割り当てる構造でしたが、RADOSの場合は1つのディスクが1つの保管場所になります。ストレージノードではそれぞれのディスクに対して、ファイルを読み書きするプロセスとなる「OSD(Object Storage Daemon)」が起動します。

またOpenStack Swiftの場合は、リソース名のハッシュ値に対してリングを参照して対応するパーティションを決定しました。一方RADOSでは、リソース名のハッシュ値から「プレースメントグループ」を計算したあと、さらにプレースメントグループに対応するOSDを決定するという2段階構成になります(**図6**)。このとき、プレースメントグループはハッシュ値の単純計算で決まりますが、プレースメントグループをOSDに紐付ける部分では、RADOSに特有の高度な処理が行われます。Cephのオリジナル開発者であるSage Weilが博士論文で発表した「CRUSHアルゴリズム」による計算です[注3]。

Cephの管理者は、まずそれぞれのOSDが管理するディスクがどのストレージノードに搭載されていて、さらに各ストレージノードがどのラックに搭載されていて……という、物理機器の配置情報、さらに複数のレプリカを配置する条件(2個目のレプリカは異なるストレージノードに配置して、3個目のレプリカは異なるラックに配置するなど)を「クラスタマップ」として登録しておきます[注4]。CRUSHアルゴリズムは、クラスタマップを参照して、それぞれのレプリカに対

注3)「CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data」
URL http://www.crss.ucsc.edu/media/papers/weil-sc06.pdf
注4) 正確には、クラスタマップの中に含まれる「CRUSHマップ」という部分になります。

▼図6　RADOSにおけるオブジェクトとOSDの対応付け



して指定された条件にあったOSDを決定します。CRUSHアルゴリズムは非常に高速に計算できるという特徴があります。そのため、リングのように計算済みの配置表をファイルに保存しておくのではなく、配置先のOSDを知る必要がある場合は、その都度CRUSHアルゴリズムによる計算を行います。たとえば、RADOSゲートウェイはCRUSHアルゴリズムで1つめのレプリカを保存するOSDを決定して、オブジェクト(ファイル)を転送します。これを受け取って保存したOSDは再度、CRUSHアルゴリズムで2個目、3個目のレプリカの保存先を決定して、該当のOSDにオブジェクトを再転送します。

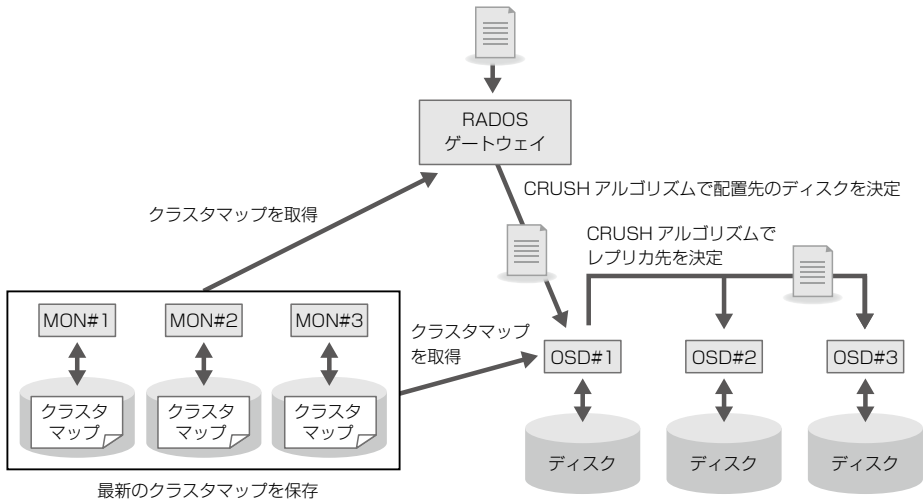
それでは、プレースメントグループとOSDの対応をその都度計算することには、どのようなメリットがあるのでしょうか？　ひとつには、クラスタマップの動的な変更に対応するという目的があります。たとえばCephでは、障害によってストレージノードが停止した際にそれを検知して、自動的にクラスタマップを更新するようになっています。RADOSゲートウェイやOSDは、新しいクラスタマップを参照してCRUSHアルゴリズムの計算を行い、オブジェクトの再配置を自動的に行います。あるいは1つのCeph環境を複数の目的で使用する場合、複数の「プール」を定義して、プールごとにクラスタマップを用意するということができます。これにより、プールごとにデータを保存するストレージノードを分けるといった使い方が実現できます。

クラスタ管理とデータ冗長化のしくみ

RADOSでは、ストレージノードの障害時にクラスタマップを自動更新するという話をしましたが、実際のところ、どのようなしくみで障害検知やクラスタマップの更新が行われるのでしょうか？　このしくみの全体像は**図7**のようになります。Cephの環境では、モニターデーモン「MON」が最新のクラスタマップを管理／保存しています。冗長化のために3台以上のストレージノード上で稼働しており、Paxosと呼ばれるプロトコルで、お互いに同一のクラスタマップを保持していることを確認し合います。そしてRADOSゲートウェイやOSDは、起動時にMONからクラスタマップを取得して、その内容をメモリ上に保持しておき、これを用いてCRUSHアルゴリズムの計算を行います。

さらに、それぞれのOSDはお互いの稼働状況をチェックしており、あるOSDが停止していることを検知すると、その情報をMONに通知します。この通知を受けたMONは、停止したOSDを除いた新しいクラスタマップを作成して、その内容をOSDに配信していきます。クラスタマップにはバージョン番号が振られており、新しいクラスタマップには新しいバージョ

▼図7　MONによるクラスタマップの管理



ン番号が付与されます。MONから新しいクラスタマップを受け取ったOSDは、自身が保持しているオブジェクトについて、新しい配置先をCRUSHアルゴリズムで計算しなおして、必要な際はオブジェクトの移動や再レプリケーションを行います。

このとき新しいクラスタマップは、MONからすべてのOSDに直接配信されるわけではなく、一般に「ゴシッププロトコル」と呼ばれる動作によって、OSD間で順次伝達されていきます。そのためタイミングによっては、OSDごとに参照するクラスタマップが異なる可能性もあります。OSD間でレプリケーションなどの処理を行う際は、お互いにクラスタマップのバージョン番号を確認して、必ず新しいクラスタマップを利用して、矛盾なく処理を行うように設計されています。

なお、**図7**においてRADOSゲートウェイは、クライアントから受け取ったファイルをそのままOSDに転送しているように描かれていますが、実際には巨大なファイルをそのまま転送することは行いません。一定サイズのファイルに分割したうえで、それぞれのファイルに個別のリソース名を付けて、別々のOSDに分散保存するようになっています。分割ファイルのそれぞれについて、3つのレプリカが作成される形になります。またOpenStack Swiftと同様に、「Erasure Coding」を利用することもできます。

まとめ

本パートではOpenStack SwiftとCephを題材として、オブジェクトストレージがファイルを分散保存するしくみを解説しました。配置ルールを事前に生成してリングファイルとして配布するOpenStack Swift、そして、クラスタマップを参照してCRUSHアルゴリズムで配置ルールをその場で計算するCephと、対照的なしくみであることがわかりました。このような配置ルールを管理するしくみこそが、オブジェクトストレージの土台となるわけです。そしてオブジェクトストレージの環境を構築する際は、ストレージノードの台数や構成、利用目的に応じて、リングやクラスタマップを適切に設計するノウハウが必要となります。

次のPart3では、サービス環境で実際に使用されているオブジェクトストレージの事例を通して、このようなノウハウの一端を紹介していきます。SD

part 3

国内・オブジェクトストレージサービス紹介

NTTコミュニケーションズ、IDCフロンティア、GMOクラウドのエンジニアが語るサービスや実装の勘所

オブジェクトストレージサービスは、日本国内でもいろいろな企業が始めています。そのサービスの実装方法には定型パターンがありますが、ユーザ企業にとってのメリットを追求した結果、各社ごとに特徴があります。Part3では、オブジェクトストレージの開発や運用などについて現場のエンジニアが本音も交え紹介します。

[Case1] Cloud^n Object Storage

by 石津 晴崇(いしづ はるたか) NTTコミュニケーションズ㈱ クラウドサービス部 ホスティングサービス部門 第三グループ
Twitter @h_ishizu

冗長化に重点をおいたシステム構成

当社が提供しているCloud^n Object Storageを紹介します。Cloud^n Object Storageではハードウェアはストレージサーバ、ロードバランサ、レイヤ2/3スイッチを利用し、ソフトウェアはクラウディアン社のCloudian HyperStoreを利用しています。また、各機器は冗長化し、専用線で接続された複数のデータセンターに分散して設置しています。

ここで各機器の役割と特徴を紹介します。ストレージサーバはユーザからのリクエスト受け付けと実際のデータを保管しています。このサーバは汎用的なIAサーバですが、ハードディスクを数多く搭載し、ユーザからのリクエストを受け付けるネットワークとサーバ間通信を行うネットワークを分離しています。さらにCloudian HyperStoreはCassandraとRedisを活用し、併用することでストレージサーバを増やし、保管可能な総容量を増やしたり、処理性能を向上させることを可能にしています。

ロードバランサはユーザからのリクエストを受け付けストレージサーバに処理を振り分けています。さらにデータセンターごとに冗長化されて設置されており、GSLB(グローバルサーバロードバラシング)によりデータセンター間の冗長化も行っています。各種スイッチは冗長化に加えてストレージサーバの増加に備えて多段構成としています(**図1**)。

堅牢性・低コストを両立させたサービス構成

Cloud^n Object Storageは、①高い堅牢性、②大容量、③低価格、④Amazon S3とAPI互換、という特徴があります。

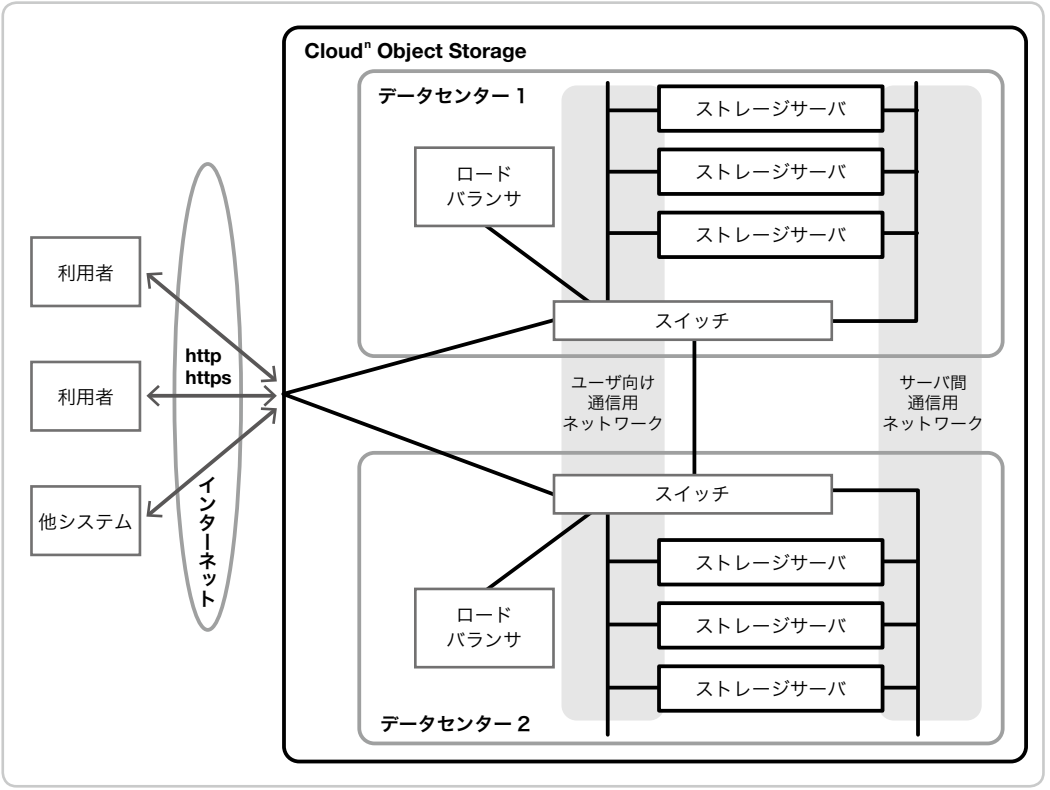
高い堅牢性を実現するために保管するファイルを3冗長化し、データセンターを跨いで異なるストレージサーバに広域分散保存しています。そのため、ストレージサーバの単一故障だけでなく、データセンター設備に故障が発生しても保管しているファイルを取得できます。

大容量を実現するためにはストレージサーバを増やす必要があります。しかし増設されたストレージサーバはシステムに自動的に組み込まれるのでユーザは保管先を変えるといった変更作業をすることなく無制限に保管することができます。

Cloud^{n}のObject Storageは保管容量だけが課金対象のため、ファイルの転送量、リクエスト数には課金されず安価になります。またコストの変動要素が容量だけですのでコストを見積もりやすいというメリットもあります。一方、Cloud^n Computeを利用して仮想サーバを作成し、ファイルをObject Storageに保存すると約10MB/s程度[注1]の速度になり、よく利用されているUSB

注1) 保存する際の性能は参考値であり、性能を保証するものではありません。

▼図1　Cloud^{n} Object Storageシステム構成



メモリと同程度の速度になります。すなわち性能よりも保管コストや保管容量を重要視する使い方に向いています。とくに一度保管したファイルを変更することがないバックアップやインターネットを利用して多数のユーザにファイルを配布する場合が最適です。

一方で、保管するたびにファイルを直接転送する必要があるため、保管しているデータを直接変更して保存しなおすといった利用には向いていません。

最後にObject Storageのインターフェースについて紹介します。公開中のインターフェースはGUIとAPIがあり、簡単な操作と他システムとの容易な連携を可能としています。とくにAPIはAmazon S3と互換性があるため、Amazon S3向けに作られたソフトウェアの中でエンドポイントが変更可能なものがCloud^{n}対応とうたわれていなくても利用できます。

お勧めの利用パターンの紹介

Object Storageに向いている使い方を紹介します。Object Storageに適した利用シーンとしてバックアップ、インターネット向け公開ファイル置き場があります。バックアップする際に最も簡単な使い方は、弊社が提供しているGUIを利用し保管したいファイルを指定するだけです。ほかにはCyberduck[注2]などのクライアントソフトウェアを利用することです。バックアップしたいファイルの指定方法が異なるだけでどちらの方法でも手軽にファイルをObject Storageに保管できます。クライアントソフトウェアについてはCloud^{n}の技術

注2）Cyberduck（URL https://cyberduck.io/）

ブログ[注3]でも数種類を紹介しています。

Object Storageに保管したファイルはインターネット向けに公開できるので、画像ファイル、動画ファイルなどのテキストファイルよりも容量の大きなファイルの保存先とすることでWebサーバのディスク容量を心配する必要がなくなります。

また、写真共有サービスなどでファイルを管理するしくみと保管するしくみが分離できる場合、公開されているAPIを利用してファイル保管先をObject Storageにすることで、保管容量やコストのメリットを得ることができます。たとえばownCloud[注4]といったソフトウェアを利用すればアカウント管理が可能な共有ファイルサーバ[注5]が比較的気軽に構築できますし、Cloud^{n}の各種サービスを利用することで冗長化や負荷分散もできます。

このようにObject Storageは自分がユーザとして利用するだけでなく、他システムと連携させることでB2B、あるいはB2B2Cのサービスの基盤の一部分として利用することができます。

ファイルの安全な保管にお勧め

Cloud^{n}ではこのObject Storageを提供するだけでなく実際に利用しています。具体的にはRDBのバックアップファイルやLoggingのログファイルといった利用者のデータ、Computeのログファイルといった内部データを保管する目的で利用しています。また電子メールで送れないようなファイルを共有する場合にも利用しています。

安心、安全なシステムを構築してサービスを提供したい場合に、Object Storageはファイルを保管するという点において大きなメリットが提供できますので一度利用してみていただきたいと思います。

ここで取り上げた以外にもさまざまな利用法があり、Cloud^{n}の技術ブログ[注6]で紹介していますのでそちらも、ぜひ参考にしてください。

[Case2] IDCフロンティアのオブジェクトストレージサービス

by 佐藤 博之(さとう ひろゆき) ㈱IDCフロンティア 技術開発本部 分散プラットフォームグループ グループリーダー

IDCFクラウドのオプションサービスとして簡単に利用できる

当社のオブジェクトストレージサービス(以下、IDCFオブジェクトストレージ)は2014年4月からサービス提供しています。当社が提供しているIaaSサービス IDCFクラウド(http://www.idcf.jp/cloud/)のオプションサービスとしての位置づけです。コントロールパネルが標準で提供されており、コントロールパネルから申し込み、オブジェクトストレージの利用が簡単にできるようになっています。

IDCFオブジェクトストレージは、一般的なオブジェクトストレージサービスの要件である、「データ容量の制限がない」、「高い堅牢性」、「REST APIでの接続可能」以外に、次の特徴が挙げられます。

1. インターネットを介さないセキュアな接続が可能
2. サービス間トラフィック課金が無料
3. 従量、定額(10TBパック)の選べる料金プラン

1、2について本稿で説明します。

3についてはIDCFオブジェクトストレージ

注3) Object StorageをGUIクライアントから簡単に利用する(URL http://www.cloudn-service.com/blog/?p=466)
注4) ownCloud(URL https://owncloud.org/)
注5) Cloudn上にownCloudを構築する(URL http://www.cloudn-service.com/blog/?p=1385)
注6) Cloud^{n}技術ブログ(URL http://www.cloudn-service.com/blog/)

の紹介ページ[注7]を参照ください。

インターネットを介さないセキュアな接続が可能

IDCFオブジェクトストレージへの接続はHTTP/HTTPSで行います。インターネットからの接続の他、閉域網経由での接続もできます。機密性の高いデータをオブジェクトストレージに保存する際にインターネットを経由することがなく、安全にデータ転送が可能です(**図2**)。

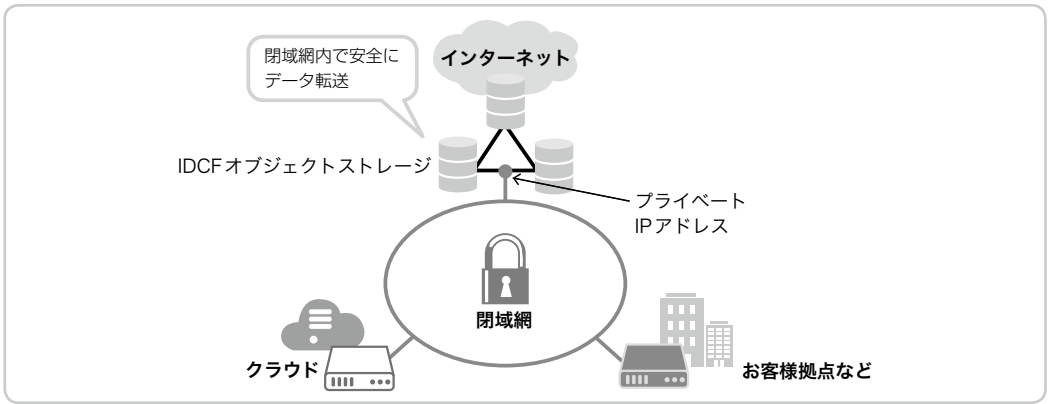
サービス間データ転送量課金が無料

IDCFオブジェクトストレージの課金対象は、オブジェクトストレージに保存したデータ容量とオブジェクトストレージからのデータ転送量(OUTトラフィック)になります。ただし、IDCフロンティアのデータセンターにハウジングしているサーバやIDCFクラウドなどのクラウド上に構築した仮想サーバとのデータ転送量は無料になります(**図3**)。つまり、**図3**の場合は、データセンター内のサーバそしてクラウド上の仮想サーバとオブジェクトストレージ間のデータ転送量が無料になります。

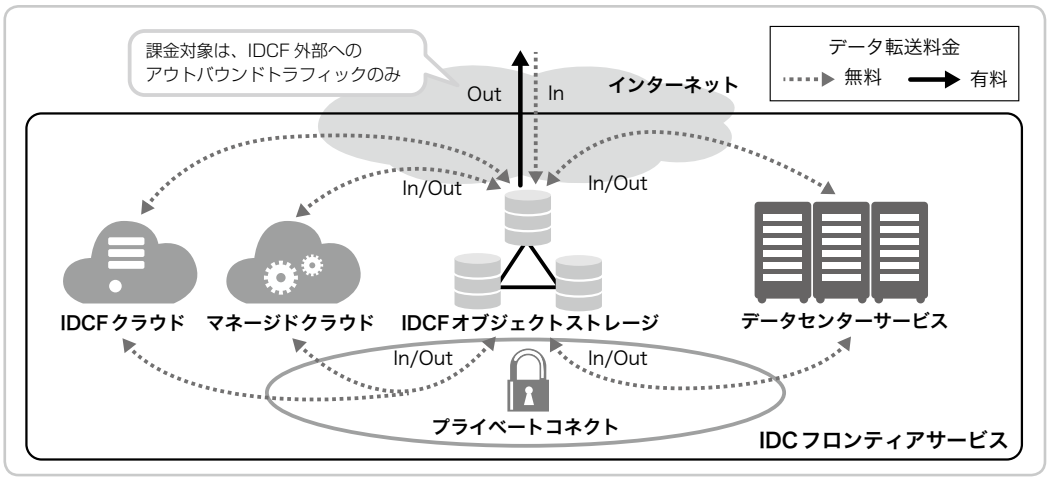
IDCFオブジェクトストレージの適合分野

IDCFオブジェクトストレージの適合分野は

▼図2　閉域網内で安全なデータ転送(※閉域網経由での利用には、別途プライベートコネクトの契約が必要)



▼図3　サービス間データ転送量課金が無料



注7) URL http://www.idcf.jp/cloud/storage/spec.html

図4のとおりです。ファイル数の制限を考慮することなく、安価に大容量を利用できることに加え、堅牢性が高いため、ログやアーカイブを長期保管することに適したストレージとなっています。また、動画や画像ファイルを配信するストレージとしての利用にも向いています。一方で、オブジェクトストレージへはHTTP/HTTPSを使ってインターネット経由で接続するため、高いIOPSを求められるストレージとしての利用は不向きとなっています。

IDCFオブジェクトストレージの利用用途

次のように大きく3つの用途で使われています。

1. データのバックアップ先としての利用
2. データ配信サーバとしての利用
3. Hadoop基盤としての利用

データのバックアップ先としての利用

ストレージの利用として最も多いのが、データのバックアップ先として利用するケースです。

IDCFオブジェクトストレージはAmazon S3 API互換となっており、Amazon S3で使えるアプリケーションはほぼそのままで利用可能です。たとえば、s3cmdやs3syncを使ってログやアーカイブデータの保存が可能です。

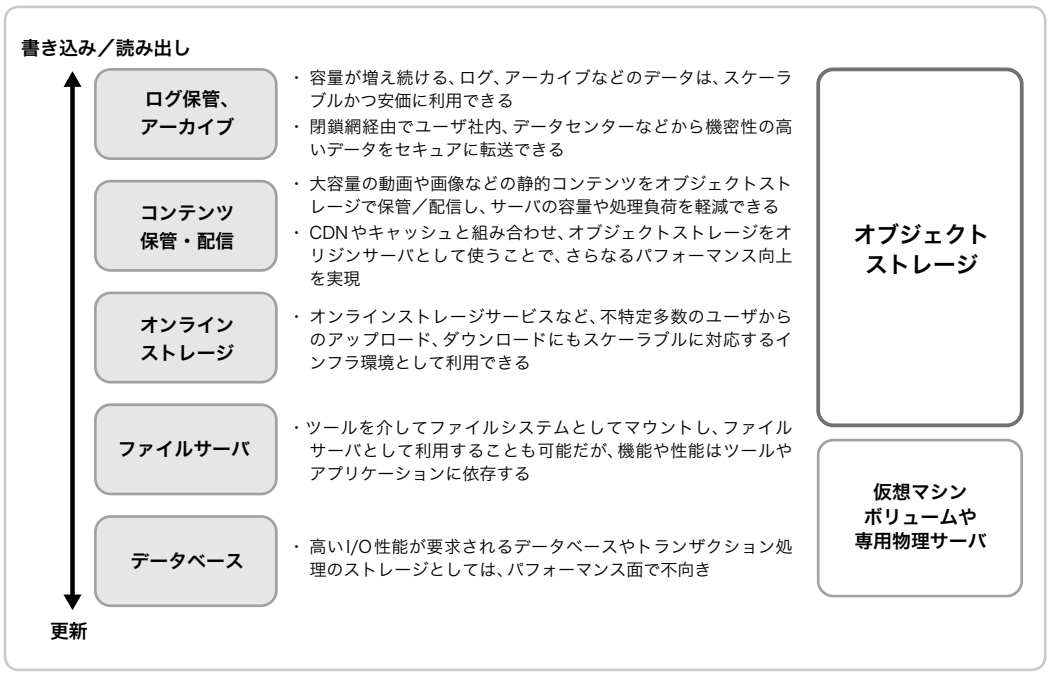
s3cmdを使ってIDCFオブジェクトストレージへデータをアップロードする例を次に示します。はじめにバケットを作成します。バケットはデータを保存する箱のようなもので、フォルダのような概念になります。次ではバケット「sample」を作成します。

```
$ s3cmd mb s3://sample/
Bucket 's3://sample/' created
```

次に、作成したバケット「sample」に保存したいログファイルをアップロードします。次では「hoge.log」をアップロードします。

```
$ s3cmd put hoge.log s3://sample/
hoge.log -> s3://sample/hoge.log  [1 of 1]
 5 of 5   100% in    0s     9.30 B/s  done
```

▼図4 IDCFオブジェクトストレージに適した使い方



ログファイルがIDCFオブジェクトストレージにアップロードされたかを確認します。

```
$ s3cmd ls s3://sample/
2016-02-10 13:20         5   s3://sample/hoge.log
```

MySQLのdumpデータをmysqldumpとs3cmdでIDCFオブジェクトストレージへ保存する方法をIDCフロンティアのエンジニアブログ[注8]で紹介しています。

またログ転送を行うオープンソフトウェアfluentdを使ってIDCFオブジェクトストレージにApacheのアクセスログを保存する方法[注9]も紹介しております。ぜひ参考にしてみてください。

データ配信サーバとしての利用

IDCFオブジェクトストレージはオブジェクトストレージ内に保存したデータをHTTP/HTTPSで公開できます。たとえば上記でアップロードした「hoge.log」をHTTP/HTTPSで公開するには、

```
$ s3cmd setacl --acl-public s3://sample/hoge.log
s3://sample/hoge.log: ACL set to Public  [1 of 1]
```

とACLを変更するだけです。同じ要領で、画像や動画をオブジェクトストレージに保存し、簡単に公開することが可能になっています。

公開されたファイルには次のような形式でアクセス可能です。

```
http://バケット名.IDCFオブジェクトストレージエンドポイント/ファイル名
```

IDCFオブジェクトストレージにはECサイトの画像、動画やゲーム会社のゲームコンテンツが保存され、日々配信用に使われています。また、弊社のCDNサービスであるコンテンツキャッシュサービス[注10]のオリジンサーバとしてIDCFオブジェクトストレージを利用し、CDNからオブジェクトストレージ内のデータを高速に配信することも可能です。

Hadoop基盤としての利用

IDCFオブジェクトストレージはIDCフロンティアが提供するビックデータ分析基盤であるYahoo!ビックデータインサイトのHadoop基盤[注11]として利用されています。Yahoo!ビックデータインサイトはビックデータの収集・保存・分析をワンストップで提供するクラウド型のデータマネージメントサービスです。

Yahoo!ビックデータインサイトではREST APIを使ってIDCFオブジェクトストレージにデータを蓄積し、データ解析のために利用しています。

◆ ◆ ◆

上記で挙げた3つの用途以外でも、例えばネットプリントサービスの写真の保管やオンラインストレージサービスとしての利用など、さまざまな用途で使われています。

今後IoTが普及し、爆発的にデータ量が増えていく上で、容量／ファイル数に制限のないオブジェクトストレージはますますニーズが高まると考えています。

IDCFオブジェクトストレージのシステム構成

IDCFオブジェクトストレージは大容量ディスクを多数搭載したIAサーバを使って構成しています。サーバ台数は数百大規模となっており、データ容量、性能状況に応じてサーバを追加しています。オブジェクトストレージは、サーバを追

注8） URL http://blog.idcf.jp/entry/storage/mysqldump/
注9） URL http://blog.idcf.jp/entry/storage/log_fluentd/
注10） URL http://www.idcf.jp/cloud/cache/
注11） URL http://www.idcf.jp/bigdata/

加することで、全体容量が増え、また性能を向上させることができます。サーバ追加時はオブジェクトストレージ内のデータの再配置が行われますが、サービスを停止することなく行われます。また、サーバやディスク故障時についてもサービス停止することなく、メンテナンスを行えるような設計になっています。

IDCFオブジェクトストレージの堅牢性

IDCFオブジェクトストレージのデータ堅牢性はサービス仕様上で99.999999999%(11ナイン)となっており、極めて安全性の高いストレージとなっています。この堅牢性を実現するために、お客様のデータは異なるサーバに3重コピーしています。さらにデータの完全性が定期的に確認され、データの不整合が発生した場合は自動的に修復するしくみが備わっています。

IDCFオブジェクトストレージのリソース監視(Ganglia、Grafana)

IDCFオブジェクトストレージではリソース監視ツールとしてGangliaとGrafanaを利用しています。

Gangliaは大規模システムのリソース状況を可視化するのに優れたツールとなっており、IAサーバを大量に扱うオブジェクトストレージに向いている監視ツールになっています。Gangliaを使って、サーバのCPU、メモリ、ディスクなどのOSリソース状況の他、オブジェクトストレージで使われているKVS(Key Value Store)のリソース状況などを含め、50以上のメトリックを監視しています(**図5**)。

またGrafanaでIDCFオブジェクトストレージのPUT/GET性能値、リクエスト数、エラー数などをリアルタイムで可視化しています(**図6**)。

▼図5　Gangliaで個々のサーバのメトリックを積み上げグラフとして確認することもできるため、クラスタ単位のリソース状況も一目でわかる

▼図6　Grafanaによるデータの可視化

GrafanaはInfluxDBに投入したデータをグラフ化できることから、あらゆる数値をInfluxDBに投入し、Grafanaで可視化することで、障害予兆や性能低下をいち早く検知できるようにしています。

◆◆◆

IDCフロンティアでは「データ集積地構想」の名のもと、データを集め、データやサービスが有機的に結合することで、お客様の新しい価値を作り出すことを目指しています。IDCFオブジェクトストレージサービスはその構想を担う中核のストレージとして、機能追加、品質向上に努めています。

[Case3] GMOクラウドオブジェクトストレージ

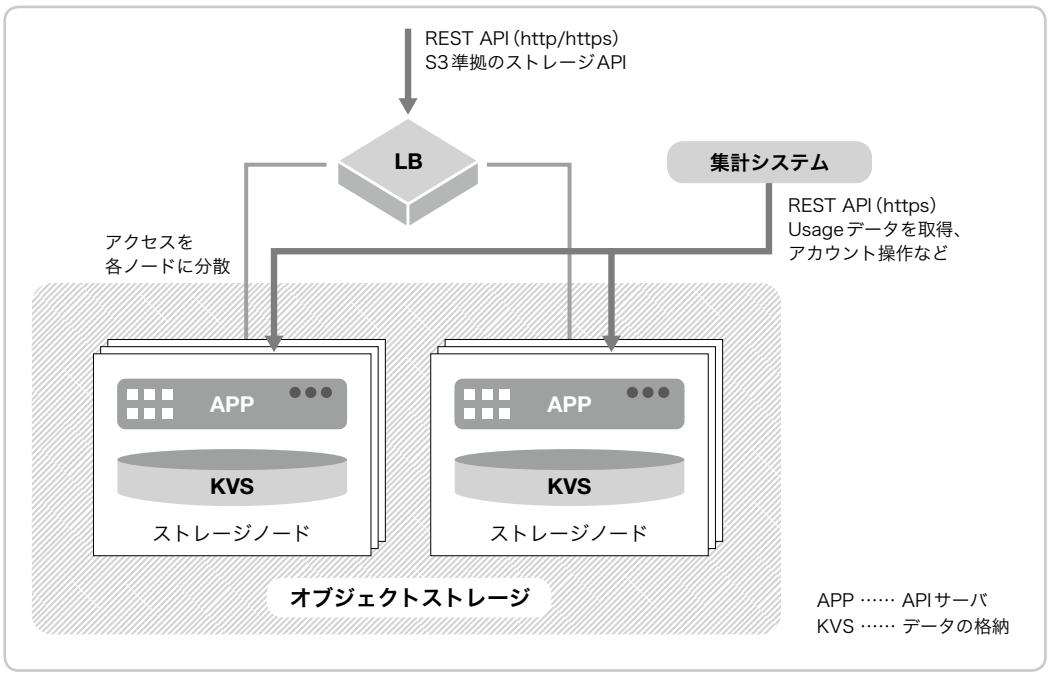
by 片柳 勇人(かたやなぎ はやと)　GMOクラウド㈱ サービス運用部

GMOクラウドALTUSオブジェクトストレージの全体構成

KVSとしてRiak CSを採用したシンプル構成

ストレージサービスの基幹部分にあたるオブジェクトストレージのアプリケーションをどうするかが最も重要だと考えますが、当社では可用性、拡張性を重視してRiakCS[注12]を採用しています。これにより**図7**のようにシンプルな構成でシステムを構築することが可能でした。当社システム内のオブジェクトストレージは、アプリケーションサーバがS3準拠のクラウドストレージAPIを提供し、PUTされたデータは3複製された状態でKVSに格納されます。KVSはクラスタ化されており、システムのリ

▼図7　GMOクラウドオブジェクトストレージの全体構成



注12) 東京エレクトロンデバイス株式会社から「Riak S2, Riak KV Enterprise」という名称で販売、サポートされており、当社サービスが事例紹介として掲載されています(URL http://cn.teldevice.co.jp/case/detail/gmo_cloud2)。

ソース・パフォーマンスが逼迫(ひっぱく)した際にはストレージノードを追加することで容易にシステムの拡張が可能となっています。

GMOクラウドALTUS オブジェクトストレージの特徴

クラウドストレージサービスとして「GMOクラウドALTUS(アルタス)オブジェクトストレージ」を提供しています。パブリック向けのストレージサービスは、お客様の重要なデータをお預かりする性質上データの堅牢性が求められる一方で、そのストレージ利用量の予測が立てにくいという問題があります。そこで、当社は本システムを構築する上で下記に重点を置いています。スタートアップから中小企業まで幅広く利用いただいているパブリッククラウドサービスとなっています。

- ストレージサービスとして高い可用性を提供できること
- スモールスタートでき、ストレージ容量の拡張性があること
- 管理・運用が容易なこと

GMOクラウド Simplemailへの応用

「GMOクラウド Simplemail」大容量ファイル添付メールゲートウェイサービスでのオブジェクトストレージ活用を紹介します。

Simplemailの生い立ち

オブジェクトを利用したソリューション

オブジェクトストレージの大きなアドバンテージとして、ビットあたりのコスト(ビット単価)が一般のストレージシステムより安価であることが挙げられます。またオブジェクトストレージのシステムとして、基本的にデータの二重、三重のレプリケーション(複製)が行われるので、ストレージに保存したデータの保全性が高いことも長所として挙げられます。

長所があれば短所もあり、検索性や高速性を追求するアプリケーションには向きません。このようなオブジェクトストレージの特性上、高速アクセスや検索性はさほど必要ではなく、長期にわたり大量のデータを安全に格納する分野のアプリケーションによく利用されます。すぐに思い付くのはログやバックアップデータの保管、クラウド事業を営んでいる弊社の場合では、VM作成用のテンプレートファイルや、VMのスナップショットなど、サイズが大きくて、一度作成されればその後変更されることがない物が代表的になります。

クラウドストレージでは不便

同様のアプリケーションとして、BOXやDropbox、Amazon CloudDrive、Google Drive、Microsoft OneDriveなどに代表されるクラウドストレージと呼ばれる一群がありますが、自分用のファイルをこれらクラウドストレージに保管して利用するぶんには申し分ないシステムですが、これを利用して他の人にファイルを送りたいような場面ではいかがでしょうか？

共有設定をするためには、先方も同じクラウドストレージシステムにアカウントが無ければいけませんし、ファイルの授受が完了したら共有を解除しておかないと、情報漏えいにつながるセキュリティホールになりかねません。

AmazonS3を「素」でお使いの方であれば、S3上にファイルを保存して、そのファイルを示すURLを先方に通知する方法をご存じかもしれませんが、その際にはファイルのACLを設定したり、先方がダウンロードしたのを確認してファイルを削除するなどの作業が必要となります。

また授受するファイルの機密性が高い場合は、そのファイルをダウンロードしたのが、本当に送りたい相手だけだったのか、URLを盗まれて別の人にもダウンロードされてしまったのか判別が難しいことも難点です。

おそらく多くの人がこのような問題を認識しており、その結果として宅ふぁいる便のような

システムが広く使われているのが現状だと言えます。

メールから直接オブジェクトへ

弊社ではこのような状況を鑑み、直接メールからオブジェクトストレージへファイルを保管し、受け手の方に安全・確実にファイルを届けられるシステムを昨年開発し Simplemail として提供を開始しました。Simplemail を開発するときに心がけたのは次の点です。

利用者に簡単であること

実際のシステム利用者の方は、普段メールにファイルを添付して送るという作業に慣れている方、別の言い方をすれば、クラウドストレージはおろか、宅ファイル便の存在もご存じない方が、いかに今までと同じ作業でシステムを利用できるかにこだわりました。

運用者にやさしいこと

導入にあたって、メールシステムの運用者の方の労力が極力少なくなるようにシステム構成を考えました。メールサーバの設定変更が最小限で済むように、システム構成はメールのリレーサーバの形でお客様のメールサーバと送信先のメールサーバの間に割って入ります。このため利用するにあたって必要な設定変更は、SMTP ルートの変更(リレーサーバの設定)だけとなります。

余談ですが、筆者がメールサーバの管理をしていたころは、まだMTAとしてはsendmailしかない時代で、当時はsendmailの設定ファイルであるsendmail.cfの変更・修正は「一子相伝」などと揶揄されていました。今ではm4やsendmail.mcのお陰でずいぶんわかりやすくなっていますが、MTAの設定変更は、筆者にはある意味トラウマに近い感覚があります。

メールサーバを選ばないこと

現在のメール環境は昔に比べてはるかに多様化されています。前述のsendmailだけでなく、Postfix、QMail、Exim4などのメールサーバだけではなく、GmailやOffice365などのクラウドメールシステムを利用している方も相当数にのぼります。これらのシステムをご利用の方々にも、問題なく利用できるシステムでないと提供する価値がないと考えました。

落ちないこと

今日、メールシステムはビジネスの根幹を支えていると言っても過言ではないと言えます。お客様の大切なメールトラフィックを中継する以上、システムダウンは絶対に許されません。

このため、サービスを提供するサーバ群は三重化、ネットワークは完全二重化したシステムを構築しています。サーバを多重化しても、単一障害点(SPF：Single Point of Failure)で全体が機能しなくなっては意味がないので、そのようなSingle Pointがないように全体を構成しています。

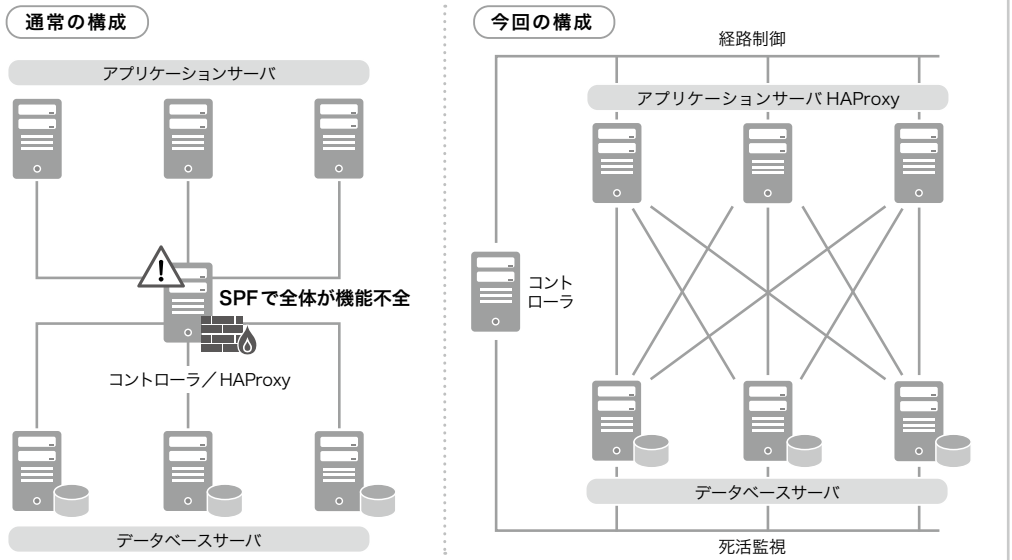
とくにデータベースへのアクセス障害は致命的ですので、データベースはMySQL HAで多重化しつつ、アプリケーションサーバからのアクセス経路は**図8**のように構成してSingle Pointを作らないように工夫しています。

さらに各ソフトウェアモジュールも多重化し、モジュール間は、Message Queueを用いた疎結合として、単一モジュールに障害が発生しても、その影響がシステム全体に広がる可能性を低くしています。また肝心のオブジェクトストレージへの経路も、経路が違う2種類の回線で接続しています。

安全であること

このようなサービスを提供する以上、当然ですがセキュリティ上のリスクを下げる実装を行っています。ただし高いセキュリティと使いやすさは、相反する面がありますので、ご利用者の方がオブジェクトからファイルをダウンロードする際のパスワードの長さや、パスワードに使

▼図8　単一障害点を作らないためのメールサーバ構成



われる文字種はお客さま自身で設定してもらうようになっています。

システム構成

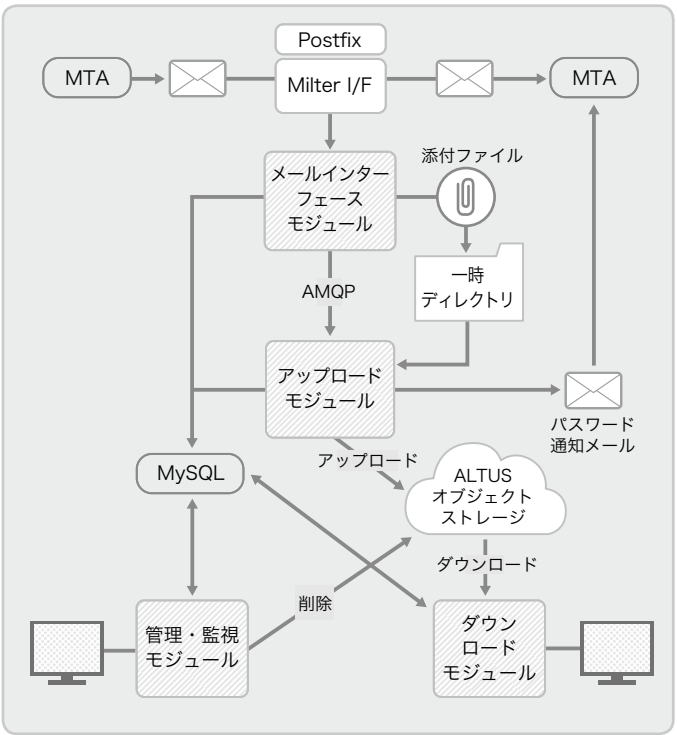
Simplemailのシステム(**図9**)は、大きく次のモジュールに分かれています。

・管理／監視
・メールインターフェース
・オブジェクトアップローダー
・オブジェクトダウンローダー

管理／監視系

システム全体で使用できるオブジェクトストレージの容量や、各お客様がご利用できるオブジェクトストレージの容量管理や、容量を超過しそうな場合にアラートメールを送信するなどSimplemail全体の動きをコントロールしています。お客様のアカウントごとの設定(生成するパスワードの複雑さや、パスワード通知メールの文言など)

▼図9　Simplemailのシステム



管理もここで行っています。また、保存期間が過ぎた添付ファイルのオブジェクトストレージからの削除や、各処理のロギングもこのモジュールで行われています。

メールインターフェース

メールインターフェースの部分は、Milterとして機能しています。MTAにPostfixを設置して受け取ったメールをMilterとして処理します。添付されたファイルを処理する前に、システムに登録されたドメインから来たものか、処理して良いメールなのかを判断して、処理するべきメールであれば添付ファイルを分離して一時ディレクトリに保存、メッセージ・キューを通じてオブジェクトアップローダーへ通知します。

オブジェクトアップローダー

メールインターフェースから通知を受け、当該ファイルをオブジェクトストレージにアップロードします。アップロードと同時にダウンロードに必要なパスワードの生成を行い、それを受信者へ送信します。

オブジェクトダウンローダー

メール受信者が添付ファイルをダウンロードするインターフェースです。

入力された受信者メールアドレスと、ダウンロードパスワードの確認を行ったあと、ファイルの最大ダウンロード回数が指定された値を超えていなければ、オブジェクトストレージからのダウンロードを許可します。ダウンロードはオブジェクトストレージから直接ではなく、本モジュールがいったん終端して中継する形をとります。

この形を取ることで、ダウンロード回数の制限や、ダウンロードできるファイルが複数有った場合、ZIPファイルにまとめて1回のダウンロードで済むようになっています。

将来的には、ここにアンチウィルス機能を組込み、ダウンロードしようとしているファイルがウィルスに感染している場合、警告を出す・ダウンロードをさせないなどのオプションも追加する計画です。

はまったところ

開発が完了し最終試験の工程で、巨大なファイルを添付した場合、ときどきオブジェクトストレージへのアップロードに失敗することがあり、原因の特定に時間がかかりました。同じサイズのファイルを分割して、1メールに複数ファイルの添付の形にすると問題は発生せず、どうやらオブジェクトストレージにアップロードするときのチャンクサイズがかかわっていると目星を付けて調査していくと、max_siblingsというパラメータで、オブジェクトストレージ全体でアップロード時に張れるセッション数が制御されていることがわかりました。

このパラメータはオブジェクトストレージのノード数と密接な関係があり、むやみに増やすと最悪の場合オブジェクトストレージのノードダウンを引き起こします。

とりあえず安全値最大まで引き上げ、前述のオブジェクトアップローダーモジュールがファイルをアップロードする際のチャンクサイズを大きくして、セッションの全体数を減らすことで対応しました。弊社の場合オブジェクトストレージを自社で運用しているので、今回オブジェクトストレージの設定そのものを変更することができましたが、他社のオブジェクトストレージを利用したサービスを利用する場合には、このようなところが問題になるかもしれません。

おわりに

オブジェクトストレージは速度を要求されないアプリケーションであれば、信頼性・大容量などまだまだ使いどころがあるサービスだと思います。読者の皆さんの「こんなことにも使えるのでは？」という声を、ぜひ私どもにお知らせください。SD

