Special Feature 1
→手になじむエディタはどれだ？
Special Feature 2
→Pythonのたしなみ

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

June 2017

6

2017年6月18日発行
毎月1回18日発行
通巻386号
（発刊320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

決定版 あなたのプログラミングを加速させるエディタ

【 Special Feature 1 】

これだけは知っておきたい設定と操作

Vim
Emacs
Atom
Visual Studio Code

マルチに使えて、コードも読みやすい、保守も楽！

【 Special Feature 2 】

今すぐはじめる Python

【 Extra Feature 1 】
「ハッシュ関数」とは何か？

【 Extra Feature 2 】
Windows Server 2016で構築する
最新ファイルサーバ（前編）

【 Extra Feature 3 】
［ニフティクラウドmobile backend］
mBaaSでIoT（第3回）

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。

目次

Contents

Software Design
June, 2017
6

Special Feature 1 → 第1特集 | Page

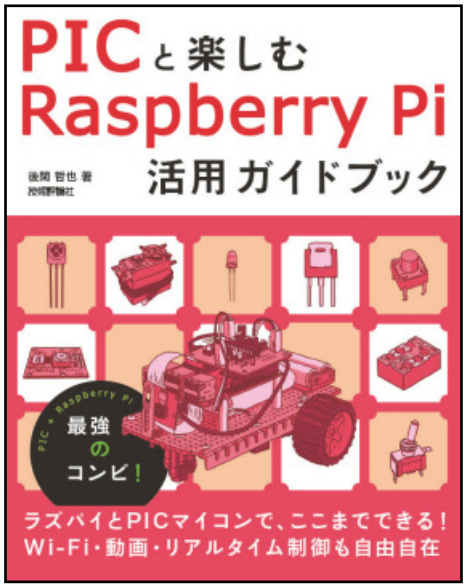
Contents

Column

Development

OS/Network

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。



ISBN978-4-7741-8919-2
B5変形判／384ページ
定価（本体2480円+税）

PICと楽しむ Raspberry Pi 活用ガイドブック

後閑哲也 著

その手軽さから電子工作愛好者に多用されているPICマイコンは、センサやモータなどのきめ細かな制御や、高速なフィードバック制御なども得意です。その反面、ネットワークに接続したり、動画を扱ったりしようとすると壁に突き当たります。
しかしRaspberry Piの力を借りれば、動画や音声、インターネットへの接続が圧倒的に楽になります。
本書では、8ビットのPICマイコンにRaspberry Piを接続して、高機能部品として使うためのノウハウをとことん解説します。汎用入出力GPIOの使い方、おしゃべり時計やリモコンカーなどの制作例、最低限必要なLinux・Pythonの知識まで収録しています。

ISBN978-4-7741-8921-5
B5変形判／192ページ
定価（本体2980円+税）

Intel Edison マスターブック

IoTデバイスをつくろう

北神雄太 著

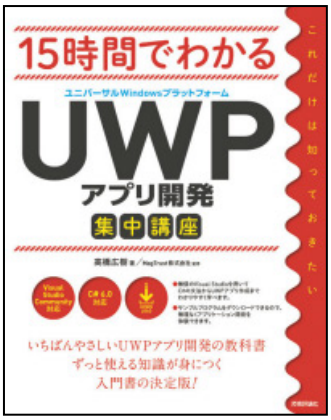
「Edison」はIntel製のコンピュータモジュールです。乾電池で動作する省電力性を持ちながら、無線LANやBluetooth、高性能プロセッサを搭載していて、話題のIoT関連のハードウェア/ソフトウェア開発に役立ちます。
本書では基本となるセットアップやLチカから、気温センサーや加速度センサーの活用、得られたデータの処理方法まで扱っているので、Intel Edisonの様々な可能性を試すことができます。

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

15時間でわかる 集中講座 シリーズ

いちばんやさしい
プログラミングの教科書
ずっと使える知識が身につく
入門書の決定版！

ISBN978-4-7741-7892-9
小田切篤、露木誠　著
B5変形判／352ページ
定価（本体2680円+税）



ISBN978-4-7741-8695-5
高橋広樹 著／MagTrust株式会社 監修
B5変形判／416ページ
定価（本体2780円+税）

ISBN978-4-7741-6798-5
宮下明弘、工藤雅人、原田僚 著／井上誠一郎 監修
B5変形判／416ページ
定価（本体2680円+税）

ISBN978-4-7741-8590-3
宮下明弘、工藤雅人 著
B5変形判／384ページ
定価（本体2680円+税）

ISBN978-4-7741-7244-6
馬場俊彰 著
B5変形判／400ページ
定価（本体2680円+税）

ISBN978-4-7741-7893-6
高橋広樹 著
B5変形判／416ページ
定価（本体2980円+税）

ISBN978-4-7741-6575-2
岡本隆史、織田翔、大山智之 著
B5変形判／256ページ
定価（本体2580円+税）

これだけは知っておきたい

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

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

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/

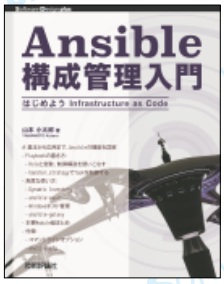
2 >> 定期購読受付専用ダイヤル
0120-223-223（年中無休、24時間対応）

Software Design plus

最新刊！

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

打田智子、大須賀稔、大杉直也、西潟一生、西本順平、平賀一昭 著
B5変形判・392ページ
定価 3,800円（本体）+税
ISBN 978-4-7741-8930-7

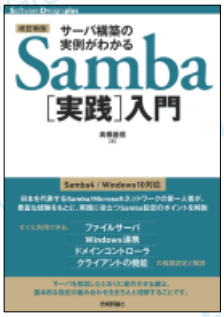


山本小太郎 著
B5変形判・176ページ
定価 2,480円（本体）+税
ISBN 978-4-7741-8885-0

養成読本編集部 編
B5判・144ページ
定価1,780円（本体）+税
ISBN 978-4-7741-8865-2

星野武史 著、花井志生 監修
A5判・256ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-8729-7

小川晃通 著
A5判・272ページ
定価 2,280円（本体）+税
ISBN 978-4-7741-8570-5



髙橋基信 著
A5判・256ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-8000-7

中井悦司 著
B5変形判・272ページ
定価 2,980円（本体）+税
ISBN 978-4-7741-8426-5

前橋和弥 著
B5変形判・304ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-8188-2

五味弘 著
B5変形判・272ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-8035-9

神原健一 著
B5変形判・192ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-7749-6

養成読本編集部 編
B5判・168ページ
定価1,980円（本体）+税
ISBN 978-4-7741-8360-2

養成読本編集部 編
B5判・232ページ
定価2,080円（本体）+税
ISBN 978-4-7741-8385-5

養成読本編集部 編
B5判・200ページ
定価2,080円（本体）+税
ISBN 978-4-7741-8034-2

養成読本編集部 編
B5判・112ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7992-6

Dockerエキスパート養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7441-9

サーバ／インフラエンジニア養成読本 基礎スキル編
福田和宏、中村文則、竹本浩、木本裕紀 著
定価 1,980円+税　ISBN 978-4-7741-7345-0

Laravelエキスパート養成読本
川瀬裕久、古川文生、松尾大、竹澤有貴、小山哲志、新原雅司 著
定価 1,980円+税　ISBN 978-4-7741-7313-9

Pythonエンジニア養成読本
鈴木たかのり、清原弘貴、嶋田健志、池内孝啓、関根裕紀、若山史郎 著
定価 1,980円+税　ISBN 978-4-7741-7320-7

データサイエンティスト養成読本 R活用編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7057-2

Javaエンジニア養成読本
きしだなおき、のざきひろふみ、吉田真也、菊田洋一、渡辺修司、伊賀敏樹 著
定価 1,980円+税　ISBN 978-4-7741-6931-6

JavaScriptエンジニア養成読本
吾郷協、山田順久、竹馬光太郎、智大二郎 著
定価 1,980円+税　ISBN 978-4-7741-6797-8

WordPress プロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6787-9

サーバ／インフラエンジニア養成読本 ログ収集〜可視化編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6983-5

フロントエンドエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6578-3

PHPライブラリ&サンプル実践活用［厳選100］
WINGSプロジェクト 著
定価 2,480円+税　ISBN 978-4-7741-6566-0

改訂版　Zabbix統合監視実践入門
寺島広大 著
定価 3,500円+税　ISBN 978-4-7741-6543-1

Vyatta仮想ルータ活用ガイド
松本直人、さくらインターネット研究所、日本Vyattaユーザー会 著
定価 3,300円+税　ISBN 978-4-7741-6553-0

アドテクノロジープロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6429-8

技術評論社

養成読本編集部 編
B5判・112ページ
定価2,180円(本体)+税
ISBN 978-4-7741-8894-2

養成読本編集部 編
B5判・192ページ
定価1,980円(本体)+税
ISBN 978-4-7741-8863-8

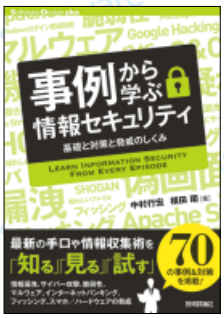
養成読本編集部 編
B5判・160ページ
定価2,180円(本体)+税
ISBN 978-4-7741-8895-9

養成読本編集部 編
B5判・240ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-8877-5

香山哲司、小野寺匠 著
A5判・176ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-8815-7

高宮安仁、鈴木一哉、松井暢之、
村木暢哉、山崎泰宏 著
A5判・352ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-7983-4

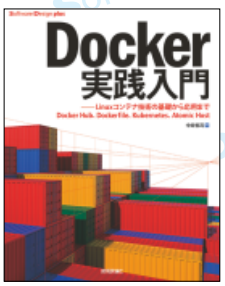
斎藤祐一郎 著
A5判・160ページ
定価 2,280円(本体)+税
ISBN 978-4-7741-7865-3



中村行宏、横田翔 著
A5判・320ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-7114-2

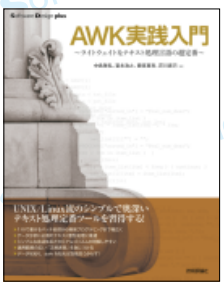
川本安武 著
A5判・400ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-6807-4

勝俣智成、佐伯昌樹、
原田登志 著
A5判・288ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6709-1

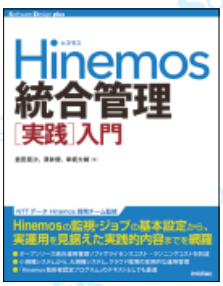


中井悦司 著
B5変形判・200ページ
定価 2,680円(本体)+税
ISBN 978-4-7741-7654-3

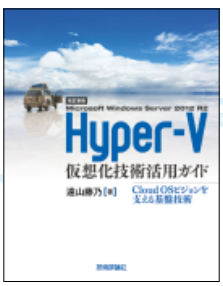
上田隆一 著
USP研究所 監修
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7344-3



中島雅弘、富永浩之、
國信真吾、花川直己 著
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7369-6



倉田晃次、澤井健、
幸坂大輔 著
B5変形判・520ページ
定価 3,700円(本体)+税
ISBN 978-4-7741-6984-2



遠山勝乃 著
B5変形判・392ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6571-4

養成読本編集部 編
B5判・176ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7993-3

養成読本編集部 編
B5判・192ページ
定価2,480円(本体)+税
ISBN 978-4-7741-7858-5

養成読本編集部 編
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7320-7

養成読本編集部 編
B5判・192ページ
定価 2,280円(本体)+税
ISBN 978-4-7741-7631-4

養成読本編集部 編
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7607-9

〒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102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。

WebGPU

次世代のWeb 3Dグラフィックス API

「WebGPU」は、Apple社から新たに発表されたWebブラウザ向けの3Dグラフィックス APIです。最新のGPU機能を活用したクロスプラットフォームなAPIの開発を目指したもので、WebブラウザエンジンWebKitの開発チームが提唱し、標準仕様の策定に向けた検討を開始しました。

現在は「WebGL」がWebブラウザにおける3Dグラフィックス描画のための標準技術として利用されています。WebGLは、組み込みシステム向けの3Dグラフィックス APIであるOpenGL ESをベースとした技術であり、Webサイトのコンテンツにおいて GPU機能を利用した高速な2D/3Dグラフィックスの描画を実現することが可能です。Safariを含むほとんどの主要なWebブラウザでサポートされており、PCだけでなくタブレットやスマートフォンでも高度なグラフィカル表現を行うことができます。

その一方で、昨今のGPU技術は日進月歩で改良されており、より高性能な新しいソフトウェアAPIが誕生しているという現状もあります。その一例としては、MicrosoftによるDirect3D 12、AppleによるMetal、Khronos GroupによるVulkanなどが挙げられます。これらの新しいAPIは、最新のGPUの能力を最大限に引き出せるように設計されており、OpenGLよりも低いオーバーヘッドでより優れたパフォーマンスを実現しています。しかし、プラットフォームへの依存性が高く、Webという標準プラットフォームから利用できるようにするにはハードルが高いという問題があります。

そこで、このような最新のGPU技術を活用することができる、WebGLよりもさらに一歩進んだ標準APIを構築しようということで提唱されたのがWebGPUです。

WebGPUに求められる要件

WebKitチームは、WebGPUの発表と同時に、新しいグラフィックAPIについて検討するコミュニティグループをW3Cに立ち上げました。このグループのゴールとして、次のような要件を満たす技術を提供することが掲げられています。

- モダンなWebプラットフォームのデザインパターンに対応するAPI
- アプリケーションのパフォーマンスを向上させるローレベルなAPI
- よりモダンなAPIによって提供され、多くのデバイス上で動作するGPU機能を利用できるAPI
- モダンなローレベルAPIを持つすべてのプラットフォームでリーズナブルに実装できる技術
- JavaScriptとWebAssemblyの両方に効率的なバインディングを提供できるように設計されたAPI

もともと、WebGPUへの取り組み自体はMetalのAPIをJavaScriptにマッピングすることからスタートしたそうです。とはいえとくにターゲットをMetalのみに限定するつもりはないとのことで、プロジェクトのスコープには、Direct3D 12やVulkanをはじめとした最新のGPUライブラリの上で実装できる、十分な汎用性を持つAPIを目指すことが明記されています。シェーディング言語についても、同様の理由でプロトタイプではMetal Shading Languageを採用していますが、最終的にはSPIR-Vのような IR形式の言語を受け入れる予定だそうです。

WebGPUと並んで提案されている次世代のWeb 3Dグラフィックス APIとしては「NXT」があります。こちらはGoogleのChromiumチームが中心となって仕様策定とプロトタイプの開発を進めているもので、WebGPUと同様にDirect3D 12やMetal、Vulkanにインスパイアされたクロスプラットフォームな APIとなっています。初期のプロトタイプはOpenGLまたはMetalの上で動作し、JavaScriptとWebAssemblyから呼び出すことができるとのことです。シェーディング言語としてはSPIR-Vが採用されています。

WebGPUもNXTもまだスタートしたばかりのプロジェクトであり、現在はそれぞれが独自の提案を行っている段階にすぎません。今後、双方の提案やプロトタイプ実装をベースとしながら、W3Cのもとで標準化に取り組んでいくことになるでしょう。SD

GPU for the Web Community Group
https://www.w3.org/community/gpu/

DIGITAL GADGET vol.222

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

デジタルガジェット的な文房具

そもそも文房具とは?

読者のみなさんも学校に通っていた時代は文房具とは無縁ではいられない毎日だったと思います。それぞれのこだわりや好みが色濃く反映されるのも文房具の特徴です。昨今パソコンやスマートフォンを始めとするデジタルデバイスが浸透し、昔ほど、手書きで文字を書く場面は少なくなってきました。デジタルデバイスでメモを取ることはできますが、それでも紙とペンに匹敵する記録や発想のためのツールはいまだ存在せず、お気に入りのペンや手帳を愛用している方も多いのではないでしょうか?

そもそも文房具とは何でしょうか?一般的な「文房具」の定義としては、何か物を書くのに必要な、書くものと、書かれるもの、そしてその補助となる道具全般を示します。「文具」とも言います。

スマートフォンの登場で売れなくなったものがいろいろあります。カメラ、電卓、電子辞書、腕時計などが主なものですが、その中の1つが文房具かもしれません。そうは言っても、デジタルデバイスで代替できる文房具の機能もあれば、代替できないものもあるでしょう。

さて、文房具の利点を考えてみましょう。

- 特定の用途向けに考えられているため、その用途に集中できる
- 手帳やノートなど、書いたものがひとところにまとめられる
- 使い心地の良い文房具や、モチベーションアップ、やる気につながる
- 文房具そのものを選ぶ、集める、使う楽しみがある
- 自分に合った文房具であれば、身体の延長として使いこなすことができる

そういった旧来からある文房具の良さ、便利さは、どれだけデジタルデバイスに展開されているでしょうか? ノートやペンは、単にメモをするためだけのものではなく、考えたり、整理したり、知的生産のための道具でもありました。スマートフォンとアプリによって、それらを代替できなくはないのですが、デジタルデバイスはおもにコンテンツや、時間を消費するためのモノとなってしまい、何かを作り上げるためのクリエイティブな道具ではなくなってきています。もちろん、スマートフォンですばらしい映像や写真を撮影する人や、スマートフォンで小説を書き上げるような人もいますが、全体からするとごくわずかでしかないでしょう。

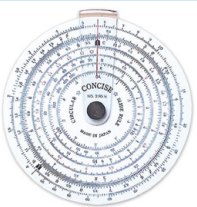
デジタルデバイスと文房具の関係

さて、文房具から進化したデジタルデバイス上のツールにはどういったものがあるでしょう? デジタル化によって必ずしも進化したとは言えず、退化したものもあるかもしれません。また、アナログ的な文房具にはなかった新しい価値を生み出しているものもあるかもしれません。

デジタル的な発想で、今までになかった文具的機能を持ったもの

T-EM10、T-EM100:
電動テープカッター(コクヨ)
https://www.kokuyo-st.co.jp/stationery/tapecutter/

セロテープを自動で繰り出し、自動でカットしてくれる文房具です(**pic.1**)。大量のテープ貼り作業のときに役立ち、テープを引き出している時間が省け、テープの量が多過ぎたり、少な過ぎたりすることもなく、ピッタリの長さに切断済みで繰り出してくれます。



コンサイスの計算尺。国内で唯一、現在も一般的に販売されている計算尺のメーカー。定規で計算できるだなんて、にわかには信じられないかもしれません。重量計算用、時間計算用、空調設備の計算や、工作機械の加工時間計算専用など、特定用途の計算尺も存在します。

※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。

従来の文房具風のデジタルデバイス

HiTAP:デジタルスタンプ（エム・フィールド）
https://hitap.jp

HiTAPは、スマホアプリに押して、スタンプの代替ができるしくみです（**pic.2**）。スタンプラリーやポイントカードなどのスタンプをスマートフォンに置き換えることができます。HiTAPでは、Ocellyという擬似的にマルチタッチするスタンプ状の機器を使用します。スタンプを押すような操作で、20種類の組み合わせのタッチ操作を模倣し、IDを取得するサービスに利用できます。ほかにも同等のしくみとして、JOUJOU Cube touch（タカラトミー）、PlusZone/Stamp（NECネッツエスアイ）などがあります。

WG-S30:電子ノート（シャープ）
http://www.sharp.co.jp/enote/products/wgs30.html

WG-S30は、特殊な機能なしで、単にメモができるだけの電子ノートです（**pic.3**）。約3,000ページ分のノートと約4年分1,000ページのスケジュール帳を記録できます。6型、600×800ドット、消費電力が極端に少ないメモリ液晶が使われています。スマートフォンとの大きな違いは、一度のフル充電で30日間使い続けられることです。もちろんコンピュータへのデータ転送や、デジタルならではの手書きのUndo（やりなおし）も可能です。ほかにも同様の製品として、マメモ（キングジム）、電子メモパッド（サンワダイレクト）、ブギーボード（キングジム）、デジタルペーパーDPT-S1（SONY）があります。

デジタルデバイスの良さを取り入れたアナログ的文房具

Lockbook:紙のノートを指紋認証でガード
https://www.indiegogo.com/projects/lockbook-own-a-personalized-notebook#/

指紋センサー内蔵のバインダーで、紙のノートやメモ、付箋紙などをしまい、ロックすることができます（**pic.4**）。開閉には、スマートフォンのように指紋認証が必要です。スマートフォンの指紋認証と異なり、360度、どの方向から触っても有効です。カバーはカラフルな色使いのものが数種類用意されています。クラウドファンディングで進められているプロジェクトで、資金は予定の2倍集まり、2016年6月に最初の商品が出荷される予定です。

ポムリエ:スタンプメーカー（カシオ）
http://pomrie.casio.jp/

デジタルデータからオリジナルスタンプを作れる文房具（**pic.5**）。スマートフォンに対応したモデルと、パソコンのみに対応したモデルがあります。ゴム印だけでなく、布転写シートも用意されており、アイロンで衣服に貼り付けることもできます。スタンプのサイズは15×15mmから最大45×90mmまで。7種類用意されているサイズから選択できます。硬質面インクを用いると、金属やプラスチックなどの面にもスタンプできるそうです。

文房具とデジタルデバイスの連携で新しい価値を生み出しているもの

ピコットフセン:QR付箋紙（カンミ堂）
http://www.kanmido.co.jp/picotto.html

ピコットフセンはQRコードが記載された付箋（ふせん）紙で、スマートフォンの中にある写真／画像と、付箋紙を結び付けることができます。普通のノートにスマートフォンで撮影した写真や画像を貼り付けたいと思っても、一度印刷しないと貼り付けられませんが、ピコットフセンを用いると、QRコードとスマートフォン内の写真／画像の関係をつなぎ合わせて取り扱うことができます。

SMAFO BUNGU sheet:自動振り分けノート用シート（エレコム）
http://app.elecom.co.jp/smafobungu/

従来から、スマートフォンで撮影する専用のノートやメモ帳が各社からリリースされていました。このSMAFO BUNGU sheetは、ごく普通のノートを、スマートフォン取り込み対応にしてしまう、透明な下敷きです（**pic.6**）。四隅に専用のマーカーが描かれており、取り込み専用のノートでなくとも、指定した範囲をスマートフォンで正確に取り込み、スキャン、分類、整理をすることができます。もちろん専用のノートであれば、さらに手間なく取り込めます。付箋紙専用、スケジュール帳専用のアプリも用意されています。

スマ単:スマートフォン連携の単語帳（ペンテル）
http://ankibungu.jp/

スマ単は、手書きで作成する単語帳。自分用の単語帳を作る部分の良さと、どこでも平易に覚え、覚えたことを確認できるスマホアプリと連携した単語帳です。この単語帳に加えて「スマホで暗記」というマーカータイプの商品も用意されています。こちらは、こ


pic.1
T-EM100


pic.2
HiTAP


pic.3
WG-S30


pic.4
Lockbook

pic.5
ポムリエ


pic.6
SMAFO BUNGU sheet

のマーカーでなぞった部分がスマホアプリで見ると黒く見えなくなり、タップすると元の単語が見えるようになるという、赤いマーカーと緑のシートを使った自作単語帳のデジタル版です。単語に限らず、さまざまな暗記ものに利用できます。

未来の文房具

人間の脳、人間の手、人間の五感は長い年月を経ても、そう大きくは変化しません。今ある文房具の多くは、少しずつ形を変えつつも、古くから使い続けられているものです。一方、デジタルデバイスの性能は指数的なスピードで進化を遂げてきています。今後はデジタルデバイスによって、人間が助けられ、そのことによって人間の能力が拡張されていくような道具が現れてくるのでしょうか?

Android OSにより、世界がもっと便利になるアイデアを募集したプロジェクト「Android Experiments OBJECT」のグランプリ作品の1つに、「Magic Calendar」というオンラインのGoogleカレンダーを壁掛けで表示するアイデアがありました(https://www.android.com/object/vote/magic-calendar/)。現在は低反射ディスプレイを利用したデモですが、将来的には電子ペーパーの利用を考えているそうです。オンラインのカレンダーと紙(のように見える)カレンダーとの同期のような、今までは別々のもの、今までそういうことは無理だったと思われることも、技術の進歩で次々に実現してきています。

Photoshopなどのデジタルツールを提供するAdobeは、Adobe Ink & Slideという文房具的な補助ツールを提供しています。また、Microsoftは Surface Dialというダイアル式の入力デバイスの提供を始めました。

デジタルデバイスが、デジタルの都合だけで作られるのではなく、少しずつ人の手で使われる道具として、歩み寄る余裕がでてきたのかもしれませんね。SD

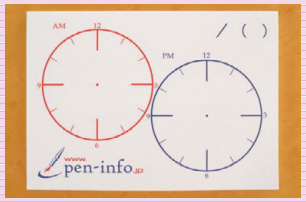
Gadget 1

時計式ToDo管理付箋

http://www.pen-info.jp/clock_top.html

円グラフで時間管理をする付箋紙

「時計式ToDo管理付箋」は、その名のとおり、ToDoの管理をアナログ時計を模した円グラフで行うための付箋紙です。仕事の種類にもよりますが、たいていの人は30分単位や1時間単位でスケジュールを管理し、1日の仕事の予定を立てたり、行き先、移動時間などを予定しておいたりするのではないでしょうか? 朝一番にその日の予定を再確認したり、予定が立て込んだ日の行動を整理したり、日報の報告用にその日を振り返ったりと、さまざまな用途に手軽に使える付箋紙です。



Gadget 2

SHOT DOGS

http://www.kingjim.co.jp/sp/shotdocs/

スキャンしやすいフォルダ

SHOT DOGS(ショットドックス)はバインダー内の書類がスキャンしやすく配慮された商品です。一般的な透明ビニールのスクラップブックに書類や名刺を入れたままで、必要な書類をそのままスマートフォンで撮影して取り込み、ゆがみや傾きが自動補正されてデジタルデータとして利用できます。同様に、見開きでスキャン撮影しやすいリングのバインダーや、撮影用の仕切りなど、さまざまな配慮がなされています。

Gadget 3

Neo smartpen

http://www.neosmartpen.com/jp/

ごく普通のスマートペン

Neo smartpen N2は、ごくごく普通の風貌と書き味のスマートペンです。重さは22g(キャップを含むと24g)、ペン芯はD1と呼ばれるリフィル(替芯)が利用できるため、気に入った書き味の芯を使えます。また、ペン以外、特殊なレシーバーを必要としないため、手軽に使うことができます。スマートフォンとの連携はBluetooth 4.0 LE(Low Energy)のため、バッテリーも節約できます。デメリットとして、Neo smartpenは専用ノートに描かれた特殊なパターンを読みとって描いた位置を認識するため、普通の紙ではなく、細かいパターンが描かれた専用のノート、専用の紙が必要なことです。

Gadget 4

カクトAR

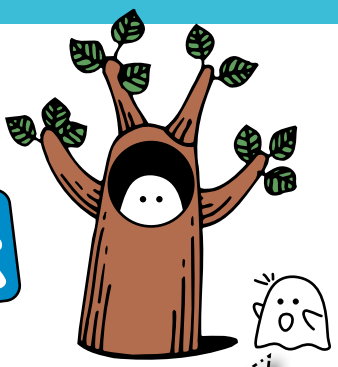
http://www.pentel.co.jp/products/degitalstationery/kakutoar/

手書きの味をARに

「カクトAR」は、ユーザが書いた文字や彩色した絵柄をAR(拡張現実)のキャラクタとして利用し、スマートフォンで楽しむことができるしくみです。パッケージに入っているカードで練習し、ダウンロードして選んだグリーティングカードに専用ペンで色を塗り、ポストに投函します。それをもらった相手は、カードにスマートフォンをかざすと、描いてもらったキャラクタがスマートフォンの画面の中で動き出します。グリーティングカードや、メッセージカードとして使うことができます。単なるカードで終わらずに、そのカードをきっかけにさらに楽しむ工夫がなされています。

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

結城浩の再発見の発想法 49



GC ―― ガベージコレクション

ガベージコレクションとは

ガベージコレクション(Garbage Collection)とは、**管理しているメモリ領域**(ヒープ)**のうち、使用しなくなったメモリを再利用可能にするしくみ**のことです。ガベージ(Garbage)は「ゴミ」、コレクション(Collection)は「集めること」ですので、ガベージコレクションを直訳すれば「ゴミ集め」という意味になります。ガベージコレクションを省略して**GC**と呼ぶこともよくあります。

プログラムは必要なメモリを確保して処理を進めます(**図1**)。やがて確保したメモリは不要になり、プログラムから参照されなくなります。GCでいうガベージとは「もうプログラムから参照されなくなったメモリ」のことです(**図2**)。

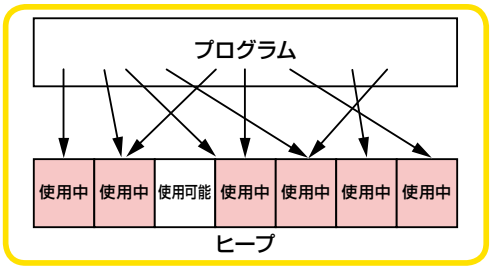
ガベージが多くなってくると、使用可能なメモリが減ってきます。そこでGCを行い、ガベージを使用可能なメモリとして管理しなおすのです(**図3**)。これによって使用可能なメモリが増えることになります。

GCは「ゴミ集め」という意味ですが、本来の目的は「ゴミ集め」ではありません。GCの本来の目的は、使わなくなったメモリの再利用、つまりメモリの「リサイクル」なのです。

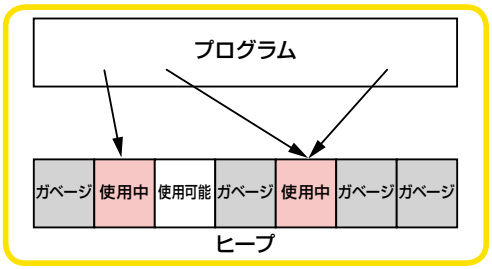
メモリのリサイクルを行うのは想像以上に難しいことです。それは、1つのメモリがあちこちから参照されている可能性があるからです。「このメモリはガベージである」という判断には、もはやどこからも**絶対に**参照されていないという保証がいるのです。

GCを行わず、プログラマがメモリを管理する方法もあります。もしもプログラマがメモリ

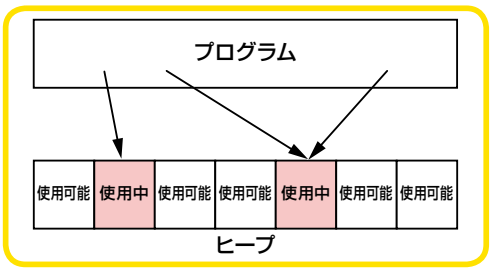
▼図1　プログラムはメモリを使用する



▼図2　参照されなくなったメモリはガベージになる



▼図3　GCが終わるとガベージは使用可能になる



の要不要を正しく判断できるならいいのですが、なかなかそれは難しいことです。判断に誤りがあるとメモリリークと呼ばれるバグとなり、思いがけない動作をする危険性があります(そしてこれはよくあるバグです)。これはしばしばセキュリティ上の問題を引き起こします。

GCを備えた言語処理系では、プログラマがメモリ管理を行う負担がなくなります。現代の多くの言語処理系、たとえばRuby、Python、Java、JavaScriptなどはすべてGCを備えていますので、メモリリークを気にする必要はありません。

GCのアルゴリズム

もっとも簡単なGCのアルゴリズムは**何もしない**ものです。メモリの再利用はまったく考えず、使い切ったらプログラムを終了するという乱暴な方法です。十分なメモリがあるなら、これでも実用上は問題ありません。

マーク＆スイープはGCの基本的なアルゴリズムです。プログラムを一時停止して、参照されているメモリのすべてにマークを付け、残りのメモリをすべてガベージだと判断します。そしてそのガベージを使用可能なメモリとして再利用するのです。

リファレンスカウントというアルゴリズムもあります。これはメモリに「現在何ヵ所から参照されているか」を表すカウンタを持つ方法です。そのメモリに対する新たな参照が生まれたらカウントアップし、参照がなくなったらカウントダウンします。こうすると、カウンタが0になった瞬間に、そのメモリがガベージになったことがわかります。

コピーGCというアルゴリズムもあります。このアルゴリズムでは、ヒープをA、Bふたつの部分に分け、その片方だけを使います。GCを行うときには、使っているメモリの内容をAからBにコピーし、リンクを付け替えます。現在使っているメモリをすべてBにコピーし終えたなら、Aに残っているメモリはガベージと判断できます。この方法は、使えるヒープが半分になるという欠点がある反面、ヒープの断片化を防ぐ効果があります。

日常生活とGC

「ガベージ」や「GC」はプログラマの軽口にもよく登場します。たとえば、**散らかった机を片づける**ときに「そろそろGCするか」と言いたくなるプログラマは少なくないでしょう。

筆者は、個人的に**仮眠で頭をスッキリさせる**のは「頭のGC」と感じます。

ここ十年くらい**断捨離**という言葉が流行していますが、これはGCに似ています。断・捨・離のうち、いらないものを捨てる「捨」はとくにGCそのものですね。

家の**大掃除で不要品を捨てる**のもGCです。GCの目的がメモリの再利用であるのと同じく、不要品を捨てて家の空間を再利用するからです。

しかし、GCにおいて「このメモリはガベージである」と判断するのが難しいのと同じように、大掃除で「これは不要品である」と判断するのは難しいことです。

プログラムの場合、ガベージだと判断する条件そのものは明確です。プログラムのどこからも参照されていないならばガベージだからです。難しいのは、**どこからも参照されていない**ことを調べるのにかかる手間です。マーク＆スイープでは、「時間を止め、メモリという空間をスキャンする」ことで判断を行います。

大掃除の場合、不要品だという判断はどうするでしょう。未来の生活を想像して「今後これは使わない」と言えれば不要品ですね。その判断は「未来という時間をスキャンする」ことで行います。確かにそれは難しい話ですね。

◆ ◆ ◆

あなたの周りを見回して、誰からも使われていないガベージはあるでしょうか。現在使っているものはどんなときにガベージになるでしょうか。

ぜひ、考えてみてください。SD

及川卓也の プロダクト開発の道しるべ

品質を高めるプロダクトマネージャーの仕事とは？

第8回

PMの用いるドキュメント

Author 及川 卓也（おいかわ たくや）
Twitter @takoratta

この連載の第3回（2017年1月号）でProduct Manager（以下PM）が使うべきドキュメントとして、PRD（Product Requirements Document）については解説しました。

PMの用いるドキュメントは記録よりも、それ自身がツールであり、フレームワークです。今回はPRD以外の、PMが使うドキュメントについて紹介します。

One Pagerの活用

前回、ドキュメントよりも動くソフトウェアを重視する[注1]アジャイル開発手法は、PRDに代表されるようなドキュメントをフレームワークとするPMと若干相性が悪いという話をし、その解決策として、One Pagerと言われるような1枚の紙に収まるレベルのシンプルなドキュメントを用いる方法をお伝えしました。

One Pagerとは

One Pagerは比較的最近用いられるようになった言葉なので、普通の英和辞典には出ていません。英英辞典でも出ているものは稀（まれ）でしょう。意味はとても簡単で、1ページの要約のことです。必要な概要をすべて1枚のドキュメントにまとめたものを、One Pagerと呼び、時間のない関係者や意思決定者に読んでもらうことを目的としています。

数十ページにものぼるドキュメントを用意しても、誰も読んでくれないということはよくありますが、そのような場合でもそのドキュメントを要約したOne Pagerを用意しておけば、そのくらいは読んでもらえる可能性があります。また、One Pagerにまとめてみたことによって、数十ページものドキュメントが不要だったということに気づくこともよくあることです。

One Pagerのルールは簡単です。1ページに収まること。これだけです。厳密に言うと、A4サイズに印刷して収まることや印刷されたものが十分読むに耐えるものである、つまりフォントを極小にするなどしなくても済むようになっていることなどもありますが、今回はそこまで厳密でなくても良いでしょう。

GitHub Issuesの利用

現在、筆者が勤務しているIncrementsでは、PRDも用いていますが、PRDを書くほどでもないような新機能や修正については、GitHub Issuesを用いて機能要件を起案するようにしています。

GitHub Issuesのテンプレートとして次のような項目を用意しています。

```
## What
<!-- どういう施策なのか、どういうタスクなのかを書く -->


### Goals
<!-- 何を達成したらこのissueはcloseできるのかを
```

続く

注1）アジャイルソフトウェア開発宣言（http://agilemanifesto.org/iso/ja/manifesto.html）を参照。

```
書く -->


### Non-Goals
<!-- このissueではやらないことを書く -->


## Why
<!-- なぜこれをするのかを書く -->


## Ref
<!-- 関連する issue, 発端になったコメント, 議論の参考になるページなどの URL を貼る -->


## Todo
<!-- 具体的なタスクが見えていればタスクリストを書く -->


<!-- Issue の種類(共有、質問、議論、依頼)に応じて、share question discussion request のいずれかの label をつける -->
```

最後のコメントにあるように、このテンプレートは新機能について起案すること以外にも使われる汎用テンプレートとなっているため、PRDの代わりとなるようなOne Pagerとして用いる場合は、もう少し機能を説明できるようなテンプレートにしたほうが良いかもしれません。

Amazonのプレスリリースから始めるという手法

筆者はエンジニアtypeという媒体で、PM探訪という各社のPMを訪ねてインタビューするという連載を持っています。その中でAmazonのPMを訪ねたことがある[注2]のですが、そこで伺ったAmazonのPMの製品の提案方法が非常にユニークだったことに驚きました。

Amazonでは製品を提案する場合、まずプレスリリースを書くことから始めるそうです。プレスリリースと言えば、製品が完成し、最後にマーケティングや広報などと用意するというのが一般的です。しかし、Amazonでは最初にプレスリリースを用意し、それに対しての承認を受けてからプロジェクトがスタートするというのです。

プレスリリースはまさにOne Pagerです。Amazonでは承認プロセス中の質疑でドキュメントに書かれていること以外にも答えることになるようですが、基本、どんなに高機能な製品であっても1枚のドキュメントでの説明が求められます。いわく、「自分の親が読んでもわかるように」というくらいのわかりやすさも求められます。

プレスリリースはまさに「何を、誰のために、なぜ提供しているのか」が書かれた、究極のOne Pager PRDと言えるでしょう。購買意欲の湧かないプレスリリースしか作れない製品は最初から失敗が見えているようなものです。

インセプションデッキ

このAmazonのプレスリリースと同じように、製品の出荷時の形態から製品のあるべき姿を考えるためのフレームワークがインセプションデッキです。

プロジェクトが終盤を迎えたころに、「えっ、今さらこんなこと聞くの？」と思うような質問がされたりした経験はないでしょうか。とっくの昔に決めて伝えていたつもりであったことが、ちゃんと伝わっていなかったり、微妙なところが食い違っていて、実は必ずしも同じことを考えているわけではなかったりします。このようなことがプロジェクトの終盤を迎えてから発覚すると、それこそ修羅場です。

このような状況が発生することを避けるためのツールとして、インセプションデッキがあります。インセプションデッキは『アジャイルサムライ —— 達人開発者への道』[注3]で紹介されているツールです。いくつかの大事な、でも若干言いにくい(書籍の中では「手ごわい質問」と説明されています)質問に答えることで、きちんと自分たちのプロジェクトを考えることができます。

注2) 「すべてプレスリリースから考えよ」アマゾンジャパンのPMに学ぶ仕事の流儀とキャリア展望【及川卓也のプロダクトマネジャー探訪】(http://type.jp/et/feature/2191)

注3) Jonathan Rasmusson 著、西村直人・角谷信太郎 監訳、近藤修平・角掛拓未 訳／オーム社 刊／ISBN978-4-274-06856-0

インセプションデッキは次のような10の質問と課題から構成されています。いずれもプロジェクトのキックオフ時に確認しておく必要があるものです。

1. 我われはなぜここにいるのか？
2. エレベーターピッチ
3. パッケージデザイン
4. やらないことリスト
5. 「ご近所さん」を探せ
6. 解決案を描く
7. 夜も眠れない問題
8. 期間を見極める
9. 何を諦めるのか
10. 何がどれだけ必要か

インセプションデッキはこれらの質問や課題に答える形で、1つにつき1枚のスライドを埋めていくというやり方をとります。そのため、デッキと呼ばれています。

それでは1つずつ見てみましょう。

我われはなぜここにいるのか?

プロジェクトチームはなぜ存在しているのか？つまり、誰に何を提供するためにこのメンバーは集まっているのかを確認します。プロジェクト(製品)の目的がこれにより明確になります。

プロジェクトをスタートしようとしているので、こんなことは自明と思われるかもしれませんが、ここには2つの視点を加えるようにしましょう。1つは顧客の視点です。実際に会社を出て、自分たちの顧客に会いに行きましょう。

もう1つの視点は会社の視点です。書籍では「司令官の意図」と書かれていますが、会社の事業として製品を開発するからには、どんなものを作っても良いわけではありません。それぞれの会社に用意されているビジョンやミッションに沿っているか、現在の会社の方向性とあっているかを考えるのを忘れてはいけません。

エレベーターピッチ

大事なことを極めて短い時間で伝えることをエレベーターピッチと言います。もともとは、ビルの1Fで自分の勤める会社の社長のようなお偉いさんとたまたま一緒になり、ビルの上階までエレベーターの中で2人きりの時間があるというような絶好のチャンスでのプレゼンテーション(ピッチ)が名前の由来です。エレベーターの中ですから、せいぜい30秒程度しか時間はありません。この時間の中で、自分のアイデアの本質を伝える必要があります。

インセプションデッキの中では、このために**図1**のようなテンプレートが用意されています。

例として、写真を撮影場所に紐付(ひもづ)け、撮影後に友人が同じ場所を訪れたときにその写真が共有されるという、SnapChat対抗の架空の製品のエレベーターピッチを考えてみました(**図2**)。

このように、テンプレートに従って、1枚に収まる2つの短い文という制約の中で書くことで、製品の本質を整理しなおすことが可能となります。

パッケージデザイン

パッケージデザインはまさにその名前のとおり、製品が箱入りで売られているとしたら、その箱のデザインはどうするかを考える作業です。実際には箱のデザインでなくても、広告やチラシでもかまいません。筆者の会社では、Webのランディングページを考えてみることもあります。

購買者の立場から見て、欲しい！と思わせるものになっているかを見つめ直すことができます。

すでにお気づきだと思いますが、1つ前のエレベーターピッチとこのパッケージデザインは、Amazonが行っている「プレスリリースから始める」と同じく、製品の最終形態を当初から検討するというアプローチです。

やらないことリスト

やらないことリストは実際には、やることとやらないことのリストです。やることは恐らく自明でしょうが、やらないことを決めておくことはとても大事です。あとになってから、「あれはやらないの？」と言われる可能性もありますし、やらないことを明確にしておくことで、製品が目指す方向をはっきりと共有できます。

「ご近所さん」を探せ

プロジェクトメンバーはすでに決まっていることは多いと思いますが、それ以外の関係者を洗い出しておきます。プロジェクト後半になって、どこからともなく現れた人がいろいろと口を出してくることがあります。とても面倒なことが起きるのはそういうときなのですが、そのような事態を事前に予測し、関係者となり得る人を最初から巻き込んでおくことが大事です。関係者は自分が思っているよりも多くいる可能性があります。一度関係しそうな人をすべてリストアップし、プロジェクトについて話しておくと良いでしょう。

残りの５つの質問と課題

ここまでで、プロジェクトが存在している理由が明確になったはずです。インセプションデッキの残りの5つの質問は、どのようにプロジェクトを進めていくかを考えるものとなります。

- **解決案を描く**：システムフローやアーキテクチャを描きます
- **夜も眠れない問題**：プロジェクトのリスクを書き出します。個人的には、これは後半の5つの中で最も重要と考えます。失敗するプロジェクトでありがちなのは、あとになってからメンバーが失敗するんじゃないかと思っていたとか、実は心配していたんですと言い出すような事態です。心配事はあらかじめ共有し、事前に備えるようにしましょう
- **期間を見極める**：ざっくりとしたレベルで良いのでスケジュール感を共有します
- **何を諦めるのか**：やらないことリストと似ていますが、こちらではスケジュールとコストなどトレードオフ対象になるものを含めて、諦めるものを明確にします
- **何がどれだけ必要か**：最後に、プロジェクトで必要になるリソースをまとめます

紙幅の関係で後半の5つの質問と課題については概要に留めました。詳しくは書籍を当たってみてください。

これ以外にも、リーンスタートアップで用いるリーンキャンバスがOne Pagerのドキュメントとしてよく用いられます。これについては、次回解説したいと思います。SD

▼図1　インセプションデッキのテンプレート。この緑色のプレースホルダの部分を埋めていく

- [潜在的なニーズを満たしたり、抱えている課題を解決したり]したい
- [対象顧客]向けの、
- [プロダクト名]というプロダクトは、
- [プロダクトのカテゴリ]である。
- これは[重要な利点、対価に見合う説得力のある理由]ができ、
- [代替手段の最右翼]とは違って、
- [差別化の決定的な特徴]が備わっている。

▼図2　Snapperという架空の製品のインセプションデッキ

- [写真を通じてコミュニケーション]したい
- [ティーンエージャー]向けの、
- [Snapper]というプロダクトは、
- [写真共有ソーシャルサービス]である。
- これは[友人が自分が撮影したのと同じ場所に来たときに、以前に自分が撮影した写真を共有すること]ができ、
- [SnapChat]とは違って、
- [時空を超えてコミュニケーションするという楽しみ]が備わっている。

Profile
ソフトウェアエンジニアとして社会人キャリアをスタートした後、MicrosoftやGoogleでプロダクトマネージャーやエンジニアリングマネージャーを経験。現在はプログラマーのための情報共有サービスQiitaのプロダクトマネージャーを務める。

第16回 サラリーマンになるな、エンジニアになれ！

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

GW明けまでOSCはお休みです

オープンソースカンファレンスは、ほぼ毎月のように開催されているため、この連載もOSC開催に連動して執筆しています。ただし、3月末から4月中は年度の切り替わりということもあって、OSCを開催することがありません。そこで今回は、秋葉原にあるよい感じの飲み屋さんを紹介します。しかし、ただ紹介するだけでは成立しないので、あれこれ悩んだあげく、本誌の編集長である池本公平氏をお誘いして、美味しいお酒を楽しみつつ、雑誌作りへの思いをお聞きしました（**写真1**）。

まず向かったのは、秋葉原駅から電気街に出て徒歩3分ほどにあるバー「Gauge（Report参照）」です。

SDでスキルを身につけてもらいたい

まず、一番気になるのはSDの編集方針です。雑誌の方針は時代の流れとともに変わっていくわけですが、現在の方針を聞くと「スキルを身につけることができる記事を掲載しています」とのこと。

確かにSDのキャッチフレーズは「OSとネットワーク，IT環境を支えるエンジニアの総合誌」ですが、単に新しい技術などを紹介する記事ではなく、実際に試せる、業務に取り入れられる実践的な内容が多いように感じます。

また、執筆者の顔ぶれを見ると、現場で実践的に何かをやられている方がほとんどです。「活きたスキル」に触れる機会を頻繁に作るというのは、Webの速報性と書籍の専門性の間にはまる形になる雑誌というメディアの性質なのでしょうね。

読者の年齢層が若返ってきている

SD編集部に配属された8年前、平均読者年齢が40〜50代になっていて、このままでは雑誌が続かないという危機感にさいなまれたということです。そこでシェルスクリプトやログ管理などの若返り企画を推進することで、ITの学習色を濃くしたのですが、結果として若い読者の呼び込みに成功して、今は20〜30代の読者が主となっているとのこと。とくにデータベースの記事は読者からの反応がよいようです。「SQLを身につければ、長く食べられるスキルになる」というお話は、私自身が新卒で日本オラクルに入社してデータベースやSQLを身につけ、いまだにそれで食べられていることからも、納得感があります。

技術者としての働き方

話題は技術者の働き方の話に。これは最近、私が感じていることですが、自分の得意なスキルを活かして活躍する「エンジニア」というよりも、仕事として言われたことだけをこなす「サラリーマン」な技術者が増えているような気がしています。池本さんからは、これまで見てきたさまざまなエンジニアのお話を伺えました。学生時代から頭角を現した人もいれば、地道な努力をしているうちに機会をとらえて世の中に出た人もいます。

共通しているのは自分の好きなことを一生懸命にやっているということ。そこがサラリーマン的な技術者との決定的な違いでしょうか。もちろん、誰もがすごいエンジニアになれるわけではありませんが、何かのきっかけでその人が大きく変わることを見てきているので、SDの記事がそのようなきっかけ作りになれば、という思いがあるのですね。今年の4月号では、新人向けの第1特集で

▼写真1　まずはカウンターで乾杯。私はいきなり日本酒から入ってます

▼写真2　お酒のボトルの上にヴィンテージの計測機器がずらりと。最高の肴ですね

▼写真3　テーブル席に移動して、議論白熱です。お酒も進みます（日本酒2杯目）

私も少し記事を書かせていただきましたが、あの記事をきっかけにエンジニアとして成長してくれたらうれしいですね。

今後の方向性は？

今後の誌面作りの方向性を伺ったところ、引き続き「OS、ネットワーク、インフラ」をしっかりと押さえていくとのことでした。現在、紙媒体の雑誌が少なくなっている中、インフラ系エンジニアが楽しく読める雑誌はSDしかありませんから、今の路線を踏襲してくれるのは私も賛成です。ただ、同じネタを2回も3回も取り上げると飽きられてしまうのが悩みの種とか。一方で、あまり奇をてらい過ぎたネタは万人向けではないですし、バランスが難しいところですね。私自身、この連載がエンジニアのみなさんの一服の清涼剤になればと思って、今後も頑張って連載を続けていこうかと思います。

このあたりでお店も混み合ってきたので河岸を変えて、昔は万世橋駅だった場所を改装した「マーチエキュート神田万世橋」にある「常陸野ブルーイング・ラボ」に移動（**写真4**）。さまざまなクラフトビールを堪能しました。ただ、今回はあえて会話を録音しなかったので、このあたりからは何を話したのか全然覚えていませんが、今後のSDに期待できる楽しいお話で盛り上がったことは間違いありません。SD

▼写真4　クラフトビールで2次会。キャッシュオンデリバリーなので1杯から楽しめます

Report

バー「Gauge（ゲージ）」

URL http://cn493.sakura.ne.jp/gauge/

秋葉原で買い物をしたあとに立ち寄りたい、大人のバーとしてご紹介したいのが「Gauge」です。Gaugeは、オシロスコープなどの計測機器を扱っている老舗「東洋計測器」が経営している計測器バーで、「計測器ランド」のお隣にあります。店内は真空管アンプのオーディオでBGMが流れる中、ヴィンテージの計測機器を眺めながら、美味しいお酒が楽しめます（写真2、3）。

とくに日本酒、焼酎の品揃えがすばらしく、ほかのお店ではなかなか飲めないような逸品が味わえるので、ついつい通ってしまいます。

銘木から切り出したテーブル、カウンターもすばらしく、カウンターで1人グラスを傾けながら、電気街巡りの疲れを癒すのもいいですね。

店内だけでなく、お店の外にもヴィンテージの計測機器が展示されています ▶

ツボイのなんでもネットにつなげちまえ道場

第24回 ESP32でAWS IoTボタンモドキを作ってツイートしてみる

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

おさらい

前回は、ESP32-DevKitC（ESP32開発ボード）とMongoose OSを使ってAWS IoTにメッセージを発行し、AWS IoTコンソールでこれを購読して発行を確認しました。AWS IoTでは、発行されたメッセージに対してルールを使い、ほかのサービスと連携をさせることもできます。たとえば、ESP32からAWS IoTにメッセージを送り、それをAWS Lambda（ラムダ）に送って何か処理をさせるといったことができます。今回は、自作AWS IoTボタンを作り、Tweet[注1]をしてみましょう。

しくみ

さてTweetをするといっても、AWS IoTから呼び出すAWS LambdaでTweetするコードを書くのは少々面倒そうです。筆者はそんなにAWS Lambdaに慣れているわけではないので、Tweetには以前この連載の11回目（2016年5月号）でも紹介したことのあるIFTTT（イフト）[注2]というサービスのMaker Webhooks[注3]とTwitter[注4]を使うことにします。全体の構成は、**図1**のようになります。