その1

ネットワーク／サーバエンジニアに求められる 光ファイバの知識

Author 佐伯 尊子(さえき たかこ)　(株)ブロードバンドタワー

はじめに

本誌2016年2月号の第2特集では、「適切なLANケーブリングの教科書」と題してツイストペアケーブルによる配線の解説を行いました。本章はその続きとして、光ファイバを中心に解説します。

光ファイバとコネクタ

光ファイバの規格

光ファイバは、大きく分けて2種類あります。MMF(マルチモードファイバ)とSMF(シングルモードファイバ)です(**表1**)。

具体的に光ファイバを使ったシステムについて考えてみましょう。MMFは近距離で安価なシステムに最適、SMFは近距離から長距離まで幅広くシステムを構築できますが、高コストです(**表2**)。近年高速大容量通信を必要とするデータセンターでは、MMFを用いたシステムが増えてきました。

▼表2　MMFシステムとSMFシステムの違い

項目	MMFシステム	SMFシステム
伝送距離	速度にもよるが、100m～2kmくらいまで	ラック間から国際海底ケーブルの長さまで多種多様に対応可能
システムコスト	ツイストペアと比べると高価だが、SMFシステムと比べると安価	高価だが、いろいろな使い方が可能
消費電力	SMFと比べると少ない	MMFより多い
利用光源	LEDもしくはVCSELレーザ	LD

▼表1　TIA-568(光ファイバ抜粋)

MMF(マルチモードファイバ)種類	単位：μm		単位：MHz・km			単位：dB/km	
			帯域		VCSELレーザによる帯域	損失	
	コア径	クラッド径	波長850nm	波長1,300nm	波長850nm	波長850nm	波長1,300nm
OM1	62.5	125	200	500	—	3.5	1.5
OM2	50	125	500	500	—	3.5	1.5
OM3	50	125	1,500	500	2,000	3.5	1.5
OM4	50	125	3,500	500	4,700	2.5	0.8

SMF(シングルモードファイバ)種類	単位：μm		単位：dB/km	
			損失	
	MF径	クラッド径	波長1,310nm	波長1,550nm
OS1	9	125	1	1
OS2	9	125	0.4	0.4

光ファイバの判別

次にそれぞれの光ファイバの見分け方について説明します。新しく購入する際は明らかに「MMF」なのか「SMF」なのか判別して発注するため、明確に分けることができますが、すでに在庫もしくは利用中の光ファイバについては、光ケーブルの表面の文字(表面印刷)で判別します。そこには「MMFかSMFか？」「どんな種類か？」などの情報が盛り込まれていることが多いです。ケーブル表面の色から判断する方もいらっしゃるようですが、表面色はいろいろあって、必ずしもMMFやSMFを判別しているわけではありません(**表3**)。必ず表面印刷を確認しましょう。そして、保管の際には長さやコネクタ種類だけで分けるだけでなく、光ファイバの種類もきちんと分けて保管しましょう。

▼表3　光ファイバコードの主な表面色

光ファイバ種類	日本	北米(TIA-598)
OM1	若草	黒、灰、ベージュ、橙
OM2	若草	黒、灰、ベージュ、橙
OM3	若草、空色	空色
OM4	空色	空色、紫
OS1	黄色	黄色、青
OS2	黄色	黄色、青

コネクタの種類

次に、光ファイバで利用されるコネクタについて確認します。光ファイバコネクタとは、光ファイバ同士を接続する場合、コネクタ先端(フェルール)を突き当てて接続するために用いるものです。**図1**に光LANでおもに利用しているコネクタを示します。

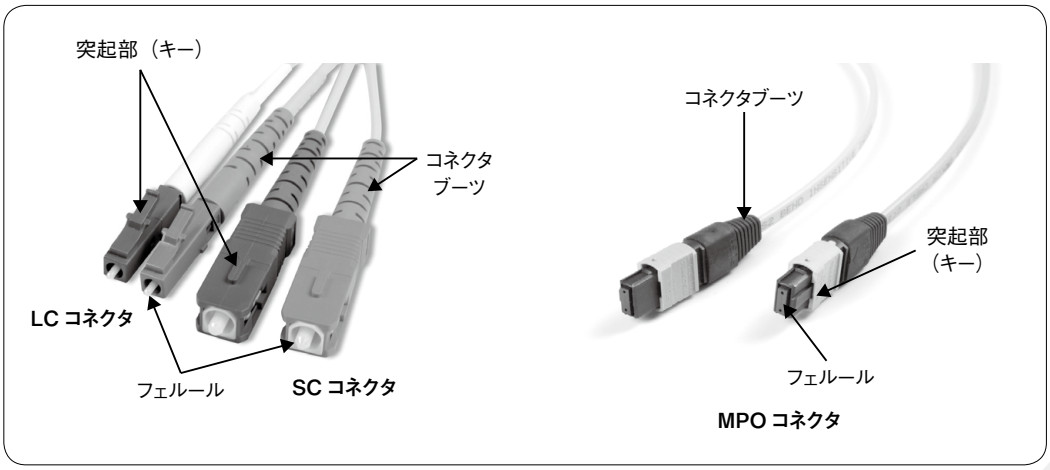
ツイストペアケーブルのコネクタはRJ-45一種類でしたが、光ファイバはコネクタが3種類(以上)あります。光ファイバはツイストペアのように一意に「ケーブル」と「コネクタ」が定まらないところに難しさがあります。

光コネクタ研磨方法

同じコネクタであっても、さらに種類があります。それはフェルールの研磨方法の違いです(**図2**)。研磨方法によって突き当てたときの接続部の反射減衰量が変化します。この数字は大きければ大きいほど、接続点での光の漏れと光の逆戻りを防ぐため、コネクタの特性が良くなります。接続するコネクタの研磨方法が、接続する左右のフェルールで異なっている場合、下位互換となるため、その接続点における反射減衰量は小さい方に依存します。

1Gbps位までは、この反射減衰量は気になら

▼図1　光コネクタ種類と名称[1]



なかったのですが、10GbE以上のシステムではこの反射減衰量を考慮しない線路(機器間すべてのケーブルや、コネクタの意味)の場合、回線が不安定になりやすいことが知られています。さらに、SCコネクタやLCコネクタは光ファイバ1芯につき1つのフェルールから成っていますが、MPO多芯コネクタを使う機会が増えてきました。今は明示されていませんが、将来、多芯コネクタも単芯同様反射減衰量について規定が設けられることが予想されます。

光モジュール

光コネクタの形状は、接続する光モジュールやパッチパネルの構造に依存します。Ethernetで利用する光モジュールと、光コネクタの関係について、**表4**に示します。

いくつかパターンがありますが、利用する光モジュールの形状によって機器と接続するコネクタは一意で決まります。そのため、どの光ファイバやコネクタを利用するかは、必ず光モジュールの光ファイバ挿し込み口を確認しましょう。

▼表4　光モジュールと光コネクタ

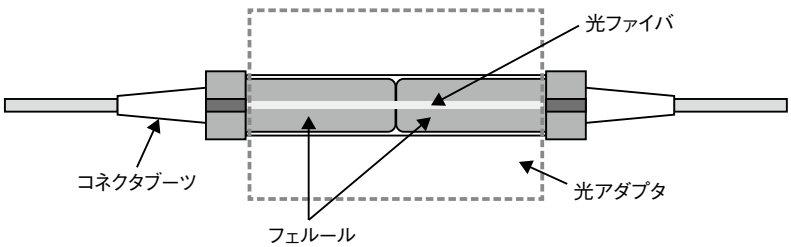
通信速度	光モジュール	コネクタ	利用光ファイバ
1000MbE	GBIC	SC2芯	MMF・SMF
	SFP	LC2芯	MMF・SMF
10GbE	XENPAK	SC2芯	MMF・SMF
10GbE	SFP+	LC2芯	MMF・SMF
40GbE	CFP、CFP2、QSFP+	MPO8芯	MMF
		LC2芯	SMF
100GbE	CFP、CFP2、CFP4	MPO10芯×2	MMF
		LC2芯	SMF
	QSFP28	LC2芯	SMF

光ファイバの構造化配線

ツイストペアケーブル同様に光ファイバの場合の構造化配線についても確認します。10GbEまではMMFシステムであれSMFシステムであれ、Ethernet通信では必ず2芯を使って配線します。1芯が送信、1芯が受信です。これは、同じ光ファイバを用いていますが1芯しか使わないFTTH(Fiber To The Home)配線と大きく異なる点です。

「光ファイバは髪の毛ほど細いので、1芯で大容量が伝送できます。」という話は、通信事業者の光ファイバサービスの話であって、構内で利用するEthernetとは考え方が違います。同じ光ファイバを使うのに、違和感があるかもしれませんが、LANが2芯を利用するのは、ケーブルを配線するスペースが通信事業者と比べ、余裕があること。配線距離が短

▼図2　コネクタ接続とフェルール研磨方法



種類	フラット研磨	PC研磨	AdPC研磨	APC研磨
接続損失[dB]	0.7	0.5	0.5	0.5
反射減衰量[dB]	規格無し	22以上	40以上	60以上
形状				

いこと。送信／受信で分けた方がシステム全体のコストは通信事業者のそれと比べ安価に構築できること。などから2芯を利用しています。最近は一部1芯で送受信できる光モジュールも販売されていますが、IEEEの規格外のため必ずそのメーカーが推奨する製品を利用してください。

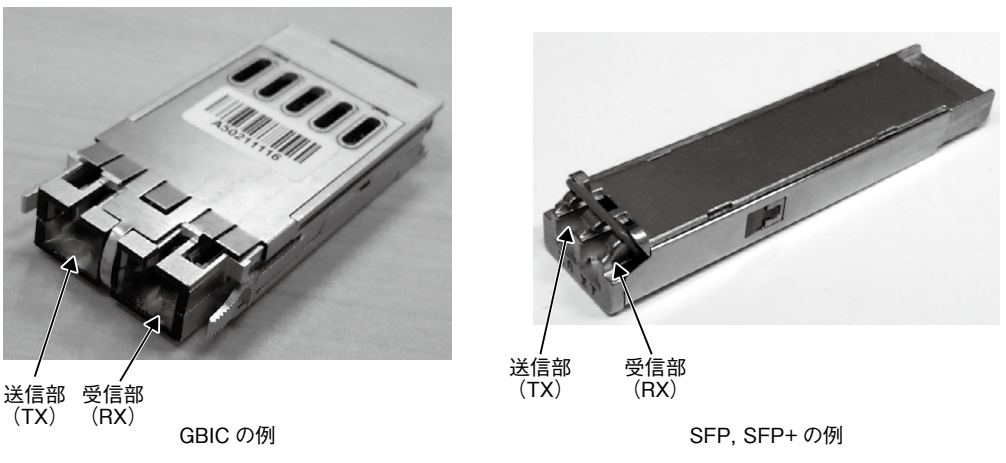
それでは、光ファイバのEthernetは、どのような配線になるか見てみましょう。まず、**図3**に示すように、光モジュール側の決まりを確認します。実はコネクタに付いている突起というのは、その突起の右か左かを示す重要な手掛かりになります。機器に挿入したとき、突起が上(キーアップ)になるか、下(キーダウン)になるか？　キーアップであれば、左が送信、右が受信になります。キーダウンであれば、左が受信、右が送信になります。この決まりは、LCコネクタであれ、SCコネクタであれ共通です。すなわち、GBICとSFP、XENPAKとSFP+、それぞれ形状は異なりますが、キー溝に対する送信と受信は同じ配列になっているのです。

次に、この送信と受信はどのように接続されるのかを確認します。2016年2月号の特集では、パーマネントリンクとチャネルの話をしましたが、光ファイバであってもツイストペアケーブルと同様にパーマネントリンクとチャネルが存在します(**図4**)。

極性の確認

先ほど説明したように、コネクタのキー溝をキーにして、送信と受信が分かれます。これを光配線の「極性」と呼んでいます。光ファイバの構造化配線の場合、気を付けて配線しないと、送信同士、もしくは受信同士がぶつかってしまいます。規格では全てのリンクをクロスで配線するように定義しています。しかし日本の場合、パーマネントリンク部分はストレートで配線し、機器コードの部分だけクロス配線を使う配線も多く存在します。また、幹線ケーブルはストレートであっても水平配線にクロス配線を用いたり、全てストレート配線にしたりと、クロス配線部分とストレート配線部分が混在していることが多く、機器間の接続を行う場合は、2芯一括の光コネクタを使うと、思わぬところで送信同士／受信同士がぶつかることがあります(**図4**参照)。LAN配線の固まりであるデータセンターであっても、データセンター毎にキー溝と極性がバラバラなことが多いため、利用開始前に光ファイバの極性について確認しておく必要があります。

▼図3　単芯光モジュールの送信／受信の例



光ファイバの品質

光ファイバの品質についても、確認しましょう。

購入

ツイストペアケーブルと比べ、専門メーカーから購入することがほとんどかと思います。品質に関しては各社ばらつきが少なく、規格どおりの製品が流通しているかと思います。ただ、光ファイバのグレードに関しては、**表1**に示す規格よりも性能が低いものしかないというケースがあります(数字が小さいほど性能が低い)。購入の際には、規格をよく確認の上発注してください。また、コネクタ端面の研磨方法についても、同様によく確認の上発注してください。

測定

ツイストペアケーブルは、たくさんの測定項目がありますが、光ファイバは「光損失」1つだけが測定項目となります。そのため、現場で比較的手軽に測定ができると思われがちです。しかし、適切な値を得るためには、測定用コードの用意や基準値の取り方など、原理の十分な理解と慣れが必要になります。まずは、**図5**に示す光源とパワーメータを用意すること、適切な波長を選択して測定することなど、初歩的な部分をしっかり押さえましょう。

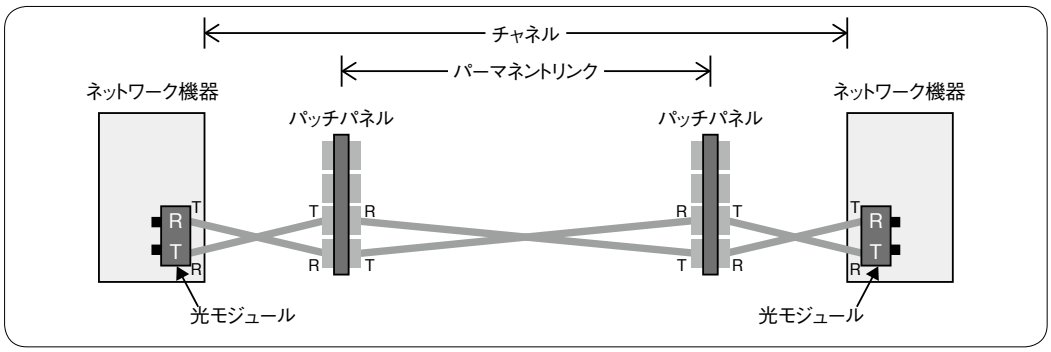
端面の清掃と、光源の危険性

そして、もう1つ忘れてはいけないのは、光コネクタ端面(フェルール先端)の清掃です。**図6**に示すように、フェルール先端を顕微鏡で覗(のぞ)くと、たとえ工場出荷時であっても汚れています。さらに現場で一度コネクタのキャップを取り外したら、汚れが必ず付着しています。そのため、光ファイバコネクタを用いて接続する際には、必ず専用のクリーナーでフェルール先端を拭いてから接続します。また清掃時、フェルール先端や、パッチパネルを覗いてしまいがちですが、通信に利用する波長は近赤外光で、可視光ではありません。また高速伝送になればなるほどMMFもSMFも大容量レーザを用いますので出力パワーが大きくなっています。そのため、知らぬ間に眼球を傷つけることがあるかもしれません。そのため、絶対に覗き込まないことで

▼図5　光損失測定(光源とパワーメータ) [2]

光源例　　パワーメータ例

▼図4　光ファイバによる構造化配線



す。また、パッチパネルやコードのキャップは利用する直前まで必ず蓋(ふた)をして、無意識であっても光が目に入らないように注意するなど、作業時に工夫が必要になります。

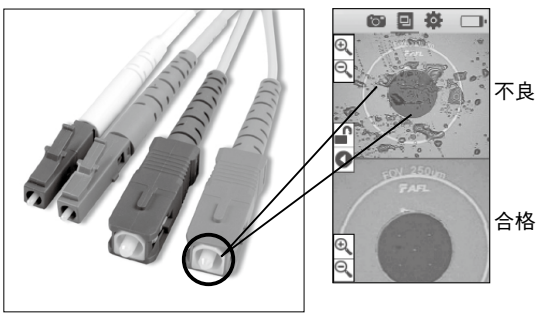
光ファイバ配線の注意事項

光ファイバを安定した品質で配線するための大事なポイントは2つです。

1つは、フェルール先端はひたすらきれいにすることです(**図6**)。単に機器間やパッチパネルと接続するコード側だけでなく、パッチパネルの穴の先にある既設光ファイバのフェルール部分も必ず拭きます。このひと手間で驚くほど配線の品質が安定します。このとき、ちょっと高いですが、必ず専用のクリーナーを利用、かつ毎回使い捨てして清掃してください。市販のティッシュやアルコール綿などで拭いたり、同じ面で何度も清掃したりしても、端面がきれいになるどころか、返って汚れを広げてしまい、端面の状態を劣化させる恐れがあります。高速の伝送を支える光ファイバのために、ぜひ専用のクリーナー(**図7**)を用いて清掃を行い、安定した通信品質を目指しましょう。

そして、もう1つは、光ファイバにストレスを与えないことです。ストレスとは、必要以上に小さく曲げないよう注意(ペットボトルの底くらいの曲げ直径を確保)しましょう。無理にねじらず、ねじれのない状態で機器やパッチパネルと接続しましょう。電力ケーブルやほかのケーブルの重さを支えるような配線にならないようにしましょう。

▼図6　フェルール先端の汚れ[2]



まとめ

光ファイバを用いたLANは2芯で1組として利用すること、コネクタのキー溝は、実は送信／受信の大切な印であること、そしてコネクタ端面はとにかく専用のクリーナーでオスメス両方とも清掃すること、余計なストレスを与える配線はしないこと。そして光ファイバやパッチパネルは絶対覗きこまないこと。これさえ守れば、光ファイバは怖くありません。SD

■参考

[1]ザ・シーモン・カンパニー
http://www.siemon.co.jp/
[2]グレイテクノス株式会社
http://www.graytechnos.com/
株式会社三喜
http://miki-fiberoptics.co.jp/handyopm.html
[3]株式会社フジクラ　端面検査装置、ワンクリッククリーナー
http://www.fujikura.co.jp/

▼図7　光ファイバクリーナー例[3]

その2

ネットワーク／サーバエンジニアに求められる ラック選定や電源の知識

Author 佐伯 尊子(さえき たかこ)　(株)ブロードバンドタワー

ラック選定

本章ではラックの選定について考えてみます。ラックと言うと、データセンターにそびえ立つ壁のイメージがありますが、オフィスで数ラックを立てて社内利用することもけっして少なくありません。オフィスのサーバ室に何気なく設置してあるラック、誰がどう決めて、どうやって設置していたのか？　また、追加で新しいラックが必要になったときに、どういう基準で何を選べばよいのか？　ここであらためて、ラックの選定から、機器の搭載のしかた、電源まわりについて細かく見ていきましょう。

ラック各部の名称

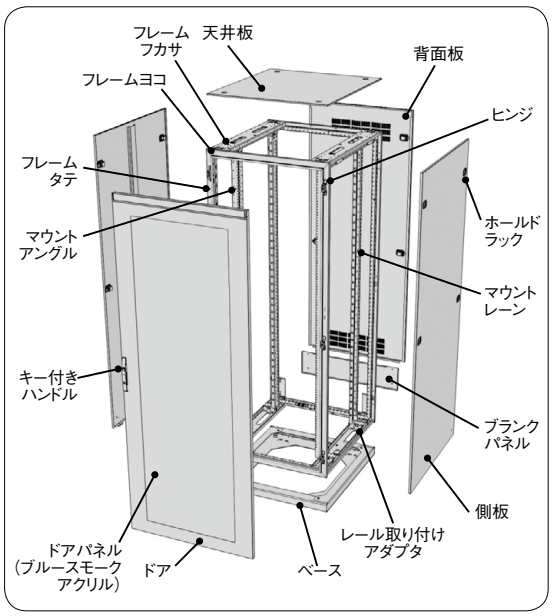
一般的に呼ばれているラック各部の名称を**図1**に示します。

サーバやスイッチなど、ネットワーク機器は19インチラックと呼ばれるラックを選定します。19インチとは、具体的にマウントフレーム間の長さになります(**図2**参照)。また、マウントフレームの項で詳しく説明しますが、ネットワーク機器やサーバを搭載する場合は、EIA(Electronic Industries Alliance：電子機械工業会)ラックを選びます。

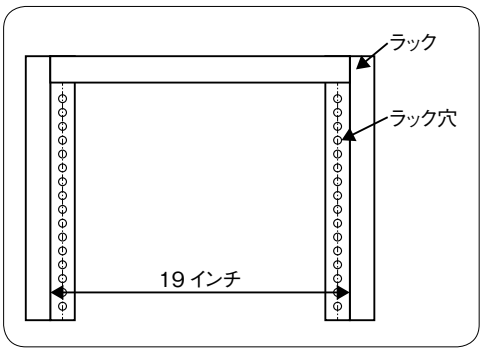
ラックの設置方法

ラックは、縦長でかつサーバやストレージなど重量物を搭載するため、単に床に設置するの

▼図1　ラック各部名称[1]



▼図2　19インチラック



ではなく、転倒防止対策や耐震固定をしなければなりません。これはデータセンターであっても一般オフィスであっても同じです。免震ビルの場合は、このような固定は不要かと思われがちですが、免震ビルは「地震そのものの揺れを吸収して、ゆっくり揺れる」構造です。したがって、免震ビルであっても転倒防止対策や耐震固定が必要となります注1。また、ラック仕様の中で耐震性能を活かすには、耐震固定する必要があります。

具体的に転倒防止対策や、耐震固定をするためのツールについて確認しましょう。転倒防止金物（スタビライザー）は、L字または板状のものがあります。L字タイプは、ラックの前後もしくは左右に取り付け、床と固定して使います。必要に応じて床タイルに穴をあけてボルト固定などを行います。また、**図3**に板状のタイプの取り付け方について示します。こちらも必要に応じて床タイルに穴をあける作業が必要になります。

転倒防止金物は、暫定的な利用や利益に直結しない用途、また2m程度の高さのラックであっても、上部まで機器を搭載しない（もしくは軽い機器やパッチパネルなどの）利用では、専用の工事をせずに設置できて手軽ですが、収益に直結するサービスなど、ラック内の機器に対して安定した稼働を要求する場合は、面倒でもしっかり耐震固定することが望ましいです。

次に、耐震固定の方法について確認してみましょう。まず、ラックの下に架台（がだい）とか基台と呼ばれる金属の枠を用意し、床スラブ（厚板）と架台を固定させます（**図4**参照）。

通常のオフィス床は、床スラブの上に10cm程度の高さを設けて、そこを通常歩く床にしています（二重床）。架台は床スラブから二重床までの高さとし、二重床上にラックが搭載されるようにします。床スラブと架台をボルトで固定したのち、ラックと架台をボルトで固定します。この時金属の架台と金属のラックの間に「スペーサー」と呼ばれるプラスチックの板を挟み込み、架台とラックの絶縁を図ります。このようにしてラックを立てることを立架（りっか）と言います。

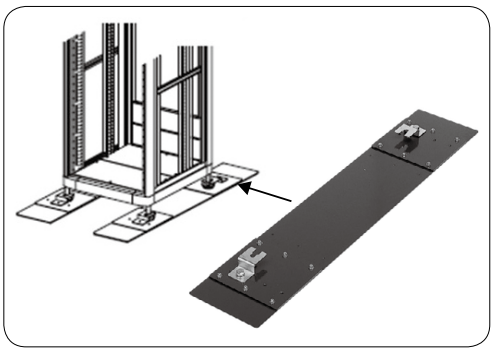
転倒防止金物の取り付けのため床に穴を開ける加工や、架台設置は、総務部や情報システム部でできる仕事ではありません。ラックメーカーに相談する、もしくは自分たちで工事会社に依頼する必要があります。

ラックの購入方法

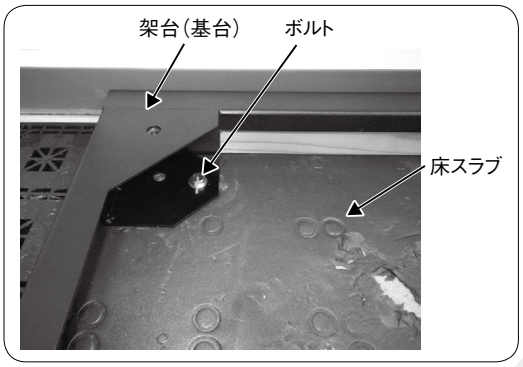
次に、ラックを選びましょう。ラック選びは、スーツのイージーオーダーと似ています。既製服のように店先で吊るしているわけではないため、まずはカタログや展示会、また設置予定場所から、おおよそのイメージを固めることから

注1）免震ビルでラックを免震固定できる台を利用する場合は、共振の恐れがあるため、必ず免震台メーカーに相談してください。

▼図3　転倒防止金物例[1]



▼図4　架台（基台）固定方法[2]



始めます。購入から機器を設置するまでに必要なポイントを見ていきましょう。

搭載機器をまとめる

まずは何を搭載するのかを洗い出します。漠然と「ラックに機器を搭載したいなぁ」というだけでラックを購入すると、「思った以上に奥行きのある機器で、機器を搭載するために、後ろのドアを外す羽目になった(機器サイズとラックサイズが合致しない)」「機器を搭載するのに、追加オプションが必要になった(棚板が必要)」「思った以上に搭載物がかさばり、設置スペースが足りなくなった」など、思わぬところで予想していなかった事実にぶつかることがあります。そのためには、まずは具体的に何を搭載するのか？　ラックの傍には設置するが、ラックには搭載しないもの、などを整理して考えることが重要です。

搭載する機器は、サーバ(機種名)○台、スイッチ(機種名)△台、などと具体的に書き出してみましょう。また、機器のほかにも通信事業者のONU(光終端装置)や、パッチパネル(2月号の第2特集参照)など、日頃気にしていなくても、ないと困るものなども忘れず洗い出しておきましょう。ONUなどラックに直接取り付けられないものは、棚板が必要になる場合もあります。そうすると、棚板も何枚かラックと併せて発注しなくてはならないことがわかります。

実装例

それでは、具体的に機器を書き出してみましょう。**図5**に搭載例を示します。

まず、ラック内にいろいろな機器を混在させて搭載する場合は、ラック前面と背面のマウントフレームの位置に注意しましょう。今回はネットワーク機器とサーバ機器、回線終端装置を混在させて収容することを想定します(**図6**参照)。

● 奥行：

搭載機器の最大奥行寸法＋前面130mm＋背面300mm程度を考慮します。サーバの寸法が支配的ですので、

750＋130＋300＝1,180mm＜1,200mm

の奥行のラックを選定すればよいことがわかります。

● 高さ：

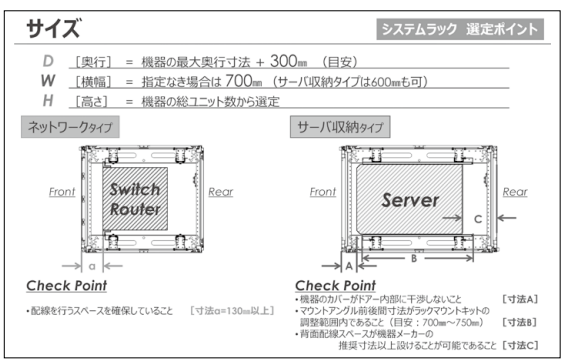
搭載機器の高さについては、U(もしくはRU：Rack Unit)という考え方を用います。ラックには、マウントフレーム穴が等間隔のJISラックと、3つの穴が1単位で数えるEIAラックの2つがあります。ネットワーク機器、サーバ機器を搭載する場合は、EIAラックを選びます(**図7**)。

この中で、1U = 44.45mmですので、ラックの高さは、

▼図5　ラック搭載機器例

- **サーバ(○○社製 型番XXXX、YYYY)×2台**
  - —寸法：幅435mm、高さ44mm、奥行き：750mm
  - —消費電力：500W
  - —重さ：15kg
- **スイッチ(△△社製 型番ZZZZ)×1台**
  - —寸法：幅435mm、高さ44mm、奥行き：160mm
  - —消費電力：12W
  - —重さ：3kg
- **ストレージ(◇◇社製 型番 AAAA)×1台**
  - —寸法：幅170mm、高さ230mm、奥行き：200mm
  - —消費電力：60W
  - —重さ：10 kg
- 通信事業者ONU 1つ
  - —寸法：幅40mm、高さ150mm、奥行き：120mm
  - —消費電力：5W以下
  - —重さ：0.2kg

▼図6　搭載機器と空きスペースの関係[1]



サーバ：440×2＝880…2U
ネットワーク機器：440×1＝440…1U
NAS：230÷44＝5.2…6U
ONU：150÷44＝3.4…4U

したがって、13U以上の高さを持つラックが必要なことがわかります。ただ、ラックの高さ方向の自由度はあまりなく、2,000mm程度のフルラック、もしくは1,000mm程度のハーフラックの2種類から選ぶことになります。今回必要な高さは、ハーフラックで十分ですが、今後の拡張性などを考慮して、2,000mmのフルラックをオーダーすることにしましょう。

ラック幅：

幅を考えるときはラック設置場所のスペースを考えます。ラック内の使い勝手を考慮すると700mmの幅を選びたいのですが、この幅がラック扉の開閉半径になります。したがって、設置する場所のラック前面と背面にドア幅700mmのスペースを確保できるかを検討します。無理な場合は、観音扉(左右に開くタイプ)など、ドアの形状で工夫します。ただ、パンチングドアの場合は、観音扉タイプで対応できないこともありますので、ラックメーカーに相談してください。今回は700mmの幅のドアを取り付けても開閉がスムーズであるという前提で、700mmとしましょう。

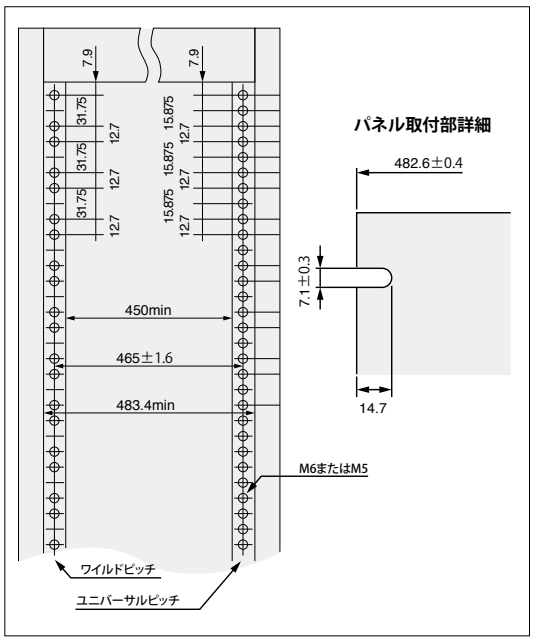
ここまでで、決定した内容として、ラックサイズは、

幅700mm×高さ2,000mm程度×
奥行1,200mm

となりました。

さあ、これで発注できるでしょうか？　ラックの寸法だけではラックのすべてが決まったわけではありません。さらに取り決めなければならない内容を示します。

▼図7　ラック内のサイズ[1]



マウントフレームとねじ

マウントフレーム

ネットワーク機器は前面の2つのマウントフレームだけで固定できるため、2柱ラックと呼ばれるラックに搭載できますが、サーバは専用レールを前後のマウントフレームに固定させて取り付けるため4柱ラックと呼ばれるラックが必要になります。面倒がらずにラック図面に4つのマウントフレームがあることを確認しましょう。

穴形状

ラックに搭載する機器によって、角穴か丸穴かを選びます(**図8**)。ネットワーク機器は、ねじの切られている丸穴に、直接ねじで留めることができますが、サーバ機器は専用のレールをマウントフレームの両端に設置し、そのレールに機器を乗せるため、角穴が必要になります。

1本のラックで丸穴と角穴を共存させることはできないので、サーバ機器を1台でも設置するならば、おのずと角穴のマウントフレームを選ぶことになります。

ラックにネットワーク機器を搭載するには、

ねじが必要です。丸穴はすでにねじが切られているため、そのままねじ固定ができます。しかし角穴の場合、ねじ山がありません。そこでケージナットを用います(**図9**)。ねじ／ケージナットにはM5 、M6等異なるサイズが存在しています。ケージナットには表記がありますが、ねじ側には表記がありません。混同させて利用すると、ねじ山を潰したり、機器をラックに固定できなかったりします。安定した機器設置のためにも、それぞれ分けて利用／保管しましょう。

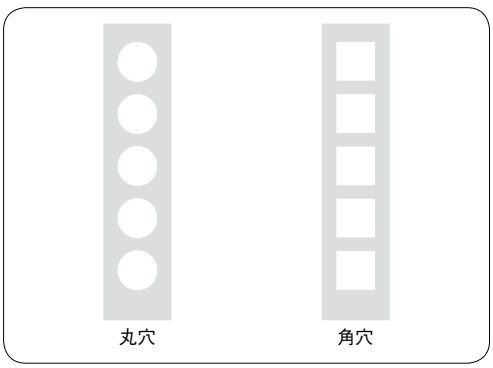
ドアの選定

ドアは、搭載する機器の総発熱量から判断します。**図5**で示した搭載機器の総発熱量は、

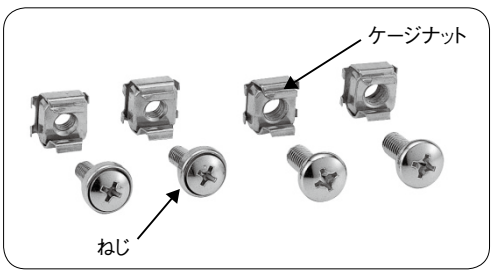
サーバ：500W×2＝1,000W
スイッチ：12W×1＝12W
ストレージ：60W×1＝60W
ONU：5W×1＝5W
合計：1077W

となります。

▼図8　マウントフレームの穴形状



▼図9　ねじとケージナット[1]



オフィス内の一角をサーバ室にする場合、サーバ室の場所によっては、オフィス空調だけで十分な冷却ができないこともあります。そのときは追加空調の増設などを実施し、機器の冷却に努めます。

総発熱量が、3,000W以下の場合はパンチングドアで対応できます。

エアフロー

次に、エアフローについて考えます。エアフローとは「空気の流れ」です。サーバ室やデータセンターでは、冷たい空気と機器の排熱の暖かい空気の流れを管理することで、機器の冷却と、省エネルギーを実現します。

データセンターなどでは、機器の冷却は必須であるため、エアフローのコントロールにとても神経をとがらせますが、1〜2ラックかつ社内用など利用が限定されているサービスの場合は、機器が稼働温度範囲で動作するようにユーザレベルで気をつける程度で、運用上は問題ないかと思います。そうは言っても、全然気にしなくて良いわけではありません。

具体的には、次のことに気をつけましょう。

- サーバ室の前後のドアはきちんと解放できるようにする(機種にもよりますが、ドアを取り外すことができるラックが多いです)
- ラック回りのデッドスペースは常に片づけて、冷風が取り込みやすいように、排熱を廃棄させやすいようにする

ただし、ラック数とラック内機器などの発熱量の合計が、既存のオフィス空調の能力を上回る場合は、ラック設置場所をオフィスのほかのスペースと分離し、独立した部屋にさらに能力の高い空調設備を用意する必要があります。しかし、ここまで本格的に実施するとなると、大規模な工事と維持管理費が発生しますので、まずは、オフィスに置くことが本当に必要なのかを吟味しましょう。場合によっては機器をデータセンターに持って行く、ないしはサービスを

クラウド上で利用することも検討したうえで、結論を出すことが望ましいと思われます。

また、空調とは直接関係はありませんが、サーバの数が増えるとそれだけ騒音も大きくなります。騒音防止を目的としてサーバ室を設けるときは、天井まで壁で塞ぎ、独立した部屋にすることが望ましいです。

適切なエアフローのためのラック付属品

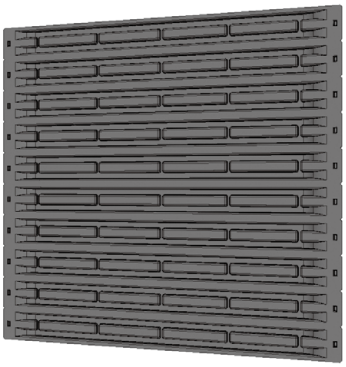
データセンターのように、適切なエアフローを設計したサーバ室にラックを設置し機器を搭載するときは、機器によって暖められた空気と空調機の冷たい空気が混ざらない工夫が必要です。暖気と冷気が混ざることで、空調機を必要以上に稼働させることになり、結果余計な電気代が掛かってしまいます。そのため、サーバ室内のエアフローをきちんと管理することが大切です。ラック列の暖気側を「ホットアイル」冷気側を「コールドアイル」といいます(**図10**)。このホットアイルとコールドアイルをきちんと分離させることで、効率よく機器を冷却することができます。

ホットアイルの暖気をコールドアイル側に流入するのを防ぐツール紹介します。

まず、思いつくのが、ブランクパネルかと思います(**図11**参照)。これは、ラックの横幅に対して背面から排出した温かい空気が前面の冷風側に戻るのを防ぎます。高さ固定タイプのブランクパネルは、機器の入れ替え頻度が高くない場合にはとても有効です。固定方法も、マウントフレームの穴に取り付けるタイプがほとんどです。このとき、マウントフレームの穴形状が、角穴か丸穴かで、取り付け方法も変わってくる場合があるので、取り付け方法について、十分確認しましょう。

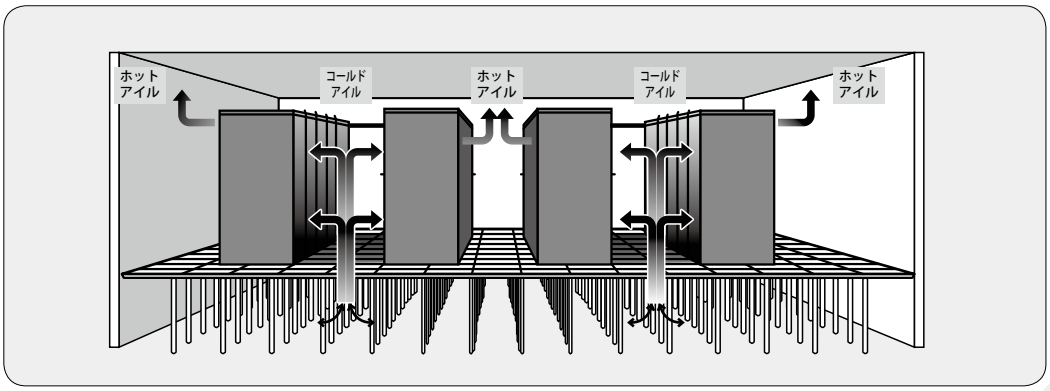
次に、ブラインドタイプのブランクパネル(**図12**参照)を確認しましょう。こちらは、空きU数が頻繁に変化する場合に適しています。とくにサーバ専用のラックの場合は、機器の入れ替えの際、そのまま引っ張って自由にU数を変えることができるためとても便利です。**図11**で示すような高さ固定式のブランクパネルを用いるよりも、1ラックに1枚常備しておけば、わざわざいくつものブランクパネルを用意する必要が

▼図11　ブランクパネル例[1]



U単位で折って使います

▼図10　ホットアイルとコールドアイル



ないため、置き場にも困りません。

次にラックのマウントフレームの外側の縦方向の空きスペースにおける背面の熱の前面戻りを防ぐツールについて紹介します。

ブラシタイプやパネルタイプの遮断方法があります。パネルタイプの場合は、ケーブルの前後を行き来する通信ケーブルの場所や量を考慮する必要があるため、ある程度機器の搭載イメージがないとなかなか選びづらいため、放置されやすいですが、エアフローを管理する上で、忘れてはならない場所です。また、ブランクパネルと異なり、空きスペースの大きさや取り付け位置がさまざまなため、設置されているラックメーカーに依存する製品も多く、選定には吟味が必要です(**図13**)。

ラックPDU

続いて、ラック内に設置するラックPDU(Power Distribution Unit)を決めましょう[注2]。ラック内に設置するサーバやスイッチなどの機器に電力を供給するための電源コンセントのことを指します。一定数のコネクタのメスの口が並んだものです。

● 電源コネクタ形状のまとめ

まずは、機器と接続する電源プラグの形状ですが、機器側の仕様を満足させるものを選ぶことが大切です。大きく分けて100V系と200V系に分かれます。100V系であれば、NEMA 5-15P/5-15Rが一般的です。アースつきの縦2本の形状です。また、サーバをおもに搭載する場合は、200V電源にも対応できるIEC 320で規格化されているC13(メス)C14(オス)を利用することも増えてきました(**表2、図14**)。

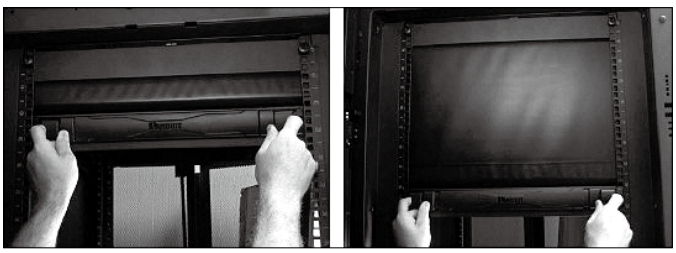
なお、NEMA規格の各記号の末尾のPとRは、それぞれオス(Plug)とメス(Receptacle)の略ですので、発注の際間違えないように注意しましょう。

また、L6-30の場合、標準的にはアメリカン電機のものを指しますが、同じL6-30と称しながら互換性のないメーカーのものもあるので、機器側がL6-30の場合、念のため発注の際、メーカーに仕様を確認することをお勧めします。

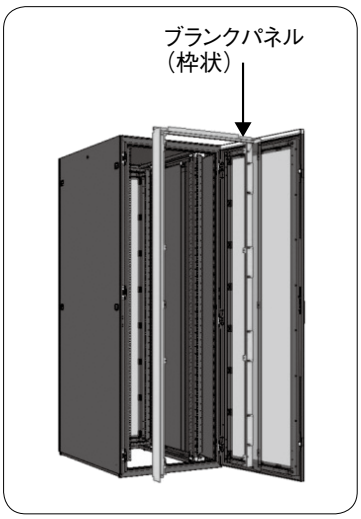
日本の場合は、電極部分はNEMA 5-15Pの形状と変わりませんが、アース極の代わりに緑色のアース線が付いているタイプがあります。この電源コードは、家庭用機器のみの利用だと考えてください。サーバ室やデータセンターでこの形状のコネクタプラグをそのまま使用すると、アース線の先端がほかの機器の電源部分に接触したり、ラックPDUの電源部分に触ったりする可能性があります。むき出しの導体部分に接触した場合、最悪、短絡事故を起こしかねません。

注2) 同じPDUという略語が電気設備では「分電盤」という意味で使われています。ここでは区別するために「ラックPDU」と呼ぶことにします。

▼図12　可動式ブランクシェードパネル(FLBSIM-51) [3]



▼図13　縦方向のブランクパネル例 [1]



めんどうくさくてもNEMA 5-15Pの形状のものと取り替えて利用しましょう(**図15**)。

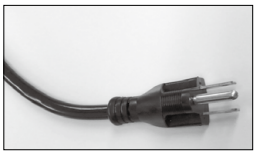
また、電源コードについても、機器に標準で付属されているケーブルは黒色で2m程度の長さがありますが、実際にラックの中で配線すると、2mは長く、ケーブルの取り回しが難しく、サーバなどの機器の排熱のエアフローを邪魔してしまいがちです。そのため最近は短い長さの電源ケーブルを扱っているメーカーも増えてきました。また、単に短いだけでなく、コネクタ先端部分に抜け止め対策したもの、さらに電源コード自体も色分けできるような製品が出てきました。このように電源系統を色で明確に分けることにより、視認性の高い電源システムが構築できるかと思います。そして、この視認性を上げることによって、電源系統が明確になり、運用時の間違いを減らすことが可能になるかと思われます。

ラックPDUを選択する際、適切なコネクタ形状で、必要な口数(接続数)が確保できれば、何を選択してもよいわけではありません。家庭用の延長コード、OAタップのような電源タップを用いても電源を供給することはできますが、サービスを提供するには向きません。とくに安価な電源タップを利用すると、機器に供給する電圧が不安定になったり、すべての口に電源ケーブルを接続したときに、十分な電圧が供給できなかったりなど、不具合が発生する可能性があります。また、ラック内にしっかりと固定できず、不安定な状態で電源を供給することになることもあります。高価に感じるかもしれませんが、ラックに取り付けるタイプの純正品を選ぶことが、機器の安定稼働には大切です(**図16**)。

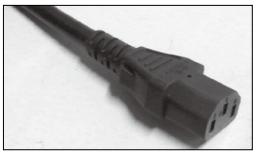
▼表2　電源コネクタ仕様例

オス	メス	電流 電圧
NEMA 5-15P	NEMA 5-15R	15A 125V
NEMA L5-20P	NEMA L5-20R	20A 125V
NEMA L6-20P	NEMA L6-20R	20A 250V
NEMA L6-30P	NEMA L6-30R	30A 250V
IEC60320 C14	IEC60320 C13	15A(UL,CSA) 250V

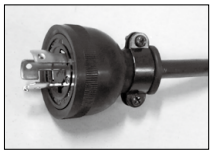
▼図14　電源コネクタ形状例



5-15P



C13



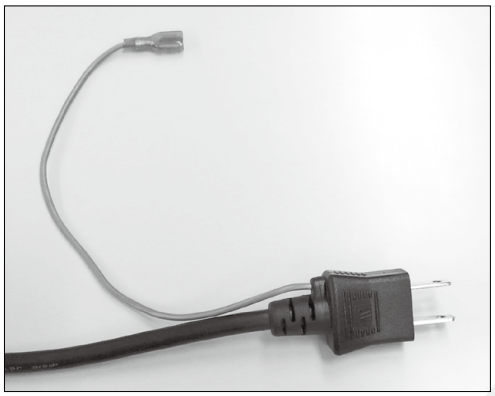
L5-20P

電源設備

最後に、このラックPDUを経由してオフィスに供給される商用電源について説明します。とくにデータセンターでラックを借りて利用する電源と、オフィスにラックを立てて利用する電源の違いについて、ここで簡単に見てみましょう(**図17**)。

同じ「電源設備」であっても、ビルに引き込む方法、ビルの中でラックまで給電する方法がまったく違います。一般的なオフィスビルの場合、高圧6,600Vを1ラインで引き込み、変圧器で200Vまたは100Vに降圧して各フロアのコンセントに分配しています。ビル設計時に、「各フロア、各ゾーンに対して提供できる電力」を一意に