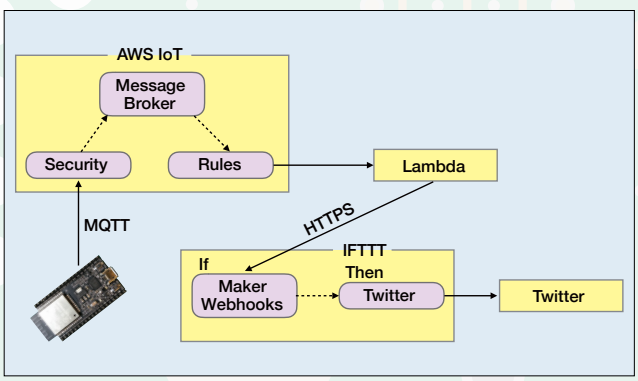
Lambdaで実行するプログラム（関数と呼ばれるみたいです）は、C#、Node.js、Java 8、Python 2.7などで記述ができるようです。Lambdaの開発環境を用意するのは面倒ですので、Googleで検索し、適当にシンプルでそのまま使えそうなコードを探してみました。すると、「AWS IOT Button + IFTTT's Maker Channel[注5]」という記事を見つけることができました（**図2**）。ここに掲載されているコードはシンプルで、何か変更をしたいときに簡単に変更すべき場所を見つけることができそうです。

ESP32で実行するコード

この記事を読むと、AWS IoTボタンは、「SINGLE」「DOUBLE」「LONG」の3種類のイベントを発行できるようです。それぞれのイベントが何を指すかは、イベント名から明らかですね。今回は、「SINGLE」だけを実装することにします。

ESP32開発ボードを接続した状態で、ターミナルで、

▼図1 全体の構成



注1） この記事を書いている時期、ちょうどMastodon（マストドン）が話題になっています。IFTTTのMaker Webhooksを使ってMastodonにポストをすることもできるみたいですが、今回はTwitterでやってみます。
注2） https://ifttt.com
注3） https://ifttt.com/maker_webhooks
注4） https://ifttt.com/twitter
注5） https://www.hackster.io/blankalpha/aws-iot-button-ifttt-s-maker-channel-476bd9

```
~/.mos/bin/mos
```

と入力すると、ブラウザが立ち上がり、Mongoose OSのインストーラの画面が表示されます。ここでページの右上に「advanced mode」というリンクがありますので、ここをクリックすると、「Device File Manager」が起動します。前回(2017年5月号)に記した「button_mqtt.js」を試していれば、init.jsにbutton_mqtt.jsのコードが記述されているはずです。のちほど設定するLambdaで実行するプログラムには、clickTypeをIFTTTのMaker Webhooksに渡すようにコードが書かれています。ですので、clickTypeを渡すよう、init.jsに記述しましょう。筆者は9行目を追記しました(**リスト1**)。

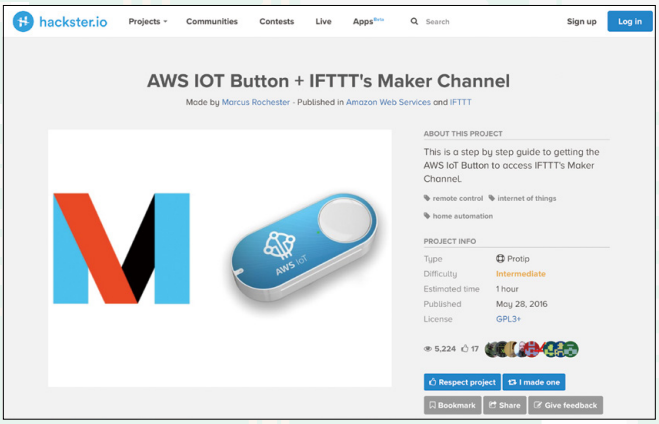
コードを書き換えたら、コード編集部分の上にあるオレンジの「Save and reboot device」をクリックし、ESP32開発ボードに変更したコードを書き込みましょう。

IFTTTの設定

まずはIFTTTにログインをしましょう。その後、Maker WebhooksとTwitterをConnectします(IFTTTでは、こういった連携機能を使えるようにすることを「Connect」と呼ぶようです)。Maker Webhooksを Connectするとき、「**https://maker.ifttt.com/use/XXXXXXXXXXXXXXX**」といった具合にキーが発行されます。この「**XXXXXXXXXXXXXXX**」の部分でIFTTTのMaker Webhooksは認証を行いますので、これを控えておいてください。

次に、Appletsの追加を行います。if thisの「this」をMaker Webhooksの Receive a web requestにし、Event Nameを「SINGLE」とします。then thatのthatは、今回はTwitterのPost a tweetとし、Tweet textの内容を適当に作ります。あと

▼図2　参考にした記事



▼リスト1　変更後のinit.jsの内容

```
 1  load('api_gpio.js');
 2  load('api_mqtt.js');
 3  load('api_sys.js');
 4
 5  let pin = 0;   // GPIO 0 is typically a 'Flash' button
 6  GPIO.set_button_handler(pin, GPIO.PULL_UP, GPIO.INT_EDGE_NEG, 50, function(x) {
 7    let topic = 'mOS/topic1';
 8    let message = JSON.stringify({
 9      clickType: "SINGLE",   ←この行を追加
10      total_ram: Sys.total_ram(),
11      free_ram: Sys.free_ram()
12    });
13    let ok = MQTT.pub(topic, message, message.length);
14    print('Published:', ok ? 'yes' : 'no', 'topic:', topic, 'message:', message);
15  }, true);
16
17  print('Flash button is configured on GPIO pin ', pin);
18  print('Press the flash button now!');
```

で編集できますので、とりあえずこのまま「Create action」をしても問題はないでしょう。なお、試しにこのTweetの内容欄に日本語を入力してみたところ、無事に日本語でTweetできました。

▼図3　トリガーの設定

Lambdaの設定

さて、AWSのアカウントは作成済みという前提になってしまいますが、次にLambdaに関数を作成します。Lambdaのコンソール[注6]にアクセスをし、「今すぐ始める」あるいは「Lambda関数の作成」というボタンをクリックします。すると「設計図の選択」という画面に遷移しますので「ブランク関数」を選択しました。次に「トリガーの設定」というステップに移るので、ここでLambdaのアイコンの左にあるブランク欄をク

注6）https://console.aws.amazon.com/lambda

▼リスト2　入力したNode.jsのコード

```
1  var https = require('https');
2
3  exports.handler = function(event, context) {
4
5      var body='';
6      var jsonObject = JSON.stringify(event);
7
8      // the post options
9      var optionspost = {
10         host: 'maker.ifttt.com',
11         path: '/trigger/' + event.clickType + '/with/key/[your private key here]',
12         method: 'POST',                                    ←ここを書き換える
13         headers: {
14             'Content-Type': 'application/json',
15         }
16     };
17
18     var reqPost = https.request(optionspost, function(res) {
19         console.log("statusCode: ", res.statusCode);
20         res.on('data', function (chunk) {
21             body += chunk;
22         });
23         context.succeed('Thanks Marcus');
24     });
25
26     reqPost.write(jsonObject);
27     reqPost.end();
28 };
```

リックし、「AWS IoT」を選択します(**図3**)。さらに、IoTタイプを「カスタムIoTルール」、ルール名には適当なものを記入(今回は「IFTTT_Maker_Channel」としました)し、「次へ」をクリックします。

最後に関数の設定という画面に遷移しますので、「名前」に「IFTTT_Maker_Channel」と記入し、ランタイムを「Node.js 4.3」にします。コードを記述するところに、先述の「AWS IOT Button + IFTTT's Maker Channel」という記事中のコードをコピー&ペーストします。最後に11行目の[your private key here]を、先ほどのIFTTTのMaker Webhooksを登録したときに得たキーに書き換えるのを忘れないでください(**リスト2**)。

「Lambda関数ハンドラおよびロール」の「既存のロール」は、「service-role/lambda_basic_exec」を選択しました。これで作成ボタンをクリックすると、Lambda関数「IFTTT_Maker_Channel」は正常に作成されましたが、トリガーの作成中に「エラーが発生しました」というエラーメッセージが表示されました。これはまだAWS IoTの設定を行っていないためですので、気にしなくてけっこうです。

AWS IoTの設定

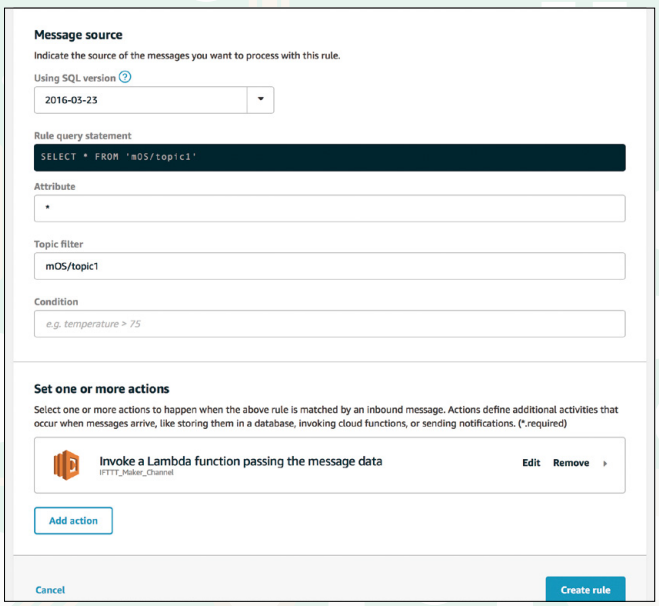
最後に、AWS IoTのコンソール[注7]にアクセスします。左ペインで、「Rules」を選び、「Create」ボタンをクリックします。すると「Create a rule」という画面に遷移しますので、ここでNameに「mOS_test」と入力し、Message Sourceにある「Attribute」に「*」、「Topic filer」に「mOS/topic1」を入力し、画面下部の「Add action」ボタンをクリックします(**図4**)。今回は先ほど作ったLambda関数を実行したいので、「Invoke a Lambda function passing the message data」を選択し、「Configure action」ボタンをクリックします。すると、Lambda関数を選択する画面になりますので、先ほど作った「IFTTT_Maker_Channel」を選択します。

これで、**図1**中の「Rules」が設定され、ESP32開発ボードからAWS IoTに発行されたメッセージのうち「mOS/topic1」というトピックを受け取ったときにLambdaの「IFTTT_Maker_Channel」関数を実行するように設定できました。

ボタンを押してみる

ここで、ESP32開発ボード上にあるBootと書かれたボタンを押してみましょう。数十秒～数分以内にTweetがなされるはずです。もしうまくいかない場合は、AWS IoTやLambdaのダッシュボード、あるいはIFTTTのActivityを確認すれば、どこの段階でうまくいっていないのかを確認できるはずです。今回は例としてTwitterを使いましたが、IFTTTのthen thatのthatを変更することで、ボタンを押したときに起こすことを手軽に変更できます。SD

▼**図4　Message Sourceの設定**



注7)　https://console.aws.amazon.com/iot

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**2017年6月15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

完全栄養食「COMP」 4名

人間に必要な六大栄養素（糖質・タンパク質・脂質・ビタミン・ミネラル・食物繊維）が理想的に配合された「完全栄養食」です。顆粒タイプで、水またはお好みの飲料とともにシェーカーなどに注ぎ、よく振ってから飲みます。味はマイルドな豆乳味で、コーヒーやジュースとの相性は抜群。今回プレゼントするのは、400kcal（90g）の使い切りサイズが計12袋入ったセットです。

提供元 コンプ http://comp.jp

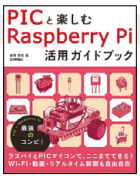
02

コンパクトVRグラス 3名

クリップでスマートフォンに挟むだけで、VR動画を手軽に楽しめるVRグラスです。ゴーグルタイプではないので、メガネをかけたままでも使用でき、そのままスマートフォンも操作できます。4インチから6インチまでのスマートフォンに対応。

提供元 フォースメディア https://www.forcemedia.co.jp

03



『PICと楽しむRaspberry Pi活用ガイドブック』＋ボード 1名

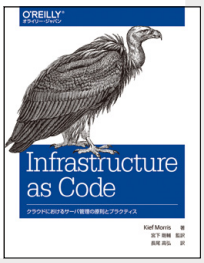
PICマイコンとRaspberry Piを使ったガジェット作りの解説書と、マイコン・ドップラセンサー・LED出力搭載のボードをプレゼント。Raspberry Pi（**※付属しません**）と合体させて、人が近づいたときに時刻を読み上げる「おしゃべり時計」が作れます。

提供元 ビット・トレード・ワン http://bit-trade-one.co.jp
提供元 技術評論社 http://gihyo.jp

04

Infrastructure as Code

Kief Morris 著



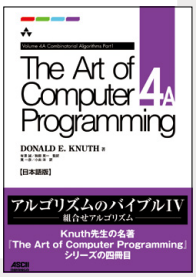
「インフラをコードで管理する」新しい開発手法について解説する本です。特定のプラットフォーム、ツールの説明よりも、それらの背景にあるコンセプトや考え方の説明に重点が置かれています。

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

05

The Art of Computer Programming Volume 4A

Donald E. Knuth 著



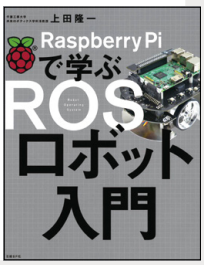
クヌース先生によるアルゴリズムの名著シリーズの復刻版です。分冊となる4巻のAでは論理代数の使い方や、グラフ理論、バックトラッキングなどの組み合わせの分野におけるアルゴリズムを取り上げます。

提供元 アスキードワンゴ http://asciidwango.jp 2名

06

Raspberry Piで学ぶROSロボット入門

上田 隆一 著



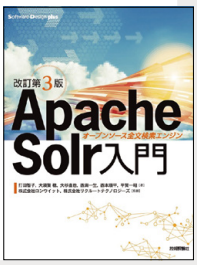
ロボット開発用ソフトウェアフレームワーク「ROS」とRaspberry Piを搭載した車輪型ロボット「Raspberry Pi Mouse」を使って、ロボットプログラミングについて解説する1冊です。

提供元 日経BP社 http://corporate.nikkeibp.co.jp 2名

07

[改訂第3版]Apache Solr入門

打田 智子 ほか 著



オープンソースの検索エンジンソフトウェア「Apache Solr」の入門書。Solrの基本、ドキュメント検索、インデクシング、クラスタなどのしくみを紹介し、具体的なプログラミング手法を解説しています。

提供元 技術評論社 http://gihyo.jp 2名

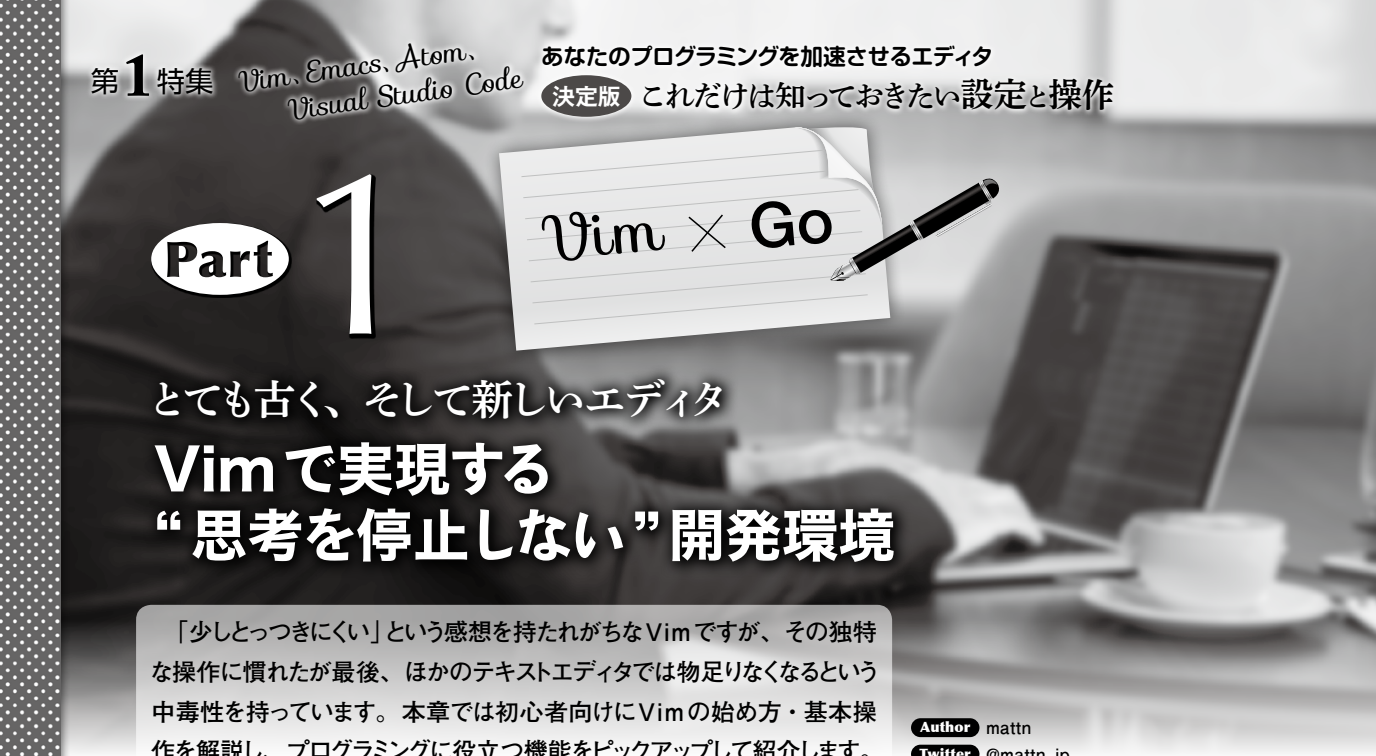
第1特集

Vim、Emacs、Atom、Visual Studio Code

あなたのプログラミングを加速させるエディタ

決定版 これだけは知っておきたい設定と操作

プログラミング環境と言えば、EclipseやVisual Studioといったいわゆる統合開発環境を利用している読者の方が多いと思われます。ですが、動作が軽く、柔軟にカスタマイズできるテキストエディタも負けてはいません。本特集ではVim、Emacs、Atom、Visual Studio Codeを取り上げ、インストール、基本操作、設定方法を解説します。また、各章では執筆者が普段使っているプログラミング言語を取り上げ、コーディングを加速させるカスタマイズ例も紹介。自分に合ったエディタを見つけ、記事を参考に自分好みにカスタマイズしてみてください。



Part 1

とても古く、そして新しいエディタ

Vimで実現する“思考を停止しない”開発環境

「少しとっつきにくい」という感想を持たれがちなVimですが、その独特な操作に慣れたが最後、ほかのテキストエディタでは物足りなくなるという中毒性を持っています。本章では初心者向けにVimの始め方・基本操作を解説し、プログラミングに役立つ機能をピックアップして紹介します。

Author mattn
Twitter @mattn_jp

Vimってどんなエディタ?

今年新入社員となられたみなさんは、Vimというテキストエディタをご存じでしょうか。もし知っておられたなら、viまたはVimにどんなイメージをお持ちでしょうか。「古いテキストエディタ」でしょうか。はたまた「操作が難しいテキストエディタ」でしょうか。

確かにVimは、1976年にBill Joy氏が開発を始めたテキストエディタviをベースに、今から20年以上も前に作り始められ今もなお開発され続けている、とても歴史のあるテキストエディタです。そしてその特異な操作感から、

簡単に終了できないテキストエディタ

の代名詞として扱われてきました。viが開発された当時のUNIXと言えばキャラクタ端末しか存在せず、その端末で動作するスクリーンエディタといえばviかEmacs、もしくはその派生物くらいしかありませんでした。

当然ながら、UNIXで作業するためにはviやEmacsを使えることが必須のスキルでした。Vimが誕生した20年ほど昔は、新人教育の一環としてまずviを教えるのが当然でした。20年も前ではありませんが、実際に筆者も新入社員として新人教育を受けた際は、上司からviを教えてもらいました。もしかすると、今でも新人教育のカリキュラムとしてvi/Vimを教えている会社があるかもしれません。それくらい、vi/VimはUNIXとは切っても切り離せないテキストエディタなのです。

今もなお使い続けられるVim

基本的に、Vimはviと同じ動作をするように設計されたテキストエディタですが、次のようにviに対して多くの機能拡張が行われています。

- カラフルなシンタックスハイライト
- 無限undo/redo
- ウィンドウ分割
- 入力補完
- GUI
- スクリプト言語Vim scriptを使った拡張

そのほか、開発を行ううえで必要と思われる多くの機能が実装されています。ただし、これらは近代的なテキストエディタとしては当然の機能かもしれませんね。しかしながら2017年になった今もなお、この古いテキストエディタ

が使い続けられているのは、いったいなぜなのでしょうか。もっときれいなGUIで、もっとユーザフレンドリな操作性を持ったテキストエディタはたくさんあります。それは、viやVimの操作がほかのテキストエディタと大きく異なることに起因しています。

たとえば、みなさんはプログラミングを行うとき、プログラムコードの先頭の行から最後の行に向かって、一度も後戻りすることなくタイピングしておられるでしょうか？　簡単なコードであればそれも可能かもしれません。ですがほとんどの場合、カーソルは縦横無尽に動いているはずです。「今書いている処理が長くなってきたから関数として切り出そう」「ライブラリを使いたいのでファイルの先頭にimport文を書こう」。そう考えながらカーソルを行ったり来たりしていると思います。Vimはこういった操作に最適化されたテキストエディタなのです。

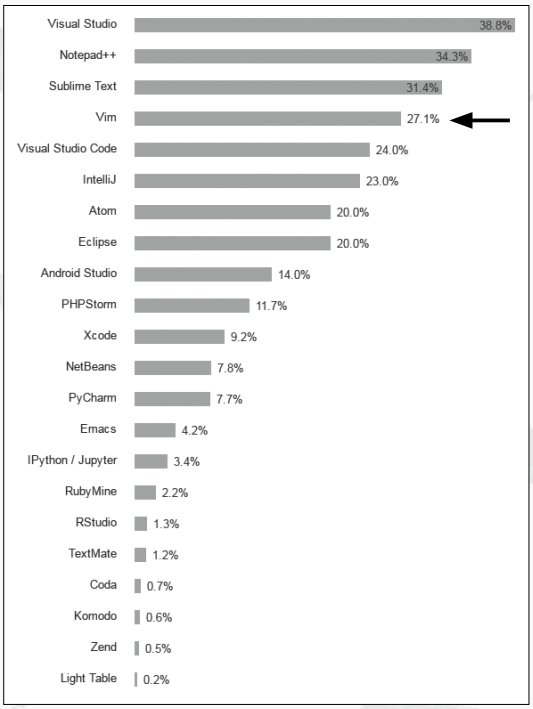
「ある行をコピーして9行貼り付け、行の頭に連番を付与したい」。外部のツールに頼らず、標準の機能だけでこういった操作が実現できてしまうテキストエディタは、筆者の知る限りそれほど多くはありません。

実際のところVimって流行っているの？

今年3月、開発者の技術的なQ&Aサイトを運営しているStackoverflowは、人気のあるツールや言語、開発環境などのランキングを公開しました[注1]。そのランキングのカテゴリの1つ「Web開発環境」は**図1**のようになっています。

テキストエディタであるVimが開発環境のランキングにおいて上位にランキングしているという、とても興味深い結果になっています。さらに、同ランキングの「システム管理者が使うエディタ」のカテゴリでは、Vimは堂々の1位にランキングしています。

▼図1　Web開発環境のランキング



開発環境というと、MicrosoftのVisual Studioのような、メニューにたくさんの機能が盛り込まれているIDE(Integrated Development Environment)を想像される方が多いと思います。そんな中、この古いテキストエディタVimが今もなお現役で、かつ多くのユーザから愛されているのです。

GitHub上でもあらゆるVimプラグインが活発に開発されています。新しいプログラミング言語の開発コミュニティが、まずVimのシンタックスハイライト機能を追加するといったことも珍しくはありません。それくらい、Vimを取り巻く技術は新しいのです。

前述のように、Vimはほかのテキストエディタに比べて操作方法が大きく異なります。使いこなせずに挫折する方もいます。新入社員のみなさんがいきなり挫折してしまわないよう、本章では導入手順から順を追って説明します。

基本編

まずはこの、古く新しいテキストエディタを

注1）URL https://stackoverflow.com/insights/survey/2017

インストールして実際に使ってみましょう。

Vimのインストール方法

Vimのインストール方法はだいたいのOSですでに確立しており、簡単にVimをインストールできるようになっています。

Linuxでのインストール方法

Linuxの汎用ディストリビューションであれば、標準構成でVimはインストール済みです。ディストリビューションによっては機能縮小版が入っていることもあります。Ubuntuであれば最小構成版のvim-tiny、GUIを含まない標準版のvim-nox、GUI付きのvim-gtkやvim-gtk3に加え、Python拡張が有効なvim-gtk3-py2などがインストールできます。

Ubuntuの場合、次のコマンドでPython2拡張が有効なGUI（gtk3）版のVimをインストールできます。

```
$ apt install vim-gtk3-py2
```

macOSでのインストール方法

macOSでも、標準でCUI版のVimがインストールされています。ただし執筆時点（2017年4月）ではバージョンが古く、拡張機能も含んでいないVim 7.3がインストールされています。なるべく新しいバージョンのVimを使いたい場合には、Homebrewを使ってインストールするのが一般的です。

```
$ brew install vim --with-python3
```

また、これとは別にMacVim[注2]というCarbon UIに移植されたVimもGitHub上で開発されており、インストールすることができます。

さらに、このMacVimに対していろいろな機能拡張を行ったMacVim-KaoriYaというバージョンが存在します。こちらはsplhack氏がこまめにメンテしているVimディストリビューションで、ほぼオフィシャル版に追従していると考えて良いでしょう。Vimのコミュニティサイトvim-jp[注3]にあるダウンロードのリンクから最新版がダウンロードできます。MacVimとMacVim-KaoriYaの違いはGitHubのページ[注4]で確認できます。

Windowsでのインストール方法

Windows版のVimも、vim-jpのダウンロードリンクからKaoriYa最新版がダウンロード可能になっています。またKaoriYaのサイト[注5]ではnetupvimというVim専用の更新ツールも配布されており、KaoriYa版そのものやvim-devから配布しているVimのスナップショット版を自動で更新できるようになっています。

Vimの基本操作

Vimの基本操作として、次の4つのモードを行き来することで編集を行います。

- インサートモード
- ノーマルモード
- コマンドモード
- ビジュアルモード

インサートモードは、一般的なテキストエディタが持ち合わせている編集操作と変わりません。もちろんVimでも、ほかのエディタと同様にカーソルキーで上下左右に移動したり、Home や End で行頭や行末へ移動したり、Ctrl-Home や Ctrl-End でファイルの先頭や末尾に移動したりできます[注6]。

しかしながら、Vimではこれらのカーソルキーや Home ／ End は使用しません。「カーソルキーを使うな」ということではありませんが、

注2） URL https://github.com/macvim-dev/macvim

注3） URL http://vim-jp.org

注4） URL https://github.com/splhack/macvim-kaoriya/wiki/DiffMacVimVsMacVimKaoriYa

注5） URL https://www.kaoriya.net

注6） Ctrl-Home は、Ctrl を押しながら Home を押す操作。

カーソルキーを使わない方がVimの上達には良いとされています。上達のために、Vim起動中はカーソルキーを無効に設定しているユーザもいるくらいです。

ではどうやってカーソルを移動するか。そのためにノーマルモードを使うのです。Vimは通常、起動直後はこのノーマルモードになっています。インサートモードからは Esc をタイプすることでノーマルモードに移ります。ノーマルモードでは、テキストのコピー（ヤンク）やペーストだけでなく、目的の場所へのカーソル移動、文字の置換や削除を行います。

コマンドモードとビジュアルモードについては後述しますので、まずはこのカーソル移動からみていきましょう。

カーソルの移動

カーソルの移動は**h**（左）、**j**（下）、**k**（上）、**l**（右）の4つを使用します。行頭へは**0**、行末には**$**を使用します。空白を除いた行頭には**^**で移動できます。ファイルの先頭行には**1G**もしくは**gg**で、末尾行には**G**で移動できます。ファイルの中ほどに移動したいのであれば**50%**とタイプします。

ノーマルモードでの操作は、次の3つの要素で構成されます。

・カウント
・オペレータ
・モーション

たとえばカーソルを3行下に移動するのであれば、**3**（カウント）と**j**（モーション）をタイプします。

また、たとえばfoo bar baz（カーソルはfの位置）というテキストで**d**（オペレータ）と**2**（カウント）と**w**（モーション）をタイプすると、カーソル位置から単語（**w**）が2つ削除（**d**）され、bazとなります。

一見とっつきにくそうに見えますが、この操作に一度慣れてしまうと、ほかのテキストエディタが触れなくなってしまうほどに手に馴染んでしまいます。ただ弊害として、「ほかのテキストエディタにjjjjなどと挿入してしまう」というジョークも聞きます。ジョークか実話かは、ご自身で体験してみてください。

検索

Vimでの検索には正規表現を使います。ノーマルモードからは**/**をタイプして検索を開始します。行頭のwordという語を検索したいのであれば**/^word**と入力します。fooもしくはbar_bazのどちらかを検索するのであれば**/\(foo\|bar_baz\)**で検索できます。

この正規表現の詳細については本誌の連載「Vimの細道」でも何度か登場してきましたので、細かい説明は割愛します。なお、もしカーソル上の単語を検索語として検索したいのであれば、単に*****をタイプしても良いでしょう。

さらに、単語の境界を意識して検索する場合は、パターンの前後を**\<**と**\>**で囲います。たとえば、**/\<\(foo\|bar_baz\)\>**とすると、**foobar**や**bar_baz_boo**にはマッチしなくなります。

Vimの正規表現の詳細が知りたい方は、ノーマルモードから**:help /pattern**と打ってヘルプを閲覧すると良いでしょう。ただしこのヘルプは英語ですので、いくぶんハードルが高いかもしれません。vim-jpでは、有志によって翻訳された日本語マニュアルを配布しています[注7]。後述するプラグインの導入方法でインストールすると、**:help**で表示されるヘルプが日本語になります。

書き換え

プログラミングでは文字の書き換えを頻繁に行います。一度書いた変数名が気に入らなくて書き換えたいならば、変数名の先頭にカーソルを移動し**cw**をタイプして書き換えます。

注7）URL https://github.com/vim-jp/vimdoc-ja

置換

置換はコマンドモードでsコマンドを実行します。カーソルがある行の中でsharpもしくはflatをroundに書き換えるのであれば、ノーマルモードから次を実行します（先頭の:によってコマンドモードに移ります）。

```
:s/\<\(sharp\|flat\)\>/round/
```

カーソルがある行の中で何度もsharpもしくはflatが現れるのであれば、最後にgフラグを付け足します。

```
:s/\<\(sharp\|flat\)\>/round/g
```

ファイルの中で何度もこの単語が現れ、それを一括で置換したいのであればsコマンドの前に%を付け足します。%は全行を表します。

```
:%s/\<\(sharp\|flat\)\>/round/g
```

これらを自分でコントロールするのも、Vim独特の操作と言えるかもしれません。なお、このパターンでは\(\)を使ってグルーピングしているので、置換文字列からこのグループ（サブマッチ）の単語を\1で参照できます。

```
:%s/\<\(sharp\|flat\)\>/目の付け所が \1 だね/g
```

これを実行すると、テキスト中のsharpもしくはflatが「目の付け所がsharpだね」や「目の付け所がflatだね」に置換されます。

これらの操作を、頭で考えなくても行えるようになるまでには、結構な時間が必要になるかもしれません。ぜひ根気よく気長に練習を続けてください。

応用編

ここからは、プログラミングに効く応用的なテクニックを紹介していきます。

タグジャンプ（難易度：低）

プログラミングをしていると、関数の定義位置にジャンプしたい場面も多いと思います。Vimにはタグジャンプという機能があり、タグファイルを使って関数や変数の宣言位置に簡単にジャンプできます。タグジャンプ機能を使う

COLUMN

ドットリピート

「検索」と「書き換え」にちなんで説明しておきたいのが、「ドットリピート」というVimの特徴的な編集方法です。前述のとおりfooまたはbar_bazにマッチする検索を行い、変数名の先頭でcwをタイプして書き換えを行ったのであれば、以降に現れる単語fooもしくはbar_bazの書き換えは、n（次の変数名の検索）と.（単語を書き換え）の2つのキーを繰り返し押すだけで済むようになります（例：n.n.n.）。Vimは最後に行った操作のカウント、モーション、オペレータを保持しており、.をタイプすることで再生できるようになっています。

言うのは簡単なのですが「この操作を行うと次にどうなるのか」があらかじめ分かっていないと、その場面に直面しても操作が思いつきません。これがVimの操作は難しい」と思われる一因でもあり、また「一度Vimに慣れると離れられない」と言われる理由でもあります。

この操作を覚えてしまうと、ほかの新入社員に対して大きなアドバンテージが生まれることは間違いないでしょう。テキパキとテキストを編集する姿を見て、うらやましいと思う人も出てくるかもしれませんね。

筆者のポリシーに、「プログラミング中は思考を止めるべきではない」というものがあり、テキストエディタやツール類は考えなくても使えるものを選ぶべきだと思っています。筆者が作るツール類も、なるべく考えなくても使えるように心がけています。

には、あらかじめctagsコマンドでtagsファイルを生成しておく必要があります[注8]。

タグジャンプは、ノーマルモードで関数名の上にカーソルを移動し、Ctrl-]をタイプします。OSごとにソースが分かれている場合には、関数宣言が2つ存在する場合もあります。その場合はg Ctrl-][注9]をタイプします。コマンドを打つと、ジャンプする候補が表示されます（図2）。何度かジャンプしたあとで元のタグジャンプ位置に戻るにはCtrl-tを使います。

▼図2　ジャンプ先の候補が表示される

```
    }
    mch_memmove(s, p, (size_t)line_len);

    /* Replace the line (unless undo fails). */
    if (!(flags & SIN_UNDO) || u_savesub(curwin->w_cursor.lnum) == OK)
    {
        ml_replace(curwin->w_cursor.lnum, newline, FALSE);
misc1.c                                                       303,5
  # pri kind tag               ファイル
  1 F    f    mch_memmove       misc2.c
                mch_memmove(void *src_arg, void *dst_arg, size_t len)
  2 F    d    mch_memmove       os_unix.h
                #   define mch_memmove(
  3 F    d    mch_memmove       os_unix.h
                #  define mch_memmove(
  4 F    d    mch_memmove       os_unix.h
                # define mch_memmove(
  5 F    d    mch_memmove       vim.h
                #  define mch_memmove(
番号と<Enter>を入力してください（空でキャンセル）:
```

連番の作成（難易度：中）

プログラミングにおいては、コードを書くことだけが作業ではありません。テストデータを用意するのも大事な作業です。

```
Tokyo
Osaka
Nagoya
Sapporo
```

このテキストの行頭に連番を振りたいと思った場合、一般的なテキストエディタでは難しいかと思います。Vimでこれをやるには、まず行頭に数字を挿入します。このファイル全体がテストデータなのであれば、**:%s/^/0,/**を実行すれば良いでしょう。またファイルの一部がこのようなデータなのであれば、Tokyoの行で**V**をタイプしてビジュアルモードに移ります。そのままSapporoの行まで移動し、そこで**:s/^/0,/**を実行すれば良いです。

```
0,Tokyo
0,Osaka
0,Nagoya
0,Sapporo
```

次に、**gv**をタイプすることで前回ビジュアルモードで選択した領域を再選択したあと、g Ctrl-aをタイプすれば、次のような連番付きのデータができあがります。

```
1,Tokyo
2,Osaka
3,Nagoya
4,Sapporo
```

Vimでは、数字の上でCtrl-a／Ctrl-xをタイプすると、その数字を増加／減少させることができます。これに**g**を付けると、ビジュアル選択した領域で連続的に数字が増減するようになります。

テキストオブジェクト（難易度：高）

プログラミングでは、"do something"の" "で囲まれた文字列の中身だけをコピーしたいとか、()で囲まれた関数の引数だけをコピーしたいといった場面が出てきます。そういった際に便利なのがテキストオブジェクトです。

前述のように、変数名を書き換えたい場合は変数名の先頭で**cw**をタイプしますが、""で囲まれた文字列の中身を書き換える場合は**ci"**をタイプします。同様に、()で囲まれた引数部分のみをコピーするのであれば**yi(**をタイプします。

コラム「ドットリピート」でも書きましたが、Vimは、最後に実行した操作のカウント、モーション、オペレータを保持していますが、この**"**や**(**もモーションの対象となります。したがって一度"foo"というテキストを**ci"zoo**で書き換えたあと、"bar"の上で**.**をタイプしても、ちゃんと"zoo"に書き換わってくれるのです。

注8）参考「ctagsと連携するように環境を構築する」
URL http://qiita.com/soramugi/items/f918020c2b3f48c93bf3
注9）gを押してから、Ctrlと**[**を同時押し。

プラグインマネージャで自在に機能追加

Vimでは、vimrc[注10]というファイルにVim scriptで設定を書くことで機能を拡張できます。

加えて、「Vimプラグイン」と呼ばれる拡張機能を使ってVim本体に新しい機能を追加できるようになっています。このVimプラグインを追加サポートするのがプラグインマネージャです。プラグインマネージャの実装にはいくつかあり、**表1**のものが有名です。筆者はvim-plugを使用しています。vim-plugはインストールやアップデートを並列実行できるようになっており、コマンド1つで高速にプラグインを更新できます。

実践編

ここからは、実際のプログラミング言語「Go言語」をVimでコーディングする方法を紹介していきます。Go言語のコンパイラは、サイト[注11]からインストールしておいてください。

VimでGo言語を編集するのであれば、vim-goというプラグインが必須アイテムと言って良いでしょう。vim-goが提供する機能は**表2**のとおりです。このほかにも、Go言語を編集するうえでとても便利な機能がたくさん盛り込まれています。

vim-goのインストールと初期設定

vim-goをインストールするにはプラグインマネージャを使うと便利です。本章では以降で記載するプラグインのインストールはvim-plugを前提に説明します。vimrcに`:Plug`コマンドを記述したあと、`:PlugInstall`を実行してください。次は、vim-goをインストールする場合の設定です。

```
Plug 'fatih/vim-go', { 'for': 'go' }
```

Vimを再起動して`:PlugInstall`を実行すると、vim-goがインストールされます。

vim-goを動作させるためにはいくつかのサードパーティ製ツールを必要としますが、次のコマンドにより、一括でインストールされます。

```
:GoInstallBinaries
```

しばらくすると**表2**のようなコマンドが使えるようになります。そのほかのプラグインマネージャでのインストール方法や、vim-goの詳細についてはREADME.md[注12]を参照してください。

vim-goには、好みに合わせて動作を細かく変更できるオプションが用意されています。**リスト1**は筆者が使用している設定です。

注10）配置場所は`:echo $VIM`で確認できる。

▼表1 おもなプラグインマネージャ

名前	URL
Vundle	https://github.com/VundleVim/Vundle.vim
vim-plug	https://github.com/junegunn/vim-plug
dein.vim	https://github.com/Shougo/dein.vim

▼表2 vim-goが提供するおもな機能

機能	コマンド
コマンドインテグレーション	:GoBuild、:GoInstall、:GoTest
カバレッジ表示	:GoCoverage
関数定義位置へのジャンプ	:GoDef
宣言位置や参照個所へのジャンプ	:GoDecls、:GoDeclsDir
編集中ファイルの実行	:GoRun
各種lintツールを使った静的解析	:GoLint、:GoMetaLinter
関数や変数のリネーム	:GoRename
playgroundでの実行	:GoPlay
import文の追加	:GoImport
structタグの追加	:GoAddTags

注11）URL https://golang.org/dl/
注12）URL https://github.com/fatih/vim-go

ひな形の生成

アプリケーションを開発し始めたいと思ったときに、ゼロから書き始めるよりもある程度できあがったひな形があると便利です。筆者が開発しているsonictemplate-vimをインストールしておくと、用途に合わせたひな形を簡単に呼び出せるようになります。

```
Plug 'mattn/sonictemplate-vim'
```

拡張子が.goのまっさらなファイルを開き、**:Template web-standard**を実行すると**リスト2**のソースコードが展開されます。展開後は、**:Template**コマンドを使ってハンドラや静的ファイルのサーブ処理を簡単に生成できるようになっていますが、その際にWebに特化した候補が優先的に表示されるようになります。

好みによってはUltiSnipsもインストールしておくと便利です。このプラグインよって、errlという省略語から次のエラー処理が展開されるようになるため、より便利になります。

```
if err != nil {
  log.Fatal(err)
}
```

UltiSnipsのインストールは次の設定で行います。

```
Plug 'SirVer/ultisnips'
Plug 'honza/vim-snippets'
```

UltiSnipsではデフォルトでTabを展開キーとして扱います。筆者の場合、Tabは通常どおり使いたかったので、**リスト3**のように他のキーを設定しています。こうしておくとCtrl-y Ctrl-u[注13]で展開されるようになります。お好みに合わせて変更してください。

さて、これでGo言語をコーディングする準備が整いました。以降では筆者が常時使用しているGo言語の開発手順を紹介していきます。

importの追加と削除

Go言語では、パッケージを使うためにimport文を追加する必要があります。毎回ファ

注13) Ctrlを押しながらyu。

▼リスト1　vim-goの設定例

```
" :GoBuild 時にエラーのある行にジャンプしない
let g:go_jump_to_error = 0
" ファイル保存時に実行されるコード整形で使用するツール名
let g:go_fmt_command = 'goimports'
" キーワードコマンドとして godoc を使わない
let g:go_doc_keywordprg_enabled = 0
" タグジャンプのキーを godef の動作で置き換える
let g:go_def_mapping_enabled = 1
" ハイライトを有効にする
let g:go_highlight_operators = 1
let g:go_highlight_functions = 1
let g:go_highlight_methods = 1
let g:go_highlight_types = 1
let g:go_highlight_build_constraints = 1
let g:go_highlight_string_spellcheck = 1
" 新規ファイルでテンプレートを適用しない
let go_template_autocreate = 0
" vimproc を使わず job を使う
let g:go#use_vimproc = 0
```

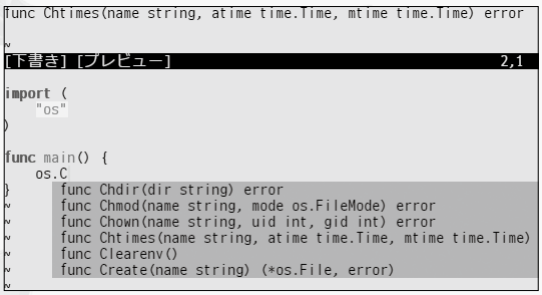
▼リスト2　web-standardのひな形

```
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter,↗
 r *http.Request) {
    fmt.Fprintf(w, "")
  })
    http.ListenAndServe(":8080", nil)
}
```

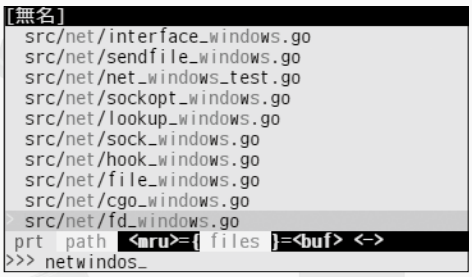
▼リスト3　UltiSnipsの展開キーマップを変更

```
let g:UltiSnipsExpandTrigger = '<c-y><c-u>'
let g:UltiSnipsListSnippets = '<c-y><c-i>'
```

▼図3 gocodeによる補完の例



▼図4 CtrlP実行時の画面



イルの先頭に移動してimport文を変更するのは面倒です。たとえば次のコマンドで、import文の追加を行えます。

```
:GoImport github.com/mattn/go-runewidth
```

また、追加したパッケージを削除するには**:GoDrop**を使います。

入力補完

vim-goではgocodeというコード補完ツールによって、Go言語のいろいろなコード補完が行えるようになっています。たとえばos.Createを補完するには、os.までタイプしてCtrl-x Ctrl-oをタイプします。Ctrl-nとCtrl-pで補完メニューの候補を選択し、「Create」を見つけてEnterをタイプします。このとき、補完候補には関数名しか表示されませんので、次の設定をvimrcに追加して、関数の詳細を表示するようにしておくと便利です。

```
set completeopt=menuone,noselect,preview
```

menuoneは、たとえばos.Cまでタイプして補完を行った際に、Cから始まる候補だけを表示するオプションです。また**noselect**は入力補完時に候補が自動選択されるのですが、その動作を無効にします。これにより、ほかのテキストエディタの入力補完と同じ動作を行います。最後の**preview**は選択している候補の詳細を表示するオプションです（**図3**）。

ファイル選択

Go言語ではパッケージごとにディレクトリを作成し、そこに同一のパッケージ名で複数のファイルを作成します。どの言語でも同じですが、このように複数のファイルを扱う場合は簡単にファイルを開けられるようにしておくと、思考が停止せずスムーズな開発が行えます。

ファイルを簡単に開くためのプラグインで有名なところではCtrlPかDeniteが良いでしょう。次の設定でインストールできます。

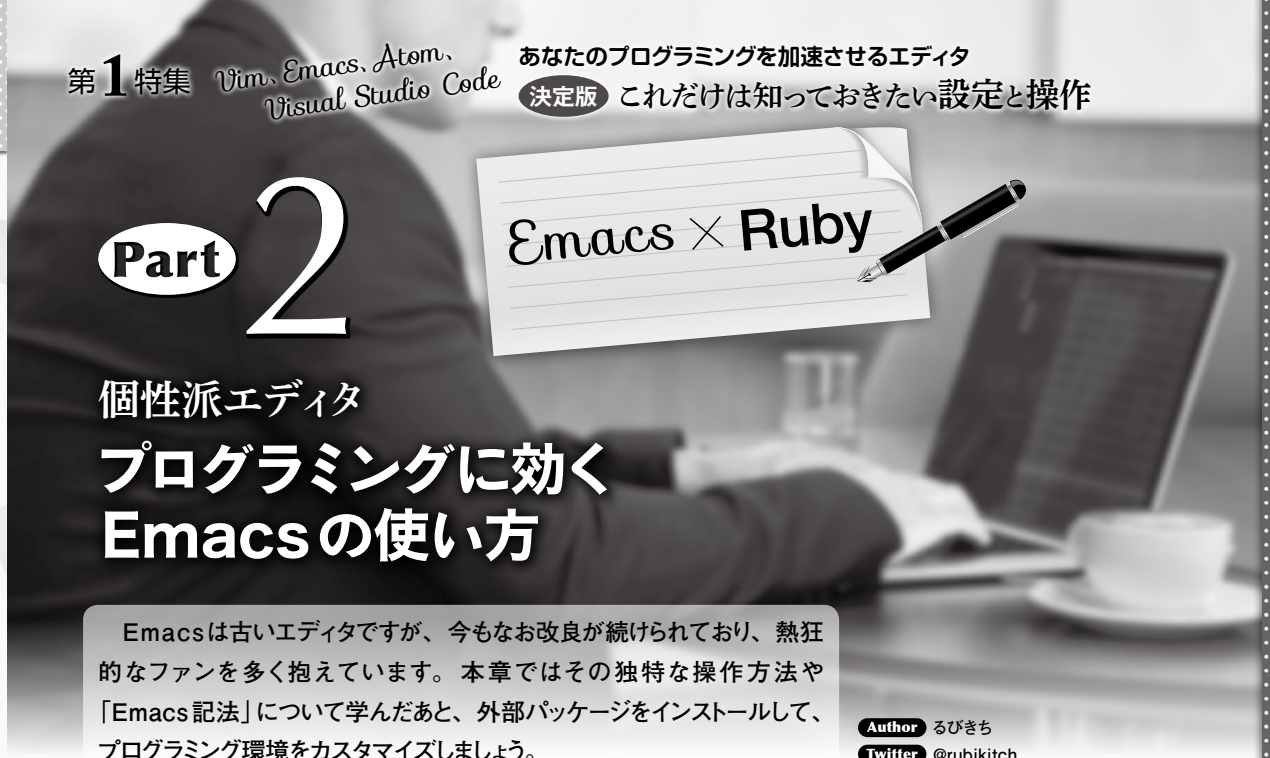
```
Plug 'ctrlpvim/ctrlp.vim'
Plug 'Shougo/denite.nvim'
```

どちらもカレントディレクトリ配下のファイルが一覧表示され、マッチする文字を数文字入れるだけで選択できます。CtrlPの場合はCtrl-pをタイプすると起動します（**図4**）。

まとめ

本章では、Vimの特徴と基本的な操作方法を紹介させていただきました。また、Go言語を編集するうえで筆者が常用している環境を説明させていただきました。紹介しきれなかったプラグインの中にも便利なものがたくさんあります。ぜひ自分にあったプラグインをGitHubなどから探してみてください。

Vimが「古くそして新しいテキストエディタ」であるという意味が、少しでも多くのみなさんに伝われば幸いです。SD



Part 2

個性派エディタ プログラミングに効くEmacsの使い方

Emacsは古いエディタですが、今もなお改良が続けられており、熱狂的なファンを多く抱えています。本章ではその独特な操作方法や「Emacs記法」について学んだあと、外部パッケージをインストールして、プログラミング環境をカスタマイズしましょう。

Author るびきち
Twitter @rubikitch

Emacsとはどんなエディタ?