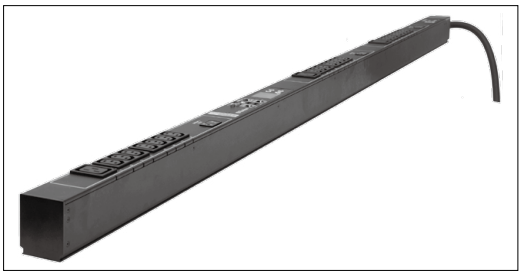
▼図15　アース線付5-15P



決めていますので、それ以上必要な場合は、テナント側の費用で電気設備を増設することになります。ビルの受電設備から増強するような大規模な工事になることや、さまざまな制約のために提供不可となる場合もあります。

また、予備の受電設備や電源設備は通常ありませんので、1個所でも不具合が発生した場合、ラックPDUに電源が来ない恐れがあります。防災用の非常発電機は法律上設置されていますが、それはあくまで防災時にビルの最低限の設備を動かすためであって、各テナントに電気を提供することは想定されていません。さらに、法定の電気設備点検が標準的には年に1回あります。これはビルすべての電気を止めて、電気設備の異常がないかどうかを確認し手入れを行うものです。だいたいまる1日点検日が設定されます。

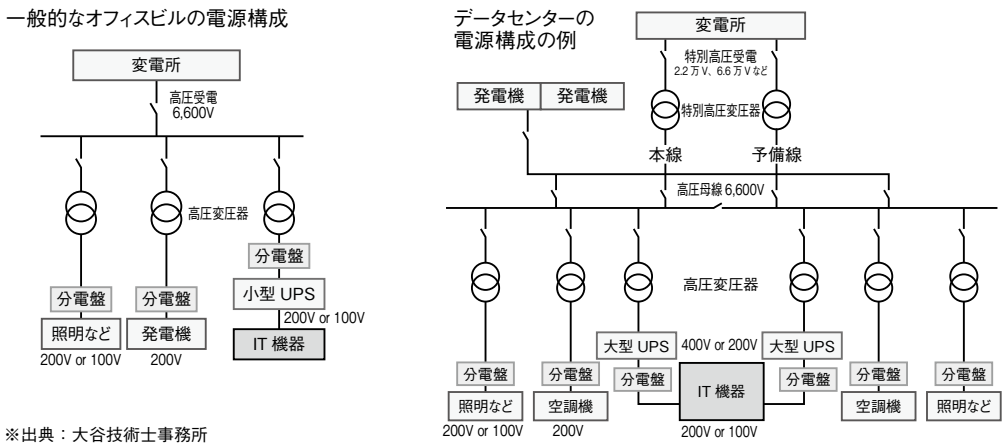
▼図16　ラックPDU例[4]



当日はビルが全館停電しますので、当然ながらサーバ室も停電します。それに備えて、事前にラック内の機器をシャットダウンさせることが必要です。

それと比べて、データセンターは、まず変電所からビルまでの引き込みを2つ以上用意することが一般的です。また、建物内の電源設備も各段階で二重化されていますので、受電や内部のどこかが停止することになっても、適切に切り替えわって給電し続けるしくみがあります。もう少し詳細に見てみましょう。データセンターは大量の電力を必要とするため、契約電力が2,000kW以上(ラック数での規模感では、おおよそ300ラック以上)の場合は、特別高圧での受電が一般的です。それをラックまで分配するための電源構成は二重化されていることが多く、どこか一ヵ所または一系統に不具合が発生してもほかの系統から供給し続けることができます。また、万が一商用電源が停電した場合でも、無瞬断でUPSのバッテリーを用いてラックへの給電が止まらないようになっています(**図18**)。その間に非常用発電機が自動的に起動し、そのあとはこの発電機から長時間に渡って各ラックや空調などに電源を供給し続けます。このUPSは大型のもので、オフィス用にサーバと一緒に購

▼図17　一般オフィスビルとデータセンターの電源設備の違い



入する小型のものとは仕様や構成がまったく異なります。この設備はデータセンター側で電源設備の一環として設置されています。

こうして商用電源と非常用発電により継続的に供給する電源ですが、データセンター内では前述のとおり2系統に分けており、必要に応じてラックに両系統の電気を供給できるようになっています。A系／B系や1系／2系といった表記になっているかと思いますので、データセンターを利用する場合はデータセンター側に確認しましょう。利用者の要求に応じてラックPDUもこの2系統分が用意され、ラック内の機器と接続しています。この系統を色で分ける工夫もされていることがあります。先に説明した電源ケーブルやラックPDUによる色分けです。ケーブルもラックPDUも黒が多いのですが、最近はカラーバリエーションが増えてきました。このように色分けすることで電源系統の間違いや、誤抜を防ぐことが期待できます。

UPS

最近は前にも増してUPSの導入を検討しているユーザが増えているかと思います。ここで、UPSについてまとめておきましょう。

UPS（Uninterruptible Power Supply/System）は無停電原電装置です。ただし、留意しておきたいのは、サーバやPCとセットで使う小型のもの（おおよそ数kVAまで）と、データセンターにあらかじめ設置されている大型のものとでは、中身や用途および動作が異なります。

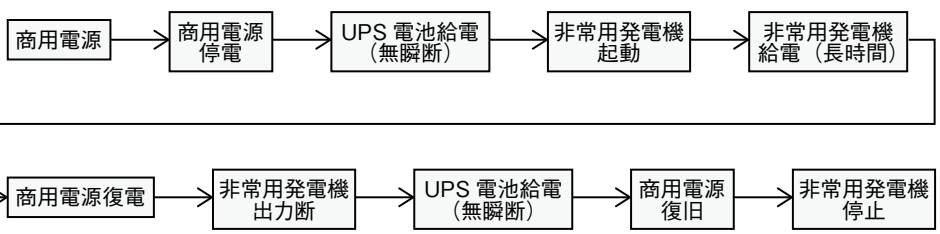
小型UPSは、ノートPCなどのバッテリーを想像するかもしれませんが、ノートPCなどのバッテリーと大きく違うのは、UPSは以下の目的で作られていることです。

・長時間の停電の場合に備えて、機器を安全にシャットダウンさせるためのもの
・瞬停や電圧変動から機器を守るもの

要は、「○時間商用電源が止まっても動き続ける」ためのものではないということです。そもそも一般的な小型UPSは、バッテリーの容量や負荷の消費電力にもよりますが、最大でも15分程度しか電力を供給できません。もしUPSに接続された大規模バッテリーで何時間も機器を駆動させようとした場合、バッテリーに関する初期費用・メンテナンス費用・更新費用・設置場所などでかなりのコストアップとなってしまいます。したがって長時間の停電でも電源を確保する必要がある場合には、非常用発電機が現実解となり、それを備えた大型電気設備が必須になります。これは一般的なオフィスビルにほんの少しの改造を施した程度では賄えません。そのため、このような設備を備えたデータセンターの利用が現実的な選択肢となります。

次に、UPSの種類について説明します。大きく3種類あります。それぞれ特徴がありますので、自分たちの利用する用途に合わせて最適なUPSを選ぶ必要があります。

▼図18　データセンターにおける商用電源の停電から復旧まで



※出典：大谷技術士事務所

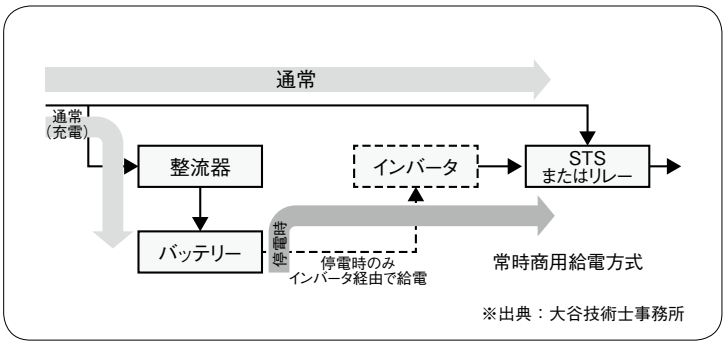
・常時商用給電方式(オフライン方式)(図19)
・常時インバーター給電方式(図20)
・ラインインタラクティブ給電方式(図21)

この方式は、通常時は商用電源をそのまま機器に供給し、一方で商用電源からバッテリーに充電していますが、商用電源が止まったときにのみ、バッテリーからインバーターを介して機器に電力を供給します。切り替え時に、ほんの数msec～10数msecの瞬断が生じますが、サーバ機器などは通常20msec程度までの瞬断であれば、機器の電源回路中のコンデンサに一時的に溜まっている電気で動き続けるため、機器動作には影響ありません。日本など、商用電源の質がよい国であれば、通常時、商用電源からの供給で問題なく動作します。

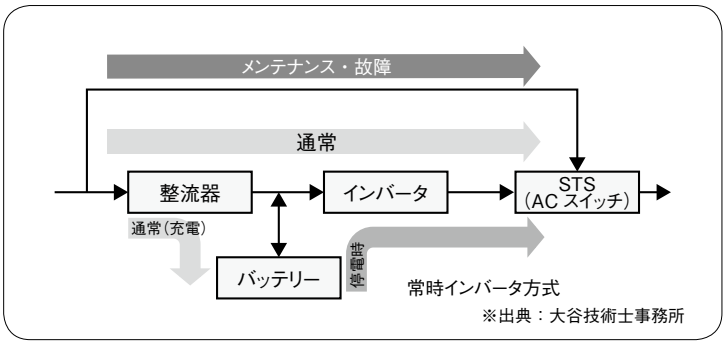
基本的には、オフィスなどに設置するサーバやPCなどにセットで用い、このUPSとRS-232Cなどで接続して停電が通知され、頃合いを見計らって電池が給電できる時間内で自動的にシャットダウンをかけるために使われます。先に説明したとおり、長時間稼働を目的としているものではないことに注意する必要があります。

この方式は、通常時も商用電源が止まったときもインバーターから電力を供給するしくみです。インバーターで制御された正確な電圧・周波数の交流を作るために、電源からの交流は一度整流器で直流にされ、ここにバッテリーが直結されています。そのため、商用電源が止まった際もまったくの無瞬断でバッテリーからインバーター経由で電力が提供されます。ただ、整流器とインバーターを介すため、商用電源利用時であっても、常時商用給電方式よりもロスが大きくなります。基本的には大型であ

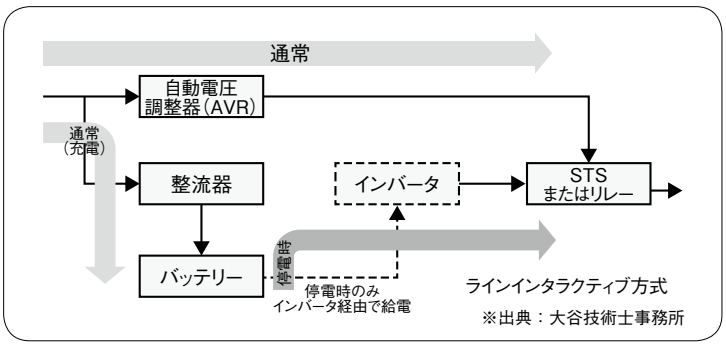
▼図19　常時商用給電方式



▼図20　常時インバーター給電方式



▼図21　ラインインタラクティブ給電方式



り、データセンターの設備の一部として、発電機給電と商用給電とのつなぎ役を果たすために使われます。データセンターでは非常用発電機とセットで用意されるため、各サーバへの停電通知は必要ありません。

なお、オフィスの一角に設けられるサーバルームでも数10kVA程度の常時インバーター方式が採用されることがあります。この場合、非常用発電機が併設できないことも多いので、停電時の自動シャットダウンのしくみを検討する必要があります。

電圧が安定しない国や地域で用いられる、常時商用給電方式に代わるものです。通常時は自動電圧調整器で電圧のみある程度調整された商用電源が給電され、並行して整流器からバッテリーに給電、商用電源が止まったときのみインバーターを介してバッテリーから電力を供給します。切り替え時には常時商用給電方式と同様に数msec程度の瞬断が発生しますが、サーバ機器の動作には問題ありません。

これらの中から、停止の可否や予算・供給時間・供給される電源の性質などの要件をまとめ、どの機器や容量を選ぶか決定します。必要な電力は、WもしくはVAで計算します。

W(有効電力)＝VA(皮相電力)×力率

力率は、IT機器では一般的に1に近い値ですので、概算であればW＝VAとみても大きな差異はありません。厳密に計算したい場合は、低めに見積もっても95%程度としてよいでしょう。USPから給電する機器すべての電力を同じ単位(WまたはVA)になるようにそろえ、すべての電力の和と、必要な供給時間から、UPSの機種やバッテリーの容量を選定します。最近はUPSメーカー各社のサイトを通して購入する際、WかVAかを選んで見積ることができるようになっています。くれぐれも電力の単位をそろえることを間違わないようにしてください。

また、供給される電源の品質・安定性という意味では最も優れている常時インバーター方式は、おおむね10kVA以上のものとなります。このタイプのUPSを選んだ場合、商用電源との接続のための電源工事を要しますし、設置スペースも必要になます。さらに定期的メンテナンスが必須で、その費用も必要になるため、容易に導入できるものではありません。導入に際しては、慎重な検討が必要です。

こうして選択したUPSですが、さらに大切なことがあります。UPSのバッテリーは数年で劣化する化学製品のため、一度導入したら未来永劫使い続けられるというものではなく、利用の有無にかかわらず5〜7年に一度は交換が必要になります(機種に依存しますので、詳しくは選定したUPSの仕様を確認しましょう)。バッテリーの電池は交換することを前提としてUPSを選ぶことです。

交換時にIT機器への給電が止まるか否か？
止まるなら、止めてよいIT機器用としてのUPSなのか？　ということです。単に供給時間の長さや性能だけで選んでも、給電するIT機器の用途や利用条件を考慮しない選択は無意味なものとなり、どんなに優れたUPSであっても正しい選択とは言えません。接続する機器の電源に対する条件をきちんと整理して、それに見合う最適なUPSを選ぶことが大切です。SD

■参考

[1]日東工業株式会社
http://www.nito.co.jp/
[2]株式会社ハイアーネット
http://higher.co.jp/
[3]パンドウィットコーポレーション日本支社
http://www.panduit.co.jp/
[4]ラリタン・ジャパン株式会社
http://www.raritan.com/jp/

※執筆・図版協力：株式会社大谷技術士事務所

interview

DevOpsは
日本に定着するのか?

春の嵐呼ぶ! DevOps座談会

——TKC、U-NEXTの2社が考える開発のあり方とは

「Amazonというシアトルのブックストアは11秒に1回、本番環境にコードをデプロイする」と、Pivotal社のアンドリュー・クレイシャファーがAmazonの事例を紹介し、そこで「DevOps」という言葉を盛んに強調していたのが、およそ2年ほど前のPivotalジャパンのイベントでした。それから2016年になり、DevOpsは本当に日本のIT部門で実現されているのでしょうか?

実際には欧米のツールベンダーが盛んに自動化やテストのためのツールを宣伝しているだけで実際の企業における実践は進んでいないのでは……。

そこで今回は、ITをビジネスの原動力として業界をリードしている、㈱TKC、㈱U-NEXTに集まっていただき「DevOpsは日本に定着するのか? そのヒントは?」というテーマで座談会を行いました。

かたや会計ソフト、かたや動画配信と組織の成り立ちも対照的な2社ですがITの力でビジネスを推進してきたことには違いはないでしょう。

また、Pivotal Labs Tokyoで実際にアジャイル開発を企業に推進する役割であるPivotalジャパン㈱の技術統括部テクニカルディレクター仲田聰氏にも同席いただきました。

●モデレータ
松下 康之(まつした やすゆき)
フリーランス・ライター／マーケティングスペシャリスト

●参加者
㈱TKC
経営管理本部システムエンジニアリングセンター　クラウド化推進部　課長
三坊 鉄平(さんぼう てっぺい)氏

㈱U-NEXT
NEXT事業本部　事業戦略室　マネジャー
柿元 崇利(かきもと たかとし)氏
NEXT事業本部　システム開発部
秋穂 賢(あきほ すぐる)氏

Pivotalジャパン㈱
技術統括部テクニカルディレクター
仲田 聰(なかだ さとし)氏

●会場提供　Pivotalジャパン㈱

ソフトウェア開発のあり方は変わるのか? ―今起きている問題は何か

――TKCさん、U-NEXTさんのビジネスそしてソフトウェアの開発スタイルについて簡単に教えてください。

三坊：TKCは、会計事務所とその関与先企業、地方公共団体、中堅・大企業をお客様として、情報サービスを展開しています。社員が約2,300名、そのうちソフトウェア開発とインフラの管理などで800名ぐらいですね。当然ですが、扱っているソフトウェアが会計や税務ということもあって非常に厳密な開発体制というか、ソフトウェア開発のスタイルをとっています。ウォーターフォールモデルをベースにした開発手法を採用しています。たとえば、税制が変わったりすれば、その内容をキチンと仕様書に落とし込

㈱TKC　経営管理本部
システムエンジニアリング
センター
クラウド化推進部　課長
三坊 鉄平（さんぼう てっぺい）氏

㈱U-NEXT　NEXT事業本部
事業戦略室　マネジャー
柿元 崇利（かきもと たかとし）氏

んでから開発に取り掛かる、という感じですね。インフラもデータセンターを自前でもっています。24時間365日、当社社員が交代制でサービスの稼働状況を監視するなどのサービス体制をとっているところが特長です。これもクラウドでお客様のデータを預かっているので当たり前のことですけども。あと弊社のサーバはほぼ100% Windows OSでデータベースもSQL Serverを使っています。開発はVisual StudioやDelphiを使用しています。開発言語はC#やDelphi XEをサービスの要件に応じて使い分けています。

柿元：U-NEXTは日本では最も早い時期、2007年から専用セットトップボックス（STB）によるTV向け定額制見放題動画サービスを開始し、そのあとPCやスマートフォン、タブレット向けへとデバイスを拡張しています。IT部門は約50名です。

今回は、約1年かけてこれまで使ってきた動画配信のシステムを全面的に刷新しましたので、その話ができればよいかと思います。当時のシステムでは今後目指すべき成長に合わなそうだということで、とにかく全部書き換える、サービスインから逆算するとこの辺から始めないと間に合わないという発想でこの刷新プロジェクトをやりきりました。実際には動画配信にはいろいろなステークホルダーが絡んでいます。動画ファイルそのものからDRM（デジタル著作権管理）のしくみ、Web、データベース、アプリ、他社サービスとのつなぎ込みなど多くのシステムがあるのですが、それらを含めて全部作り直しました。

――U-NEXTさんのシステム刷新はかなり大胆な決断だったと思いますが、逆算という発想がユニークですよね。

柿元：そうかもしれないですね。実際にはシステムの設計をやりながら、人を採用して、どんどん開発をしてもらってテスト……というのを繰り返していたという感じです。今回同席させていただいた秋穂も、このプロジェクトのスタート直後に入社してもらったエンジニアですし、私も比較的最近入った人間です。うちの役員がとにかく書き直す、全面的に刷新すると決めたというのが大きかったですね。

三坊：そういう決断ができたのが大きかったんですね。TKCの場合は毎年の税制改正への対応も含めて常にシステムを維持していかなければいけないので、なかなかそういう思い切ったことが簡単ではありません。実際にシステムの開発に際しては仕様を決めるチームが法律や税制変更の内容を検討して仕様に落とし込むという作業が発生しますし、仕様書も非常に大きくなってしまうんですよね。その仕様作成が完了してから実際の開発に進むという感じで。実際には開発チームはそれを待っているのではなくて税制改正とは別のレベルアップ機能を先行して開発して、後から税制改正部分を結合するスケジュールとして効率化を図っています。

仲田：実際にお話を伺うとTKCさんのソフト

DevOps座談会 ――TKC、U-NEXTの2社が考える開発のあり方とは

㈱U-NEXT　NEXT事業本部
システム開発部
秋穂 賢(あきほ すぐる)氏

Pivotalジャパン㈱
技術統括部テクニカルディレクター
仲田 聰(なかだ さとし)氏

ウェア開発は非常にキチンとしたやり方で本当にITILの教科書どおりという感じなんですね。でも実際には……。

三坊：まぁ、製品のリリース直前に前工程に戻ることもありますし、緊急で懸命に開発してデバッグして……というのを繰り返すこともあります。なかなか製品を超えた知見の共有というのも進みづらいんですよね。ソースコードのアクセスも厳密に制限されていますし。

秋穂：U-NEXTの場合ですと、すべてのデベロッパーがほとんどのソースコードを共有できる環境にあります。たとえばあるプロジェクトで困ったことがあっても、この件に詳しいのは誰か？――というのは簡単にわかるんですよね。非常に有機的にプロジェクトを共有して、自然に知識の集約を共有できていくというしくみです。

開発と運用が仲良くすることがDevOps成功への鍵か

――そうすると開発するソフトウェアの性格によって、やはりアジャイル開発というかDevOpsというのは難しいということなんでしょうか？

座談会感想 *1*

U-NEXT(柿元)

進化論の基礎理論を組み立てたダーウィンの言葉に「変化に最もよく適用した種が生き残る」とありますが、激化する競争環境を生き抜くために、とにかく素早く反応できるU-NEXTの組織、文化づくりを考えた結果選んでいた思想がまさにDevOpsだったんだ、と図らずも発見できたのが、この座談会でした。

動画配信ビジネスの世界は今まさに戦国時代、群雄割拠の様相を呈しています。みな生き残りをかけて必死に戦っています。技術面でもとにかくスピードが速く、つい最近まで標準だった要素技術が1～2年ですっかり時代遅れ、なんてことも当たり前です。変化することを前提にしないとそもそもサービスの安定運営すらおぼつかないんです。仲田氏の「開発が早くなってもビジネスに反映されないと意味がない」という言葉は最前線の肌感覚としても正にそのとおりで、アジャイルとDevOpsは思想的に切り離せるものではなく、両方セットで導入して初めて意味を持つ、「真に変化に強くなる」ことができるととらえて定着のため力を入れています。ソフトウェア開発の世界は北米圏から学ぶことが多いですが、これらの考え方はもとをただせば日本の製造業から発祥しているもの。我々が実践できない理由はないですよね。謙虚に学んで改善を重ね、世界最高レベルのDevOpsを持つチームになりたいと意欲を燃やしています。

仲田：すべてをアジャイル開発にする必要はないんですよね。レガシーなコードを無理矢理変えようとする必要もありません。どちらかと言えばこれから作るアプリケーション、これからの変化に対応したビジネスを支えるアプリをアジャイルで作りましょう、ということだと思います。さらに言えば実際にアジャイル開発で開発そのものが速くなってもそれを本番環境、つまりビジネスの現場に導入できなければ意味がないんです。なので当社のアンドリュー・クレイシャファーがイベントで言っていたようにとにかくコードを書いたら、即本番環境にプッシュして使ってみる、変更が少なければ手戻りも少なく済む、それを毎日毎日回すというのがDevOpsの本質です。単に「自動化ではなくてビジネスに活かせるかどうか？」ですから、ビジネスとDevOpsはつながっていないとおかしいんですよ。アンドリューには前に「そもそもDevOpsという言葉が悪い」と言ったら「オレもいいとは思わなかったけどいまさらしょうがない」と言ってました(笑)。

三坊：実際にDevOpsの説明やプレゼンテーションを聞いてもやっぱりツールの話が多くて開発サイドの話ばかりなんですよね。開発と運用を一緒にといってもそれぞれ目的も違うし、同じ話でも聞こえ方が違う。運用サイドにとってみれば「DevOpsをやると私たちの仕事がなくなるの?」みたいな話にしか聞こえてこない。なのでDevOpsを進めるのであれば運用側がハッピーになるシナリオでないと。

仲田：実際に私たちがDevOpsの話をしているのはおもに開発サイドですね。運用側とはなかなか話が進まない。弊社としては運用の人と開発の人を一緒にして説明しないようには心がけています。

――「混ぜるな危険!」みたいな話ですね(笑)

柿元：U-NEXTでもだいぶ素早く開発と本番環境へのデプロイができるようにはなってきましたが、それでも1日に数回という感じです。ところで、社員に以前北米企業に在籍していた人がいまして、彼いわくその会社はDevOpsをやり過ぎていたと言うんですね。つまり年に数回しか実行しないようなジョブでも全部ガチガチに自動化してとにかく何でもかんでも自動化だと。デベロッパーが10人に対してDevOps専任が3名もいるような感じで、自分の権益というか居所を守るために仕事をするようになってしまう。そういう知見もありますので、U-NEXTの中ではDevOps担当は開発と運用からそれぞれ人を出していこう。兼任で行こうと話しています(笑)。

仲田：DevOpsについて言えば、運用の仕事がなくなるということではありません。つまり仕事の性格が変わるという部分がなかなか伝わっていないと思いますね。これまでオペレータという形で「操作」していたものが「プログラムで自動化」するということになると操作そのものはなくなるかもしれませんが、それをプログラムする仕事は残る、その部分に運用サイドの仕事は残るので。

三坊：兼任ということではないですが、運用のオペレータもシニアな人は運用の業務全体が理解できていますので「どこかに無駄はないか？

自動化できる部分はないか？」を見つけることができるんですよね。……なのでそういう人に運用業務だけをさせるのではなく、自動化や効率化の方向に向かって徐々に方向転換していけば運用の自動化も進むし、開発部門とも話ができるようになる、そして上級オペレータとしてのキャリアパスとしても進む方向が見えてくるんです。それがDevOpsにつながるのではないか、そのような構想を社内で話しています。

――その辺になるとITの技術論ではなくて組織論、人事論になったりしますよね。

DevOpsを考える手掛かりは、ツールではなく考え方の変化

会計ソフトと動画配信、ビジネスも違えば開発スタイルも正反対な2社ですが開発から運用までのスピードアップはゴールとして共通でしょう。DevOpsは単なるツールの導入ではなくいかにビジネスにインパクトを与えるのか？　組織はビジネスのゴールを達成するためにどうあるべきか？　などにもつながるキーワードであると言うことが実感できた座談会でした。今回の座談会を通じてDevOpsの実践につながるヒントになれば幸いです。SD

座談会感想 2

TKC（三坊）

DevOpsという言葉に最初に出会ったのは2011年ごろです。当時は開発部門と運用部門でお互いにストレスなくサービスを提供していくための流行り言葉という程度のイメージしか持っていませんでした。アジャイル開発やソフトウェアの品質において新しいムーブメントの予感はありましたが、ウォーターフォールモデルをベースにした開発手法を採用している当社では、縁のない言葉としてとらえていました。

本格的にDevOpsを意識し始めたのは2014年ごろです。MicrosoftやVMwareによるDevOpsに関連するソリューションに触れる機会が増えてきました。2015年夏のVMware社のイベント（VMWorld2015）においても、DevOpsのソリューション、コンテナ技術との連携を強くアピールしていました。

これまで私たちには遠いと思っていた言葉が身近に感じるようになったとともに、DevOpsの目的は何か、その本質はどこにあるのか、自分の中で確立させたい、と考えるようになりました。

最近、DevOpsに関するイベントに参加して、次のような話を聞きました。「DevOpsは『開発者に力を与える動き』である。これまでの運用主体の時代から開発主体への時代に移り変わっていく。開発者の視点で話がす進むようになる」そして、HashiCorpなどのOSSを積極的に利用しこう！　マイクロサービスとコンテナ技術は熱い！　という内容でした。非常に盛り上がっている雰囲気を感じながら、DevOpsはツールや開発の話だけなのか、という疑問が残っている状況でした。

今回の座談会で、Pivotal社の事例やU-NEXT社の話をうかがうことができました。OSSなどのツールに囚われることなく、会社ごとのこれまでの文化やサービスレイヤごとの要件にあわせたDevOpsへの取り組みがあっていいと感じています。ウォーターフォールでの開発やオンプレミス環境で提供するサービスには、それに見合ったDevOpsがあり、10 deploys per dayの世界もあれば、テストありきのリリース後の戻りを少なくするDevOpsがあるのだと。ビジネスと開発と運用が協力して、早く頻繁に安全にサービスを提供・改善していく世界を構築できたら、それがDevOpsだった、という夢を抱いています。

DevOpsを実践する環境を考えるには、Pivotalのやり方が面白い

松下 康之(まつした やすゆき)
フリーランスライター&マーケティングスペシャリスト。DEC、マイクロソフト、アドビ、レノボなどでのマーケティング、ビジネス誌の編集委員などを経てICT関連のトピックを追うライターに。オープンソースとセキュリティが最近の興味の中心。

この座談会をしてみたかったきっかけは、Pivotalのサンフランシスコのラボを訪れた体験にあります。オープンなスペースに細長い島状に配置されたデスクと椅子、大きめのモニターに向かってプログラマたちが隣り合って話をしながらキーボードを叩く。大きなフロアの一部では勉強会のようなものが開かれていてプロジェクターから映しだされたエディタの画面を見ながら何かの解説が行われていました。その脇にある大きめのキッチンではインド料理のケータリングが広げられて、エンジニアと思わしき人たちが紙皿に盛った料理を食べながら質疑応答に参加したりしています。そう、なぜか全体的にザワザワしているのです。これまで筆者が欧米のIT企業で見てきた様子とだいぶ違います。筆者はアメリカの企業に在籍していたことがありますが、そこでは1人ずつ個室が用意され、個人のスペースが確保されていました。なのに最新のアジャイル開発をしているこのエンジニアたちは昔の日本のオフィスのように隣り合って話をしながら開発をしています。これはかつての日本のオフィスのようではないか——プログラマたちが話をしながら開発する、それがアジャイル開発にそしてDevOpsにつなげられるのか？　そんな疑問を日本の企業にぶつけてみたい、それがこの座談会のきっかけでした。TKCとU-NEXTはソフトウェアをビジネスに活かしているといっても両極端と言っていいでしょう。ビジネスを変革するソフトウェア開発、システム運用はどうしたら良いのか？——今回の座談会が、わずかでもIT部門の皆さんの手がかりになればと思います。

▼Pivotalジャパンオフィス内にあるオープンキッチンスペース

▼卓球台も設置されており、気分転換にいつでもプレイできる

C o l u m n

特別コラム

なぜ、PivotalはDevOpsをリードできるのか?

Pivotalの成り立ちとは

Pivotalという会社名に読者のみなさん、馴染みがないと思いますが、Pivotalのミッションステートメントは「世界のソフトウェアの在り方を変えましょう」という高邁(こうまい)なものです。Pivotalは2013年の4月にEMC、VMwareの複数の事業部をスピンオフして統合し、いわゆる「Cloud Native Applicationを実現する」会社として設立されました。2016年1月現在で2千人強の社員と2千社以上のお客様に製品やサービスを提供しています。主なソフトウェア製品はPaaS(Platform as a Service)を実現する"Pivotal Cloud Foundry"とHadoop、DWH(Data Warehouse)、インメモリデータグリッド製品などを統合した"BDS(Big Data Suite)"です。

Pivotalのビジネスの柱は何か?

Pivotalにとって戦略的に重要なアジャイル開発の支援サービスを提供するPivotal Labsがあります。Pivotal Labsは現在Pivotalの一部門ですが、その生い立ちは古くて、1989年にRob Mee(現在のPivotalのCEO)が設立した会社です。Kent Beckが中心となり「アジャイル開発宣言」が公表されたのが2001年ですから、その前にPivotal Labsは誕生していたわけです。現在、東京を含め世界16ヵ所で総勢600名の技術者がお客様と一緒にソフトウェアを開発しています。PaaS製品もBig Data製品も魅力的に価値のあるソフトウェアを創造するための道具でしかありません。最も重要なことはいかに魅力的でビジネス価値の高いソフトウェアを作り上げるかです。

Pivotal Labsはスキル移転の場

Pivotal Labsは単にアジャイル開発の専門集団というだけでなく、リーンスタートアップとアジャイル開発を統合したコンセプトと実践的な方法論でほかに類のない価値を世界の大企業に提供している専門家集団です。ちなみに2016年に入ってからですが、日本でもベストセラーになった『リーンスタートアップ』の著者でリーンスタートアップの第一人者のEric Riesと戦略的パートナーシップを発表しました。

アジャイル開発という言葉の概念は広くて、人それぞれのとらえられ方をされていると思いますが、Pivotal Labsの特徴はXP(Extreme Programming)、お客様とPivotal Labsのエンジニアがペアーで開発をする「ペアープログラミング」、アプリケーションを開発する前にまずはテストケースを開発する、TDD(Test Driven Development)など、まさに「アジャイル開発宣言」で宣言したすべてを徹底的に基本に忠実に実施するところでしょう。それに加えてPivotal Labsのオフィスにお客様に来ていただいて作業をすることでPivotal Labs流の仕事の進め方をお客様にOJT的にスキル移転することです。Pivotal Labsのビジネスゴールはお客様がPivotal Labsの助けを不要として自らの力だけで自社内でアジャイル開発を実践できるようになっていただくことです。

ウォーターフォールからアジャイルへの実践的転換

アジャイル開発宣言の原点を踏まえてソフトウェア開発をあらためて考えると、これまで(現在もですが)大企業で主流であるウォーターフォール開発という方式は、完成度の高い「要件定義」ができることを大前提に考えられた開発プロセスととらえることができると思います。一方アジャイル開発はビジネス要件は不確かで変化し続けるものということを前提にしていかにして価値の高いソフトウェアを作りあげるかということで提唱されている取り組みだと思います。このようにとらえると「ウォーターフォール」の考え方のほうが傲慢で、「アジャイル開発」の考え方のほうが「謙虚」に思えます。「ウォーターフォール」の考え方は製造業における同一製品を機械的に大量生産する発想に似ていて、開発に携わる個人の人格、尊厳、自主性などを排除したうえでいかにソフトウェアを一定の品質で生産するかということを実現する

ための手法に見えてきます。一方、アジャイル開発の取り組みは開発に携わるプロダクトマネージャ、デザイナ、デベロッパの一人一人の創造性、自主性を尊重していてソフトウェアを開発することがエンジニアにとって「創造的で楽しい」、「自己実現の実感をもたらしてくれる」取り組みではないかと思います。ソフトウェア開発者としてプロフェッショナルたらんとすれば、魅力的なソフトウェアを開発するメンバの一員となり、自らが開発したソフトウェアが一般消費者あるいは企業内ユーザに受け入れられて活用されてこそ達成感も生まれてくるわけで、そのための自己研鑽は必要なことだと思います。

PivotalとDevOps

PivotalのCEOのRob Meeは2015年に来日した際に日本のお客様に「シリコンバレーの本質は単なる地名ではなく心の持ち方です」とお話しました。新しい取り組みをするためにシリコンバレーにオフィスを開設することが重要なのではなくて「リーンスタートアップ」、「アジャイル開発」などのシリコンバレー精神と具体的な手法を社内に取り込み定着させることが重要だという日本に対する応援メッセージだと私は理解しています。

ここまでアジャイル開発に関して思うところを述べさせていただきましたが、今回のテーマはDevOpsです。DevOpsの全体の文脈から考えるとアジャイル開発は非常に重要ではありますが、開発フェーズに限った話でありDevOpsの一部でしかありません。アプリケーションの開発が俊敏にできたとしても、その前の過程のインフラの用意、ミドルウェアなどのアプリケーション開発にとって必須の作業に数週間、数ヵ月かかっていたり、アプリケーションの開発が完了しても、アプリケーションを本番リリースするまでに期間を要するようであればビジネスの価値は失われてしまいます。

継続的デリバリー／インテグレーションの重要性

昨今DevOpsの議論に注目が集まったり、アジャイル開発という四半世紀以上の積み重ねのあるテーマが今日熱く語られるようになった背景は「継続的デリバリー」(CD：Continuous Delivery)、継続的インテグレーション(CI：Continuous Integration)についてビジネス的な重要性の認識の高まりがあるのではないでしょうか。CI、CDはGoogle、Amazonなどインターネットジャイアントだけでなく Uber、AirBNBなどのソフトウェアを武器に従来の業界に破壊的インパクトを与えている企業では当たり前のように実現されています。従来型企業においても彼らと同じスピードでソフトウェアを武器とした製品、サービスを開発するスキルとプロセスを企業内に築き上げないとビジネス優位を維持できない、最悪なシナリオとしては市場からの撤退という事態すら起きかねないという認識が欧米の経営者に明確にあります。

CD/CIよりも重要なこと

今日のグローバル経済の状況を考えれば日本企業においても多少の時間差はあれ同様のことが起きるはずです。金融の分野では米国Googleが個人の損害保険のビジネス市場に参入していますし、ご存じのようにSquareはすでに日本でもサービスを展開しています。CD、CIを実現するために必要なコンポーネントの一部としてアジャイル開発があり、アプリケーションの開発に必要なハードウェアリソースを直ちに用意できるためにIaaS(Infrastructure as a Service)が存在し、ミドルウェアを含めて開発、テスト、実行環境のデプロイを支えるためにPaaS(Platform as a Service)が存在し、CD、CIの全体プロセスを支えるためにソースコード変更管理などのさまざまなツール群がすでに市場に投入されてきているととらえると全体が見てえくるのではないでしょうか。DevOpsを実現するためのテクノロジや方法論はすでに利用できますがそれらは道具でしかありません。最も重要なのはそれを実現するビジネスコミットでしょう。テクノロジや方法論の導入と同時に社内プロセスの変更、組織の見直し、大げさになるかもしれませんがIT部門だけではなく、ビジネス部門や人事などの関連部門含めた企業文化の変革が求められるテーマがDevOpsであり、企業の競争力を強化するという大きなテーマです。

RDB性能トラブルバスターズ奮闘記

原案 生島 勘富(いくしま さだよし) info@g1sys.co.jp ㈱ジーワンシステム
構成・文 開米 瑞浩(かいまい みずひろ) イラスト フクモトミホ

第2回 SQLのための仕様書は書くだけムダ

前回、システムのレスポンス悪化の原因を調査し、本来データベース(DB)側で処理すべき集計をアプリケーション(AP)側で処理していたことを突き止めた生島氏と大道君。今回は、設計工程までさかのぼって真の原因を追求します。

登場人物紹介

生島氏
DBコンサルタント。性能トラブルの応援のため大道君の会社に来た。

大道君
浪速システムズの新米エンジニア。素直さとヤル気が取り柄。

五代氏
大道君の上司。プロジェクトリーダーでもある。

複雑なSQLを書いてはいけないのか?

それはXX年前の春、大阪城公園で咲き始めた桜が窓から見えるオフィスでのこと。浪速システムズ㈱(仮名)が開発中のECサイトのレスポンス問題を解決するため、私、生島は若手エンジニアの大道君と話をしていました。

問題は商品一覧画面での在庫数表示に時間がかかるというもので、当連載の第1回では、その原因を探りながら大道君とともに次のようなポイントを確認したところでした。

(1)本来DB側で処理すべき集計をAP側での多重ループ処理で行っているのが遅くなる原因
(2)AP側での多重ループ処理をやめて、DB側で集計させるように全力で考えるべき
(3)DB側で集計させようとすると、発行するSQL文はどうしても複雑なものになる

「そうなんですね……」と大道君。
「気になることは遠慮せんと聞いてな」
「実は、複雑なSQL文はできるだけ書くな、と前の上司から言われたことがあるんです。もう辞めちゃった人なんですけど」
「そら困った指示やな。本来は、複雑なSQLも使いこなせるようにキッチリ勉強せえ!が正しいんやけどな」

ハッキリ言えば「複雑なSQLを書くな」というのは、よくある本末転倒な間違った指示です。実際はSQLをきちんと理解せずに適当に書くから遅いうえに保守もできないものができるだけのことで、「複雑なSQL」自体を敵視するのは筋違いです。

しかし似た考え方をする技術者は大勢います。手続き型言語の発想では複雑なSQLを使うとわけがわからなくなるため、SQLを使うこと自体を避けるようになり、できるだけAP側で処理しようとするわけです。多重ループはその結果として起きる問題です。

と、そこへ新しい声が響きました。大道君の上司でプロジェクトリーダーの五代さんです。
「どうも〜、はじめまして、五代です。よろしゅうお願いします」

五代さんを含めて話を聞いてみると、いろいろわかりました。今回の案件は、すでに稼働しているECサイトのマイナーな仕様変更の1つ。大道君の前の上司が基本設計とプログラム仕様書を書いて急に辞めてしまったそうです。それ

でも、在庫表示を加えるだけのちょっとした変更なので簡単と思っていたら、性能トラブルに見舞われて困った困った、という状況だったそう。五代さん自身は顧客との折衝、要求仕様の調整といった役割が主で、DB技術には詳しくないので、ぜひ大道君を指導してやってほしい……とのこと。これは「間違った考え方に凝り固まった上司がいなくなり、残ったのは学ぶ意欲のある若者と、それをサポートする姿勢のある上司」という組み合わせなので、案外、災い転じて福となすかもしれません。大道君なら急速に成長できそうです。……よし、やってみよう。

SQLの発想で考えるとは?

「ほな聞くけど、俺がさっきSQL書くとこ見てて、どう思った?」

「えっ……あんな複雑なのをスラスラっと書いてしまうなんて、すごいなあ、と」

ちなみに**リスト1**に示したのがそのSQL文です。確かに初心者には複雑に見えそうです。

「いやいや、君もできるようになるんやで!」

「慣れればできるようになりますか?」

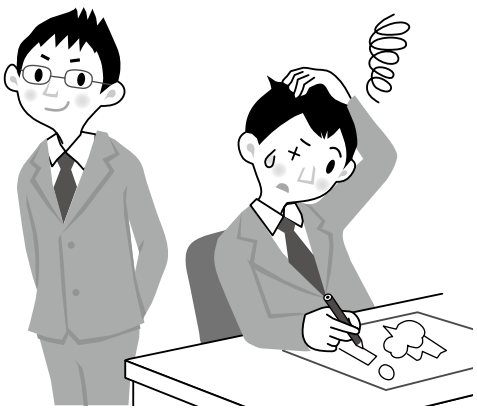
「慣れだけじゃあかんな……」

「ほかに何が……?」

「そやな……ほな、まず、この理論在庫算出がどういう処理なのか、イメージ湧くような絵を描いてみ?」

「え、絵? ですか?」

「理論在庫はこう出すんです、いうて、素人さ



▼リスト1 理論在庫算出SQLサンプルコード

```
SELECT
       商品マスタ.商品CD
       , 商品マスタ.商品名
       , 商品マスタ.定価
       , (SELECT COALESCE(SUM(棚卸数), 0) FROM 在庫データ WHERE 在庫データ.商品CD = 商品マスタ.商品CD)
       + (SELECT COALESCE(SUM(入庫数), 0)
          FROM 入庫データ
          WHERE 入庫データ.商品CD = 商品マスタ.商品CD
              AND NOT EXISTS (SELECT * FROM 在庫データ
                     WHERE
                     在庫データ.商品CD = 入庫データ.商品CD
                     AND 在庫データ.倉庫CD = 入庫データ.倉庫CD
                     AND 在庫データ.棚卸日 > 入庫データ.入庫日)
       )
       - (SELECT COALESCE(SUM(出庫数), 0)
          FROM 出庫データ
          WHERE 出庫データ.商品CD = 商品マスタ.商品CD
              AND NOT EXISTS (SELECT * FROM 在庫データ
                     WHERE
                     在庫データ.商品CD = 出庫データ.商品CD
                     AND 在庫データ.倉庫CD = 出庫データ.倉庫CD
                     AND 在庫データ.棚卸日 > 出庫データ.出庫日)
       )
    AS 理論在庫数
FROM
       商品マスタ
WHERE
       画面からの検索条件(例えばカテゴリーで絞る);
```

んに説明するときに使えるような絵や」

「えっ……と」

「ダジャレやのうて、絵！」

くだらないボケツッコミはやめておきますが、少し考え込んで大道君が描いたのはざっくり言って**図1**のようなものでした。

「これは、その、仕様書を思い出して、そのイメージで書いてみたんですけど」

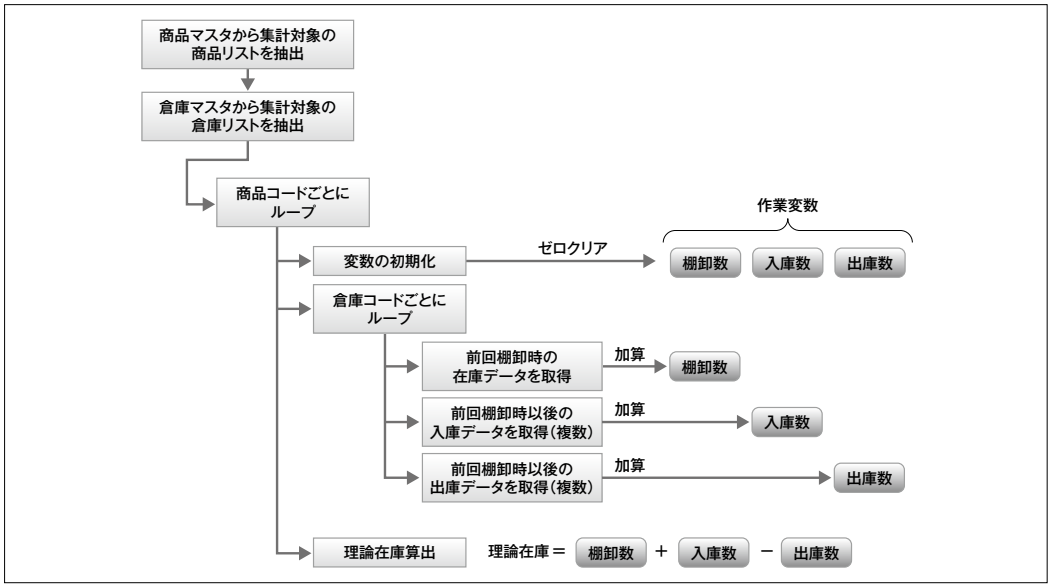
辞めてしまった上司が書いたという仕様書は**リスト2**でした。こうした仕様書がたいへん多いのですが、実はSQLには役に立たない、無駄なものです。

「やっぱりこれは手続き型言語のループで書く発想やなあ。SQLを使うときはSQLの発想で考えたほうが楽やで。そうすれば今さっき見たようにスラスラ書けるようになる」

「SQLの発想って……」

「まあそんだけ言われてもわからんよな。ほ

▼図1　理論在庫算出処理イメージ



▼リスト2　理論在庫算出処理仕様書

```
在庫数は倉庫ごとに管理している。
商品検索をされたとき、以下のとおり、理論在庫数を算出して表示する。

1  画面からの条件で商品マスタを抽出する。　※
2  倉庫マスタを読み込み、配列に保存する。　※
3  抽出された商品マスタをループする。
   3.1  倉庫マスタ配列をループする。
      3.1.1  在庫データを以下の条件で抽出し、棚卸数を変数棚卸数に加算する。　※
             在庫データ.商品CD ＝ 商品マスタ.商品CD
             AND在庫データ.倉庫CD ＝ 倉庫マスタ配列.倉庫CD
      3.1.2  入庫データを以下の条件で抽出し、入庫数を変数入庫数に加算する。　※
             入庫データ.商品CD ＝ 商品マスタ.商品CD
             AND入庫データ.倉庫CD ＝ 倉庫マスタ配列.倉庫CD
             AND入庫データ.入庫日 ＞＝ 在庫データ.棚卸日
      3.1.3  抽出された入庫データを変数入庫数に加算する。
      3.1.4  出庫も入庫と同様に処理し、変数出庫数に加算する。　※※
   3.2  倉庫マスタ配列のループがなくなったとき、理論在庫数を算出し、画面に出力する。
        理論在庫数 ＝ 変数棚卸数 ＋ 変数入庫数 － 変数出庫数

※ の部分にはSQLのイメージも入ります。
※※ の部分は、実際は「同様に処理」と省略せずに書きます。
```

な、今度はそれを説明しよか！」

カタマリを切り出す考え方を身につける

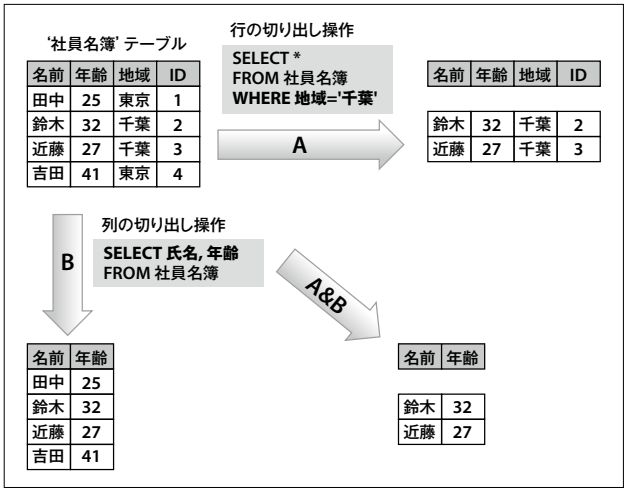
「SQLで考える」ための発想術として、まず大事な原則は「カタマリを切り出す」ように考えることです。在庫の話だと複雑になるので、簡単な社員名簿の例を**図2**に書きました。名前、年齢、地域、IDが載っている簡単な「社員名簿」テーブルが大元の「カタマリ」です。このカタマリから「一部の行を切り出す」のがAの操作で、切り出す行をWHERE句で指定します。一方、「名前、年齢」という「一部の列を切り出す」のがBの操作で、切り出す列をSELECT句で指定します。もしAとBの両方をやると右下のようなカタマリが切り出せます。

カタマリを切り出したものもやはりカタマリ、つまりテーブルになっているので、さらにSELECT操作をすることができる、ということに注意してください。それを重ねていくことで、ループなしでの複雑なデータ処理が可能になります。具体的には**図3**を見てください。

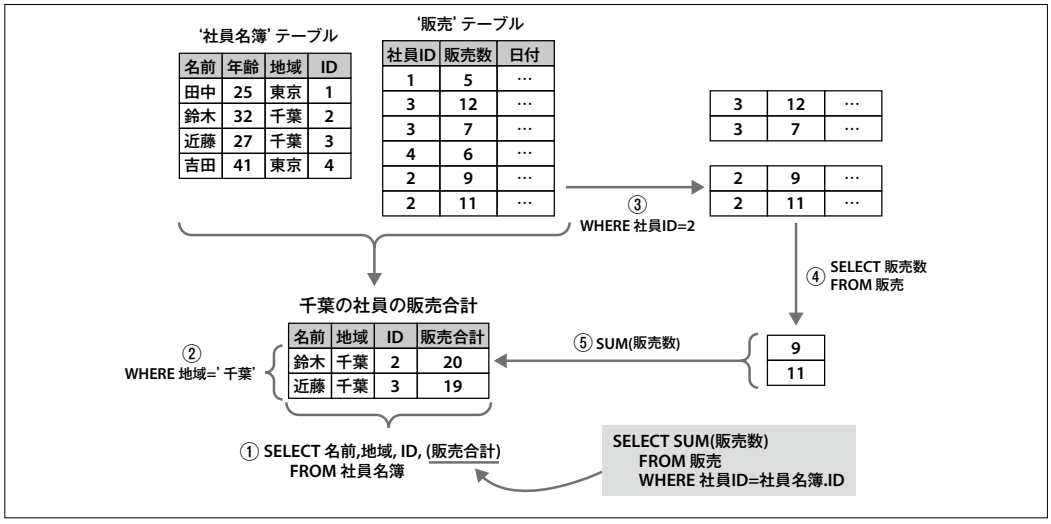
社員の販売数を記録した「販売」テーブルを集計してみましょう。たとえば千葉の社員のみの販売実績を集計して、**図3**の下部にある「千葉の社員の販売合計」テーブルがほしいとします。ベースになるのは「社員名簿」ですので、①FROM句で「社員名簿」、SELECT句で「名前,地域,ID」を指定します。「販売合計」は社員名簿には存在せず、集計して出すため、いったん括弧に入れておきます。また、②WHERE句には「地域='千葉'」を指定します。

次に社員別の「販売合計」を出します。社員ID=2の場合を例示すると、③「販売」テーブルか

▼図2　行と列のカタマリ切り出しイメージ



▼図3　サブクエリーを使った切り出し・集計処理イメージ



ら「社員ID=2」の行を切り出し、④そこから「販売数」の列を切り出し、それを⑤「SUM(販売数)」で集計すれば販売合計になります。この③④⑤を1つのSQL文として組み立てると右下のSELECT文になりますので、これを①の括弧内の「販売合計」の欄に入れてやれば、サブクエリを使った集計処理になるわけです。

「あ、そうか……つまり、①②も③④も、全部カタマリの切り出しをやってるんですね？」

「そういうこと！」

「⑤は集計ですけど、これだって1行1列のカタマリを作ってると思えば同じか……」

「そやね」

「理論在庫の算出でやってることも、カタマリ切り出しの段数が増えるだけで、本質的にこれと同じですよね……？」

「そのとおり！」

全体像のイメージを持てばSQLは書ける

「なんだ、そうか、単純な話なんだ……こう書けば全体像が見えますね。そしてSQL文にそのまま対応するんですね、これ？」

「そこなんよ！」

大道君、大事なことに気がついてくれました。脳内に全体像のイメージを描ければ、1つ1つの「カタマリを切り出す」ステップをそのままSELECT句やWHERE句などのSQL文の句として表現することができます。句ができれば後はそれをペタペタと組み合わせるだけですので、一見どんなに複雑に見えても中身は単純なSQL文として理解できるのです。

「なんだかできそうな気がしてきました」

「一見複雑そうなSQLでも、1つ1つのステップは単純なんよ。イメージをもってトップダウンで考えるのに慣れれば、頭の中でサクサクっと組めるようになるよ」

「なんか、元気が出てきました(笑)」

それは私も同じでした。筋のいい若者が成長していく姿は本当に気持ちがいいものです。

「でも、そうすると……**リスト2**みたいな仕様書って書く意味あるんですか？」

仕様書は書いてもムダ?