ども、るびきちです。筆者はEmacs一筋20年、我が娘のように育ててきました。本誌2017年4月号まで36回に渡って「るびきち流Emacs超入門」の連載をしていました。

Emacsは古さと新しさが調和した超個性派なテキストエディタです。Emacsにはさまざまな実装がありますが、一般にEmacsといえば「GNU Emacs」を指します。GNU Emacsは今までの30年間ずっと、開発が活発に行われています。30年前の姿を保ちつつ、先進的な機能も取り込んでいます。ITの世界の変化のスピードには目を見張るものがありますが、そこでもEmacsは淘汰されずに生き残っています。この事実は驚くべきことです。世界中に熱狂的ユーザも多く、今後も存続し続けるでしょう。

Emacsの根本的な部分は30年前から変わっていませんが、なぜ変化についていけているのでしょうか。それは、Emacsを拡張するための言語(マクロ言語、拡張言語)にLisp(Emacs Lisp)を採用したことが挙げられます。Lispはシンプルながらも、あらゆるスタイルのプログラミングができます。一言で言うと「シンプルながらも無限の可能性を持った古代の叡智」です。現代のプログラミング言語の多くに、Lispのエッセンスが取り入れられています。

「プログラマはLispを学べ」という昔からの格言があります。Lispの考え方を学ぶことで、プログラマとしての教養や感性が身に付くからです。とはいえ実務でLispに触れることは、ほとんどないのが現状ですね。

そこで、Emacs Lispを学んでみるのはどうでしょうか。Emacs Lispを通してLispを学ぶことで、Emacsを自由にカスタマイズでき、生産性が上がり、同時にプログラミングのセンスまで磨かれます。Lispを学ぶためのモチベーションとしては十分ではないでしょうか。

本記事ではEmacs入門者へ向けてプログラミング環境、とくにRubyの環境を整える内容ですが、実はRubyとLispは相性抜群なのです。Rubyは読み書きしやすい文法でありながら、Lisp的な要素を多く含んでいます。実際、Rubyの生みの親であるまつもとゆきひろ氏は、言語オタクで熱狂的なEmacs Lispハッカーです。

一方で、RubyのアイデアをEmacs Lispに輸入している人もいます。筆者も、RubyとEmacs Lispでは同じような感覚でプログラミングできているのを感じています。

筆者はプログラミングに限らず、

・メール
・文書作成
・ブラウザのテキストエリア
・Twitter
・予定表管理

など、あらゆるテキスト入力をEmacsで行っています。これらを行うためのEmacs Lispプログラムが存在するのです。「何をするにあたっても常に同じ操作で行えるEmacsがある」という安心感は計りしれないものです。

Emacsをインストール

Emacsをインストールするための方法を各プラットフォーム別に解説していきます。

GNU/Linux

Emacsを使うのであれば、OSは断然GNU/Linuxをお勧めします。Emacsをトコトン使うには、外部プログラムと連携する必要があり、そのほとんどはディストリビューションのパッケージで簡単にインストールできるからです。

GNU/Linuxは昔からパッケージングシステムが当たり前の世界ですので、インストールでつまづくことは本当に少ないです。たとえばあるソフトウェアをインストールする際、ほかのライブラリやソフトウェアが必要な場合は自動でインストールしてくれます。しかもパッケージ数がほかのOSとは比較にならないほど多いです。説明書には「○○もインストールしてください」とだけ書かれ、手作業でインストールすることと比べれば、天と地ほどの差です。

Ubuntu等Debian系であれば、

```
$ sudo apt-get install emacs25 emacs25-el
```

を実行してください。

macOS

macOSではテキスト編集環境にて、部分的にEmacsの操作が使えます。そのため、本物のEmacsを学ぶことでテキスト編集効率が上がるメリットがあります。

macOSにも、非公式ながら主流となっているHomebrewパッケージングシステムがありますので、ソフトウェアのインストールが楽です。EmacsもHomebrewからインストールできます。

```
$ brew tap railwaycat/emacsmacport
$ brew install emacs-mac --with-modern-icon $ brew linkapps
```

Windows

GNU/Linuxとは反対に、WindowsでEmacsを使いこなすのは茨の道です。

外部プログラムを使うEmacs Lispプログラムを Windowsで動作させるためには中級以上のEmacs力が必要です。また、Windowsを使っていないEmacs Lispプログラマもいますので、動かないものも出てきます。

Emacsをちょっと試す程度ならば、Windowsでもかまいません。ただ、Emacsを開発環境として本気で使うのであれば、Windowsならではの試練がたくさん待ち構えています。

代表的なEmacs on WindowsにはNTEmacs64とgnupackがありますが、NTEmacs64[注1]をお勧めします。

NTEmacs64はWindowsネイティブですので動作が速いです。また、公式バイナリでは日本語入力に問題がありますので、IMEパッチが当てられています。注1のGitHubのページからzipファイルをダウンロードして展開し、exeファイルを起動することで、インストールできます。

gnupackはCygwinというUnixエミュレー

注1） URL https://github.com/chuntaro/NTEmacs64

ション環境ですので、動作が遅く、パス変換がとてもやっかいです。

Emacsを使いこなすためには外部プログラムが不可欠ですが、現時点でおもに2つのパッケージマネージャーがあります。ChocolatyとMSYS2です。

Chocolaty[注2]は着々と成長していて、執筆時点で4,700のパッケージがメンテナンスされています。

MSYS2[注3]は、Windowsネイティブで動作するUnixライクな環境です。MSYS2はArch Linuxで使われているpacmanパッケージマネージャーが用意されています。現在は2,000くらいのパッケージが登録されています。

Windowsにおける環境構築が苦痛になる最大の原因が、確固としたパッケージマネージャーが欠如していたことでした。ChocolatyやMSYS2のおかげで、以前よりも楽に環境構築できるようになりました。けれどもこれらは、GNU/Linuxとは違ってOS公式ではないため、パッケージが古かったりバイナリがだぶったりする問題は残ります。

Windows10の、「Windows Subsystem for Linux (WSL)」を使えば、Windows上でUbuntuそのものが動作します。Emacsを本気で使いたければ、WSL経由で使うことも視野に入れてください。素のWindowsでは挫折した、または困難だった機能が、いとも簡単に使えるようになるでしょう。

基本的な使い方

それでは、Emacsの基本的な使い方を覚えていきましょう。チュートリアルで手を動かしながら学び、最低限の機能を使えるようにしましょう。

キー操作の記法を知る

Emacsは昔から、キーボード操作が中心のテキストエディタです。キーボード操作を簡潔に表記するため、独自の「Emacs記法」が用いられます（**表1**）。Emacsでは歴史的事情により、Alt キーのことをメタキーと呼びます。またAlt+X（**M-x**）は、Esc を押してから x を押す操作でも代用できます。

ファンクションキーなどの特殊キーは**<f1>**などのように**<>**で囲みます。おもな特殊キーに対応するEmacs記法は**表2**になります。キーに対応するEmacs記法を知るには、**<f1> c**（F1 押してから c）のあとにそのキーを押せば調べられますので、無理に覚える必要はありません。

▼表1　Emacs記法

操作方法	通常の記法	Emacs記法
Ctrl を押しながら x	Ctrl+x	C-x
Alt を押しながら x	Alt+x	M-x
Ctrl と Alt を押しながら x	Ctrl+Alt+x	C-M-x
ファンクションキー F1 を押す	F1	<f1>

▼表2　Emacs記法（特殊キー）

特殊キー	Emacs記法
F1	<f1>
Tab	<tab>
Enter	<return>、RET
Esc	<escape>、ESC
Back Space	<backspace>
Insert	<insert>
Delete	<delete>、<deletechar>
Home	<home>
End	<end>
Page Up	<prior>
Page Down	<next>
↑	<up>
↓	<down>
←	<left>
→	<right>
テンキーの 1	<kp-1>
Space	SPC

注2）URL https://chocolatey.org
注3）URL http://www.msys2.org

チュートリアルで手を動かしながら学ぶ

Emacsは歴史あるソフトウェアですので、操作方法が独特です。とくにWindowsのショートカットキーに慣れている人にとっては、大きな関門となるでしょう。ですが、Emacsはテキスト編集の「専門家」ですので、ショートカットとは比べものにならないほど多種多様な機能に、キーボードのみでアクセスできます。テキスト編集をすべてEmacs上で行うという覚悟があれば、覚えられるでしょう。Windows上のアプリケーションをEmacsの操作で行うXKeymacsやAutoHotkeyというソフトウェアもありますので、Emacsの操作に「改宗」しても問題ありません。

幸い、Emacsには手で動かしながら学べる日本語チュートリアルが付属しています。Emacsを立ち上げたら、**<f1> t**を押してチュートリアルを起動しましょう（**図1**）。これでEmacsの基本は押さえられます。チュートリアルに書かれていることをマスターすれば、Emacsをかなり実用的に使えます。今のEmacsは、とくに設定を加えなくてもそれなりに快適に使えるようになっています。設定しないとまともに使えないのは、過去の話です。

メニューから操作する

チュートリアルを使っていると自然に操作を覚えられますが、どうしても操作方法を思い出せないときがあります。そのときはメニューから操作できます。**図2**はメニューバーのFileをクリックしたところです。操作の横にキーを書いていますので、何度かメニューから操作していくにつれてキーボード操作を覚えていけるようになります。

カーソル移動

テキストエディタに慣れていない人やメモ帳を常用している人は、カーソルを移動させるときに矢印キーなどの特殊キーやマウスを使うでしょう。Emacsでも特殊キーやマウスは普通に使えます。しかしもちろん、Emacs独自のキー操作を覚えることで編集効率は大幅に上がります。なぜなら、ホームポジションから手を離さずに済むからです。無理に覚えようとせずとも、気になったときに調べてみると良いです。基本的なカーソル移動については**図3**をおさえていれば十分です。

ファイルを開く

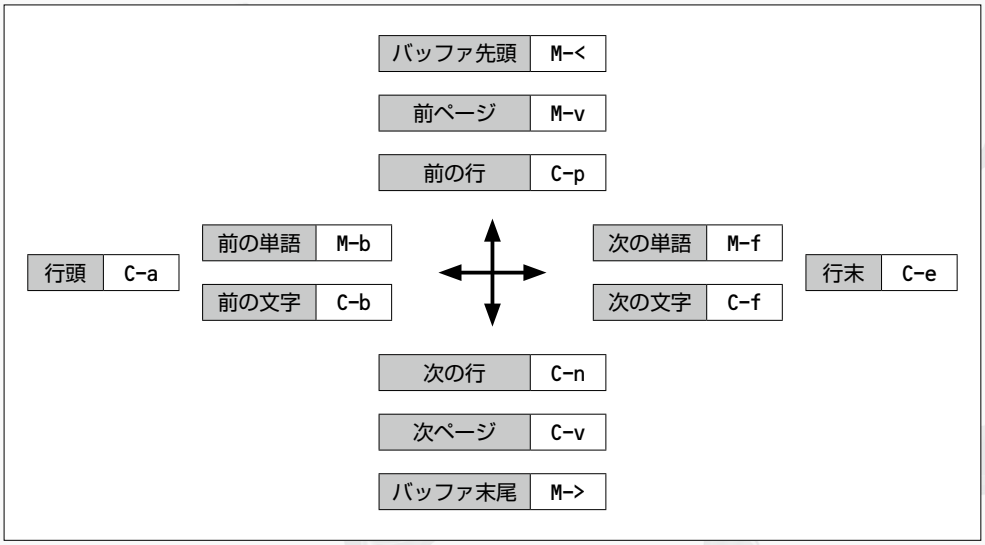
ファイルを開くのはテキストエディタの基本

▼図1 日本語チュートリアル

```
Emacs 入門ガイド.  本ファイルの著作権については最後を御覧下さい。

 NOTICE: The main purpose of the Emacs tutorial is to teach you
 the most important standard Emacs commands (key bindings).
 However, your Emacs has been customized by changing some of
 these basic editing commands, so it doesn't correspond to the
 tutorial.  We have inserted colored notices where the altered
 commands have been introduced. [More]


Emacs のコマンドを入力するには、一般にコントロールキー（キートップに
CONTROL とか CTRL とか CTL と書いてある）やメタキー（キートップに META
とか ALT とか EDIT と書いてある）を使います。そこで、CONTROL とか META
とかを書く代わりに、次のような記法を使うことにします。

 C-<文字>    コントロールキーを押したまま、<文字>キーを押します。例えば、
             C-f はコントロールキーを押しながら f のキーを押すことです。
 M-<文字>    メタキーを押したまま、<文字>キーを押します。もしメタキーがな
             い場合は、エスケープキーを押してから離し、それから<文字>キー
             を押します。以降エスケープキーのことを <ESC> と書きます。
J:---  TUTORIAL.ja     Top L1      (Fundamental)
```

▼図2 Fileメニュー

▼図3　カーソル移動のまとめ



ですね。Emacsでファイルを開くには**C-x C-f**と操作します。すると、最下行（ミニバッファ）に、

```
Find File: ~/src/
```

などのカレントディレクトリがついたプロンプトが出てきます。ここでファイル名を入力すればそのファイルを開けます。

ファイル名はすべて入力する必要はありません。Tabを押せばファイル名を補完してくれます。補完とは、Emacs全体で使われる入力補助機能です。Emacsでの補完はTabで行われるため、しばしば「TAB補完」とも言われます。たとえば、カレントディレクトリ上に、

・a.txt
・bar1.txt
・bar2.txt

が存在するとき、bのあとにTabを押せばbarまで入力されます。なぜなら、bから始まるファイル名はbar1.txtとbar2.txtであり、どちらもbarから始まっているからです。ここでもう一度Tabを押すと、候補がポップアップ表示されます。そこで1を押してTabを押すと、bar1.txtに確定され、bar1.txtが開かれます。

ファイル名が思い出せないときは、プロンプトにディレクトリが表示されているときにいきなりTabを押せば補完候補として全ファイルが出てきます。

プロンプトに表示されているディレクトリ名とは異なるディレクトリのファイルは、次のいずれかの方法で開けます。

①ディレクトリ名を編集する
②「../」と入力して親ディレクトリを参照する
③そのままフルパスを入力する

存在しないファイル名を入力した場合は、ファイルをその名前で新規作成します。

バッファを切り替える

Emacsで一度ファイルを開くと、ファイル名と同名の「バッファ」が作成されます。バッファとは編集可能なテキストのことです。ファイルを開いて作成したバッファを特別に、ファイルバッファと呼ぶことがあります。

Emacsではファイルバッファ以外のバッファがあります。情報を表示したり、ファイルに保

存しない入力を求めたりするときにファイルと関連付けられていないバッファが使われます。慣習としてバッファ名は*で囲まれるようになっています。たとえば、

・起動時に表示される*scratch*バッファ
・ヘルプ表示に使われる*Help*バッファ
・補完候補を表示する*Completion*バッファ

などがあります。

バッファを切り替えるには**C-x b**を使います。すると、

```
Switch to buffer (default *scratch*):
```

などのプロンプトが現れます。defaultは直前にアクセスしたバッファですので、この場合はそのまま Enter を押せば*scratch*に切り替わります。ほかのバッファはTAB補完を使って選択します。存在しないバッファ名を入力した場合は、新しいバッファが作成されます。

画面を分割する

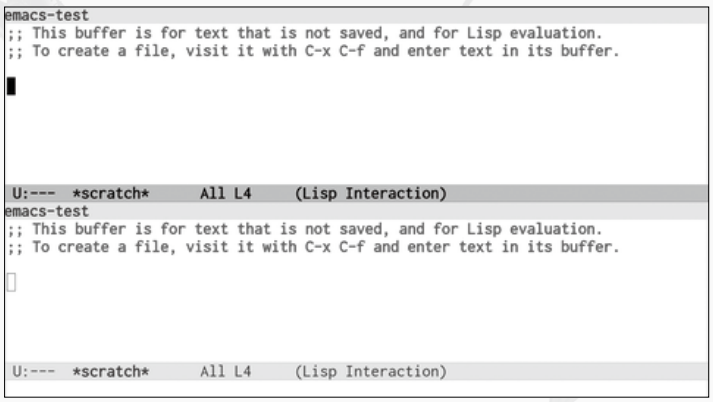
Emacsでは、ウィンドウ分割もキーボードで行えます。ウィンドウ分割することで、ほかのバッファも表示できます。たとえば、

・**C-x 2**で上下分割（図4）
・**C-x 3**で左右分割（図5）

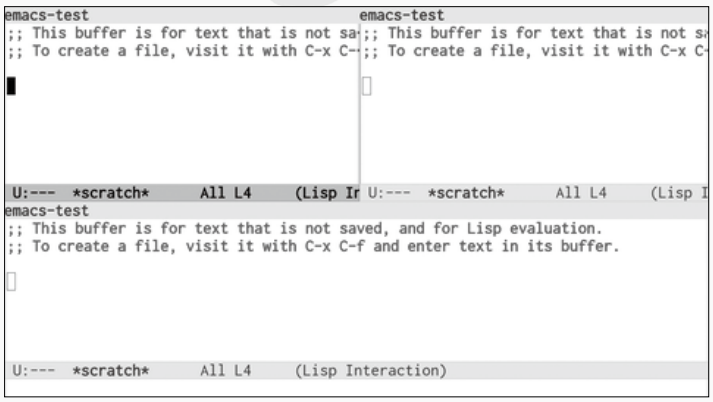
といったことができます

ここで注意したいのが「ウィンドウ」という用語です。一般的なGUIにおけるウィンドウは、アプリケーションを表示している領域全体（画

▼図4 C-x 2



▼図5 図4の状態からC-x 3



▼表3　ウィンドウ関係のコマンド

キー	解説
C-x 2	フレームを上下に分割して新しいウィンドウを作成する
C-x 3	フレームを左右に分割して新しいウィンドウを作成する
C-x 1	フレーム内のほかのウィンドウを消す
C-x 0	選択しているウィンドウを消す

▼表4　インクリメンタルサーチのコマンド

キー	解説
C-r	前方向インクリメンタルサーチ
C-s	後方向インクリメンタルサーチ

面）を意味しますが、Emacsにおいてはこれを「フレーム」と呼びます。Emacsのウィンドウとは、フレームの中にバッファを表示している部分を意味します。紛らわしいのですが、EmacsはGUIがメジャーになる前に誕生したため、歴史的事情により今もその用語になっています。ウィンドウを分割、削除するコマンドは**表3**のものがあります。

インクリメンタルサーチ

バッファ内の文字列検索は、通常の検索用途だけでなく、すばやいカーソル移動の手段にもなります。なぜなら、Emacsでおもに使われているバッファ内文字列検索は「インクリメンタルサーチ」だからです。

インクリメンタルサーチとは、検索文字列をタイプするたびにカーソルが移動する機能です。そのため、最小のタイプ数で目的の場所にカーソルを移動できます。インクリメンタルサーチ中に**C-r**や**C-s**を押すと、同じ文字列で再度インクリメンタルサーチを行います（**表4**）。

たとえば、

```
1: abcd
2: aefg
3: 0
4: ★
```

というテキストで★の位置にカーソルがあるとして、**C-r a**で2行目のaに移動します。そこで再び**C-r**を押すと1行目のaに移動します。また、★の位置で**C-r ab**とすると、1行目に移動します。このようにインクリメンタルサーチでは、abcdなどと長い文字列を入力しなくて済みます。

移動したい場所が目に入ったときにインクリメンタルサーチをすることで、カーソル移動コマンドよりもすばやく移動できます。ぜひとも使いこなして、バッファ中を自在に移動できるようにしてください。

取り消し・やり直し

テキストエディタを使っていると、操作ミスは付き物です。パニックになる前に立ち直る方法を覚えておきましょう。

- **C-g**は間違ったコマンドを取り消す
- **C-/**は間違った編集をやりなおす

たとえば**C-x b**を使おうとして**C-x C-f**を押してミニバッファが出てきたとき、**C-g**を押せばQuitと出てミニバッファが取り消されます。また、**C-x**と押すべきときに**C-c**を押してしまったときにも、**C-g**を押せばコマンドをやりなおせます。

変な編集をしてしまったときには、**C-/**でバッファの変更内容を元に戻します。

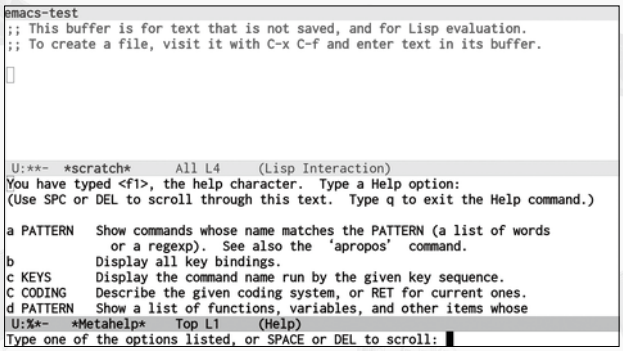
とにかく、困ったら**C-g**や**C-/**と覚えてください。

わからないときは<f1>を

Emacsには昔からオンラインヘルプ機能があります。わからないことは、Emacsに訊けばEmacsが教えてくれます。**<f1> <f1>**でヘルプのメニューが出てきます（**図6**）。

ヘルプメニューでは、その後どういうキーを打てば必要とする情報が得られるかを教えてくれます。たとえば、「c KEYS Display the command name run by the given key sequence.」という項目によれば、**<f1> <f1>**を押したあとに**c** <あるキー>

▼図6 <f1> <f1>でヘルプメニュー



と打てば、そのキーのEmacs記法と、そのキーに割り当ててられているコマンドが表示されます。

ヘルプメニューは画面に表示しきれないほどの選択肢がありますので、Space でスクロールできます。また慣れてくると、ヘルプメニューを介さないでもヘルプコマンドを実行できます。そのとき、**<f1>**をタイプする回数を1回に減らせます。たとえば**<f1> <f1> c**ならば、**<f1> c**で実行できます。

メニューにリストされているおもなヘルプコマンドを**表5**に挙げます。

プログラミングに効く機能

Emacsはハッカー向けのエディタですので、プログラミングがしやすくなる機能がたくさんあります。テキスト色付け、正規表現、Emacs内でのプログラムの実行は欠かせません。ここでは触れませんが、各種バージョン管理システムとの連携も得意とします。

▼表5 ヘルプコマンド

項目	説明
b	使えるキーバインドを一覧する
c KEYS	キーに割り当てているコマンドを表示する
k KEYS	キーに割り当てているコマンドの説明を表示する
o SYMBOL	変数・関数・コマンドの説明を表示する
t	チュートリアルを起動する

正規表現にマッチする行をリスト

M-x occurは正規表現にマッチした行をリストします。いわゆるバッファ内grepです。

たとえばRubyスクリプトにて、**M-x occur RET ^ *def RET**というコマンドでメソッド一覧が隣のウィンドウに表示されます。その後**M-g M-n**で次にマッチした行に移動できます。**M-g M-p**は反対方向です。

ただし、Emacsの正規表現はRubyのものとは異なります。普段使いのものとして、**表6**の3つの違いを押えておけば良いです。

複数のファイルを検索

M-x grepはgrepプログラムをEmacs内で動かし、マッチした行に移動できるようにします。grepの細かいオプションはプラットフォームによって異なりますが、Emacsがあらかじめ算出してくれます。

M-x grepを実行すると、筆者の環境では「grep --color -nH -e 」とミニバッファに出てきます。あとは通常のgrepと同様に、正規表現と検索対象のファイル名（ワイルドカード）を入力してください。なお、**M-x grep**は「grep -n形式（ファイル名:行番号:〜）」であれば、ほかの検索プログラムも実行できます。

スクリプトをEmacs内で実行

M-x executable-interpretはカレントバッファのスクリプトをEmacs内で実行します。

▼表6 EmacsとRubyの正規表現のおもな違い

Ruby	Emacs	解説
|	\	区切られた正規表現のどれか
(〜)	\(〜\)	グルーピング
\d	[0-9]	数字

実行すると、ミニバッファにコマンドライン入力を求められます。エラーや警告が出たときは**M-x grep**同様**M-g M-n**と**M-g M-p**で該当行に移動できます。getsメソッドなど入力が求められる場合は、**C-x o**で*interpretation*ウィンドウを選択してから入力してください。

同種のコマンドに**M-x compile**があります。コンパイラを始めとする任意のプログラムを実行できますが、入力は受け付けていません。

カスタマイズのための入口

今のEmacsはデフォルトの設定でもそれなりに動作してくれますが、Emacsの持ち味はとてつもなく柔軟にカスタマイズできることです。

初期設定ファイルinit.elを書く

Emacsは昔からあるソフトウェアなだけに、設定はGUIよりも初期設定ファイルを書くほうが主流です。最初はとっつきにくさを感じますが、慣れてくるとバージョン管理や検索が可能であるメリットを強く感じるようになるでしょう。

Emacsは起動時に~/.emacs.d/init.elを読み込みます。古い情報源では~/.emacsや~/.emacs.elと書かれていますが、今からEmacsを始める方はinit.elに書きましょう。

自分の理解度に応じてinit.elを加筆修正していくのがEmacsのカスタマイズです。Emacsに対する理解度が深まるにつれ、より細かいカスタマイズができるようになり、Emacsはあなた色に染まってきます。ぜひとも子育て感覚でEmacsを育てていってください。

自動で実行属性を付ける

Unix系OSにおいては、1行目が#!から始まり、実行属性が付いているファイルであれば、ファイル名を指定するだけでスクリプトとして実行できます。そこで、1行目が#!である場合は自動で実行属性を付けるよう設定すると、わざわざシェルから**chmod +x**を実行する手間を省けます。init.elに**リスト1**の設定を書き加えましょう。

パッケージの設定

RubyにおけるRubyGems同様に、Emacsにも外部パッケージが簡単に導入できます。そのためにinit.elの先頭に**リスト2**の設定を書いてください。

パッケージのインストール・削除・更新は**M-x list-packages**で行います。タイプするとパッケージの一覧が出てきますので、**i**でインストールしたいパッケージを1つ以上マークし、**x**で実際にインストールします。

昔のEmacsでは、パッケージをインストールすると初期設定ファイルに設定を書く必要がありましたが、パッケージシステムを使えば、いきなりパッケージのコマンドが使えるようになる場合が多いです。最低限の設定はパッケージシステムが面倒を見てくれるからです。

安易にパッケージに手を出さない

パッケージシステムを使えば、幾千ものパッケージを簡単にインストールできます。とはい

▼リスト1　chmod +xを省略する

```
(add-hook 'after-save-hook
          'executable-make-buffer-file-executable-if-script-p)
```

▼リスト2　外部のリポジトリと接続する設定

```
(package-initialize)
(setq package-archives
      '(("gnu" . "http://elpa.gnu.org/packages/")
        ("melpa" . "http://melpa.org/packages/")
        ("org" . "http://orgmode.org/elpa/")))
```

え、安易にいろいろなパッケージに手を出すのは良くありません。人は自分の理解を超えるものは使いこなせないからです。

ネットで検索していると、いろいろな人が「このパッケージは便利だよ、オススメ」などと書いていますが、人それぞれレベルが違います。人のお勧めが、必ずしもあなたに合っているとは限りません。ネット上で設定を検索してコピー&ペーストする場合、その設定の意図をしっかり理解してください。それを怠ると、あとあとトラブルの元になります。

Emacsを使い始めたのならば、まずは基本操作をしっかりと身に付け、土台を固めてください。基本なくして応用はありません。利便性には代償が伴います。いくら便利なパッケージでも、途中で問題を起こしてしまったら、問題解決に時間と労力を費やすハメになります。

EmacsでRubyを書くには

それではここからは具体例として、EmacsでRubyプログラミングをするために便利なパッケージを3つ厳選します。`M-x list-packages`のあとに、

- inf-ruby
- seeing-is-believing
- ripgrep

を`i`で選択して`x`でインストールしてください。

対話型環境をEmacs内で動かす

Rubyには、irbという対話型環境があります。inf-rubyパッケージは、irbをEmacsで呼び出します。何も設定しなくても、`M-x run-ruby`あるいは`M-x inf-ruby`を実行すると、irbが立ち上がります。

inf-rubyは、irbを強化したpryにも対応しています。pryを使うには、次の設定をinit.elに加えます（pryは「gem install pry pry-doc」でインストールします）。

```
(setq inf-ruby-default-implementation "pry")
```

Rubyスクリプトに値を注釈する

seeing_is_believingは、Rubyスクリプトに値を注釈する優れたgemです。そのseeing_is_believingのEmacsインターフェースが、seeing-is-believing（アンダーラインではなくてハイフン）パッケージです。

seeing_is_believingは、筆者がかつて開発していたxmpfilter（rcodetools）の後継版です。xmpfilterやseeing_is_believingは、irb/pryよりも優れた対話的環境です。なぜなら、Rubyの式のすぐ横にコメントの形で値が注釈されるからです。複数の式を一度に評価でき、注釈後のスクリプトがそのまま実行可能で、スクリプトを更新すると、再実行すれば注釈がすぐに更新されます。

seeing_is_believingは「gem install seeing_is_believing」でインストールします。たとえば、

```
1+3  # =>
[1,
2]   # =>
```

のような複数行に式が渡ったRubyスクリプトでも、seeing_is_believingを実行すると、

```
1+3  # => 4
[1,
2]   # => [1, 2]
```

と、値を注釈してくれます。xmpfilterでは複数行に渡る式に対応できていませんでしたが、seeing_is_believingはRubyスクリプトを厳密に解析するため、可能になっています。

Emacsでseeing_is_believingを使うには、**リスト3**の設定をinit.elに加えます。すると、Rubyスクリプトで、**表7**に挙げた操作が使えるようになります。

seeing_is_believingのデフォルトの設定は、

対象となるRubyスクリプト全行を注釈します。そのため、試行錯誤のために数行のRubyスクリプトをサッと書いて**C-c ? s**を実行するのが、一番手軽です。

一方で特定の式の値のみ知りたい場合は、**C-c ? t**でその行に明示的に注釈コメントを挿入し、**C-c ? x**を実行します。これは、xmpfilterに慣れている人や、試行錯誤のスクリプトが長くなった場合に向いています。

マッハ検索!

ripgrep(実行ファイル名はrg) は、筆者の知る限り最速の万能ファイル検索ツールです。Emacsのripgrepパッケージは、ripgrepを簡単に呼び出せるようにします。Rubyプログラマに限らず、全Emacsユーザ必携です。

ripgrepを導入すべき理由は5つあります。

- the_silver_searcher (ag) のように複数のファイルを高速に検索できる
- 余計なファイルを検索対象から外す
- 単一ファイルの検索も高速である
- マッチした行だけでなく、列にも移動してくれる
- 多くの場面でGNU grepやagより数倍速い
- Windows/Mac/Linuxバイナリも配布されている

長い間、Emacs on Windowsにおける検索環境は劣悪でしたが、これは決定打と言えます。

ripgrepを使うためには、

❶https://github.com/BurntSushi/ripgrep/releasesからripgrepバイナリを入手する
❷展開してバイナリファイルrg (Windowsではrg.exe) を適当な場所に配置する (ここでは~/bin/rg)
❸次の設定をinit.elに加える

```
;;; rgバイナリを設定する
(setq ripgrep-executable "~/bin/rg")
```

WindowsバイナリはMinGW(GNU)とMicrosoft Visual C++(MSVC) という2種類のバイナリがあります。MinGW版はそのまま動作しますが、MSVC版を使うならばMicrosoft VC++ redistributableをインストールする必要があります。

▼リスト3　Emacsでseeing_is_believingを使う

```
(require 'seeing-is-believing)
(add-hook 'ruby-mode-hook 'seeing-is-believing)
```

Emacsでのripgrepの使い方は4ステップです。

①**M-x ripgrep-regexp**を実行する
②検索する正規表現を入力する
③検索するディレクトリを入力する
④**M-g M-n**や**M-g M-p**で該当行へ移動する

おわりに

Emacsは大昔から存在する超個性派テキストエディタです。独特な操作性ゆえ、若い人は気難しさを感じるかもしれません。けれども使い込めば使い込むほど、Emacsはあなたの最良のパートナーとなってくれます。現に筆者は20年以上Emacsを使っていますが、飽きませんし、さらなる進化の可能性を感じています。

本記事では駆け足でEmacsについて触れてきましたが、さらにEmacsを学ぶには筆者のサイト「日刊Emacs」を参照してください。パッケージについても書いていますが、本誌で先月まで連載していた「るびきち流Emacs超入門」のバックナンバーを全文公開しています。またるびきち塾 (http://emacs.rubikitch.com/juku/) では月527円で毎週土曜日のメルマガ+Emacs無制限個別指導を行っています。最後まで読んでいただき、ありがとうございます。SD

▼表7　seeing-is-believingのおもな操作

キー	解説
C-c ? s	全行の式の値を注釈する
C-c ? c	注釈コメントを消す
C-c ? t	行末に注釈コメント「# =>」を挿入する
C-c ? x	注釈コメントの行のみ式の値を注釈する

Part 3

Atom × JavaScript

そのままもヨシ、思い切り強化するもヨシ

使い手を選ばない Atomのはじめかた

AtomはGitHubとの関係が深いテキストエディタとして着実にユーザを増やしています。GitHubを利用する世界中の開発者たちを巻き込んで、開発効率を上げるための機能が次々に提供される点が魅力です。一方で標準機能も充実しており、手を加えなくてもしっかり使えることを本章で紹介していきます。

Author 大竹 智也（おおたけ ともや）
Blog http://d.hatena.ne.jp/tomoya/

新世代エディタAtom

AtomはWebエンジニアであれば、誰もがお世話になるGitHubが中心となり開発されているテキストエディタです。公式のコピーに「A hackable text editor for the 21st Century」とあるように、21世紀に誕生したとても新しいエディタです。そのため、エディタとしての機能だけでなく、Atom自身のアーキテクチャもこれまでのエディタと異なる、とても新しくて魅力的なものになっています。

筆者は以前、『Emacs実践入門』というEmacsの入門書を執筆したことがありますが、Atomにも強い興味を持った結果、『Atom実践入門』という書籍も執筆しました。

本章では、2つのエディタの入門書を執筆した経験のある筆者だからこそ伝えられる、エディタの魅力と使い方を解説していければと思います。

Atomとは

Atomは、GitHub創業者の一人defunkt（Chris Wanstrath）氏[注1]のサイドプロジェクトとして2008年に開発がスタートしました。彼の夢は、Web技術を使用してEmacsのようにカスタマイズ可能なエディタを作ることでした[注2]。その後、2015年6月に1.0がリリースされたAtomは、約1年後の5月には、4,000を越えるパッケージと100万人のユーザを持つ巨大なコミュニティへと成長しました[注3]。

また、Atomの技術基盤であるElectron[注4]は、Web技術を利用したクロスプラットフォームアプリケーションを実現した革新的なアーキテクチャとして、ソフトウェア開発の世界にも大きな影響を与えました。

Atomの特徴と魅力

さまざまな特徴と魅力を持つAtomですが、筆者が考えるAtomの最大の特徴と魅力は次の2つに収束します。1つめはすぐれた標準機能を持っていること、2つめは豊富で強力なパッケージがそろっていることです。

すぐれた標準機能と豊富なパッケージを持つ

注1）URL https://github.com/defunkt

注2）defunkt氏はEmacsのヘビーユーザ。
URL https://www.wired.com/2015/06/github-atoms-code-editor-nerds-take-universe/

注3）URL http://blog.atom.io/2016/05/06/two-years-open-source.html

注4）URL https://electron.atom.io/

Atomは、とくに設定することなく、自動補完や30以上の言語サポートなどを利用した編集が行えます。また、足りない機能があった場合でも、目的にあったパッケージを見つけてインストールすることで、すぐに補えるようになっているので、初心者であっても簡単に自分の理想のエディタへと成長させることが可能です。

インストール、画面構成、設定

ここからは、Atomのインストール、画面構成、設定について解説していきます。

インストール

まずはOS別のインストール方法を解説します。基本的には、公式サイト[注5]からダウンロードしてインストールを行います。

Macの場合atom-mac.zipというファイルがダウンロードできますので、ダウンロードしたあとダブルクリックします。すると、Atom.appファイルが同じディレクトリに展開されるので、これをアプリケーションディレクトリにドラッグ&ドロップするだけでインストールが完了します。

Windowsの場合はAtomSetup-x64.exeをダウンロードして、インストーラを起動します。インストールが完了すると、自動的にAtomが起動します。

Linuxはディストリビューションにあわせて2通りあります。Debian系の場合はatom-amd64.deb、Red Hat系の場合はatom.x86_64.rpmをダウンロードして、次のコマンドからインストールします。

Debian系
```
$ sudo dpkg -i atom-amd64.deb
```

Red Hat系
```
$ sudo yum install -y atom.x86_64.rpm
```

ついでにアップデートについて解説しておくと、Atomは自動アップデートのしくみを持っているため、一度インストールすれば、以後Atomが更新されたときに自動的にダウンロードが行われ、再起動するだけで自動的にアップデートが完了します。

以降の記事はMac版での解説となります。Windows版やLinux版をお使いの場合は適宜読み替えてください。

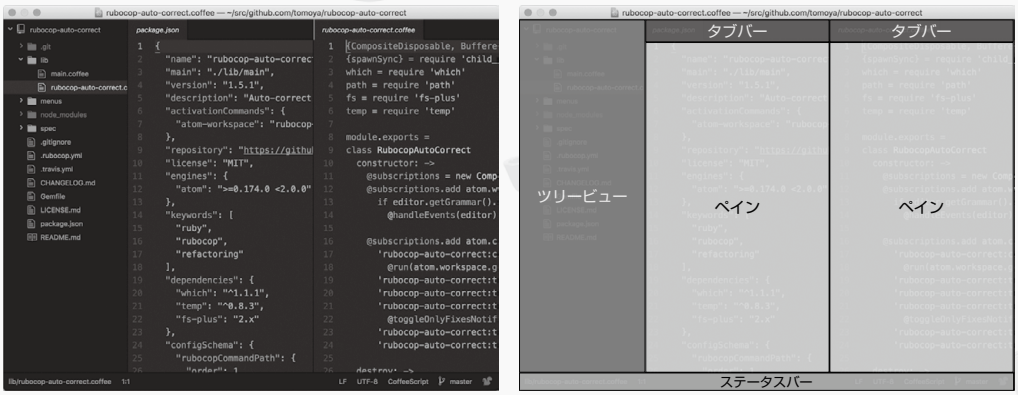
画面構成

Atomの画面はChromeブラウザのようなタブバーを持っています。**図1の左側**は、AtomでGit管理しているプロジェクトのファイルを開いたときの画像です。このスクリーンショットに、それぞれの領域の説明を記述したものが**図1の右側**になります。

Atomはファイルを開くと、ウィンドウの左

注5）URL https://atom.io/

▼図1　Atomの画面構成



にツリービューという専用のファイルブラウザが表示されます。こちらのファイル名をクリックすると、ファイルが開かれてAtomに表示されます。このファイルが表示されている場所をAtomではペインと呼び、自由に分割してウィンドウ内にいくつも並べられるようになっています。

ウィンドウの最下部にはAtomのさまざまな情報が表示されるステータスバーがあります。ここに表示されている情報をクリックすると、それぞれが提供する機能を使うことができるので、ぜひ試してみてください。

基本設定

Atomメニューの[Preferences…]を選択、あるいはキーボードショートカットのCommand-,[注6]を実行すると、設定パネルが開きます。この設定パネルからAtomのさまざまな設定を行えます。左のメニューから設定項目が選択できますので、それぞれ簡単に説明していきます。

- Core：Atom本体の基本的な設定
- Editor：編集機能や画面表示に関する共通の設定
- Keybindings：Atomに登録されているショートカットをすべて確認
- Packages：Atomにインストールされているパッケージの管理。パッケージを選択すると、パッケージ固有の設定が行える
- Themes：Atomにインストールされているテーマの管理。テーマを選択すると、テーマ固有の設定が行える
- Updates：インストールされているパッケージやテーマの更新の確認と実行
- Install：パッケージやテーマの検索とインストール

Atomは使い勝手の良いデフォルトの設定を提供してくれているため、とくに変更の必要はないかもしれませんが、Atomを起動したら最初に確認してみるといいでしょう。

基本操作

それでは、Atomの操作について解説していきます。エディタの操作は「ファイルを開き、キーボードをタイプして編集する」というのが基本です。Atomにおいての基本的な操作は同じですが、ここでは、Atomの持つすぐれた標準機能を利用した操作方法を交えて解説します。

プロジェクトを開く

Atomには一般的な「ファイルを開く」ではなく「プロジェクトを開く」というしくみがあります。プロジェクトで開いたAtomでは、次に開いたときにタブやカーソル位置の状態などを復元したり、プロジェクト内のファイルのみを対象として検索できたりなど、さまざまな利点が得られるので、ぜひ覚えておきましょう。

プロジェクトを開くには、[File]メニューの[Open…]を選択します。すると、Mac標準のダイアログが表示されます。通常はここでファイルを選択しますが、このとき、ファイルではなくディレクトリを開くことができます。ディレクトリを開くと、そのディレクトリをプロジェクトルートとするAtomのウィンドウが作成されツリービューが表示されます。

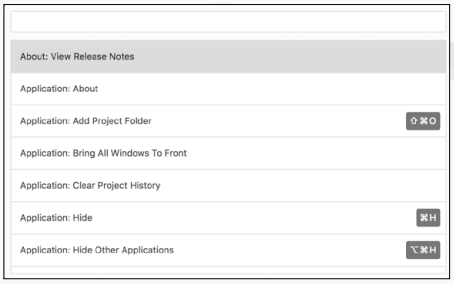
開いたプロジェクトは履歴に記録され、[File]メニューの[Reopen Project]からすぐに開くことが可能です。なお、記録されるプロジェクトの個数は「Core」設定の「Reopen Project Menu Count」から変更できます（初期値は15）。

コマンドパレット

コマンドパレットは、Atomでコマンドを実行するインターフェースです。コマンド操作は、EmacsやVimなどのコマンドを使えるエディタを利用したことのある人にとってはお馴染み

注6）本稿中では、コマンドキーを押しながらコンマキーを押すような操作をこのように表記。

▼図2　コマンドパレット



▼図3　File Finder

の操作です。ですが、コマンド操作を知らない人は、おそらく最初は難しい操作だと感じることでしょう。

しかし、Atomのコマンドパレットはとても簡単にコマンドを利用できるようになっています。ですので、とくに難しく考えず、「メニューを選択する」あるいは「キーボードショートカットを実行する」と同じだと思ってください[注7]。

コマンド操作は、一度利用できるようになると、二度とコマンド操作のない世界には戻ることができないほどの重要性を秘めているので、ぜひ覚えておきましょう。

コマンドを実行する

実際にコマンドパレットからコマンドを実行してみましょう。コマンドパレットを使うには、キーボードから Command - Shift - p を実行する、あるいは、[View] メニューから [Toggle Command Palette] を選択します。すると、**図2**のような小さなウィンドウがAtomウィンドウ上に現れます。これがコマンドパレットです。

ここで Enter を押すと、ブラウザが開いてAtomのリリースノートのページが開きます。これは「About: View Release Notes」コマンドが実行されたためです。

コマンドパレットの使い方は、Googleの検索ボックス、MacのSpotlightとよく似ています。インプットボックスに入力された文字列からコマンドを検索して、下に並んだコマンド一覧を上下カーソルキーで選択して Enter によって実行します。

先ほど Enter を押してリリースノートが開いたのは、何も入力していない状態では、アルファベット順に並んだコマンドが一覧に表示されていたためです。

コマンドパレットを使った操作方法は、コマンドを実行する以外にも、次に解説する「ファイルを開く」や「タブを切り替える」でも使われる、Atomの中でも非常に重要な基本操作の1つになっています。操作に慣れるまで少し時間がかかるかもしれませんが、使いこなせるようになると、とてもすばやくさまざまな操作が行えるようになります。

ファイルを開く／タブを切り替える

Atomでファイルを開く方法は、プロジェクトを開くときのように [File] メニューの [Open…] からファイルを選択したり、ツリービューからファイルをクリックして開いたりと、複数の方法が用意されています。しかし、これらの方法だと、ディレクトリが深くなってきた際や、ファイルの数が増えてきた際に、目的のファイルを見つけるまでに時間がかかってしまいます。そこで便利なのが「Fuzzy Finder」です。ここでは、Fuzzy Finderの機能の中からFile FinderとFind Bufferを解説します。

File Finder は Command - p か Command - t を実行して起動します（**図3**）。すると、プロジェクト内のファイルが一覧として表示されます。コマンドパレットと同じく文字列をタイ

注7） 実際にこれらの操作はコマンド実行を行っている。

プすると絞り込みが行われ、上下カーソルキーで選択を行います。そして、Enterを押すと、選択したファイルがまだAtomで開かれていなければファイルを開き、すでに開かれている場合はタブを切り替えます。

Find BufferはCommand-bを実行して起動します。Find Fileと似ていますが、こちらはAtomで開いているタブのみを一覧として表示してくれます。そのため大量のファイルがあるプロジェクトなどの場合、余計なファイルを無視して、あくまで現在Atomで開いているファイルのみを対象としてタブを切り替えることができます。

ファイルを開いたりタブを切り替えたりする操作は、頻繁に行いますが作業としては無駄な時間とも言えます。ですが、この方法を覚えておけば無駄な時間を限りなく少なくできると思いますので、早めに身に付けておくと良いでしょう。

検索と置換

検索と置換はエディタの操作の中で最も重要だと筆者は考えています。それは、紙と鉛筆では絶対にできない操作であり、単純でありながらも、それゆえに最も生産性に影響を与えてくれるからです。

Atomの検索は一般的なファイル内のテキスト検索だけでなく、プロジェクト内のファイルすべてを対象に検索する2つの機能があります。基本的な操作はどちらも同じになっていますので、すぐに使いこなせるようになるでしょう。

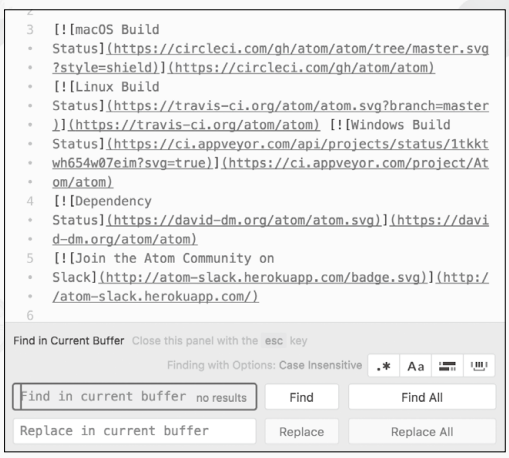
ファイルを検索・置換する

ファイルを開いた状態で、Command-fを実行すると、ウィンドウ下部に検索パネルが表示されます（**図4**）。

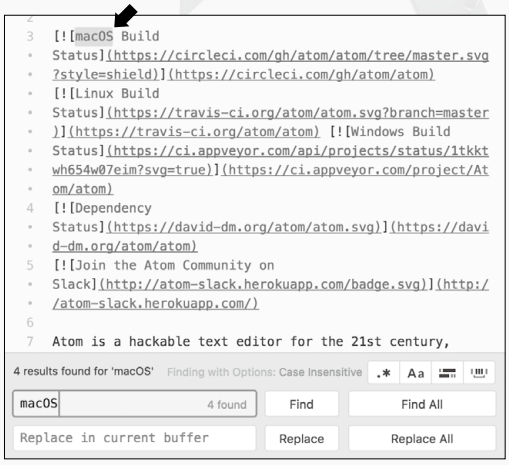
検索パネルの「Find in current buffer」と書かれたインプットボックスに検索したい文字列を入力すると、リアルタイムに検索が行われ、検索にマッチする個所がハイライトされます（**図5**）。検索中にCommand-gを実行すると、上から順に検索でマッチしている個所へとカーソルを移動します。そして、Command-Shift-gを実行すると、今度は下から上へとカーソルを移動します。

置換を行う場合は、検索後に続けて「Replace in current buffer」と書かれたインプットボックスに、置換したい文字列を入力します。[Replace]ボタンを押すと、カーソルのある場所のテキストが置換されます。次にマッチする文字列へとカーソルを移動します。もしすべて置換しても問題なければ、[Replace All]ボタンを押しましょう。

▼図4　検索前の画面



▼図5　検索後の画面



プロジェクトを検索・置換する

次にプロジェクト検索ですが、Command-Shift-fを実行すると、先ほどの検索パネルによく似たプロジェクト検索パネルが開きます。

こちらは、現在ツリービューで表示されているファイルすべてを対象として検索を行います。ファイル数が多い場合のために、ディレクトリやファイルの種類を指定することもできます。

検索後は置換も行えますが、こちらはファイル検索と違って1つ1つ確認して置換することはできません。そのため、置換を行う場合は、よく確認してから行いましょう。

パッケージのインストール

パッケージのインストールは設定パネルから検索して行います。それでは、一例としてlanguage-babel[注8]というパッケージをインストールしてみましょう。

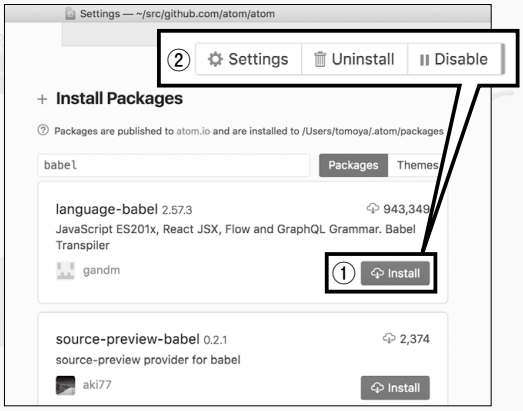
まずは設定パネルのインストールメニューを選択した状態で、「Search packages」と書かれたインプットボックスに「babel」と入力してEnterを実行、あるいは［Packages］ボタンを押します。しばらくすると、**図6**のようにlanguage-babelパッケージが画面に表示されるので、［Install］ボタンを押してインストールします。

インストールが完了すると、［Install］ボタンが**図6-②**のように［Settings］［Uninstall］［Disable］ボタンへと表示が変わります。

インストールしたパッケージはすぐに利用し始めることが可能ですので、気になるパッケージはどんどんインストールしてみましょう。

もしインストールしたパッケージが気に入らなければ、無効化（Disable）、あるいは削除（Uninstall）しましょう。削除は文字どおり削除しますが、無効化は削除しないため、アップデートがあれば更新が行われます。しかし、機能を利用することはできなくなります。

▼図6　language-babelのインストール例



理想のコーディング環境とその構築

ここまでAtomの基本的な操作を解説してきましたが、ここからはAtomを使った理想のコーディング環境とその構築方法を考えていきたいと思います。

Atomなど拡張可能なエディタは鍛えれば鍛えた分だけ、快適な編集が行えるようになり、生産性が向上します。ぜひここでいろいろな機能を学んで活用していってください。

なお、本稿ではJavaScriptのコードをベースにして解説していきます。しかしながら、AtomはもちろんJavaScriptだけが得意というわけではなく、数多くのプログラミング言語で、ここで解説している内容が適用できますのでご心配なく。

自動補完とスニペット

まずは、自動補完とスニペットを強化していきましょう。この2つは、タイプ数を軽減するだけでなく、タイプミスをなくす効果もあるため、予期せぬエラーを未然に防ぐことにもつながります。

自動補完を強化する

Atomに自動補完機能を提供しているauto

注8）URL https://atom.io/packages/language-babel

complete-plusパッケージは、プロバイダと呼ばれる追加パッケージよって補完機能を強化できるようになっています。

代表的なプロバイダはAutocomplete Providers[注9]のページにまとめられていますので、興味のある機能を見つけたらインストールして試してみると良いでしょう。筆者のオススメのプロバイダを**表1**にまとめておきますので、まずはこちらからインストールしてみてはいかがでしょう。

スニペットを強化する

スニペットは「snippet」でパッケージ検索を行うと、特定の言語やフレームワークなどスニペットを集めたパッケージが大量にヒットします。希望どおりのスニペットを持ったパッケージが見つかれば、それをインストールします。

もし見つからない場合は、自分でスニペットを作成します。[Atom]メニューの[Snippets…]を選択すると、~/.atom/snippets.csonファイルが開きます。こちらに次のようなスニペットを定義して保存すると、**図7**のように補完候補に追加したスニペットが表示され利用できるようになります。

```
'*': # スコープ名
  'My Snippet': # スニッペット名
    'prefix': 'mysnippet' # 接頭辞
    'body': 'My first snippet' # スニペット
```

より高度なスニペットの定義方法については、Atomマニュアルの Snippets[注10]を確認しましょう。

シンタックスハイライトと自動インデント

シンタックスハイライトと自動インデントはコードの視認性を向上し、文法ミスなどを軽減してくれます。Atomは標準で多くの言語をサポートしていますが、もし標準でサポートされていない言語を使用する場合は、パッケージを追加することで対応させることができます。

また、標準サポートしている言語であっても、たとえばReactライブラリのJSXような、別の言語仕様を持つ場合も追加パッケージによるサポートが必要となるでしょう。

言語パッケージを追加する

実際に筆者が利用している言語パッケージを**表2**に紹介します。もし必要そうだと思ったら実際にインストールして試してみてください。

文字コード、改行コード、インデントなどの自動修正

最近の開発プロジェクトでは、文字コード、改行コード、インデントなどはプロジェクトご

注9) URL https://github.com/atom/autocomplete-plus/wiki/Autocomplete-Providers

▼表1 オススメの自動補完プロバイダ

名前	機能
autocomplete-paths	補完候補にパスを表示する
autocomplete-emojis	補完候補に絵文字を表示する
autocomplete-en-en	補完候補に英単語を表示する

▼図7 追加したスニペットの利用

注10) URL http://flight-manual.atom.io/using-atom/sections/snippets/

▼表2 オススメの言語パッケージ

名前	内容
language-diff	.diffや.patchファイルをサポート
language-docker	Dockerfileや.dockerignoreファイルをサポート
language-babel	ES2016、React、GraphQLなど先進的なJavaScriptの文法をサポート
language-nginx	Nginxの設定ファイルをサポート
language-asciidoc	AsciiDocファイルをサポート

とに規約（コーディングスタイル）が用意され、統一が図られています。しかし、人間が規約の違いを意識しておくのは、開発の本質と異なる部分で、余計な苦労を生じさせるため、何か別の方法で解決する方法が理想的です。

プロジェクトによって解決方法は異なりますが、ここでは、EditorConfig[注11]を使った方法を紹介します。

EditorConfigを導入する

EditorConfigはコーディングスタイルの定義ファイル（.editorconfig）と、テキストエディタプラグイン（Atomの場合はeditorconfig[注12]パッケージ）を組み合わせて利用します。すると定義ファイルを用意するだけで、対応しているエディタを利用している人は、自動的に規約に沿ったコーディングが行えます。