鋭い疑問です。実際そこが問題なのです。

「率直に言えば、意味ないね、というか、むしろマイナスしかない」

「じゃあ……どんな仕様書なら役に立つんでしょうか？」

仕様書と言ってもいろいろですので、意味あいを整理するために**図4**を見てください。左側の「成果物」の欄は、成果物として要求される文書類の名前、「例」でそこに書く情報の例を示しました。「役割」の欄にあるのは、それぞれを象徴する言葉として私が選んだものです。

「要求仕様書」のおもな役割は、利用者にとっての「ゴール」を示すことです。今回の案件では「在庫数量の表示」がゴールです。そしてプラモデルがパーツを組み立てて作るように、そのゴールを達成するために必要な手順や情報、たとえば在庫数量なら「棚卸数」「入庫数」「出庫数」などは「パーツ」とみなせます。システムの利用者はパーツを意識しませんが、開発者は考える必要があるので、その仕様を記載するのが「基本設計書」です。今回のような小さな変更案件では、要求仕様書と基本設計書を区別せずまとめて書く場合もあります。

また、「パーツ」を実装するための「アルゴリズム」を書いた仕様書を、しばしばプログラム仕様

▼図4 「仕様書」の役割

成果物	例	役割
要求仕様書	商品ごとに在庫数量を表示すること	ゴール
基本設計書	在庫数量は全倉庫の棚卸在庫・入庫・出庫データを集計して算出する	パーツ
プログラム仕様書（詳細仕様書）	入庫データを読み出して作業変数に格納し加算する処理を件数分行う……	アルゴリズム
ソースコード	static void calculate(..){	コード

書や詳細仕様書のように呼びます。

最後にそのアルゴリズムをソースコードとして実装して初めてシステムが稼働します。

そこまで説明して、大道君に1つ尋ねてみることにしました。

「さて、SQL文の役割は**図4**のゴール、パーツ、アルゴリズム、コードのどれだと思う？」

「えっ……コードですよね？」

という回答。予想どおりです。そこが実は極めてよくある誤解なのです。

「じゃあ、**図3**はどれに該当すると思う？」

「**図3**は、千葉の社員の販売合計がゴールで、そこに至る手順の部分はパーツ……ですか？」

「そのとおり。**図3**はゴールとパーツに該当する。ところで、**図3**の各部ってそのままSQLの句に置き換えられたよね？」

「……そうですね」

「てことは？」

SQL自体が仕様書のようなもの

「あっ……まさか？」

そのまさかです。実はSQL文というのは「コード」ではなく、ゴールとパーツを表している、つまり要求仕様／基本設計レベルのことを表していると考えたほうがいいのです。

「え、じゃあ、**リスト2**は？」

リスト2のような仕様書は、「アルゴリズム」を書いています。しかしコードとして動くSQL自体がゴールとパーツの役割を兼ねているので、実は中間の「アルゴリズム」はそもそも書く必要がありません。つまり、**リスト2**のような「アルゴリズム」を示す仕様書は、SQLを書くための仕様書としては意味がないのです。

すると、そこで今まで黙っていた五代さんが大声を上げました。

「な、なんやって‼」

「実はもともとSQLの設計思想の1つが、アルゴリズムをなくすことなんですぁ。残念なことに、**リスト2**のレベルの仕様書を書けば書くほど、SQLの実力を潰してしまいます。

デスマーチになったとき、『仕様書の不備(仕様書が足りなかった)』という理由にされて、あれも書け、これも書けとなるでしょ。それこそ工数が増えてパフォーマンスが落ちる最大の原因ですぁ。

その証拠に、私が直したときはテーブル定義(ER図)とソースしか見てへんし、逆に、直したSQLが出てくる詳細設計書は書かれへんでしょう？」

「ほんまかいな……」

リスト1、2をじっくり見た五代さんは

「確かに、このSQLのために**リスト2**みたいな仕様書を書くのは無駄やな……。ちゃんとSQLを使えば、仕様書が減って、ソースコードが短くなって、テストパターンが減って……、工数的にもめっちゃおいしいやん！」

と、今後の開発では「無駄な仕様書は書かない」方針に同意してくれました。必要なのは**図3**のように「カタマリを切り出す」イメージがわかる資料と、SQL文そのものです。

「よし……ほな、大道君やろうか？　RDBとSQLをきちっと勉強して、自分でトラブル解決できるようになろうな！」

「はい！」

とはいえ、「詳細仕様書を書かない」ことを実践するためには社内ルールを変える必要があり、社内の意識を変える必要があり、たいへんな困難が待ち構えているのでした。SD

presented by
Japan Android Group
http://www.
android-group.jp/

コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

第4回 MIDI音源から音を出す

Androidは世界で出荷される8割のスマートフォンに搭載※される、事実上スマートフォンの標準OSです。このOS上で動くアプリケーションを開発することは、自分のアイデアを世界に発信し、最も多くのスマートフォン上で動かしてもらえる可能性がある行為です。このAndroidの最新情報と、それを取り巻くコミュニティを知って、魅力的な開発をはじめませんか？

※IDC Worldwide Mobile Phone Tracker, August 7, 2013

嶋崎 聡（しまざき さとし）
よこいど／
日本Androidの会 横浜支部
Twitter @sato_c

今回のテーマ

横浜支部の嶋崎です。今回はAndroid 6.0からAPIレベルで対応が行われたMIDIデバイスのコントロールを使ったアプリを紹介します。iOSではかなり早い段階からMIDIデバイスへの対応があり、Androidはその点で一歩遅れていた印象がありましたが、今回の対応によりほぼiOSと同様の機能への対応が行われています。まずは対応の第一歩ということで、外部MIDIデバイスをつないで使う1トラック16ステップシーケンサを作りました。MIDI OUTのみの簡単な構成ですが、音を出す部分を中心に作っています。

MIDI対応

6.0以前でMIDIを利用する場合、USB Host機能を使ってドライバで対応されている例はありました。6.0からは「android.media.midi」として、APIが用意されました。

android.media.midiクラスでは、USB接続でのMIDI IN/OUTとBluetooth Low Energy（以下、BTLE）を使ったMIDI over Bluetooth LE（以下、MIDI BTLE）も利用できます。MIDI BTLEは低遅延＋ケーブルレス接続ということで注目されている技術で、iOSのCoreAudioの目玉として発表されてから現在まで、徐々に対応するハードウェアが増えてきています。Androidでも正式対応したことで今後ますます増えてくるのではないでしょうか。

このほかに注目したい点として仮想MIDIデバイスを使ったソフトウェアによるシンセサイザー（以下、ソフトシンセ）の作成支援やソフトウェア間でのMIDIデータのルーティングなど、現状考えられる必要な機能はひととおり装備されています。とくに内部のルーティングが可能になったことで、アプリ間の同期が可能になります。アプリ側の対応によるところはありますが、アプリに搭載されているソフトシンセの外部利用ができたり、単体アプリでPCのVSTi（Virtual Studio Technology instrument）プラグインのような音源が登場するかもしれません。

簡単なシーケンサを作る

AndroidからMIDIデバイスをコントロールするとしたら、ということでシーケンサを作りました。さすがに最初から規模の大きいものはいろいろと無理だったので、無難なところから1小節4拍（4/4）を16分音符で表現するリズムマシンのようにしました。

テスト用機材

今回、筆者が用意した環境はNexus 5、ポケッ

トミク NSX-39、USB OTG Hubです。

ポケットミクは、音源チップと電源、スピーカーが一体化された音源です。数年前に雑誌[注1]の付録として販売されていたときに入手したもので、これだけで音が出るので実験にちょうどいいものです(**写真1**)。

この音源にはeVocaloidとGM音源がワンチップになっているNSX-1というチップが使われています。eVocaloidは、初音ミクなどのVocaloid技術を1チップで実現したものです。雑誌のほうも販売が続いているようですので、興味がある方は探してみてください[注2]。

Nexus 5との接続には、Hubを通してmicro USBケーブルを使います(**写真2**)。音源の電源をUSB経由で供給する必要があるため、今回は端末の充電に対応したHubを使いました。これでNexus 5本体にも充電しながら利用できます。Hubがない場合でもUSB OTGケーブルとmicroUSBケーブルを使えば接続できます。

シーケンサ仕様

音源をコントロールするのであれば音階の指定ができるほうがいいのですが、今回はまず基本となる部分を作りたかったのでリズムマシン風のものにしました。リズムマシンの本体での打ち込みは、1小節を16分音符に分けたものになっています。1拍を4分割した1つ分を1ステップと数えて、4ステップ×4拍で16ステップです。再生する音はドラムでも比較的わかりやすいハイハットの音にしました。

ポケットミクはGM音源を内蔵しています。GM音源ではリズムはMIDIチャンネルの10に割り当てられていますので、このシーケンサでも出力チャンネルは10としました。

使う音を画面で変更できるようにもしたかったのですが、今回はクローズハイハット(キーナンバー42／ファ#)固定で出力しています。音を出すときに鍵盤を押した強さ(ベロシティ)が必要になりますが、ハイハットなので最大の127で固定しました。ピアノなどの楽器音に変更する場合は、指定する楽器によっては100まで落としたほうが音色の変化が起きずに自然な音が出ると思います。こうした値についてはプログラムで定数定義していますので、違う音を出すには内容を変更してビルドしなおす必要があります。

画面仕様

画面構成は次ページの**図1**のようになっています。画面上部から、16ステップ分のスイッチと演奏中の位置を示すマーカー、5段階のテンポ指定、再生／停止ボタンになっています。停止ボタンは1回押すと一時停止、もう一度押すと完全に停止(演奏マーカーが最初に戻る)となっています。

注1) 『大人の科学マガジン 特別編集　歌うキーボード　ポケット・ミク』(学研マーケティング刊)

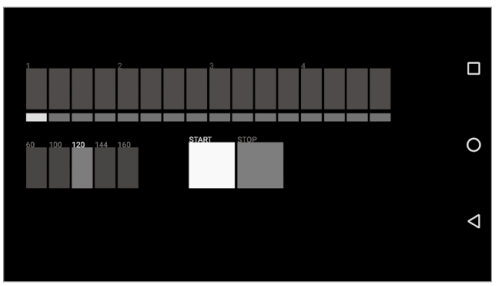
注2) 出版社のオンラインショップからも購入できるようです。
http://otonanokagaku.net/nsx39/

▼写真1　ポケットミク NSX-39の本体

▼写真2 USB OTG Hubを介して実験用に機材を接続

再生ボタンを押すと指定されているテンポでマーカーが動き、マーカーのある位置のスイッチ状態によって音を鳴らします。音を鳴らすために必要なMIDIデータはキーオンのコマンドにキーナンバーとベロシティで、これらを組み合わせたものをデバイスに対して送信します。ドラム系の音を使っているので、止めるコマンド(キーオフ)はとくに送信していません。楽器音に変更する場合は、キーオンしたあとでしばらく待ってからキーオフを送る処理が必要になることもあります。

▼図1　シーケンサの画面構成



プログラムの解説

今回作るものは、アプリ単体ではなくUSB Hostで外部デバイスを利用するので、ADB(Android Debug Bridge)をネットワーク経由で使う必要があります。手順についてはAndroid Developerサイトなどを参照するのがいいですが、簡単にコラムにもまとめましたので参考にしてください。

なお、プロジェクトファイルはGitHubに載せてありますのでcloneして使ってください[注3]。

画面表示

画面表示にはSurfaceViewを使いました。描画については、drawというメソッドの中でCanvasの図形および文字描画メソッドを利用しています。基本は四角で描画していて、各パーツの色の切り換えは複数のPaintを状況によって切り換えています。

注3) https://github.com/sato-c/sd_midiplay

COLUMN

ADBをネットワーク経由で利用する手順

PCとAndroid端末が同じネットワーク内にあることが前提です。次の❶～❼を実行します。

❶通常どおりUSBケーブルでPCと端末を接続します。
❷コマンドラインから、adbコマンドを入力します(ポート番号は5555～5559の範囲で変更できます。変更した場合は読み替えてください)。

```
$ adb tcpip 5555
```

❸これでTCP/IPのポート5555で接続できる設定になったので、adbサーバを再起動します。

```
$ adb kill-server
$ adb start-server
```

❹Android端末のIPアドレスを調べます(端末設定の端末情報や本体設定でIPアドレスが確認できます)。
❺コマンドラインから、次のadbコマンドを入力します。

```
$ adb connect [IPアドレス]:5555
```

❻コマンドラインから、次のadbコマンドを入力します。端末が表示されるのを確認してください。

```
$ adb devices
```

❼あとはAndroid Studioの端末選択画面でネットワーク経由の端末が表示されれば利用できます。

描画およびタッチ位置は、表示物の座標を起動時に一括生成して使っています(**リスト1**)。表示に倍率補正が入っていると使えませんが、Nexus 5ではそうした補正が入っていなかったのでこの方法を使いました。表示とタッチのずれがでないおかげでプログラムがわかりやすくなりました。

MIDIデバイスの準備

MIDIデバイスの準備や設定については、android.media.midiの解説に載っている内容を順に実装していきます[注4]。**リスト2**が実装したプログラムです。アプリが起動して画面の準備が終わってから、MIDIデバイスの準備をする部分です。

処理の流れとしてはほかのAPIの準備と変わりません。MIDI APIが対応しているかを調べて(**リスト2-001行目**)、MIDIマネージャサービスを取得します(**リスト2-002行目**)。

次にマネージャが持っているデバイスの接続状態を取得します(**リスト2-003行目**)。起動前

注4) http://developer.android.com/intl/ja/reference/android/media/midi/package-summary.html

▼リスト1　タッチ位置の検出

```
001: private int checkTouchPosition( ArrayList<Rect> rectList, int x, int y ) {
002:     Rect r = rectList.get(0);
003:     if ( r.top <= y && r.bottom >= y ) {
004:         for ( int i = 0; i < rectList.size(); ++i) {
005:             r = rectList.get(i);
006:             if ( r.left <= x && r.right >= x ) {
007:                 return i;
008:             }
009:         }
010:     }
011: }
```

▼リスト2　MIDIデバイスの確認と準備

```
001: if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_MIDI)) {
002:     MidiManager manager = (MidiManager)context.getSystemService(Context.MIDI_SERVICE);
003:     MidiDeviceInfo[] instruments = manager.getDevices();
004:     if ( instruments.length > 0 ) {
005:         for ( int i = 0; i < instruments.length; ++i) {
006:             MidiDeviceInfo instInfo = instruments[i];
007:             Bundle properties = instInfo.getProperties();
008:             String manufacturer = properties.getString(MidiDeviceInfo.PROPERTY_NAME);
009:             if ( instInfo.getInputPortCount() > 0 ) {
010:                 manager.openDevice(instInfo, new MidiManager.OnDeviceOpenedListener() {
011:                     @Override
012:                     public void onDeviceOpened(MidiDevice device) {
013:                         useMIDI = true;
014:                         inputDevice = device;
015:                         inputPort = device.openInputPort(0);
016:                         noteBuf = new byte[32];
017:                         noteBuf[noteBufSize++] = (byte)(0x90 + MIDI_CH - 1);
018:                         noteBuf[noteBufSize++] = MIDI_NOTE;
019:                         noteBuf[noteBufSize++] = MIDI_VELOCITY;
020:                     }
021:                 }, new Handler(Looper.getMainLooper()));
022:             }
023:         }
024:     }
025: }
```

から接続されているデバイス数分、情報を取得できます(**リスト2-004、005～024行目のループ**)。今回は接続時にデバイスがある場合のみを想定しています。

シーケンサアプリを作るときはアプリ起動中にMIDIデバイスをつないだり外したりする場面が出てきますので、MIDIマネージャのコールバックを設定して、デバイスの状況を取得するよう実装します。コールバックを使うことでデバイスの状況が変わるたびにonDeviceAdded/onDeviceRemovedが呼び出されます。これらのイベントで接続、切断の処理を行うことになります(MidiManager.DeviceCallback)。コールバックが呼ばれるときにはMidiDeviceInfoクラスでデバイス情報が送られます。その情報をもとに接続、切断の確認や準備を行います。おおざっぱに分けるとonDeviceAddedイベントではデバイスの初期化、onDeviceRemovedイベントではデバイスの破棄を行います。どちらの処理も演奏している、していないにかかわらず、影響がでないようにする必要がでてきます。

取得したデバイス情報は、コールバックでの処理と同じようにMidiDeviceInfoクラスです。この情報を元にgetInputPortCountメソッドでMIDI INポートの数を調べます。ポートが1つ以上あることがわかったら(**リスト2-006～009行目**)、getPortsメソッドでポートの配列を取得して、情報を保存します(**リスト2-010～021行目**)。今回は配列の最初のデータを使うよう、0を指定していますが、MIDI IN/OUTポートが複数ある場合は使うポートを決定するための処理が必要になるでしょう。また、MIDI OUTポートが必要な場合は、同じようにgetOutputPortCountでポート数を調べてポート情報を保存する処理が必要です。

これらのメソッドで使われているMIDIポートの指定方法は、APIの指定がAndroidデバイスのポートではなく、MIDIデバイス側のポートMIDI IN/OUTを基準にしています。そのため、アプリからの出力がinput、アプリへの入力がoutputとなっています。今回のプログラムもSDKサンプルと同じように命名していますので、この向きに注意してください。

使えるポートがあった場合はMIDIマネージャからデバイスのオープン処理をします。この処理は非同期で行われます。呼び出すときにオープンできた場合のためのコールバック関数とコールバック関数を処理するスレッドを指定します。とくに理由がなければメインスレッドで処理するように指定しておけばいいでしょう。接続、切断のコールバックを利用する場合はほかのスレッドを用意する必要がでてくるかもしれません。

▼リスト3　Handler処理

```
001: public void handleMessage( Message msg) {
002:     if ( playing && msg.what == PLAYSEQUENCE ) {
003:         long nexttime = (long)(60 * 1000 / tempoList[speedPosition] / BEATS);
004:         if ( keyOn[markPosition] == 1) {
005:             if ( useMIDI ) {
006:                 try {
007:                     inputPort.send(noteBuf, 0, noteBufSize);
008:                 } catch ( Exception ioException) {
009:                 }
000:             }
011:         }
012:
013:         draw(mHolder);
014:
015:         markPosition = (markPosition + 1) & 0x0f;
016:         sendMessageDelayed(obtainMessage(PLAYSEQUENCE), nexttime);
017:     }
018: }
```

デバイスのオープンに成功したら、利用するデバイスのinputポートの準備と保存をしてから、MIDIデータ送信のためにバッファを作成しています(**リスト2-013〜019行目**)。ここまでがMIDI出力を行うための初期設定となります。

定期的な呼び出し

MIDIデバイスの準備ができたら、定期的に画面更新とMIDIデバイスへの出力を行う必要があります。今回はHandlerでメッセージ処理する方法を使いました(**リスト3**)。この中ではMIDIデータの送信と描画、1ステップの待ち時間の計算を行っています。

1ステップの待ち時間をHandler内で実行しているのは、演奏中にもテンポ切り換えができるようにしてあるためです。音楽のテンポは1分間にどれだけの音符を演奏するかなので、60,000msをテンポで割れば1拍あたりの待ち時間が計算できます。今回は1拍を4個の音符に分割していますので、さらに4で割ることで1ステップあたりの待ち時間を計算しています(**リスト3-003行目**)。

MIDIデータの送信は、デバイスの準備の際に作ったbyteのバッファを送信しています。バッファとそのサイズが必要になるのでデータバッファを組み立てる際には気をつけてください(**リスト3-007行目**)。

描画については、SurfaceViewを使っているのでHandler内から呼んでも正常に画面が更新されます。演奏中は演奏位置マーカーを更新するために毎回呼び出しています。

また画面タッチでデータが更新された場合にも描画が呼ばれています。いつでも描画指定できるのでとくに描画のタイミングを計っていないため、データ更新で描画が行われた後にHandler側の処理でも描画が行われたり、その反対にHandler側で処理が終わったタイミングでタッチ側が描画を呼び出すといったことが起こるかもしれません。速度を気にする処理が増えてくると、どのような理由であっても描画処理が何度も呼ばれてしまうのはあまりうれしくありません。そこで、タッチ処理側は演奏中かどうか判定を行い、なるべく描画を省略するといった対策を行えば、必要ないタイミングの描画によるロスが減るでしょう。

まとめ

今回は1トラックのリズムマシンのようなシーケンサを作りました。出力しか使っていないため、APIの機能をすべて紹介できたわけではありません。実行時のつけ外しやMIDI入力など出力以外の機能については機会があればまとめたいと思います。

音楽アプリは、Androidでもシーケンサと音源が統合されたアプリは結構な数出ていましたが、外部の音源なども使えるものはiOSのほうが数が豊富でした。Android 6.0での対応が始まったことで、今後はAndroidでこうしたアプリが増えてくるでしょう。加えて、国内でNexus以外の高速な6.x端末や、Androidを組み込んだMIDIワークステーションなどが出てくるのではないかと期待しています。

MIDIと聞くと鍵盤や音源などの楽器を演奏するためのものというイメージが強いと思いますが、最近は照明などの舞台装置をコントロールする規格も制定されています。今回のAPIではこうしたデータの取り扱いもできるようになっていますので、舞台装置のコントロール用コンソールにAndroidが組み込まれた機器が出てくる可能性もありそうです。

YAMAHAがiOS用として、既存のMIDIデバイスを手軽にMIDI BTLE対応にできるようにするMD-BT01[注5]やUD-BT01[注6]のようなインターフェースを出してきましたので、モバイル端末とハードウェアシンセを使ったライブをする人も出てくるのではないでしょうか。SD

注5) http://jp.yamaha.com/products/music-production/accessories/interfaces/md-bt01/

注6) http://jp.yamaha.com/products/music-production/accessories/interfaces/ud-bt01/

一歩進んだ使い方のためのイロハ

Vimの細道

第7回

Vimの正規表現をマスターする

mattn
twitter:@mattn_jp

検索／置換に便利なVimの正規表現ですが、一般的な正規表現とは表記方法が一部異なるなど注意が必要です。今回は、一般的な正規表現について復習したあと、Vim特有の正規表現事情について解説します。Vim月報では、ソケット通信機能の追加について深掘りします。

正規表現 or Excel？

世の中にあるほとんどのテキストエディタは、検索や置換が行えます。またそのほとんどは、正規表現が扱えます。正規表現はパターンにより文字列をマッチさせるための表現方法であり、エンジニアであれば親しみのある機能だと思います。

たとえば、文章の中で行の先頭が「しかし」と書かれている部分を探すのであれば、**^しかし**というパターンを使うことで目的の文字列にジャンプできるでしょう。また、置換文字列に「されども」を設定することで置換も行えるでしょう。

もう少し例を示します。

```
# 品目,重量(kg),金額
人参,1,100円
白菜,3kg,300円
トマト,3,350円
```

このテキストの中央の列「重量」で、「kg」という単位の付け忘れがあったとします。みなさんであればどのように修正するでしょうか？ たった2行だから手で書き直しますか？　確かにそのほうが早いと思います。しかしこの行が50行あり、なおかつ単位が付いていない行が入り乱れていたらどうしますか？　スクリプトを書きますか？

このようなテキストの置換を行うためには、置換の対象となる文字の前後も合わせてマッチさせる必要があります。単純な正規表現であれば、検索文字列は次のように書けます[注1]。

```
^([^#,]+),([0-9]+),(.*)$
```

また、置換文字列は次のように書けます。

```
$1,$2kg,$3
```

置換を実行すると、

```
# 品目,重量(kg),金額
人参,1kg,100円
白菜,3kg,300円
トマト,3kg,350円
```

というテキストに修正できます。正規表現を知っている方々には釈迦に説法ですが、念のため解説しておきます。

まずパターンの^は行頭を、$は行末を意味します。このどちらかがないと、行の途中部分にマッチしてしまいます。今回であれば、先頭が#で始まる行は無視する必要もありますし、中央の数字にマッチしないといけません。また「,」の個数が限定されます。よって、^と$を付けて厳密にマッチさせる必要があります。今回の例であれば、パターンが1行分になっているのが

注1) この正規表現は一般的なパターンです。Vimでは若干異なります。

わかるかと思います。

次に、()は後方参照のためのキャプチャと呼ばれます。これにより、置換文字列内で$1、$2といった表記で、キャプチャ部分にマッチした文字列を引用できるようになります。

[]は文字クラスを示します。[0-9]と書くことで0から9の数字1文字が表せます。また、[^#,]のように初めに^を書くことで、#や「,」ではない文字にマッチします。

そして正規表現では、「.」は任意の文字にマッチします。実際の「.(ドット)」にマッチさせたいのであれば、\.のようにエスケープする必要があります。

最後に*と+ですが、量指定子と呼ばれるものです。B*は「B」が0個以上の文字列にマッチし、B+は「B」が1文字以上の文字列にマッチします。AB*Cは「ABBC」にも「AC」にもマッチしますがAB+Cは「AC」にはマッチしません。

これらパターンを使うことで、最初の#が付いた行は無視され、かつ中央が数字だけで作られている行だけがマッチする対象となり、それを$1,$2kg,$3に置換することで目的のテキストに置き換えられる、というしくみです。正規表現の書き方は1つではありません。人の癖もありますし、速度重視やメンテナンスコスト重視といった理由によって合理的に作られるため、正解は1つだけではないのです。

正規表現はテキストエディタだけの機能ではなく、多くのプログラムでも使用されます。正規表現が不得意なエンジニアの方は、ぜひ克服することをお勧めします。しかしながらまれに、このようなテキストの一括編集にExcelを使う人を目にします。それはそれですごいなとは思いますが、なぜ正規表現を使わないのかと不思議に思います。

一見、万能な魔法のように思えますが、正規表現をサポートしているテキストエディタであっても、できることには限界があります。たとえば、先ほどできあがったテキストに対して、今度は右端の金額に消費税率を掛けたいとします。そうなるとみなさんも、Excelを起動してしまうかもしれませんし、スクリプトを書いてしまうかもしれません。

でもVimを使うと、簡単にこれを実現できてしまうのです。今回はこのように不可能を可能にする、魔法の「Vimの正規表現」を紹介します。

実は、Vimと正規表現はとても密接につながっており、シンタックスハイライト機能においても正規表現がふんだんに使われています。さらにはその特殊な機能を実現するために、Vimは正規表現エンジンを独自にカスタマイズした形で実装しており、一般的な正規表現にはない記法も存在します。

正規表現をサポートする機能

magic機能

Vimでは、( ) + |といった、パターンで使用する制御文字をエスケープする必要があります。これはシェル上で実行するgrepやsedなどでも同じですね。次は、Vimで/(検索)を行うときのパターンです。

```
/^\([^#,]\+\),\([0-9]\+\),\(.*\)$
```

一見、\が並んでいて入力がたいへんそうです。あらかじめ\を入力する数が多いとわかっている場合は、パターンの先頭に\vを入力しておくことで、一般的な正規表現と同じ扱いにできます。

```
/\v^([^#,]+),([0-9]+),(.*)$
```

これは「very magic」と呼ばれる機能で、正規表現のメタ文字をエスケープしなくてもいいように切り替えられます。そのほかにも、very magicと逆の動作をする\V(very nomagic)や、一部の文字だけmagic動作を行う\m(magic)、その逆の\M(nomagic)があります。詳しくは:help magicを参照してください。

ハイライト機能と省略機能

set hlsearchを設定しておくと、マッチした部分がハイライト表示されるため、とてもわかりやすくなります。また、set incsearchを設定しておくと入力中の正規表現に対してもハイライトが行われるため、とても直観的に正規表現を入力できます。

なおVimでは、いったん検索を確定しておいたあとに、

```
:s//# &/g
```

のように実行することで、前回入力したパターンを省略できます。先の例であれば、単位「kg」が入力されていないすべての行の先頭に#が付与されます。

ちょっと特殊なVimの正規表現

\{n,m}

一般的な正規表現では?を使うことで最短マッチを表せます。たとえば、UNIX上のパス/usr/share/の下に存在するとあるディレクトリと、その直下にあるREADMEファイルをパターンとしてマッチさせたい場合、一般的な正規表現だと、

```
^/usr/share/.*?/README$
```

と書けます。もちろんここで最短マッチを使わないと、さらに下の階層のフォルダにあるREADMEにもマッチしてしまいます。一方、Vimの/検索では、次のようになります。

```
^\/usr\/share\/.\{-}\/README$
```

Vimでは.*?というパターンが.\{-}となります。ところで、\/が多くてちょっと煩わしいですね。Vimの検索時には、/は必ず\/にエスケープする必要があるので、

```
:cnoremap <expr> / getcmdtype()➥
 == '/' ? '\/' : '/'
```

のように設定しておくと、/が勝手に\/に置き換わって便利かもしれません。ただし、すでに\/でエスケープする癖が付いてしまっている人には若干使いづらかったりします。

ここでさらに「そのディレクトリは『n』から始まり3文字から5文字」という検索を行いたい場合、Vimでは、

```
/^\/usr\/share\/n.\{3,5}\/README$
```

のように書けます。詳しくは:help \{を参照してください。

\%V

Vimで矩形を選択して、その矩形内でのみ置換を行いたい場合、そのまま:をタイプして、

```
:'<,'>:s/foo/bar/g
```

を実行しても、期待どおりには動作しません(図1)。

確かに矩形の開始行から終了行内では置換されますが、矩形の右側や左側にある「foo」も「bar」に置換されてしまいます。期待どおりに矩形内のみで置換を行いたい場合は次のように実行します。

```
:'<,'>:s/\%Vfoo/bar/g
```

\%Vは、矩形選択された中のパターンだけにマッチさせるおまじないです。

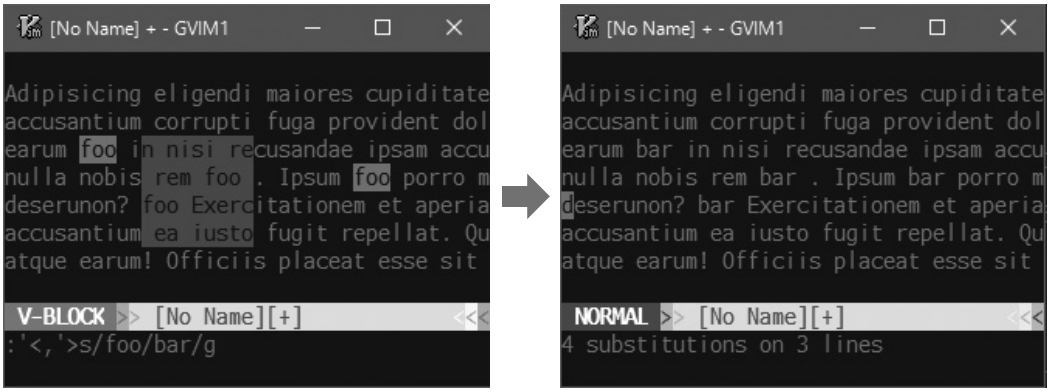
//e

テキストを編集していると、ある文字列を検索したあとでその検索語の次の位置から文字をタイプしたいときがあるかと思います。たとえば、「ヨーデル」の次の位置からタイプしたい場合は、

```
/ヨーデル/e+1
```

と入力します。//eはマッチ文字列の末端の文字「ル」に移動するコマンドです。さらにオフセットを1つ進めるには//e+1と入力します。逆に、マッチ文字列の先頭を指すにはsを使います。そ

▼図1　矩形の外側にある「foo」も置換されてしまった



のほか便利なオフセット機能が用意されています。詳細は:help search-offsetを参照してください。

\zsと\ze

正規表現には先読み／後読みというマッチ方法があり、それぞれ肯定と否定という条件指定ができます(**表1**)。

表現方法は異なりますが、Vimの正規表現でもこれらは同様に使用できます。しかしこれらは直観的とはとても言えず、入力についてもリーズナブルではありません。そこでVimの正規表現には\zsと\zeという特殊な識別が用意されており、これを使用することで、簡単にキャプチャ対象を部分的に適用できます。たとえば、「吾輩は猫である」という文章を「吾輩は犬である」でもマッチさせ、かつ実際にキャプチャするのは「猫」か「犬」という文字だけとしたい場合、

```
吾輩は\zs\(猫\|犬\)\zeである
```

と書けます。これを、**表1**の肯定先読みと肯定後読みで書き直すと、

```
\(吾輩は\)\@<=\(猫\|犬\)\(である\)\@=
```

と、直観的でない正規表現になってしまいます。また既出の、野菜の価格表テキストの重量部分のみキャプチャするのであれば、

```
^[^#,]\+,\zs[0-9]\+\ze,
```

と書くことができ、パターンをそれほど壊さずにキャプチャ対象を変更できます。

また、\zsよりも前の部分、\zeよりも後ろの部分はキャプチャ対象からは外れているため、$1や$3を検出するためにキャプチャする必要もありません。Vimでは置換文字列に&を使うとマッチした部分全体に置き換えられるため、\zsと\zeを使った場合の置換文字列は&kgとだけ書けばよくなります。まとめると、「抜けた単位kgを入れる」置換コマンドは次のようになります。

```
:%s/^[^#,]\+,\zs[0-9]\+\ze,/&kg/g
```

最初は難しいかもしれませんが、慣れるとこういったパターンがさらさらと出てくるように

▼表1　先読み／後読みマッチング

名称	パターン	Vimでのパターン	マッチする対象
肯定先読み	foo(?=bar)	foo\(bar\)\@=	直後にbarがあるfoo
否定先読み	foo(?!bar)	foo\(bar\)\@!	直後にbarがないfoo
肯定後読み	(?<=bar)foo	\(bar\)\@<=foo	直前にbarがあるfoo
否定後読み	(?<\!bar)foo	\(bar\)\@<\!foo	直前にbarがないfoo

なり、テキスト編集が楽しくなります。

\%[]

たとえば、あるテキストの中にある書きかけの文章を、単語「read」で検索したいとします。「r」「re」「rea」「read」のどれかにマッチしたい場合は、\%[read]と書くことで文字クラスをシーケンシャルにマッチするパターンを記述できます。日本語でも扱えるので、

```
本当に\%[ありがとうございました]ry$
```

と書くことで、「本当にあり(ry」といった中途半端な日本語にもマッチさせられます。

▼表2　Vimの正規表現におけるアトム文字

アトム	意味
\i	識別文字(isindentオプションに依存)
\I	数字を除いた識別文字
\k	キーワード文字(iskeywordオプションに依存)
\K	数字を除いたキーワード文字
\f	ファイル名で使える文字(isfnameオプションに依存)
\F	数字を除いたファイル名で使える文字
\p	表示可能な文字(isprintオプションに依存)
\P	数字を除いた表示可能な文字
\s	スペースやタブといった空白文字
\S	空白文字以外
\d	数字：[0-9]
\D	数字以外：[^0-9]
\x	16進文字：[0-9A-Fa-f]
\X	16進文字以外：[^0-9A-Fa-f]
\o	8進文字：[0-7]
\O	8進文字以外：[^0-7]
\w	単語：[0-9A-Za-z_]
\W	単語でない文字：[^0-9A-Za-z_]
\h	単語の先頭文字：[A-Za-z_]
\H	単語の先頭文字以外：[^A-Za-z_]
\a	アルファベット文字：[A-Za-z]
\A	アルファベット文字以外：[^A-Za-z]
\l	小文字：[a-z]
\L	小文字以外：[^a-z]
\u	大文字：[A-Z]
\U	大文字以外：[^A-Z]

\<と\>

通常の正規表現の\bに相当します。\<と\>で囲うことで、単語の区切りにマッチさせられます。Vimのノーマルモードで*をタイプすると、カーソル配下のキーワードが検索できますが、実際にはこれは**\<カーソル配下のキーワード\>**という正規表現検索が行われています。

アトム

Vimの正規表現で使用できるアトム文字は**表2**のとおりになります。たとえば、プログラミング言語で用いる変数名は一般的に、

・アルファベット文字、もしくは「_」で始まり
・2文字目以降はアルファベット文字、数字、「_」のいずれか

となります。よって、

```
\<[a-zA-Z_][a-zA-Z0-9_]*\>
```

のように書けもしますが、アトム文字を使うことで、

```
\<\I\i*\>  または  \<\h\w*\>
```

と短く書けます。

submach()

Vimでは置換文字列にて\=...と書くことで、「式」を書けます。たとえば、今回何度も登場してきた野菜の価格表のテキストで、金額に消費税率を掛ける置換コマンドは、**リスト1**のようになります。

submatchには、キャプチャした結果の文字列が返ります。submatch(0)はマッチした文字列全体が得られ、1以降はパターンの中で\(\)によりグループ化させた結果の文字列が得られます。Vim scriptは文字列の連結を「.」で行いますが、浮動小数点は「.」で連結できないのでfloat2nrで整数化しています。結果として、小数の切り捨てにもなって良いですね。

▼リスト1　submatchを使って正規表現の式を書く

```
%s/^\(.*\),\(\d\+\)円$/\=submatch(1) . "," . float2nr(1.08 * submatch(2)) . "円"/g
```

Vimに限らず、正規表現を扱う場合に必要となるのは“発想力”です。「3列目をキャプチャしてsubmatchを使えば、消費税込みの金額に置換できる」といった思いつきの力、それこそがVimの正規表現をうまく使えるかどうかの決め手です。Vimは置換してしまったあとでも、undoで何度でも元に戻せます。何度も書き直して何度も実行し、正しい正規表現パターンが書けるまでいくらでも繰り返せば良いのです。

そうやって繰り返すことで、この“思いつきの力”が養われていくのです。Vimの正規表現や置換機能を使えば、ちょっとしたデータの加工にわざわざスクリプトを書く必要はない、そう思えてくるはずです。SD

Vim自報

「その機能はテキストの編集には関係ない」

13年前、筆者はVimにソケット通信機能を追加するパッチを書き、Vimの開発グループに送りました。その際、Vimの作者Bram Moolenaar氏から受けたのがこの言葉です。

それから時は流れ、現在ではVimユーザのほとんどは何かしらのVim pluginを使っています。みなさんがVimに期待しているものも、あの時とは変わってきました。そして今年の1月末、パッチ7.4.1191にて、Vimにchannelと呼ばれるソケット通信機能が追加されました。あのメールを送って以来、テキストエディタVimにソケット通信機能なんか付いたりはしない、そう思い込んでいた筆者には目が飛び出るかのような思いでした。確かに、筆者が書いたソケット通信機能はブロッキング通信であり、テキストエディタの編集操作に支障をきたす酷い物だったかもしれません。

今回Vimに追加されたchannelは、データを受信したタイミングでコールバックが呼ばれる非同期通信方式を採用しており、テキスト編集中でも通信が行えるようになっています。

リストAはソケット通信相手からのデータを都度表示するスクリプトです。また、外部プロセスを扱うための、job-controlという機能も追加されました。**リストB**はteeコマンドに「echo something」という文字列を書き込み、その結果を読み取って表示するスクリプトです。本記事執筆時点ではまだ安定しておらず、APIの名称についても変わってしまうかもしれません。しかしながら今後、Vimの非同期処理はchannelとjob-controlへと置き換わっていき、テキストを編集しながら別のバッファでは異なる情報が自動で更新されていくといった、Vimの近未来がやって来ることになるでしょう。

▼リストA　通信相手からのデータを都度表示

```
function! Callback(handle, msg)
  echo a:msg
endfunction
let handle = ch_open("127.0.0.1:8888", "json")
call ch_sendexpr(handle, "GET", "Callback")
```

▼リストB　コマンド結果を読み取って表示

```
let job = job_start("tee")
let handle = job_getchannel(job)
echo ch_sendraw(handle, "echo something\n")
call job_stop(job)
```

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://rubikitch.com/

第24回 少し休憩。Emacsでゲームを遊ぼう

今回は箸休め。Emacsで遊べる5つのゲーム、テトリス・五目並べ・ヘビゲーム・精神科医と相談・ハノイの塔を紹介します。どれもインストールの必要なし、かつ簡単操作ですぐに遊べますが、歯ごたえのあるものばかりで熱中してしまうかも!?

迫るEmacs25

ども、るびきちです。先月末にEmacs25のpretestがリリースされました。正式リリースが真近に迫っているのは間違いありません。Emacs25は、webkit埋め込み機能とダイナミックリンク(拡張ライブラリ)がサポートされます。webkit埋め込み機能を使えば、グラフィカルなWebブラウザをEmacsのバッファで表示させる離れ業が可能になります。もともと規格外なテキストエディタであるEmacsで、さらにブッ飛んだ芸当ができるようになるのです。ダイナミックリンク機能によりユーザはC言語でEmacsの関数を記述できるようになります。elispやプロセス間通信だと遅くて実用的でなかったことが実現できます。楽しみになってきましたね!

◆ ◆ ◆

あなたは、Emacsがゲームプラットフォームであると聞いたら驚くでしょうか? 実はEmacsはインストールした時点でいろいろなゲームが遊べます。ファイラー(dired)やメーラ(gnus)やシェル(eshell)がelispで実装されているのですから、ゲームがあってもおかしくはありません。今回はEmacsで遊べるゲームについて採り上げていきます。

テトリス

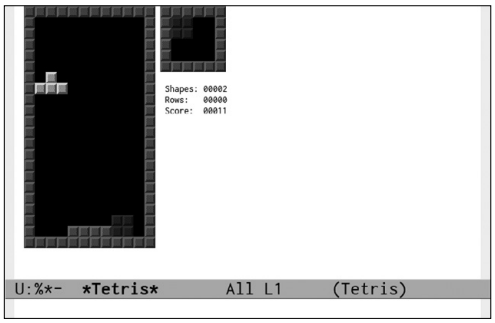
落ちものパズルゲームの元祖テトリスもEmacsで遊べます。Emacsでは任意のテキストに色が付けられることを知っていれば当たり前かもしれませんが、本家さながらの色付きブロックでテトリスができるのはすばらしいことです(**図1、2**)。

`M-x tetris`で起動します。ブロックは左右キーで動かし、上下キーで回転させ、Spaceで落下させます。Spaceで落下するとすぐに固まるので、落下直後に移動させるテクニックは使えません。

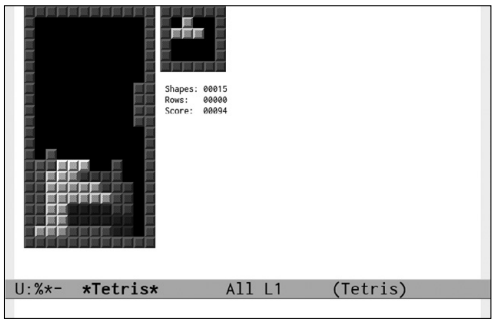
スコアの算出方法は本家とはまったく異なります。本家テトリスでは複数行消しに対してボーナス得点が加算[注1]されますが、Emacsのテトリスでは実装されていません。また、行を消すよりも高い位置から落下させることによる得点のほうが多いです。よって、Emacsのテトリスで高得点を出すには本家とは異なる戦略が必要になります。本家では積極的にテトリスを狙うことで高得点が出せますが、Emacsでは落下を使いつつ地道に消していったほうが良いです。

注1) たとえば、シングル(1行消し)500点に対して、テトリス(4行消し)10,000点など。

▼図1　テトリスをプレイ



▼図2　棒が来た!!



デモプレイ

autotetris-modeパッケージをインストールすれば、テトリスのデモプレイが楽しめます。M-x autotetrisを実行すればEmacs自身がテトリスをプレイします。地道に消していくスタイルですので、プレイの参考になるでしょう。

解像度を下げよう

M-x tetrisや、あとに紹介するM-x snakeを高解像度でプレイすると、ゲーム画面が小さくなってしまいます。しかも、テキストのフォントを変更しても変わりません。その原因は内部で使用しているgamegrid.elです。これはグリッドベースのゲームを作成するためのライブラリですが、あいにくグリッドの大きさが16ドットに固定されています。見やすい大きさにするには、解像度を下げれば良いです。helmをインストールしていればM-x helm-xrandr-setで対話的に解像度を変更できます。

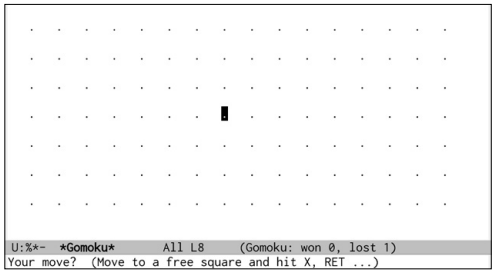
五目並べ

M-x gomokuを実行すれば、五目並べが遊べます(**図3**)。五目並べとは、ボード上に相手と交互に石を並べ、先に連続5つの石を並べたほうが勝ちとなるゲームです。石を置く位置はカーソル移動コマンドで指定します。Emacs式(C-p/C-n/C-b/C-f)とvi式(h/j/k/l)が使えます。石は Space Enter x のいずれかで置きます。

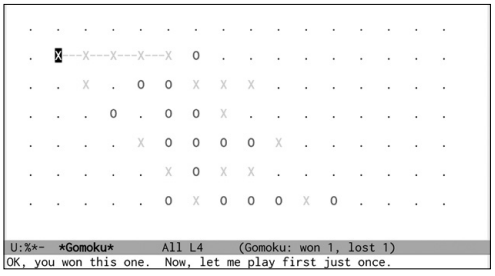
M-x gomokuを実行すると、まず先攻後攻を決定します。「Do you allow me to play first? (y or n)」と出てくるので、nと答えて先攻を選ぶのが無難です。

正直、Emacsの五目並べAIはめちゃくちゃ強いです。勝利のスクリーンショット(**図4**)を載せていますが、筆者では20回対戦して1回勝てるかどうかです。決着がついたあと再戦を要求(「Another game? (y or n)」)してくるのですが、そこでnと答えて断ると、「Chicken!(この腰抜けが！)」と罵ってきます(笑)。

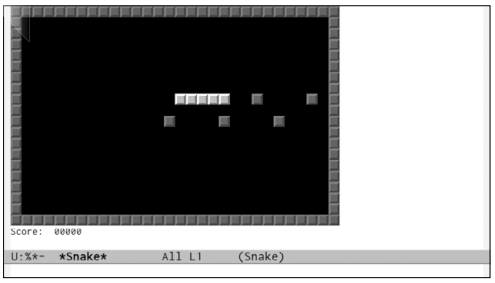
▼図3　五目並べスタート!



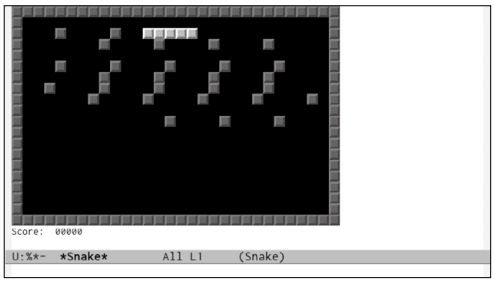
▼図4　勝った!



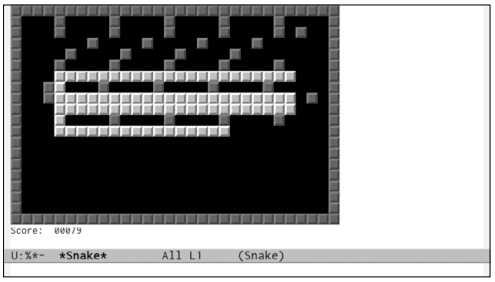
▼図5　ヘビゲームスタート



▼図6　どんどんエサが配置される



▼図7　エサを食べるとものすごく体長が長くなる



ヘビゲーム

M-x snakeを実行すると、ヘビゲームが遊べます。ヘビゲームとは、エサを食べるようにヘビを操作して自分自身を成長させる古典的なゲームです(図5)。ヘビは動き続け、壁や自分自身にぶつかったらゲームオーバーです。最初は体長が短いのでとても簡単ですが(図6)、しばらくすると長く成長した自分自身が邪魔になってくるので(図7)、通り道を塞がないように気を付ける必要があります。止まって休めばいいのに、などという突っ込みは無用です(笑)。

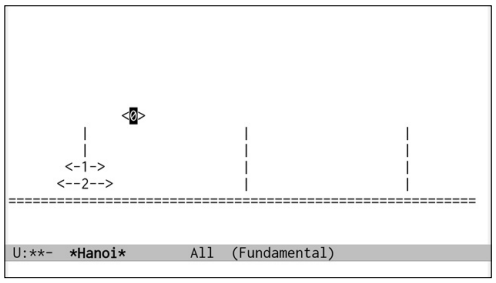
精神科医と相談

M-x doctorを実行すると、Emacsに内在する精神科医さんにあなたの悩みを相談できます。これは、ELIZAという初期の素朴な自然言語処理の人工知能(人工無脳)プログラムを基にしたものです。EmacsといえばLisp、Lispといえば

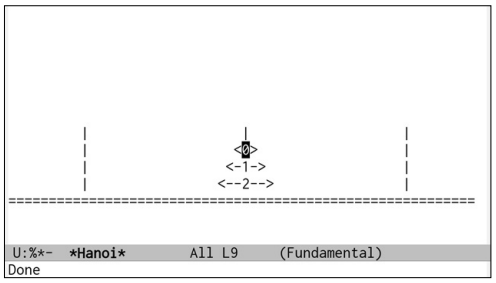
▼図8　精神科医と会話(アミ掛け部分がユーザの入力)

```
I am the psychotherapist.  Please, describe your problems.  Each time
you are finished talking, type RET twice.

My Emacs does not work well
Emacs?  Hah!  I would appreciate it if you would continue.

My text editor
Why do you say your text editor?

I use it everyday
Is it because of your plans that you say you use it everyday?

what?
Can you elaborate on that?

I cannot use other text editors
Maybe your life has something to do with this.

yes
I see...  Well, what makes you believe this is so?
(続く)
```

▼図9　ハノイの塔スタート



▼図10　完成!



人工知能、それならば人工知能がelispで実装されても不思議ではありません。しょせんはオモチャですが、ときにあまりにも人間的な会話をすることもあるので侮れません(**図8**)。

ハノイの塔

M-x hanoiでハノイの塔のデモを見られます。ハノイの塔とは、3本の杭が用意され、左端の杭にある円盤をすべて中央の円盤に移動させるパズルゲームです。円盤は1回につき1枚ずつほかの杭に移動できますが、そのとき小さい円盤の上に大きい円盤を乗せることはできません。プログラミングの学習で再帰呼び出しの例題によく採り上げられます。

円盤の数は数引数で指定します。C-u M-x hanoiだと4枚、C-u 6 M-x hanoiだと6枚になります。n枚の円盤をすべて移動させるには$2^n - 1$手必要になりますので、大きくし過ぎると延々と時間がかかります(**図9、10**)。

そのほかのゲーム

ほかにもEmacs上ではいろいろなゲームが遊べます。ゲームのelispファイルはlisp/play/以下(例:/usr/local/share/emacs/24.5/lisp/play/)に置かれています。そのディレクトリを簡単に開くには、M-x find-functiontetrisなどのゲームコマンドの定義を開き、そこからdiredを立ち上げてください。Emacsは、基本的にテキストエディタですので地味なものばかりですが、数多く用意されているので暇潰しにはもってこいです(**表1**)。もし「GNU/Linuxでのゲームって何がある？」と聞かれたら「Emacs」と答えるのも、あながち間違っていません。

おわりに

筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指すべく日々更新しています。手元でgrep検索できるよう全文をGitHubに置いています。またEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでも御答えします。「こんなパッケージ知らない？」「挙動がおかしいからなんとかしてよ！」はもちろんのこと、自作elispプログラムの添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

▼表1　MELPAパッケージからインストールできるゲーム

パッケージ	内容
2048-game	2048
sokoban	倉庫番
gnugo	GNU Go(碁)
minesweeper	マインスイーパー
typing-game	タイピングゲーム
slime-volleyball	バレーボール

書いて覚えるSwift入門

第13回 Protocol-Oriented Programming

Writer 小飼 弾（こがい だん） twitter @dankogai

前回に引き続きPOP

今回はProtocol-Oriented Programmingが、実際どれほど使い物になるかを実例とともに紹介していきます。

PONS＝Protocol-Oriented Number Systemの紹介

前回取り上げた実例は、複素数(complex numbers)の実装、swift-complex[注1]でした。記事にはこうあります。

・誰かが任意精度の数値ライブラリを用意すれば、それを使うことも可能

でもそれって本当？

それが、今回紹介するPONS = Protocol-Oriented Number Systemを書くことになったきっかけです(**図1**)。

使い方は簡単。

①`git clone`して`PONS.xcworkspace`を開いて、`Framework-OSX`をビルドしたら`OSX Playground`の実例が実際に動くようになります。試しに`(1...100).reduce(BigInt(1),combine:*)`と打ってみてください。

②もちろんREPLでも動きます。`make repl`するとREPLが立ち上がるので、`import PONS`してから`(1...100).reduce(BigInt(1),combine:*).description`とか打ってみてください。

③読者ご自身のプロジェクトで使いたい場合も、Frameworkをコピーしてもよし、ソースファイルをコピーしてもよしです。

これで、次のようなことができます。

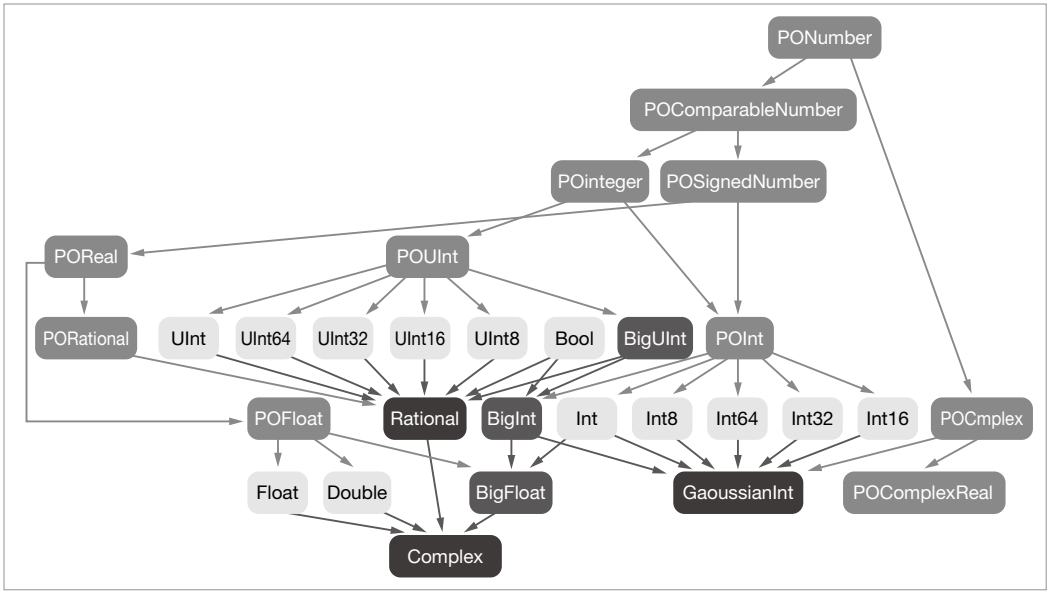
・RubyやPythonやHaskellではおなじみの任意精度整数(`BigInt`)が、GMP[注2]などの外部ライブラリなしで使えるようになります。
→素数判定メソッドも付いてきます。いや、同じく任意精度整数が組込みのPerl6にも組込みだったので。

・有理数型(Rational)もついてきます。分子と分母の型を引数とする総称型ですので、もちろん任意精度有理数＝`Rational<BigInt>`も使えます。よく使うので`BigRat`ととして`typealias`してあります。

・任意精度浮動小数点数(`BigFloat`)もついています。`BigRat`だけでも好きなだけ小さな数も実現されているのですが、より高速かつ省スペースです。
→総称的に定義された初等関数(elementary functions)の実装も付いてきます。たとえば`Float128`とか、新たな数値型を実装したときに`exp`や`log`や`sin`や`cos`を書き直す必要

注1）https://github.com/dankogai/swift-complex
注2）https://gmplib.org

▼図1　swift-pons（https://github.com/dankogai/swift-pons）



はありません。実際BigRatとBigFloatはそれぞれまったく別の型なのに、これらの関数のソースは共通です。

しかし、PONSの本当のウリはそこじゃないんです。

1つで十分ですよ、わかってくださいよ!

古のCの時代、同じことをするコードは、型ごとに必要でした。試しにman cosしてみると……、

```
NAME
     cos -- cosine function

SYNOPSIS
     #include <math.h>

     double
     cos(double x);

     long double
     cosl(long double x);

     float
     cosf(float x);
```

倍精度double用にcos()、単精度float用にcosf()、そして拡張精度long double用にcoslと、標準で用意されているだけで3種類もあります。四倍精度(float128)とかが標準装備になったらcosq()でも加えるんですか？　昨今GPUで採用されはじめている半精度(float16)はcosh()ですか？　でも待って、coshはもう双曲線コサイン(hyperbolic cosine)に取られちゃってますよ？

ぶっちゃけ付き合ってられませんよね？

Swiftは、はじめからこの問題にある程度対処されています。

```
#if os(Linux)
import Glibc
#else
import Darwin
#endif
```

された状態でXcodeにてcosと打つと……、**図2**のように、DoubleもFloatもCGFloatも、どれも同じcosで呼び出せることがわかります。

しかし、自分でこのような同名別型関数を用意しようとした場合、どうしたらよいでしょう？

こうですか？

```
func fib(n:Int8)->Int8   { return n < 2 ? n : fib(n-2)+fib(n-1) }
func fib(n:Int16)->Int16 { return n < 2 ? n : fib(n-2)+fib(n-1) }
func fib(n:Int32)->Int32 { return n < 2 ? n : fib(n-2)+fib(n-1) }
func fib(n:Int64)->Int64 { return n < 2 ? n : fib(n-2)+fib(n-1) }
```

だが断る！

だがしかし、Swiftには総称型があります。こうは書けないのでしょうか？

```
func fib<T>(i:T)->T { return i < 2 ? i : fib(n-2)+fib(n-1) }
```

でもTを足したりTどうしを比較する方法をSwiftは知りませんから、残念！　よろしい。ならばプロトコルだ。比較できて足せる型Hogeがあれば、

```
func fib<T:Hoge>(i:T)->T { return i < 2 ? i : fib(n-2)+fib(n-1) }
```

で行けるはずだ。でもそのHogeってどこにあるの？　PONSは、まさにそのためにあるのです。実際に試してみましょう。PONSでは、上記のHogeはPOIntegerが相当します。

```
import PONS

func fib<T:POInteger>(n:T)->T {
    if n < T(2) { return n }
    var (a, b) = (T(0), T(1))
    for _ in 2...n {
        (a, b) = (b, a+b)
    }
    return b
}
```

で、実際に下記がそのまま動けば、公約が果たされたことが確認できるわけです。

▼図2　COSの呼び出し

```
f  Double cos(Double)
f  Double cos(Double)
f CGFloat cos(x: CGFloat)
f  Double cos(x: Double)
f   Float cos(x: Float)
f   Float cosf(Float)
f   Float cosf(Float)
f  Double cosh(Double)
cosine function More...
```

```
let F11 = fib(11 as Int8)
let F13 = fib(13 as UInt8)
let F23 = fib(23 as Int16)
let F24 = fib(24 as UInt16)
let F46 = fib(46 as Int32)
let F47 = fib(47 as UInt32)
let F92 = fib(92 as Int64)
let F93 = fib(93 as UInt64)
```

ぜひご自身でご確認を。

しかし、このプロトコルは既存の型だけではなく、どこからか持ってきた別の型にも適用できるのでしょうか？

BigIntでやってみましょう。

```
let F666 = fib(666 as BigInt)

 685935696388048441387540130217643178807321423453572526486043772015797214210889451126489836614552862254308264662614052709773955669907870808 8
```

になりましたか？

でも、プロトコルなら運命を変えられる。避けようのない重複コードも、嘆きも、すべて君が覆せばいい。だからPONSと契約して、数学ガールになってよ！

……失礼しました。SEGVです。XcodeでProtocolを多様したプログラムを書いていると本当によくお目にかかれます:-(。

しかし数値は整数だけではありません。整数だけで満足できるのは小学生とクロネッカー[注3]先生

注3）ドイツの数学者（https://ja.wikipedia.org/wiki/レオポルト・クロネッカー）

だけです。有理数も浮動小数点数もあるんだよ。

とはいえこれらを漠然と並べただけでは、体系(system)とはいえません。同じ/だって、整数型と実数型で違いますし。しかも、ただ符号なし整数→符号付き整数→実数→複素数とトップダウンにするわけにもいかないのです。確かに複素数は、四則演算と冪乗根(べきじょうこん)に対して閉じていますが、大小比較ができないというほかの数値型にはない特徴があります。

向き合った結果が、冒頭のグラフになります。ご覧いただければわかるとおり、複素数は実数からできているけど、大小比較はできないという関係が確かに成立しています。

だから、きちんとこのようになります。

```
Double.sqrt(-1) // NaN
Complex.sqrt(-1)// (0.0+1.0.i)
// そもそも比較できない
1.0+0.0.i < 2.0+0.0.i
// 絶対値を見ればおk
(1.0+0.0.i).abs < (2.0+0.0.i).abs
```

これが、「本当の数値と向き合えますか？」の筆者なりの回答になります。

Protocol-Oriented Programming = 正しいものが報われる世界

PONSが目指したもの、それは「正しいものが報われる世界」ということに尽きます。掛け算1つとっても、固定長の数値型では、その半分の型におさまる数値しか安全にできません。63357を自乗するだけで、32ビット整数はオーバーフローするのです。『C言語によるアルゴリズム辞典[注4]』というロングセラーがあります。PONSの実装でも大いに参考にさせていただいたのですが、intやdoubleの制約を回避するのに涙ぐましいほどの努力をしていて時代を感じさせます。正しい世界とは、そうではなくてアルゴリズムをそのまま書き下せばそのまま動く世界のはずです。

たとえばモンゴメリー乗算というアルゴリズムがあります。これを使うと割り算なしで冪剰余を計算できたりするので素数が捗ったりします。PONSにも実装されているのですが、普通に実装すると簡単に整数オーバーフローしてしまいます。そのため固定整数のみで実装しようとするとたいへんなのですが[注5]、BigIntがあれば「オーバーフローしそうならそこだけBigInt使って」ということがとても簡単に実現できます。実際に現時点におけるPONSの冪剰余はそのように実装されています。

その一方、BigIntは自分では文字列化メソッドを持っていません。整数の文字列化には、整数型に依存するアルゴリズムがすでに存在するからです。PONSでは、そのメソッドはPOIntegerでこう実装されています(**リスト1**)。

元の数を底(base)で割っていき、その余りをまとめるという操作は、その整数型の実装にはまったく依存しません。つまりPOIntegerに準拠した数値型は、何も加えなくても文字列化できるということです。

そもそもBigIntがあればほかの整数型はいらないという意見もあり得ます。大は小を兼ねるじゃないかという意見もごもっともですが、しかしSwiftを含め、多くの言語で整数型が固定なのには立派な理由があります。任意精度整数はとても重いのです。軽くベンチマークしてみると、64ビットにおさまる20! = 2432902008176640000を計算するのに、PONSのBigIntではSwift組込みIntのなんと500倍も時間がかかるのです。これはPONSの実装がしょぼいから、ではなく実は相場どおりで、Perl 5標準装備で、ただし組込みではないMath::BigIntもネイティブな64bit整数の250倍でした。ちなみにPONSのBigIntは世界最速の任意精度整数からはほど遠いのですが、それでも、Swiftがネイティブコードコン

注4) 奥村晴彦 ISBN978-4874084144
注5) http://www.hackersdelight.org/MontgomeryMultiplication.pdf

▼リスト1　PONSの実装例

```
public func toString(base:Int = 10)-> String {
    guard 2 <= base && base <= 36 else {
        fatalError("base out of range. ¥(base) is not within 2...36")
    }
    var v = self
    var digits = [Int]()
    repeat {
        var r:Int
        (v, r) = Self.divmod8(v, Int8(base))
        digits.append(r)
    } while v != 0
    return digits.reverse().map{"¥(POUtil.int2char[$0])"}.joinWithSeparator("")
```

パイラーということもあってかMath::BigIntの5倍の速度が出ています。

コードを使い分けずとも、型は使い分けられる。PONSでそのことを示せたと自負しています。

予告

実はPONSのようなものは、Swiftの中の人もTodoにしていたようです。Swift-Evolutionメーリングリストに、次のような書き込み[注6]がありました。

> *I have been working for some time on a rewrite of all the integer types and protocols https://github.com/apple/swift/blob/master/test/Prototypes/Integers.swift.gyb. One goal of this effort is to enable operations on mixed integer types, which as you can see is partially completed. In-place arithmetic (anInt32 += aUInt64) is next. Another important goal is to make the integer protocols actually useful for writing generic code, instead of what they are today: implementation artifacts used only for code sharing. As another litmus test of the usefulness of the resulting protocols, the plan is to implement BigInt in terms of the generic operations defined on integers, and make BigInt itself conform to those protocols.*
>
> *「現在数値型の書き直しに取り組んでいる。その次には(anInt32 += aUInt64 のような)異なる型通しの演算が控えている。もう1つのゴールは、整数型プロトコルが総称的なコードを書くのに実際に役立つようにすること。実際にプロトコルに準拠したBigIntを実装するというのは、そのためのリトマス試験紙となりうる」*

Swift 2.1でもできちゃいましたが、何か？

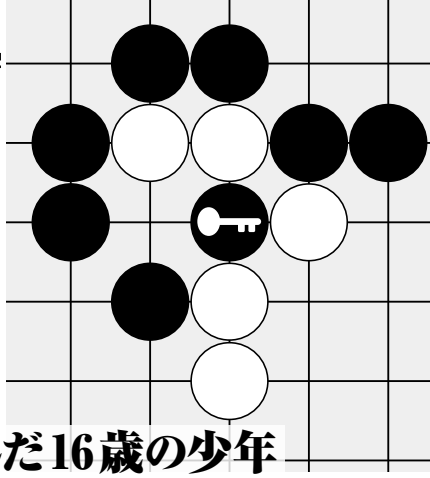
ただし、現時点で組み込まれているIntegerTypeとかはそのまま使えませんでした。見てのとおり、現在の組込みプロトコルでは、複素数のように「整数や実数のできることはほとんど何でもできるけど、比較はできない」といったような関係を反映させるのが困難だったからです。とはいうものの、ComparableやHashableといった、準拠していないと利便性があまりに下がるプロトコルは控えめに導入しています。たとえばBigIntでも(1...100).reduce(BigInt(1),combine:*)できるのは、POIntegerがRandomAccessIndexTypeでもあるからです。

次回はそんな「隠れプロトコル」を取り上げます。それがわかれば、なぜsin(1.0)と書かなくてもsin(1)で型エラーを起こさないかが見えてきます。SD

注6） https://lists.swift.org/pipermail/swift-evolution/Week-of-Mon-20151214/002445.html

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第三一回】米国CIA長官のメールを盗んだ16歳の少年

本連載ではこれまで、バレルセオリーという概念や、情報セキュリティの中で最も脆弱な部分は人間である、ということを考えてきました。今回は、このように今まで学んできたことが、現実の世界ではどのような形で現われているかというケーススタディをしてみます。

英国在住16歳少年が逮捕される

2016年2月中旬から、欧米の各メディアは「米国CIAトップのメールをハッキングした英国在住のティーンエージャーが逮捕される」という内容で溢れました。

まず筆者が最初に見つけた2016年2月12日付けのWeb版のThe Washington Post紙に掲載された“British teen arrested in hacking of top U.S. intelligence officials”という記事[注1]から重要な点を挙げてみます。

- **英国で16歳の少年が、米国諜報部局高官の個人メールのアカウントをクラックした容疑で逮捕された**
- **少年は自らを“Crackas With Attitude（CWA）”の一員と名乗っていた**
- **逮捕したことをFBIと米国司法省も認めた**
- **FBIと連邦検事らが数ヵ月に渡り捜査していた**
- **CWAはDHS（Department of Homeland Security：米国国土安全保障省）とFBIの勤務者の名前、メールアドレス、電話番号を大量にリークしていた**

今回の逮捕劇の第一報はCNN[注2]です。CWAに関しては、すでに2015年10月にNew York Post紙がコンタクトを取り、相手が米国高校生だと名乗っていること、またその手口を報道していました[注3]。また今年に入ってからも、雑誌Motherboardにインタビューが掲載されています[注4]。こちらはDNI（Director of National Intelligence：アメリカ合衆国国家情報長官）の、Verizon FiOSというサービスのネットアカウントをクラックしていたずらをしたと語っています。DNIは本連載のスノーデン事件の話題の際（本誌2014年5月号）にも説明しましたが、米国のすべての諜報部局を取りまとめている諜報部局の頂点です。

しかし、仮にもCIAのトップやDNIの個人メールのアカウントをクラックしているわけですから、「高校生の自分がやりました」という言葉を鵜呑みにできるわけもなく推移を見守っていました。

日本ではあまり興味が持たれていないようでしたが、海外、とくに米国ではトップシークレットを扱うクラスからの情報流出ですから、たいへんな騒ぎになっていました。FBIやDHSに勤務する人間へコンタクトできる情報だけではなく、ソーシャル・セキュリティ・ナンバー（社会保障番号）も流出しています。ですから、連絡先のリストが漏れたという話

注1） British teen arrested in hacking of top U.S. intelligence officials（February 12, 2016） http://wpo.st/tHlD1
注2） First on CNN: FBI, British police nab alleged 'crackas' hacker（February 12, 2016） http://cnn.it/1QZeUQT
注3） Teen says he hacked CIA director's AOL account（October 18, 2015） http://nyp.st/1QKgc1f
注4） Teen Who Hacked CIA Email Is Back to Prank US Spy Chief（January 12, 2016）
http://motherboard.vice.com/read/teen-who-hacked-cia-email-is-back-to-prank-us-spy-chief

ではないのです。FBIでなくても一般企業のレベルでも秘匿すべき内容を持つリストが流出しているわけですから、米国政府の国家安全保障上の問題だといっても大げさではありません。

実際にあったかどうかは別として、内容が内容だけに「敵国スパイの仕業なのではないか」とか、「ティーンエージャーのふりをするのは捜査の撹乱(かくらん)なのではないのか」といった疑念が出てきても不思議ではありません。

しかし、今回、実際に英国でティーンエージャーが逮捕されたという事実が広く報道されたことで、過去にCWSが語っていた手口の信憑性も増したと言えるでしょう(少なくともティーンエージャーだったのは、本当だったわけですから)。

CIA長官のメールを盗む

ジョン・ブレナンCIA長官

ジョン・ブレナン氏はCIA分析官として1980年にキャリアを開始し、一貫して諜報(Intelligence)に携わってきました。サウジアラビアの現地責任者として赴任、ビル・クリントン大統領時代には毎日諜報に関しての報告を行う担当者(daily intelligence briefer)、そしてジョージ・テネット長官時代(1996～2004年)にはCIAの主席副長官をしていました。オバマ政権で国家安全保障担当補佐官としてホワイトハウス入りして[注5]、2013年からCIA長官となります[注6]。諜報に携わる人物としてはエキスパート中のエキスパートであり、名実ともに諜報の世界のトップクラスにいる1人でしょう。

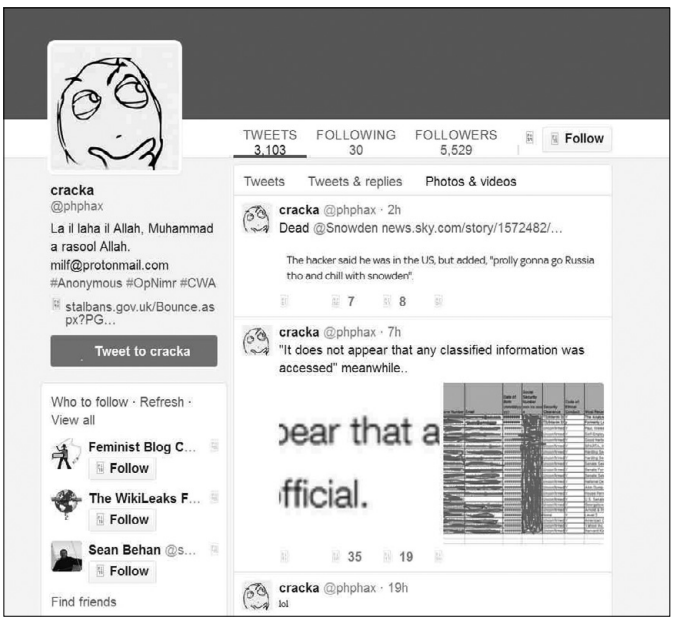
そのような人物の個人メールアドレスがクラックされ、情報が盗み出されたわけですから、尋常ならぬ出来事であるのは誰の目にも明らかです。

2015年10月時点での情報

犯人の少年(達)は、Twitterのアカウント「cracka」(ユーザ名は@phphax)で、情報を公開しています。当然ながら、すでにアカウントは凍結されており、現在は見ることはできませんが、Cryptome Archiveのサイトにcrackaのツイートの痕跡をPDFにして記録[注7]してありました(**図1**)。また、このPDFの中には、crackaがどのようなファイルを入手し公開していたかの記録も入っています。ジョン・ブレナンCIA長官のAOL(米国の大手インターネットサービス会社)のメールアカウントをクラックしたのがよくわかる資料となっています。

図1ツイートを見てみると、まさにティーンエー

◆**図1　crackaのツイート**



Cryptome Archiveにアーカイブされているcrackaの記録より。
これを見ると、「ちょっとこれはどうかな」と思うティーンエージャーにしか見えないが、それもディスインフォメーション(撹乱情報)という可能性もある。今でもFBIがミスをしていて最後の最後でどんでん返しがくる可能性もある。

注5) North Bergen man is homeland security assistant for President Obama (December 5, 2009)
http://www.northjersey.com/news/north-bergen-man-is-homeland-security-assistant-for-president-obama-1.264084
注6) John O. Brennan - Director, Central Intelligence Agency (March 8, 2013)
https://www.cia.gov/about-cia/leadership/john-o-brennan.html
注7) https://cryptome.org/2015/10/cracka-phphax.pdf

ジャーにしか見えません。ちなみに図1のツイートでリンクしているのは次のURLです。

http://news.sky.com/story/1572482/teen-brags-of-hacking-cia-directors-email

これはcrackaの公開したリストを、ニュース記事にしているページです。それを見せびらかして「スノーデンは死んだ(Dead @Snowden)」とツイートしているわけですから、どうみても精神年齢はティーンエージャーです。しかし、CIA長官のメールアカウントをクラッキングし、情報を盗むことができた人材なわけですから、このようなティーンエージャーのふりをすればするほど、犯人のプロファイリングを避けるためのカモフラージュなのではないかと言う人がいたとしても、筆者は不思議に思いません。

この当時はまだ、crackaがティーンエージャーかどうかは確実ではありませんでしたが、1つだけ確実に言えることがありました。入手したリストを安易に公開したということは、この情報をマネタイズする方法を持っていなかったか、はじめからマネタイズする気がなかったと考えられます。つまり、プロではないということです。

2015年10月時点で、CWAは少なくとも2つのメディアにコンタクトを取っていました。1つは先ほど述べたNew York Post紙、もう1つはMotherboard[注8]というWebメディアです。

これらの記事の要点は次のとおりです。

- **オンラインのチャットやフォーラムで知り合ったティーンエージャーが6人集まってできたのがCWA。そのうち、メディアにコンタクトを取っている(そして「cracka」のアカウントを使っている)のは、15歳の米国の高校生である**
- **2015年10月12日にブレナンCIA長官のメールアカウントに侵入し、40通のメールに添付されていたドキュメントを盗み出した**
- **我々は政治的な意図はないし、敵対する国の人間でもない。ブレナンCIA長官を懲らしめることを狙ったわけではなく、辱めるためである**
- **Verizon(米国の携帯電話会社)のコールセンターに連絡を取り、同社の顧客対応担当(live chat department)のふりをして「顧客情報を確認するツールがうまく動かないので、顧客の対応をするために情報を確認したい」と言ってブレナンCIA長官の個人情報を入手した**
- **次にAOLに電話をかけ、入手した情報を使うことで、ブレナンCIA長官の使っているアカウントのパスワードをリセットした**

これらを実行するために、本格的な学習を必要とする基礎知識も特別な技術も必要ありません。極端なことを言えば、英語が流暢(りゅうちょう)に話せれば良いだけです。

犯行の手口を推測してみる

どのようにターゲットを決めたか

ここで最もクリティカルなのは、Verizonから顧客情報を抜き出したことです。

まず、なぜVerizonに狙いを定めたのかということを考えてみます。Verizon(正式名称はVerizon Wireless)は、米国最大手のモバイル電話会社です。米国で全米をカバーしているモバイル電話会社は、Verizon Wireless、AT&T Mobility、T-Mobile US、Sprint Corporationの4社です。Verizon Wireless、AT&T Mobilityが2強で、それ以外のT-Mobile USはドイツの会社であるT-Mobile International AGの傘下で、Sprint Corporationはソフトバンクの傘下です。米国の政府要人が使っているのは、米国資本の会社であることはほぼ間違いないでしょうから、VerizonかAT&Tかの2分の1の確率でヒットします。まず、Verizonに問い合わせることになるでしょう。

末端の販売店や顧客対応担当からかかってくるト

注8) Teen Hackers: A '5-Year-Old' Could Have Hacked into CIA Director's Emails(October 19, 2015)
https://motherboard.vice.com/read/teen-hackers-a-5-year-old-could-have-hacked-into-cia-directors-emails

ラブルシューティング窓口(コールセンター)については、一般に公開されていないといっても国家安全保障レベルの秘密の電話番号というわけではありません。その手のマニアの集まるフォーラムで、簡単に知ることができるでしょう。

では、なぜブレナンCIA長官を狙ったのか、あるいは狙えたのか、です。窓口に問い合わせるときに、どんな確認をされるかを想像してみました。通常は名前、住所、年齢、そして電話番号だと思います。ですが、データベースから検索するには、最低限の情報として名前と郵便番号あれば検索できます。ここには情報は載せませんが、実はブレナンCIA長官の自宅の住所はすでに知られています。ですので、本人が住んでいるかどうかは別として、電話登録しているであろう公式の住所は今や誰でも知ることができます。

Verizonコールセンター

次に、オペレータの作業を考えてみます。効率を考えた検索システムでは、名前、郵便番号／住所、年齢、電話番号を入れていくことで次々に絞り込んでいって、1人に絞れた段階で終了させ、余計な入力をさせることはないでしょう。

ここからは筆者の想像なのですが、この電話対応をしていたコールセンターは、米国内ではなく国外にあった可能性があります。実際に、American Express、Dellといった米国会社はコールセンターをインドにアウトソーシングしていますし、Verizonも同様です[注9]。

インドのコールセンターがどのようなものか、日本人には、いやインドのコールセンターで働いている人以外には、想像がつかないと思います。ですが、非常にわかりやすい参考になるものがあります。映画「スラムドッグ$ミリオネア」(2008年)です。

この映画の中でも、英国女性が英国でかけた電話がインドのコールセンターにまわされています。顧客の個人情報がインドに持ち出されて利用されていることが、よくわかるような演出をしています。そして、オペレータが迅速に大量に問い合わせをこなすことが、評価の基準であることがよくわかります。

ブレナンCIA長官は政府高官だけあって広々とした郊外の住宅地に住んでいるので、近所に同姓同名(さらにミドルネームも同じの)John O. Brennanという人が偶然に住んでいるとは思えません。オペレータが名前と住所(あるいは郵便番号)を入力した時点でデータベースの情報が画面に現れるでしょう。あとは効率のため、自分の成績のために、ソーシャル・セキュリティ・ナンバーという重要な個人情報であっても、オペレータは問われれば躊躇(ちゅうちょ)なく表示されている情報を答えるでしょう。

Verizonのコールセンターがインドにあったとして、さらにそこのオペレータが受け答えしたとして、John O. Brennanなる人物が何者なのか、気がつくとしたらそれは奇跡に近いと思います。もしかすると、これはインドに限らず、どの米国国内のオペレータでも同じかもしれません。

余談になりますが、現状のEUデータ保護規則では、EU域内から「十分な保護措置」を備えていない国や地域に、EU内の個人データを移転するのは違法となっています。現在、EUから認定を受けている国は11ヵ国・地域です。2015年末においては日本も米国も認定されていません。EUとUSの間では双方の合意の新しいフレームワークを作り、その範囲でプライバシー情報の移転を許すという方向になるようです。日本政府とEU間に関して具体的にどう進んでいるかは調べたのですが、見つけたのは調査やかけ声的なものばかりで、具体的なものは見つけられませんでした。

AOLのアカウント

そもそもVerizonに問い合わせたのは、AOLのパスワード・リセットの際に、ソーシャル・セキュリティ・ナンバーが必要なためでした。ここから先は、すでに必要なものが手に入っているので、問題なくパスワードをリセットできます。

2015年10月12日(月)にパスワードをリセットし

注9) The Truth Behind Indian Call Centers
http://www.marieclaire.com/culture/news/a2961/outsourcing-indian-call-centers/

て侵入、そして金曜日（10月16日）に使えなくなったとありますから、最低でも12～15日の4日間はログインできたということです。個人アカウントがクラックされての4日間は十分に長い時間です。

たぶんメールに添付された資料は、zipされ暗号化されていたと思います。しかし、これもまたよく見かける風景ですが、次のメールでパスワードを送るという手順だったのではないかと考えるほうがいいでしょう。一般的なメールサービスでしょうから、侵入者の振る舞いによる異常検知といった特殊なセキュリティもないでしょうし、そうなればあとは取られ放題です。

■クリントン元国務長官の問題

これより約半年前の2015年3月に、クリントン元国務長官が任期中（2009～2013年）に個人の電子メールアカウントを使い、公務のメールをやりとりしていたことが発覚し、大きな問題になりました。この問題には大きく2つの問題がからんでいます。

プライバシーや国家安全保障に関して以外、米国政府の文書は国民が所有するもので、一定の期間が過ぎれば公開されます。電子メールも例外ではなく、政府のメールシステムでやりとりされるものは、すべてがアーカイブされ保管されています。

もう1つはやはりセキュリティの問題です。メールはクリントンの個人事務所で管理されていたそうですが、上級の技術者がついていたとしても、米国政府の専門部局が責任をもって管理するのはミリタリークラスですから、民生品とは一段違うレベルで管理されます。

CIA長官とバレルセオリー

CIA長官が政府で使っているメールシステムの安全性に関して疑問をはさむつもりはありません。ですが、現実には民生用のメール環境であるAOLを使い、そこから情報が漏れ出しているわけです。

バレルセオリーとは、樽を作っている木の板の一番低いところまでしか水がたまらないという事象を、セキュリティに当てはめて、「いくら部分的にセキュリティレベルを高度にしても、またお金をかけたとしても、最もセキュリティレベルの低い部分以上のセキュリティは望めない」とする考え方です。まさに今回のケースがこれに当てはまるわけです。

また、最も弱かった部分はどこかというと、コンピュータの技術などまったく関係ない、電話で応対するオペレータです。システムの中で最も脆弱なのは人間であるということの典型例です。これが素人のおじさんのメモ帳程度なら笑い話になるのですが、CIAに25年間勤務して、のちにCIA長官になったという諜報のエキスパート中のエキスパートがターゲットになり、まんまとメールを盗まれ、そこにあった情報が盗まれたのです。

今回AOLのパスワードのリセットでソーシャル・セキュリティ・ナンバー使われていたように、盗まれた個人情報のリストの中にはソーシャル・セキュリティ・ナンバーが入っているわけですから、それらを使って2次、3次の被害が出てくる可能性も大きいでしょう。

CWAは、DNIであるジェームス・クラッパー氏が使っているVerizon FiOSという総合ネットサービスのアカウントをクラックし、家の電話にかかる通話をすべてFree Palestine Movementの事務局に転送するという設定もしました。Verizonですので、たぶん同じような問題があって、似たような手口で突破したのでしょう。クラック自体はDNIオフィスの広報官が認めています。また、妻であるスーザンさんの持つYahoo!のメールアカウントも同様にクラックされています。ただし、アカウントが使われていなかったのか、こちらからはメールなどの流出はないようです。

筆者は「アメリカだから」や「アメリカですら」なのではなく、「アメリカでも」だと思います。どんな状況でもバレルセオリーが適用できると思います。

どんなに高度な技能を持つセキュリティ・エンジニアを投入しても、どんなに高価なセキュリティ機材を導入しても、全体を見回してバランスが悪いときには、そこが穴となってしまうことがよくわかる事例かと思います。そこが情報セキュリティの一筋縄ではいかない難しいところなのです。SD

Sphinxで始めるドキュメント作成術

清水川 貴之 SHIMIZUKAWA Takayuki Twitter @shimizukawa

第13回 MarkdownではじめるSphinx

SphinxはMarkdownも使える

本連載も今回から2年目に突入します。今回は、新しいSphinxの使い方の1つである、Markdown記法を使ったドキュメントの書き方を紹介します。

最近では、多くのシーンでMarkdown記法を見かけるようになりました。GitHubが公式の記法として採用し、日本でも、はてなブログ[注1]やQiita[注2]がMarkdown記法を採用しています。このように、「ドキュメントを書くと言えばMarkdown」という世の中の流れもあり、ドキュメンテーションツールがMarkdown記法に対応しているのは、必須と言えるかもしれません。

今回は、これからSphinxをはじめる方も試しやすいように、Sphinxでできることや、Sphinxのインストール手順にも、あらためて触れていきたいと思います。

注1） http://hatenablog.com/
注2） http://qiita.com/

Sphinxでできること

SphinxはPythonで作成されたドキュメンテーションジェネレータで、1つのドキュメントソースから、HTMLやPDFなどの複数のフォーマットのドキュメントを生成できます。Sphinx拡張（プラグイン）によって、出力フォーマットの追加、HTMLテーマの追加、拡張記法

COLUMN

Sphinxを使っているPython以外のプロジェクト

SphinxはPythonのドキュメントを書くために生まれてきたツールですが、さまざまな要望をもとに改良が重ねられ、汎用的に使えるようになっています。Pythonで作成されているツールのドキュメントはSphinxで作成されたものが多く見られますが、Pythonとは関係ないところでも利用されています（**表A**）。

▼表A　Sphinxで作成されているドキュメント（Python以外）

ソフトウェア／サービス	URL
Chef	https://github.com/chef/chef-web-docs
Symfony	https://github.com/symfony/symfony-docs
CakePHP	https://github.com/cakephp/docs
Varnish	https://github.com/varnish/Varnish-Cache/tree/master/doc/sphinx
OpenCV	http://docs.opencv.org/3.0-last-rst/
MathJax	https://github.com/mathjax/MathJax-docs
Selenium	https://github.com/SeleniumHQ/selenium/tree/master/py/docs

の追加などが行えます。Sphinx-1.3からは拡張子別のパーサーを指定できるようになりました。これによって、Markdown記法のソースも読み込めるようになりました。

Sphinxのインストールと Markdownを使う設定

Sphinxを使うには、Python-2.6以上が必要です。Sphinxは安定版の最新である1.3.6をインストールするのが良いでしょう。ここではインストール手順を簡単に紹介します[注3]。

Pythonがインストールされていない環境の場合、まず初めにPythonをインストールして、pipコマンドを使えるようにしてください。Python-2.7.9以降であればPythonのインストーラにpipが含まれるため、新規インストールする場合はPython-2.7.9以降をお勧めします。

Sphinxのインストールにはpipコマンドを利用します。また、SphinxがMarkdownドキュメントを読み込めるようにするため、サードパーティ製Markdownパーサーのrecommonmark[注4]も同時にインストールします。コマンドは次のように実行します。

```
$ pip install sphinx recommonmark
```