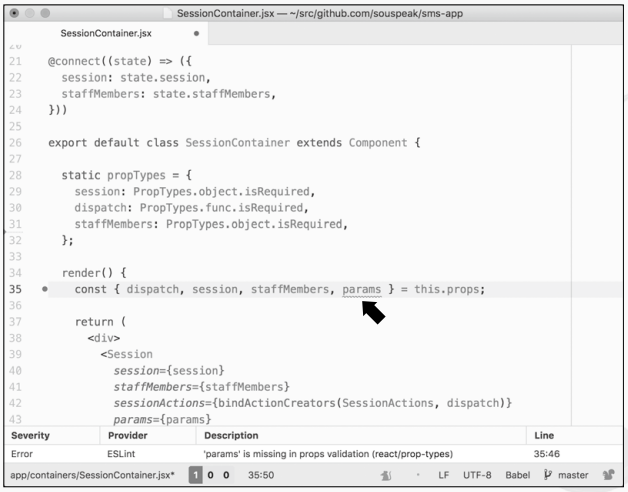
Atomでは、editorconfigパッケージをインストールすると、.editorconfigファイルの存在するプロジェクトを編集するとき、自動的に定義ファイルが利用され、コーディングスタイルの統一が図られるようになります。ほかに悪影響を与えることもないため、何か特別なことがない限りはインストールしておくと良いでしょう。

なお、編集途中で定義ファイルを変更した場合は、コマンドパレットから「EditorConfig: Fix File」コマンドを実行すると、修正が反映されます。

文法チェックと自動修正

筆者の経験上、エンジニアの中には文法ミスを残したまま、うっかりコミットを行ってしまう人がいます。筆者としては、こういった行為は経験の大小ではなく、開発環境に問題があると見ています。

▼図8　linter-eslintによる文法チェック



そこで、筆者が提案する方法は、エディタに文法チェッカを導入することと、それらを用いた自動修正機能をエディタから利用することです。Atomでは、linterパッケージの導入によって対応可能です。

linterを導入する

linter[注13]はautocomplete-plusと同様に、プロバイダによって対応する言語（文法チェッカ）を追加できるようになっています。たとえば、ESLintを使ったチェックを行いたい場合、linter-eslint[注14]をインストールしましょう。初回インストール時には、依存パッケージのインストールを確認するダイアログが表示されるので［Yes］を押してインストールしましょう。

linter-eslintをインストールすると、ESLintが使われているプロジェクトで、**図8**のように文法チェックが行われるようになります。linter-eslintは、ESLintを利用した自動修正機能をファイル保存時に実行する機能が用意されているため、パッケージ設定からこちらを有効にするだけで、ファイルの自動修正が行われるようになります。

注11）URL http://editorconfig.org/
注12）URL https://atom.io/packages/editorconfig
注13）URL https://atom.io/packages/linter
注14）URL https://atom.io/packages/linter-eslint

ほかにもRubocop[注15]を利用して自動修正を行ってくれるrubocop-auto-correct[注16]というパッケージもありますので、Rubyを書く人は試してみてください。

このように、Atomに自動修正機能を追加すると、たとえば、文字列のシングルクオートやダブルクオート、行末セミコロンの有無など、言語やコーディグスタイルによって異なる仕様などを、エディタが自動的に修正してくれるようになるため、うっかり間違ったままコミットすることもなくなります。

筆者推奨オススメパッケージ

ここまで、Atomを使った理想の環境について解説しましたが、残りは、ここまでで紹介しきれなかった便利なパッケージを紹介していきます。

コード全体のプレビュー——minimap

minimap[注17]はペインの右側に編集中のファイルの縮図を表示してくれるパッケージです。ファイル全体を俯瞰して確認できるようになります。

このminimapはプラグインを組み合わせることによって、より便利になります。たとえば、Gitの差分をハイライトするminimap-git-diff[注18]や、Linterのエラー個所をハイライトするminimap-linter[注19]などがあり、これらを組み合わせると**図9**のような表示になります。

これらのパッケージを組み合わせることで、変更個所やエラー個所がより見つけやすくなります。

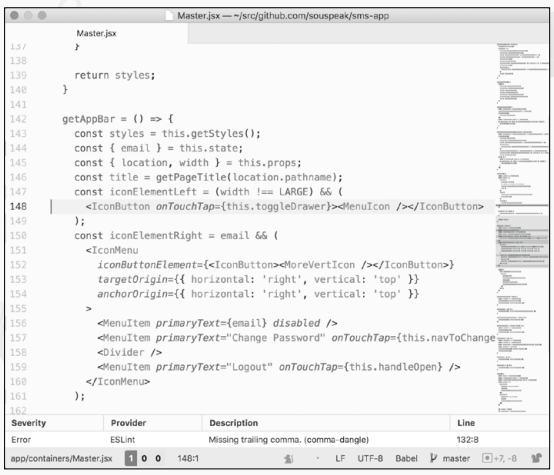
カラーコードのハイライト表示——pigments

pigments[注20]はさまざまカラーコードの文字列に対応する色の背景色を付けてくれるパッケージです。こちらを利用すると、カラーコードの文字列を見るだけで、どのような色か判別できるようになります。また、先ほど紹介したminimap用にminimap-pigments[注21]というプラグインも用意されていて、組み合わせると**図10**のような表示になります。

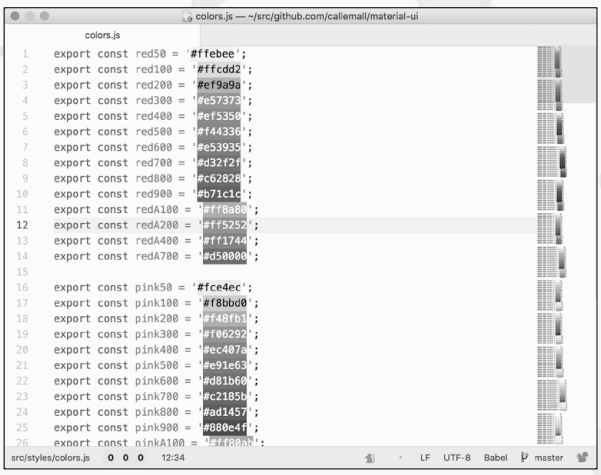
pigmentsは、カラーコードを色付けするだけのパッケージではありません。カラーコードを相互変換する機能も用意されており、右クリックメニューの［Pigments］から［Convert to RGB］などの

注15) URL http://batsov.com/rubocop/
注16) URL https://atom.io/packages/rubocop-auto-correct
注17) URL https://atom.io/packages/minimap
注18) URL https://atom.io/packages/minimap-git-diff
注19) URL https://atom.io/packages/minimap-linter
注20) URL https://atom.io/packages/pigments
注21) URL https://atom.io/packages/minimap-pigments

▼図9 minimap



▼図10 pigments



▼表3　git-plusの代表的なコマンド

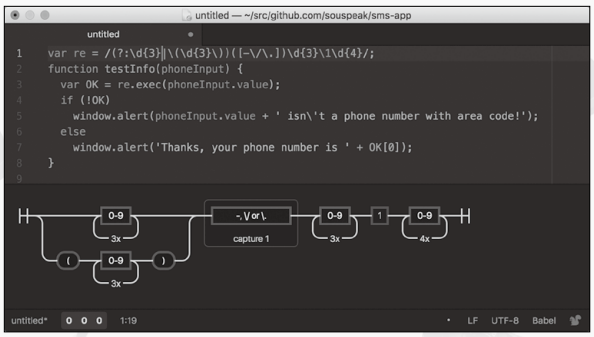
コマンド名	機能
Git Plus: Log	選択したログの詳細を表示する
Git Plus: Diff (All)	カレントバッファの差分を表示する（Allの場合はすべて）
Git Plus: Add (All)	カレントバッファのファイルをステージする（Allの場合はすべて）
Git Plus: Commit	コミットメッセージを書くためのペインを表示して、保存するとコミットする
Git Plus: Add And Commit	カレントバッファをステージしてコミットする（それ以前にステージしていたファイルも対象）
Git Plus: Commit Amend	1つ前のコミットを修正する
Git Plus: Reset	ファイルをすべてアンステージする

メニューを選択することで、16進数からRGBなどへの変換が可能となっています。

正規表現解析——regex-railroad-diagram

regex-railroad-diagram[注22]はカーソルが正規表現の上にあるとき、記述されている正規表現を解析してウィンドウ下部に処理の流れを路線図のような形で図示してくれるパッケージです（**図11**）。

▼図11　regex-railroad-diagram



正規表現は記号が多いため理解するのに時間がかかる場合がありますが、このパッケージを利用することで直感的に正規表現を理解できるようになります。

Git操作——git-plus

AtomはGitをサポートしていますが、あくまで現在ブランチを表示したり切り替えたり、差分を可視化したりするだけでコミットなどリポジトリの操作はできません。そこでAtomからGit操作を行いたい場合はgit-plus[注23]をインストールしましょう。git-plusを導入すると、ステージングやコミットなど基本的なGit操作のすべてをAtomから行えるようになります。

git-plusは多くのコマンドが用意されているので、よく使われるものを**表3**に整理しました。基本的には「Git Plus: Add」と「Git Plus: Commit」を利用すると、Atom上でコミットが行えます。

おわりに

さて、ここまでAtomについて解説してきましたが、いかがでしたでしょうか。

日々、コーディングを行うプログラマにとって、エディタは一番大事な仕事道具と言えます。そのため、エディタの機能や設定をメンテナンスすることは、将来にとって大きな投資になるでしょう。

本特集では、Atom以外にも素敵なエディタが紹介されていますので、もし時間が許すようであれば、すべてを試して比較してみることをオススメします。この記事がきっかけとなり、これからのエンジニアライフがより良いものになることを願っています。SD

注22）URL https://atom.io/packages/regex-railroad-diagram
注23）URL https://atom.io/packages/git-plus

Part 4

Visual Studio Code × C#

Webアプリ開発で体験

ゼロからはじめるVisual Studio Code

ソフトウェア開発環境「Visual Studio」を提供するマイクロソフトが、オープンソースで公開して人気を集めているエディタがVisual Studio Codeです。マルチプラットフォーム対応の柔軟性を持ちつつVisual Studioで培ってきた技術が盛り込まれています。Webアプリ開発をとおして、その使いやすさと充実の機能を体験してください。

Author 戸倉 彩（とくら あや）
日本マイクロソフト㈱
テクニカルエバンジェリスト
Twitter @ayatokura
Blog https://blogs.msdn.microsoft.com/ayatokura/

マイクロソフトの「Visual Studio Code」とは

どうすればプログラミング初心者がもっと楽しく、気軽にコードを書くことができるでしょう？　お気に入りの環境は当然ですが、軽量かつ高速な自分好みのエディタが快適に使えることが開発効率を上げるために必要と考えます。本稿ではそんな願いを叶える選択肢の1つとして、マイクロソフトが無償で提供しているデスクトップアプリケーション「Visual Studio Code（ビジュアル スタジオ コード）」を紹介します。

Visual Studio Codeの登場の背景

マイクロソフトは、1990年代より「Microsoft（マイクロソフト） Visual Studio（ビジュアル スタジオ）」という、ソフトウェアを開発するための開発環境をWindows版のソフトウェアとして提供し続けており、今年で20周年を迎えました（**図1**）。近年は、スマートフォンやIoTデバイスの普及やクラウドサービスの利用拡大などから、開発者に求められる開発環境も変化をし続けています。

Visual Studioの開発ビジョンとして「Visual Studio 1つですべての開発者があらゆるアプリケーションを開発」できることを掲げています。それを実現すべく、数年前から開発コード「Monaco（モナコ）」によってInternet ExploreのF12[注1]ツールや、Visual Studio Online“Monaco”と呼ばれるHTMLベースのエディタ機能を開発し、提供してきました。GitHub社の「Electron」を採用したことで、WindowsだけでなくMacやLinuxのクロスプラットフォーム環境でもVisual Studioの機能が使える軽量かつ高速なコードエディタのアプリケーションとして生まれ変わりました。約1年の一般公開ベータ期間を経て、昨年4月に「Visual Studio Code」バージョン1.0を正式版として、無料でダウンロード提供を開始しました[注2]。

世界中が注目するコードエディタ

このVisual Studio Codeの開発を率いているのは『オブジェクト指向における再利用のためのデザインパターン』（ソフトバンククリエイティブ刊）の著者、The Gang of Four（GoF）の一人として知られているエリック・ガンマという人物です。2015年にVisual Studio Codeをオープンソース化することを発表した際には、

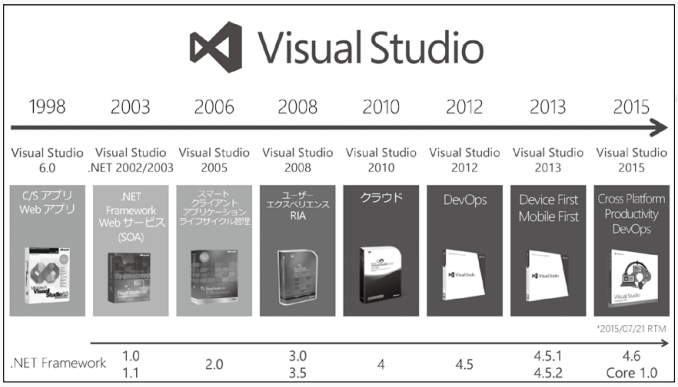
注1）Webページの作成とデバッグをサポートする開発ツール。URL https://docs.microsoft.com/en-us/microsoft-edge/f12-devtools-guide
注2）URL https://code.visualstudio.com/

本人がセッションに登壇し、その場でGitHubを操作したことでネットでも大きな話題として取り上げられました。現在も引き続き、“最も人気のあるコードエディタ”を目指して、オープンソースプロジェクトとしてVisual Studio Codeの開発を支えています。

昨年末にアメリカで開催されたマイクロソフト主催の開発者向けオンラインイベント「Connect();// 2016」では、Visual Studio Codeをグローバル規模でアクティブに利用している月間ユーザ数が100万を突破したことが紹介され、成長スピードが加速していることが明らかになりました。

最近ではGoogleとコミュニティによって開発されているAngular JSの開発において、TypeScriptに加えてVisual Studio Codeが活用されているなど新しい流れも出てきました。

▼図1　Visual Studioの歴史



Visual Studio Codeの特徴と魅力

高機能エディタ

Visual Studio Codeは単にマルチプラットフォームで動作するエディタというだけでなく、Visual Studioが備える高度な機能を提供しています。たとえば、シンタックスハイライトや対応括弧の強調／移動機能はもちろん、対応言語においては従来のVisual Studioでも人気のあるIntelliSenseやPeek表示、デバッグ機能など、IDEと同等の機能を使用できます。バージョン管理システムとしてはGitを採用し、エディタ内でcommitやpush、pullなどの作業を実行できます。加えてターミナル機能も備えており、bashやPowerShellなどを呼び出せるため、コマンドライン実行もVisual Studio Code上で完結できます。

拡張機能による機能強化

標準で多数のプログラミング言語をサポートしており、拡張機能を追加することで100以上もの言語に対応することが可能です。ほかにも「Visual Studio Marketplace」にてデバッグ機能やスニペット、テーマなどに役立つ3,000近い数の拡張機能が公開されています。必要な機能のみを選んで導入できるため重宝されています。

オープンソース化

Visual Studio Codeは、MIT Licenseに基づいてGitHubに公開されているオープンソースのコードエディタです[注3]。Visual Studio Codeは次の2つのビルドを提供しています。これから初めて使ってみる方にはStable版がお勧めです。

Stable Build

毎月更新される新機能とバグ修正が施された安定しているビルドです[注4]。

Insiders Build

いち早くVisual Studio Codeの最新機能を試

注3）厳密には、このリポジトリはVisual Studio Codeのベースとなるoss-Codeのリポジトリとなり、最終的にoss-Codeにコンポーネントの追加など調整を行ったものがマイクロソフトから提供されるVisual Studio Codeとなります。そのため、Visual Studio Code自身にはMIT Licenseは適用されず、MICROSOFT SOFTWARE LICENSE TERMS（マイクロソフトソフトウェア ライセンス条項）が適用されます。

注4）2017年4月21日現在のVisual Studio Code（Stable版）の最新バージョンは1.11.2。

すことができる最新のプレビュービルドです。こちらはmasterからビルドされ毎日更新されます。

Visual Studio Codeは何よりもコミュニティの方々のフィードバックが不可欠と考えており、問題の報告や機能追加のリクエスト、Pull Requestなど誰もがVisual Studio Codeを改善できるようにGitHubに集約されています[注5]。

GitHub Octoverse 2016[注6]によると、Visual Studio Codeのcontributor数は世界6位という盛り上がりと大きな広がりをみせています。

Visual Studio Codeのインストールと実行方法

Visual Studio Codeをインストールするためのシステム要件は**表1**のとおりです。インストールは簡単です。Windows、Mac、Linuxそれぞれの手順を記しておきます。

インストール方法（Windows）

1. ブラウザからダウンロード[注7]にアクセスし、「Windows」ボタンをクリックする
2. インストーラ（VSCodeSetup.exe）のダウンロードが始まる
3. ダウンロードしたインストーラを実行し、セットアップウィザード画面に従って操作する
4. インストール完了
5. Windows 10の場合、タスクバーなどにピン止めしておくとそこから実行できる

インストール方法（Mac）

1. ブラウザからダウンロード（同Windows）にアクセスし、「Mac」ボタンをクリックする
2. ZIPファイル（VSCode-darwin-stable.zip）のダウンロードが始まる
3. ダウンロードされたZIPファイルを解凍する
4. 解凍したフォルダの中にある「Visual Studio Code.app」を「アプリケーション」フォルダへコピーする
5. インストール完了
6. （オプション）コマンドから起動できるようにするためには、Visual Studio Codeのコマンドパレット（[表示]－[コマンドパレット]）に「> Shell Command」と入力し、Shell Commandをインストールする
7. LaunchpadやDockからVisual Studio Codeを実行できる

インストール方法（Linux）

1. ブラウザからダウンロード（同Windows）にアクセスし、利用しているLinuxに合わせて「.deb」または「.rpm」ボタンをクリックする
2. ファイルのダウンロードが始まる
3. ダウンロードしたファイルを解凍する
4. 解凍したフォルダの中にある「Code」を実行するとアプリケーションが実行できる[注8]

注5） URL https://github.com/Microsoft/vscode
注6） URL https://octoverse.github.com/
注7） URL https://code.visualstudio.com/Download

注8） 詳細は「Running VS Code on Linux」サイトを参照。
URL https://code.visualstudio.com/docs/setup/linux

▼表1 Visual Studio Codeをインストールするためのシステム要件

ハードウェア（推奨）	
CPU	1.6GHz以上
メモリ	1GB以上のRAM
プラットフォーム（検証済みの環境）	
Windows*	7、8.0、8.1、10（32ビットおよび64ビット）
Mac	OS X Yosemite
Linux（Debian系）**	Ubuntu Desktop 14.04、Debian GNU/Linux 7
Linux（Red Hat系）**	Red Hat Enterprise Linux 7、CentOS 7、Fedora 23

* Visual Studio Codeには.NET Framework 4.5.2が必要なため、Windows 7の場合は.NET Framework 4.5.2がインストールされていることを確認すること
** GLIBCXX version 3.4.15以降、GLIBC version 2.15以降

インストールが完了したらコマンドラインからも起動できます（共通）。

1. ターミナルを起動する
2. 次のコマンドを実行すると、そのディレクトリからVisual Studio Codeが起動する

```
> code .
```

Visual Studio Codeの画面構成と設定

Visual Studio Codeの画面構成は**図2**のようになります[注9]。各エリアは自由に表示／非表示を切り替えられます。

① メニュー：Visual Studio Codeを利用するためのメニューが表示される
② アイコン：「エクスプローラー」「検索」「ソース管理」「デバッグ」「拡張機能」をクリックで呼び出せる
③ サイドバー：エクスプローラーのアイコンの選択時は、そのディレクトリ配下に格納されているフォルダやディレクトリが表示される
④ エディタ：コーディングするためのエディタを表示
⑤ 統合ターミナル：メニューまたはショートカットキーで呼び出されたターミナル画面
⑥ ステータスバー：左側にはGit連携状況、エラー表示、右側にはエディタが取り扱っているコードのプログラミング言語、追加情報などが表示される

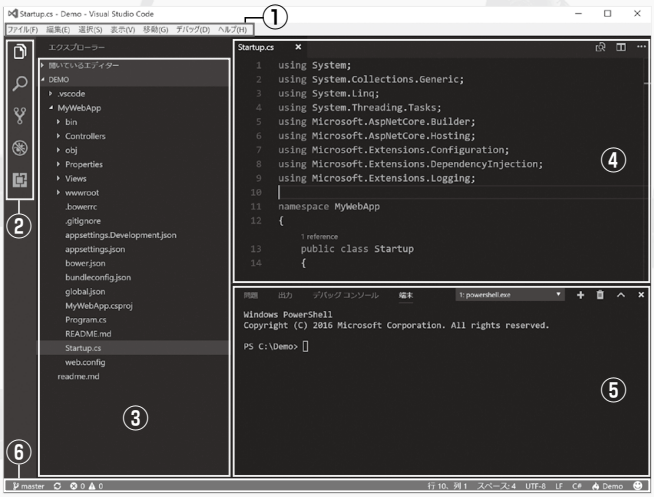
注9）バージョンによってアイコンデザインや配置される場所が異なる場合がある。

Visual Studio Codeの設定（settings.json）

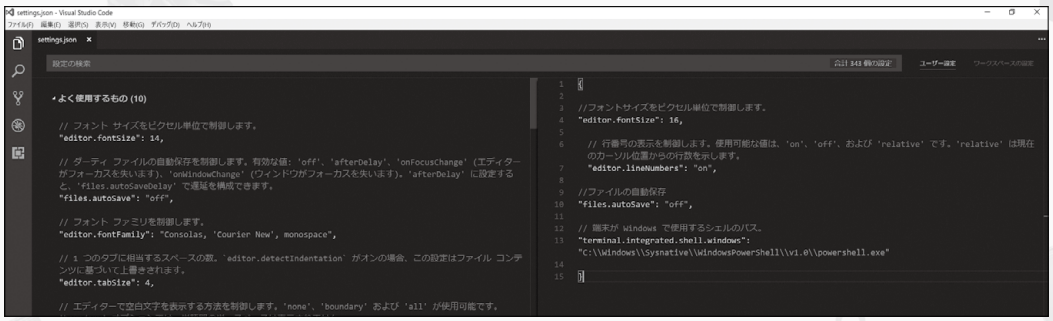
Visual Studio Codeのエディタ部分の表示や各種設定は、メニューの［ファイル］－［基本設定］－［設定］からsettings.jsonファイルを編集して行います[注10]。JSONにあまり馴染みがない方でも、**図3**のように画面の左側にデフォルト設定が表示され、右側にカスタム設定の内

注10）Settings.jsonファイルはプラットフォームによって異なる場所に格納されている。
- Windows　%APPDATA%¥Code¥User¥settings.json
- Mac　$HOME/Library/Application Support/Code/User/settings.json
- Linux　$HOME/.config/Code/User/settings.json

▼図2　Visual Studio Codeの画面構成



▼図3　settings.jsonファイルの編集画面



容を追加で記述していくため、容易に取り扱うことができます。

たとえば、フォントサイズやアイコン表示サイズをカスタマイズしたい場合は、settings.jsonの中で数値を変更して記述します。ほかにもWindows環境でターミナルからコマンドラインではなくPowerShellを呼び出したい場合には、使用するシェルのパスを指定しておくことで有効になります。

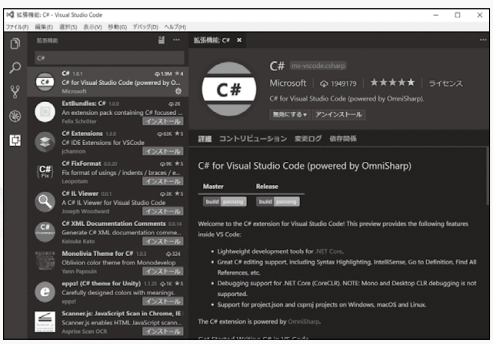
Visual Studio Codeの表示言語(locale.json)

Visual Studio Codeのメニューなどの表示言語を日本語以外に設定したい場合、コマンドパレットに「言語」と入力し、「言語を構成する(Configure Language)」を選択することで言語の設定を指定できます(**図4**)。設定はlocale.jsonファイルで、先ほどと同様にJSONで記述します。

```
locale.json
{
    // VSCodeの表示言語を定義します。
    "locale": "jp"
}
英語に変える場合はjpをenに変更してファイルを保存します
```

▼図4 コマンドパレットからの言語設定呼び出し

▼図5 C#の拡張機能



執筆時現在、日本語、中国語、フランス語、ドイツ語を含む10ヵ国語にローカライズされ、グローバル規模で使われています。

Visual Studio Codeではじめての C#

C#のプログラミングをVisual Studio Codeで行う場合は、「C#」の拡張機能のインストールが必要です。このように、エディタではサポートされないプログラミング言語を扱うための機能を容易に追加できることもVisual Studio Codeの特徴であり、さらに、その言語をサポートするさまざまな拡張機能も公開されています(**図5**)。

なお、ここでの操作はWindows版を使って説明しています。基本的には同じですが、それぞれのプラットフォームで少しずつ違いがありますので適宜読み替えてください。

ASP.NET Coreを用いたWebアプリケーション開発

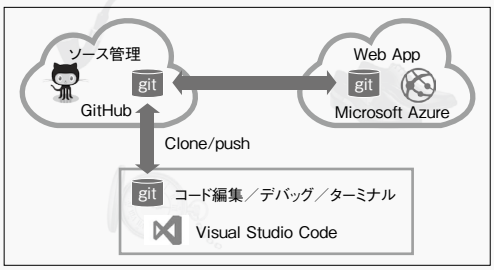
ASP.NET Coreは、マイクロソフトがオープンソースとして公開している環境で、C#を用いてWebアプリケーションを開発できます。本稿ではYeoman[注11]ジェネレータを使ってWebアプリケーションを作成してみます(**図6**)。

事前準備

開発をはじめる前に、Node.js、npm、Yeoman

注11) フロントエンドプロジェクトを簡単に生成できるツール。
URL http://yeoman.io/

▼図6 ASP.NET Core Webアプリケーション開発



をインストールして環境を整えます。

1. 最新Node.js 7.9.0をサイト[注12]からダウンロードしてインストールする
2. Visual Studio Codeを起動しなおし、メニューの［表示］－［統合ターミナル］[注13]で統合ターミナルを起動（bash）。次のコマンドを実行してYeomanとBowerをインストールする

```
> npm install -g yo bower
```

3. Yeomanを使用して.NET Coreアプリケーションを作成するため、ASP.NETジェネレータをインストールする（もし権限に関するエラーが発生した場合は、実行するコマンドをsudoを使って実行する）

```
> npm install -g generator-aspnet
```

ASP.NET Coreを用いたWebアプリケーション開発環境の構築

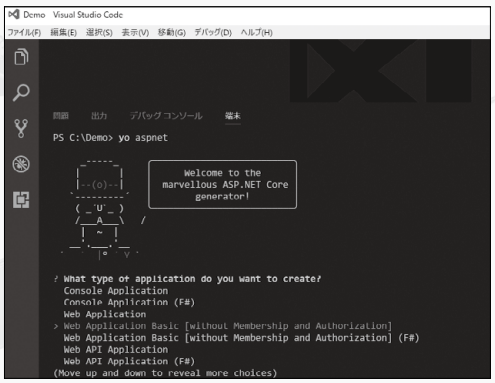
続いて、C#拡張機能のインストールと作業ディレクトリを作成します。

1. 各プラットホームに応じて.NET core SDK 1.0.1[注14]をサイト[注15]からダウンロードしてインストールする[注16]
2. 「Demo」ディレクトリを作成する

統合ターミナルを使ったYeomanによるWebアプリケーションの作成

それでは、Visual Studio Code上で統合ターミナルを使って開発を体験してみましょう。Webアプリケーションの作成をサポートするYeomanを呼び出して開発していきます。

▼図7　Yeomanを使ったジェネレート画面



1. Visual Studio Codeの統合ターミナルを起動する
2. コマンドラインで作業ディレクトリに移動後（たとえばDemoの場合は、cd Demo）、ASP.NETジェネレータを実行する

```
> cd Demo
> yo aspnet
```

3. 起動すると、どのようなアプリケーションを作成したいか選択肢が表示される（図7）。カーソルを使って、「Web Application Basic [without Membership and Authorization]」プロジェクトを選択し Enter を押す
4. 「Which UI Framework would you like to use? (Use arrow keys)（どのUIフレームワークを利用したいか）」という選択肢が表示される。「Bootstrap」を選択し Enter を押す
5. 「What's the name of your ASP.NET application? (WebApplicationBasic)」と入力をうながされるので、ASP.NETアプリケーション名を入力し（今回はMyWebApp）、 Enter を押す
6. 自動的にプロジェクトが生成される
7. 次のコマンドを実行してリストアを行う（図8）

```
> cd MyWebApp
> dotnet restore
```

8. 次のコマンドを順番に実行してビルドする（図9）

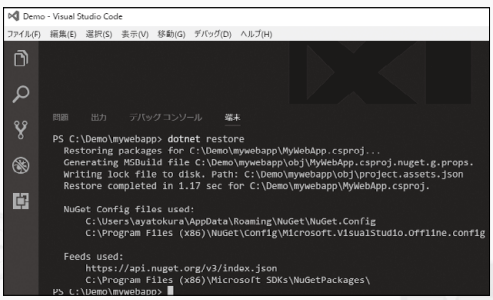
注12）URL https://nodejs.org/ja/
注13）ショートカット Ctrl -@ を覚えておくとスムーズに統合ターミナルを呼び出せる。英語キーボードの場合は Ctrl -`
注14）Windowsの場合はdotnet-dev-win-x64.1.0.1というファイル名ですが、1.0と1.1が収められた配布物のバージョンとして認識しておくと良いでしょう。
注15）URL https://www.microsoft.com/net/download/core
注16）最新のYeomanを使う際に.NET core SDKの最新バージョンを使っていない場合、整合性が取れない事象が発生する可能性があるため、今後も最新のものを推奨。

```
> dotnet build
> dotnet run
```

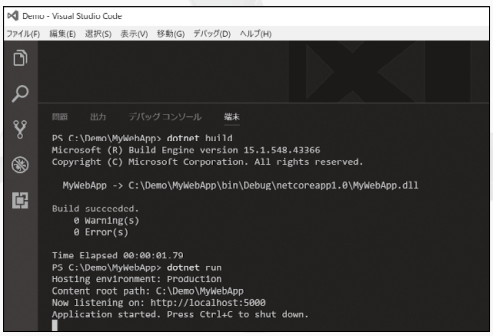
9.「Now listening on: http://localhost:5000」でポート番号が5000番であることを確認する
10. Visual Studio Codeで「MyWebApp」フォルダを開き、F5キーを押してデバッグ実行する
11.「.NET Core」を選択する（図10）
12. デバッグ実行時に必要なlaunch.jsonファイルは自動生成される（図11）
13. Webブラウザより、http://localhost:5000へアクセスする

これで図12のように、ASP.NET Coreのサイトが表示されていたら成功です。

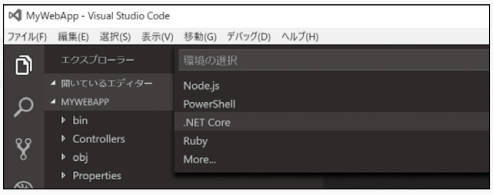
▼図8　リストアの実行



▼図9　ビルドの実行



▼図10　デバッガの選択



クラウド環境にWebアプリケーションを公開

事前準備

GitとMicrosoft Azureが利用できる環境を整えます。

Git環境のインストール

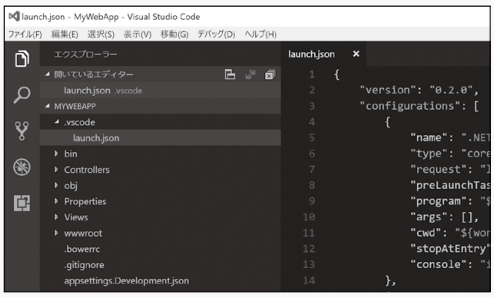
Gitのインストール方法は次のとおりです。

1. Gitサイト[注17]にアクセスする
2. 対象OSのインストーラを実行する
3. インストーラの画面に従ってインストールを進める（ここではデフォルトの設定にします）
4. インストールが完了したら続いて初期設定を行うために、Visual Studio Codeの統合ターミナルから次のコマンドを順番に実行する

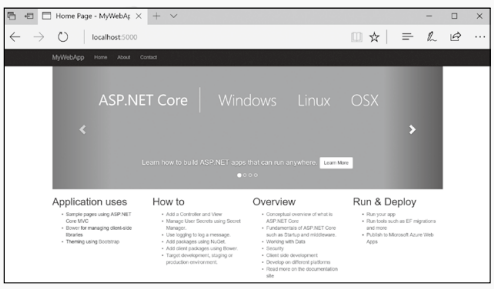
```
> git config --global user.name “(Your Name)”
> git config --global user.email “(Your Email)”
```

注17）URL https://www.git-scm.com/download/

▼図11　自動生成されたlaunch.json



▼図12　作成したASP.NET Coreのサイト



以上でGitの設定は完了です。

Gitによるソース管理

インストールしたGit環境をVisual Studio Codeから使う手順を示します。

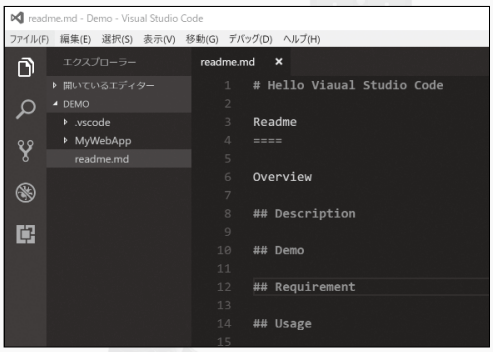
1. Visual Studio Codeでreadme.mdを書き保存する（図13）
2. 画面左側のバーから上から3番目のGitアイコンを選択する[注18]（図14）
3. Gitリポジトリの作成を完了後、readme.mdを何か編集して上書き保存する
4. Gitアイコンに変更が加わったファイル名（ここではreadme.md）が表示される
5. ファイル（readme.md）を右クリックして「Stage Change」を選択する（図15）
6. 「STAGED CHANGES」の配下にreadme.mdが表示される（図16）
7. 「ソース管理: GIT」の下に配置された入力ボックス（図16参照）に任意のコミットメッセージを入力する
8. 入力が完了したらチェックマークをクリックするか、Ctrl-Enterでコミットを実行する
9. 正常終了するとステージされていた変更が保存され、「STAGED CHANGES」のアイテムが空になる
10. ファイルをGitリポジトリ管理下におくことで、Visual Studio Codeは再びreadme.mdを編集した際に、左側に青いインジケータを表示し、変更された行が可視化される

GitHubにpushする

ローカルのGitリポジトリがVisual Studio Codeから使えるようになったので、これをGitHubに上げて管理することにします。なお、GitHubへの登録についてはここでは割愛します。

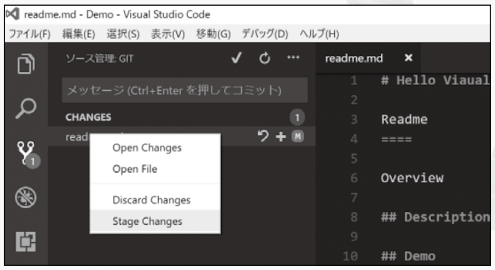
注18）開いているディレクトリにGitリポジトリがない場合は、このディレクトリに対してGitリポジトリを作成する。

▼図13　readme.mdの編集



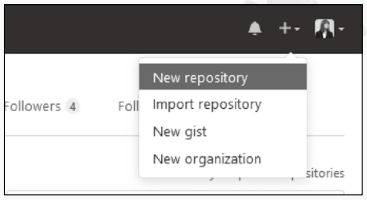
▼図14　Gitアイコンの選択

▼図15　readme.mdをステージング状態にする



▼図16　ステージング状態になったreadme.md

▼図17　GitHubに新しいリポジトリを作成（Webブラウザ）



1. GitHubにログインし、右上の［+］ボタンで「New repository」を選択して新しいリポジトリを作成する（図17）
2. 任意のリポジトリ名を入力し、「Create repository」を選択する
3. リポジトリの作成が完了し、Quick setup画面が表示される（図18）
4. Visual Studio Codeの統合ターミナルからコマンドラインでgit pushする（図19）

```
> git remote add origin “(your URL)”
> git push -u origin master
```

5. GitHubで作成したサイトを更新するとreadme.mdを含めたファイルやフォルダがアップロードできたことを確認できる（図20）

Microsoft Azureアカウントの作成

もしMicrosoft Azureのアカウントをお持ちでない場合は、開発者向け無償プログラム「Visual Studio Dev Essential[注19]」を登録してください。クラウド環境を月間2,500円相当分、12ヵ月無償で利用いただくことが可能です。

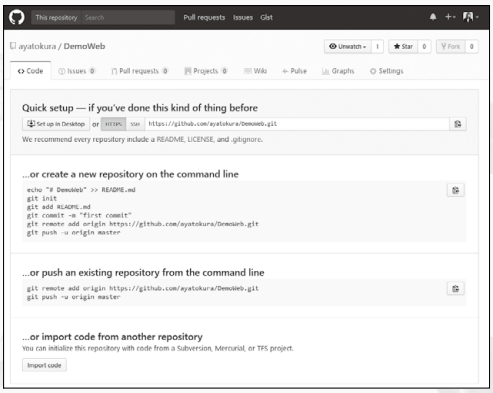
Webアプリケーションをクラウドにホスティング

Microsoft AzureへWebアプリケーションをホスティングする手順を示します。

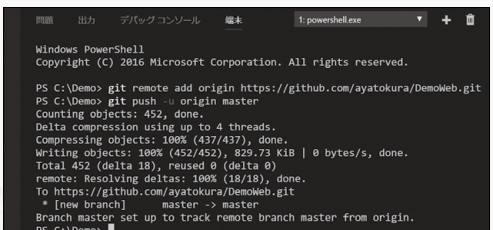
1. ブラウザからAzure管理ポータル[注20]へアクセスし、ログインする
2. 左上の［+］をクリックし［新規］－［Web＋モバイル］－［Web App］の順に選択する
3. Webアプリを作成するために必要な情報を入力する（図21）
   - アプリ名：任意（デフォルトのドメイン名は***.azurewebsites.net）

注19）URL https://www.visualstudio.com/ja/dev-essentials/
注20）URL https://portal.azure.com

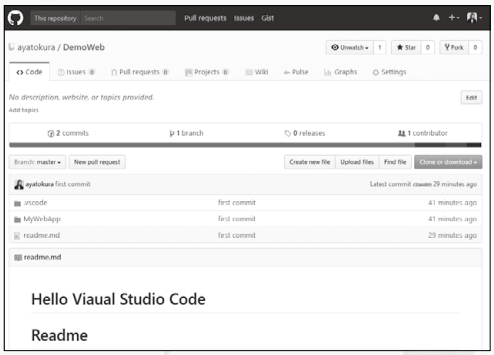
▼図18　新しいリポジトリが作成された



▼図19　Visual Studio Codeからのgit push



▼図20　GitHubでアップロードの確認（Webブラウザ）



▼図21　Microsoft Azureでの新規Webアプリ登録

［参考URL］ Visual Studio Code：https://code.visualstudio.com/
Visual Studio Code Documentation：https://code.visualstudio.com/docs

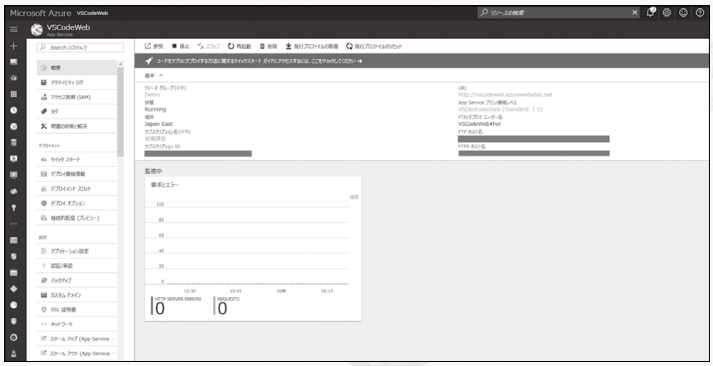
・リソース グループ：任意（Azure上で管理するためのグループ名）
・App Serviceプラン：Japan EastまたはJapan Westを指定

4. 入力が完了したら［作成］を押す
5. Webサイトをホスティングするための領域が確保できる概要が表示される（図22）。左側のメニューから「デプロイオプション」を選択する
6. 「ソースの選択」で「GitHub」を選択する（図23）
7. 「承認」では、AzureがGitHubアカウントにアクセスするためのアクセス許可を求めるので、「承認する」を選択する（図24）
8. 今回はパフォーマンステストに関する設定をスキップし、［OK］ボタンを押す（図25）
9. GitHubとの連携が行われると自動的にAzure上にWebアプリケーションのホスティングが行われる
10. Webアプリケーションを配置したURLにアクセスする（手順の3番で作った***.azurewebsites.netがURLになる）
11. ページが表示されたら成功（図26）

いかがでしたでしょうか。Webアプリケーションを作りながらVisual Studio Codeの優れた機能にも自然に触れることができたのではないかと思います。これを機にぜひVisual Studio Codeを極めて、プログラミングに挑戦して活躍する一歩を踏み出してみませんか。SD

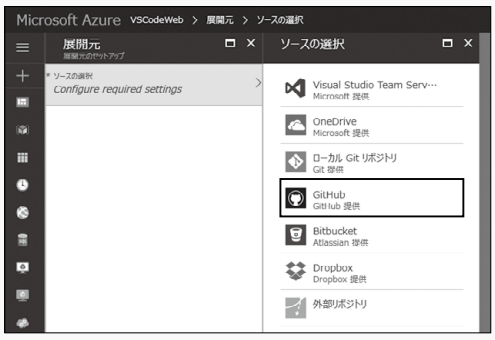
▼図24　GitHubアカウントへのアクセスを承認

▼図22　確保されたホスティング領域

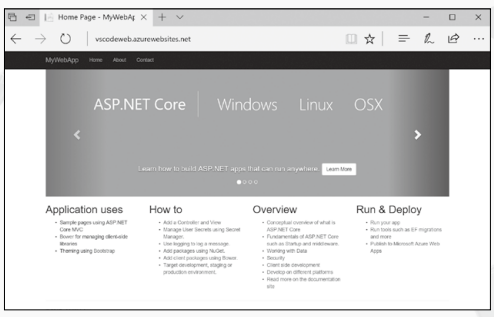


▼図25　［OK］ボタンを押す

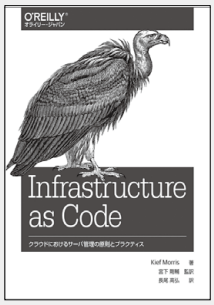
▼図23　GitHubの選択



▼図26　Azureに配置したWebアプリケーションが表示されたことを確認



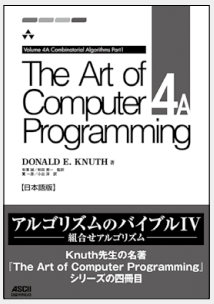
［参考URL］GitHub/Visual Studio Code：https://github.com/Microsoft/vscode
Twitter/Visual Studio Code：https://twitter.com/code



Infrastructure as Code

Kief Morris 著、宮下 剛輔 監訳、長尾 高弘 訳
B5変形判／352ページ
3,600円＋税
オライリー・ジャパン
ISBN＝978-4-87311-796-6

Infrastructure as Codeと言えば、Ansibleのような「サーバの構成管理ツール」とTerraformなどの「低レイヤのインフラの定義ツール」がすべてのように思っていたが、違っていた。この本ではそれらに加え、AWSといった「スクリプトを使って動的に構築できるインフラ」、そして「それらをサポートするツール群」の計4要素によって構成される領域だという。本書ではおもにそれらの概要、選び方、構築パターンを解説している。TerraformやChefのコードが例として使われるが、特定のツールに依存した内容ではない。「OpenStackでChefを使っていても、AWSでPuppetを使っていても、ベアメタルでAnsibleを使っていても」と書かれているとおり、インフラをコードで管理したいと考える幅広い読者に有用な1冊である。

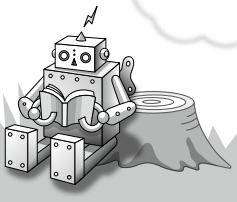


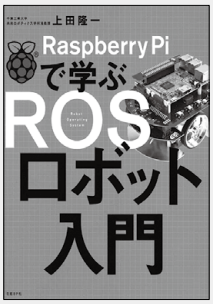
The Art of Computer Programming Volume 4A

Donald E. Knuth著
B5判／888ページ
4,800円＋税
アスキードワンゴ
ISBN＝978-4-04-893055-0

TEXの開発者ドナルド･クヌース先生によるアルゴリズムの名著の第4巻。この巻は分冊となっており、本書はその最初の1冊である「4A」とナンバリングされている。今回もページ数888と辞典なみのボリュームであるが、含まれている章は「第7章　組合わせ探索」だけである（しかも、その第7章がすべて収まっているわけではない）。内容としては、組み合わせアルゴリズムを学ぶために使う、Boole演算や二分木決定図といったデータの適切な表現方法を解説したあと、「すべての順列」「すべての組み合わせ」などを生成するアルゴリズムについて論じている。ちなみに本書序文によれば、組み合わせアルゴリズムはクヌース先生にとって「我が最愛のプログラミング」らしく、第4巻はまだまだ長大になるかもしれない。

SD BOOK REVIEW

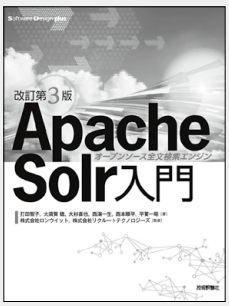




Raspberry Piで学ぶROSロボット入門

上田 隆一 著
B5変形判／344ページ
3,700円＋税
日経BP社
ISBN＝978-4-8222-3929-9

Raspberry Piを搭載したロボットを題材に、ロボットのプログラミングを学ぶ本。初めはLinuxコマンドで擬似ファイルシステムに値を書き込むことでロボットのLEDを点灯したり、車輪を回したりする。これはLinuxでの通常のファイル操作とほぼ同じため、じつに簡単。自分にもできそうな気がしてくる。その後はROS（Robot Operating System）を使った本格的な開発に入る。ROSの基本操作と、Pythonでの実装方法を学び、総仕上げにセンサとモータを連携させてロボットを走らせる。本の随所にQRコードが載っており、そこにアクセスすれば、本書で実装するロボットの動作を動画で確認できる。走るロボットの動画は見ていて楽しいが、同じことが自分にもできると思うと、本書の内容を実践してみたい意欲が湧いてくる。



Apache Solr入門

打田 智子、大須賀 稔、大杉 直也、西潟 一生、西本 順平、平賀 一昭 著／株式会社ロンウィット、株式会社リクルートテクノロジーズ 監修
B5変形判／392ページ
3,800円＋税
技術評論社
ISBN＝978-4-7741-8930-7

Apache Solrとは、Apache Luceneを各種スクリプト言語で開発できるようしたオープンソースプロダクトである。今回の改訂第3版は最新版Solr6への対応が主である。本書はSolrコミュニティの中から生まれ、ともに成長しているのが特徴である。それは執筆陣の変化に表れている。初版から大きく入れ替わり、若いエンジニア達が執筆を引き継ぎアップデートしているのだ。Solrによる検索サービスの構築はいろいろな企業ですでに導入・実践されており、特殊なものではなくなっている。しかしながら、検索サービスの原理を学ぶことは、自然言語解析やリコメンデーションなど非常に奥深いテーマに向き合うことができるので、プログラマにとって大事なスキルになっている。ぜひ、定番検索技術を本書で学んでいただきたい。

第2特集

多用途に使いこなせ、コードが読みやすく保守しやすい 今すぐはじめる Python

GoogleやFacebookでも採用されているPythonは、現在最も旬な言語ではないでしょうか。Pythonでデータ分析や統計処理、また人工知能や機械学習などを扱った事例も多くなってきています。

本特集では、なぜ今Pythonが使われ、どんなことに使えるのかにスポットを当てます。第1章では、Pythonの特徴、2系と3系のバージョンの違いとコーディングスタイルなどを紹介します。第2章では、言語仕様とライブラリについて紹介します。第3章では実際に機械学習を実行して、ツールやライブラリの使い方、学習と予測のやり方を体験します。

本特集を読んで、Pythonをはじめてみませんか。

Special Thanks：@iktakahiro、@shimizukawa、@terapyon

第1章

今から始めるならPythonがお勧め！

なぜPythonを導入するとお得なのか

Author 鈴木 たかのり(すずき たかのり)
一般社団法人PyCon JP／株式会社ビープラウド
Twitter @takanory

Pythonとは

Python(パイソン)は世界中で広く使用されているプログラミング言語です。Pythonという名前は、イギリスのコメディグループ「モンティ・パイソン」からとられています。

Pythonはフリーかつオープンソースのソフトウェアとして、コミュニティで開発が進められています。現在、Pythonのソースコード、ロゴなどの知的財産権は、PSF(Python Software Foundation)という非営利団体で管理されています。

Pythonは多くのプログラミング言語と同様、Windows、macOS、Linuxなどさまざまなプラットフォーム上で動作します。

豊富なライブラリ

Pythonには正規表現、数学関数、通信プロトコル、GUIフレームワークといった豊富な標準ライブラリが用意されています。これらの標準ライブラリは、Pythonをインストールすると使用できるようになります。これは、必要な機能が最初からそろうように、という考えに基づくもので、Pythonの「バッテリー付属(Batteries Included)」という哲学を反映しています。

Python標準ライブラリのリファレンスマニュアル[注1]は日本語化されたものがWebで参照できます(**図1**)。

サードパーティー製のパッケージも豊富に提供されており、Webアプリケーションフレームワークや機械学習などのパッケージがあります。これらのパッケージはPyPI[注2]というサイトで共有されており、簡単にインストールして利用できます(**図2**)。

ライブラリの使い方、パッケージのインストール方法については第2章で説明します。

▼図1　Python標準ライブラリのリファレンスマニュアル

Pythonでできること

Pythonは、標準ライブラリだけでテキスト解析やネットワーク通信、ファイル操作など、さまざまなことができます。Pythonで作成されているツールやフレームワークを導入することによって、さらに大規模なプログラミングも可能になります。

表1にPython製の著名なツール、フレームワークについて一部を紹介します。

たとえば、次のようなことがPythonの

注1) https://docs.python.jp/3/library/
注2) the Python Package Index：パイピーアイ。https://pypi.python.org/pypi

プログラムだけで実現できます。

・Webページからスクレイピングしてデータを抜き出す
・抜き出したデータを機械学習やディープラーニングで分析
・分析した結果をWebページで公開
・サーバの環境を構築
・ライブラリのドキュメントを作成

環境構築、データ解析からWebシステムまで、Pythonという共通のプログラミング言語で、さまざまなことが実現できるのが大きなメリットです。お勧めのフレームワーク、ツール、ライブラリなどの情報がまとまっているサイトとしてAwesome Python[注3]があります。

機械学習関連のライブラリ、ツールについては第3章でさらに詳しく解説します。

読みやすいコード

Pythonの設計思想に「シンプルで読みやすいコードが書けること」があります。同じ動作をするプログラムは似たようなコードになることが設計思想に組み込まれており、このため、コードの保守がしやすいと言えます。

Pythonの特徴の1つに、インデントがあります。**if**や**for**などの制御構文の影響する範囲を、括弧で囲むのではなくインデント（通常はスペース4つ）で定義します。次のコードは1から100までのFizz Buzz[注4]を返すコードです。インデントで構造を表すことにより、コードを見たときに**if**や**for**の範囲がどこなのかがわかりやすくなるという利点があります。

```
for num in range(1, 101):
    if num % 15 == 0:
        print('Fizz Buzz')
    elif num % 3 == 0:
        print('Fizz')
    elif num % 5 == 0:
        print('Buzz')
    else:
        print(num)
```

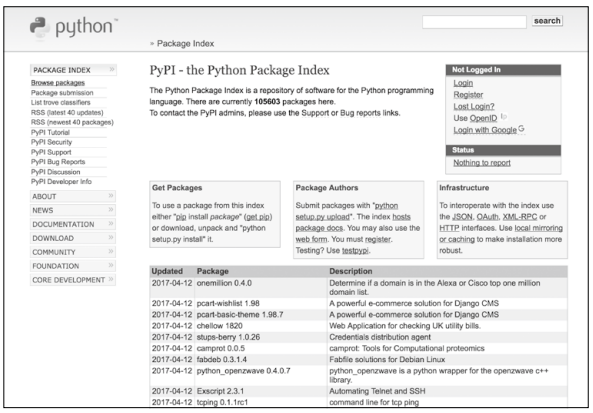
PEP8：コーディングスタイルガイド

Pythonではプログラムのどこにスペース、空行を入れるべきか、変数名や関数名の付け方などのコーディングスタイルが定義されています。このコーディングスタイルはPEP8[注5]（PEPについては後述）という名前で呼ばれています。コーディングスタイルに従うことにより、ほかの人のコードも読みやすくなります。

また、コーディングスタイルをチェックするためのツールも用意されており、

注3）https://awesome-python.com/

▼図2　PyPI



注4）https://ja.wikipedia.org/wiki/Fizz_Buzz
注5）https://www.python.org/dev/peps/pep-0008/
日本語訳　http://pep8-ja.rtfd.io/

▼表1　Python製の著名なツール、フレームワーク

名前	内容	URL
scrapy	Webページをスクレイピングするフレームワーク	https://scrapy.org/
scikit-learn	機械学習ライブラリ	http://scikit-learn.org/
TensorFlow	ディープラーニングライブラリ	https://www.tensorflow.org/
Django	Webアプリケーションフレームワーク	https://www.djangoproject.com/
Ansible	サーバの構成管理ツール	http://docs.ansible.com/
Sphinx	ドキュメンテーションビルダ	http://sphinx.readthedocs.io/ja/stable/

PyPIで提供されています(**表2**)。

Pythonでプログラムを書く場合は、ぜひこれらのツールを活用してください。一通りチェックしてくれるflake8を使用するのがお勧めです。

PEP：Pythonの拡張提案

Pythonの設計思想「シンプルで読みやすいコードが書けること」を保つために、拡張機能についてコミュニティで議論して決定するしくみがあります。それがPEP(Python Enhancement Proposal：Python拡張の提案)です。

Pythonの言語仕様を追加、変更するためには、PEPドキュメントの提出が必要です。PEPを作成して、コミュニティに提案、議論ののちに、採用・不採用が判断されます。

表3のようなPython 3.6の新機能も、PEPによって提案されて採用されています。

Pythonの歴史とバージョン

Pythonは1991年にバージョン0.9.0がリリースされ、25年以上の歴史があります。現在はPython 2系と3系が並行して開発されており、それぞれ最新版は2.7.13と3.6.1です。

Python 2系とPython 3系

Python 2系と3系には、文法の変更を含むいくつかの違いがあります。Python 2系のサポートは2020年で終了することがPEP 373[注6]に明記してあります。また、Python 2.8がリリースされないことがPEP 404[注7]で決定しています。今からPythonでプログラミングを始めるならばPython 3を使用しましょう。

多くのフレームワーク、ライブラリはPython 3に対応しているので、問題なく使えます。**図3**の画像はPython 3 Wall of Superpowers[注8]というサイトの画面で、ライブラリがPython3に対応しているかをダウンロードの多い順に掲載しています。Python 3に対応しているものを緑で、対応していないものを赤で表示しています。見てのとおり、現在はほとんどがPython 3に対応しています。

Python 2系と3系の違い

ここではPython 2系と3系の違いについていくつか紹介します。

printが文(2系)から関数(3系)に変更

文字列を出力する`print`は2系では文でしたが3系では関数となりました。次のコードのように、書き方が異なります。

```
Python 2系
>>> print 'Hello Python 2!'
Hello Python 2!
```

注6) https://www.python.org/dev/peps/pep-0373/
注7) https://www.python.org/dev/peps/pep-0404/
注8) https://python3wos.appspot.com/

▼表2　コーディングスタイルをチェックするためのツール

パッケージ名	用途
pycodestyle	コーディングスタイルに合っているかをチェックするツール
autopep8	プログラムをコーディングスタイルに合っているように書き換えるツール
pyflakes	不要なimportや未使用の変数の検出など、pycodestyleで扱っていないプログラミング上の問題を検出するツール
flake8	pycodestyleに加えて使用されていない変数など、論理的なチェックを行うツール

▼表3　PEP提案により採用されたPython 3.6の新機能

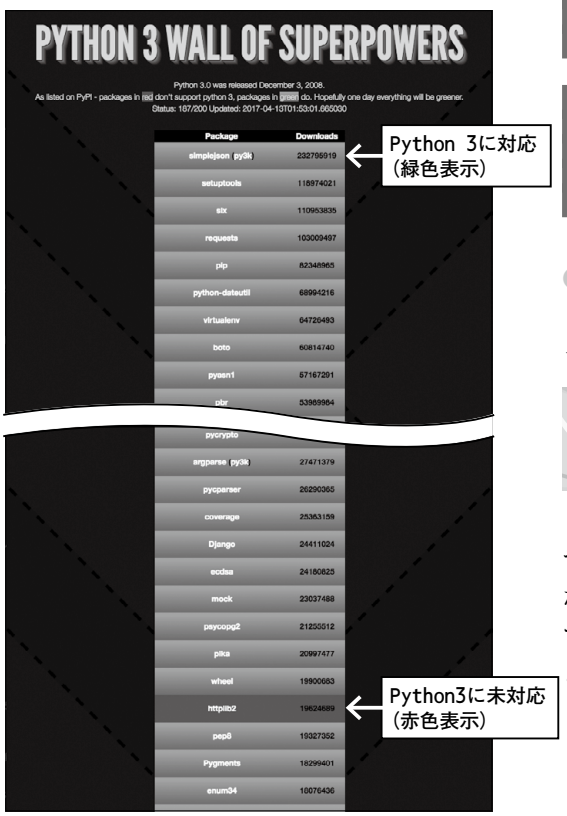
PEP番号	機能	URL
PEP 498	フォーマット済み文字列リテラル	https://www.python.org/dev/peps/pep-0498/
PEP 515	数値リテラル内のアンダースコア	https://www.python.org/dev/peps/pep-0515/
PEP 506	標準ライブラリにSecretsモジュールを追加	https://www.python.org/dev/peps/pep-0506/

```
Python 3系
>>> print('Hello Python 3!')
Hello Python 3!
```

● 文字列がUnicode文字列に統一

Python 2系では文字列には、非Unicode文字列(str型)とUnicode文字列(unicode型)の2種類が存在しました。そのため、str型とunicode型の変換には文字コードを指定したencode/decodeが必要でした。

▼図3　Python 3 Wall of Superpowers



Python 3系のstr型はUnicode文字列のみとなったため、日本語の扱いで文字化けなどに遭遇することが少なくなりました。

● 整数同士の割り算が浮動小数点を返す

Python 2系では割り算の結果は、端数を切り捨てた整数(int型)でしたが、3系では浮動小数点数(float型)になりました。

```
Python 2系
>>> 1/3
0
>>> 8/2
4
```

```
Python 3系
>>> 1/3
0.3333333333333333
>>> 8/2
4.0
```

● 標準ライブラリの再構成

標準ライブラリの名前や構成が変更されています。一部を**表4**にまとめました。

まとめ

本章ではPythonの特徴として、コードの書きやすさや豊富なライブラリについて紹介しました。また、2系と3系のバージョンの違いについても説明し、今からならPython 3系を使うべきという話もしました。

次章ではPython 3の導入と基本について解説します。SD

▼表4　Python 2系から3系で変更された標準ライブラリの名前や構成

Python 2系	Python 3系	説明
BaseHTTPServer、SimpleHTTPServer、CGIHTTPServer	http.server	基本的なHTTPサーバ機能のライブラリはhttp.serverに統合された
urllib、urllib2、urlparse	urllib.request、urllib.parse、urllib.error	URLへのアクセスや構文解析のためのライブラリはurllib.*に再構成された
StringIO、cStringIO	io	テキストI/OはioモジュールのStringIOクラスに統合された
ConfigParser	configparser	設定ファイルのパーサは名前が変更された

第2章 Pythonの導入と基本
文法からライブラリまで

Author 鈴木 たかのり(すずき たかのり)
一般社団法人PyCon JP／株式会社ビープラウド
Twitter @takanory

インストール

まずは、Pythonをインストールしましょう。最新版は3.6.1です。WindowsやmacOSの場合は、Python.orgのDownload Pythonページ[注1]からインストーラをダウンロードしてインストールしましょう。Windowsの場合はインストール時にPATHの設定を行うかどうかを聞かれます。チェックをしてPATH設定することをお勧めします。

Ubuntu 16.04にPython 3.6をインストールするには、deadsnakes[注2]というPythonの各バージョンがインストールできるPPAを**図1**の手順で設定し、インストールします。

Windowsではコマンドプロンプトで**python**、macOSではターミナルで**python3**、Ubuntuでは**python3.6**と入力するとPythonが起動します。引数なしでPythonを実行すると**対話モード**で起動します(**図2**)。対話モードでは**>>>**プロンプトが表示され、Pythonコードの入力を受け付けます。

実際にPythonでプログラムを作成する場合は、プログラムコードをファイル(拡張子が**.py**)に保存して実行しますが、対話モードは簡単なPythonコードの動作確認をするときに便利です。以降の言語仕様の解説では対話モードで確認する前提で記述するので、プロンプトの**>>>**があります。

注1) https://www.python.org/downloads/
注2) https://launchpad.net/~fkrull/+archive/ubuntu/deadsnakes "deadsnakes

▼図1　Ubuntu 16.04にPython 3.6をインストール

```
$ sudo add-apt-repository -y ppa:fkrull/deadsnakes
$ sudo apt-get -y update
$ sudo apt-get -y install python3.6 python3.6-dev python3.6-venv
```

言語仕様

ここからはPythonの言語仕様について説明します。

データ型

基本的なデータ型と演算子などでの操作方法について説明します。

数値と数値演算

数値型には整数(int)、浮動小数点数(float)、複素数(complex)の3種類があります。数値演算には**表1**のような演算子が使えます。整数には精度の制限がないので、Pythonでは非常に大きな桁の整数も扱えます。基本的にint同士の演算の結果はint、floatの場合はfloatとなりますが、割り算のみint同士でも結果がfloatになります。虚数には**j**をつけます。

▼図2　Pythonを対話モードで起動

```
$ python3
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 / 3
0.3333333333333333
>>> [x ** 2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
>>>
>>>
```

```
>>> 1 + 1
2
>>> 1.1 + 1.1
2.2
>>> 1 + 2j
(1+2j)
```

型変換は組み込み関数の**int()**、**float()**で行えます。

```
>>> int(1.1)
1
>>> float(1)
1.0
```

● 文字列(str)と文字列操作

文字列はシングルクォーテーション(')またはダブルクォーテーション(")で囲んで定義します。たとえば'を含む文字列を定義するときに"を囲むと、エスケープをしなくてよいので便利です。

```
>>> 'Hello World'
'Hello World'
>>> "I'm takanory"
"I'm takanory"
>>> 'I\'m takanory'
"I'm takanory"
```

長い文章を定義する場合は三重引用符('''または""")で文字列を囲みます。エディタでの改行がそのまま文字列の改行となるので、長い文字列を定義する必要があるヘルプメッセージやテンプレートを定義する場合に便利です。なお、対話モードで複数行の入力を行う場合は、対話モードのプロンプトが**...**に変わります。また、**print()**関数を使用すると文字列に変換された値が出力されるため、改行コードは改行文字となります。

```
>>> longtext = '''long
... long
... text'''
>>> longtext
'long\nlong\ntext'
>>> print(longtext)
long
long
text
```

文字列への変換は、組み込み関数の**str()**を使用します。

```
>>> str(1)
'1'
>>> int('1')
1
```

Pythonの文字列には非常にたくさんの便利なメソッドがあります。**表2**に、いくつかよく使う文字列メソッドを紹介します

次のコードは文字列メソッドの使用例です。

```
>>> 'spam ham eggs'.find('ham')
5
>>> 'spam ham eggs'.endswith('ham')
False
>>> 'spam ham eggs'.replace('ham', 'spam')
'spam spam eggs'
>>> 'spam ham eggs'.split()
['spam', 'ham', 'eggs']
```

▼表1　数値演算

演算	結果
x + y	xとyの和
x - y	xとyの差
x * y	xとyの積
x / y	xとyの商
x // y	xとyの商を切り下げたもの
x % y	x / y の剰余
x ** y	xのy乗

▼表2　よく使われる文字列メソッド

メソッド	説明
str.find(sub)	subの出現する位置を返す
str.startswith(prefix)	文字列が指定されたprefixで始まるかをTrue/Falseで返す
str.endswith(suffix)	文字列が指定されたsuffixで終わるかをTrue/Falseで返す
str.strip()	文字列の先頭と末尾の空白文字列を削除する
str.replace(old, new)	文字列中のoldをnewに置換する
str.split(sep)	文字列を指定した区切り文字で分割してリスト(後述)を返す。引数を指定しない場合は空白文字で分割する

・文字列のformat()メソッド

書式を指定した文字列を生成するには、format()メソッドを使用します。format()メソッドでは波括弧({})の中に引数として指定した値が出力されます。数字を書くと引数で指定した順番が使用されます。

```
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{1}, {2}, {0}'.format('a', 'b', 'c')
'b, c, a'
```

引数の名前でも指定できます。

```
>>> '{fn} {ln}'.format(fn='takanori', ln='suzuki')
'takanori suzuki'
```

リスト(list)とタプル(tuple)

複数の値をまとめるデータ型について説明します。まずはリスト(list)です。リストは複数の値を配列形式で格納します。リストは全体を角括弧([])で囲んで、カンマ(,)区切りで値を指定します。リストには値を追加したり、値を削除したりできます。

```
>>> test_list = ['spam', 'ham', 'eggs']
>>> test_list.append(1)
>>> test_list.remove('ham')
>>> test_list
['spam', 'eggs', 1]
```

Pythonには、リストと似たデータ型のタプル(tuple)があります。タプルは全体を丸括弧(())で囲みます。タプルはイミュータブル(不変)なため、後からデータは変更できず、変更しようとすると次のようにエラーが発生します。

```
>>> test_tuple = ('spam', 'ham', 'eggs')
>>> test_tuple.append(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

・リストとタプルの基本的な操作

リストとタプルはいくつかの操作ができます(表3)。

実際に動作している例は次のとおりです。このコードではリストを使っていますが、タプルでもまったく同じように動作します。なお、タプルに対してのスライスの結果はタプルを返します。

```
>>> test_list = ['spam', 'ham', 'eggs']
>>> test_list[0]
'spam'
>>> test_list[-1]
'eggs'
>>> test_list[0:2]
['spam', 'ham']
>>> test_list[1:]
['ham', 'eggs']
>>> len(test_list)
3
>>> 'spam' in test_list
True
>>> test_tuple = ('spam', 'ham', 'eggs')
>>> test_tuple[0:2]
('spam', 'ham')
```

インデックスとスライスの指定にはマイナスの数値も使用できます。マイナスを指定した場合は後ろから数えます。また、スライスで値を省略すると最初から、または最後までとなります。

・文字列(str)もシーケンス型

文字列もシーケンス型(順序のある要素の集まり)のため、リスト、タプルと同じ操作が行えます。

▼表3　リストとタプルの操作

名前	書き方	内容
インデックス	data[0]	任意の位置の値を返す(この例では0番目(最初)の要素)
スライス	data[0:2]	任意の範囲の値を返す(この例では0番目から2番目の前(0番目と1番目)の要素)
len()関数	len(data)	項目数を返す
in演算子	'param' in data	指定した要素の存在をTrue/Falseで返す

```
>>> txt = 'python3'
>>> txt[2]
't'
>>> txt[-2]
'n'
>>> txt[0:3]
'pyt'
>>> len(txt)
7
>>> '2' in txt
False
```

辞書(dict)

辞書(dict)はキーと値をセットで持つデータ型で、RubyではHashと言われているものです。辞書は波括弧(**{}**)で囲んで、キーと値をコロン(**:**)でつなげて定義します。キーを指定して値の上書きや追加ができます。

なお、辞書データの順番は不定のため、順番に依存した処理を行いたい場合はcollectionsモジュールのOrderedDictを使用してください。

```
>>> test_dict = {'spam': 100, 'ham': 50}
>>> test_dict['spam']
100
>>> test_dict['ham'] = 80
>>> test_dict['eggs'] = 20
>>> test_dict
{'spam': 100, 'ham': 80, 'eggs': 20}
```

辞書もリストなどと同じような操作ができます。**in**演算子はキーに対しての存在チェックを行います。

```
>>> len(test_dict)
3
>>> 'spam' in test_dict
True
```

辞書のキーには不変な値のみを設定できます。文字列以外にタプルもキーに指定できます。リストはキーに指定できません。

集合(set)

集合はリストのように値のみを持ちますが、順番はなく同じ値は1つしか存在できません。集合は波括弧(**{}**)で囲んで、要素をカンマ区切りで記述します。**len()**関数と**in**演算子も今までと同様に動作します。

```
>>> test_set = {'spam', 'ham', 'spam', ↵
'eggs', 'spam'}
>>> test_set
{'eggs', 'ham', 'spam'}
>>> len(test_set)
3
>>> 'ham' in test_set
True
```

集合はいわゆる数学の集合なので、2つの集合の和集合、排他的論理和などを計算するのに便利です(**表4**)。

制御構文

ここでは繰り返し、条件分岐などの制御構文について説明します。

for文

繰り返しのfor文は、「**for** 変数 **in** 繰り返し可能オブジェクト**:**」というように書きます。繰り返し可能オブジェクトにはリスト、タプル、文字列などが指定できます。変数に1つずつ値が入って、ブロック内のコードが繰り返し処理されます。

```
>>> for data in ['spam', 'ham', 'eggs']:
...     print(data)
...
spam
ham
eggs
```

・ブロック

Pythonではfor文、if文(後述)などの影響がおよぶ範囲(ブロック)をインデントで指定します。括弧で閉じたり、endなどの文を書いたりはしません。

▼表4　集合の和集合、排他的論理和の演算子

演算子	意味
set | other	和集合
set & other	積集合
set - other	差集合
set ^ other	排他的論理和

Pythonでは同じインデントが同じブロックであることが視覚的にわかるので、コードが見やすくなります。インデントは通常はスペース4つを使用します。正しくインデントされていないとIndentationErrorが発生するので注意してください。

```
>>> for data in ['spam', 'ham', 'eggs']:
... print(data)
  File "<stdin>", line 2
    print(data)
        ^
IndentationError: expected an indented ↵
block
```

・range()関数、enumerate()関数

任意の数値の範囲を繰り返したい場合は、**range()**関数を使用するのが便利です。

```
>>> for i in range(3):
...     print(i)
...
0
1
2
```

また、リストなどの値を取り出しつつ、その順番を取得したい場合は**enumerate()**関数を使用するのが便利です。

```
>>> for num, data in enumerate(['spam', ↵
'ham', 'eggs']):
...     print(num, data)
...
0 spam
1 ham
2 eggs
```

if文

条件分岐のif文は「**if 式:**」と書いて条件となる式が真(True)の場合に処理が実行されます。さらに条件を指定する場合は「**elif 式:**」と書きます。**else if**ではないのでご注意ください。式には**表5**のような比較演算子が使用できます。

ファイル入出力

ファイル入出力にはopen関数を使用します。日本語のファイルを扱う場合にはencodingを指定しましょう。read()メソッドでファイルの中身を読み込みます。最後にclose()メソッドでファイルを閉じるようにしましょう。

```
>>> f = open('spam.txt', encoding='utf-8')
>>> data = f.read()
>>> f.close()
```

ファイルを書き込む場合は**'w'**を、追記の場合は**'a'**をモードとして指定します。未指定の場合はデフォルト値として**'r'**(読み込み)が指定されています。**write()**メソッドでファイルに書き込みます。

```
>>> f = open('ham.txt', 'w', ↵
encoding='utf-8')
>>> f.write('スパム、ハム、エッグ\n')
10
>>> f.close()
```

with文

ファイルを開く場合に**with**文を使うと、ファイルの閉じ忘れがなくて便利です。

```
>>> with open('ham.txt', ↵
encoding='utf-8') as f:
...     data = f.read()
...
>>> print(data)
スパム、ハム、エッグ
```

関数

処理をまとめる関数を作成します。Pythonでは**def**キーワードで関数を宣言します。関数名は変数名と同様にsnake_caseで記述します。

次の例は2つの引数をとり、その2つの値を加算した結果を返す関数の宣言例です。関数の戻り値は**return**文で指定します。関数の定義

▼表5　比較演算子

比較演算子	意味
a == b	aとbが等しい
a != b	aとbが等しくない
a >= b	aがb以上
a > b	aがbより大きい
a <= b	aがb以下
a < b	aがbより小さい
a in b	aがbに含まれている
a is b	aとbのオブジェクトが同じ

でもインデントによるブロック構造が使用されています。

```
def function_name(arg1, arg2):
    return arg1 + arg2
```

キーワード引数

関数の引数は順番での指定だけでなく、引数の変数名を記述しての指定もできます。また、引数にはデフォルト値も指定できるのでうまく使いこなすと関数を呼ぶ側がシンプルに書けます。次の例は単純な足し算の結果を返す関数ですが、いろいろな引数の指定方法があります。

```
>>> def add(a=1, b=10):
...     return a + b
...
>>> add()        # 引数なし
11
>>> add(2)       # 第1引数のみ指定
12
>>> add(2, 20)   # 両方の引数を指定
22
>>> add(a=3)     # 引数aの値のみを指定
13
>>> add(b=30)    # 引数bの値のみを指定
31
```

可変長引数

引数の数が不定の場合は、引数の変数名にアスタリスク(*)をつけます。次の関数は、引数で与えられた数をすべて足した結果を返す関数です。なお、**args**には引数がタプル形式で入ります。

```
>>> def adds(*args):
...     print(args)
...     total = 0
...     for num in args:
...         total += num
...     return total
...
>>> adds(1, 2)
(1, 2)
3
>>> adds(1, 2, 3, 4, 5)
(1, 2, 3, 4, 5)
15
```

可変長引数はキーワード形式にも対応しています。その場合は引数の変数名に**を付けます。次の関数では受け取った引数全てを「キー:値」という文字列にして返しています。

```
>>> def keywords(**kwarg):
...     result = ''
...     for k in kwarg:
...         result += '{}: {}\n'.
format(k, kwarg[k])
...     return result
...
>>> print(keywords(a=1))
a: 1

>>> print(keywords(a='a'))
a: a

print(keywords(a='spam', b='ham',
c='eggs'))
a: spam
b: ham
c: eggs
```

モジュールとパッケージ

大規模なプログラミングでは複数のファイルで処理を分割します。Pythonではプログラムを書いたファイルをモジュールとしてほかのプログラムから使用します。

次のように2つの関数が書いてある**calc.py**というファイルを用意します。

```
def add(a, b):
    return a + b

def sub(a, b):
    return a - b
```

このファイルがあるディレクトリでPythonの対話モードを起動し、**import文**でモジュールを利用できるようにします。

```
>>> import calc
>>> calc.add(1, 2)
3
>>> calc.sub(1, 2)
-1
```

import文では次のように関数を直接インポートする書き方もあります。

```
>>> from calc import add
>>> add(3, 4)
7
```

標準ライブラリ

Pythonには、インストールしたらすぐに使える便利なモジュールが、標準ライブラリとして提供されています。

標準ライブラリは非常にたくさんあります。どんなモジュールがあるか、その使い方を知るには、Python標準ライブラリ[注3]のページを参照してください。ライブラリの一覧とその使い方が確認できます。ここではdatetimeとrandomの使い方を紹介します。

datetime

datetimeモジュールは日付、時刻を扱うための機能を提供します。詳細は「8.1. datetime — 基本的な日付型および時間型[注4]」を参照してください。

datetimeモジュールには、date(日付)、time(時刻)、datetime(日時)、timedelta(日時の差)などの機能があります。次のコードは現在日時を取得して、来年までの日数(254日)を取得しています。

```
>>> from datetime import datetime
>>> now = datetime.now()
>>> now.isoformat()
'2017-04-21T19:12:28.284149'
>>> nextyear = datetime(2018, 1, 1)
>>> delta = nextyear - now
>>> delta.days
254
```

random

randomモジュールは擬似乱数を生成します。詳細は「9.6. random— 擬似乱数を生成する[注5]」を参照してください。

randrange()関数で任意の範囲の数値を返します。choice()関数は要素を取り出し、shuffle()関数は中身をシャッフルします。

```
>>> import random
>>> random.randrange(100)
48
>>> lunch = ['カレー', 'うどん', '寿司']
>>> random.choice(lunch)
'うどん'
>>> random.shuffle(lunch)
>>> lunch
['うどん', '寿司', 'カレー']
```

例外処理

ファイルが存在しない、型変換に失敗したといったときには例外が発生します。例外処理を行いたい場合には**try文**を記述します。

以下の例ではファイルを開いていますが、ファイルが存在しない場合には**except節**に書いてある**print()**関数が実行されます。ファイルが存在する場合は何も実行されません。

```
>>> try:
...     f = open('spam.txt')
... except:
...     print('ファイルが開けませんでした')
...
ファイルが開けませんでした
```

exceptには例外の種類が指定できます。また、発生した例外を変数に与えて、例外の中身を確認できます。**図3**のコードはファイルを開いて中身を数値に変換するため、ファイルが存在しない場合とファイルの中身が数値の文字列では

注3) https://docs.python.jp/3/library/
注4) https://docs.python.jp/3/library/datetime.html
注5) https://docs.python.jp/3/library/random.html

▼図3　例外の中身を確認する

```
>>> try:
...     f = open('spam.txt')
...     num = int(f.read())
...     print('数値は {} です'.format(num))
... except OSError as err:
...     print('OSErrorが発生しました: {}'.format(err))
... except ValueError:
...     print('数値への変換に失敗しました')
OSErrorが発生しました: [Errno 2] No such file or directory: 'spam.txt'
```

ない場合に例外が発生します。この例ではファイルが存在しない場合のエラーが発生しています。

仮想環境と外部パッケージ

Pythonには、標準で便利なライブラリが提供されていますが、第1章でも説明したようにサードパーティ製の外部パッケージも便利なものが多数提供されています。ここでは仮想環境を作成して、安全に外部パッケージを使用する方法を説明します。

仮想環境(venv)

仮想環境とはなんでしょうか?

複数のPythonを使用したプロジェクトがあり、同じサードパーティ製パッケージだがプロジェクトごとにバージョンが異なるということがあります。1つのPython環境に複数バージョンのパッケージはインストールできません。そこで、Pythonの仮想環境を作成して、それぞれの環境にバージョンを指定してパッケージをインストールします。

仮想環境は次の手順で利用します。

①仮想環境を作成する
②仮想環境を有効化する
③外部パッケージをインストールする
④プログラムを作成して実行する
⑤仮想環境を無効化する

● 仮想環境の作成

仮想環境の作成はPythonの標準ライブラリであるvenvモジュールを使用します。以前はvirtualenvという外部パッケージをインストールしていましたが、最新のPythonに同様の機能が組み込まれました。

次のコマンドで仮想環境を作成します。envと書いているところには仮想環境に関連するファイルを作成するディレクトリ名を指定します。

```
$ python -m venv env
$ ls env
bin/      include/  lib/      pyvenv.cfg
```

● 仮想環境の有効化

仮想環境を有効化するには、さきほど作成した仮想環境にあるactivateというスクリプトを実行します。仮想環境を有効にすると、プロンプトに環境名が付きます。

```
$ . env/bin/activate
(env) $
```

● 外部パッケージをインストール

仮想環境に外部パッケージをインストールします。ここでは例としてRequestsというHTTP通信を行うためのパッケージをインストールします。パッケージのインストールにはpipコマンドを使用します。**pip install パッケージ名**を実行すると、指定したパッケージをPyPI[注6]からダウンロードしてインストールします。

なお、依存パッケージがある場合には、そのパッケージも併せてインストールされます。

```
(env) $ pip install requests
Collecting requests
  Using cached requests-2.13.0-py2.py3-none-any.whl
Installing collected packages: requests
Successfully installed requests-2.13.0
```

なお、バージョンを指定してインストールする場合は、次のように==でバージョン番号を指定します。

```
(env) $ pip install requests==2.12.5
```

● プログラムを作成して実行する

この仮想環境ではRequestsを利用したプログラムが作成できるようになりました。インストールしたパッケージは標準ライブラリと同様にimportして使用します。以下は対話モードでrequestsが正常にimportできることを確認しています。

注6) https://pypi.python.org/pypi

```
(env) $ python
>>> import requests
>>> r = requets.get('http://gihyo.jp')
>>> print(r.text[:10])
<!DOCTYPE
```

仮想環境の無効化

仮想環境を無効化するには**deactivate**コマンドを実行します。無効化されるとプロンプトが元の状態に戻ります。元の環境にはrequestsがインストールされていないため、importしようとするとエラーが発生します。

```
(env) $ deactivate
$ python
>>> import requests
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named ➥
'requets'
>>>
```

パッケージ一覧の保存

複数人で開発を行う場合は、それぞれの仮想環境で使用するパッケージのバージョンを合わせる必要があります。**pip freeze**コマンドを使用して、現在の仮想環境で使用しているパッケージの情報をファイルに書き出します。ファイル名は慣例として**requirements.txt**がよく使われます。

```
(env) $ pip freeze > requirements.txt
(env) $ cat reqirements.txt
requests==2.13.0
```

requirements.txtファイルをソースコードと合わせてバージョン管理します。**pip install -r requrements.txt**を実行することにより、同一バージョンのパッケージをインストールした環境が構築できます。

▼表6　おすすめのパッケージ

パッケージ名	用途
requests	扱いやすいHTTPクライアント
beautifulsoup4	HTML/XMLの解析
python-dateutil	日付関連の便利なユーティリティ
pytz	世界中のタイムゾーンデータ
paramiko	SSHプロトコル通信
pycrypt	ハッシュと暗号化
Pillow	画像編集
openpyxl	Excelを編集

▼表7　Web開発などに使用するフレームワークや便利なツール

パッケージ名	用途
django	Webフレームワーク
scrapy	Webスクレイピングのフレームワーク
sphinx	ドキュメント作成ツール
awscli	AWSを管理するコマンド
SQLAlchemy	データベースのORマッパー
pytest	テストフレームワーク

外部パッケージ

ここではPythonでの定番とも言える、いくつかお勧めのパッケージを紹介します(**表6**)。同様の機能が標準ライブラリで提供されているものもありますが、より便利なパッケージがある場合は、そちらを使用することをお勧めします。

Web開発などに使用するフレームワークや便利なツールについても紹介します(**表7**)。

なお、機械学習関連のパッケージについては第3章で紹介します。

Requests

先ほど例として用いましたが、Requests[注7]は人が使いやすいHTTPクライアントです。非常に人気で、HTTP接続を行う他のパッケージからよく利用されています。Requestsのインストールは**pip install requests**で行います。

次のコードはSoftware Designのトップページにアクセスしているところです。ステータスコードと取得したHTMLの先頭10文字を出力しています。

```
>>> import requets
>>> r = requests.get('http://gihyo.jp/➥
magazine/SD')
>>> r.status_code
200
>>> r.text[:10]
'<!DOCTYPE '
```

注7) http://docs.python-requests.org/en/master/

Beautiful Soup 4

Beautiful Soup 4[注8]はHTML/XMLのパーサーです。Requestsとセットで、簡単なWebスクレイピングによく利用されています。Beautiful Soup 4のインストールは`pip install beautifulsoup4`で行います。

次のコードはSoftware Designのトップページをパースし、最初はtitleタグの中身を出力しています。その次は`magazineTopOutline`というidの要素に特集などの情報が入っているので、先頭3件のタイトルを取得しています。

```
>>> import requests
>>> from bs4 import BeautifulSoup
>>> r = requests.get('http://gihyo.jp/magazine/SD')
>>> soup = BeautifulSoup(r.content, 'html.parser')
>>> soup.title.text
'Software Design | gihyo.jp … 技術評論社'
>>> outline = soup.find(id='magazineTopOutline')
>>> for title in outline.find_all(class_='title')[:3]:
...     print(title.text)
...
Linux入門(UNIXネットワーク編)
サービス改善につなげるドッグフーディング環境の作り方
いまから学ぶブロックチェーンのしくみ
```

注8) http://www.crummy.com/software/BeautifulSoup/

参考資料

Pythonを学んでいくうえでの参考となるサイトや書籍を紹介します。

Webサイト

Pythonは標準のドキュメントが充実しているので、チュートリアルやライブラリのリファレンスを参照するのがお勧めです(**表8**)。

書籍

Pythonに関する書籍をいくつか紹介します(**表9**)。

まとめ

駆け足でPythonの文法(データ型、制御構文、関数)から、標準ライブラリ、仮想環境、外部パッケージのインストール方法について解説しました。Pythonでのプログラムを始めるのに必要最低限のことは説明しましたが、クラス、デコレータ、リスト内包表記などPythonの便利な機能はまだまだたくさんあります。ぜひ、Webサイトや書籍で学びつつ、Pythonでのプログラミングを始めてみてください。そして、その成果をコミュニティなどで発表してくれることを期待します。SD

▼表8 チュートリアルやライブラリのリファレンス

サイト名	URL
Python 3.6 チュートリアル	http://docs.python.jp/3/tutorial/
Python標準ライブラリ	http://docs.python.jp/3/library/index.html
Dive into Python 3 日本語版	http://diveintopython3-ja.rdy.jp/
Python HOWTO	http://docs.python.jp/3/howto/

▼表9 おすすめ書籍

書名(著者)出版社	ISBN	内容
Pythonチュートリアル 第3版(Guido van Rossum 著、鴨澤眞夫 訳)オライリー・ジャパン	978-4-87311-753-9	Webサイトにもあるチュートリアルの書籍版
Pythonプロフェッショナルプログラミング 第2版(株式会社ビープラウド 著)秀和システム	978-4-7980-4315-9	Pythonを使って仕事をしていくためのノウハウ本
Pythonライブラリ厳選レシピ(池内孝啓、鈴木たかのり、石本敦夫、小坂健二郎、真嘉比愛 著)技術評論社	978-4-7741-7707-6	Pythonを便利に使えるライブラリ情報

第3章

Jupyter Notebookを使って納得！

機械学習にはPythonが最適なワケ

Author 福島 真太朗（ふくしま しんたろう）
株式会社トヨタIT開発センター
Twitter @shifukushima
mail sfukushim@gmail.com

Pythonが機械学習に向いている理由

昨今、人工知能がブームとなっています。その中でも機械学習は有望な技術と目されています。機械学習とは、Wikipedia[注1]によれば「人工知能における研究課題の一つで、人間が自然に行っている学習能力と同様の機能をコンピュータで実現しようとする技術・手法」です。平たく言うと、機械学習を用いることにより、人間が大量のデータを分析して知識を取り出すのではなく、自動的に知識を取り出せるようになります。

Pythonは機械学習を実行するうえで有望なプログラミング言語だと言われています。それはなぜでしょうか。以下では、Pythonが機械学習に向いていると言われる理由を説明していきます。

豊富な機械学習／データ解析のライブラリ

Pythonには機械学習のライブラリが豊富にそろっています。機械学習を実行する代表的なライブラリであるscikit-learn、自然言語処理のNLTK[注2]、gensim[注3]などが提供されています。

また、機械学習を実行しようとすると、データを可視化して性質を把握したり、表形式のデータを集計したり、統計量を計算したりする操作も必要になってきます。Pythonにはこうしたデータ解析を支えるライブラリも豊富に提供されています。代表的なライブラリとして、行列計算のNumPy、科学技術計算のSciPy、データ集計・加工・分析のpandas、可視化のmatplotlib、seabornなどが挙げられます（**表1**）。

深層学習のツール／ライブラリ

昨今の人工知能（AI）ブームを引き起こした大きな要因として、深層学習（ディープラーニング）が提唱され、日々進化をしていることが挙げられます。Pythonには深層学習のツール／ライブラリがそろっています。代表的なライブラリとして、Caffe、Theano、Chainer、Tensorflow、

注1）https://ja.wikipedia.org/wiki/機械学習
注2）http://www.nltk.org/
注3）https://radimrehurek.com/gensim/

▼表1　データ解析の代表的なライブラリ

ライブラリ	概要	URL
scikit-learn（サイキット-ラーン）	分類、回帰、クラスタリング、前処理など多くの機械学習の手法を提供	http://scikit-learn.org/
NumPy（ナムパイ、ナンパイ）	行列計算用のライブラリ。高速な行列計算が可能	http://www.numpy.org/
SciPy（サイパイ）	統計計算、最適化、信号処理、フーリエ変換などの科学技術計算を多数提供。NumPyの配列がベースとなっている	https://www.scipy.org/
pandas（パンダス）	データ集計・加工・分析のさまざまな機能を提供。機械学習で頻繁に使う表形式のデータであるデータフレーム構造も提供	http://pandas.pydata.org/
matplotlib（マットプロットリブ）	Pythonの可視化ライブラリのデファクトスタンダード	http://matplotlib.org/
seaborn（シーボーン）	matplotlibベースの可視化パッケージ	https://seaborn.pydata.org/

▼表2　深層学習（ディープラーニング）のツール／ライブラリ

ライブラリ	概要	URL
Caffe / Caffe2（カフェ／カフェツー）	カリフォルニア大学バークレー校のBVLC（Berkeley Vision and Learning Center）が中心となり開発。C++で実装されており、Pythonインターフェースを利用することで深層学習の高速な実行が可能。学習済みのモデル（Caffe Zoo Model）を配布しているため、実績のあるモデルを即座に利用できる。Caffe2はFacebook社が公開しているOSSであり、Caffeをベースとして柔軟性を持たせスケーラブルな深層学習の実行を目指している	**Caffe** http://caffe.berkeleyvision.org/ **Caffe2** https://caffe2.ai/
Theano（セアノ、シアノ）	多次元配列（テンソル）を用いた数式の記述、コンパイル、評価を行うライブラリ。記号演算を導入し、自動微分、GPU対応などの機能を提供する。実行時にPythonのコードをC++のコードにコンパイルするため、高速な実行が可能。深層学習自体を実装したライブラリではないが、こうした機能、特徴を持つため深層学習にもよく用いられる	http://deeplearning.net/software/theano/
Chainer（チェイナー）	Preferred Networks社が開発している日本発のライブラリ。“Define by Run”という方式を採っており、深層学習のネットワークの記述と学習を同時に行う。そのため、多くの深層学習のアルゴリズムに比較的容易に対応することが可能	http://chainer.org/
Tensorflow（テンソルフロー、テンサーフロー）	Google社によるテンソル（多次元配列）の計算機能とそれに基づく深層学習の実行ライブラリ。計算を動的なグラフ（ネットワーク）で表現し、ノード（頂点）が数学の演算、エッジ（辺）がテンソルのデータに対応する。そのため柔軟な記述が可能で、多くの深層学習のアルゴリズムに対応が可能	https://www.tensorflow.org/
Keras（ケラス）	Theano、Tensorflowをバックエンドとして実行する。深層学習のアーキテクチャに層を追加するなどの直感的な記述が可能	https://keras.io/ja/

Kerasなどがあります（**表2**）。Caffe、Theano、Chainer、Tensorflowはそれぞれ単体で深層学習を実行するツールですが、Kerasはその背後でTheanoやTensorflowを実行しながらユーザにとって直感的な記述を可能にしたツールという違いがあります。

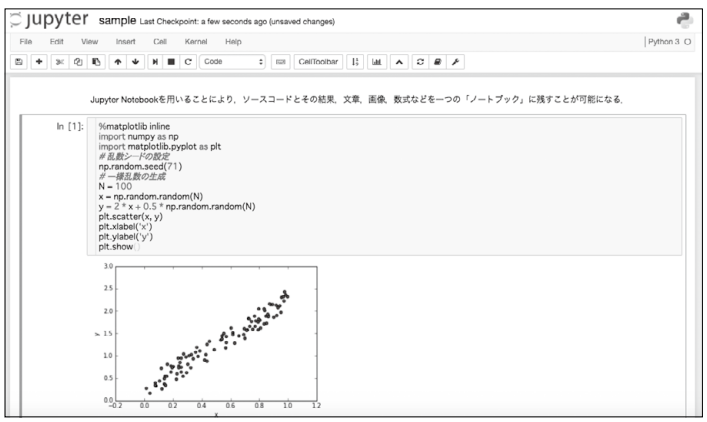
データ解析の実験ノート

機械学習を用いてデータ解析を行う際、さまざまな試行錯誤が必要になります。たとえばデータを可視化してその性質を把握したり、データを機械学習の複数のモデルに適用して結果を比較したりなどの作業が必須です。このような試行錯誤を行った過程のソースコードとその結果を保存しておけると非常に便利です。Jupyter Notebookと呼ばれるWebアプリケーションを用いて、こうしたことを実現できます。Jupyter Notebookでは、「セル」と呼ばれる領域にソースコードを記述して実行したり、数式、可視化の結果、説明の文章を残したりすることが可能になります（**図1**）。Jupyter Notebookの使用方法については、後述します。

Jupyter Notebookの前身は、対話的実行環境であるIPythonとそれをWebブラウザ上で実行するIPython Notebookです。2014年にProject Jupyter[注4]としてPythonだけでなくRやJulia

注4）http://jupyter.org/

▼図1　Jupyter Notebookの例



などの言語もサポートするようになりました。

Anacondaによる実行環境構築の負荷軽減

Anacondaを用いることにより、機械学習の実行環境を容易に整えられます。Anacondaをインストールするだけで、前述のscikit-learn、NLTK、NumPy、SciPy、pandas、matplotlib、seaborn、Jupyter Notebookなど、機械学習関連の多くのライブラリ、ツールを利用できるようになります。ライブラリを個別にインストールする必要はありません。また、Anacondaはバイナリ配布されているため、OSにコンパイラがなくても簡単に環境を構築できるというメリットがあります。しかし、使用しないライブラリなどもインストールされるため、ディスクを大量に消費するなどのデメリットもあります。

ここで紹介したAnacondaは、第2章で紹介した公式版Pythonインストーラとは別のインストール方法となります。機械学習に特化しており、手軽に機械学習を始めたい方にお勧めします。一方、システム開発やWeb開発など別の用途でPythonを使用するには公式版Pythonインストーラを用いるのをお勧めします。それぞれの特徴を確認して、自分自身に合う環境構築方法を見つけましょう。

Anacondaは独自のパッケージ管理機能を提供しています。たとえば、次のコマンドを打つことによりライブラリをインストールできます。

```
$ conda install ライブラリ名
```

Anacondaをインストールするには、まず**図2**のAnacondaのダウンロードページ[注5]から、使用しているOSに合わせたインストーラを取得します。本記事を執筆している時点では、AnacondaはPython 2系と3系に対応したものがそれぞれ提供されています。とくに理由がなければPython 3系をダウンロードしてください。

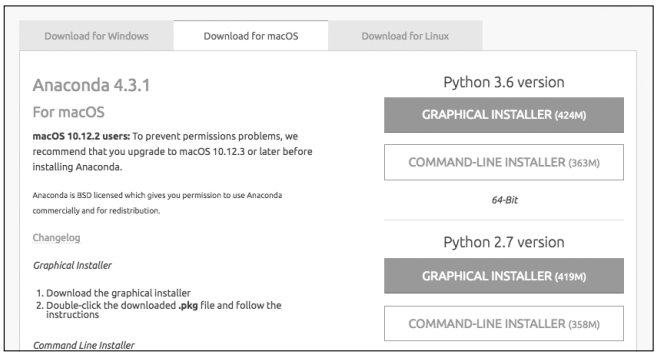
インストーラはグラフィカルにインストールの操作を行う「GRAPHICAL INSTALLER」とコマンドラインでインストールを行う「COMMAND-LINE INSTALLER」の2種類が提供されています。どちらのインストーラでも操作の手間に大差はありません。たとえば「GRAPHICAL INSTALLER」を選択した場合は、インストーラのダウンロード後にダブルクリックして「continue」を押し続けることによりインストールを行えます。

汎用的なスクリプト言語

第1章でも説明されているように、Pythonは汎用的なスクリプト言語でありプログラマやエンジニアにとって取り組みやすいこと、作成した機械学習のモデルをWebアプリケーションで提供しやすいことなどがPythonが機械学習に適している理由として挙げられます。

図3は、全世界のデータ解析者に対して使用している解析ツールのアンケートを行い、回答を集計した結果です。この結果は、KDnuggetsというデータ解析に関連したトピックをまとめたポータルサイトで2016年5月に公開されました[注6]。2016年はPythonのシェアは45.8%であり、データ解析に特化したRの49%に迫る勢い

▼図2　Anacondaのダウンロードページ



注5）https://www.continuum.io/Downloads

注6）http://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html

になっています。Pythonが機械学習、データ解析で注目を浴びていることがわかります。

Pythonで機械学習ひとめぐり

アヤメの種類の分類

ここでは機械学習でよく用いられるIrisデータセットを使って、機械学習のイメージをつかんでいきます。Irisデータセットはアヤメのがくや花びらの長さや幅とそれぞれの花の種類を記録しています。Irisデータセットを用いてアヤメの種類を分類する問題を考えてみましょう。

Irisデータセットには、150枚のアヤメの花を計測したデータが収録されています。**表3**に示すように、行方向(縦の方向)にはそれぞれのアヤメが、列方向(横の方向)にはそれぞれのアヤメの5つの特徴が記録されています。この5つの特徴の意味を**表4**に示します。

表3の「Species」の列を見ると、1番目から50番目のアヤメは花の種類が「0」、51番目から100番目は「1」、101番目から150番目は「2」となっています。花の種類が「0」は「Setosa」と呼ばれる品種を、「1」は「Versicolor」、「2」は「Virginica」という品種を表しています。

Jupyter Notebookの起動と使用方法

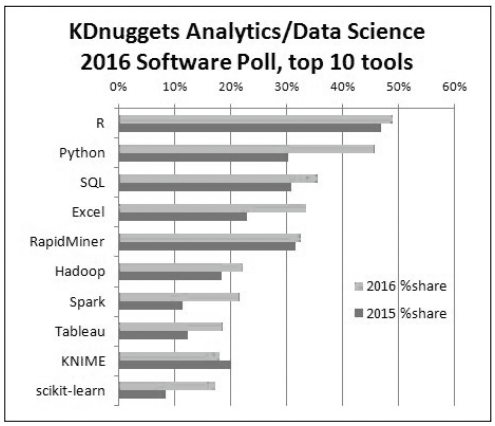
ここからはJupyter Notebookを起動し、ソースコードを記述、実行していくことを前提に説明します。Anacondaがインストールされていれば、コマンドラインから次のコマンドを実行してJupyter Notebookを起動します。

```
$ jupyter notebook
```

コマンドを実行すると、**図4**の画面がWebブラウザに表示されます。また、コマンド実行時のディレクトリの内容が画面に表示されます。

新しいJupyter Notebookの「ノートブック」を作成するには、この画面の右上にある「New」を押し、「カーネル」としてPythonを選択します。「ノートブック」はプログラムとその結果、文章などをまとめたメモ帳を、「カーネル」はコードを実行し結果を返すプロセスを指します。**図5**の例では「Python 3」を選択しています。すると、

▼図3　データ解析者が使用している解析ツール



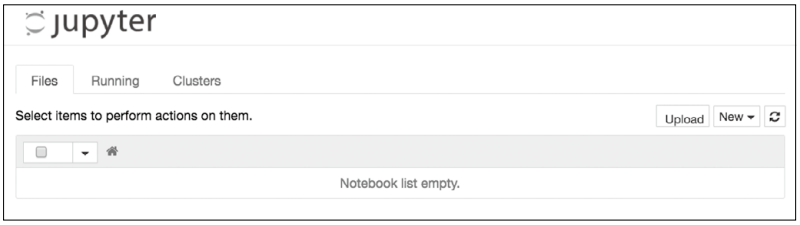
▼表3　Irisデータセットのデータ

	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
1	5.1	3.5	1.4	0.2	0
2	4.9	3.0	1.4	0.2	0
⋮	⋮	⋮	⋮	⋮	⋮
50	5.0	3.5	1.4	0.2	0
51	7.0	3.2	4.7	1.4	1
⋮	⋮	⋮	⋮	⋮	⋮
100	5.7	2.8	4.1	1.3	1
101	6.3	3.3	6.0	2.5	2
⋮	⋮	⋮	⋮	⋮	⋮
150	5.9	3.0	5.1	1.8	2

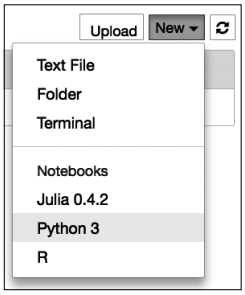
▼表4　アヤメの5つの特徴

特徴	意味
Sepal Length	がくの長さ
Sepal Width	がくの幅
Petal Length	花びらの長さ
Petal Width	花びらの幅
Species	花の種類(「0」「1」「2」の3種類)

▼図4　Jupyterを起動して表示される画面



▼図5　カーネルの選択



※インストールした環境によってプルダウンで表示されるカーネルは異なる。

▼図6　Jupyter Notebookのノートブック

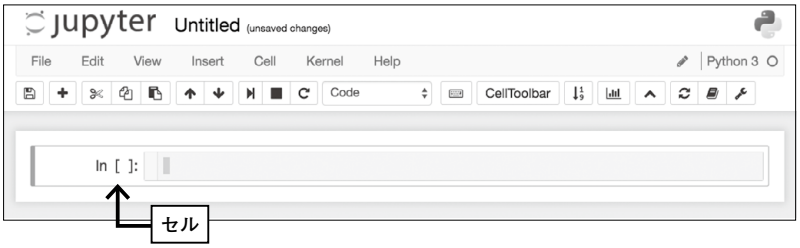


図6の画面がWebブラウザに表示されます。

図6で示したセルにソースコードを記述して Ctrl + Enter（macOSの場合は control + Enter）を押すと、ソースコードが実行されて結果が表示されます。Shift + Enter を押した場合は、セル内のコードが実行されたうえで、1つ下のセルに移動しそのセルを選択した状態になります。

図7は100個の一様乱数を発生させて変数xに代入し、それを2倍したものと一様乱数を0.1倍したものを足し合わせたものを変数yに代入しています。そして、横軸をx、縦軸をyとする散布図をプロットしています。

Jupyter Notebookの使用方法の詳細については、『データサイエンティスト養成読本 登竜門編』[1]の5.2節「Jupyter Notebookで対話しよう」などを参照してください。

これ以降の例で使用するライブラリやツールのバージョンを**表5**に示します。とく にscikit-learnはバージョンアップで変更が行われることが多いので、注意してください。

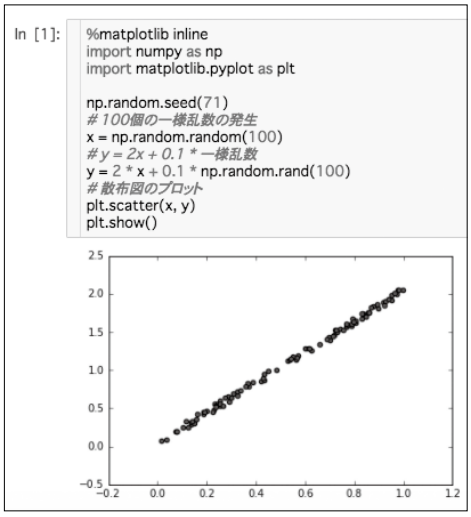
▼表5　本章の例で使用するライブラリ等のバージョン

ライブラリ等	バージョン
Anaconda	anaconda3-4.3.1
scikit-learn	0.18.1
NumPy	1.11.3
pandas	0.19.2
matplotlib	2.0.0
seaborn	0.7.1
Jupyter	4.2.1

データの読み込み

scikit-learnのdatasetsモジュールでは、Irisデータセットを読み込むload_iris関数が提供されています。これを用いてIrisデータセットを読み込み、アヤメの花の特徴を表す4つの変数（Sepal Length、Sepal Width、Petal Length、Petal Width）を変数Xに、アヤメの種類を表す変数を変数yに代入します。

▼図7　セルの実行