Sphinxプロジェクトの作成

Sphinxでドキュメントを作成するには、まず「Sphinxプロジェクト」を作ります。そのためのコマンドがsphinx-quickstartです。sphinx-quickstartの実行は、コマンドラインで行います。Windowsの場合はコマンドプロンプト、Mac OS XやLinuxの場合は端末エミュレータ(ターミナルなど)を起動させます。sphinx-quickstartを実行すると、Sphinxプロジェクトを構成する基本的なファイルやディレクトリが生成されます(**図1**)。

Sphinxプロジェクトは、sphinx-quickstartの引数で指定されたディレクトリに**図2**の構造で生成されます。

Markdownを使う設定

Markdownを使うには、Sphinxプロジェクトの設定ファイルであるconf.pyをエディタで開いて、source_suffixの行を**リスト1**のように書き換えてください。

この設定により、拡張子が「.md」のファイルは

注3) 日本のSphinxユーザ会では、インストール手順を詳しく紹介しています。 http://sphinx-users.jp/gettingstarted/

注4) http://recommonmark.readthedocs.org/

▼図1 sphinx-quickstartの実行例

```
$ sphinx-quickstart -m -q -p project_name -a kawamoto -v 1.0 project_dir
Creating file project_dir/conf.py.
Creating file project_dir/index.rst.
Creating file project_dir/Makefile.
Creating file project_dir/make.bat.

Finished: An initial directory structure has been created.
  (... 以下略 ...)
```

ディレクトリを指定

▼図2 sphinx-quickstartで生成されるファイルとディレクトリ

```
project_dir/
  ├── _build/      ……ビルドした結果の出力先
  ├── _static/     ……ロゴや css などを格納
  ├── _templates/  ……カスタム HTML テンプレートを格納
  ├── conf.py      ……Sphinx プロジェクトの設定ファイル
  ├── index.rst    ……デフォルトのトップページ
  ├── make.bat     ……Windows 用 make コマンド
  └── Makefile     ……Linux/Mac 用 Makefile
```

▼リスト1 Markdownを読み込む設定

conf.py

```
# source_suffix = '.rst'                ←もとの設定をコメント化
source_suffix=['.rst','.md']            ←この設定を追加
source_parsers={                        ←source_parsersの設定も追加
    '.md': 'recommonmark.parser.CommonMarkParser'
}
```

recommonmarkパーサーで読み込まれるようになります。

MarkdownでSphinxドキュメントを書こう

それでは、Markdown記法を使ってドキュメントを書いてみましょう。ここでは本連載第2回(本誌2015年5月号)でも利用した議事録のサンプルをMarkdown記法で書きなおしたものを利用します。**リスト2**の内容をSphinxプロジェクトのディレクトリに「sample.md」というファイル名で保存します。

このサンプルでは画像の埋め込みを行っています。「sphinx-flow.png」というファイル名で画像を用意して、sample.mdと同じディレクトリに置いてください。

次に、このsample.mdをトップページからリンクするために、index.rstを開いて`.. toctree::`と書かれている部分に、sampleという行を次のように追加します。

index.rst
```
.. toctree::
   :maxdepth: 2

   sample
```

この`.. toctree::`という記述はドキュメントの構造を定義します。ここに先ほど作成したファイル名(拡張子を除く)を加えることで、議事録をドキュメントの一部として組み込むことができます。別のファイルを新たに作成した場合は、ここにファイル名を並べていきます。

これで変換の準備が整いました。`make html`コマンドを実行してHTMLを生成してみましょう(**図3**)。

▼リスト2　sample.md

```
Sphinxサイト ミーティング 6/30
==============================
* 日時: 2000/06/30 10:00 - 12:00
* 参加者: shimizukawa, tk0miya, usaturn, r_rudi


進捗状況について
----------------
まず進捗状況の共有を行いました。
前回ミーティング [Sphinxサイト ミーティング 5/13](meeting-0513.html)
からの進捗確認。

* サイト概要: 未着手
* Sphinxの紹介: 大まかに完了。 *肉付けと見直しが必要*
* インストールページ: **完了**

<!--
公式ドキュメント翻訳については省略
-->


### 検討課題

1. Sphinxの紹介で、以下の絵のような、Sphinxの全体像を
   表すイメージ図が必要

   ![sphinx-flow.png](sphinx-flow.png "入力から出力までの全体像、名称")

2. [sphinx-doc.org] の紹介とリンクを追加しよう

3. [進捗状況について](#進捗状況について) で確認したような進捗を自動的に
   確認する方法

議事録に補足があればbitbucketの
[ここ](https://bitbucket.org/user/path) でコメントを
付けてください。

[sphinx-doc.org]: http://sphinx-doc.org/
```

生成されたHTMLは「_build/html/」ディレクトリ以下に出力されます。「_build/html/index.html」をブラウザで開いて、閲覧してみてください(**図4**)。

▼図3　make htmlを実行

```
$ make html
Running Sphinx v1.3.6
making output directory...
loading pickled environment... not yet created
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 2 source files that are out of date
updating environment: 2 added, 0 changed, 0 removed
reading sources... [ 50%] index
reading sources... [100%] sample

looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [ 50%] index
writing output... [100%] sample

generating indices... genindex
writing additional pages... search
copying static files... done
copying extra files... done
dumping search index in English (code: en) ... done
dumping object inventory... done
build succeeded.

Build finished. The HTML pages are in _build/html.
```

▼図4　Markdownから生成したHTML

Sphinxで使えるMarkdown記法

オリジナルのMarkdown[注5]はJohn Gruber氏が開発しました。Markdownは、HTMLマークアップを覚えなくても、読み書きしやすいテキストベースのシンプルな記法からHTML文書を簡単に生成できるようにするために作られました。HTMLは文章に対して「マークアップ」して装飾や意味を「書き加えていく」ことで、デザインや構造などを表現します。これに対して、Markdownは装飾や意味を表す記述をそぎ落として、できるだけ簡単な記述で書きつつ、同様の結果を得られるようになっています。

このように、オリジナルのMarkdownはシンプルな記法を目指して作られました。その後、多くの派生Markdown記法が生まれ、テーブル記法などオリジナルでは提供されていない記法をそれぞれ独自に追加しました。有名なのはGitHub Flavored Markdown(GFM)[注6]やPHP Markdown Extra[注7]などでしょう。そして、Markdownの標準化と機能強化を目指すCommonMark[注8]の策定が2014年に始まりました。現在のところ、CommonMark記法はオリジナルの記法よりも表

注5) http://daringfireball.net/projects/markdown/syntax
注6) https://help.github.com/articles/github-flavored-markdown/
注7) https://michelf.ca/projects/php-markdown/extra/
注8) http://spec.commonmark.org/

現力はあるものの、テーブル記法はまだサポートされていません。これからの議論で仕様が標準化されていくことが期待されています。**表1**は、マークアップによって表現できることの差異についてまとめたものです。

本記事ではrecommonmarkパーサーを使用しています。このパーサーは、名前のとおりCommon Mark記法をサポートしています。

SphinxでMarkdown記法を利用するメリット

すでにMarkdownでドキュメントを書いている人は、Sphinxを使ってビルドすることによって、HTMLやPDFなどの各種フォーマットへの変換[注9]や、gettextを利用した他言語への翻訳出力、といったSphinxの機能をすぐに利用できます。また、MarkdownとreStructuredText(以下、reST)のどちらも書いたことがない人にとっては、Markdownは記法が少なく覚えやすいので、最初のハードルが低いというメリットがあります。

これまでのSphinxはreSTでしか記述できなかったため、Sphinxを使ううえでのハードルとなっていました。Markdownは最初のハードルが低く、よく知られているため、Markdown記法から始めることでSphinxを導入しやすくなるでしょう。

ではreST記法のメリットは?

reSTは、1つのドキュメントソースをさまざまな出力形式に変換することを目標に作られています。このため、表現の幅はとても広く、そのぶん数多くの記法が用意されています。これまでの連載で紹介してきた範囲だけでも、目次記法、ページ間の相互参照、テーブル記法、図表への自動採番、図表番号の参照、用語集、索引の自動生成、Graphvizによる図の描画などがあります。ほかにも外部ファイルの埋め込みや

注9) 前回紹介したmake epubを実行すれば「_build/epub/」ディレクトリにEPUBファイルが生成されます。

▼表1 マークアップごとの記法の違い

表現	CommonMark	GFM	reStructuredText
強調、斜体、インラインリテラル	○	○	○
絵文字、チェックボックス、打ち消し線	×	○	×
箇条書き	○	○	○
定義リスト、フィールドリスト、オプションリスト	×	×	○
引用ブロック	○	○	○
画像表示	○	○	○
URLリンク	△(専用記法が必要)	○(直接記述も可能)	○(直接記述も可能)
ページ内リンク	△	△	○
ページ内目次	×	×	○
相互参照	×	×	○
テーブル表記	×	○	○
脚注	×	×	○
番号付き参照	×	×	○
HTMLタグ	○(直接記述)	○(直接記述)	○(rawディレクティブで可能)
コメントアウト	△(HTMLのコメントアウトを使う)	△(HTMLのコメントアウトを使う)	○
外部ファイルinclude	×	×	○(include、literalinclude)
role、ディレクティブによる記法拡張	×	×	○
テキストからの図生成	×	×	○(Sphinx拡張で可能)

脚注など、多くの記法があります。

また、reSTプラグインによって記法を拡張できます。Sphinxはこのしくみを使って、オリジナルのreSTでは提供されていないいくつかの記法を追加しています。前述の`.. toctree::`もそのうちの1つで、複数あるドキュメントソースファイルを1つのツリー構造に連結することで、ドキュメントの論理構造を決定します。

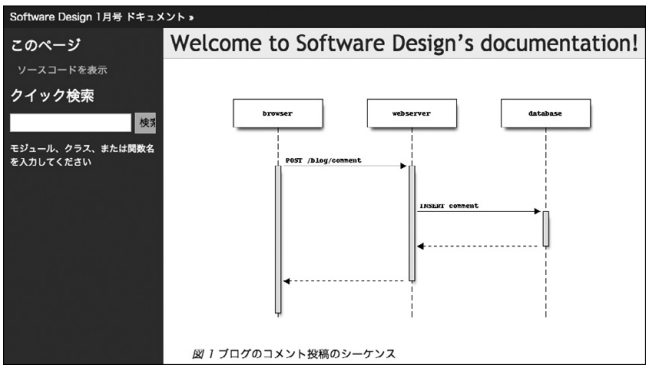
ほかにも有志が作成したSphinx拡張を利用することで、さまざまな機能が使えるようになります。たとえば、「sphinxcontrib-cacoo」や「sphinxcontrib-visio」などによる外部からの図の埋め込み、「sphinxcontrib-seqdiag」の専用記法によるシーケンス図の描画(**図5**)などが挙げられます。

こういった機能を利用するには、Markdownではなく、reSTで記述する必要があります。そこで、まずはCommonMark記法でドキュメントを書いていき、それ以上の表現力が必要になった場合には、reSTで記述することを検討してみてください。

次回予告

次回は、Sphinxの国際化機能について紹介します。gettextを利用した他言語への翻訳出力は、ドキュメントソースがreSTでもMarkdownでも利用できます。英語ドキュメントの日本語への翻訳などを検討している方は、ぜひご参照ください。SD

▼図5　sphinxcontrib-seqdiagの出力結果



COLUMN

recommonmarkの拡張記法

recommonmarkは独自の拡張記法を持っています。これを利用すれば、Markdownドキュメントの中にreST記法でディレクティブを記述したり、数式を記述したりといったことも可能です。ただし、拡張記法は「新しいMarkdownの派生」なので、ほかのCommonMarkと互換性がなくなることに注意してください。使い方についてはrecommonmarkのドキュメント[注A]を参照してください。

注A) http://recommonmark.readthedocs.org/

COLUMN

Sphinx-1.4 alpha1リリース

Sphinx開発チームは、Sphinx-1.4のalpha1をリリースしました。本誌が発売されているころには1.4の正式版もリリースされている予定です。

最新のSphinxでは、多くの機能が追加されています。たとえば、用語集でのカテゴリ指定、EPUB3ビルダー、日本語検索にJanomeを選択可能、Sphinx拡張にsphinx.ext.githubpagesとsphinx.ext.autosectionlabelを追加など、35個の機能が追加されました。

使ってみて感想や不具合などがありましたら、ぜひメーリングリストまでご連絡ください。よろしくお願いします。

Mackerelではじめるサーバ管理

Writer 田中 慎司(たなか しんじ) ㈱はてな
Twitter @stanaka

第14回 式を使って柔軟なグラフを書こう

Mackerelの肝となる機能「メトリックのグラフ化」を発展させ、「式」を組んで値を計算し、グラフ化する方法を解説します。過去のグラフを同時に表示して変化を見たり、差分グラフを作って変化量を把握したりと、監視対象に何が起きているか、より詳しい分析が可能になります。

メトリックを式で計算する

Mackerel[注1]では、さまざまなメトリックを投稿し、それらをグラフとして可視化できます。今回は、単純に投稿したものを表示するだけではなく、式を利用して投稿したメトリックをさまざまに加工してグラフ化する方法を紹介します[注2]。

サーバの負荷の可視化を行っていると、初期のころはCPU使用率やMySQLのクエリ数などの値を直接投稿してグラフ化するだけで一定の効果は得られるのですが、徐々にそれらの値を組み合わせて、一歩進んだ分析・可視化を行いたくなってきます。Mackerelではそのための機能として、式を定義することでメトリックの値を計算し、グラフとして可視化する機能を備えています。

式を利用することで、たとえば次のような値をグラフ化できるようになります。

- memcachedのキャッシュヒット率
- loadavg5の過去と現在の値の比較
- fluentdの各プラグインの最大キュー長

これらの値は、投稿された複数のメトリックを対象に四則演算を行ったり、読み込みの対象時間帯をずらしたり、複数のメトリック値から最大値を取得したりすることで計算できます。Mackerelではこれら計算を柔軟に行うための関数を提供し、式に組み込めるようにしています。

式の仕様と利用例

式によるグラフは、「**https://mackerel.io/orgs/<オーガニゼーション名>/advanced-graph?query=<メトリック>&unit=<単位>&title=<タイトル>**」のようなURLで利用できます。式は<メトリック>部分に書いていきます。またtitleとunitのパラメータは省略できます。

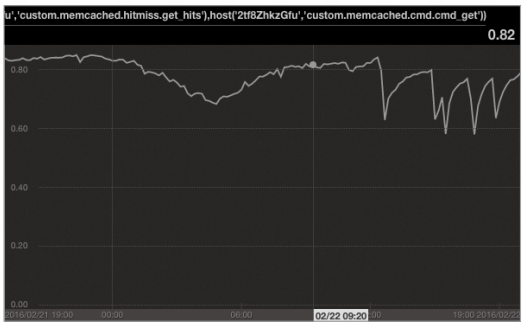
たとえば、memcachedのキャッシュヒット率のグラフは**リスト1**のURLとなります。式の部分を取り出すと**リスト2**のようになります。

▼リスト1　memcachedのキャッシュヒット率のグラフのURL

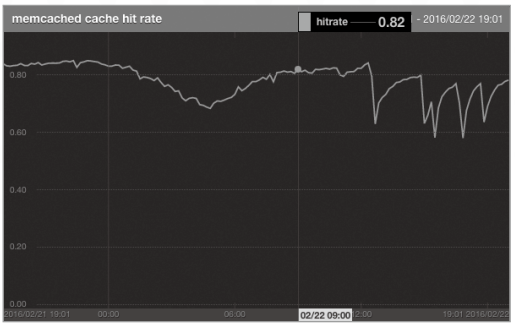
```
https://mackerel.io/orgs/<オーガニゼーション名>/advanced-graph?query=divide(host(%272tf8ZhkzGfu%27,%27custom.memcached.hitmiss.get_hits%27),host(%272tf8ZhkzGfu%27,%27custom.memcached.cmd.cmd_get%27))&unit=&title=memcached
```

注1）URL https://mackerel.io
注2）この式を利用したグラフ描画は、現在は実験的機能として提供されています。
URL http://help-ja.mackerel.io/entry/advanced/experimental-features-config

▼図1　キャッシュヒット率の計算



▼図2　ラベルを付与



この式は、host関数で取得した2つのメトリックに対してdivide関数で除算をしています。host関数は第一引数で指定したIDのホストから第二引数で指定したメトリックを取得します。

ここでは、「2tf8ZhkzGfu」というIDのホストから、custom.memcached.hitmiss.get_hitsとcustom.memcached.cmd.cmd_getのメトリックを取得しています。前者のメトリックはmemcachedのGET命令実行時のヒット数で、後者はGET命令の実行数になります。前者を後者で割ることでヒット率の計算ができますので、その除算をdivide関数で実行しています。この式をそのままqueryパラメータに埋め込むことでグラフを描画させることができます(**図1**)。

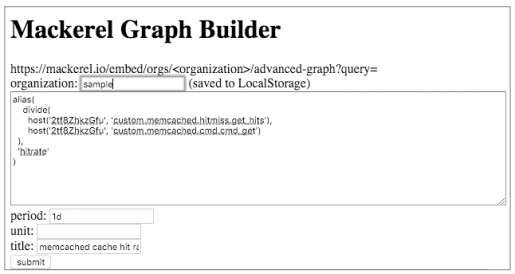
次に、この系列に名前を付けるためにalias関数を利用して「hitrate」という名前を付けます(**リスト3**)。こうすることで、マウスオーバーした際のラベルを適切なものに変更できます(**図2**)。

式の組み立て方

執筆時点(2016年2月)では、式を組み立てるためのUIはまだサポートできておらず、自力で適切なqueryパラメータを組み立てる必要があります。

ただ、この試行錯誤を人手で行うのはかなり煩雑であるため、簡単なヘルパーページ[注3]が公開されています。

▼図3　式組み立てヘルパー



ヘルパーページを利用すると、テキストエリア上でインデントさせつつ式を構築できます(**図3**)。式を書き終えて[submit]ボタンを押すと、下にiframeで埋め込まれたグラフが表示されます。式が正しくない場合は空のグラフになってしまいますので、適切に修正してください[注4]。

▼リスト2　リスト1の「式」部分

```
divide(
  host('2tf8ZhkzGfu', 'custom.memcached.hitmiss.get_hits'),
  host('2tf8ZhkzGfu', 'custom.memcached.cmd.cmd_get')
)
```

▼リスト3　alias関数で名前を付ける

```
alias(
  divide(
    host('2tf8ZhkzGfu', 'custom.memcached.hitmiss.get_hits'),
    host('2tf8ZhkzGfu', 'custom.memcached.cmd.cmd_get')
  ),
  'hitrate'
)
```

注3) URL https://gist.github.com/stanaka/eae027169de6a35dd76a
注4) このあたりはまだまだ不親切ですので、今後改善していきます。

利用可能な関数

前述の利用例ではdivideとhostを紹介しましたが、ほかにもさまざまな関数が使用できます。使用可能な関数の一覧を**表1**に示します。すべての関数はメトリック型(metrics)を返します。また関数の引数の要素は**表2**のとおりです。

具体例：過去の値との比較

式の利用例として、あるメトリックを過去のものと比較する方法を紹介します。timeShift関数を利用することでメトリックの過去の値を取得できます。

たとえば1週間前の値との比較を行い、同じ曜日でのメトリックの挙動の差分を見る、ということができます[注5]。具体的に1週間前の値と比較する場合は**リスト4**のようにします。roleSlots関数は第一引数で指定されたロールに所属するホストの、第二引数で指定されたメトリックを表示します。次に、timeShift関数で第一引数に先ほどのメトリックを指定し、第

▼表1　使用可能な関数

関数	説明	例
host(hostId, metricName)	ホストメトリックを返す	host('22CXRB3pZmu', 'memory.*')
service(serviceName, metricName)	サービスメトリックを返す	service('Blog', 'access_count')
role(roleFullname, metricName)	ロールに現在所属しているホストのメトリックを返す	role('Blog:db', 'memory.*')
roleSlots(roleFullname, metricName)	ロールのメトリックを返す。過去にロールに所属していたホストから送られた一部のメトリック※も取得できる	roleSlots('Blog:db', 'loadavg5')
avg(metrics)	各時刻ごとに引数のメトリックの平均値を返す	avg(group(host('22CXRB3pZmu','loadavg5'), host('22CXRB3pZmu','loadavg5')))
max(metrics)	各時刻ごとに引数のメトリックの最大値を返す	max(host('22CXRB3pZmu','custom.foo.jobs.*'))
min(metrics)	各時刻ごとに引数のメトリックの最小値を返す	min(host('22CXRB3pZmu','custom.foo.jobs.*'))
sum(metrics)	各時刻ごとに引数のメトリックの合計を返す	sum(host('22CXRB3pZmu','custom.foo.jobs.*'))
product(metrics)	各時刻ごとに引数のメトリックを掛け合わせた値を返す	product(group(service('Blog:db','foo.bar'), service('Blog:db','foo.baz')))
diff(metrics, metrics)	各時刻ごとに1つめの引数のメトリックから2つめのメトリックを引いた値を返す	diff(service('Blog:db','foo.bar'), service('Blog:db','foo.baz'))
divide(metrics, metrics)	各時刻ごとに1つめの引数のメトリックを2つめの引数のメトリックで割った値を返す	divide(service('Blog:db','foo.bar'), service('Blog:db','foo.baz'))
scale(metrics, factor)	定数倍したメトリックを返す	scale(service('Blog:db','foo.bar'), 10.0)
timeShift(metrics, duration)	指定した期間分時刻をずらしたメトリックを返す	timeShift(service('Blog:db','foo.bar'), '1d')
movingAverage(metrics, duration)	移動平均	movingAverage(service('Blog:db','foo.bar'), '1d')
group(metrics, metrics, ...)	引数のメトリック列を1つにまとめる	group(service('Blog:db','foo.bar'), service('Blog:db','foo.baz'))
stack(metrics)	グラフをスタック表示	stack(service('Blog:db','foo.bar'))
alias(metrics, displayName)	メトリックの表示名をカスタマイズ	alias(service('Blog:db','foo.bar'), 'blog:db:bar')

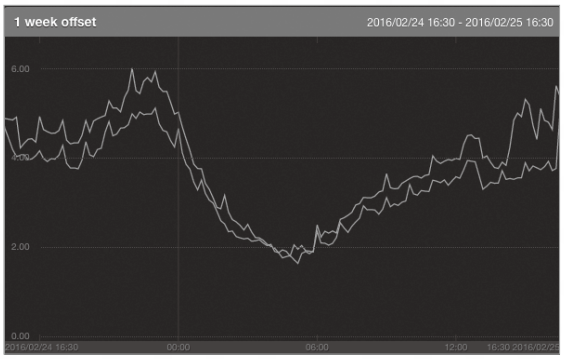
※ loadavg5, processor_queue_length, cpu.user.percentage, cpu.iowait.percentage, cpu.system.percentage, interface.rxBytes.delta, interface.txBytes.delta, disk.reads.delta, disk.writes.delta, memory.used, memory.cached

注5) この手の比較を行う場合は、同じ曜日でも祝日のときには挙動が異なることが多いため、注意してください。

▼表2　関数の引数の要素

引数	型	説明	例
hostId	string	ホストID	'22CXRB3pZmu', "22CXRB3pZmu"
metricName	string	メトリック名	'loadavg5'
serviceName	string	サービス名	'Blog'
roleFullname	string	サービス名とロール名を：で連結したもの	'Blog:db'
metrics	metrics	メトリック列	alias(host('22CXRB3pZmu', 'loadavg5'), 'L5')
displayName	string	表示名	'blog max loadavg5'
duration	string	整数に続けて単位を指定した期間。使用できる単位はm(分)、h(時間)、d(日)、w(週)、mo(月)、y(年)	'5m', '1d'
factor	float	係数	1.5, 200

▼図4　1週間前の値との比較



二引数で遡りたい時間を指定します。そして、avg関数によりそれらのメトリックの値の平均を計算します。さらにalias数でメトリックの名前を指定し、group関数でこれらのメトリックをまとめて表示できるようにします(**図4**)。

過去の値と重ね合わせるのではなく、差分を計算したい場合はdiff数を利用します(**リスト5**)。

まとめ

今回は式を利用して、メトリック同士を計算してグラフ化する方法を紹介しました。現在はグラフ化までですが、近日中に計算結果を対象に監視を行えるようにする予定です。

式を利用することで、サーバ／インフラの挙動がより把握しやすくなります。障害の兆候を先取りでつかんで先回りで対処するためにも、ぜひ式によるグラフ化を試してみてください。SD

▼リスト4　1週間前の値と比較し、グラフを重ね合わせる式

```
group(
  alias(
    avg(
      timeShift(
        roleSlots('some:app','loadavg5'),
        '1w'
      )
    ),
    'avg-lastweek'
  ),
  alias(
    avg(
      roleSlots('some:app','loadavg5')
    ),
    'avg'
  )
)
```

▼リスト5　1週間前の値と比較し、差分を計算する式

```
alias(
  diff(
    avg(
      timeShift(
        roleSlots('some:app','loadavg5'),
        '1w'
      )
    ),
    avg(
      roleSlots('some:app','loadavg5')
    )
  ),
  'diff'
)
```

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第29回 ❖bhyveでOpenBSDファイアウォール on FreeBSDを構築（その4）

◉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。

OpenBSD on FreeBSD ——ステートの重要性

FreeBSDハイパーバイザでOpenBSDを動作させ、そこでOpenBSDネイティブのパケットフィルタリング／ファイアウォール機能の最新版pf(4)を使ってみよう、というシナリオで、ここ数回にわたってpf(4)の概要、マクロ、リスト、テーブル、基本的なシンタックスなどを紹介してきました。

pf(4)はステートテーブルにコネクションごとの情報を保持しています。この情報を利用することで、たとえばすでに接続が完了したパケットでルールセットを経由する必要がないものに関しては、即座にファイアウォールを通過して通信を行うといったことが可能になります。この機能によってルールセットをシンプルに保つことが可能になるほか、パフォーマンスを向上させることにもつながります。

パケットフィルタではステートの保持やステートを加味した処理が重要になってきます。pf(4)ではネットワーク接続の状態や進捗状況をステートテーブルに保持しており、この情報を使った処理が可能です。今回はステートを中心に説明します。

keep state/ modulate state/ no state

pf(4)のpassルールはそのルールに一致したパケットに関して自動的にステートエントリを作成するしくみになっています。たとえば次のように、vio0（VirtIOネットワークデバイス）を経由した外側へのTCPパケットを許可した場合、自動的にこのパケットの返信パケットに関しても通行が許可されるようになります。

```
pass out on vio0 proto tcp all
```

これは次のようにkeep stateオプションを指定したことと同じ処理になります。

```
pass out on vio0 proto tcp all keep state
```

このようにステートを保持することで毎回ルールセットをチェックしにいく処理を回避できるため、結果的に処理速度の向上につながっています。このステート保持を明示的に避けたいのであれば、次のようにno stateオプションを指定します。

```
pass out on vio0 proto tcp all no state
```

keep stateと似たような処理をするオプションにmodulate stateがあります（**リスト1**）。これはkeep stateと同じように動作しますが、適用対象がTCPパケットだけに限定されることと、外に出て行くTCPパケットの初期シーケンス番号（ISN；Initial Sequence Number）がランダム化されるという特徴があります。

modulate stateはTCPパケットにのみ適用されますが、記述を簡単にする目的から**リスト2**のようにほかのプロトコルと組み合わせても記述できるようになっています。**リスト2**のルールセットは**リス**

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。

ト3のように個別のルールセットを記述したのと同じになります。

modulate stateはTCPパケットに限定した指定ですが、pf(4)では通常のステートはUDPに対しても作成されます。UDPそのものはステートレスなプロトコルであるためUDPに対してはステートは作成されないと思われがちですが、pf(4)ではUDPに対してステートを作成することができます。pf(4)はルールセットに一致した段階でステートを作成し、タイムアウトになるまでステートを保持します。タイムアウトの時間もpf(4)で設定できます。

ステートオプション

ステートはpassルールで自動的に作成されるわけですが、ルールにはステートに関するオプションが指定できるようになっています。たとえばno stateはステートを作成しないようにするというステートオプションの1つです。次のようなオプションが用意されています。

- max number……ルールに対して保持できるステートの数の上限を指定する。作成されたステートが上限数に到達している場合、それ以降でルールに一致したパケットに対してステートは保持されなくなる。保持しているステート数が上限値を下回ると、以降はまたルールに一致したパケットに対してステートが作成されるようになる
- source-track rule(またはglobal)……ソースIPアドレスごとに作成するステートに関して上限数を指定できるようにするオプション。ruleが指定されている場合にはルールごとの指定、globalが指定されている場合にはグローバルな指定となる
- max-src-nodes number……同時にステートを作成することができるソースIPアドレスの上限数を指定する。source-track ruleと同時にのみ指定できる
- max-src-states number……ソースIPアドレスに対して同時に作成できるステートエントリの上限数を指定する。制限の範囲はsource-trackオプションの指定に依存する
- no state……ステートを自動的に作成しないようにする

※numberは任意の数値

TCPコネクションに対してはさらに次のオプションによる規制を指定することができます。

- max-src-conn number……単一のホストにおいて同時に3-wayハンドシェイクを完了させることができるTCPコネクションの上限数を指定する(source-track ruleが有効になる。source-track globalとは非互換)
- max-src-conn-rate number(またはinterval)……指定された期間において作成できる新しいコネクションの上限を指定する(source-track ruleが有効になる。source-track globalとは非互換)
- overload <table> flush……上限を超えたIPアドレスに対してすべてのステートをフラッシュする

※intervalは任意の数値、tableは任意のテーブル名

▼リスト1　modulate stateを指定した場合

```
pass out on vio0 proto tcp all modulate state
```

▼リスト2　TCP以外のプロトコルと組み合わせた記述も可能

```
pass out on vio0 proto {tcp,udp,icmp} all modulate state
```

▼リスト3　リスト2と同じ意味の複数のルールセット

```
pass out on vio0 proto tcp all modulate state
pass out on vio0 proto udp all
pass out on vio0 proto icmp all
```

チャーリー・ルートからの手紙

これらのオプションはたとえば**リスト4**のように指定します。この指定によって、このルールに対して同時に作成されるノードの最大数は100個、ソースIPごとに同時に作成できるステートの上限数は3個に制限されます。

TCPフラグ

pf(4)ではTCPパケットに対してデフォルトで`flags S/SA`というルールが指定されたのと同じ状態に設定されています。flagsの最初の指定はヘッダで指定されているTCPフラグ、2つ目の指定は指定されたフラグのみをチェックするというものです。

次の2つのルールは同じことを意味しています。

```
pass out on vio0 proto tcp all
pass out on vio0 proto tcp all flags S/SA
```

flagsで指定できるフラグは次のとおりです。

- F…… FIN（セッションの終了）
- S…… SYN（セッションの開始要求）
- R…… RST（コネクションのリセット）
- P…… PUSH（パケットを即座に送信）
- A…… ACK（ACKパケット）
- U…… URG
- E…… ECE
- W……CWR

`flags S/SA`がデフォルトで指定された状態になっているため、passのルールは基本的にセッションの開始要求時、さらにSYNとACKが指定されたもののみが対象となります。セッションのスタートでもACKのみの場合には対象となりません。つまり、通信の最初のパケットが一致の対象となり、一致したあとはステートによって処理が行われることになります。

しくみとしてはこうなっていますが、通常flagsを明示的に指定することはないでしょう。`flags S/SAFR`のように指定することもできますが、2つ目の引数のFRは結果的に不要ですので、S/SA（デフォルト）で充分ということになります。

TCP SYN flood攻撃対策

`keep state`や`modulate state`と似たような指定に`synproxy state`があります。これはTCP SYN flood攻撃に対する防御壁として利用できる機能です。`synproxy state`指定がない場合、pf(4)はTCP接続の最初の段階であるハンドシェイクをすべてサーバに対して透過的に渡します。`synproxy state`を指定するとハンドシェイク部分はpf(4)が担当するようになり、接続が確立した段階で処理をサーバにまで持っていくようになります。

TCP SYN flood攻撃を受けるとサーバ側で接続がさばけないという事態が発生するわけですが、この前段階でpf(4)がTCP SYN flood攻撃を落とすようになりますので、サーバをその分だけ保護することにつながります。TCP SYN flood攻撃を受けている場合にはいちど試してみるとよい機能です。

IPスプーフィング対策

送信元IPアドレスを詐称してパケットを送信するといった攻撃を受けることがあります。pf(4)にはこうした攻撃をブロックするための`antispoof`というルールが用意されています。

```
antispoof for vio0 inet
```

たとえば上記ルールは、インターフェースに割り当てられているIPアドレスを用い、**リスト5**のようなルールに展開されて使用されます。展開される内容は

▼**リスト4　オプションを指定したステート指定の例**

```
pass in on vio0 proto tcp \
  from any to 192.168.1.24 port = 80 \
  keep state \
  (source-track rule, max-src-states 3, max-src-nodes 100)
```

割り当てられているIPアドレスによって変わります。

antispoofというキーワードで自動的に展開されるため、簡単にIPスプーフィング対策が記述できます。

オペレーティングシステムフィルタリング

pf(4)には、オペレーティングシステム(OS)のTCP SYNパケットの特性に基づいてリモートホストのOSを判別し、フィルタの対象とする機能(OSFP;パッシブOSフィンガープリント)を提供しています。どのOSをフィルタリングできるかは/etc/pf.osファイルを見るとわかります。

または指定として記述しやすい出力がほしいなら、**図1**のようにpfctl(8)コマンドを実行してリスト表示させることができます。

たとえば**リスト6**のように指定できます。条件によっては検出できなくなるほか、新しいOSは検出できません。古いOSからのアクセスを拒否するといった場合に使うことになると思います。

パケットフィルタリングの基本

これまでの記事でパケットフィルタリングの機能の基本的な部分はほぼ網羅しました。あとはNATとポートフォワーディングが使えるようになると、ファイアウォールとしての基本はおさえられます。発展系としてポリシーフィルタリングやロードバランシング、CARPとpfsyncを使った冗長性構成などもありますが、そこまでの話題はこの連載の範囲を超えますので別の機会に譲ります。

次回、NATとポートフォワーディング、ランタイムオプションとルールのショートカット表記あたりを取り上げて、OpenBSD pf(4)の説明をいったん切り上げようと思います。SD

▼**図1　OSFPで指定できるオペレーティングシステム一覧の表示**

```
# pfctl -s osfp | head
Class   Version Subtype(subversion)
-----   ------- ------------------
AIX
AIX     4.3
AIX     4.3     2
AIX     4.3     2-3
AIX     4.3     3
AIX     5.1
AIX     5.1-5.2
AIX     5.2
# pfctl -s osfp | tail
Windows NT
Windows NT      4.0
Windows Vista
Windows XP
Windows XP      cisco
Windows XP      RFC1323
Windows XP      SP1
Windows XP      SP3
Zaurus
Zaurus  3.10
```

▼**リスト5　展開されたantispoofルール**

```
block drop in on ! vio0 inet from 192.168.1.0/24 to any
block drop in inet from 192.168.1.36 to any
```

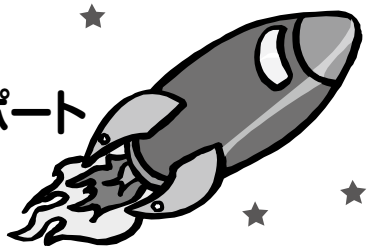
▼**リスト6　OSFP機能の使用例**

```
block in on vio0 proto tcp from any os "Windows 98"
block in on vio0 proto tcp from any os "Windows ME"
block in on vio0 proto tcp from any os "Windows 2000"
block in on vio0 proto tcp from any os "Windows XP"
block in on vio0 proto tcp from any os "Cisco"
block in on vio0 proto tcp from any os "Contiki"
block in on vio0 proto tcp from any os "DOS"
block in on vio0 proto tcp from any os "FortiNet"
block in on vio0 proto tcp from any os "Linux 2.0"
block in on vio0 proto tcp from any os "Linux 2.2"
block in on vio0 proto tcp from any os "NMAP"
block in on vio0 proto tcp from any os "Novell"
block in on vio0 proto tcp from any os "SymbianOS"
```

34

Debian Developer　やまねひでき　henrich@debian.org

Hyper-VでDebianをフルサポート相当にするための挑戦



Debian Hot Topics

最近のホットトピック

デバッグ用パッケージ“-dbgsym”の自動生成

利用しているプログラムの動作に問題があってクラッシュしたときなどに、GDB[注1]を使ってスタックトレースを追う、ということをされたことのある方も多いかと思います。Debianで同様にクラッシュするソフトウェアのパッケージに対してGDBを使って……とやろうとすると、残念ながらうまくいきません。この理由は、Debianパッケージのポリシーにより、デフォルトではスタックトレースを追うのに必要なデバッグシンボル情報が削られるようになっているからです。

ではDebian上ではどうやってデバッグを行うかと言うと、<package>に対してデバッグシンボルを含んだ別パッケージである<package>-dbgをインストールします。これでデバッグできるようになるので、万事解決です……と言いたいところなのですが、残念なことに、すべてのパッケージにデバッグパッケージが存在しているわけではなかった(パッケージメンテナが明示的にソースパッケージに記述しないと作成されません)ので、パッケージに含まれているプログラムをデバッグしたいときに困るということがままありました。

これに対し、debhelper[注2]の新機能として、特別な設定をしなくても自動でデバッグ用パッケージ「<package>-dbgsym」が生成されるようになりました。たとえば、筆者がメンテナンスしているlibxmlbirdのパッケージであれば「libxmlbird1-dbgsym」というような名称になります。

また、これまでデバッグパッケージがデフォルトでは作成されていなかった理由として、そのサイズの大きさからパッケージリポジトリを圧迫する、という点が挙げられていましたが、この問題は分離したリポジトリとミラーを使うことで解決するようです。dbgsymパッケージの取得をしたい場合はapt lineを

```
deb http://debug.mirrors.debian.org/
debian-debug/ unstable-debug main
```

という形で追加します。この記事を書いている段階では、ビルドされたdbgsymパッケージの数はまだ少ないですが、徐々に充実してくることでしょう。

GitLab、Debian公式パッケージになる

本誌2015年12月号で取り上げたgitlabパッケージですが、無事に依存関係にあるすべての

注1) The GNU Project Debugger。CやC++などで書かれたプログラムのデバッグで利用されるスタンダードなツール。
URL https://www.gnu.org/software/gdb/

注2) Debianパッケージの作成の肝になるツール。Perlで実装されている。

パッケージ(300個超らしいです……)がDebian公式パッケージとなり、unstableへアップロードされました。これで別途リポジトリを追加しなくてもgitlabが使えるようになりましたので、興味のある方は試してみてください。

Hyper-Vと「サポート」ディストリビューション

現在の市場にはさまざまな仮想化プロダクトがあります(**表1**)が、シェアは意外なことに、Microsoft社のHyper-Vが2012年ごろからトップだそうです(おそらくWindows Serverの出荷数も含まれるからだと思いますが)。

Hyper-V上でLinuxがサポートされ始めたころは、ゲストOSであるLinux側でサポート用のドライバを別途インストールする必要がありました。しかし今では、このドライバがLinuxカーネルのmainline(本体)にマージされ、最新バージョンであればどのディストリビューションでも手間をかけずに利用できるようになりました。「Hyper-V上でDebianをとりあえず使ってみる」という点では、とくに不具合はありません(Debianでは、Debian 7「Wheezy」からサポートされています)。

しかし、Hyper-Vの説明によると「フルサポートするディストリビューション」は限られており、その中にはDebianはありません。よくよく見ていくと、サポートされるディストリビューションでは「LIS(Linux Integration Services)」というものがあり、Windows向けのHyper-Vサポート(「統合サービス」)と同様の機能をLinux向けに提供しています。本稿執筆時点では、LISのバージョンは4.0が最新ですが、Debian用にはリリースされていません[注3]。LISの機能は**表2**のようになります。

表2にある機能をフルで利用したい場合は、カーネル組み込みのドライバと、追加のデーモンパッケージが必要となるにもかかわらず、Debian 8にはLIS 4.0で提供されているデーモンが存在しないために「一部機能のみサポート」となってしまいます。

Debianもサポート(相当)にしたい

ここでDebianを愛する筆者的には「サポート外とされてしまうのは、ちょっと悔しいな」と思いました。そして、「LISに含まれるデーモン相当のパッケージを作って、Debianに放り込めば、サポートディストリビューション相当として取り扱われるだろう」ということで、何とかできないかと調査を始めました。

- Ubuntuはサポートされているので、Ubuntu用のLISでインストールされるパッケージの中身をもってくれば良いだろう
- Ubuntuでのサポートパッケージ相当は、Debianではなんだろう? どうやら、Debian

注3) URL https://www.microsoft.com/en-us/download/details.aspx?id=46842

▼表1 仮想化プロダクトと特徴

プロダクト	特徴
VMWare	プロプライエタリ。x86アーキテクチャ上での仮想化のはしりであり、現在も最大の仮想化基盤製品
KVM	Red Hat社が買収したQumranet社により開発された。現在はLinuxカーネルに組み込まれている
Xen	ケンブリッジ大学で開発が始まり、XenSource社が開発を進めた。後にCitrix Systems社が買収したが、現在はXen ProjectがThe Linux Foundationへ寄贈されて開発が継続されている(有名どころだとAWSもXenベース)
VirtualBox	当初はドイツのInnotek社が開発、これをSun Microsystems社が買収し、さらにOracle社に買収された。OSSだが一部機能はプロプライエタリ。昨今では仮想化環境構築ツール「Vagrant」のバックエンドとして利用されることが多い
Hyper-V	プロプライエタリ。Microsoft社のWindows Server製品やWindowsのProfessional Edition以上で利用可能

では「linux-tools」というソースパッケージ[注4]のようだ。ここで何か足りないものがあるのだろう

- linux-toolsパッケージをいじくって、ソース内のtools/hvディレクトリ以下をビルドして、Hyper-Vサポート用の別パッケージにインストールするようにすれば良さそうだ
- ほかのディストリビューションパッケージと名称を合わせたほうがユーザにやさしいだろうと確認して、名称は「hyperv-daemons」に決定(Ubuntuは「hv-kvp-daemon-init」という名前でわかりづらかった)
- CentOS/Red Hat Enterprise Linuxの該当のパッケージにはsystemd対応が入っていたので、こちらを流用することにして手間を省こう

注4) linux-toolsのソースはlinuxカーネルパッケージに含まれているものと同じなのですが、Debianパッケージとしてはlinuxパッケージとは分けられて、別物のlinux-toolsパッケージとして扱われています。通常であればlinuxパッケージの一部になるはずなのですが、何らかの意図で別物にしているようです。ややこしいですね。

上記のような手順を経て、linux-toolsソースパッケージに「hyperv-daemons」パッケージを追加するパッチを作りました。

パッチを作成している最中にlinux-toolsパッケージのバグレポートページ[注5]を確認したところ、「Please include tools/hv daemons in a binary package」というバグレポート[注6]が出ていたので、そちらに対して何度か投稿を繰り返しました(何度も投稿したのは、「まずは取りかかっているよ」、「作業しているよ」というのを示しておいたほうがいいな、と判断したからです)。

ある程度のところで、バグの報告者に「試してみて」とお願いしたところ「動作しているよ」ということだったので、さらにパッケージの修正を加えて「新規パッケージの登録には時間がかかるからアップロードしてほしい」とメンテナに要望を送りました。

注5) URL http://bugs.debian.org/src:linux-tools
注6) URL https://bugs.debian.org/782761

▼表2　Debian7/8でのLIS機能サポート比較

機能		Windows Serverバージョン	Debian 7	Debian 8 ※1
コア部分		2012 R2、2012、2008 R2	◎	◎
ネットワーク機能	Jumbo frames	2012 R2、2012、2008 R2	◎	◎
	VLAN tagging and trunking	2012 R2、2012、2008 R2	◎	◎
	Live Migration	2012 R2、2012、2008 R2	◎	◎
	Static IP Injection	2012 R2、2012	×	×
	vRSS	2012 R2	×	×
	TCP Segmentation and Checksum Offloads	2012 R2、2012、2008 R2	×	×
ストレージ機能	VHDX resize	2012 R2	◎	◎
	Virtual Fibre Channel	2012 R2	×	◎
	Live virtual machine backup	2012 R2	×	×
	TRIM support	2012 R2	×	×
メモリ機能	Configuration of MMIO gap	2012 R2	◎	◎
	Dynamic Memory - Hot Add	2012 R2、2012	×	×
	Dynamic Memory - Ballooning	2012 R2、2012	×	×
ビデオ機能	Hyper-V-specific video device	2012 R2、2012、2008 R2	×	◎
その他	Key-Value Pair	2012 R2、2012、2008 R2	×	×
	Non-Maskable Interrupt	2012 R2	◎	◎
	PAE Kernel Support		◎	◎
	File copy from host to guest	2012 R2	×	×
Generation 2 virtual machines	Boot using UEFI	2012 R2	×	◎
	Secure boot	2012 R2	× ※2	× ※2

参考　URL https://technet.microsoft.com/en-us/library/dn614985.aspx

※1　Debian 8.0～8.2
※2　今のところサポート予定なし

こうして、「linux-toolsパッケージのリポジトリにコミットしたから、動作確認の報告をして」とメンテナからの返信をもらったのですが、あいにく筆者の手元のマシンではHyper-Vを動作させられるようなものがなかったため、ここで若干作業が滞ることになりました。

とはいえ、悩んでいてもしかたがない、とDebian JPのメーリングリストに協力者の募集をしたところ、本誌のSamba記事でもお馴染みの、たかはしもとのぶさんに検証をお願いすることができました。

検証内容を添えてメンテナに返信したところ、並行して検証目的のexperimentalリポジトリにアップロードしてもらえ、さらにunstableリポジトリを経てtestingリポジトリまでたどり着きました。

これで、Debian 9「stretch」ではHyper-Vサポートが充実した形で利用できるようになることが、ほぼ確定しました。

現在の安定版でも使えるように

いったんtestingに入ったlinux-toolsパッケージは、Debian 8でも利用できるようにjessie-backportsリポジトリにも追加されました。

しかし、linux-toolsのバージョンが上がったのを機に、Debian 8ではカーネルとの不整合が起こってbackportsのバージョンではエラーになってしまう、と利用者からの報告がありました。

これに対して、メンテナがDebian 8のバージョンのlinux-toolsにhyperv-daemonsを追加する形で対応が行われました。そして2015年12月末に、ポイントリリースの候補リポジトリであるproposed-updatesにアップロードされ、無事に8.3リリースに追加となりました。

mission (almost) complete!