```
from sklearn.datasets import load_iris
# irisデータセットを読み込む
iris = load_iris()
# 説明変数と目的変数
X = iris.data
y = iris.target
```

なお、これ以降Xに代入した4つの変数を「説明変数」(機械学習の文脈では「特徴」、「特徴量」と呼ばれることも多い)、yに代入したアヤメの種類を表す変数を「目的変数」と呼びます。以降で説明変数や目的変数の用語を頻繁に使用するので、**表6**にまとめます。

次に説明変数の先頭5行を表示します。変数XはNumPyが提供する2次元の配列となっています。この2次元の配列は、カンマを区切りとして1次元目のインデックスをカンマの前に、2次元目のインデックスをカンマの後ろに指定することにより部分的な配列を抽出できます。たとえば先頭5行の抽出は、配列の1次元目のインデックスに**:5**と指定します。また、2次元目のインデックスに**:**と指定することにより列方向のすべての成分を抽出しています。

```
# 説明変数の先頭5行を表示する
print(X[:5, :])
```

実行結果は次のとおりです。表示された各行の左から、先ほど説明したSepal Length、Sepal Width、Petal Length、Petal Widthが抽出されています。

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])
```

同様に、目的変数に対して先頭の5つの成分を表示します。変数yはNumPyの1次元配列であり、インデックスに**:5**と指定することにより先頭の5つの成分を抽出できます。

```
print(y[:5])
```

実行結果は次のとおりです。

```
array([0, 0, 0, 0, 0])
```

多変量連関図のプロット

次に各説明変数の関係を確認するために、多変量連関図と呼ばれる図をプロットしてみましょう。多変量連関図は、対角線上に各変数の分布(ヒストグラム)をプロットし、非対角線上に2つの変数の散布図を表示した図です。多変量連関図をプロットすることにより、複数の変数の関係性を1枚の図で確認できるようになります。多変量連関図ではなくそれぞれの変数間の関係を確認すると非常に大量の図が生成され、確認にも手間がかかってしまいます。

Pythonで多変量連関図をプロットするには、seabornのpairplot関数を用いるのが便利です(**リスト1**)。seabornは、Anacondaをインストールした場合はすでにインストールされています。

seabornのpairplot関数はpandasのデータフレーム(表形式のデータ構造)を引数にとるため、あらかじめIrisデータセットをこの形式に変換

▼リスト1　多変量連関図をプロットする

```
%matplotlib inline
import pandas as pd
import seaborn as sns
# 説明変数をpandasのデータフレームに変換
df = pd.DataFrame(X, columns=iris.feature_names)
# 目的変数を結合
df['species'] = y
# 多変量連関図をプロット
sns.pairplot(df, hue='species',
markers=['o', 'x', '^'], plot_kws={'s': 50},
palette='copper')
sns.plt.show()
```

▼表6　説明変数と目的変数

名前	ほかの言い方	Irisデータセットの項目
説明変数	特徴、特徴量、入力値	Sepal Length、Sepal Width、Petal Length、Petal Width
目的変数	種類、クラスラベル、期待する出力	Species

しておきます。pairplot関数のhue引数にアヤメの種類を指定することにより、種類ごとに図の点の色を変化させられます。また、markers引数に点のマーカを指定できます。**リスト1**では、クラス0に●(o)、クラス1に×(x)、クラス2に▲(^)を指定しています。hue引数には点のマーカや色を変化させるデータ項目を指定します。**リスト1**では、speciesの値ごとにマーカや色を変化させています。plot_kws引数にはプロットするときのパラメータの名前と値を指定します。**リスト1**では、sというパラメータ(点のサイズ)を50に指定しています。palette引数には図の色に用いるパレットの種類を文字列で指定します。**リスト1**では、白黒の誌面でも区別がつきやすい**copper**を指定しています。なお、ソースコードの1行目にある**%matplotlib inline**は、Jupyter Notebookのノートブック内に描画した図を表示するために指定しています。

実行結果は**図8**です。この多変量連関図を見ると、たとえば3行1列の横軸がSepal Length、縦軸がPetal Lengthの散布図はアヤメの種類が'0'とそれ以外の種類のデータは明確に区別できることがわかります(**図9**)。また、アヤメの種類が'1'と'2'のデータも比較的分離していることが確認できます。以上のことから、機械学習を用いてアヤメの種類を分類できそうです。

分類モデルの作成

次に、3種類のアヤメを4つの特徴(Sepal Length、Sepal Width、Petal Length、Petal Width)をもとに分類する機械学習のモデル(分類モデル)を作成しましょう。このとき、モデルを作成するために使用するデータ(学習データ)と、作成したモデルの精度を検証するデータ(テストデータ)に分割するのが鉄則です。

ここでは、学習データの割合を全体の7割、テストデータの割合を残りの3割とします。そのために、scikit-learnのmodel_selectionモジュールで提供されているtrain_test_split関数のtrain_size引数に0.7を指定します(**リスト2**)。その結果、それぞれの変数には**表7**のデータが入ります。

続いて、今回は多数ある機械学習のアルゴリズムのうち、例としてサポートベクトルマシンと呼ばれるアルゴリズムを用いて、アヤメの種類を分類するモデルを構築します。svmモジュールのSVCクラスをインスタンス化し、fitメソッドに学習データの説明変数と目的変数の変数を指定します。

```
from sklearn.svm import SVC
# SVCクラスをインスタンス化
svc = SVC()
# 学習データを用いて学習
svc.fit(X_train, y_train)
```

なお、サポートベクトルマシンはクラスを分類するために、クラスができるだけ分離するようにクラス間の境界線(高次元の場合は境界面)を引く手法です(「マージン最大化」と呼ばれます)。詳細は、『Python機械学習プログラミング』[4]の3.4節「サポートベクトルマシンによる最大マージン分類」などを参照してください。

テストデータに対する予測

学習したサポートベクトルマシンのモデルを用いてテストデータのクラスラベルを予測して

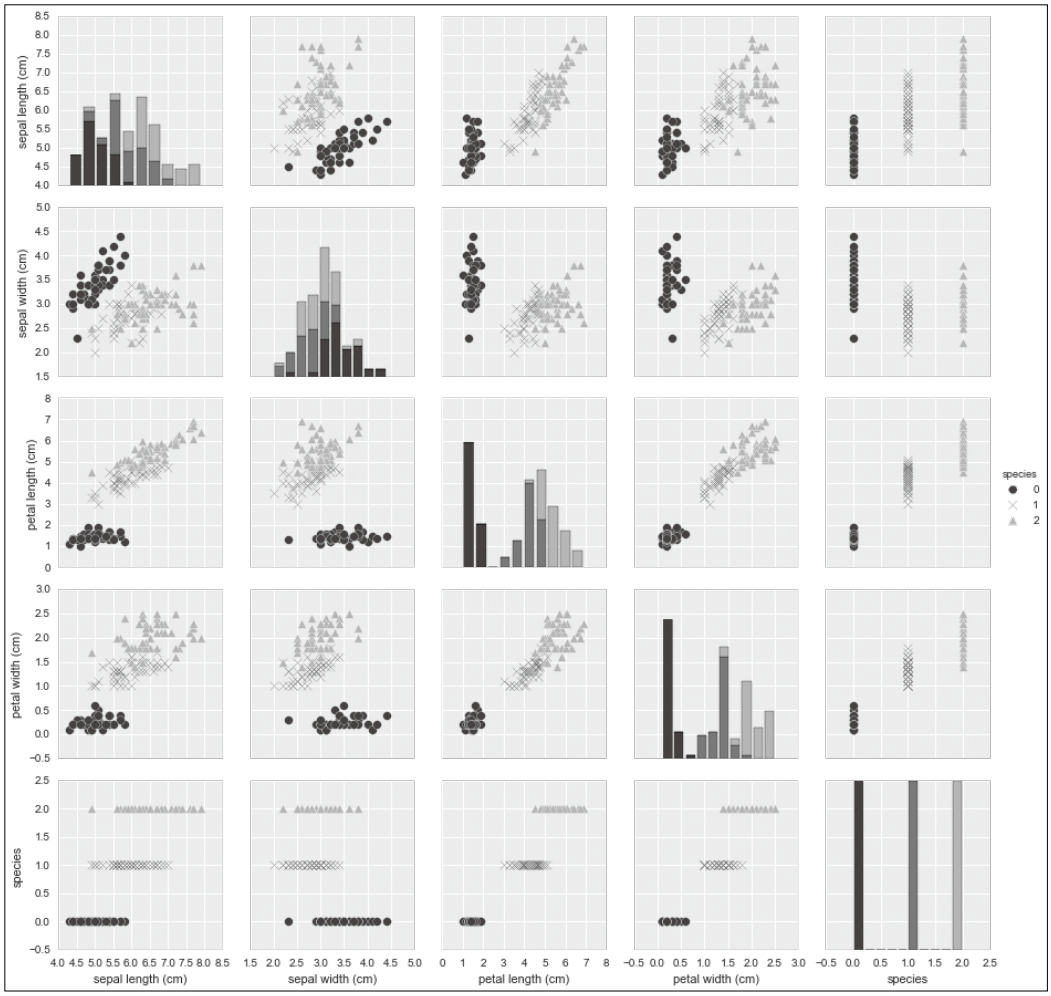
▼表7　各変数に入っているデータ

変数名	意味
X_train	学習データの説明変数
X_test	テストデータの説明変数
y_train	学習データの目的変数
y_test	テストデータの目的変数

▼リスト2　データを学習データとテストデータに分割する

```
from sklearn.model_selection import train_test_split
# 学習データとテストデータに分割する
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=0)
```

▼図8　Irisデータセットの多変量連関図



みましょう。学習したサポートベクトルマシンのモデルのpredictメソッドにテストデータの説明変数を指定します。

```
# テストデータに対してクラスラベルを予測
y_pred = svc.predict(X_test)
```

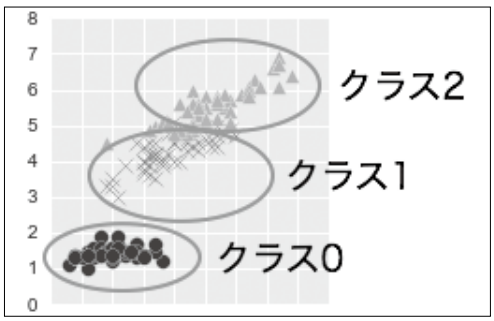
予測結果の先頭の5つの成分を表示します。

```
# 予測結果の先頭5成分を表示
print(y_pred[:5])
```

結果は次のとおりです。

```
[2 1 0 2 0]
```

▼図9　3行1列の散布図（横軸がSepal Length、縦軸がPetal Length）



以上の結果を見ると、先頭の5つのデータに対してそれぞれクラス2、1、0、2、0と予測されていることが確認できます。

予測結果の評価

予測がどの程度正解しているかを評価するために、混同行列、適合率、再現率、F値などの評価指標を算出してみましょう。

混同行列とは、行方向(縦方向)に実際のクラス、列方向(横方向)に予測されたクラスとして件数を集計した表です。**表8**のように集計した値が入ります。混同行列の対角線上の成分が予測と実績が一致して、正解した件数を表しています。それ以外の成分は予測が外れた件数を表しています。

scikit-learnではmetricsモジュールのconfusion_matrix関数を使用して混同行列を算出します。

```
from sklearn.metrics import ↵
confusion_matrix
# 混同行列を算出
print(confusion_matrix(y_test, y_pred))
```

算出した混同行列は、次のようになります。

```
[[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
```

整理すると**表9**のようになります。対角線上が各クラスの正解した件数を表しているので、予測が良好であることがわかります。この混同行列の見方について説明します。

▼表8　混同行列のイメージ

	クラス0(予測)	クラス1(予測)	クラス2(予測)
クラス0(実績)	クラス0と予測して実際にクラス0	クラス1と予測して実際はクラス0	クラス2と予測して実際はクラス0
クラス1(実績)	クラス0と予測して実際はクラス1	クラス1と予測して実際にクラス1	クラス2と予測して実際はクラス1
クラス2(実績)	クラス0と予測して実際はクラス2	クラス1と予測して実際はクラス2	クラス2と予測して実際にクラス2

▼表9　算出された混同行列

	クラス0(予測)	クラス1(予測)	クラス2(予測)
クラス0(実績)	16件	0件	0件
クラス1(実績)	0件	17件	1件
クラス2(実績)	0件	0件	11件

・クラス0と予測した件数は16件ある(1列目の合計)。そのうち、クラス0の件数が16件(1行1列)、クラス1、クラス2の件数は0件(2行1列、3行1列)となっている
・クラス1と予測した件数は17件ある(2列目の合計)。そのうち、クラス0の件数は0件(1行2列)、クラス1は17件(2行2列)、クラス2は0件(3行2列)となっている
・クラス2と予測した件数は12件ある(3列目の合計)。そのうち、クラス0の件数は0件(1行3列)、クラス1は1件(2行3列)、クラス2は11件(3行3列)となっている

続いて、適合率、再現率、F値を算出してみましょう。適合率や再現率、F値などの評価指標は混同行列の成分をもとに定義されます。

・適合率

予測したクラスの件数のうち、どの程度の割合が実際にそのクラスであったかを定量化する指標。表9の混同行列の場合、クラス0と予測した16件のうち実際にすべてがクラス0なので、クラス0の適合率は、16/16＝1.0となる。同様にして、クラス1の適合率は、17/17＝1.0、クラス2は、11/12＝0.92となる

・再現率

実際のクラスの件数のうち、どの程度の割合が予測できていたかを定量化する指標。表9の混同行列の場合、クラス0の16件のうちクラス0としたのは16件なので、クラス0の再現率は、16/16＝1.0となる。同様にして、クラス1の再現率は、17/18＝0.94、クラス2は、11/11＝1.0となる

・F値

適合率と再現率の調和平

均。つまり、F値＝2/(1/適合率＋1/再現率)＝2×適合率×再現率/(適合率＋再現率)と定義される。表9の混同行列の場合、クラス0のF値は、2×1.0×1.0/(1.0＋1.0)＝1.0となる。同様にして、クラス1の再現率は、2×1.0×0.94/(1.0＋0.94)＝0.97、クラス2は、2×0.92×1.0/(0.92＋1.0)＝0.96となる

適合率、再現率、F値の算出には、metricsモジュールのclassification_report関数を用いるのが便利です(**リスト3**)。

結果は**リスト4**のようになります。この結果について説明します。行方向(縦方向)は各クラスが記載されています。「0」「1」「2」はそれぞれクラス0、クラス1、クラス2、「avg / total」はすべてのクラスの平均や合算した結果を表しています。列方向(横方向)には評価指標やデータ数が記載されています。「precision」は適合率、「recall」は再現率、「f1-score」はF値、「support」はデータ数を表しています。したがって、次のことが読み取れます。

- クラス0は適合率が1.00、再現率が1.00、F値が1.00、データ数が16である
- クラス1は適合率が1.00、再現率が0.94、F値が0.97、データ数が18である
- クラス2は適合率が0.92、再現率が1.00、F値が0.96、データ数が11である
- すべてのクラスを合算すると(avg/total)、適合率が0.98、再現率が0.98、F値が0.98、データ数が45である

いずれのクラスでも適合率、再現率、F値が1.00(100%)に近い値になっており、おおむね良好に予測できていることがわかります。しかし、たとえばクラスの適合率0.92に対して、適合率が0.99(99%)以上でなければならない場合は不十分で、逆に適合率が0.9(90%)以上であれば問題ない場合は十分となります。このように、機械学習を用いて評価を行うときは、問題に応じてどの程度の精度が要求されているかを考慮したうえで予測の良否を判断すると良いでしょう。

ここまでで簡単な例を用いてPythonでの機械学習の流れを一通り見てきました。次のステップに進むためには、『データサイエンティスト養成読本 機械学習入門編』[2]、『Python機械学習プログラミング』[4]などの書籍で勉強することをお勧めします。

▼リスト3　適合率、再現率、F値を算出する

```
from sklearn.metrics import classification_report
各クラスの適合率(precision)、再現率(recall)、F値(f1-score)、
データ数(suport)を算出
print(classification_report(y_test, y_pred))
```

▼リスト4　リスト3の実行結果

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        16
          1       1.00      0.94      0.97        18
          2       0.92      1.00      0.96        11

avg / total       0.98      0.98      0.98        45
```

まとめ

本章では、Pythonが機械学習に適している理由を説明したあとに、Pythonで機械学習を実行するイメージを喚起するために分類の例について説明しました。本記事が、読者のみなさんがPythonで機械学習を実行するきっかけとなれば幸いです。SD

参考文献

[1]『データサイエンティスト養成読本 登竜門編』(技術評論社、2017年)
[2]『データサイエンティスト養成読本 機械学習入門編』(技術評論社、2015年)
[3]『ITエンジニアのための機械学習理論入門』(技術評論社、2015年)
[4]『Python機械学習プログラミング』(インプレス、2016年)
[5]『ゼロから作るDeep Learning』(オライリー・ジャパン、2016年)

特別企画

さまざまなシステムで使用されている ハッシュ関数を 前編 使いこなしていますか？

ソフトウェア開発のクリティカルポイント

ハッシュ関数は近年のソフトウェア開発において、とても重要な役割を担っています。この特集ではハッシュ関数とは何か、どのような性質を持ち、どのように利用されているのか、そして、利用の際にはどのようなことに注意する必要があるのかを前後編の2回に分けて解説します。前編となる今回はそもそもハッシュとは何なのか、どのように利用されているのかを解説します。

Author 長谷川 智希（はせがわ ともき） デジタルサーカス㈱ 副団長 CTO　Twitter @tomzoh
イラスト フクモトミホ

あるソフトウェア開発現場での出来事

Aくんは今年の4月に新卒でソフトウェア開発会社に就職した新人エンジニアです。

高校生のころからPHPで掲示板を作ってみたりして、独学でプログラミングを修得してきました。そんなAくん、開発を担当しているプログラムについて上司であるBさんから、ある依頼をされたようです。

上司Bさん：セキュリティ上の理由でユーザのログインパスワードをデータベースに保存しないでほしい。
新人Aくん：ではパスワードを暗号化してデータベースに保存しましょう。
上司Bさん：それはあまりよくないな。ハッシュを使った実装を考えてくれ。
新人Aくん：はい。承知しました（ハッシュ？暗号化と同じことじゃないか。……調べてみよう）

――みなさん、いかがですか？　Aくんの疑問にどう答えるでしょうか。

ハッシュとは何か？

先ほどの上司Bさんは「ハッシュ」と言っていましたが、「ハッシュ」とは何でしょうか。

「ハッシュ」は英語では「hash」と綴り、単語の意味としては「細かく切り刻む」「ごた混ぜ」といった意味を持ちます。これから転じて、コン

ピュータ用語としては、あるデータが与えられたときにそのデータから一定のアルゴリズムで算出した小さな値のことや、そのアルゴリズムのことを指します。正確に言うと、算出された小さな値のことを「ハッシュ値」、アルゴリズムのことを「ハッシュ関数」と呼びます。また、データからハッシュ値を算出することを「ハッシュ化」と呼ぶこともあります。

コンピュータプログラムではハッシュはよく利用されています。みなさんもきっと聞いたことがあると思いますが、MD5、SHA-1、SHA-512などがハッシュ関数の例です。

ハッシュ関数の持つべき特性

あるデータを入力したときに、そのデータから計算された値を出力すればそれはハッシュ関数と呼べますが、良いハッシュ関数は一般にいくつかの特性を持っています。

■同じデータを入力すると必ず同じハッシュ値を出力する

ハッシュ関数にデータを入力した場合、入力したデータが同じものならば何度実行しても必ず同じハッシュ値を出力します。

macOSではMD5アルゴリズムでハッシュ値を計算するmd5コマンドが標準で使用できます。また、Linux系のOSではmd5sumコマンドが利用可能なことが多いでしょう。これらのコマンドは引数にファイル名を与えるとそのファイルの内容をデータとしてハッシュ関数に入力し、ハッシュ値を計算して表示します。

試しにmacOSのmd5コマンドを使って、同じファイル（＝データ）のハッシュ値を2回計算する実験をしてみましょう。

```
$ md5 /bin/ls
MD5 (/bin/ls) = 1b7272164bcb953bb2b9be73038666b6
$ md5 /bin/ls
MD5 (/bin/ls) = 1b7272164bcb953bb2b9be73038666b6
```

同じハッシュ値が出力されていることがわかります。

ここで指定している/bin/lsはmacOS標準の実行ファイルです。筆者の手元のmacOS Sierraではこのハッシュ値になりますが、macOSのバージョンが違ったり、別OSだったりという理由で/bin/lsの内容が異なる場合は別のハッシュ値になります。それでも、同じ環境で2回計算をすれば、必ず同じ値が出力されます。

■どんなデータを入力しても決まった長さのハッシュ値を出力する

ハッシュ関数にどんなデータを入力しても、必ずそのハッシュ関数ごとに決まった長さの値を出力します。

たとえば、MD5は128ビット（＝16バイト）の値を出力します。macOSのmd5コマンドを使って、いくつかのファイルを指定してハッシュ値を計算してみましょう。

```
$ md5 /bin/ls
MD5 (/bin/ls) = 1b7272164bcb953bb2b9be73038666b6
$ md5 /bin/cp
MD5 (/bin/cp) = bcb185c45786a8db87699df66bd0c455
```

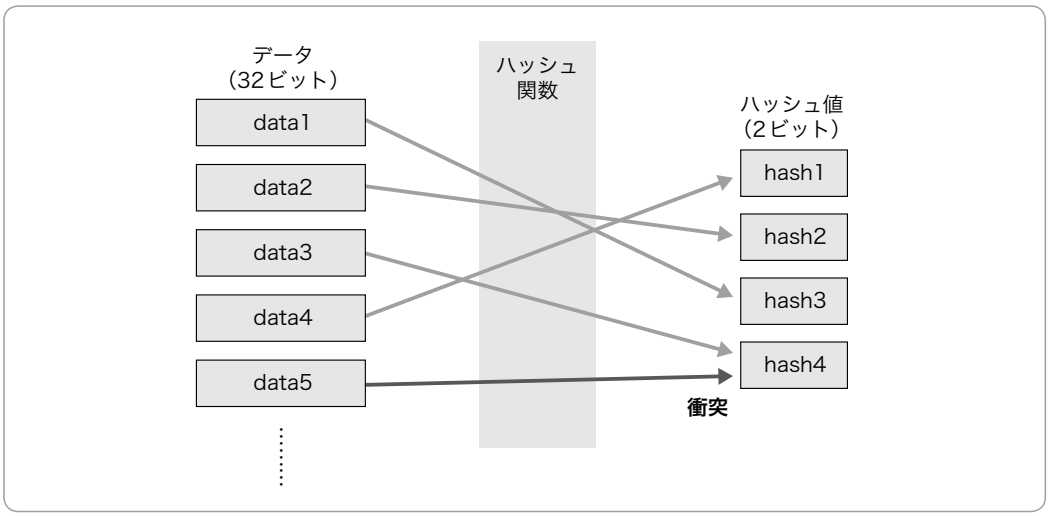
別のデータ（ここではlsコマンドのバイナリファイルと、cpコマンドのバイナリファイル）を入力しても16バイトのハッシュ値が出力されていることがわかります。つまり、MD5は0バイトのデータを入力しても、100GBのデータを入力しても必ず16バイトのデータを出力します。MD5に限らず、広く使われているハッシュ関数はどんなデータを入力してもそのハッシュ関数が決めた長さのハッシュ値を出力します。

■別のデータに対してはほぼ別のハッシュ値を出力する

ハッシュ関数にデータを入力した場合、別のデータに対しては、ほぼ別のハッシュ値になります。「ほぼ」と歯切れの悪い感じですが、これには理由があります。

たとえば、**図1**のように、2ビットの値を返

▼図1　ハッシュ値の衝突



すあるハッシュ関数があったときに、そのハッシュ関数に32バイトのデータを与えることを考えてみましょう。

このとき、データは2^{32}個(＝約43億個)あるのに対して、ハッシュ値は4つしかありません。そのため、データをハッシュ値に対応させていくとそのうち必ず同じ値に対応させることになります。この現象をハッシュ値の衝突と呼びます。

ハッシュ関数に与えるデータのサイズとハッシュ値のデータのサイズはその利用法によって異なりますが、**図1**のようにデータのサイズのほうがハッシュ値のサイズより大きい場合は理論上必ず衝突が発生します。また、データのサイズのほうがハッシュ値のサイズより小さい場合でも、データによっては衝突が発生する可能性があります。つまり、ハッシュ関数を使用する場合は、必ずハッシュ値の衝突が発生するという前提で使用する必要があります。

MD5やSHA-1などよく使用されるハッシュ関数ではハッシュ値の長さがもっと長く、また、衝突が発生しにくくなるように設計されているので、ハッシュ値の衝突はそう頻繁に起きることではありません。それでも理論上はハッシュ値の衝突は発生しますので、ハッシュ関数を使用する際は必ず衝突が発生する、という前提で使用する必要があります。

■ハッシュ値から元のデータを算出することはほぼできない

ハッシュ関数のアルゴリズムは公開されています。そのため誰でも実装でき、あるハッシュ値があったときに、総当たりで元のデータ(そのハッシュ値を出力するデータ)を算出することは理論上はできます。

この「総当たりで」というのは、「最初に0バイトのデータをハッシュ化してハッシュ値と比較し、次に1バイトのデータ、00からffまでをハッシュ化してハッシュ値と比較し、次に2バイトの……」と計算していく方法です。この場合、たとえばハッシュ値の長さがnビットであれば2^n通りのハッシュ値が存在するので、2^n回計算すれば「あるハッシュ値」を出力する元のデータを算出できることが期待されます。

優れたハッシュ関数では、ハッシュ値から元のデータを算出することは非常に難しい設計になっていますが、これは言い換えると、総当たりの計算量が期待値に近くなるような設計になっている、ということです。この「ハッシュ値から元のデータを算出することが難しい」性質を「原像計算困難性」と言います。つまり、優れた

ハッシュ関数は原像計算困難性が高い、ということになります。そして、この原像計算困難性がハッシュアルゴリズムの安全性の大きな指標になります。

ハッシュアルゴリズムの脆弱性

先日、GoogleによってSHA-1の脆弱性が指摘されるというニュースがありましたが、しばしば、ハッシュアルゴリズムの「脆弱性」が話題に上がることがあります。この脆弱性とはどのようなものなのでしょうか。

脆弱性について考えるには、脆弱でない状態、つまり目指すべき状態のことを考えるのが良いのですが、ハッシュアルゴリズムの安全性の指標として先に紹介した「原像計算困難性」のほかにも、「衝突困難性」「第2原像計算困難性」という性質があります。

・衝突困難性
……同じハッシュ値を持つ2つのデータを見つけることが難しいという性質。つまり、ハッシュ値もデータも何でも良い、という前提のもとでハッシュ値を衝突させる攻撃への耐性

・第2原像計算困難性
……あるデータがあったときに、そのデータのハッシュ値と同じハッシュ値を持つほかのデータを見つけることが難しいという性質

衝突困難性と第2原像計算困難性は似ているのですが、前者はハッシュ値の長さがnビットのとき$2^{\frac{n}{2}}$回の計算で見つけることができます。一方、後者は2^n回の計算が必要です。つまり、後者の第2原像計算のほうが困難です。そして、これらのどれかを実現する方法が見つかった場合に「ハッシュアルゴリズムの脆弱性が見つかった」と表現することになります。

先日のGoogleの例は、あるPDFファイルがあったときに、それと内容の異なるPDFで同じSHA-1ハッシュを持つPDFを作成する方法が見つかった、ということで、上記で言うと「第2原像計算困難性」が破られた、ということになります。もう少し正しく言うと、SHA-1のハッシュ値は160ビットだが2^{160}回の計算より少ない、現実的な時間で計算可能な方法を見つけた、ということになります。

ハッシュ関数の利用例

ハッシュ関数は前述のとおり、

・同じデータを入力すると必ず同じハッシュ値を出力する
・どんなデータを入力しても決まった長さのハッシュ値を出力する
・別のデータに対してはほぼ別のハッシュ値を出力する
・ハッシュ値から元のデータを算出することはほぼできない

という性質を持った関数でした。

ここではハッシュ関数のこれらの性質がどのように活用されているかを紹介します。

ファイルの同一性チェック

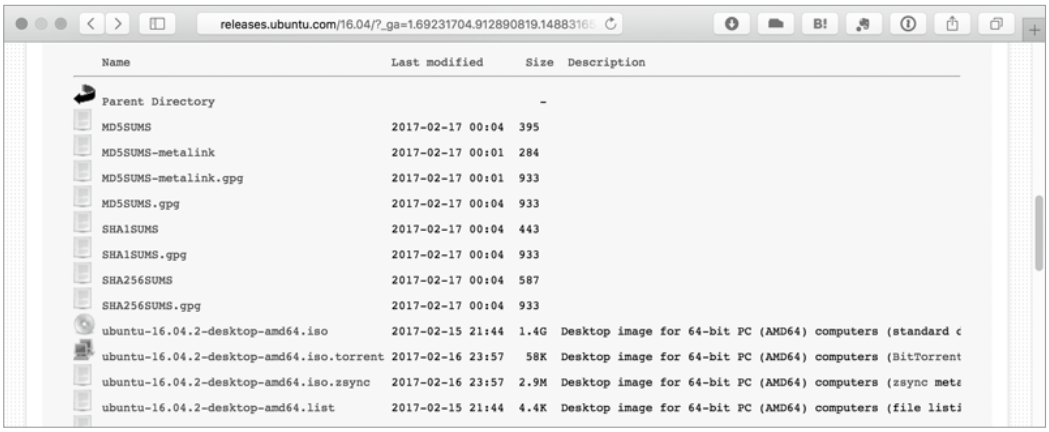
ハッシュ関数のもっともイメージしやすい使い方は、2つのファイルが同じものかをチェックする、ファイルの同一性チェックでしょう。

たとえば、大きなファイルをダウンロードしたときに、ダウンロード元に表示されたハッシュ値とダウンロードしたファイルから自分で計算したハッシュ値を比較することで、データが破損したり途中で途切れたりしていないかを確認できます。

この使い方ではハッシュ関数の持つ「同じデータを入力すると必ず同じハッシュ値を出力する」「別のデータに対してはほぼ別のハッシュ値を出力する」の性質を利用しています。

たとえば、**図2**は、Ubuntu公式サイトのCD/DVDイメージのダウンロードページです。ここにMD5SUMSというファイルがアップロー

▼図2　UbuntuのCD/DVDイメージダウンロードページ



ドされていますが、その内容は図3のようになっています。

▼図3　MD5ハッシュ値

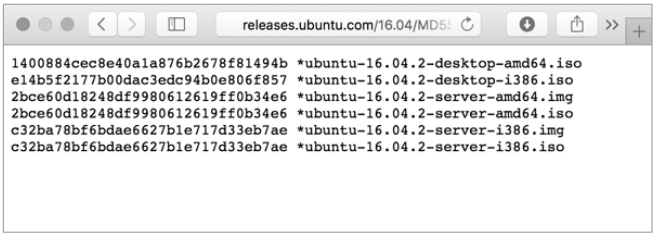


図2のページからUbuntu 16.04.2の64ビットデスクトップ版ubuntu-16.04.2-desktop-amd64.isoのファイルをダウンロードしたユーザは、ダウンロードしたファイルのMD5ハッシュ値を計算し、1400884cec8e40a1a876b2678f81494bになれば正しくダウンロードできたことがわかります。

また、この方法を使うと、ミラーサイトなど、公式でない経路からファイルを入手した場合にも、公式サイトに掲載されたハッシュ値と自身が入手したファイルのハッシュ値を比較することで、そのファイルが正しいファイルなのかを確認できます。

この例のほかにも、ハッシュ値を使ったファイルの同一性チェックは、yumやaptなど、Linuxのパッケージマネージャにも採用されており、内部的にハッシュ値による同一性チェックが行われています。

さて、先に、macOSのmd5コマンドでファイルのMD5を計算する方法を紹介しました。これと同様に、たいていのプログラミング言語では標準またはそれに近いライブラリなどでハッシュアルゴリズムが実装されています。

次の例は、PHPでファイルのMD5ハッシュ値を計算する例です。

```
$ php -r 'print(md5(file_get_contents("/bin/ls")).PHP_EOL);'
1b7272164bcb953bb2b9be73038666b6
```

実装されているアルゴリズムの内容は同じですので、出力されるハッシュ値も同じになっています。

アプリケーションとしてハッシュ関数を使用する場合は、このようにプログラミング言語が用意するハッシュ関数を使用するのが良いでしょう。

推測しにくいURLの生成

Ruby on RailsやCakePHP、Drupalのような最近のフレームワークやアプリケーションでは、データベースのテーブルのキーが連番の数字になっていることがあります。このようなシステムで、「特定の記事ページ」や「特定のユーザのプロフィールページ」を表示するときに、

素直に設計するとURLは次のようになります。

・連番の数字を使ったURLの例

```
https://example.com/article/127
```

ぱっと見て、連番であることがわかります。このようなURLだと、記事が何本あるかがわかってしまう、少ない労力ですべての記事のデータを取得できてしまうなど、あまりうれしくないシステムになってしまいます。

このようなときに、データベースにキーから生成したハッシュ値を保存しておき、URLではハッシュ値を使用するようにするとURLを推測しにくいものにできます。

・ハッシュを使ったURLの例

```
https://example.com/article/ec5decca5ed3d6b80ア
79e2e7e7bacc9f2
```

この使い方はハッシュ関数の「どんなデータを入力しても決まった長さのハッシュ値を出力する」「別のデータに対してはほぼ別のハッシュ値を出力する」の性質を利用しています。ただし、前述のとおり、ハッシュ値は必ず衝突しますので、データベースにハッシュ値を保存するときにはすでにデータベースに同じ値がないかをチェックする必要があります。

上記はMD5を使用した例なのでハッシュ値として16バイトの数字の16進数表記になっています。そのため、使用されている文字は数字とaからfまでのアルファベット小文字です。もし、実際のアプリケーションに使うにあたって、このURLが長過ぎるようであれば、アルファベット小文字のgからzやアルファベット大文字を利用して62進数で表現するなどすると良いでしょう。

ログインパスワードの保存

ハッシュの利用方法のうち、もっともよく知られているのは、冒頭のAくんの例にもある、ログインパスワードのデータベースへの保存でしょう。

ユーザがIDとパスワードを入力してシステムにログインすることを考えてみましょう。システムは入力されたIDとパスワードでシステムにログインさせても良いかを判断しますが、具体的には「今ユーザが入力した文字列(パスワード)は、パスワード設定時にユーザが設定した文字列と同じか」を判断することになります。

このとき、実は、システムはユーザのパスワード文字列そのものを持っている必要はありません。ユーザがパスワードを設定したときに、そのハッシュ値をデータベースに保存します。次にユーザがログインするときにはユーザが入力した文字列からハッシュ値を計算して、データベースに保存されているハッシュ値と比較すれば良いのです。

この方法は、ハッシュ関数の持つ「同じデータを入力すると必ず同じハッシュ値を出力する」「ハッシュ値から元のデータを算出することはほぼできない」性質を利用しています。つまり、万が一、攻撃者にデータベースに保存されたユーザ情報を盗まれても、そこに保存されているのはユーザのパスワードのハッシュ値であり、元のデータを算出することはほぼできないため、ユーザのパスワードは守られます。

データベースにパスワードそのものでなくハッシュ値を保存する方法は、WordPressやDrupalをはじめ、多くのソフトウェアで採用されています。また、完成品のソフトウェアでなくソフトウェアを構成するフレームワークなどでも採用されています。

まとめ

前編である今回は、ハッシュ関数とは何か、どのような性質を持ち、どのように利用されているのかを解説しました。

いくつかのハッシュ関数の利用例を紹介しましたが、これらはどれも、ハッシュ関数が持つべき特性を活かしたものになっていました。

●ハッシュ関数が持つべき特性

A)同じデータを入力すると必ず同じハッシュ値を出力する

B)どんなデータを入力しても決まった長さのハッシュ値を出力する

C)別のデータに対してはほぼ別のハッシュ値を出力する

D)ハッシュ値から元のデータを算出することはほぼできない

●ハッシュ関数の利用例

・AとCの特性を活かしたファイルの同一性チェック

・BとCの特性を活かした推測しにくいURLの生成

・AとDの特性を活かしたログインパスワードの保存

今回の内容だけでも、冒頭のAくんにBさんの意図を半分くらいは説明できそうですね。

Bさんは「セキュリティ上の理由でユーザのログインパスワードをデータベースに保存しないでほしい」「ハッシュを使った実装を考えてくれ」と言っていましたが、これは、前記の3つめの例のようにしてほしいということで、データベースにはパスワードのハッシュ値を保存しておき、ユーザが次にログインするときに入力したパスワードのハッシュ値を計算してデータベースに保存されたものと比較するようにしてほしい、ということでした。

ただ、ここまでのお話では、Aくんの疑問「ハッシュ？　暗号化と同じことじゃないか。」には答えることができませんね。また、実は今回解説した使い方にはいくつかの注意すべき点があります。後編となる次回では、暗号化などハッシュとよく混同される概念のお話や、今回紹介した利用例で注意すべきことなど、もう少し掘り下げてハッシュについて解説します。SD

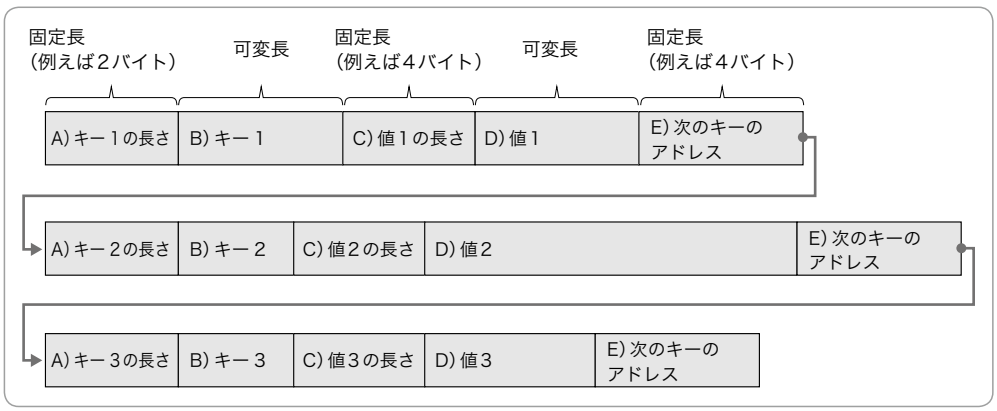
Column 「連想配列とハッシュ」

PerlやRubyなどのプログラミング言語では文字列をキーとして値を格納する配列である連想配列のことをハッシュと呼んでいます。これはまさに内部でハッシュ関数が利用されているためです。

ある連想配列があり、その連想配列のキーSoftwareに、Designという文字列を書き込むプログラムが実行されたとき、プログラミング言語は文字列をメモリに格納する必要がありますが、あとで値を取り出すときのためにキーの文字列から格納したメモリの番地がわかるようにしておく必要があります。

特別の工夫なくこの連想配列を実装すると、たとえば、次のようになります(**図Ⅰ**)。

▼図Ⅰ　連想配列の実装(リスト)



・連想配列への書き込み
……メモリに「A) キーの長さ」「B) キーの内容」「C) 値の長さ」「D) 値」「E) 次のキーのアドレス」と連続して格納する。複数のキーに対する書き込みが発生したらメモリに、A～Eのセットを保存して、最後のEに追加したAのアドレスを格納する

・連想配列からの読み込み
……メモリの先頭から「Aを見てキーをBから取り出し、指定されたキーと比較、一致しなければCを見てその長さだけ読み飛ばし、次のAのアドレスを見る」を繰り返して指定されたキーに対応する値を取り出す

これで連想配列が実装できました。しかし、この方法は連想配列からの読み込み時に、最悪のケースではすべてのキーを走査する必要があり、パフォーマンスに問題があります。

そこで、この問題を解決するための方法として、連想配列の実装にハッシュ関数を使う例を紹介します。ここでは**図Ⅰ**と合わせてメモリのアドレスは4バイトで表現されることとし、インデックスの計算用に2バイトの値を返すハッシュ関数を利用する前提としています(**図Ⅱ**)。

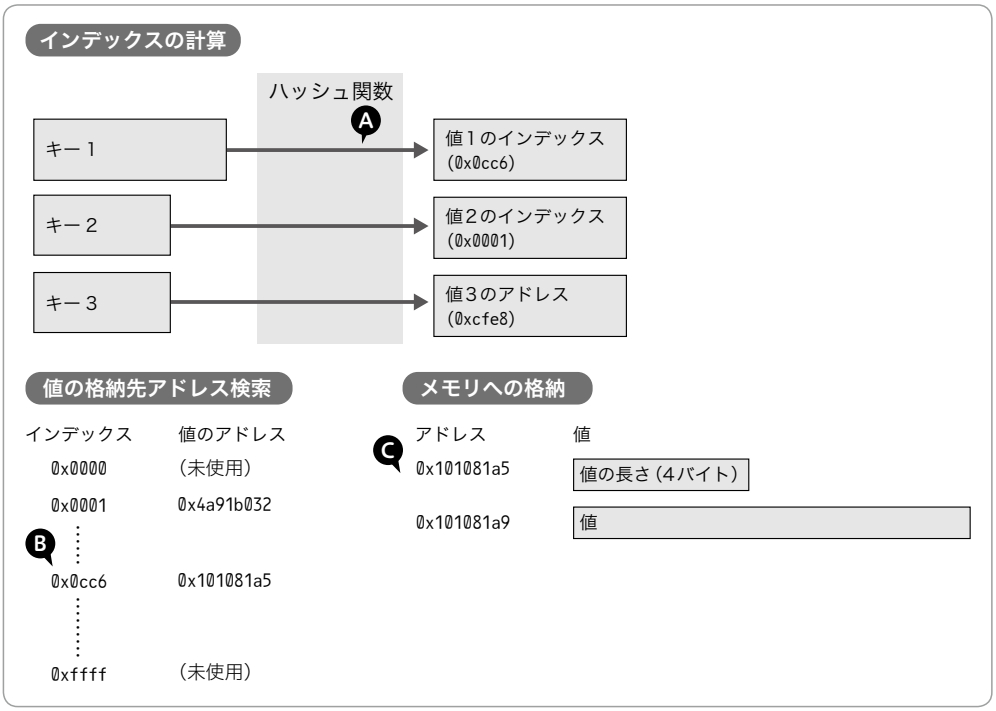
・連想配列への書き込み
……メモリ上の任意の位置に値の長さと値を格納する(C)。キーのハッシュ値を計算し(A)、得られたハッシュ値をインデックスとして値の長さと値を格納したアドレスを格納する(B)

・連想配列からの読み込み
……キーのハッシュ値を計算し(A)、与えられたハッシュ値をインデックスとしてメモリのアドレスを取得(B)、取得したアドレスを参照して値を取り出す(C)

図Ⅱの方法だとキーの数が増えてもハッシュ値を計算してインデックスを見るだけで値に行き着くことができます。連想配列の実装にハッシュ関数を利用する方法は、配列変数をハッシュと呼ぶPerlやRubyに限らず、PHPなど、多くの言語で採用されています。

なお、ハッシュ関数を利用するということで、ハッシュ値の衝突のことを考える必要がありますが、このような場合にはたとえば**図Ⅰ**のリストを使った実装を、同じハッシュ値を持つキーの中だけで使用するなどの工夫をします。この場合、ハッシュ値の衝突がなければ繰り返しなしに、ハッシュ値の衝突があればリストの走査が発生しますが、そのループ数はハッシュ値が衝突したキーの数だけで済みます。

▼図Ⅱ　連想配列の実装(ハッシュ)



正しいデータ共有のススメ

Windows Server 2016で構築する最新ファイルサーバ(前編)

進化した機能で効率化を推進

Author 高添 修(たかぞえ おさむ)
blog https://blogs.technet.microsoft.com/osamut/
日本マイクロソフト㈱

クラウドの台頭によってITの使い方が変わろうとしている今でも、ファイルサーバは企業にとってなくてはならないものとして確固とした地位を築いています。しかし、古くなったファイルサーバが社員の生産性向上の邪魔をしたり情報漏洩リスクの温床になったりして管理者を悩ませているという話もよく聞きます。そこで本記事では、最新サーバOS Windows Server 2016が提供するファイルサーバの機能やファイルサーバとともに見直すべきポイントについて解説します。

ファイルサーバとして必要な機能とは

顧客から求められるファイルサーバへの期待値を最新機能に落とし込むと、次の項目が浮かび上がってきます。

1：ディスククォータ(容量制限機能)
2：ファイルスクリーン(拡張子ベースの書き込み制御)
3：記憶域レポート
4：ファイルの自動分類(FCI:File Classification Infrastructure)
5：共有フォルダの一元管理(サーバマネージャへの統合)
6：通信の超簡単 暗号化設定
7：低コストで実現するファイルサーバ用のストレージ
8：通信の効率化／高速化(SMBマルチチャンネルとNICチーミング)

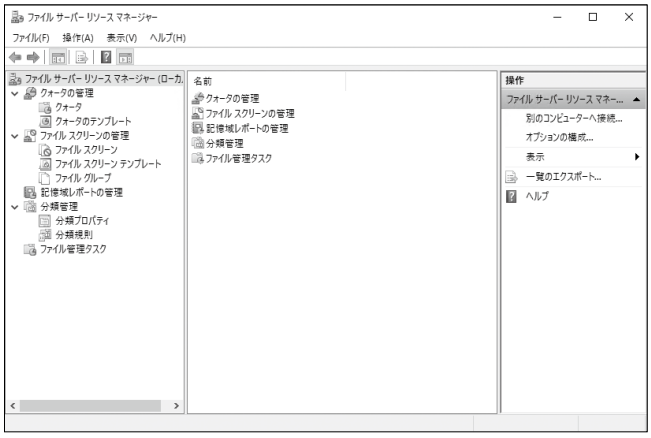
この中には、Windows Server 2008 R2のころに完成していたものも含まれています。しかし、企業がこぞって仮想化技術によるサーバ集約を始めた時期と重なったため、ファイルサーバに焦点が当たる機会も少なく、有用でありながら知られていない、使われていない機能が多いのも事実です。

今回は、あらためてファイルサーバの既存機能も振り返りながら確認していきましょう。

ディスククォータ(容量制限機能)

ハードディスクや専用ストレージの論理ボリュームは、物理的であれ論理的であれ、それ自体が容量を持っています。しかし、ファイルサーバの共有フォルダには容量というものが存在しません。昨今、ビジネスで利用されるファイルのサイズは巨大化の一途をたどっており、共有フォルダに対して何の工夫もしなければ、

▼図1 ファイルサーバリソースマネージャの画面



あっという間にディスクがいっぱいになる可能性もあります。そこで出てくるのがディスククォータというユーザの使用容量制限の機能です。

それでは、使い方について解説していきましょう。まず、ディスククォータはファイルサーバリソースマネージャというWindows Serverの標準機能を使います(**図1**)。このファイルサーバリソースマネージャは、デフォルトで有効化されていないため、使いたい場合には、[役割と機能の追加]から有効にします。

設定はいたって簡単で、①テンプレートの作成(もしくは標準で用意されているものを編集)、②フォルダへの適用という2段階の作業を行うだけです。

図2で標準のテンプレートに使えそうなものがない場合は、既存のものを編集するか、右上の[クォータテンプレートの作成]からオリジナルを作成することになります。作成(編集)画面は**図3**で示します。

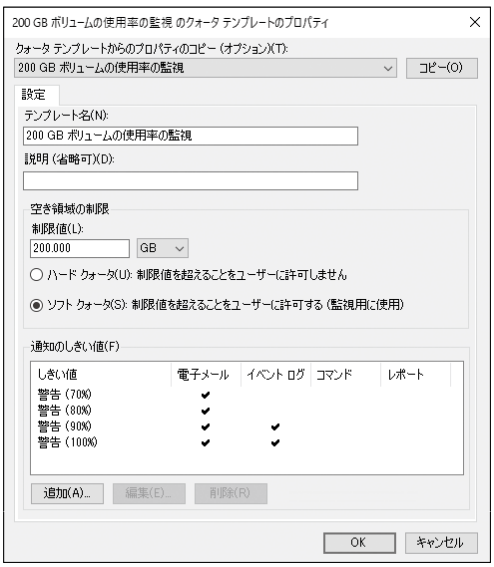
図3を見るとわかるように、このクォータ機能にはハードクォータとソフトクォータという2種類が存在しています。ハードクォータは強制的な容量制限で、利用者には設定した容量以上のデータの保存を許しません。対してソフトクォータは利用者に容量以上のデータ保存を認めつつも状況を管理者に通知する機能です。両方とも通知機能があるので、たとえば設定した容量に対して90%を超えた段階で管理者に電子メールを送ったりイベントログに書き込んだりできます。

さて、テンプレートの準備ができたら、[クォータの作成]という作業を通じて共有フォルダにテンプレートの割り当てを行います(**図4**)。

作業はこれで終了です。指定した共有フォルダに対してユーザがデータを保存していくと、設定した値(%)で警告が通知され、10GBに達するとデータは保存できなくなりま

▼図2　クォータテンプレートの一覧(右上の[クォータテンプレートの作成]からオリジナルを作成可能)

▼図3　クォータテンプレートの作成(編集)画面



▼図4　「F:¥Shares¥部門別フォルダー¥営業部」という共有フォルダに「10GBの制限」というテンプレートを割り当てている

す。とても簡単な操作なのでいろいろと制限したくなるかもしれませんが、必要以上に厳しく制御をするとユーザがローカルPCに保存してデータを紛失したり、無料のクラウドサービスを勝手に利用したりする可能性もあるので、注意をしましょう。たとえば、共有フォルダの容量監視機能として利用するだけでも便利です。

ファイルスクリーン(拡張子ベースの書き込み制御)

ファイルスクリーン、もしくはファイルスクリーニングと呼ばれるこの機能を使うと、共有フォルダに対して許可されていない拡張子を持つファイルの保存を制限できます。こちらも使い方は簡単で次の3段階の作業です。

① 同種類のファイルをグループ化
② ファイルスクリーンテンプレートを作成
③ テンプレートを共有フォルダに割り当てる

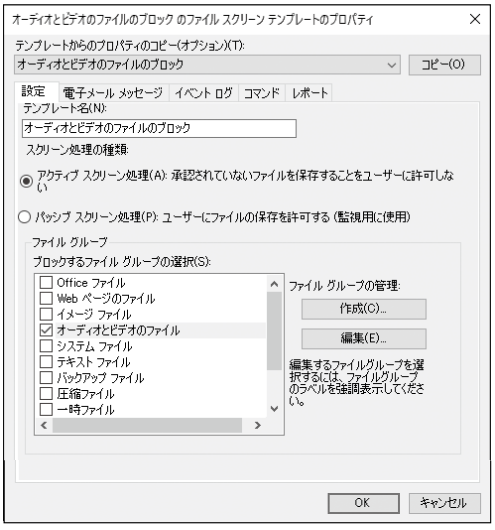
まずは、ファイルグループから見ていきましょう。デフォルトでオーディオとビデオファイルやイメージファイル、実行形式のファイルなどがファイルの拡張子によってグループ化されています。このまま利用することもできますが、会社で利用するアプリケーションによっては特別な拡張子を持つものがあるかもしれません。そのようなファイルを制御の対象にしたい場合には、独自のグループを作成したり、既存のグループを編集したりします(**図5**)。

次にテンプレートを作成します。どのようなファイルグループに対してどのような処理を行うかを指定します。

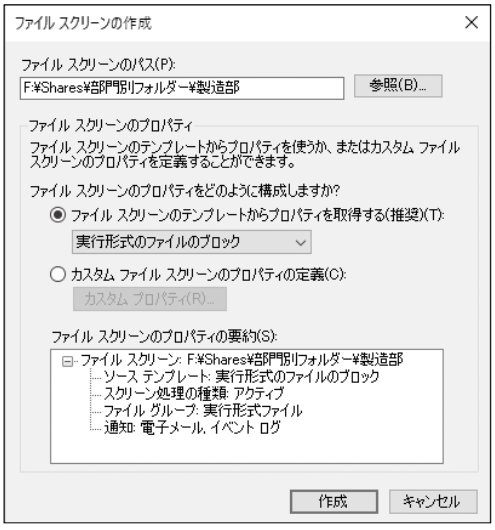
図6を見てもわかるとおり、この

▼図5　ファイルを拡張子でグループ化したところ

▼図6　ファイルスクリーンテンプレートを作成(編集)しているところ



▼図7　「F:¥Shares¥部門別フォルダー¥製造部」で実行形式のファイルの保存をブロックしているところ



ファイルスクリーンにはアクティブスクリーン処理とパッシブスクリーン処理の2種類があります。アクティブスクリーン処理では、承認されていない拡張子を持つファイルの保存を許しません。一方、パッシブスクリーン処理ではファイルの保存は許可しつつ管理者に通知したり独自のコマンドを実行したりできます。先ほどのクォータ同様、制約は時に利用者の生産性を落とすことになるため、強制的に制御すべきか監視だけにして別のアクションにつなげるかは利用前にきちんと検討しましょう。

さて、テンプレートの設定まで終わったら、ファイルスクリーンの作成という作業を通じて共有フォルダにテンプレートを割り当てます(**図7**)。

3段階とはいえ、設定はとても簡単です。この設定により、製造部の共有フォルダには実行形式のファイルが保存できなくなり、保存しようとするとイベントログへの書き込みと管理者へのメール通知も行われます。単に余計なファイルを保存させないという用途だけでなく、ファイルサーバを安全に利用するためにも使えそうです。

記憶域レポート

記憶域レポート機能もまた、ファイルサーバリソースマネージャの機能です。前節で設定したクォータの使用率やファイルスクリーン処理の監査、最終アクセスから90日以上たっているファイル、重複しているファイルの特定など、ファイルサーバ運用時に管理者として知っておくべき項目をレポート化できます。

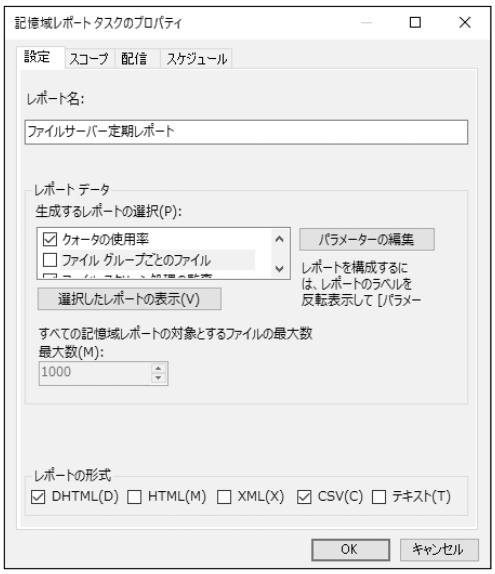
この記憶域レポートは次の2段階の作業で作成できます。

①レポートの設定
②レポートの生成

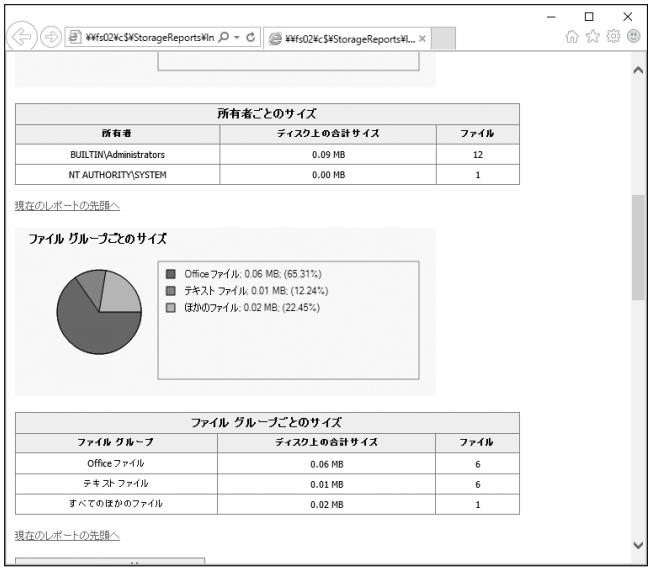
図8の設定を見てわかるように、レポートの形式はXMLやCSV、テキストなども選択できます。また、DHTMLにするとグラフィカルで見た目にわかりやすいレポートを作ったり、メールで配信したりすることもできます(**図9**)。

ちなみに、生成されたレポートはファイルとしてレポート用のフォルダに保存されるため、

▼図8　クォータの使用率などをDHTMLとCSV形式のレポートとして出力するための設定画面



▼図9　所有者ごと、ファイルグループごとのサイズを可視化した記憶域レポート



自動化ツールなどで読み込んで別のレポートに組み込むこともできます。ファイルサーバの効率的な運用には、まずは現状を知ることから始めたいところです。すでにWindows Serverでファイルサーバを運用している企業は、レポートを出力してみるとよいでしょう。

ファイルの自動分類(FCI:File Classification Infrastructure)

一般的に、ファイルの分類は共有フォルダ名やファイル名で行われ、ファイルを作った人がファイル名や保存する場所を考えることになっています。しかし、企業にとって財産ともいうべきデータが含まれたファイルの管理にはもう一手間かけて、各ファイルにビジネス上の意味付けをしておくとよいでしょう。この、意味付けをする際に利用するのがファイル分類機能です。この機能を使うと、**図10**のように、ファイルに新しいプロパティ情報を付与できます。

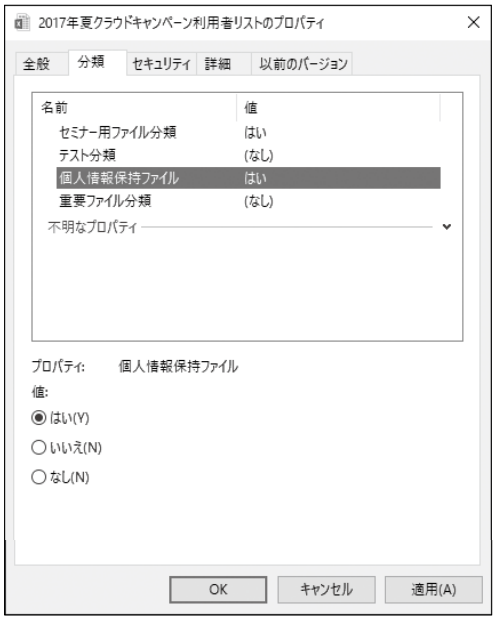
このように、ファイル分類機能を使えば、ファイル名や保存してあるフォルダ名に依存するのではなく、プロパティ情報を使ってファイルにビジネス上の意味付けができるようになるわけです。それでは、設定の流れについてみていきましょう。次の3段階です。

① 分類プロパティの作成
② 分類規則の作成
③ ファイル管理タスクの設定

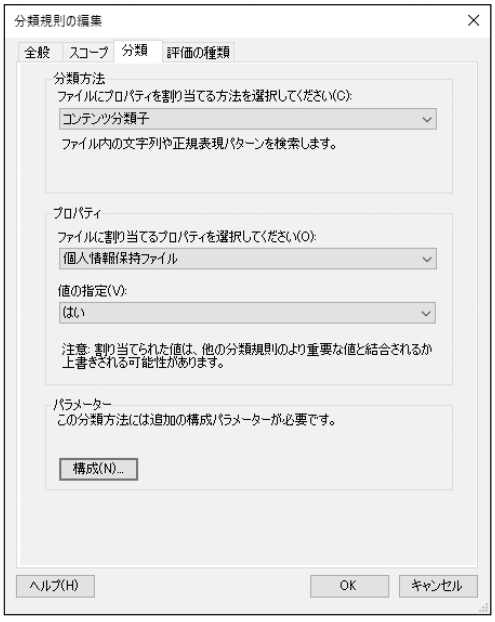
まずは①[分類プロパティの作成]ですが、たとえば**図10**の例の場合、個人情報保持ファイルというプロパティを作成します。また、設定値を「はい」もしくは「いいえ」で指定できるようにします。ちなみに「なし」という分類は「はい」でも「いいえ」でもない状態、つまり分類がされていない状態を表現していると考えてください。

さて、このファイル分類機能の強力なところは、ファイルの中身を見てプロパティを自動設定できるところにあります。その処理を実現するために必要なのが②[分類規則の作成]という作業です。この作業では、「条件の定義」と「変更する

▼図10 プロパティに個人情報保持ファイルであるという設定がされているExcelファイル



▼図11 分類規則の作成画面(その1)[個人情報保持ファイル]プロパティを「はい」にする設定画面



プロパティ値との紐づけ」を行います。

図11は、[個人情報保持ファイル]というプロパティを「はい」にするための設定画面です。設定画面の一番上で分類方法の選択ができるようになっていますが、ファイルならコンテンツ分類子、フォルダならフォルダ分類子、PowerShellスクリプトを使って分類する場合はWindows PowerShell分類子を選択します。今回はファイルに対して処理を実行するため、コンテンツ分類子を選択することになります。

さて、次の設定は[条件の定義]で、何をもって「個人情報保持ファイル」プロパティを「はい」にするかという大事な設定になります。この[条件の定義]は**図11**のパラメーターという項目の中にある[構成]ボタンをクリックして設定します。

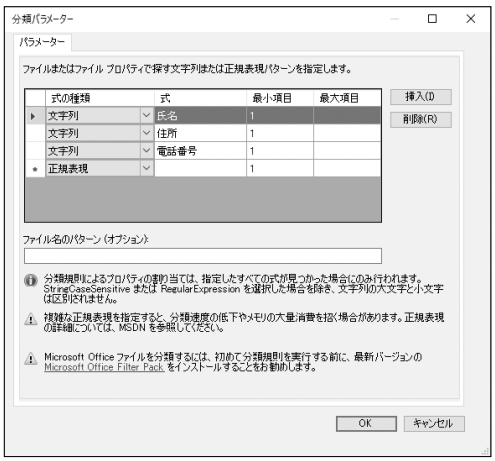
図12では、「ファイルの中に『氏名』『住所』『電話番号』という3つの文字列が1回でも出てくること」と定義しています。Excelファイルなどでユーザリストを作成する際、先頭行にタイトルを付けることが多いところに目を付けたルールとなっていて、この条件に合致したファイルに対して、「個人情報保持ファイル」プロパティを「はい」にするわけです。

ほかにも、「プロジェクトxxに関連するファイル」「電子メールアドレスのフォーマットに合致するデータが2つ以上含まれるファイル」「マイナンバーのフォーマットに合致するデータが2つ以上含まれるファイル」を特定するなど、文字列や正規表現と合致した回数の情報を使ってさまざまなデータを表現できます。ちなみに、**図12**の設定画面内の記述を見てもわかるように、ファイルサーバにOfficeをインストールしなくてもMicrosoft Office Filter Packをインストールすればファイルの中身も見に行くことができます。

さて、定義が終わったあとには、ファイルの中身を見ながらプロパティ設定をするという作業を行うことになります。

図13のように、ファイルサーバリソースマネージャ右側の操作ペインには、[分類スケジュールの構成][すべての規則で今すぐ分類を実行する]などのメニューが表示されていますので、必要に応じて使い分けましょう。なお、この設定が実際に動くかどうかの確認を営業時間中に実行すると、ファイルサーバに余計な負荷がかかり利用者へのレスポンスに影響を及ぼす可能性があります。急ぎでなければ営業時間を外したスケジュール設定を、急ぎであればテスト用の環境を用意するなどで不要なトラブルを

▼図12　分類規則の作成画面（その2）
分類の条件となるパラメーター定義画面



▼図13　分類管理の設定画面

回避しましょう。

さて、分類の処理が終わり、適切なプロパティ値が各ファイルに割り当てられたとしましょう。その次に出てくるのが③[ファイル管理タスクの設定]です。よくある例としては、個人情報が含まれる疑いのあるファイルを別のフォルダに移動するとか、Microsoftが提供している情報保護ソリューション(Rights Management)と連携して、個人情報などが含まれるファイルは自動的に暗号化とアクセス権設定を行ってしまうなどです。設定画面を見ていただくとわかりますが、Rights Management連携は特別な設定画面が用意されているものの、基本的には処理をコマンドやスクリプトとして定義し実装するため、かなり柔軟に処理を埋め込むことが可能になっています。

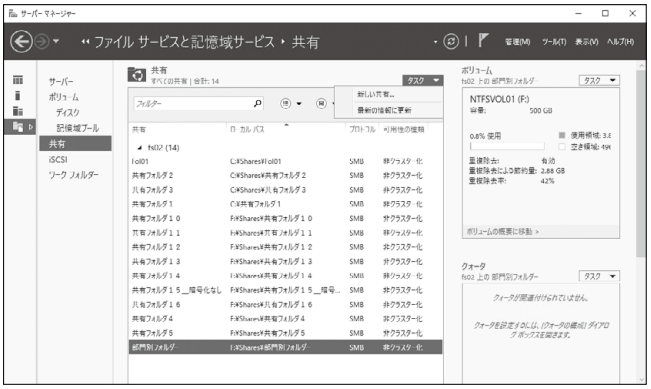
共有フォルダの一元管理(サーバマネージャへの統合)

ここまでファイルサーバリソースマネージャが持つ4つの機能を紹介してきました。どれも条件を作って共有フォルダに割り当てるという設定をするのですが、そもそもの共有フォルダはどうやって作り、どうやって管理すべきなのでしょうか? 多くの管理者はファイルサーバのエクスプローラでフォルダを作り、アクセス権や共有設定をしている可能性も高いですが、最近のWindows Serverには共有フォルダを管理するユーザインターフェースが用意されています。

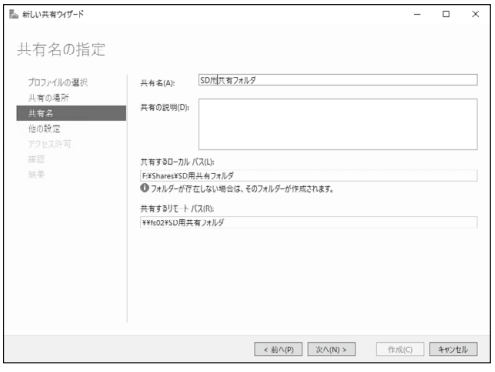
図14のように、Windows Serverの標準管理ツールであるサーバマネージャには、ファイルサービスと記憶域サービスという管理メニューがあり、そこには共有フォルダを管理する画面も用意されています。タスクメニューの新しい共有という作業を選択するとウィザードが立ち上がり、共有フォルダの作成からアクセス権の設定、クォータの割り当てなども行えるようになっていてとても便利です(**図15**)。

この機能を使わないといけないわけではありませんが、利用すること

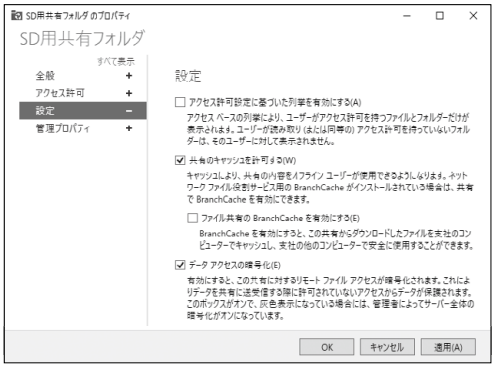
▼図14 サーバマネージャに標準で用意されている共有フォルダ管理画面



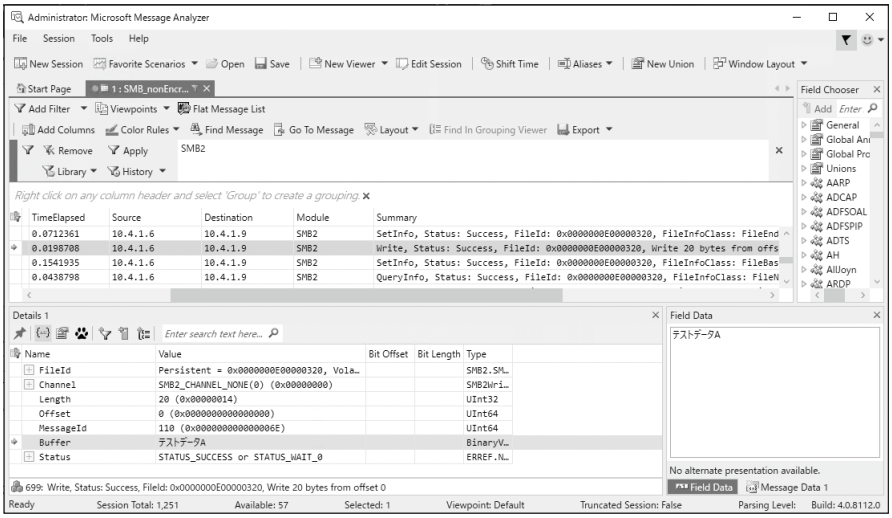
▼図15 新しい共有ウィザードの画面。共有フォルダを作成しつつ、共有設定をしているところ



▼図16 データアクセスの暗号化のチェックを付けたところ



▼図17　暗号化されていない共有フォルダに送ったデータのキャプチャ結果



で、どのドライブのどのフォルダに共有設定をしたのか一目瞭然となりますので、管理負荷を減らせる可能性があります。また、この設定画面から簡単に設定できるオプションの機能があります。それが暗号化です。

通信の超簡単 暗号化設定

昨今のセキュリティの脅威に対抗するには、これまでの常識を捨て去る必要があります。たとえば、ファイアウォールの中は安全というのは過去の話で、多くの企業ネットワークの中に攻撃者は侵入済みであると言われています。このような状況の中で、ファイルサーバそのもののセキュリティを強固にすることも重要なのですが、ユーザのPCからファイルサーバへと書き込まれる通信データの保護も重要となります。

Windows Server および Windows で使われるファイル転送プロトコルSMBはバージョン3からエンドツーエンドの通信の暗号化の機能が含まれており、それを使わない手はありません。しかも、この暗号化の設定は**図16**の共有フォルダ作成のウィザードの中でチェックボックスにチェックを付けるだけで実現できます。

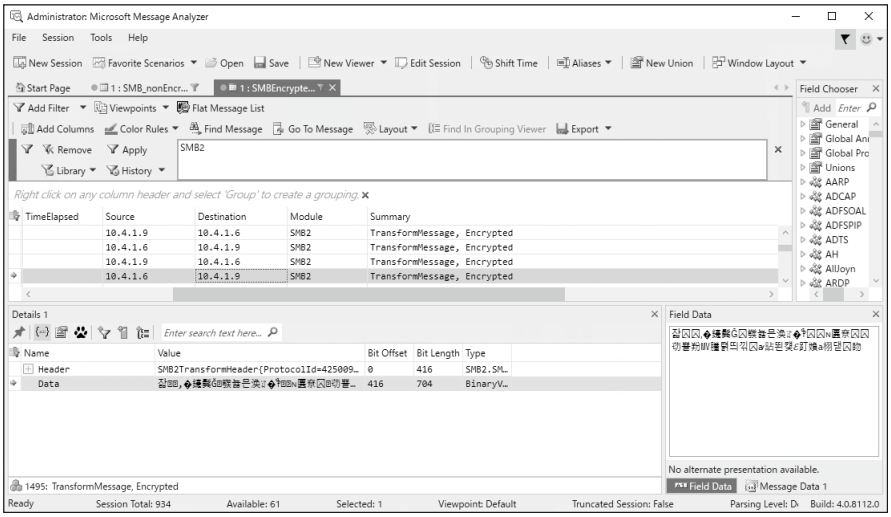
暗号化設定の効果を測るべく、ネットワークキャプチャツールでキャプチャしてみました。**図17**のように、暗号化設定をしていない共有フォルダへの書き込みでは、ファイル名ではなくファイルの中に書いてある「テストデータA」という文字列が見えていました。

そして**図18**のように、暗号化の設定をした場合は「TransformMessage, Encrypted」としてデータの中身が見えなくなりました。難しい設定をする必要もなく暗号化をしてくれるため、ぜひとも重要なファイルを保存する共有フォルダには設定をお勧めします。

低コストで実現する ファイルサーバ用のストレージ

いざファイルサーバを導入しようとすると、ハードウェア機器の調達が必要です。物理サーバは多種多様な機器が販売されており、形状やスペック、サービスなどで選択ができると思います。ただ、ファイルサーバの容量はどんどん増えていくことが想定されるため、ストレージの選定はとても重要になり、専用のストレージ装置の場合はかなり高額になるケースもあります。そこで登場したのがSoftware Defined Storage（以降、SDS）という選択肢です。最終的に物理ディスクは必要ですが、複数ディスク

▼図18　暗号化設定がされた共有フォルダへ書き込みを行った際のキャプチャ結果



のプール化や階層化、RAID相当の制御を一般的なIAサーバとソフトウェアで実現できるようになりました。

Windows Serverも、昔から提供してきたソフトウェアRAID機能ではなく、新たに開発したSDSの機能を2012 R2に搭載し、ディスクのプール化や階層化、重複除去などを提供し始め、最新のWindows Server 2016ではSDSを大幅に強化しました。これにより、安価で高速で容量効率の高いファイルサーバを構築することが可能になっています。

図19では、Windows Serverのサーバマネージャの画面でディスクをプール化し、論理ボリューム(LUN)を管理し、そしてFドライブは重複除去率42%の状態で重複除去が動作していることが確認できます。

通信の効率化／高速化(SMBマルチチャンネルとNICチーミング)

最近の物理サーバはコア数も増え、大量のメモリを載せることもでき、しかもSSDの台頭と階層化ストレージの効果で、環境によってはネットワークがボトルネックになる可能性があります。Windows Serverには複数のネットワークカードを効率的に利用するしくみとして、SMBマルチチャンネルやNICチーミングの機能が組み込まれています。

まず、SMBマルチチャンネルとは、複数の通信パスを持つマシン同士の通信時に、その複数のパスを自動的に利用してくれるWindows Serverの機能です。NICを2枚ずつ搭載したサーバ間で大きなファイルをコピーしてみると、両方のNICを利用していることがわかります。ただし、両方のネットワークのパスが違う経路でつながっていたり、ホップ数に大きな違いがあったりすると、効率的な通信が難しくなるため、クライアントとファイルサーバ間ではなく、データセンタ内のサーバ間通信などに効果を発揮するはずです。

やはり、一般的にネットワークの高速化と可用性の担保の両立には、NICチーミングを利用することが多いでしょう。最近のサーバにて採用が増えてきた10GbpsのNICであっても、スイッチも安価に手に入る1GbpsのNICであっても同じです。ただ、Windows Server 2008や2008 R2のころからWindows Serverを利用している人は覚えていると思いますが、昔はNICのチーミングを行うためにIntelやBroadcomの専用ドライバとチーミング管理用のツールをサーバにインストールしていました。安定したドライバのバージョン選定、ドライバとツールのバージョンの整合性の調整など、手間と不安材料が多い印象がありましたが、Windows Server 2012以降はNICチーミングをOSの標準機能としてサポートするようになったため、Windows Serverに準拠したドライバをそのま

ま使え、しかもサーバマネージャで設定できます(**図20**)。

ちなみに、仮想マシンとしてファイルサーバ環境を構築する場合、物理サーバ側でチーミングの設定をしておくことになります。ただし、Hyper-V環境では仮想スイッチとNICチーミングが同居することになり、制約がありました。Windows Server 2016のNICチーミングでは、Hyper-Vの仮想スイッチとの整合性が保てるSwitch Embedded Teaming(SET)というしくみを搭載しています。

▼図19　Windows Serverのストレージ設定画面と重複除去が有効になったボリューム

▼図20　4枚のNICをチーミングしているところ

まとめ

本記事では、Windows Serverのファイルサーバの機能を復習もかねて解説しました。ただ、高速なファイルサーバを安価に実現するWindows Server 2016 SDSの詳細や、管理者の負荷が大きいファイル共有のアクセス権管理に対する新しいアプローチ「動的アクセス制御」など、今回紹介できなかったものは次回で紹介予定です。SD

ご存じですか？　StorSimple

最近、ファイルサーバとともに導入されるケースが増えているストレージ装置があります。それは、Microsoft自身が提供するStorSimpleというiSCSIストレージです。なぜ、Microsoftが物理ストレージ装置を提供しているのか、と驚かれた方がいるかもしれません。このストレージはSSD＋HDD＋Azureという3階層になっていて、利用者はパブリックなクラウドAzureをあまり意識することなく、オンプレミスにあるファイルサーバ用のストレージ装置としてハイブリッドクラウドを活用できるのです。このStorSimpleも機能が増え、今ではAzure上の仮想アプライアンスとしてオンプレミスのStorSimpleとデータ同期をしたり、その仮想アプライアンスをオンプレミスの仮想化基盤上でも動かせるようになったりしています。

特別企画

スマホアプリ開発をもっと便利に簡単に！

ニフティクラウド mobile backend mBaaSのしくみ紹介

第3回

IoTドア監視アプリを作ろう

Author 富士通クラウドテクノロジーズ㈱
Dinh Thuy Duong（ディン トゥイ ズオン）
鈴木 耀平（すずき ようへい）、池田 夏藻（いけだ なつも）

連載最後となる今回はニフティクラウドmobile backendを実際に使って体験してみます。最近にわかに注目を集めているIoTと組み合わせて簡単なアプリを作ります。手を動かしてみればmBaaSがどういったものなのか、すぐにわかるはずです。

Part1 スマホとmobile backendでIoTアプリを作る

● Author Dinh Thuy Duong（ディン トゥイ ズオン）

「IoT」という言葉は昔からありましたが、ここ数年はとくに注目を浴びています。その理由は、近年急速に進む、I（Internet）とT（Things：もの）の技術発展にあります。通信インフラのめざましい発展により、無線環境の導入でコストが下がりました。その結果モバイル通信環境が充実し、場所を問わず気軽にインターネットにアクセスできるようになりました。

IとTをつなぐデバイスも技術革新が進みました。5年ほど前は、デバイス開発と言えば、マイコンのボード／抵抗／電圧計などを購入し、ハンダ付けをして組み立てるという流れが一般的でした。今ではそうした作業をすることなく、気軽に使用できるマイコンボードがあります。このようなデバイスを活用して、センサーと連携して情報収集をすることも簡単にできるようになりました。

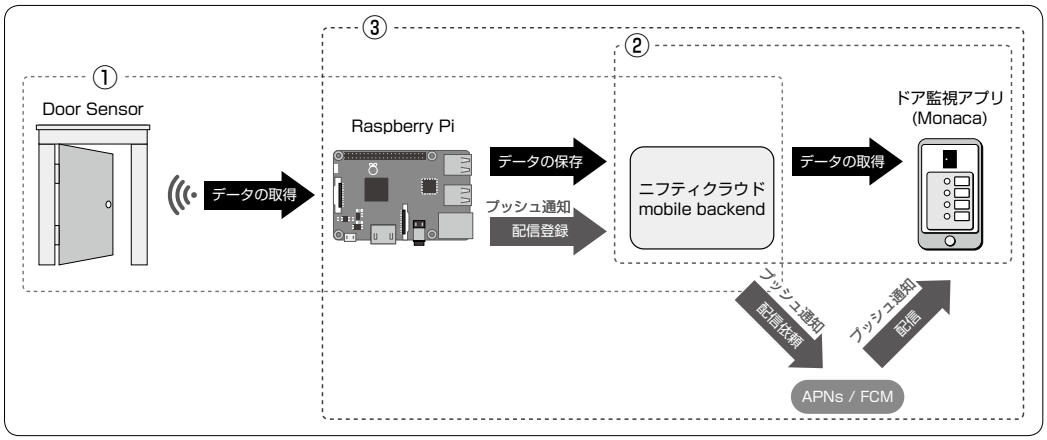
人口の約70%[注1]にまで普及が進んだスマートフォン（以降、スマホ）も高機能デバイスの1つです。手元にあるスマホを活用すれば、データの可視化のためのディスプレイや、データを操作するための専用コントローラなどの開発がいりません。これは単にデバイスの開発コストの削減ができるだけではありません。ユーザがアプリをダウンロードすることで、簡単にIoTサービスを導入できるようになったのです。その結果、スマホを活用したIoTシステムが続々と開発されています。

本稿では、このようなIoTを実現するものの1つとして「ドア監視アプリ」を作成します（**図1**）。

アプリ側は、HTML5ハイブリッドアプリ開発ツール「Monaca」を使用します。これでiOS/Android両方に対応したハイブリッドアプリを制作できます。ドアの開閉データの「保存先」サーバと、開閉状況を知らせる「プッシュ通知の配信」サーバには、ニフティクラウドmobile backend（以下mobile backend）を使用します。mobile backendから発行されるAPIキーをクライアント側に埋め込むだけでインターネット連携ができます。SDKがあるので実装も簡単にできます。

注1） https://marketing-rc.com/article/20160731.html

▼図1 ドア監視アプリ全体図



Part2 ドア監視のデバイスと回路作成

● Author 鈴木 耀平(すずき ようへい)

図1の破線部分①を作って、ドアの開閉情報をmobile backendに保存できるようにしましょう。ドアの開閉状態はRaspberry Piと磁石リードスイッチセンサーで取得します。

使用するパーツの準備

まず、ドア監視の回路を作るために必要なデバイス、部品の一覧が次になります。

・Raspberry Pi 2 model B.....1台
・磁石リードスイッチセンサー.....1個
・カーボン抵抗(1/4W 1kΩ).....2個
・ジャンパーコード(オス：メス).....3本
・ブレッドボード.....1個

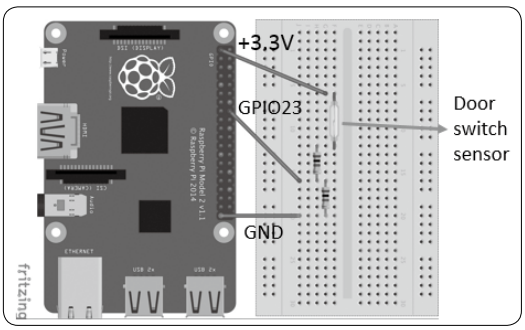
磁石リードスイッチセンサーを使用してドアの開閉状態を信号で受け取ります。これらはすべて秋月電子通商の通販サイト[注2]で購入できます。

デバイスの設計と組み立て

前節で用意した部品を**図2**のように組み立てていきます。基本的な作業は、Raspberry PiのGPIOとブレッドボードに電子部品を差し込むだけです。このGPIOとは、デジタル信号の入出力を行うRaspberry Piのインターフェースです。各Pinごとにそれぞれ役割が違います。今回は、Pin1の電源とPin16のGPIO23、Pin39のGroundを用います。ジャンパーコードのメス側をこれら3つのGPIOに差し込んでください。

さらにデバイスを作り込みます。**写真1**のように、カーボン抵抗と磁石リードスイッチセンサー、それに先ほどのGPIOに差し込んだジャンパーコードのオス側をブレッドボードに差し込んでください。最後に磁石リードスイッチセ

▼図2 回路図



注2) http://akizukidenshi.com/catalog/default.aspx

ンサーをドアに設置すれば、デバイスの準備は完了です。

Node-REDのインストール

Raspberry PiにNode.jsベースのNode-RED[注3]をインストールします。これは特定の処理をノードという単位にまとめ、組み合わせることができるソフトウェアです。このNode-REDを使うと簡単にアプリを作成できます。

Node-REDの環境設定と準備

まずRaspberry Piにターミナルかsshでログインし、Node.jsとnpmのインストールをします。それからNode-REDのインストールをします。

次にインストールしたNode-REDとmobile backendを連携させるための準備をします。/root/.node-red配下に下記のリポジトリをgit cloneします。

```
https://github.com/NIFTYCloud-mbaas/nodered_ncmb
```

cloneしたリポジトリ内のnodered_ncmb/nodesに移動して、npm -g install.を行います。そして、/root/.node-red/nodered_ncmb/flows_raspberrypi.jsonを/root/.node-red配下にコピーします。コピー後、Node-REDを起動します。

これでNode-REDの準備は完了です。

Node-REDの動作確認

Raspberry PiのNode-REDにブラウザからアクセスする必要があります。Webブラウザで次のURLにアクセスします。

```
http://【RaspberryPiのIP】：1880
```

Node-REDをWebブラウザで開くと**図3**のような3つのノードが画面の中央に表示されます。

これらの内DoorStatusINとSaveDataToServerという2つのノードをおもに使用します。DoorStatusINノードではドアセンサーの信号を受信し、その信号をmobile backendに保存するのがSaveDataToServerノードです。

DoorStatusINノードの設定をしましょう。中央の画面にある[DoorStatusIN]をダブルクリックして設定画面を開いてください。設定画面の中にあるGPIO Pinを[16 - GPIO4 - BCM 23]にして[Ok]をクリックしてください。

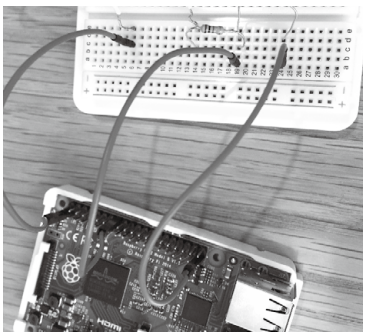
この状態でNode-RED画面右側のdebugタブをクリックして、ドアを開閉してください。センサーによってドアを開けたときに0、閉めたときに1という値が送信され、その値がdebugログに出力されます。

mobile backendと連携

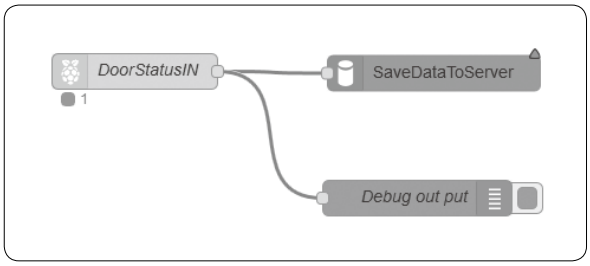
mobile backendに登録・連携を行い、動作確認します。

注3）http://nodered.org/

▼写真1　デバイスと回路図



▼図3　ノード（DoorStatusINとSaveDataToServer、Debug out put）



mobile backendへ登録・連携

次のURLよりmobile backendの会員登録を行い、ログインします。

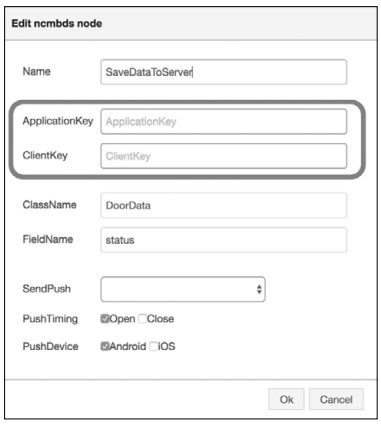
```
http://mb.cloud.nifty.com
```

mobile backendのログインが完了し、アプリの新規作成を行うと、APIキーとしてアプリケーションキーとクライアントキーの2つの鍵が発行されます。アプリケーションキーは各アプリケーションを判別するための鍵、クライアントキーはユーザを認証するための鍵として用います。その2つのキーをNode-REDのSaveDataToServerに設定します(**図4**)。アプリケーションキーをApplicationKeyに、クライアントキーをClientKeyにコピー&ペーストしてください。貼り付け終えたら[Ok]をクリックし、右上にある[Deploy]をクリックします。

動作確認

実際にドアを開閉させてみたあと、mobile backendの管理画面を見てみましょう。管理画面の左のタブにあるデータストアをクリックしてください。DoorDataクラスをクリックすると、ドアの開閉状態のデータが確認できます(**図5**)。

▼図4　SaveDataToServer設定画面



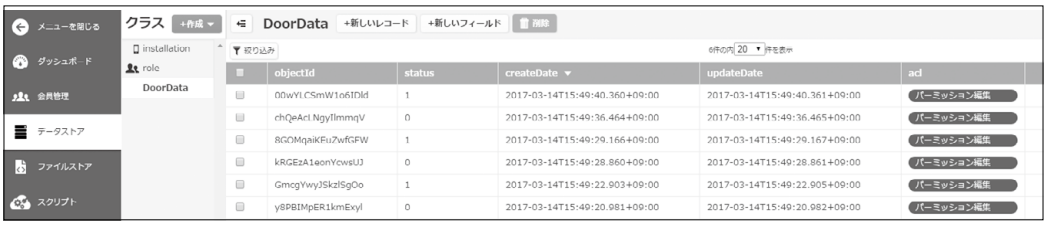
解説

ここからNode-REDで実装されているコードを示します(**リスト1**)。SaveDataToServerノードの実装を見てください。最初に、`var NCMB = require("ncmb");`でncmbのモジュールを読み込んでいます。そして、Node-RED上で設定したApplicationKeyとClientKeyを用いて、ncmbを初期化しています。そのあと、DoorDataクラスにドアの開閉状態を保存しています。この実装では、ncmb.DataStore(this.

▼リスト1　SaveDataToServerノードの実装(抜粋)

```
var NCMB = require("ncmb");
…… (省略) ……
RED.nodes.createNode(this,config);
    this.applicationkey = config.applicationkey;
    this.clientkey = config.clientkey;
    this.classname = config.classname;
    this.fieldname = config.fieldname;
    this.sendpush = config.sendpush;
    this.sendtimeopen = config.sendtimeopen;
    this.sendtimeclose = config.sendtimeclose;
    this.sendandroid = config.sendandroid;
    this.sendios = config.sendios;

    var node = this;

    this.on('input', function(msg) {
        var ncmb = new NCMB(this.applicationkey, this.clientkey);
        var NCMBClass = ncmb.DataStore(this.classname);
        var ncmbClass = new NCMBClass();
        ncmbClass.set(this.fieldname, msg.payload);
        ncmbClass.save();
  …… (省略) ……
    });
```

▼図5　mobile backendダッシュボード(DoorDataクラス)



classname);でデータストアのクラスを作成しています。そして、ncmbClass.set(this.fieldname, msg.payload);でオブジェクトにドアの開閉状態のデータを持たせたあとに、ncmbClass.save();でmobile backend上に保存しています。

Part3 ドア監視アプリを作る

● Author 池田 夏藻(いけだ なつも)

前節では、ドアの開閉データをmobile backendに保存しました。ここからはそのデータを使って、いつでもどこでもドアの開閉状況を確認できるようにアプリを作ります。**図1**の破線②の範囲です。

Monacaでアプリを作成する

プロジェクトの準備

ドア監視アプリのできあがりのイメージ画面は**図6**のようになります。

まずは、Monacaの会員登録を行い、ログインします。

```
https://ja.monaca.io/
```

コーディング部分は、すでに実装済みのプロジェクトを利用します。[Import Project]から次のURLを指定してアプリ作成してください(**図7**)。

```
https://github.com/NIFTYCloud-mbaas/IoT_DoorApp/archive/master.zip
```

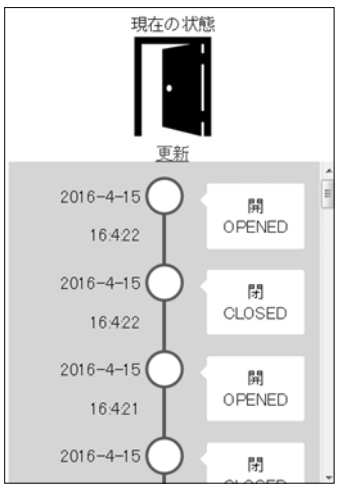
mobile backend JavaScript SDKの設定と初期化

mobile backendとアプリの連携のために、JavaScript SDKを導入します。[設定]→[JS/CSSコンポーネントの追加と削除…]から、「ncmb」を追加します。[追加]を押下したら画面に従い、最新バージョンのSDKを導入します。**図8**のように一覧に表示されたら導入完了です。

次に導入したSDKの初期化処理を行います。「index.html」ファイルを開きます(**リスト2**)。

リスト2にmobile backendから発行されたAPIキーを記述します。YOUR_NCMB_APPLICATION_KEYとYOUR_NCMB_CLIENT_KEYの部分をmobile backendのAPIキーに置き換えてください(注意:必ず**図4**で使用したものと同じAPIキーを使用してください)。

▼図6 ドア監視アプリ完成イメージ



▼リスト2 SDKの初期化(抜粋)

```
// 0. APIキーの設定と初期化
var applicationKey = "YOUR_NCMB_APPLICATION_KEY";
var clientKey = "YOUR_NCMB_CLIENT_KEY";
var ncmb = new NCMB(applicationKey, clientKey);
```

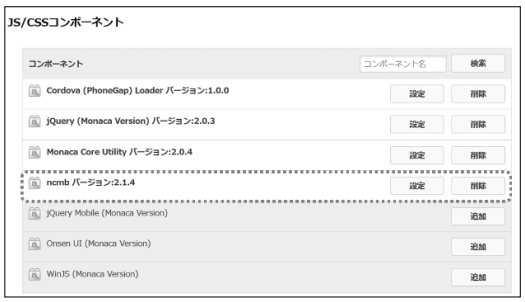
▼図7 ＜Monaca＞プロジェクトインポート

ドアステータスの表示処理

displayStatusImage()メソッドを見てください(**リスト3**)。mobile backend上に保存されたドアの開閉情報を取得するには、まず検索先クラスのオブジェクトを生成します。次に.order("createDate", true)で保存された順番で並べて、最後に.fetch()で1件検索を実行しています。これにより、最新のデータ1件だけを取得できます。

検索成功時は.then、失敗時は.catchでそれぞれ処理を行います。検索に成功した場合は、取得した値をもとにドアの開閉状況を画像で表示しています(**図9**)。

▼図8 ＜Monaca＞SDK一覧



タイムラインの表示処理

displayTimeline()メソッドを見てください(**リスト4**)。先ほどとほぼ同様にmobile backendから値の取得ができます。タイムラインに表示するため、検索条件は.order("createDate", true)を用いて順番に並べ、.limit(30)で上限30件を指定したうえで該当するデータを全件取得.fetchAll()しています。これにより、最新の30件のデータを検索できます。検索に成功した場合は、取得したデータをタイムラインに表示しています(**図10**)。

動作確認

Monacaのプレビュー画面で動作確認をしてみましょう(**図11**)。まずは、開閉データが正しく表示されることを確認します。ドアを開閉して、プレビュー画面の[更新]ボタンをクリックして、画面を更新してみましょう。正しく反映されるか確認してください。

これでドア監視アプリは完成です。アプリを起動しなくても、「いつドアが開いて、いつ閉まったか」をリアルタイムで知ることができたらもっと便利です。この機能を実現するために、mobile backendの「プッシュ通知」を使用します。

▼リスト3 displayStatusImage()メソッド(抜粋)

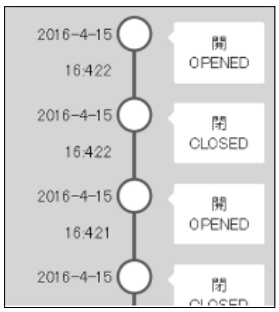
```
function displayStatusImage() {
    // 1. ドアステータスの表示
    // 1.1. DoorDataクラスを検索するオブジェクトを準備
    var Status = ncmb.DataStore("DoorData");
    // 1.2. DoorDataクラスの検索条件設定と検索
    Status.order("createDate", true)
          .fetch()
          .then(function(result){
              // 1.2.1. 検索結果を表示
              if(result.status == 1) {
                  $("#statusImg").attr("src", "image/closed.png" );
              } else {
                  $("#statusImg").attr("src", "image/open.png" );
              }
          })
          .catch(function(err){
              console.log(err);
          });
}
```

▼図9 ドアステータス表示

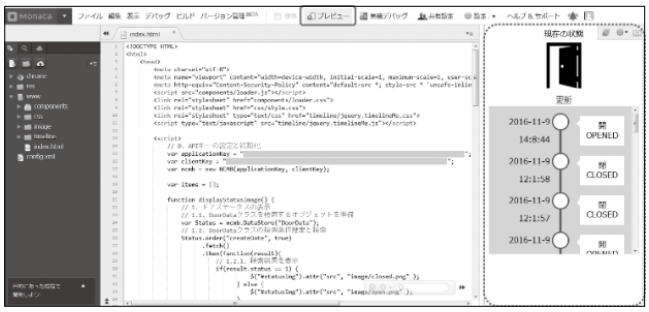
▼リスト4　displayTimeline()メソッド(抜粋)

```
function displayTimeline() {
    // 2. タイムラインの表示
    // 2.1. DoorDataクラスを検索するオブジェクトを準備
    var Status = ncmb.DataStore("DoorData");
    // 2.2. DoorDataクラスの検索条件設定と検索
    Status.order("createDate", true)
          .limit(30)
          .fetchAll()
          .then(function(results){
              // 2.2.1. 検索結果を表示
              for (var i = 0; i < results.length; i++) {
                  var object = results[i];
                  var item = {};
                  item.type = "smallItem";
                  item.label = displayDateTime(convertDate(object.createDate));
                  if (object.status == 1) {
                      item.shortContent = "閉 CLOSED";
                  } else {
                      item.shortContent = "開 OPENED";
                  }
                  item.forcePosition = 'right';
                  items[i] = item;
              }
              $('#timeline-container').timelineMe({"items": items});
          })
          .catch(function(err){
              console.log(err);
          });
}
```

▼図10　タイムライン表示



▼図11　＜Monaca＞プレビュー画面



Part4　プッシュ通知でリアルタイムお知らせ機能導入

● Author 池田 夏藻(いけだ なつも)

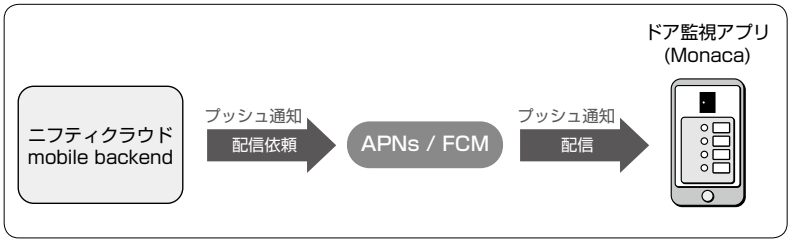
前節でmobile backendに保存されているデータを使って、遠隔でドアの開閉状況を確認するアプリが完成しました。ここに「プッシュ通知」機能を追加して、アプリを閉じた状態でもドアの開閉状況がリアルタイムで把握できるようにしましょう。**図1**の破線③の範囲です。

mobile backendのプッシュ通知機能

mobile backendには、iOS/Android端末に対して、プッシュ通知を配信する機能があります(**図12**)。SDKを導入し簡単な実装を行うだけで、サーバを用意せずともプッシュ通知の実装ができます。また、今回はドアセンサーからの

▼図12 プッシュ通知

▼図13 プッシュ通知のしくみ



信号をもとにプッシュ通知を自動で配信するため、少し高度な使い方となりますが、管理画面からテキストを入力するだけで配信もできます。また、予約配信・端末の絞込み配信・Webページ配信(リッチプッシュ)など、簡単に利用できるのも利点です。

プッシュ通知のしくみ

mobile backendのプッシュ通知は、Google/Appleが提供している通知配信サービスであるFirebase Cloud Messaging/Apple Push Notification Service(以下FCM /APNs)を利用しています(**図13**)。mobile backendはプッシュ通知の配信登録を受けると、FCM/APNsから発行されるdeviceToken(端末を識別するためのキー)をもとに、FCM/APNsに配信依頼をします。依頼を受けたFCM/APNsは認証情報を確認し、端末へ配信を行っています。

アプリにプッシュ通知を組み込む

Node-REDの追加設定

ドアの開閉を感知したらRaspberry Piからプッシュ通知の配信がされるように追加設定をしてください。

▼リスト5 プッシュ通知登録(実際のコードを簡略化)

```
var push = new ncmb.Push();
target = ["android"];
push.set("immediateDeliveryFlag", true)
    .set("message", "Door is opened now")
    .set("target", target);
push.send();
```

プッシュ通知の配信登録処理

プッシュ通知の配信登録(**リスト5**)は、new ncmb.Push()でプッシュ通知オブジェクトを生成したあと、データの登録と同様、.set("key", "value")の形でプッシュ通知に必要な設定をしていきます。ここでは、immediateDeliverFlag(即時配信設定)、message(プッシュ通知のメッセージ)、target(iOS / Android)を配信端末ごとに分けて実装しています。設定後、.send()で配信登録されます。

Android端末でビルドする場合(FCM認証情報)

用意する認証情報は次の2点です。

・Server key
・送信者ID

認証情報の取得は、Firebase Consoleにログインして行います(Googleアカウントが必要)。

```
https://console.firebase.google.com/
```

mobile backend公式ドキュメントに掲載されている情報[注4]を確認のうえ、必要な認証情報を準備してください。

iOS端末でビルドする場合(APNs認証情報)

用意する認証情報は次の4点です。

・開発用(developer)証明書(p12)[注5]
・AppID作成時に記載した「Bundle ID」

注4) チュートリアル(Android):mobile backendとFCMの連携に必要な設定(http://mb.cloud.nifty.com/doc/current/tutorial/push_setup_android.html)

注5) cer形式からp12形式をキーチェーンアクセスより書き出す必要があります。

・Provisioning Profile
・APNs用証明書(p12)[注5]

認証情報の取得には、Apple Developer Consoleへのログインが必要です(Apple Developer Programへの登録・有償)。

```
https://developer.apple.com/account/
```

mobile backendに掲載されている情報[注6]を確認のうえ、必要な認証情報を準備してください。

mobile backendの追加設定

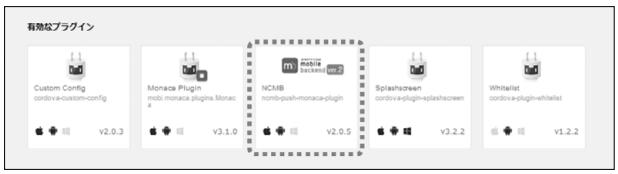
プッシュ通知の配信時、FCM/APNsと認証を行うためのAPIキー／証明書を設定します。右上の[アプリ設定]から、[プッシュ通知]を開き、プッシュ通知を許可します(**図14**)。

・Android端末で受信する場合……「APIキー」にFirebase Consoleで取得した「Server key」を設定する
・iOS端末で受信する場合……「証明書(p12)」にApple Developer Consoleで取得した「APNs用証明書(p12)」を設定します。

注6)【iOS】プッシュ通知の受信に必要な証明書の作り方(開発用)
https://github.com/NIFTYCloud-mbaas/iOS_Certificate

▼図14　＜mobile backend＞アプリ設定(プッシュ通知)

▼図15　＜Monaca＞mobile backend Cordovaプラグイン一覧



Cordovaプラグインの設定

Monacaでmobile backendのプッシュ通知を利用するためにプラグインを設定します。

[設定]→[Cordova プラグインの管理…]で[Nifty(ncmb-push-monaca-plugin)]を有効にします。「有効なプラグイン」の一覧に表示されれば設定完了です(**図15**)。

デバイストークンの取得

index.html(**リスト6**)を見てください。デバイストークンを取得して、mobile backendinstallationクラスに保存します。Android端末の場合は「SENDER_ID」をFirebase consoleで取得した「送信者ID」に置き換えます。

Monacaアプリのビルド

プッシュ通知の受信には必ずビルドが必要です。

Android端末の場合

[ビルド]→[Androidアプリのビルド]から**図16**の手順でデバッグビルドを行います。数秒後にapkファイルが作成されますので、画面に表示されるいずれかの方法で端末にダウンロードをしてください。

iOS端末の場合

[設定]→[iOSアプリ設定]を開き、「App ID」をApple Developer Consoleで取得した「Bundle ID」に書き換えます。証明書登録します。Apple Developer Consoleで取得した「開発用(developer)証明書(p12)」をインポートします。

[プロファイルのアップロード]をクリックして、Apple Developer Consoleで取得した「Provisioning Profile」をアップロードします。

▼リスト6 プッシュ通知受信設定(抜粋)