こうして最終的に、Hyper-V対応の「hyperv-daemons」パッケージがDebianの公式リポジトリに追加されることとなりました。hyperv-daemonsパッケージは現在開発中のDebian 9「strech」のみならず、Debian 8「Jessie」でも利用できるようになります。

さらに、このことをBTS(Bug Tracking System)上で情報をシェアすることで、Debianに詳しい人だけが知っているのではなく、Microsoftが提供するHyper-V公式情報の方にも反映するようにお願いしました[注7]。

ただ、1点だけ残念なことがあります。先にhyperv-daemons相当のパッケージが入っていたUbuntuからもらってきたスクリプトが、メンテナから「スクリプトがbashとPythonで複数実装があるし、Pythonもコーディング規約に従っていない。これは受け入れられない」ということで蹴られてしまい、まだ一部の機能が使えない状況にあります。こちらについては、別実装の提案が必要になります。

最後に

ある企業が公式な情報として発信している事柄を改善しよう、というのはなかなか難しく思えます。「Microsoftが自社の仮想化環境でサポートしているディストリビューション」などと聞くとなおさらそれは絶対であり、覆すことのできない決定事項であるように思えてしまうかもしれません。

しかし、実際に必要な作業は些細なものだったりして、OSSにうまく参加(Contribute)できれば変えることもできたりする、ということを、本稿を読むことで少しでもお伝えできたら……と思います。

とくにトップダウンで物事を決めない、コミュニティベースのOSSであれば、「やってやる！」という気力とほんの少しの技術力と、できる人に素直に頼ることで、意外とエイヤッと叶ってしまうものです。恐れずにチャレンジしてみてください。SD

注7) Microsoftテクノロジに精通している人が、Debianにも詳しいとは限りません。必要な情報が見つからなければ存在しないも同然です。

Raspberry Pi 2を普通のデスクトップとして使用する

Ubuntu Japanese Team
あわしろいくや

Raspberry Pi 2に簡単な方法でXubuntuをインストールし、デスクトップPCの代わりとして使用する方法を紹介します。

Ubuntu Pi Flavour Makerで環境構築が簡単に

Ubuntu 14.04 LTSのRaspberry Pi 2用イメージは以前より配布されており、インストール自体はできますが、環境構築は一からする必要があり、手間がかかります。

Ubuntu Pi Flavour Maker[注1]というプロジェクトが2015年11月に開始されました。これはUbuntu MATEのスピンアウト企画で、Raspberry Pi 2用のUbuntu（おもにそのフレーバー）イメージを作成するというプロジェクトです。このプロジェクトでは、イメージ作成用のスクリプトと作成したイメージを公開しています。このイメージは一通り必要な環境構築が済んでおり、起動後OEMインストール[注2]の開始と同じく使用する言語やユーザ名とパスワードなどの設定を一通り済ませればすぐに使える、というお手軽さです。OEMインストールに関してはご存じない方も多いと思いますので、Raspberry Pi 2のものではないもののスクリーンショットを掲示することにします（**図1**）。

デスクトップPCの代わりにするのであれば、無線LANが使用できると便利です。また、プリンタから印刷したいこともあるでしょう。本稿では、Raspberry Pi 2でそれらを行いたい場合は、どのようなところに気をつければいいのかも紹介します。

注1） https://ubuntu-pi-flavour-maker.org/

図1　OEMインストールの開始画面

イメージのダウンロードとインストール

Raspberry Pi 2のインストールイメージ転送には、Ubuntuなどがインストールされた母艦PCが必要です。Ubuntu 14.04 LTS以降がインストールされたPCと8GBのmicroSDカードが用意されているのを前提としていますが、BitTorrentでイメージのダウンロードができ、それをmicroSDカードに転送できればUbuntuでなくてもけっこうです。

注2） http://gihyo.jp/admin/serial/01/ubuntu-recipe/0001

何はなくともイメージのダウンロード[注3]を行います。Xubuntuのロゴをクリックするとtorrentファイルがダウンロードできるため、これをTransmissionなどのBitTorrentクライアントで開き、ダウンロードを開始します(**図2**)。

ダウンロードしたイメージは、[ディスク](gnome-disks)で開くと、簡単にmicroSDカードに転送できます(**図3**)。

イメージは4GBのmicroSDカードに合わせて作成されているため、8GB以上のmicroSDカードだとすべての領域を使用できません。そういった場合は、GPartedを使用して領域を拡大するのが簡単ですが、詳細は割愛します。

起動

イメージを転送したmicroSDカードをRaspberry Pi 2に挿入し、起動します。そのあと、使用する言語やユーザ名とパスワードなどの設定を一通り済ませればログインできるようになります。

最低限必要なパッケージと使用感

Raspberry Pi 2のメインメモリは1GBと心もとないため、スワップ領域を作成します。これはとても簡単で、dphys-swapfileというパッケージをインストールし、再起動するだけです。デフォルトでは2GBのスワップファイルが作成されますが、**/etc/dphys-swapfile**を編集することによって変更できます。

Raspberry Pi 2の初回起動時にインターネット接続が行われていない場合、日本語の入力に必要なパッケージがインストールされていません。[設定]-[言語サポート]からインストールしてもいいのですが、コマンドからインストールしたい場合は次を実行してください。

```
$ sudo apt-get install $(check-language-support)
```

デフォルトのリポジトリだと遅いため、ミラーサーバに変更するのもいいでしょう。その場合は**/etc/apt/sources.list**の“**http://ports.ubuntu.com/**”を“**http://jp.archive.ubuntu.com/ports/**”に書き換えてください。

カーネルをアップデートしたい場合は、次のコマンドを実行してください。

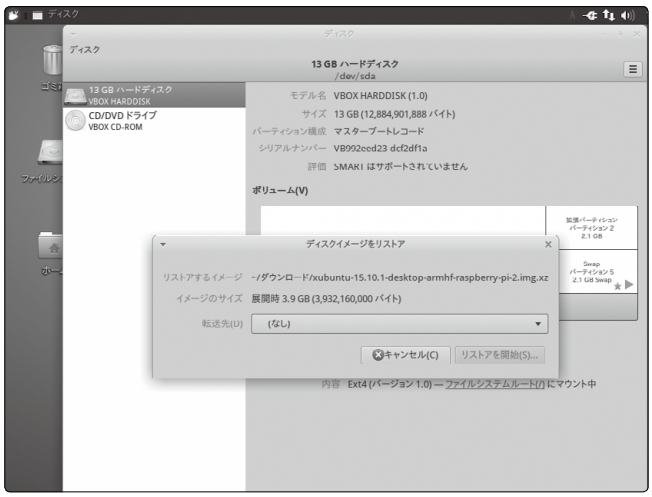
```
$ sudo rpi-update
```

ここからもわかるとおり、このイメージはUbuntuのリポジトリにあるカーネルを使用しているわけではありません。

注3) https://ubuntu-pi-flavour-maker.org/download/

図2 インストールイメージはBitTorrentでダウンロードする

図3 ディスク(gnome-disks)を使用すると簡単にイメージの転送ができる



最低限必要な設定はこれだけですが、ほかにも必要なパッケージがあればインストールしてください。VimとかEmacsとかいろいろあるのではないかと思います。

ちなみに筆者は、今回この原稿の大部分を実際にRaspberry Pi 2で執筆しています。Markdownエディタとして ReTextをインストールしました。詳細はUbuntu Weekly Recipe 第390回[注4]を参照いただきたいのですが、Qt5を使用した軽快なMarkdownエディタです。ファイルはGigoloというリモートファイルシステムをマウントするアプリケーションを使用し、ownCloudのWebDAV機能を経由してサーバに直接保存しています。Raspberry Pi 2はmicroSDカードで運用せざるを得ず、信頼性には疑問があるのでこのようにしました。リポジトリのMozcはやや古いため、自前でビルトした最新のMozc(非公開)にアップデートしてはいるのですが。

たしかに引っかかりを感じる部分はあるものの、割に快適に執筆ができました。十分に実用的といえます。

無線LANアダプタ

Raspberry Pi 2で無線LANが使えれば便利ですが、筆者が知る限りでは残念ながらUSBポートに接続しただけで即使用できるようになる無線LANアダプタは現在市販されておりません。過去にはあり、筆者はいくつか所有しています。詳しくは本連載第61回(2015年5月号)をご覧ください。

安価で容易に入手が可能であり、かつコンパクトでドライバの設定も簡単なものがいいという我がままを満たすものも探せば存在しており、今回はPlanex GW-450S[注5]を例にします。型番がよく似てるGW-450D/D2[注6]は使用しているチップが異なるため、今回の内容を流用することはできません。しかし、Planex自身がRaspberry Pi 2で使用する方法を公開[注7]しています。

インストール方法は**図4**のとおりです。

まず、今回のイメージにはLinuxカーネルのヘッダファイルが含まれていません。よって、別途取得する必要があります。もちろんカーネルをアップデートするたびに必要であり、手間がかかります。

GW-450S用のカーネルモジュールのソースはいろいろと配布されていますが、紹介したGitHubリポジトリのものを使用するのが最も簡単でしょう。ただ、これはPC用の設定になっているので一部Makefileを書き換えています。あとはDKMSで自動的に再コンパイルするようにしてあります。

数日間放置してフリーズなどが起きなかったか確認しましたが、安定して動作していたようです。

プリンタ

筆者が知る限りでは、ARM用のプロプライエタリ

注4) http://gihyo.jp/admin/serial/01/ubuntu-recipe/0390

注5) http://www.planex.co.jp/products/gw-450s/
注6) http://www.planex.co.jp/products/gw-450d/
注7) http://www.planex.co.jp/articles/RaspberryPi_GW-450D/

図4 無線LANアダプタをインストールする

```
$ sudo apt-get install git-core dkms
$ mkdir temp
$ cd temp/
$ wget https://www.niksula.hut.fi/~mhiienka/Rpi/linux-headers-rpi/linux-headers-$(uname -r)_$(uname -r)-2_armhf.deb
$ sudo dpkg -i (ダウンロードしたdebファイル)
$ sudo apt-get -f install
$ git clone https://github.com/gnab/rtl8812au.git
$ sed -i 's/CONFIG_PLATFORM_I386_PC = y/CONFIG_PLATFORM_I386_PC = n/' rtl8812au/Makefile
$ sed -i 's/CONFIG_PLATFORM_ARM_RPI = n/CONFIG_PLATFORM_ARM_RPI = y/' rtl8812au/Makefile
$ sudo cp -r rtl8812au/ /usr/src/8812au-4.2.2
$ sudo dkms add -m 8812au -v 4.2.2
$ sudo dkms build -m 8812au -v 4.2.2
$ sudo dkms install -m 8812au -v 4.2.2
```

なバイナリドライバを配布しているプリンタメーカーは存在しないため、Ubuntuのリポジトリにドライバが存在するプリンタ、あるいはPPDを配布しているプリンタから選択する必要があります。レーザープリンタの場合はPostScriptまたはその互換プリンタにしておくのが無難でしょう。最近はPCLに対応したプリンタも増えているので、そちらでもいいです。インクジェットプリンタの場合はHPあるいはEPSONがいいでしょう。Canonのプリンタは、ドライバの都合上Ubuntuで使う場合はレーザープリンタでもインクジェットプリンタでも避けるのが無難です。

EPSONのプリンタを使用したい場合は、事前に“printer-driver-escpr”パッケージをインストールしておきます。残念ながら15.10でインストールできる1.4.5だと非対応ですが、16.04 LTSの1.5.2だと昨年発売された最新モデルにも対応しています。また、オープンソース版のドライバは基本的な印刷機能しかないのは注意点ですが、Raspberry Pi 2でそれほど凝った印刷をするとは思えないので、それほど強い制限ではないでしょう。

筆者が所有するEPSON EP-805Aではまったく問題なく印刷できました。そればかりか、律儀にインクが少なくなっていることを告知してくれました(**図4**)。

インストールイメージの作成

冒頭にも書いたとおり、イメージを作成するためのスクリプトは公開されています。どうしてもBitTorrentからイメージをダウンロードできない場合、最新のパッケージにしたい場合、あらかじめ転送するmicroSDカードのサイズを4GBから8GBまたは16GBに変更したい場合などに便利です。

もちろんイメージの作成には時間がかかります。それなりのマシンパワーが必要ですので、Raspberry Pi 2でセルフビルドは現実的ではありません。筆者のあまり速くないCPUとあまり速くない回線で1時間ほどかかりました。

母艦となるPCにgitをインストールし、次のコマンドを実行してビルドスクリプトを入手します。

```
$ git clone https://git.launchpad.net/ubuntu-pi-flavour-maker
```

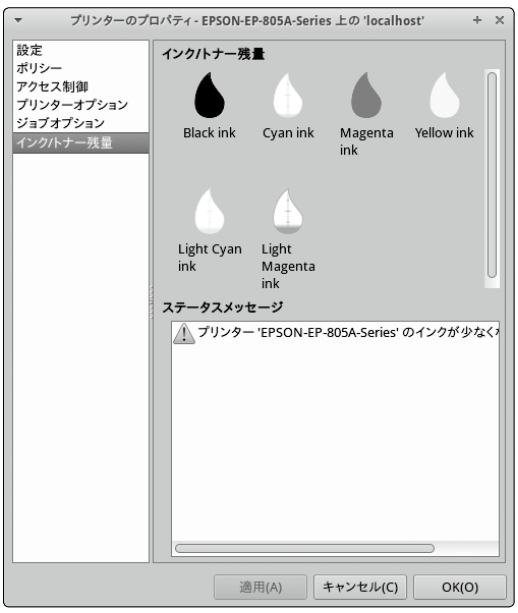
URLをよくよく見ると少々驚きます。Launchpadはいつの間にかBazaarだけではなくgitもサポートしていました。gitが使えないことでLaunchpadを敬遠している人にはうれしいニュースです。

ビルド方法はBUILD.mdに書かれているのでさほど混乱することもないと思いますが、まずはbuild-settings-xubuntu.shをbuild-settings.shに変更します。あと変更すべきなのは“FS_SIZE”くらいのものでしょう。8GBのmicroSDカード用イメージにしたい場合は“8”にします。

あとはbuild-image.shをroot権限で実行するだけです。やはりパッケージはports.ubuntu.comから取得しているため、ミラーサーバに書き換えたほうがダウンロードは速くなります。

作成したイメージは、デフォルトでは“$HOME/PiFlavourMaker/(コードネーム)/”に作成されます。SD

図4 エプソン製のドライバなので、プリンタのステータスも表示できる



Linux カーネル観光ガイド

第49回

Linux 4.1から4.4までのeBPFに関する開発

Text：青田 直大　AOTA Naohiro

1月にリリースされたLinux 4.4に引続き、Linux 4.5の開発が進んでいます。2月20日にLinux 4.5-rc5がリリースされているので、3月末ごろにはLinux 4.5がリリースされるでしょうか。今回はLinux 4.1から4.4までのeBPFに関する開発についてまとめて見ていきます。

eBPFとは

本連載でも何度か取り上げていますが、eBPFとはBPF(Berkeley Packet Filter)というもともとパケットのフィルタリングに使われる機能を拡張したものです.BPFでは、(1)パケットから任意のアドレスのデータを読み込み、(2)そのデータを算術計算やビット操作などを行って、(3)フィルタするかどうかを判断し返すという3つの機能が実行可能となっています。拡張されたeBPFでは、これに加えて(あらかじめ指定された一部の)カーネル関数を呼び出す機能、マップ(いわゆる「辞書」や「連想配列」のようなデータ構造)にデータの読み書きを行う機能を追加したものです。これらの機能を使うことで、たとえばカーネルのprandom_u32()関数を使ってある条件の通信のパケットをランダムに落とすというような処理や、あるsocketを通るパケットの統計情報を記録するといった処理もeBPFで書くことができるようになります。

またBPFでは、もともとパケットのフィルタリングを対象としていましたが、eBPFではパケットフィルタリングのほかにもkprobeのtraceの選択や、通信パケットのクラス分け、iptablesのマッチにも使うことができるようになっています。今回はperfコマンドを使ったkprobe用のeBPFの読み込みと、tcコマンドを使ったフローの通信パケットのクラス分け用のeBPFについて見ていきます。

kprobeと記録とフィルタ

kprobeはカーネルコードの任意の場所に、デバッグ用のコード(probe)を動的に埋め込みデバッグやパフォーマンス用の情報を集めるためのシステムです。probeを埋め込むにはperfコマンドを使うのが便利です(**図1**)。“`perf probe -L`”で対象関数のコードを見ながら、埋め込める行を確認します。ここではread()システムコールに相当するsys_read()関数の0行目にファイルデスクプタ番号も記録するように“`perf probe -a 'sys_read:0 fd'`”でprobeを追加します。perf recordコマンドを使い、追加されたprobe

をイベントとして指定するとデフォルトで“perf.data”というファイルにprobe dataが記録されます。記録はrecordのあとに指定したコマンドが実行されている間、行われます。記録された情報は“perf script”コマンドで記録されたイベントのリストを見ることができます。どのプロセスがどんなファイルデスクリプタ番号のファイルを読んでいるかがわかります。

“perf record”コマンドでは、フィルタで指定された条件に一致するものだけを記録するように設定できます。フィルタには“fd == 0”のような単純な数値比較もできますし、**図2**にあるように“~”を使ってglobパターンマッチを行うこともできます。ここではchromeらしきプロセスのread()のみを記録しています。

kprobeとeBPF

さて、より複雑な処理をしたい場合にはどうしたらよいでしょうか。たとえば各プロセスが連続して同じファイルデスクリプタ番号からreadした場合は記録したくないというときは、上記のフィルタでは書くことができません。もちろん、すべてを記録しておいて後で処理することもできますが、その場合多くのデータを一時的に蓄積することになります。

こうした複雑な処理をeBPFであれば書くことができます。BPFは以前であればバイトコードで書いていましたが、eBPFではLLVMを使って(一部機能が制限された)CでeBPFバイトコー

▼図1　readシステムコールへのprobe埋め込み

```
$ sudo perf probe -L sys_read
<SyS_read@/usr/src/linux-4.4.2-gentoo/fs/read_write.c:0>
      0  SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
         {
                 struct fd f = fdget_pos(fd);
      3          ssize_t ret = -EBADF;

      5          if (f.file) {
      6                  loff_t pos = file_pos_read(f.file);
      7                  ret = vfs_read(f.file, buf, count, &pos);
      8                  if (ret >= 0)
                                 file_pos_write(f.file, pos);
                         fdput_pos(f);
                 }

$ sudo perf probe -a 'sys_read:0 fd'
Added new event:
  probe:sys_read       (on sys_read with fd)

You can now use it in all perf tools, such as:

        perf record -e probe:sys_read -aR sleep 1

$ sudo perf record -e probe:sys_read -aR sleep 1
perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.457 MB perf.data (914 samples) ]
$ sudo perf script
            perf 20336 [000]  9494.636661: probe:sys_read: (ffffffff811a82a0) fd=9
           sleep 20337 [001]  9494.637040: probe:sys_read: (ffffffff811a82a0) fd=4
          chrome  6477 [002]  9494.643328: probe:sys_read: (ffffffff811a82a0) fd=18
          chrome  6477 [002]  9494.648551: probe:sys_read: (ffffffff811a82a0) fd=18
          chrome  6477 [002]  9494.649550: probe:sys_read: (ffffffff811a82a0) fd=18
 Chrome_IOThread  6509 [003]  9494.649862: probe:sys_read: (ffffffff811a82a0) fd=51
          chrome  6563 [003]  9494.649945: probe:sys_read: (ffffffff811a82a0) fd=10
 Chrome_IOThread  6509 [003]  9494.650119: probe:sys_read: (ffffffff811a82a0) fd=51
          chrome  6563 [003]  9494.650163: probe:sys_read: (ffffffff811a82a0) fd=10
 Chrome_IOThread  6509 [001]  9494.650323: probe:sys_read: (ffffffff811a82a0) fd=51

 ⋮
```

▼図2　perf recordへのfilterの設定

```
$ sudo perf record -e probe:sys_read --filter 'comm ~ *hrome*' -aR sleep 1
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.439 MB perf.data (717 samples) ]
$ sudo perf script
          chrome  6477 [002] 10451.034512: probe:sys_read: (ffffffff811a82a0) fd=18
          chrome  6477 [002] 10451.034533: probe:sys_read: (ffffffff811a82a0) fd=18
          chrome  6477 [002] 10451.034641: probe:sys_read: (ffffffff811a82a0) fd=18
          chrome  6477 [002] 10451.043879: probe:sys_read: (ffffffff811a82a0) fd=18
 Chrome_IOThread  6509 [003] 10451.045177: probe:sys_read: (ffffffff811a82a0) fd=51
          chrome  6563 [003] 10451.045261: probe:sys_read: (ffffffff811a82a0) fd=10
 Chrome_IOThread  6509 [003] 10451.045492: probe:sys_read: (ffffffff811a82a0) fd=51
          chrome  6563 [003] 10451.045538: probe:sys_read: (ffffffff811a82a0) fd=10
 Chrome_IOThread  6509 [000] 10451.045712: probe:sys_read: (ffffffff811a82a0) fd=51
          chrome  6563 [003] 10451.045792: probe:sys_read: (ffffffff811a82a0) fd=10
 :
```

▼リスト1　proc-read-first.c

```
 1 #include <linux/types.h>
 2 #include <asm/ptrace.h>
 3
 4 #define BPF_ANY 0
 5 #define BPF_MAP_TYPE_HASH 1
 6 #define BPF_FUNC_map_lookup_elem 1
 7 #define BPF_FUNC_map_update_elem 2
 8 #define BPF_FUNC_get_current_pid_tgid 14
 9
10 static void *(*bpf_map_lookup_elem)(void *map, void *key) =
11     (void *) BPF_FUNC_map_lookup_elem;
12 static void *(*bpf_map_update_elem)(void *map, void *key, void *value, int flags) =
13     (void *) BPF_FUNC_map_update_elem;
14 static __u64 (*bpf_get_current_pid_tgid)(void) =
15     (void *) BPF_FUNC_get_current_pid_tgid;
16
17 struct bpf_map_def {
18     unsigned int type;
19     unsigned int key_size;
20     unsigned int value_size;
21     unsigned int max_entries;
22 };
23
24 #define SEC(NAME) __attribute__((section(NAME), used))
25 struct bpf_map_def SEC("maps") pid_table = {
26     .type = BPF_MAP_TYPE_HASH,
27     .key_size = sizeof(int),
28     .value_size = sizeof(int),
29     .max_entries = 256,
30 };
31
32 SEC("func=sys_read:0 fd")
33 int bpf_func__sys_read(void *ctx)
34 {
35     int pid = bpf_get_current_pid_tgid() & 0xFFFF;
36     int *last = bpf_map_lookup_elem(&pid_table, &pid);
37         struct pt_regs *regs = ctx;
38         int fd = regs->di;
39
40     bpf_map_update_elem(&pid_table, &pid, &fd, BPF_ANY);
41     if (!last || *last != fd) {
42                 return 1;
43     }
44         return 0;
45 }
46 char _license[] SEC("license") = "GPL";
47 int _version SEC("version") = LINUX_VERSION_CODE;
```

ドを出力できます。さっそく、Cで上記のようなフィルタを書いてみましょう。

proc-read-first.cを見ていきましょう(**リスト1**)。4行目から8行目ではプログラム中で使うマップ用の定数と、eBPFから呼ぶことができる関数の番号をdefineしています。10行目から15行目はdefineした番号を使って、eBPFの関数を宣言しています。17行目から30行目ではフィルタで使うマップであるpid_tableの設定を記述しています。perfはこの値を"maps"セクションから読んでマップを作成します。ここではハッシュテーブルを作っていますが、BPF_MAP_TYPE_ARRAYを使った配列型マップにすることもできます。

その次がフィルタの本体となるbpf_func_map_lookup_elem()関数です。イベントの対象は、"perf probe -a"のときと同様の形式でセクション名として設定します。この関数は、まず35行目でbpf_get_current_pid_tgid()を使ってプロセスIDを取得します。これは下位32bitにプロセスIDが、上位32bitにtgidが返ってくる関数です。pid_tableからプロセスIDをキーにして、前回のファイルデスクリプタを読みだします。ここではポインタが返ってきていて、指定したキーに対応するデータがマップにない場合はNULLが返ってきます。現在のファイルデスクリプタ番号はレジスタから取得します。kprobeの場合、ctxに"struct pt_regs"型のポインタが入っているので、あとはどのレジスタにfdが入っているのかが問題となります。これはちょっとトリッキーなのですが、先ほど"perf probe -a"して追加されたイベントを/sys/kernel/debug/tracing/kprobe_eventsから調べてみるとよいと思います。最後にbpf_map_update_elem()でマップを更新し、イベントを記録する場合は1を、記録しない場合は0を返します。

では、このeBPFコードを読み込ませてみましょう(**図3**)。eBPFのロードには"perf record -e"を使います。これは先ほどイベント名を指定していた引数ですが、.cファイルが指定されるとLLVMがCコードをeBPFバイトコードへとコンパイルし、その結果が自動的にロードされ、イベントの記録が開始されます。記録されたデータを表示してみると、たしかに上記条件のとおりに記録されていることがうかがえます。

パケットのクラス分け

前述したようにeBPFはパケットのクラス分けにも使うことができます。しかし、そもそも

▼図3　eBPFによるイベントフィルタリング

```
$ cat /sys/kernel/debug/tracing/kprobe_events
p:probe/sys_read _text+1737376 fd=%di:s64      # fdは"di"レジスタに入っている
$ sudo perf record -e ~/proc-read-first.c -aR sleep 1
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.414 MB perf.data (15 samples) ]
$ sudo perf script
            perf  9548 [000] 16223.211229: perf_bpf_probe:func: (ffffffff811a82a0) fd=11
           sleep  9805 [002] 16223.212054: perf_bpf_probe:func: (ffffffff811a82a0) fd=3
          Nozbe  8271 [000] 16223.252993: perf_bpf_probe:func: (ffffffff811a82a0) fd=14
 Chrome_ChildIOT  7340 [001] 16223.343621: perf_bpf_probe:func: (ffffffff811a82a0) fd=23
 Chrome_ChildIOT  8895 [002] 16223.343856: perf_bpf_probe:func: (ffffffff811a82a0) fd=23
 Chrome_ChildIOT  9064 [000] 16223.344060: perf_bpf_probe:func: (ffffffff811a82a0) fd=23
 Chrome_ChildIOT  8521 [000] 16223.344090: perf_bpf_probe:func: (ffffffff811a82a0) fd=23
 Chrome_ChildIOT  8470 [001] 16223.344471: perf_bpf_probe:func: (ffffffff811a82a0) fd=23
 Chrome_ChildIOT   625 [001] 16223.702691: perf_bpf_probe:func: (ffffffff811a82a0) fd=23
          Nozbe  8349 [000] 16223.739254: perf_bpf_probe:func: (ffffffff811a82a0) fd=15
 Chrome_ChildIOT  6789 [003] 16223.905824: perf_bpf_probe:func: (ffffffff811a82a0) fd=23
 Chrome_IOThread  6509 [002] 16223.905890: perf_bpf_probe:func: (ffffffff811a82a0) fd=42
 Chrome_IOThread  6509 [002] 16223.939824: perf_bpf_probe:func: (ffffffff811a82a0) fd=51
 Chrome_IOThread  6509 [002] 16224.043263: perf_bpf_probe:func: (ffffffff811a82a0) fd=205
 Chrome_IOThread  6509 [002] 16224.064700: perf_bpf_probe:func: (ffffffff811a82a0) fd=51
```

パケットのクラスとはなんでしょうか。

アプリケーションの通信は、多くの場合1つのネットワークデバイスを通じて行われます。すなわち、もし1つのアプリケーションが多くの帯域を使ってしまえば、ほかのアプリケーションの通信も影響を受けることになります。この問題を解決する手段の1つがTraffic Controlです。Traffic Controlは、アプリケーションのパケットをどのようにネットワークデバイスに送信するかを制御します。たとえばデフォルトではシンプルにpfifoという単純にFIFOで送信するqdisc(queuing discipline)が使われますし、systemd環境においてはデフォルトがfq_codelになるようです。そうした様々なqdiscの中には"クラス"ごとに流量制御を行うものもあります。たとえば、DRR(Deficit Round Robin Scheduler)では分類されたクラスごとにround robinにパケットを送信します。

すると次はパケットをクラス分けするシステムが必要となりますが、これはfilterと呼ばれています。filterにはcgroupを使うものや、経路情報を使ってたとえばLAN内への通信と外への通信を別クラスにするものがあります。そんなfilterの1つとしてeBPFを使うことができるようになっています。

eBPFによるパケットのクラス分け

では、eBPFによるパケットのクラス分けのコードを見ていきましょう。**リスト2**は"**man tc-bpf**"に掲載されている、クラス分けのeBPFコードの抜粋です。kprobeのSECと同様に、**__section("cls")**でセクションが切られている関数が、クラス分けのメイン関数になります。kprobeでは引数としてレジスタ情報が来ていましたが、こちらは"**struct __sk_buff ***"とあるようにソケットバッファが引数となっています。

ここではソケットバッファから事前に付けられたマーク(**skb->mark**)を読み取り、単純にそれをそのままクラス分けに使っています。戻り値の"**TC_H_MAKE(TC_H_ROOT, mark)**"がこのパケットが所属するクラスを示しています。ここで0を返すとこのfilterに適合しなかったことを意味し、別のfilterが使われます。また、**-1**を返すとデフォルトのクラスを使うという意味になります。

また、このfilterコードではクラス分けだけでなく、cls_update_stats()関数内で累計パケット数と累計パケットサイズをクラスごとに集計

▼リスト2　クラス分けのeBPFコード(抜粋)

```
static inline void cls_update_stats(const struct __sk_buff *skb,
                                    uint32_t mark)
{
        struct tuple *tu;

        tu = bpf_map_lookup_elem(&map_stats, &mark);
        if (likely(tu)) {
                __sync_fetch_and_add(&tu->packets, 1);
                __sync_fetch_and_add(&tu->bytes, skb->len);
        }
}

__section("cls") int cls_main(struct __sk_buff *skb)
{
        uint32_t mark = skb->mark;

        if (unlikely(mark >= BPF_MAX_MARK))
                return 0;

        cls_update_stats(skb, mark);

        return TC_H_MAKE(TC_H_ROOT, mark);
}
```

しています。このようにクラス分け以外の操作も行うことができます。

eBPFによるパケット操作

filterではもっと変わったことをすることもできます。もう1つのmantc-bpfのサンプルコード(**リスト3**)は80番に来たパケットを8080から8087番ポートで動くサーバに振り分け、ロードバランシングするというものです。これまでと同様にセクションが設定されているlb_mainがエントリポイントとなります。ソケットバッファからプロトコルの部分を取り出し、IPv4であればlb_do_ipv4()関数を呼び出しています。lb_do_ipv4()関数はTCP通信の80番ポートへのパケットであるかどうかを確認し、set_tcp_dport()で宛先ポート番号の書き換えを行っています。set_tcp_dport()関数は、ポート書き換えおよび、それに応じたチェックサムの更新を行っています。ここでもbpf_l4_csum_replaceやbpf_l4_csum_replaceといった、eBPFの外部関数を呼び出せる機能が活かされています。SD

▼リスト3　eBPFによるパケット書き換え

```
static inline void set_tcp_dport(struct __sk_buff *skb, int nh_off,
                                 __u16 old_port, __u16 new_port)
{
        bpf_l4_csum_replace(skb, nh_off + offsetof(struct tcphdr, check),
                            old_port, new_port, sizeof(new_port));
        bpf_skb_store_bytes(skb, nh_off + offsetof(struct tcphdr, dest),
                            &new_port, sizeof(new_port), 0);
}

static inline int lb_do_ipv4(struct __sk_buff *skb, int nh_off)
{
        __u16 dport, dport_new = 8080, off;
        __u8 ip_proto, ip_vl;

        ip_proto = load_byte(skb, nh_off +
                             offsetof(struct iphdr, protocol));
        if (ip_proto != IPPROTO_TCP)
                return 0;

        ip_vl = load_byte(skb, nh_off);
        if (likely(ip_vl == 0x45))
                nh_off += sizeof(struct iphdr);
        else
                nh_off += (ip_vl & 0xF) << 2;

        dport = load_half(skb, nh_off + offsetof(struct tcphdr, dest));
        if (dport != 80)
                return 0;

        off = skb->queue_mapping & 7;
        set_tcp_dport(skb, nh_off - BPF_LL_OFF, __constant_htons(80),
                      __cpu_to_be16(dport_new + off));
        return -1;
}

__section("lb") int lb_main(struct __sk_buff *skb)
{
        int ret = 0, nh_off = BPF_LL_OFF + ETH_HLEN;

        if (likely(skb->protocol == __constant_htons(ETH_P_IP)))
                ret = lb_do_ipv4(skb, nh_off);

        return ret;
}
```

April 2016

NO.54

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
波田野 裕一　HATANO Hirokazu　tcsh@tcsh.csh.sh
榎 真治　ENOKI Shinji　enoki-s@imail.plala.or.jp
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

これからのエンジニアのあり方、コミュニティのあり方

JPNIC（日本ネットワークインフォメーションセンター）主催のInternet Weekが、2015年11月に開催されました。今回もjusは後援団体として参加し、プログラムの企画や告知などの協力を行いました。4日間で約40本のプログラムが実施され、約2,600人の参加者を集めました。実施したプログラムの中から、jus幹事が企画にかかわったセッションについて報告します。

Internet Week 2015

■Internet Week 2015
【日時】2015年11月17日（火）～20日（金）
【会場】富士ソフト アキバプラザ

■クラウドネイティブ時代のインフラエンジニア（11月17日16:15～18:45、6F セミナールーム6）

本セッションでは、クラウドサービスを牽引しているAmazon Web Services（AWS）を具体的な参考事例として、「クラウドサービス」の変化がもたらす次の2点

- 企業や組織のITインフラの設計、構築、および運用への影響
- 企業やインフラエンジニアが直面するであろう変化

について、クラウド提供者の視点およびクラウド利用者の視点を交えて紹介が行われました。

最初に、波田野裕一さん（運用設計ラボ合同会社）から、クラウド利用の動向（利用者側の変化）、クラウド事業者の動向（提供者側の変化）の紹介がありました。利用者側の変化としては、従来は低コストを理由に利用されていたクラウドが、最近は可用性や安定性、迅速性について積極的な評価を受けており、ITインフラエンジニアの主要な価値とされていた「物理的なIT資産に関連する業務」にコモディティ化の波が押し寄せていることが紹介されました。一方で提供者側にも大きな変化があり、従来はサーバやネットワークなどのいわゆるIaaSがクラウドコンピューティングそのものでしたが、最近は各種マネージドサービスに主戦場が変化しており、物理的なIT資産だけでなく「サーバやOSに関連する業務」についてもコモディティ化が進むだろう、との展望が紹介されました。

次に、荒木靖宏さん（アマゾン ウェブ サービス ジャパン㈱）から、クラウド提供者の視点からの変化について、「クラウドネイティブ」というキーワードを軸とした話がありました。クラウドネイティブとは、クラウドで提供されるサービス利用を前提に構築するシステムおよびアプリケーションのことを指し、それらの多くは将来的にはサーバレスアーキテクチャに移行するとのことでした。これにより開発コストや運用コストが最小化し、スケーラビリティやキャパシティ、セキュリティについて心配することなくビジネスにフォーカスできるようになっていくと言います。そのときには今のネットワークエンジニアのコア・コンピタンス（独自の強み、得意分野）の多くは消失し、「ユーザがほしいのはサービスであり、テクノロジそのものではない」時代になっているだろう、という言葉が印象的でした。

続いて、小早川知昭さん（ソニー㈱）から、クラウド利用者の視点からの変化についての話がありました。同社の基盤システムを刷新するにあたり、自社で保有運用していたプライベートクラウドを閉鎖し、AWSへの移行を実施した事例が紹介されました。そして、クラウド利用者として実感したこととして、がんばってプライベートクラウドを実装しても利用者からは「当たり前程度」にしか評価されず、インフラレイヤで付加価値を付けることは難しくなっていることを挙げました。今後は一般企業でのインフラエンジニアの活躍の場は縮小し、ソフトウェアの素養が求められていくだろうとのことでした。

最後に波田野さんから、クラウドネイティブ時代に求められるエンジニアスキルの紹介があり、インフラエンジニアの専門性の変化や求められるスキルについて紹介がありました。

立場の異なる登壇者3人が三者三様にインフラエンジニアを取り巻く環境の変化と、現在置かれている状況への危機感を語り、共通した認識のもとで現状からの脱却や将来への展望を紹介する貴重なセッションだったと思います。満員の会場では聴講者が熱心にメモを取る姿が印象的でした。

■ITコミュニティの運営を考える
（11月19日19:00～20:30、6F セミナールーム5）

jusでは各地で「ITコミュニティの運営を考える」をテーマとするセッションを行っていますが、それをInternet WeekでもBoFという形で実施しました。はじめに、参加者が6人と少なかったので、全員に自己紹介やコミュニティとのかかわりを話していただきました。その後、こちらで用意したいくつかのお題と、そこから派生した話題について自由に討論しました。用意したお題ごとに、印象に残ったコメントをお伝えします。

――みなさんが参加されているコミュニティの特色は何ですか？

参加者それぞれが日ごろかかわっているコミュニティを例に特色を挙げていただきましたが、それに付随して出てきたコメントとしては、「目的がはっきりしているコミュニティのほうが参加しやすい」、「新しいコミュニティは新しいことができそうなので新しい人が入ってきやすい」、「長年活動しているコミュニティは人が滞留して、まったりした活動になりがち」、「新しい人に参加してもらうには自分も何か手伝えそうと思わせる雰囲気作りが必要」といったものがありました。

――良い運営をしていると思うコミュニティは？

この質問に対しては、特定コミュニティを取り上げた意見よりも、時代に応じて話題を転換できているコミュニティや、人材の新陳代謝があるコミュニティが良いという、コミュニティの体質に関する意見が多く聞かれました。また、ここから派生した話題として、業務でコミュニティ運営にかかわることについての議論がありました。日本には寄付する文化がないことや、利用しているから貢献しなければならないという考えが根づいていないため、業務でコミュニティ運営にかかわることが認められにくい傾向があるようです。

――ほかのコミュニティと手を取り合うことについてどう思いますか？

これについては、ほかのコミュニティとの交流を望んでいる団体は多いのですが、きっかけがなかったり、業種やレイヤをまたいだ連携の機会が少なかったりと、なかなか実際に交流するところまではいかないようです。解決策として、「共通点を見つけてそこから交流を始めると良いのではないか」とか、「ITコミュニティを総合的に紹介するサイトがほしい」といった意見が出ました。

同時間帯にコミュニティ主催によるBoFが多く重なったため、こちらに参加してくださる方が少なかったのは残念ですが、参加者のみなさんが積極的に意見を出してくださったので、有意義な議論ができたと思います。ありがとうございました。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

第52回

エフスタ!! TOKYOでデザインと開発を考える

●Hack For Japan スタッフ
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha

本連載でたびたび登場している、ITエンジニアのためのスキルアップ応援コミュニティ「エフスタ!!」。今回は東京で開催された「エフスタ!! TOKYO」の様子をレポートします。福島を拠点とするエンジニア達がどのような思いで活動しているのか、その一端が見えれば幸いです。

「エフスタ!!」とは

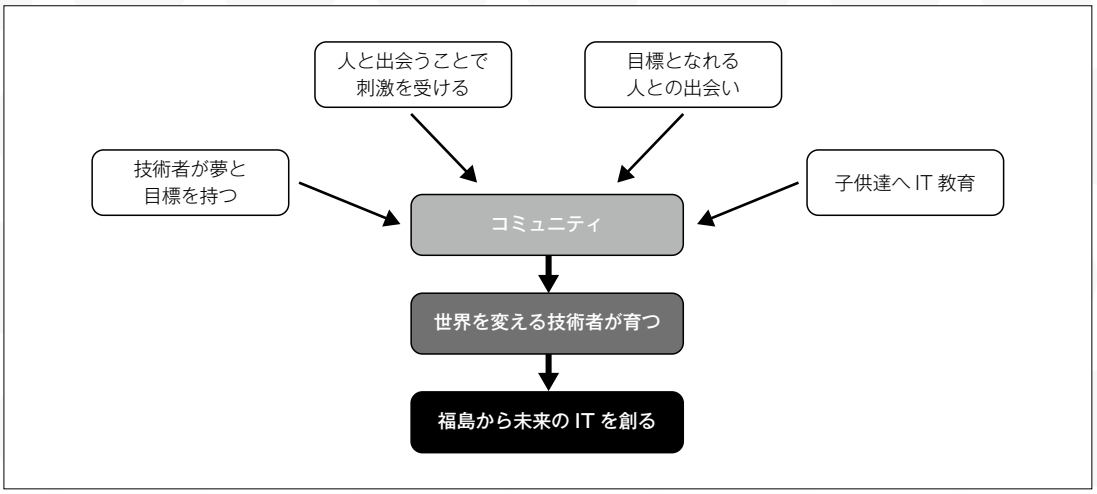
初めて「エフスタ!!」というコミュニティを目にする読者の方のために、簡単にこのコミュニティについて紹介しようと思います。「エフスタ!!」では「IT業界で働く人たちが楽しんで仕事ができるよう身の回りのIT業界を変えたい」「夢と希望をもった技術者を育てたい」「そのために教育に力を注ぎたい」という願いが根底にあります。であれば、まず「エフスタ!!」の地元である福島のITが変われば、面白くなれば、夢と希望を持った技術者が増え、やがて世界を変える技術者が福島から誕生するという発想です。そのような理念を実現するためのきっかけ作りの場として「エフスタ!!」というコミュニティは運営されています（**図1**）。

また、「エフスタ!!」の語源は「福島のスタイルを変える」から「エフスタイル」に略され、語呂の良さから今の「エフスタ!!」というコミュニティ名に落ち着いたそうです。

デザインとエンジニアリング

さて、今回紹介する「エフスタ!! TOKYO」のテーマは「デザインの流儀 ～ITエンジニア×デザイナー」でした。ITエンジニアとデザイナーが一緒になってシステム開発を行うシーンが増えてきている最近の流れの中で、お互いの仕事の間にある隔たりや、それぞれの仕事上のこだわるポイントが異なっている点で相互に理解し合えていない状況があります。こういった状況でなかなかうまく仕事が進められない現場の課題を解決するために、デザイナーの方やデザイナーとエンジニアの間で活躍されている方達に登壇いただき、双方がうまく仕事を進めてい

◆**図1 「エフスタ!!」活動理念**



くためのポイントや、デザイナーはどういった発想で物作りに当たっているかという観点からのお話をしていただきました。

デザインをする際に考えていることとその視点

最初にフリーランスのデザイナーとして活躍されているTIMING DESIGN代表の北村崇さんより、デザイナー目線から見た、物作りのうえでのこだわりのポイントを紹介いただきました(**写真1**)。北村さんはデザイナーがアーティストだと勘違いされ、ディレクターやエンジニア、営業といった職種の人たちから、見た目優先で作っていると誤解されやすい現状に警鐘を鳴らします。デザインという作業の本質やデザイナーが何を考え、何をゴールとしてデザインをしているのか、またはするべきなのか。デザインの基礎のお話を踏まえたうえで、構築側が知っているとやり取りがスムーズになる「デザイナーのこだわりポイント」などのお話がありました。

まず「デザインは本能に忠実」というのが前提にあり、視線の動きであったり、心理学や社会学も含めた形でデザインについて包括的に人間の行動に沿った形を意識して考えられているという点があります。また、デザインは目的を持って提供していくという観点から、アクセシビリティのしっかりしているサイトはタブ移動にも意味が感じ取れるが、人間の行動を考慮せずに目的を持たないようなデザインになってしまっているサイトは使いづらいという指摘は会場からも納得の声も多く挙がりました。

◆写真1　北村さんによるデザイナーの考え方の紹介

デザインの工夫でユーザの負担を大きく下げるという話では、「いつ」「どこで」「誰が」「なにを」「どうする」「どのようにする」というのを考えると、それぞれの状況でユーザが置かれている状況が変わることが理解できるとのこと。そのような状況を議論の前提に置くことで、相手のやっていることを理解する、自分のやっていることを理解してもらう、といったコミュニケーションをはかる能力があがります。これがデザイナーとエンジニアがうまく仕事を進めていくうえでも非常に大切なものになるという、当たり前のようで、なかなかできていない部分にフォーカスしたお話でした。

異なるアプローチで同じゴールへ向かうチーム

続いては、インフラジスティックス・ジャパン代表取締役の東賢さんによる「デザイナーとデベロッパー：異なるアプローチで同じゴールへ向かうチーム」というテーマのお話で、デザイナーとエンジニアがうまく仕事を進めるうえで前提となる考え方を確かめる問いとして「デザイナーとエンジニアは目的が本当に一緒になっているか？」というものを挙げられました(**写真2**)。そして、実際にデザイナーとエンジニアが一緒になって作っているプロダクトについて、目的が一致しているかどうかを確かめる手段として、エレベーターピッチをエンジニアもデザイナーも同じように伝えられるかどうか？というものを紹介いただきました。

エレベーターピッチとは、自分たちが開発してい

◆写真2　東さんによるデザイナーとエンジニアの双方に必要な姿勢のお話

るプロダクトが、どういった顧客向けで、なんという製品名で、どのようなカテゴリに属していて、そのプロダクトの重要な利点や対価に見合う説得力のある理由、競合製品と差別化する決定的な特徴は何かなどといった説明を、エレベーターに乗っている短い間で同乗する出資候補者に対して行うプレゼンテーションのことを言います。このエレベーターピッチを通して、エンジニアとデザイナーが目的を1つに取り組めているかどうかが測れるというお話はたいへんうなずけるものでした。

また、デザイナーとエンジニアの間のギャップを埋めるアーキテクチャとして「ここから先は変えていい、ここから先はダメという線引」「デザインを実装しやすくするための共通言語を利用する」「デザインは想定した仕様を維持できる、さらに良くできる」「UIのアーキテクチャについてはいくつかの検討要素がある」といったキーワードを挙げました。これらを軸にして、デザインをしていくうえでの制約の整理を行うのがエンジニアの仕事、というふうにお互いの職務領域を明確にしてからコミュニケーションすることの重要性を唱えられました。

デザイン・開発の両立から見えること

講演の最後は我々、Hack for Japanのメンバーでもある株式会社dott 共同代表取締役の清水俊之介さんによる、デザインという仕事と開発という仕事の両立を通じて見えてきた、デザイナーとエンジニアが一緒になって働くコツをお話いただきました(**写真3**)。

◆**写真3　清水さんによるデザインと開発の間のお話**

清水さんは絵画修復技術士という経歴から、システム開発の道に入られた異色の経歴を持つ方で、背景として芸術があったことからシステム開発の道に入ってからもデザインから実際のコーディングまで、一人で作業することが多かったそうです。このことから、双方の立場に立って相手の求めているものを歩み寄って知ることの大切さを伝えてもらいました。

ライトニングトークで知る福島

さて、「エフスタ!!」のイベントでは参加者によるライトニングトークが恒例となっています。「エフスタ!! TOKYO」では東京に住む人たちに向けて、福島の現状を伝えるお話も過去数回にわたりライトニングトークの場で聞くことができました。今回は直接参加者の方が福島の現状を伝えるといった内容から趣を変えて、今、福島で行われている一つの試みが紹介されました。

福島を伝えるドキュメンタリー

「エフスタ!!」のスタッフでもある山中英治さんのライトニングトークで紹介されたのは「1/10 Fukushimaをきいてみる[注1]」という福島の今を切り取って伝えるドキュメンタリー映画でした(**写真4**)。震災から数年が経ち、福島県外に住んでいる人たちには

注1　http://fukushima-ask.info/

◆**写真4　「1/10 Fukushimaをきいてみる」を紹介する山中さん**

現地の実際の情報が正しく伝わって来ることがありません。読者のみなさんも福島原発での除染の話や子育ての話、福島での食事の話や帰宅困難地域に住んでいた人たちのニュースを耳にすることは少なくなったのではないでしょうか？　また、聞いたとしてもそれが本当の話なのかどうか、福島に住む知人がいなければ確かめることもできません。このドキュメンタリーは福島の現地に住む人の声をひたすら集めて1年単位で公開していくスタイルで、監督の古波津陽さん、出演は福島県出身の女優である佐藤みゆきさん、撮影が柏崎佑介さんといった3人で2013年から始まった企画として紹介されました。

ドキュメンタリーの中で取り上げられている話を少しだけ山中さんより紹介いただきましたが、除染袋を引き取っている住職の話や成人式を迎える新成人に関する話など、今の福島を伝えるドキュメンタリーとして、大変興味深いものがありました。福島県外での上映や海外での上映なども予定されています。ご興味をお持ちの読者のみなさんもぜひホームページにアクセスして、上映情報をご覧ください。

「エフスタ!!」とHack For Japan

さて、「エフスタ!! TOKYO」も今回で数えて7回目を迎えました。2012年12月8日に都内にて第1回が開催された際には、Hack For Japanからは及川卓也さんによる「見る前に跳べ～ギークの工夫で社会を変えよう～2012年冬」と題した「Developers Summit 2012」で発表された内容のアップデート版での発表がありました。これを皮切りに、「エフスタ!!」の勉強会では毎回IT業界のプロフェッショナルを招いて講演をしてもらうことで「エフスタ!!」に集まるエンジニア達のスキル、マインドの底上げという理念の一部実現を狙っています。

「エフスタ!!」に参加されたことのない方から見ると、ITプロフェッショナルの講演や福島の話となると非常に固い勉強会という印象を持たれてしまうかもしれませんが、これは実際に参加していただくとわかるとおり、「エフスタ!!」の勉強会はどこかアットホームな雰囲気となっています。毎回、福島名物のおやつが配られるおやつタイム（「エフスタ!! TOKYO」では東京のおやつも合わせて配られます）もあり、リラックスしながら参加者同士で交流し、新しい人達とつながりが生まれています。また、メリハリを付けたトークセッションではゲストの真剣な技術の話と参加者を交えて面白おかしいトークを織り交ぜることで、参加者のみなさんは真剣に聴講することと、会を楽しんでリラックスして参加することの両方ができているのです。そして、参加者のスキル、マインドを高め、リラックスした後に福島の現状を語る場も合わせて用意することで参加者に福島にも興味を持ってもらうのが定番のスタイルとなっています。

7回目の東京開催となった今回はもう1つ大きなニュースがありました。「エフスタ!! TOKYO PJ」立ち上げメンバーでもある代表の大久保仁さんは当日に業務が重なってしまい参加が叶いませんでしたが、同じく「エフスタ!!」初期から参加し「エフスタ!! TOKYO」の責任者でもある影山哲也さんより、運営スタッフの中から誕生したカップルがご結婚されたというニュースが発表がされました（**写真5**）。

明るく福島の未来のために活動する「エフスタ!!」も長い時間をかけて、コミュニティとしても大きく育っているうれしいニュースでした。こうして福島を飛び出して大きく活躍する人たちと共にこれから長い時間をかけて復興していく福島を、みなさんもどうぞ応援してください。SD

◆写真5　エフスタ!!スタッフ同士の結婚の立会い人となった大久保さん

温故知新
IT むかしばなし

第53回

マシン語
～高速なプログラムを目指して～

速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami

はじめに

プログラム言語を用いてソフトウェアを記述するのが主流の現在では、コンピュータが直接実行するマシン語は、人の目に触れない存在になっています。表に見えないマシン語ですが、低機能なマイコンボードを扱う場合には、少し姿を現します。今回は、大きく表面の舞台で活躍していたころの「マシン語」についてお話ししましょう。

マシン語が脚光を浴びていた時代

1970年代後半から80年代の前半のパソコンは8bit CPUで、BASIC言語をシステムとして動かしているものがほとんどでした。そもそもCPU自体のスピードが遅いうえに、その上で動作するBASICインタプリタの実行速度はさらに非常に遅く、動作状況が目で追えるレベルでした。メモリ空間の狭さも相まって、BASICのみで実用的なプログラムを作成することは難しかったのです。とくにホビーユーザの目指していたゲームプログラミングにおいては、BASICプログラミング以外の方法が必要でした。そこで登場するのがマシン語です。高速性が必要な処理は、BASICからマシン語を呼び出して実行することが、重要なテクニックになっていたのです。

8bit CPUのマシン語

自分でマシン語プログラムの作成にチャレンジすることになると、このプログラミングのハードルはかなり高く、ハードルを越えるのに苦労したユーザが多くいたようです。

ハードルを乗り越え、わずかでもマシン語プログラムができると、その処理部分のスピードアップはすさまじいもので、プログラミングの苦労が報われる気分を味わうことができました。

当時、マイコンに搭載されていた8bit CPUは、MZ-80(シャープ)、PC-8001(NEC)などがザイログ社のZ80、ベーシックマスターLEVEL3(日立)とFM-8(富士通)がモトローラ社のMC6809を搭載していました。

マシン語のプログラミング

当初のマシン語のプログラミングは、ハンドアセンブルが基本でした。CPUの独自のマシン語(16進数)を、人にわかりやすいニーモニックに置き換えたアセンブリ言語を紙面の上に手書きし、CPU命令表を見ながら16進数に変換するという作業(これが当時のプログラミング)を行ったこともありました。その作業に慣れてくると、よく使う命令はCPU命令表を見なくてもマシン語に変換することができるようになりました。長いプログラミングになりますと、ハンドアセンブラは限界です。そこで使用したのが、1パス・アブソリュート・セルフアセンブラでした。

Z80と6809のプログラミング

Z80は、インテル社i8080から発展し、その互換性を有していました。レジスタが豊富にあり、命令の種類も多種多様に備えており、実用的なプログラムを作成しやすく設計されていました。しかし、レジスタと命令とメモリにアクセスするアドレッシングとの組み合わせの直交性が低いため、レジスタ利用に特殊性を覚えてからプログラミングを

する必要がありました。

PC-8001(**写真1上**)を使用してマシン語プログラミングにチャレンジしてみました。**リスト1**は、テキストVRAMに文字AからZを表示するもので、この程度のプログラミングは、ハンドアセンブラで簡単に組むことができます。その中で使用したDJNZ命令[注1]のような便利な命令も多くあり、慣れれば楽にプログラミングすることができました。

次に、6809でプログラミングをするためにLEVEL3を使ってみました(**写真1下**)。同じ動作をするプログラムを作成したところ、そのハンドアセンブル作業が非常に簡単でした。6809は、マシン語コードを形成するオペコード、オペランド、ポストバイトがきちんと整理されており、その法則性を理解することで命令をマシンコードに簡単に変換できます。豊富なアドレッシングモードは、その多様さに理解するのはたいへんですが、理解したあとは便利に利用することが可能になります。STA ,X+[注2]などのようにアドレッシングを効果的に利用することで、見やすくて小さなコードサイズを実現できます。

ここでのプログラミングサイズ は、Z80 が 13byte、6809 が 14byteとほぼ同じになりました。それぞれのCPUの特徴を活かすことで、効果的なプログラミングが可能になります。しかし、実際に大きなプログラミングを行うと考え方の違いに戸惑うことが多々あり、当時のユーザは、Z80派と6809派に分かれていたようです。

マシン語の実行スピード

筆者が実際にPC-8001とLEVEL3で同じ処理を行う2つのプログラムを、ループ回数を増やして実行したところ、予想どおりほぼ同じ実行時間でした。

同じ処理をBASIC言語のみで作成したものを実行してみたところ、PC-8001では約60倍、LEVEL3では約80倍の差がありました。

16bit CPUへ

時代は、8bitマイコンから16bitパソコンへ移っていきます。

16bitパソコンが登場した当初は、まだBASICは遅く、MS-DOS上で動作する優れたコンパイラも手に入りづらい状況にあり、クリティカルな処理をプログラミングする場合などのようにしばらくはアセンブラ言語が使用される時期が続きました。

マシン語プログラミングは、I/Oポートを直接制御してタイミングをとる際やコンピュータの基本を理解する場合などには重要なプログラミング技術です。最近Arduinoに搭載されているAtmel社AVRなどのように優れた設計の8bit CPUがありますが、より教育的な効果が高くマシン語プログラミングにおいて効率の良い、独自の8bit CPUがあってもよいのではないかと考えます。SD

▼写真1　PC-8001(上)とベーシックマスターLEVEL3(下)

▼リスト1　Z80と6809の機械語の比較

```
Z80------------------------------+6809-------------------------
D000  06 1A       LD   B,26      7900  C6 1A       LDB #26
D002  3E 41       LD   A,41H     7902  86 41       LDA #$41
D004  21 00 F3    LD   HL,F300H  7904  8E 04 00    LDX #$0400
D007  77       L1: LD   (HL),A   7907  A7 80    L1: STA ,X+
D008  3C          INC  A         7909  4C           INCA
D009  23          INC  HL        790A  5A           DECB
D00A  10 FB       DJNZ L1        790B  26 FA        BNE L1
D00C  36          RET            790D  39           RTS
```

注1) レジスタBの内容を1だけ減じて、レジスタBの値がゼロであれば次の命令に進み、ゼロでない場合は、相対ジャンプする。

注2) 16bitインデックスレジスタXの値が示すアドレスにレジスタAの値を格納し、その後インデックスレジスタの値を1増やす。

うまくいく チーム開発のツール戦略

第1回 メールベースでの進行管理の限界、ツール活用による変革

Author リックソフト㈱ 熊井 亮輔(くまい りょうすけ)、大塚 和彦(おおつか かずひこ)

メールによるプロジェクト内コミュニケーションの限界

メールによる情報伝達によって摩擦が生じたことはありませんか？ 進行状況の把握はメールで十分にできていますか？ そろそろメールでは限界ではないだろうか？と、ふと感じたことがあると思います。

メールは手軽に相手に依頼を行い、情報提供できるツールですが、相手に伝わっているか確認することはできません。送信者が気軽にメールを送る一方で、大量の受信メールに埋もれて相手は気づいていない場合も多いです。これでは円滑なコミュニケーションは成り立たず、結果としてプロジェクトの情報共有や意思疎通が十分にできずに、プロジェクトが失敗に至ることもあるでしょう。

しかし、慣れてしまったメール文化を変えろ！と言われても、早々に変えるのは難しいです。ツールを導入するので今日からすべてツールを使って作業しなさいと言われても、すぐには慣れないのが現実でしょう。メールの手軽さを残しつつ、進行管理(プロセス管理)や履歴管理(トラッキング)を行う方法があれば、このような問題は解決すると考えられます。

プロジェクト管理ツール「JIRA」とアドオンの組み合わせで解決

プロジェクト管理ツールであるアトラシアン製品「JIRA(ジラ)」は多くの企業で利用されています。JIRAにはメールを課題(チケット)として取り込む機能や、チケットへの更新やコメントをメールで通知する機能がありますが、JIRAにメール送信者となるユーザが登録されていない場合には利用できません。

そこで今回は、このようなケースも補うためのアドオンと組み合わせる方法を紹介します。JIRAとメールを統合するアドオンはいくつかありますが、その中でも一番実績があって使いやすい「Email This Issue」を利用します。

「Email This Issue」アドオンの特徴

「Email This Issue」アドオンには次のような特徴があります。

●誰にでもメールでチケットを送信できる

ユーザがJIRAに登録済みかどうかにかかわらず、ロールやグループ、カスタムフィールドを利用して、チケットやコメント、添付ファイルをメールで送信できます。各チケットには「Email」ボタンが表示され、ここから直接メールを送信することもできます。JIRAがメールクライアントになるので、今まで利用されていたメールクライアントとのやりとりは不要となります。また、チケットの「Emails」タブからはチケットで受信、送信したメールの履歴を確認できます。

JIRAに関しては次の記事をお読みください。

http://gihyo.jp/dev/serial/01/project_manager_tool/0001

●何かが発生したタイミングで通知できる

チケットの作成、編集、コメント追加といったさまざまなイベントが発生したタイミングで、メール通知ができます。とくに優れているのは、通知の条件としてJQL(JIRA Query Language)を利用できる点です。チケットのステータスやコンポーネント、担当者などの条件によって、通知する／しない、メールの内容を変える、といった細かな指定が可能になります。JIRAチケットとして登録されたタイミングで、あらかじめ用意したメールテンプレートで自動応答するといったことも容易です。

●メールを受信してメールアドレスを保存できる

受信したメールからチケットを作成、コメントを追加するだけでなく、メールの送信元やCc:、Bcc:といったメールアドレスもカスタムフィールドへ登録できます。メールのCc:も保存できるので、依頼者はチームのメンバー間でやりとりを共有できます。

●送信メールがカスタマイズできる

JIRA管理者は、あらかじめ用意された送信メールのフォーマットを選び、それをカスタマイズしてメールテンプレートとして設定できます。メールはメールテンプレート(Apache JakartaのVelocity)で作成でき、保存する前にプレビューで確認できます。メールテンプレートはカテゴリ分類が可能で、チケットからメールを直接送信する際にも利用でき、無駄な手入力を極力減らすことができます。また、Field pickerを利用することで、チケットの項目をより手軽にメールの件名や本文に値として埋め込めます。

設定概要

実際の設定を見てみましょう。ここでは最小の設定で動作するようにしているので、お好みに合わせて変更してください。

ユーザ、プロジェクト、カスタムフィールドを用意する

まずは依頼者とメールでやりとりするため、「JIRA管理」>「ユーザー管理」>「ユーザー」からユーザをJIRAへ追加します。このユーザ(ここでは「セールス」とします)を、メールから作成されたチケットの暫定の担当者、報告者として利用します。

次に、依頼者からの作業依頼を受け付けてチケットを登録するプロジェクトを作成します。ここでは「プロジェクトタイプ」として“JIRA規定スキーム”を選択し、「名前」を“販売業務”、「キー」を“SALES”とします。プロジェクトリーダーには先ほど登録したユーザ「セールス」を指定し、規定の担当者とします。

さらに、JIRAから依頼者へメールで返答するため、送信元メールアドレス(From:、Cc:)を登録するカスタムフィールドを作成します(**図1**)。送信元メールアドレスは対象のチケットへ登録されます。

「Email This Issue」アドオンには、担当者がチケットに添付されたファイルの中から指定したものだけを依頼者へ送るための便利なカスタムフィールドである、“Issue Attachment Selector Field”が用意されています。このフィールドを「フィールドタイプ」に追加しておくと、メールの添付ファイルの容量を気にする必要がなくなり、無駄なファイルのやりとりもなくなるので便利です。

▼図1　カスタムフィールドの作成

問合せメール(Cc:)	テキストフィールド (複数行)	課題タイプ: グローバル (すべての課題)	• 既定の画面	⚙-
問合せメール(From:)	テキストフィールド (複数行)	課題タイプ: グローバル (すべての課題)	• 既定の画面	⚙-

「Email This Issue」の設定

ここからようやく、「Email This Issue」の設定です。インストールするには「JIRA管理」>「アドオン」>「Find new add-ons」から“Email This Issue”と検索します。評価目的なら“無料トライアル”が利用できます。

インストール後、「JIRA管理」>「アドオン」>「JIRA EMAIL THIS ISSUE」>「Configuration」から設定を開始します。

メール通知の設定

まず、依頼者へのメール通知に関するイベントを設定します。「Add」ボタンからテンプレートを追加し、そのテンプレートに対してイベントを登録します。ここでは「問合せ通知」というテンプレートを追加し、依頼者に自動応答したり返答したりするためのイベントを登録します。「Recipients(To:)」は先ほど登録したカスタムフィールドの“問合せメール(From:)”、「Copy recipients(Cc:)」は“問合せメール(Cc:)”とし、「Email Settings」には送信するメールのフォーマットとして、"Text email"を選択します。

プロジェクトとメール通知の関連付け

アドオンには、メール通知をプロジェクトや課題タイプといった条件と関連付けるための「Contexts」という設定が用意されています(**図2**)。「Add」ボタンによって追加可能で、「Notifications」に登録済みのメール通知、「Sender Name」に送信者の名前、「From Address」に依頼者へのメールの送信元メールアドレスを指定します。

受信したメールからのチケット作成設定

「Mail Handlers」では、受信したメールからチケットを作成する先となるプロジェクトや課題タイプを指定します。具体的には、依頼者から受信したメールの処理方式や、チケットを作成した際の自動応答を設定できます。

新しくハンドラーを作成し、「Project」は“販売業務”、「Issue Type」は“問合せ”、「Email Processing Strategy」は“Create or Comment Issues (Supports Split Regex)”とします。

送信メールサーバの設定

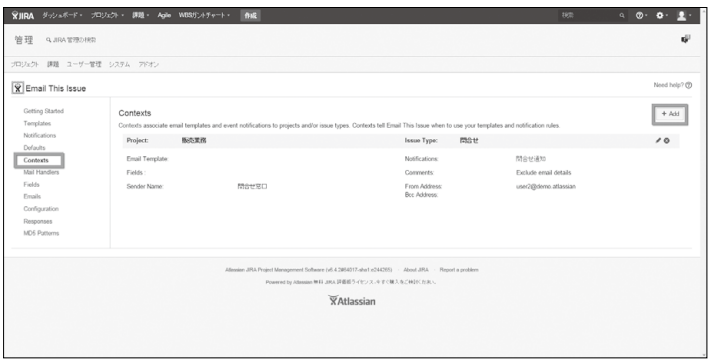
依頼者への回答をメールで送信するために、「JIRA管理」>「システム」>「送信メール」からSMTPメールサーバを設定します。各項目を設定し、「Test Connection」で接続が成功することを確認してください。

受信メールサーバとメールハンドラーの設定

作業の依頼をメールで受信するために、「JIRA管理」>「システム」>「受信メール」からPOP/IMAPメールサーバを設定します。送信メールサーバの設定と同様に各項目を設定し、テスト接続が成功することを確認します。依頼者には、作業を依頼するメールをPOP/IMAPメールサーバに指定したアドレスへ送信してもらうことになります。

続けて、同じ画面にある「受信メール ハンドラーの追加」から「Email This Issue Mail Handler」を選択して追加します。

▼図2 「販売業務」に「問合せ通知」を関連付ける



「プロジェクト」に“販売業務”、「課題タイプ」に“問合せ”、「規定の報告者」に最初に登録しておいたユーザを指定します(**図3**)。「ユーザーの作成」は、メールの送信者がJIRAに登録されていないユーザのメールアドレスだった場合に、自動的にユーザを登録してしまうオプションなので、チェックを入れないでください。

実際に利用してみる

設定が完了したら、実際に利用してみましょう。依頼者として、作業を依頼するメールをPOP/IMAPメールサーバに指定したアドレスへ送信します。JIRAがメールを受信すると自動でチケットが作成され、要約としてメール件名や送信元メールアドレスといった情報が登録されます。依頼者がJIRAにユーザ登録されている場合は、チケットの報告者として割り当てられます。受け取ったらすぐに検知できるように、JIRAの通知スキームやHipChatのルームへの通知を設定しておくとよいでしょう。あとは、タスクの優先度や担当者を割り当て、作業をこなし、終わったらステータスをクローズしていくことになります。

担当者がチケットへコメントすると、メールを通じて依頼者と情報共有ができます。また、依頼者は必要に応じてメールに返信することで、チケットのコメントを通じて担当者へ情報発信できます。これにより、それぞれが作業に集中できるのです。

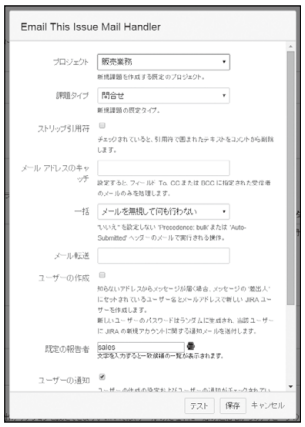
改善

メールを中心にタスク管理を行っていると、タスク内容を確認するたびに該当メールをスレッドから探し出したり、最後までフォローできていたかスレッドを追って確認したりする必要があります。1日数千通のメールが来るような状況では必要なスレッドを見つけ出すだけでもかなりの時間を要し、膨らみ続けるメールボックスによって作業の効率は悪くなる一方です。

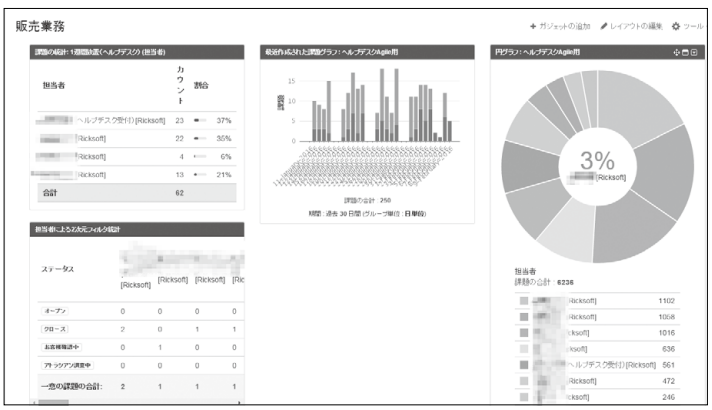
JIRAの場合、タスクはすべてチケットとして登録し、その後は進捗に応じてステータスや担当者を変更していきます。これだけで、プロジェクト、チーム、人ごとに、現在抱えているタスクや状況を、JIRAの優れた検索やガジェットによってリアルタイムに追うことができ、プロジェクトや会社全体で情報を共有できるようになります(**図4**)。

情報共有、可視化、意思疎通、これらがJIRAの利用による大きな改善ポイントとなります。SD

▼図3　「Email This Issue Mail Handler」の追加



▼図4　プロジェクトなどの単位で課題やステータスを可視化できる



Software

パラレルス、
Parallels Remote Application Serverの最新版「version 15」を発表

パラレルス㈱は2月17日、Parallels Remote Application Serverの最新版である「version 15 日本語版」の発売を開始した。

Parallels Remote Application Serverは、企業などの従業員が使用するあらゆるデバイスに仮想Windowsアプリケーション、および仮想化デスクトップを配信するための、企業向け製品。version 15ではおもに、ユーザの自動登録機能の追加、操作性の向上が図られた。

企業がBYOD（個人端末の業務利用）やCYOD（企業が選定した複数の端末から従業員が選んで利用すること）を実践していると、企業内にはさまざまな種類の端末が存在することになる。その端末に、WindowsおよびWindowsアプリケーションを配信するにあたり、本製品を導入することで、IT管理者は非常に柔軟な対応ができるようになる。また本製品には、レポーティング、HA（高可用性の提供）、二要素認証、Windowsクライアントマネージメントといった、リモートデスクトップおよびアプリケーション配信における機能が通常オプションとして備わっている。

料金はサブスクリプション形式で、同時利用ユーザ1人あたり12,000円／年となっている。

CONTACT
パラレルス㈱ URL https://www.parallels.com/jp

Report

ITエンジニアに読んでほしい！技術書・ビジネス書大賞、決定

㈱翔泳社が行う、「ITエンジニアに読んでほしい！技術書・ビジネス書大賞」の技術書部門大賞、ビジネス書部門大賞、ゲストによる特別賞が2月18日のDevelopers Summit 2016（以下デブサミ）にて決定し、発表された。

本大賞では、Webからの投票によって選出された技術書・ビジネス書のそれぞれベスト3について、デブサミのセッション内で著者または担当編集者がプレゼンを行い、会場の観覧者・特別ゲストの投票によって受賞書が決定される。結果は次のとおり。

○技術書大賞
『プログラマ脳を鍛える数学パズル シンプルで高速なコードが書けるようになる70問』

○ビジネス書大賞
『人工知能は人間を超えるか ディープラーニングの先にあるもの』

○ゲスト特別賞
『ワーク・ルールズ！』（大塚 弘記さん推薦）
『ハッカーの学校』（長田 絵理子さん推薦）
『HARD THINGS』（倉貫 義人さん推薦）

CONTACT
㈱翔泳社 URL http://www.shoeisha.co.jp

Hardware

アイ・オー・データ機器、
存在証明のためのタイムスタンプ付与機能を組み込んだアプライアンス製品「WE1-TS5/PACK」を発表

㈱アイ・オー・データ機器は2月24日、セイコーソリューションズ㈱が提供する長期署名クラウドサービス「eviDaemon」を組み込んだアプライアンス製品「WE1-TS5/PACK」を発表した。

「eviDaemon」は、デジタルデータの存在証明と、そのデータが改ざんされていないことを証明するタイムスタンプを付与する、クラウド型のエビデンスソリューション。標準規格「PDF長期署名（PAdES）」に準拠し、長期署名データの生成や検証ができる。eviDaemonが組み込まれたWE1-TS5/PACKを社内に配置して監視フォルダを設定することで、そのフォルダにファイル（おもにPDFが対象）がコピーされると、自動的にタイムスタンプが押されるしくみ。タイムスタンプが重要となる、特許出願といった業務を行う企業にはとくに重宝する製品と言える。

価格はオープン。発売は3月下旬を予定している。


▲WE1-TS5/PACK

CONTACT
㈱アイ・オー・データ機器
URL http://www.iodata.jp

Service

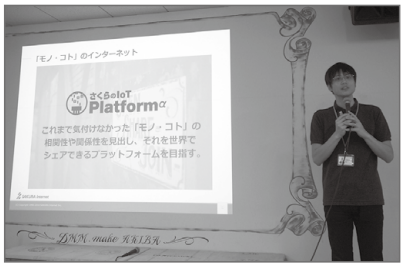
さくらインターネット、「さくらのIoT Platform」を発表

さくらインターネット㈱は2月8日、新サービス「さくらのIoT Platform」を2016年度中に提供開始することを発表した。

さくらのIoT Platformは、通信環境とデータの保存や処理システムを一体型で提供するIoTのプラットフォーム。ハードウェア製品である「さくらのIoT通信モジュール」と、キャリアネットワークをL2接続した閉域網を用意し、ストレージ、データベース、ルールエンジンを含むバックエンド、外部のクラウドやサービスと連携できるAPIまでを垂直統合型で提供する。

さくらのIoT通信モジュールは、UART/SPI/I²Cでマイコンなどからセンシングデータを取得し、キャリアネットワーク（ソフトバンク㈱または㈱ソラコム）を通じてさくらインターネットの閉域網にのみデータを送信する。送信先としてはパブリックな領域、プライベートな領域を選択でき、ユーザはAPIを通じてデータを利用する。将来的には、ユーザが収集したデータが第3者に利用される際、収集者に利益が還元されるしくみも構築していくとのこと。料金は月額・従量課金制ではなく、データのやりとり、プライベート領域への保存、インターネットとつなぐAPIの利用に対して課金される。

同社の代表取締役社長である田中氏によると、本サービスがイメージしているものは「モノのTwitter」。ネットワークにつながったモノが情報をつぶやき続け、データの送信者・受信者以外の第3者も、情報を利用できるエコシステムを目指すとのこと。今後の流れとしては、2016年4月より「さくらのIoT Platform α」、9月より「さくらのIoT Platform β」を提供する。



▲さくらインターネット㈱ 代表取締役社長 田中邦裕氏

CONTACT
さくらインターネット㈱ URL http://www.sakura.ad.jp

Report

F5ネットワークスジャパン、「F5 Japan Security Forum 2016」開催

2月10日、「F5 Japan Security Forum 2016」が、F5ネットワークスジャパン合同会社主催で開催された。その中の2つのセッションを紹介する。

○BIG-IP APMとPassLogicで実現する高セキュア＆高パフォーマンスのマルチデバイス認証インフラ

パスロジ㈱の酒井寛庸氏が紹介したのは、2月8日に同社から発表された、SSL-VPN業界初となるスマートデバイスの端末固有情報自動登録機能。本機能はF5ネットワークスジャパンとの共同開発で、リモートアクセス装置「BIG-IP APM（Access Policy Manager）」と連携する形で提供される。

端末の多様化・BYODの普及によって、1人の社員が複数の端末を使ってさまざまな場所から企業のネットワークにアクセスするという場面が格段に増えたが、場合によっては数万台にもおよぶような端末の固有情報を認証のためのDBに登録するのは、運用担当者にとって非常に手間のかかる作業となる。

今回発表されたソリューションにおいてはユーザは、当該端末でBIG-IP APMに一度アクセスするだけで認証DB内に端末固有情報が自動登録されるので、担当者による登録の手間が大幅に削減される。

○総SSL通信化に備える。SSLの技術動向とセキュリティ対策

F5ネットワークスジャパンの桐谷彰一氏からは、SSLの普及と問題点についてのセッションが行われた。

現在、インターネットの全通信の1/4がすでにSSL化済みで、フリーのSSL/TLS証明書発行サービス「Let's Encrypt?」の登場などにより、今後も毎年30%の増加が見込まれている。しかし、SSLの増加には「セキュリティ対策製品が通信を検査できない」「証明書の運用・管理で業務が煩雑になる」といった問題点もある。

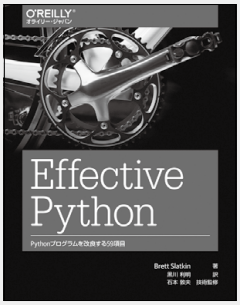
これら問題に対して桐谷氏は、BIG-IPのようなロードバランシング機能を持った製品でSSLを終端し、証明書を一括管理するといった方法が有効であると語った。

CONTACT
F5ネットワークスジャパン合同会社 URL https://f5.com/jp

エンジニアのための Gitの教科書

株式会社リクルートテクノロジーズ、株式会社リクルートマーケティングパートナーズ、河村 聖悟 ほか 著
B5変型判／200ページ
2,200円＋税
翔泳社
ISBN＝978-4-7981-4366-8

Gitの初級者・中級者に向けたGitの本。本書まえがきによると、「古くならないGitの普遍的な知識」「検索しても見つからない現場のノウハウ」を伝えることを念頭に置いている1冊である。第1章では、初級者がつまずきやすいワーキングディレクトリ、ステージングエリア、リポジトリの位置関係やインストール方法、基本コマンドを解説している。第2章では、基本操作は覚えたが運用はこれからという中級者に向けて、git-flowなどのブランチの運用手法、運用戦略といった、Gitを活用したチーム開発手法を解説している。最後の第3章ではそのさらに発展として、GitHub-flowというブランチ運用戦略、GitやGitHubのフック機能を活用した継続的インテグレーション／デリバリも紹介している。

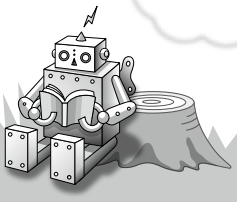


Effective Python

Brett Slatkin 著／黒川 利明 訳／石本 敦夫 技術監修
B5変型判／256ページ
3,200円＋税
オライリー・ジャパン
ISBN＝978-4-87311-756-0

Pythonの人気がこれほどまでになるとは、隔世の感ありではなかろうか。今やPythonは、ブームになっている機械学習の分野でよく使われることもあり、若い人にとってもはや当然かつ必須の言語の1つになっている。本書の内容は、最初はかなり基礎的に項目1「Pythonのバージョンを知っておく」から始まり、項目2「PEP 8スタイルガイドに従う」と続き、中ほどでは項目37「スレッドはブロッキングI/Oに使い、並列性に使うのは避ける」などのように次第に難易度が上がっていく構成である（全部で59項目）。本書に入門書的な内容を期待する読者は少ないと思われるが、Pythonのスペシャリストの工夫や考え方を見ることができるので、より優れたコーディング能力を得たいと考える方にとって良い手本になるだろう。

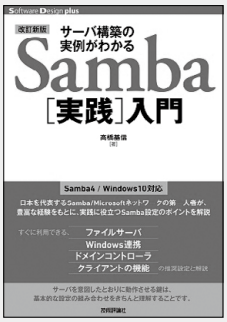
SD BOOK REVIEW



ネットワーク運用管理の教科書

のびきよ 著
B5変型判／272ページ
2,980円＋税
マイナビ出版
ISBN＝978-4-8399-5780-3

ネットワーク運用管理業務を「定常業務」「非定常業務」「トラブル対応」「Q&A対応」の4つに分け、前から3つについて章を設けて、必要とされる技術ポイントを解説している。定常業務の章では、日々のルーチンワークをこなすためのネットワークの構成、各機器の特性といった基本知識を解説している。非定常業務の章では、業務システム変更に伴う経路の変更やIPv6の導入など、その多くは手順書がない中での作業ノウハウを紹介している。そして、トラブル対応の章では、問題の切り分け方やログ解析について深掘りしている。そのほか、ユーザの突発的な頼みを安易に受けない、小さなトラブルでも上長とコンセンサスをとってから対応するといった仕事面でのケーススタディも紹介されており、企業での実業務を志向した本と言える。



【改訂新版】サーバ構築の実例がわかるSamba［実践］入門

髙橋 基信 著
A5判／256ページ
2,680円＋税
技術評論社
ISBN＝978-4-7741-8000-7

2010年に出版された『サーバ構築の実例がわかるSamba［実践］入門』の改訂版である本書は、最新のSamba 4.xとWindows 10に対応したものだ。前書にあったNTドメインの章がなくなり、その分、Active Directoryのドメインコントローラ機能の解説が追加されている。元々のコンセプトである「GUIを使わず設定ファイルを直接修正し、コマンドラインから設定を行い、具体例を掲載する」という方針はそのままで、現行のLinuxディストリビューション（CentOS 7/Ubuntu 14.04 LTS/FreeBSD 10）での解説が行われている。

現在ではWebを検索すれば、smb.confの設定は多く見つけられるが、「何がどうなってこういう設定なのか」という部分を理解をするために、本書は役立つだろう。

ひみつのLinux通信

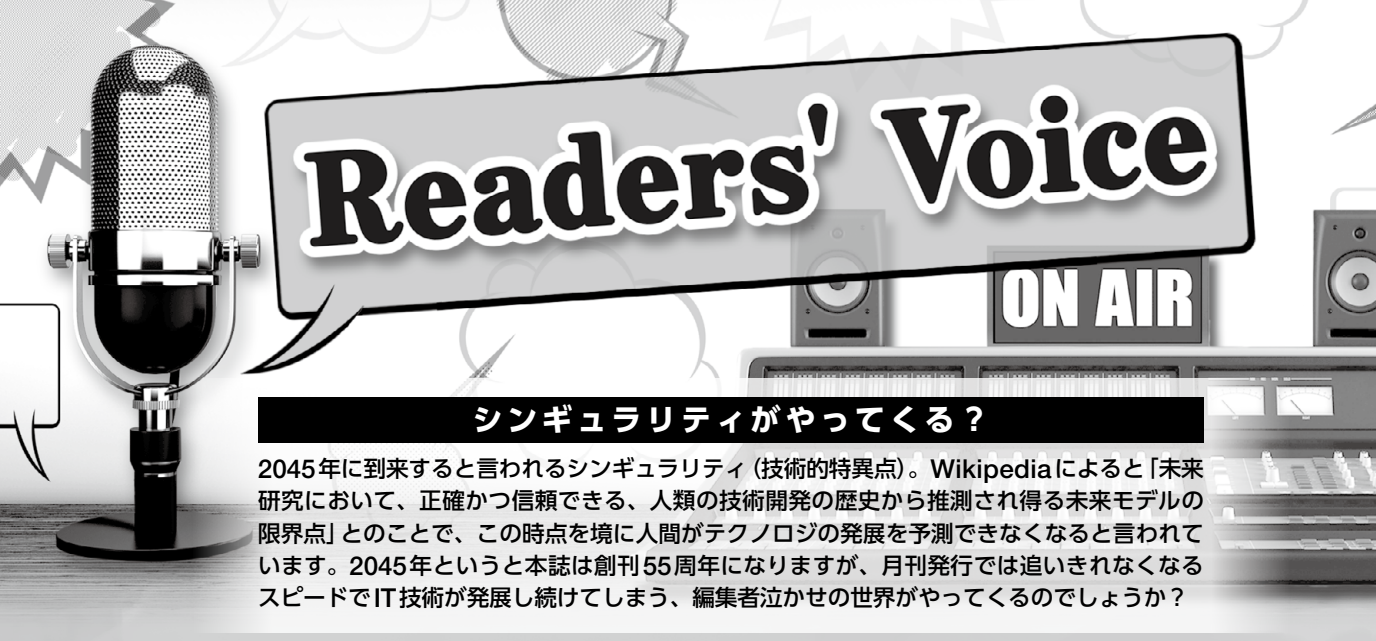
作)くつなりょうすけ
@ryosuke927

第26回 シグ松さん

アレほどBL展開はダメよと釘を挿したのに作画しようとした裏乙女なくつな先生の連載が読めるのは本誌だけ！

to be Continued

あるプロセスにメッセージを送るシグナルという仕組みがあります。キーボードの Ctrl＋C を押せば「INT」シグナルが送られて実行中のプロセスが終了、Ctrl＋Z を押せば「TSTP」が送られて一時停止します。キーボードの他に、killコマンドで、シグナルと送信先プロセスを指定して送ることもできます。killは「プロセスを殺すコマンド」なんて物騒なものではなく、実は「プロセスにシグナルを送るコマンド」なんですよ。「man kill」を見ると好奇心爆発してソースコードに飛び込みたくなりますね。「kill -1 -9」の例も実行してみたくなり、趣き深い。他にもpkill、skill、killallなどのシグナルコマンドがあります。これらでも6つ子できるかな？



シンギュラリティがやってくる？

2045年に到来すると言われるシンギュラリティ（技術的特異点）。Wikipediaによると「未来研究において、正確かつ信頼できる、人類の技術開発の歴史から推測され得る未来モデルの限界点」とのことで、この時点を境に人間がテクノロジの発展を予測できなくなると言われています。2045年というと本誌は創刊55周年になりますが、月刊発行では追いきれなくなるスピードでIT技術が発展し続けてしまう、編集者泣かせの世界がやってくるのでしょうか？

2016年2月号について、たくさんの声が届きました。

第1特集　MySQLとPostgreSQL徹底比較

OSSの2大データベース管理システム「MySQL」「PostgreSQL」について、利用シーン・アーキテクチャ・ライセンス・SQL構文・機能性・拡張性・新追加機能・コミュニティといった観点から徹底的に比較しました。

基本的にMySQLしか使ったことがなかったので良い勉強になった。
tokichieさん／東京都

実は第1特集の著者の知り合いで、「特集読んだよ！」と教えてあげたところ、たいそう喜んでいました。
福名一さん／岡山県

何に気を付ければ良いか、わかりやすく書かれていた。　**桑村さん／兵庫県**

Tips的な内容も詰まっていて良かったです。MariaDBの紹介もライセンスのとこであったほうが良かったかもしれません。　**コメットさん／兵庫県**

MySQLとPostgreSQLにこんなに違いがあるとは知りませんでした。
psiさん／東京都

会社での比較検討の材料として使えそう。　**くんさん／石川県**

仕事ではいつもPostgreSQLしか使っておらず、MySQLを使用するモチベーションがいまいち湧かなくて違いもわからなかったのですが、この特集でMySQLに興味を持つことができました。今回の特集ではなかったのですが、レプリケーションなどの機能も深堀りした記事が読みたいです。
今井さん／千葉県

「こんな違いがあったなんて……」という声が多く寄せられました。DBMSの性能はシステムの性能に直結すると言っても過言ではないため、扱うデータや利用シーンによってうまく使い分けができれば良いですね。

第2特集　適切なLANケーブリングの教科書

1Gbps超時代を迎えるにあたり、通信ケーブルの選択やラック内の配線といった、ネットワークエンジニアに求められるケーブリング技術を特集しました。ケーブルの規格からPoEのしくみまで盛りだくさんの内容でした。

敷設してしまうと忘れられがちなものですが重要です。　**下平さん／東京都**

openDCIMは知らなかったですが、便利そうなので使ってみます。
橿山さん／埼玉県

ケーブルの性能が上がった分、扱いも大分デリケートになったと感じる。エアフローへの影響も大きい。
隼さん／岩手県

とっても参考になりました。社内サーバ室も同じ問題になりがちなのでよかったです。　**引田さん／埼玉県**

有線LANの特集記事はたいへんわかりやすかったです。次回は、ぜひ無線LANを有線並みに安定して使用する方法などをお願いします。
中村さん／大阪府

今回は最初から最後までハードウェアのお話でした。クラウドが普及しても社内サーバは残り続けると思うので、ケーブルやラックについてのノウハウは、会社の中で誰かが知っておかなくてはならない知識と言えますね。

一般記事　Android Studioのスタイルで効率アップ！

Androidアプリ開発に最適な統合開発環境「Android Studio」の特集です。お

2月号のプレゼント当選者は、次の皆さまです

①ハンディ洗濯機「COTON」
尾曲克己様（愛知県）

②Lexar JumpDrive M20i（32GB）
まっきー様（東京都）

③ExcelCreator 2016
梅田嗣也様（愛知県）

④プログラマのためのDocker教科書
勇和博様（大阪府）、d_kawakawa様（茨城県）

⑤『新・明解C言語 実践編』
田村拓哉様（大阪府）、横山良英様（神奈川県）

⑥『サイバーリスクの脅威に備える』
橋本裕一郎様（埼玉県）、永作肇様（東京都）

⑦『お金をドブに捨てないシステム開発の教科書』
澤崎敏幸様（富山県）、n0ts様（東京都）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

もにAndroid Developer Toolsからの移行者に向けて、Android Studioでの効率的なアプリ開発の方法を解説しました。

自分も使っているので、とても関心を持てた。 宮寄さん／大阪府

開発環境はもの凄く効率に影響すると思います。IDEをいくつか渡り歩いてますが、ちょっと便利なエディタ（jEdit）が一番使用頻度が高いかもしれません。 NGC2068さん／愛知県

Android Studioの性能・魅力を最大限に活用するヒント満載で良かったです。 オミオさん／宮城県

けっこう、今まで効率が悪かったのがわかりました。 佐伯さん／熊本県

Androidアプリの開発にはマストとも言えるAndroid Studioですが、IntelliJ IDEAがベースとなっているだけあってやや癖があるようです。特徴を押さえて、機能に振り回されないようになることが肝心ですね。

短期連載　クラウド時代のWebサービス負荷試験再入門【3】

ITインフラの中心がオンプレミスからクラウドへ移るに伴って、システムに対する負荷試験も、それに合わせたものを用意する必要があります。本連載ではクラウドに載せたWebサービスの「スケーラブル」を担保するための負荷試験について見ていきます。第3回では実際の負荷試験の進め方を解説しました。

自社の運営しているサービスと構成が似ているため非常に参考になりました。段取りや目的についてあいまいになることが多く、負荷試験をすること自体が目的になることもあったため、とくに留意したいです。 yurakawaさん／東京都

個人的には、10数年以上前から（Webではないもので）似たような負荷試験みたいなことをしていました。その当時からの方法や考え方とほぼ同じだったので、自分で考えていたことがまちがいではなかったと安心できました。 福田さん／神奈川県

負荷試験においては、想定される実際の負荷を再現することが大事ですね。TVで取り上げられたり、有名人にSNSでつぶやかれたりといった「○○効果」「○○砲」と呼ばれる事態を具体的にイメージして、負荷試験を進める必要があります。

短期連載　SMB実装をめぐる冒険【最終回】

Windows共有フォルダをChromeOSのファイルアプリにマウントするアプリ。その開発には「SMBプロトコル」の理解とJavaScriptでの実装が必要でした。最終回では、アプリ側にファイル操作を実現していきました。

探索って、辛いけれどもおもしろい作業ではあります。 鈴木さん／熊本県

どれだけ苦労したのか、考えさせられる記事。 とーふやさん／神奈川県

本連載で4回に渡って開発の過程を解説したアプリは、「File System for Windows」という名前でChromeウェブストアに公開されています。気になった読者の方は、ぜひ使ってみてください。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
May 2016

2016年5月号
特別定価（本体1,420円+税）
192ページ

4月18日発売

［特別付録］
IoTを始めよう!　SORACOM SIM限定バージョン

［第1特集］すべての道はエディタに通ず
［超定番］Vim入門
――ステップアップのための特別講座

インフラエンジニアも、プログラマーも、Webデザイナーも、先輩達はみんなVimを使っていませんか？　新年度の今、初心者からデキる人にステップアップしましょう。まずはWindows/Linux/Macで環境設定を盤石にするノウハウを紹介します。初心者が押さえておくべきVimのテクニックを押さえてから中級者になるための実技を紹介、そして覚えておくとよいVimキーバインドをまとめ、Git/GitHubとの連携についてももちろん言及します。Vimでエンジニアの世界に仲間入りしましょう!

［第2特集］2年ぶりのLTS
安定のUbuntu 16.04の新機能

2年に一度のUbuntu長期間サポート版（LTS）がリリースされます。Ubuntuの基礎知識から16.04 LTSの変更点の概要、Ubuntuとそのフレーバー（Ubuntuをベースとし、デスクトップ環境を変更した公式派生版）、そして広く使われるようになったサーバの、Ubuntu独自機能を中心とした使い方を、Ubuntu Japanese Teamのメンバーが総力を挙げてお届けします。

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

SD Staff Room

●2月某日、いきなりノロウィルスに感染し、たいへんヤバイことに。嘔吐発熱下痢。私から娘へ感染し、数日遅れて家内に感染して、家庭機能ダウン。あやうく幼子を誰もみれない危機に。その間に仕事は遅延し、いろいろな方に迷惑をかけてしまいました。国家予算を投じてノロウィルスワクチンを開発すべきだ!（本）

●2月末に「かまぶの部屋」の打ち上げをやりました。登場20人中12人が参加。ありがとうございます。編集後記に写真載せろという提案されましたが、いかがですか?（幕）

●ただいま人身事故にて電車停車中。あと2駅なのでそのまま乗って待つか、動き始めた反対方面の電車に乗って3駅戻って別の路線に乗り換えるか。お、反対側の電車が入ってきたぞ。チャレンジする？　1本スルーして次？　いやそれこそ失敗する選択だ、どうする!　プチ・“未来の分岐点”を堪能。（キ）

●久しぶりにNHK大河ドラマ（今年は「真田丸」）を見ています。ちゃんと続けて見られているのは、2008年の「篤姫」以来です。大河ドラマは、歴史を楽しく学ぶには良いのですが、所詮は史実をもとにしたフィクションなので、どこが史実でどこが作り話なのかがわからないところが難点です。（よし）

●本誌過去記事をまとめた『オブジェクト指向をきちんと使いたいあなたへ』が発売中です。著者は計13人と大所帯!　1つのテーマについていろいろな人の考えが読めるのは、ムック本ならではです。書店で見つけたときは、ぜひ手に取ってみてください。“組み木で形作った矢印（↑）”が目印です。（な）

●手縫いで小さなぬいぐるみを作っていて、先日初めて自己企画ではないグループ展にお呼ばれして参加させていただきました。会期中、たくさんの出逢い、温かいお言葉をいただけたのが何よりも嬉しかったです。これを励みに今後も頑張ります。（弊社某誌で執筆されていたイラストレータさんも参加されていてびっくりぼん!（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

［E-mail］
sd@gihyo.co.jp

［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。

Software Design
2016年4月号

発行日
2016年4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
中田瑛人

●編集アシスタント
根岸真理子

●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。