```
// 3. プッシュ通知の受信設定
document. ↵
addEventListener("deviceready", ↵
function(){
    // 3.1. デバイストークンを取得して ↵
installationに登録する
    window.NCMB.monaca.setDeviceToken(
        applicationKey,
        clientKey,
        "SENDER_ID"  /* Android端末へ ↵
配信する場合に設定する送信者ID */
    );
},false);
```

以上で設定は完了です。[ビルド]→[iOSアプリのビルド]から図16と同様の手順でデバッグビルドを行います。数十秒後にipaファイルが作成されますので、画面に表示されるいずれかの方法で端末にダウンロードをしてください。

▼図16 ＜Monaca＞Androidデバッグビルド

動作確認

ビルドが完了したら動作確認をしましょう。ドアを開閉するとプッシュ通知が届きます(図17)。

Part5 おわりに

● Author Dinh Thuy Duong(ディン トゥイ ズオン)

「ドア監視アプリ」の作成を通して次の内容に取り組んできました。

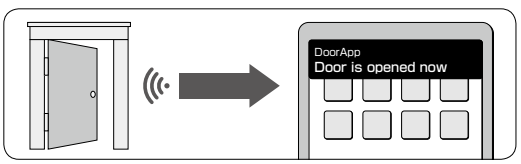
- Raspberry Piを使って、センサーからドア開閉情報の取得
- Node-REDでmobile backendへデータの保存
- mobile backendからデータを取得して、Monacaで監視アプリを作成
- プッシュ通知機能を使ったリアルタイムお知らせ機能の実装(図17)

この取り組みを通じて、T(もの)からデータを取得してI(インターネット)を通して利用するには、データを保管するバックエンド機能(サーバ)は大きな役割を持っていることが理解できたと思います。本来バックエンド機能を自前で構築するとなると、開発だけでなく保守／運用を行う必要があるため、たいへんな工数が必要になります。気軽に楽にIoTのアイデアを形にしていくためには、mobile backendのようなバックエンドサービスを活用すると、いろいろなIoTアプリを作ることができます。

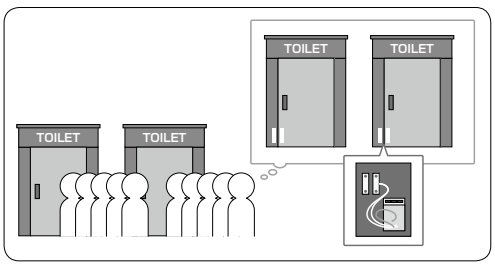
手始めに、このドア監視アプリの構成をそのまま活用してIoTを実現してみてはどうでしょうか。たとえば「家の玄関を監視するアプリ」のようにホームセキュリティIoTを作ってみるとか。あるいは「オフィスのトイレの空き情報通知アプリ」を作ることができれば、もうトイレに並ぶ必要はありません(図18)。

あなたはどのようなアイデアをお持ちですか？ぜひ身近なものからIoT化してみてください！SD

▼図17 プッシュ通知配信イメージ



▼図18 トイレの空き情報通知アプリイメージ



RDBアンチパターン

PostgreSQLとMySQLの失敗と対策

Author 曽根 壮大（そね たけとも） ㈱はてな Twitter @soudai1025

第2回 失われた事実

本連載では、開発の現場で発生しやすいリレーショナルデータベース（RDB）全般の問題をRDBアンチパターンとして紹介していきます。今回のアンチパターンの主人公は、消費税率改正の対応に追われるECサイト担当のエンジニアです。

前回は「データベースの迷宮」というテーマで、カラムの名前やリレーションシップの大切さを説明しました。今回は「失われた事実」と題して、設計の重要性を説明します。

失われた事実

RDBは、“時間軸と直行するような設計”が大切です。ですがそれを使ったサービスとしては、時間軸と直行しないデータ＝履歴を保存することが同じくらい重要です。履歴の保存を怠ると、

- このデータがどのようにして今の値になったかわからない
- ある日を境に売上データと商品マスタの単価データが合わない
- 払い戻しの処理が特別対応となる

のようなケースと戦うことになります。まずは、このような履歴にまつわるアンチパターンを紹介します。

事の始まり

今日は消費税率の切り替え日で、5%から8%へと消費税率の切り替え作業を行う必要がある。ECサイトの担当エンジニアのSさんは、この消費税率変更対応の真っ只中。

Sさん：現状のテーブルがどうなっているか確認しよう（**図1**）。よし、設定マスタテーブルの消費税率のレコードを更新すれば、正常に消費税率が切り替わるな。売上マスタも切り替わってるのが確認できたぞ。日付が変わるタイミングで自動更新するようにバッチも設定したし、これで大丈夫！

――数日後――

サポート担当者：数日前にsoneさんという方が購入された『SQL実践入門』、1冊だけ返品処理したら売上がズレるんですけど。

Sさん：あっ……、払い戻しのときの処理が変更後の消費税率で計算されるからだ……。

このように、設定マスタを作っていても過去の履歴を持っていないと、あとあとの取り消し

▼図1　今の設定マスタテーブルの状態

```
demo=# SELECT * FROM 設定マスタ WHERE name = '消費税率';

 name    | value
---------------------
 消費税率 | 0.05
(1 行)
```

処理で苦労することになります。

問題点

今回のアンチパターンを図にすると**図2、図3**のようになります。この例の場合、「売上id 1」の売上金額は、次のような計算になります(小数点以下切り上げ)。

・最初の計算

(2,580円 × 3個 + 2,592円 × 3個 + 4,200円 × 3個) × 1.05[消費税率] = 29,522円

・返品で個数が、日付で消費税率が変わる

(2,580円 × 2個 + 2,592円 × 3個 + 4,200円 × 3個) × 1.08[消費税率] = 27,579円

・本来あるべき計算

(2,580円 × 2個 + 2,592円 × 3個 + 4,200円 × 3個) × 1.05[消費税率] = 26,813円

このように、本来あるべき数値と変わってしまいます。

似たようなアンチパターン

このほかにも類似例として、次のようなパターンがあります。

▼図2　最初の計算

売上(変更前)

売上id	売上金額	売上日	配送状態
1	29,522	2014-03-31 23:59:59	配送済
2	6,480	2014-04-01 00:00:00	発注中
:	:	:	:

消費税率の状態が隠れており、日付をまたぐと5%から8%へ変わる

カート

売上id	商品id	個数	購入者
1	1	3	sone
1	2	3	sone
1	3	3	sone
2	4	3	sone

設定マスタ

名前	値
消費税率	0.05

商品

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

▼図3　返品で個数が、日付で消費税率が変わる

売上(変更前)

売上id	売上金額	売上日	配送状態
1	29,522	2014-03-31 23:59:59	配送済
2	6,480	2014-04-01 00:00:00	発注中
:	:	:	:

売上(変更後)

売上id	売上金額	売上日	配送状態
1	27,579	2014-03-31 23:59:59	配送済
2	6,480	2014-04-01 00:00:00	発注中
:	:	:	:

消費税率が5%から8%に

カート

売上id	商品id	個数	購入者
1	1	2	sone
1	2	3	sone
1	3	3	sone
2	4	3	sone

設定マスタ

名前	値
消費税率	0.08

商品id 1の個数を3から2に

売上idが1の返品処理をする場合、カートの値のみを変更して再計算すると、本来あるべき値から誤差が生まれる

商品

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

▼図4　商品の名前・価格を上書き

商品名が変わったら……、価格が変わったら……、
売上の事実と不整合が生まれる

商品

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

商品

商品id	商品名	価格
1	SQL実践入門 第二版	2,480
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

▼図5　商品の配送状態を上書き

売上（変更前）

売上id	売上金額	売上日	配送状態
1	27,579	2014-03-31 23:59:59	発注済
2	6,480	2014-04-01 00:00:00	発注中
:	:	:	:

(2)

売上（変更後）

売上id	売上金額	売上日	配送状態
1	27,579	2014-03-31 23:59:59	配送済
2	6,480	2014-04-01 00:00:00	発注中
:	:	:	:

(1)

いつ配送状態が変わったかわからない

売上（途中経過）

売上id	売上金額	売上日	配送状態
1	27,579	2014-03-31 23:59:59	キャンセル
2	6,480	2014-04-01 00:00:00	発注中
:	:	:	:

売上（途中経過）

売上id	売上金額	売上日	配送状態
1	27,579	2014-03-31 23:59:59	再発注
2	6,480	2014-04-01 00:00:00	発注中
:	:	:	:

✥ 過去の事実が失われるパターン1（値）

1つめは値が失われるパターンです。過去の購入商品の金額や商品を上書きしてしまうことで、事実と差異が生じます（**図4**）。

✥ 過去の事実が失われるパターン2（過程）

2つめは状態の変化が保存されない、つまり過程が失われるパターンです。たとえば商品の配送を例にとると、配送状況を上書きしてしまうと途中経過がわからなくなります。

(1)発注済 → キャンセル → 再発注 → 配送済
(2)発注済 → 配送済

の2つは結果のみを見ると同じですが、その過程は大きく違います（**図5**）。このように、途中経過の事実を上書きすることで過程の事実を失うことになります。

◆ ◆ ◆

このように、失われた事実のアンチパターンでは、いずれも正常系のときには問題が見えないのが大きな罠の1つです。今ある事実のみを保存してしまうと過去の事実を失ってしまうので、例外処理を行うときやトラブル時に状況把握する場合に、情報が不足します。このような問題は消費税率がからむEC系のほかにも、管理画面の作業ログなどエンタープライズな実務でもたびたび発生します。

どうすれば良かったのか

このアンチパターンの解決策は、もちろん過去の履歴を保存することです。

たとえば消費税率の問題では、次の2つの設計で今回の問題を回避できます。

- 消費税率に有効期限を持たせることで商品の購入日から消費税率の履歴を遡る（さかのぼ）（**図6**）
- 購入商品に利用した消費税率を保存し、履歴として持たせる（**図7**）

▼図6　消費税率に履歴を持たせる

消費税

消費税率	有効日	失効日
0.05	1997-04-01	2014-03-31
0.08	2014-04-01	null

「消費税」のテーブルを新たにつくり、有効期限を持たせることで、売上日から消費税率を遡ることができ、また自動的に切り替えるしくみもつくりやすくなる

▼図8　最新のレコードを有効のレコードとしてみる

配送状況

売上id	配送状態	作成日時
1	発注中	2014-03-31 11:59:59
1	配送済	2014-03-31 15:59:59
1	納品済	2014-03-31 18:59:59
2	発注中	2014-03-31 23:59:59

「配送状況」のテーブルを新たにつくり、配送状況の履歴を持たせる。状態は更新ではなく追加し、最新のレコードを現在の状況として扱う

▼図7　購入時の消費税率の履歴を持たせる

売上

売上id	売上金額	消費税率	売上日	配送状態
1	29,522	0.05	2014-03-31 23:59:59	配送済
2	6,480	0.08	2014-04-01 00:00:00	発注中
:	:	:	:	

売上テーブルに、購入時の消費税率の情報を持たせる。図6のような消費税率の有効期限の情報がない場合は、これがないと返品時の払い戻し処理に対応できない

▼図9　金融系のシステムでよくみられる設計

口座名	担当名	状態	金額	作成日
曽根	○○会社	振込	600,000,000	2014-03-31 11:59:59
曽根	クレジットカード	引落	-150,000	2014-03-31 15:59:59
曽根	□□電力	引落	-10,000	2014-03-31 18:59:59
曽根	△△会社	振込	500,000	2014-03-31 23:59:59
曽根	○○会社	取消	-600,000,000	2014-03-31 11:59:59
:	:	:	:	:

打ち消しのINSERTも保存しておく

また配送状態の問題では、最新のレコードを有効のレコードとしてみる**図8**のような設計もよく見られます。

金融系のシステムでよく使われる処理としては、削除処理も「打ち消しのINSERT」として保存し、合計値を算出する設計を取ることもよく見られます(**図9**)。

このように、RDBに履歴を保存させる設計はいくつかありますが、次のようなデメリットもあります。

・レコードの保存量が増えるためテーブルサイズが増える
・集計が単純な主キー検索ではなくなるため、テーブルサイズが肥大化した際に検索速度が劣化する

このように、パフォーマンスとトレードオフの設計となります。そのため、これらの課題を解決するために、さらにマテリアライズド・ビュー を利用したり、集計済み結果を保存したサマリーテーブルを作成したりします。

このアンチパターンのポイント

今回紹介したアンチパターンは「後からデータを遡りたいときに事実が失われている設計をしてしまう」ということになります。このアンチパターンに陥っていないか確認する場合は、

・払い戻しなどの取り消し処理に対応できるか
・配送状況などステータス変化を追えるか
・トラブル対応時、欲しい情報が失われていないか

を確認してみましょう。

しかし、前述のデメリットであるパフォーマンスの劣化を考えて、あえて履歴を保存しない設計を取るケースもあります。その場合に大切なこととして、もしRDBの責務内で履歴を持たない場合は、次のように別手段で必ず持たせるようにしましょう。

・遅延レプリケーションを使う（コラム参照）
・アプリケーションログとしてElasticsearchなどの分析ツールに保存する

エンジニアは神ではないので、知らない事実を読み解くことはできません。予測することはできても正しいと判断することはできず、実践ではかなり苦戦を強いられることになります。

また、事実は後から作り出すことができません。一度失われてしまった事実を新たに集める方法もありませんので、初期設計時に事実の履歴についてしっかりと検討することがとても大切になります。そのため、新たなアプリケーションを設計する場合は、このアンチパターンについてもしっかりと振り返る機会を持っていただければと思います。

リレーショナルデータモデルと履歴データ

RDBを活かすにはリレーショナルデータモデルに準拠することがとても大切です。しかしリレーショナルデータモデルはそもそも時間軸と直交するデータモデルのため、今回の題材である履歴データとの相性が悪いのです。

しかし実務では履歴データを取り扱う例は多くありますし、その場合のデータベースの多くにはRDBを使うことになるでしょう。そのため相性の悪いデータを記録するために正規形を崩したり、データ量が増えることでパフォーマンスとトレードオフな設計をしたりする必要が生まれやすいのが、この履歴データです。詳しく知りたい方は奥野幹也さんの『理論から学ぶデータベース実践入門』[注1]をぜひ読んでみてください。

次回のRDBアンチパターン

今回のRDBアンチパターンはいかがでしたでしょうか？　次回はリレーショナルデータベースの要とも言えるJOINがテーマです。ORMを使っている方の中には、知らない内にJOINしてる人もいるのでは？　普段何気なくJOINしてる人にもうってつけのテーマです。次回の「やり過ぎたJOIN」もお楽しみに！ SD

注1）奥野 幹也 著, 技術評論社, 2015年, ISBN＝978-4-7741-7197-5, http://gihyo.jp/book/2015/978-4-7741-7197-5.

RDB TIPS 遅延レプリケーションについて

RDBのレプリケーションには、遅延レプリケーションという機能があります。この機能は指定した時間分、スレーブDBに対して、マスタDBからのレプリケーションを遅延させることができます。たとえば1日遅れのスレーブDBを作ったり、2時間遅れのスレーブDBを作ったりすることができます。遅延レプリケーションはDBに柔軟な設計を与えてくれ、おもな目的として次のような場合に利用されます。

・マスタDB上で行われた誤った作業から保護する
・システムのデバッグ時の再現手法として使う

このように、遅延したスレーブDBを作ることができる機能が遅延レプリケーションです。遅延レプリケーションはとても便利ですが、行っていることはDBの複製ですので物理的なコストは高くなります。またバグなどでデータが壊れてしまった場合に遅延予定時間を超えると、当然反映され、戻せなくなります。そのため、バックアップは別に必ず取るようにしましょう。

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第39回 困ったときは

サルミアッキ、パクチー、ドクターペッパー、ルートビール、チェリーコーク、これらに何か通じるものがあると確信した担当がダメ出しをするマンガが読めるのは本誌だけ。

to be Continued

GitHubライクなリポジトリ管理サービスを提供している某サイトが、DBのレプリケーションを再構築しようとしたら、誤ってマスターDBのデータを削除してしまい、用意されていた手順で復旧を試みようとしたが、複数あったバックアップがすべてエラーで機能していなかった、という事件がありました。最終的にはLVMのスナップショットで、数時間分のデータロストはあったが復旧したとのこと。この話題を聞いたら他人事じゃねぇ……って各所チェックしましたね。バックアップは取ってるから大丈夫、って思っても、実際にリストアを試みたらエラーになる話もあるので、バックアップ確認だけでなく、リストア試験もやりましょうね。ああ、怖い怖い。

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16回 結合条件と抽出条件の区別をイメージで考える

SQLのSELECT文でデータを抽出するとき、JOIN ON ～、WHERE ～、HAVING ～といろいろな条件を指定しますが、それぞれの意味合いをきちんと理解していますか？　結合条件（JOIN ON）と抽出条件（WHERE）が組み合わさったときは、とくにややこしいです。今回は基本に立ち返り、SELECT文の処理の流れを解説します。

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。

ON句の本当の意味が知られていない?

大阪を中心に20年間システム開発に携わったあと、現在は東京で仕事をしている「SQLの伝道師」ことジーワンシステムの生島です。

「ちょっとこれ見てもらえますか？」

と、いつものように大道君が尋ねてきました。話を聞くと、学習塾や学校向けの成績管理システムで、成績レポートを出す機能の中の「学費免除などの対象になる特待生であるかどうか」の情報を付加して出力するモードでバグが起きていたそうです。おおまかに書くと**図1**のように成績データ（誰がどの試験でどんな成績を取ったかの明細情報）に、生徒マスタから名前を、科目マスタから科目名を付与し、特待生マスタから特

▼図1　試験の成績データとマスタ情報をもとに成績レポートを出す機能

待生情報を付与して成績レポートを作る機能です。

ほしかったのは**図2**の(A)のように特待生マスタで削除フラグ = 1の場合は特待生ランク欄を空白にしたレポートでしたが、これが(B)のように削除フラグ = 0で特待生ランクを持っている生徒の情報しか表示されなくなるというバグを起こしていました。このバグで使われていたのが**図2**のSQL文-2です。これを、サブクエリを使うSQL文-1のように修正したということです。

「それでひとまず狙いどおりに結果(A)が得られるようにはなりまして」

「ああ、これはまあよくあるバグやね」

「でも、サブクエリを安易に使って性能問題起こすケースも多いみたいですし……これ、大丈夫でしょうか？　問題あるようなら、ほかの方法はありませんか？」

「まあ、うかつに使うと性能トラブル起こしやすいのは確かやね。それに、実はこのコードならほかの方法があるし、そのほうがええよ。具体的には**リスト1**のSQLを使えばいい」

「えっ、抽出条件の0 = t.削除フラグをWHERE句からON句に移せばいいんですか？」

「そう。これに驚くってことは、大道君もまだON句の本当の意味を知らないみたいやね」

「ええっ……」

「大丈夫、知ってしまえば簡単なことやから」

実は私が開催しているSQL勉強会でこの問題をよく扱うのですが、正解率は10%にも届かないぐらいで、SEでも知らない方が大半です。最近の勉強会でもこれで驚かれたところなので、本記事で扱うことにしました。SQLの基本的な仕様なのにほとんど知られていない「ON句の本

▼リスト1　サブクエリの代わりにON句を使用する

```
SELECT
    (..略..)
FROM
    成績データ r
    INNER JOIN 生徒マスタ s
        ON r.生徒ID = s.ID
    INNER JOIN 科目マスタ k
        ON r.科目ID = k.ID
    LEFT OUTER JOIN 特待生マスタ t
        ON r.生徒ID = t.生徒ID
       AND 0 = t.削除フラグ
```

▼図2　修正前後の出力結果

(A)ほしいレポート

生徒ID	名前	科目ID	科目名	点数	特待生ランク	削除フラグ
1	石野 一基	1	英語	19		
1	石野 一基	2	数学	27		
2	岩崎 研二	1	英語	18		
2	岩崎 研二	2	数学	25		
3	岩間 直美	1	英語	93	B	0
3	岩間 直美	2	数学	46	B	0

特待生マスタで削除フラグ=1の場合は
特待生情報を出さないようにしたかった

```
【SQL文-1】
SELECT
    (.. 略 ..)
FROM
    成績データ r
    INNER JOIN 生徒マスタ s
        ON r.生徒ID = s.ID
    INNER JOIN 科目マスタ k
        ON r.科目ID = k.ID
    LEFT OUTER JOIN
        (SELECT * FROM 特待生マスタ
         WHERE 削除フラグ = 0) t
        ON r.生徒ID = t.生徒ID
```

(B)実際に出力されたレポート(バグ)

生徒ID	名前	科目ID	科目名	点数	特待生ランク	削除フラグ
3	岩間 直美	1	英語	93	B	0
3	岩間 直美	2	数学	46	B	0

ところが、特待生フラグ=0 の特待生の
データしか表示されなくなってしまった

```
【SQL文-2】
SELECT
    (.. 略 ..)
FROM
    成績データ r
    INNER JOIN 生徒マスタ s
        ON r.生徒ID = s.ID
    INNER JOIN 科目マスタ k
        ON r.科目ID = k.ID
    LEFT OUTER JOIN 特待生マスタ t
        ON r.生徒ID = t.生徒ID
WHERE
    t.削除フラグ = 0
```

当の意味」とは何なのでしょうか？

SELECT処理の流れをイメージしよう

当連載の第14回(本誌2017年4月号)でも「SQLは文法から考えても理解できない。表をカタマリで操作するイメージを持とう」という話をしましたが、今回も同じです。これを機会にSELECT処理の流れを全体としてイメージできるようにしましょう。それができれば性能アップの勘どころもつかめますし、余計なバグも減らせます。

まずは**図3**です。これが2つのテーブルをJOINして、WHERE句で絞り込み、ほしい項目を選んでGROUP BYをかけ、HAVINGでまた絞り込む、という流れの全体像です。

「テーブルAとBは今回の例で言えばどれに該当する？」

「Aが成績データで、Bが生徒マスタとかでしょうか」

「そうやね。1人がいくつもの科目の試験を受けるから、Aの件数のほうがBより多くなる。じゃあ、結果1が意味するものは？」

「試験の成績データに、生徒の名前を付加したもの……でしょうか」

「そのとおり。じゃ、結果1と結果2の違いは？」

「件数が減っているわけですね。たとえば、4月以降の試験のデータだけとか条件を付けるとこうなりそうです」

「そう。WHERE句に何か条件を付けると、JOINの結果をその条件で抽出することになる。普通は抽出条件と呼ばれるのがこれやね。じゃ、結果2と結果3の違いは？」

「GROUP BY……あ、生徒を単位にグループ化するとこうなるかな？」

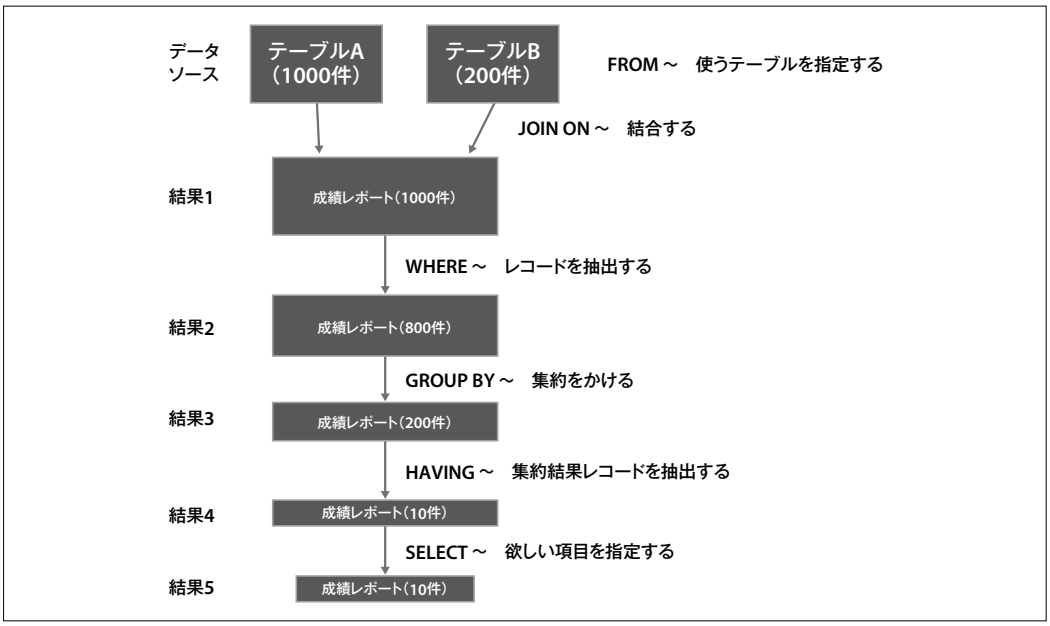
「そう、複数科目の平均点とか合計点、最大最小みたいな集約データがほしいときはこれをするわけや。じゃ、結果3と結果4の違いは？」

「200件から10件に一気に減ってますけど、HAVINGということは……合計点の成績上位10人を選ぶとか？　そんな処理ですか」

「そういうこと。じゃ、最後、結果4と結果5の違いは？」

「件数が減らずに横幅が狭くなってるというのはつまり、カラムを選択しているわけですね？

▼図3　SELECT文の処理の流れイメージ



SELECT句の直後で必要なカラムを列挙する部分がこれですか」

「まさにそういうことやね。これがSELECT文の全体像。こういうイメージ、わかっとった？」

「わかっていませんでした……でも、なるほど‼って感じです！　そうか、それでGROUP BYのあとはWHEREじゃなくてHAVINGなんですね。レコードを抽出するところが2ヵ所あるからそれを区別するために、WHEREとHAVINGって違う言葉を使っているんですね。ああ、こう描けばイメージできますね」

「そやろ？」

結合条件と抽出条件を区別する

こうしてみると、SELECT文の処理の流れというのは、実は非常にシンプルなものであることがわかります。自由に手続きを記述できる手続き型言語と違い、SQLでやっているのはこうした決め打ちの処理ばかりですから、一見どんなに複雑に見えてもやっていることは単純な操作ばかりなのです。

「もう1つ大事なのが、結合条件と抽出条件を区別すること。結合条件はどこに書く？」

「JOINのあとのON、ですよね？」

「そう。まあ昔のOracleだと結合条件もWHEREに書かなきゃいけない時代もあったんやけど、標準SQLではJOINを使ってON句に書くことが推奨されていて、Oracleもそれに対応してきているから現在はそれが主流。で、気をつけなきゃいけないのが、同じ条件式でもそれを結合条件としてON句に書くのと抽出条件としてWHERE句に書くのとでは、結果が違ってくることがあるってこと。今回のバグはその影響やね」

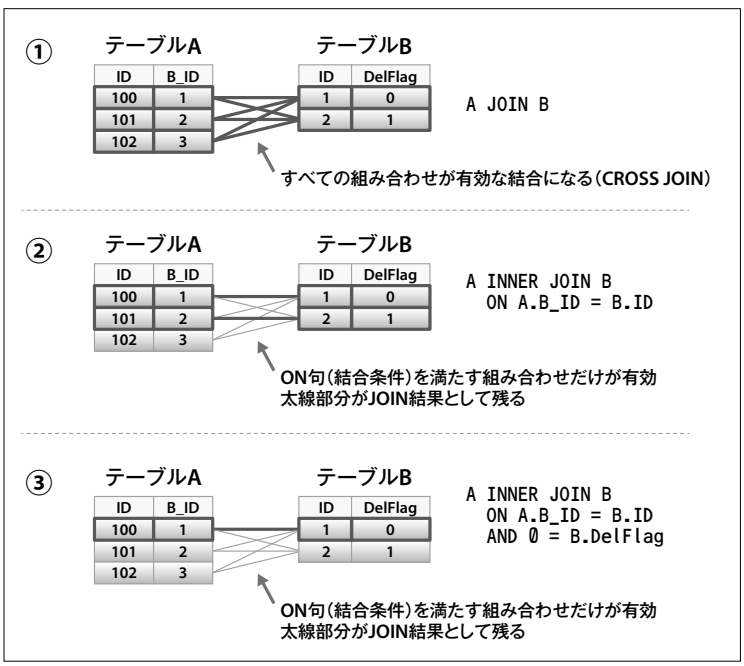
「え、えっ……」

実はここが、私のSQL勉強会で正解率が10％にも届かないという、問題の部分なのです。大道君がわかっていないのも無理はありません。では、いよいよ「ON句（結合条件）の本当の意味」を詳しく語ってみるとしましょう。

ON句の本当の意味は「結合」のフィルタリング

図4を見てください。テーブルAとBのJOINをするとして、ON句を指定せずJOINする、あるいは`SELECT * FROM A,B;`のようにJOIN句自体を省略すると、①のようにAの3件とBの2件の「すべての組み合わせ」の結合を作ります。このような無条件のJOINをCROSS JOINとも言います。もし1万件のテーブル同士のCROSS JOINをすると結果は1億件になるわけです。実用的なシステムでこんなJOINを使うことはまずありませんが、これがJOINの本来の意味です。

▼図4　INNER JOINのしくみ



一方、実用的に使われるJOIN操作はほとんどの場合、②のように片方のテーブルのキー項目を使って結合する処理になり、A.B_ID = B.IDのように等号の左右にそれぞれのテーブルの項目が現れます。これは、「CROSS JOINのすべての組み合わせをチェックしながら、ON句で指定した条件を満たす結合だけを有効なものとして残していく」処理と考えてください。実際にそのアルゴリズムで処理されるわけではありません。たとえばインデックスがあればそれを使うので、「すべての組み合わせのチェック」は行いません。しかし結合条件の動作を理解するうえではCROSS JOINをベースに考えるとシンプルです。結局②の太線部分の結合がJOINの結果セットとして残ります。

「そうですね、これが普通ですよね。いつもこのパターンなので、これしか思いつきませんでした」と大道君。

「まあ普通はそうなんよ。ところが結合条件には片方のテーブルだけを指定することもできるわけや」

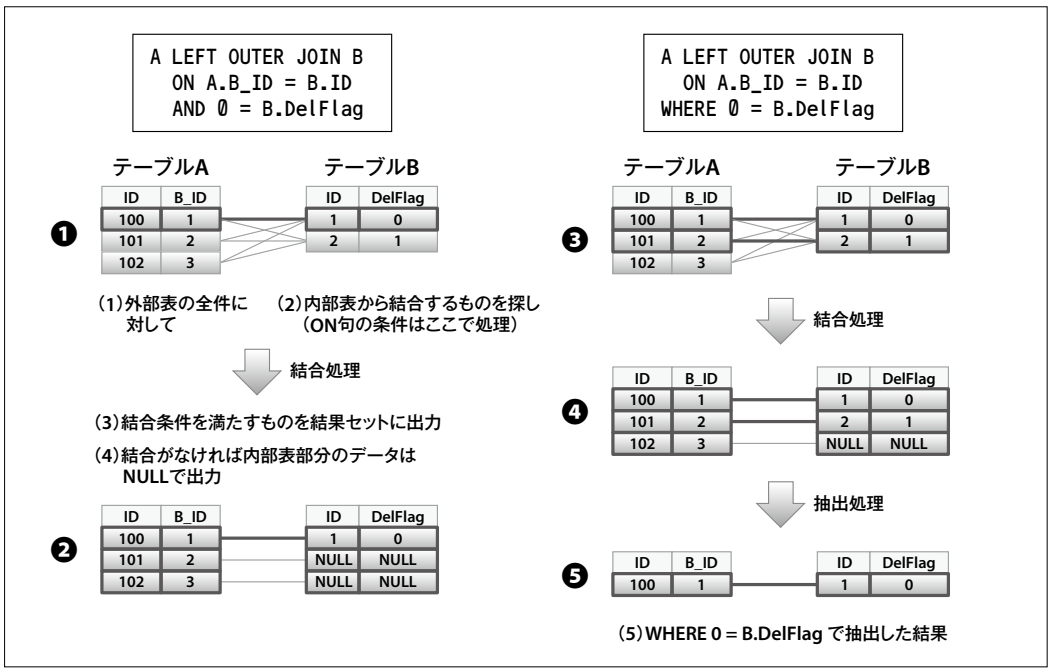
それが③のパターンで、②に0 = B.DelFlagという条件をANDで加えています。Bテーブルのカラムだけを結合条件に指定することもできるわけです。これも先ほどと同じく「CROSS JOINの結合のうち、ON句で指定した条件を満たす結合だけを残していく」という考え方で理解できます。結局太線部分の結合がJOINの結果セットとして残ります。

「えっと……あれ？　**リスト1**で使ったのはLEFT OUTER JOINですよね？」

「そう、そこがポイントで、OUTER JOINだと結果が違ってくるんよ」

0 = B.DelFlagという条件をOUTER JOINのON句に書いた場合を**図5**の❶、❷に整理しておきました。OUTER JOINでも「CROSS JOINのすべての組み合わせをチェックしながら、ON句で指定した条件を満たす結合だけを有効なものとして残していく((1)～(3))」という基本は同じですが、結合がなかった場合の処理が違います。OUTER JOINでは、(4)外部表のレコードのうち、内部表に結合条件を満たす

▼図5　OUTER JOINでON句（結合条件）またはWHERE句（抽出条件）を指定した場合



結合を作れるレコードが存在しなかったものについては、内部表のレコード部分をNULLとして結果セットに出力します。結局**図5**の❷のような結果セットが残ります。これはテーブルAに対して、テーブルBを`WHERE B.DelFlag = 0`の条件で抽出したサブクエリをLEFT OUTER JOINで結合した場合と同じ結果です。INNER JOINの場合の結果(**図4**の③)と違うことを確認してください。

一方、LEFT OUTER JOINでWHERE句に`0 = B.DelFlag`を指定した場合は、**図5**の❸～❺のように処理されます。❹に`B.DelFlag = 1`のレコードが残っていること、そこに`WHERE 0 = B.DelFlag`で抽出をかけると❺のようになり、結果が❷と違うことを確認してください。

「ああ、つまり、`0 = B.DelFlag`の働く場所が違うんですね。ONに書くと結合の処理で働き、WHEREに書くと抽出の処理で働いて最終結果が違ってくる、と……」

「そう、これが結合条件と抽出条件の違い。わかったかな？」

「OUTER JOINのWHERE句に書くと、INNER JOINのON句に書いたときと同じ結果ですね……」

「そう。だからINNER JOINのときは、ONに書いてもWHEREに書いても結果セットには差はないよ。OUTER JOINのときに差が出てくる。今回のバグはそのケースだったわけだ」

「なるほど……!!」

OUTER JOINに注意!

「OUTER JOINをするときは、結合条件と抽出条件の違いに気を使わなければいけないんですね」

「そうなるね」

「こういうことが起きるのは、うーん……」と何やら考え込む大道君。「いわゆる、外部表に対して内部表の一部をJOINする必要があるケースですよね？　内部表のほう、今回で言えば特待生マスタにはすでに無効になったデータがあるから、それを除外してOUTER JOINしなきゃいけない。そのための、内部表の無効データを除外する条件はONに書かなければいけない。これをWHEREに書くと結果が違ってしまう、ということ？」

「そのとおり！」

「今回のもそうですけど、この問題、外部表に対してちょっとした情報を付加するためにテーブルを追加したようなときに起こりやすそうですね。特待生マスタなんてたぶん最初はなかったと思うんですよ。あとで必要になってテーブル追加したんじゃないでしょうか。だからSELECTするときにOUTER JOINが必要になって、そこで無効なデータを除外しようとしたときに、うっかりしやすい？」

「まさにそうやね。比較的発見しやすいバグだとは思うけど、気をつけてや」

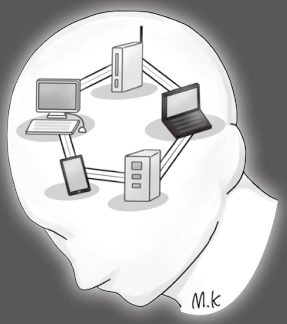
再び、SQLはイメージで考えよう

くどいようですが、この「結合条件と抽出条件」の違いをわかっていないとバグや性能悪化を誘発するコードを書いてしまいがちなのに、現実にはほとんど理解されていません。

「僕もわかっていませんでしたけど、こうしてイメージがわかると簡単な話なんですね……」と大道君。

「何度も言うけど、SQLは文法で考えてもピンと来んのよ。だからデータのカタマリを切り出してつなげていく、そのイメージを持ってほしいんよね。イメージで考える習慣を持てばSQLは本当に簡単な言語なんやから……」

今回はそのイメージをつかんでもらうために**図3～5**を書きました。SQLの解説と言うと、SQL文そのもののほかにSELECT結果の無味乾燥な表そのものしかない場合が多いですが、それだけではなかなか「イメージを持つ」のは難しいものです。今回の記事がそのために役に立つことを願っています。SD



仮想化の知識、再点検しませんか?
使って考える仮想化技術

本連載は「仮想化を使う中で問題の解決を行いつつ、残される課題を整理する」ことをテーマに、小さな仮想化環境の構築・運用からはじめてそのしくみを学び、現実的なネットワーク環境への実践、そして問題点・課題を考えます。仮想化環境を扱うエンジニアに必要な知識を身につけてください。

Author
笠野 英松(Mat Kasano)
オフィス ネットワーク・メンター
URL http://www.network-mentor.com/indexj.html

第13回 仮想環境の運用管理(2)

連載後半「仮想ネットワーク環境で使ってみよう〜現実的な使い方」の7回目です。

前回は、ホスト物理システムと仮想環境全体は専門の運用管理部門が担当し、個々の仮想マシンの運用については、利用部門に運用管理を任せるのが合理的であろうというお話をしました。今回は前回に引き続き、仮想環境の運用管理で必要となる各種管理ツールの利用方針などを解説します。

運用管理には、KVMで標準的に用意されている機能や技術を使用しますが、これらはKVMに精通した技術者(以下記事中では"KVM技術者"と書くことにします)が知っている(知っているべき)ものであって、KVMに詳しくはない技術者(以下記事中では"一般的な技術者"と書くことにします)には縁遠いものです。個々の仮想マシンの運用管理を利用部門に任せるのであれば、まずはこの問題を洗い出してみましょう。

基本利用ツールと実装化

KVMを運用管理するために標準で用意されているのは、次の4つの基本的なツールです(これらをまとめて「仮想環境運用管理インフラ」と呼ぶことにします)。

・virt-installコマンド:仮想マシンを作成するCUIツール
・libvirtdデーモン:ライブラリコールを処理し、仮想マシンを管理し、ハイパーバイザを制御するデーモンサービス
・virt-manager:仮想マシンマネージャー。仮想マシンを管理するグラフィカルツール
・virshコマンドラインツール:仮想マシンマネージャーの管理API。libvirtサーバへアクセスし、コマンドラインまたは特別な仮想化シェルから仮想マシンとハイパーバイザを管理し、制御するCUIツール

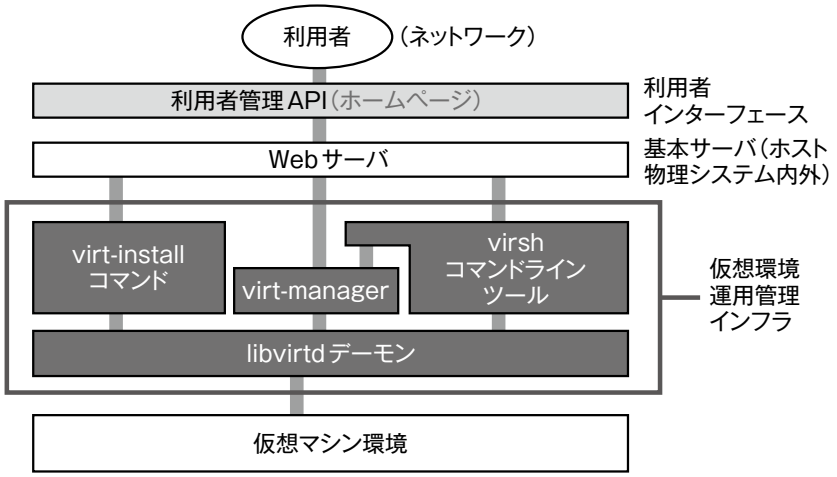
ここでの問題は、上記のようなツールはKVM技術者以外では使いこなせないことです。

しかし、これらを使わなければKVMの基本的な運用管理(インストール、起動/休止/復帰/停止/再起動/強制停止、基本設定、状態/設定確認、など)ができません。

そこで、専門技術者ではなく一般的な技術者の目線に立った操作を提供するために一般的な「利用者管理API[注1]」が必要となります。APIを用意するということはつまり、一般的なアプリケーションのWebブラウザからの管理操作を想定した、物理ホストシステム内・外のWebサーバ上の、ホームページデザインをすることにほかなりません(**図1**)。

注1) ここで言う一般的な「利用者管理API」とは、KVMが提供するインフラツールの機能を、一般的な技術者が利用可能な一般的なアプリから使えるようにするAPIのこと。

▼図1　仮想環境運用管理インフラと利用者管理API



セキュリティ基本ツールと実装化

KVMの仮想化管理のリモートあるいはセキュリティのために、ホスト物理システムのためのツールと仮想マシンのためのツールが用意されています。

ホスト物理システムのためのツールとしては従来からの、Linuxマシンのリモート管理ツールがあります。一方、仮想マシンのリモート管理用には、同様なセキュリティインフラを利用するKVMツールが提供されています(**図2**)。

ただし、いずれも、これらのツールを利用するリモートクライアントシステムはlibvirtdがインストールされた(Linux)システムであることが前提です。

つまり、リモートクライアントシステムのOS／ソフトウェアに制限がある、ということ、ここが一般的な技術者が、そして、一般的なシステムで運用管理を行う際の大きな問題点です。また、一般的な技術者が使いづらいCUIではなく、どのように一般的な技術者が利用可能なGUIにすべきかも問題です。

仮想マシンの障害対策機能

KVMでは仮想マシンの障害の事前対策用に、仮想マシンイメージのバックアップ／復元、あるいは移動などを行うことができる3つの標準機能が用意されています(**図3**)。

ライブマイグレーション機能

この機能は、仮想マシンが稼働中あるいは停止中にかかわらず、1つのホスト物理システムで実行中の仮想マシンを別のホスト物理システムに移動することを可能にするしくみです。基本的にはこの機能の利用者はホスト物理システムの管理者となるでしょう。

なお、移動する仮想マシンのイメージは、ネットワーク接続されている共有ストレージに格納することが一般的です。

ライブマイグレーション機能の利用用途は次のようなケースです。

ホスト物理システムの負荷分散

負荷が高いホスト物理マシン上の仮想マシンを負荷が低いホスト物理システムに移動することで、ホスト物理システムの負荷分散が図れます。

▼図2　仮想マシンのセキュアリモート運用管理のためのKVMの基本機能
（参考：Red Hat Enterprise Linux 6／仮想化管理ガイド／第6章 ゲストのリモート管理 より）

1. SSH/libvirtdリモート管理

リモート管理システム（Linux）に、libvirtd、および、virt-manager、そして、ssh（クライアント）をインストールしておき、SSHトンネル内のlibvirt管理用接続により、virt-manager／VNCコンソールで仮想マシンを管理します。

2. TLS/SSL/libvirtdリモート管理

SSHトンネルの替わりにTLS/SSLトンネルを利用します。TLS/SSL接続のために、CA（認証局）およびサーバ、そして、クライアントの証明書を作成する必要があります。これは通常のSSL接続のための双方向証明です。

3. トランスポートモード利用リモート管理

libvirtでは上記のSSHやTLS/SSLなどのほかのトランスポートツールの利用が可能です。

①UNIXソケット

UNIXドメインソケットでIPC（Inter-Process Communication、プロセス間通信）を行なう方法です。ソケットは暗号化されないので認証にSELinuxやUNIXパーミッション（rwx属性）を使用します。標準ソケットは/var/run/libvirt/libvirt-sockと/var/run/libvirt/libvirt-sock-ro（読み取り専用接続）です。なお、UNIXドメインソケット(*)はローカルマシン上でしか利用できないので、リモート接続でソケットを使うならば、これと、もうひとつのソケット（インターネットソケット）通信を（セキュアトンネルも）使って連携させます。

(*)ソケット

プロトコルファミリー	内容
PF_UNIX	UNIXドメイン。ローカルシステム内通信
PF_INET	IPv4インターネットプロトコル／ネットワーク通信
PF_INET6	IPv6インターネットプロトコル／ネットワーク通信

②リモートURI（Uniform Resource Identifier）

リモートホスト（KVM物理ホスト）のqemu URL（qemu://リモートホスト/）を指定して直接、KVM物理ホスト上のqemu-kvmに接続してvirshとlibvirtによってKVMを操作します。

✷システムのハードウェア分離と稼働継続

仮想マシンを安全に別のホスト物理システム内に移動することで、ハードウェアデバイスのアップグレード、追加または削除などのほか、システムのメンテナンス時にホスト物理システムを（ライブで）停止することなく、安全にシステムの継続性が図れます。

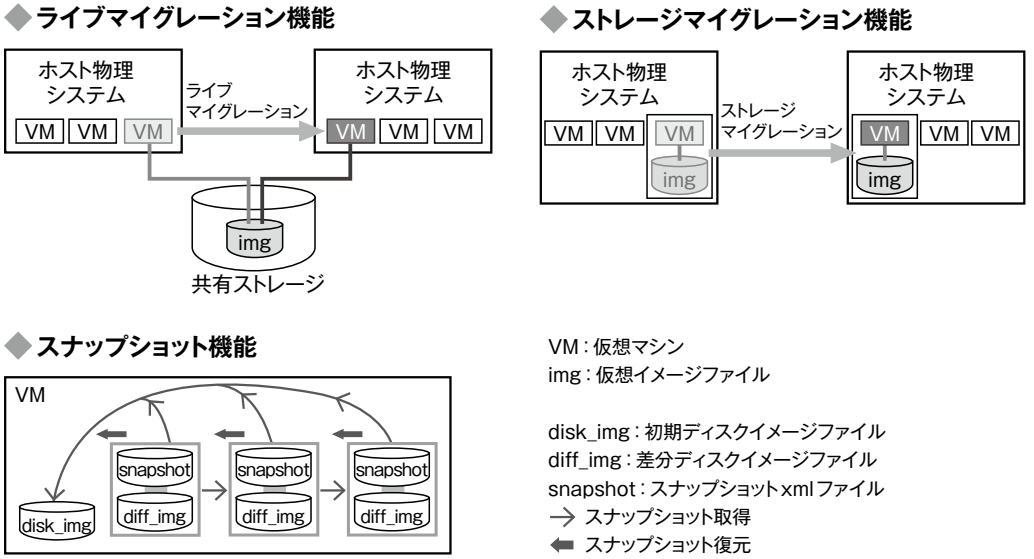
✷耐震、耐災害用の場所移動

震災や災害などに対する仮想マシンの退避用のしくみで、危険性のある、あるいは被災場所の仮想マシンを別の安全な場所に移動できます。

✷バックアップ／復元

さまざまな障害の事前対策（事前バックアッ

▼図3　仮想マシンの障害対策機能



プと事後復元)として利用できます。

ストレージマイグレーション機能

ライブマイグレーションとは異なり、仮想マシンイメージを含むストレージを移動させます。

基本的な利用の要件や用途はライブマイグレーションと同様です。この機能の利用者もホスト物理システムの管理者となるでしょう。

スナップショット機能

仮想マシンの状態を取得・保持する機能で、指定された時点の仮想マシンのディスク、メモリ、およびデバイスの状態が対象です。

利用用途として、OSインストール直後や正常時のイメージを保存しておき、異常時に保存しておいたイメージに戻すことでそのトラブルから即復帰させたり、あるいはまた、教育システムなどで常に一定の設定状態を保持しておき、授業の開始時に同じ初期状態に戻したり、など幅広い用途が考えられます。

なお、これを利用する場合、CUIのvirshコマンドを使う前提です。

この機能の利用者は、仮想マシンの利用部門の管理者(一般的な技術者)になると考えられます。したがって、仮想マシンのリモート運用管理と同様に、利用者管理APIのWebサーバ上のホームページデザインが必要です。

次回予告

次回は、今回の洗い出しで課題となった、「仮想環境運用管理インフラ」の一般的な「利用者管理API」の実装について、「セキュリティツール」とWebサーバ経由の、リモート運用管理の機能の例を取り上げて、具体的な方法を考えてみます。SD

連載各回では、読者皆さんからの簡単な「ひとくち質問(QA)コーナー」や「何かこんなこともしてほしい要望(トライリクエスト)コーナー」を設けて「双方向連載」にできればと思っていますので、質問や要望をお寄せください。

宛先：sd@gihyo.co.jp
件名に、[仮想化連載]とつけてください

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

presented by
Japan Android Group
http://www.android-group.jp/

第15回 新しくなったAndroid Wearは何が変わったのか!?

Androidは世界で出荷される約9割のスマートフォンに搭載される標準OSです※。そのため、多くのAndroidアプリが開発され続けており、そして多くのエンジニアが活躍しています。Androidで広がる新しい技術に魅了されたエンジニアが集うコミュニティもあり、そこでは自分が愉しむための技術を見つけては発信しています。その技術の一幕をここで紹介します。

※Gartner Worldwide Smartphone Sales to End Users by Operating System in 3Q16

中谷 克紀(なかたに かつき)
日本Androidの会 神戸支部、GDG神戸
Twitter:@KatsukiNakatani
http://kobegdg.blogspot.jp/

Android Wear 最新動向

Android Wearのこと、みなさんどれくらいご存じでしょうか? Wear用のOSはリリースされた1.0から1.4になるまでマイナーバージョンアップを行ってきました。マイナーバージョンアップと書いてはいますが、Android Wear 1.0はKitKat 4.4ベースで、Android Wear 1.4はMarshmallow 6.0ベースになるため中身は大幅に変わっています。ただユーザ目線では、電話のAndroid 4.4からAndroid 6.0で変わった(たとえば、HoloテーマからMaterial Designのテーマに変更)ほどには、Wear OSの見た目に大きな変化(進歩)は見られなかったように思います。

しかし、リリースから約2年を経たGoogle I/O 2016で、Android Wear 2.0(Nougat 7.1.1ベース)が発表されました。バージョン番号の変化(1.xから2.0)のとおり、ユーザにも、開発者にとっても大きな変更が入っています。

昨今のAndroid OSは発表後すぐにリリースされるのではなく、Developer Previewとしていくつかのリファレンス端末(電話の場合はNexusやPixel)向けにイメージファイルが提供され、開発者のフィードバックを経てリリースするサイクルになっています。そのためWear 2.0についても同様に、Huawei WatchとLG Watch Urbane 2nd Edition用にDeveloper Previewビルドがリリースされ、開発者とのフィードバックを繰り返したあと、2017年2月に「LG Watch Style」「LG Watch Sports」に搭載されて発売となりました。デバイス形状は、1.0ではスクエア型のデバイスがリファレンス端末でしたが、昨今ではサークル型のデバイスがリファレンス端末になっており、サークル型が主流になっています。

また、ここにきて良い流れになってきているのが腕時計メーカーの参入の増加です。1.0のころはいわゆるスマートフォンメーカーがAndroid Wear搭載デバイスを開発・発売していましたが、CASIOやFossil、TAG Heuerなどの腕時計メーカーがデバイスを発売するケースが増えてきました。利用者は、お気に入りの腕時計メーカーでAndroid Wear端末を手にすることのできる機会がやってきそうです。このことも開発者にとっては利用者が増えるという意味で良い傾向になっています。

さて、既存端末への2.0アップデートですが、本稿執筆時点で「Fossil Q Founder」「CASIO Smart Outdoor Watch WSD-F10」「Tag Heuer Connected」へ早々に、2.0へのアップデートが配信されました。Android OSはバージョンの断片化が問題になっていますが、これらのバージョンアップが早めに流れてくることを見れば、Wearに関しては少しずつですが問題解決に向

かって良い流れになっているのではないかと思います。

2.0になって何が変わったか開発者はどう実装すべきか

まず1.x時代のおさらいをしましょう。

1.x時代は、Android Wear端末は、母艦となるAndroid OS(一部の機能はiOSとペアリングしても利用可)が搭載されている電話(もしくはタブレットなど)とBluetoothまたはWi-Fiで接続し、ペアリングを行う形で利用する前提の設計になっていました(**図1**)。

2.0では、このペアリングを前提とするアプリ設計が必須ではなくなりました。Wear上のアプリは、単独でPlay Storeからインストールし、単独で利用できるようになります。そのために開発者への変更がいくつかあります。

①Wearのapkファイルが直接Play Storeへアップロード可能になった

1.xの場合は、**リスト1**のように、電話用のパッケージファイルにWearのパッケージを含む形でビルドを実行し、電話用とWear用とを一緒に内包したapk[注1]をPlay Storeへアップロードしていました。インストールについても、電話側でPlay Storeを開いてアプリをインストールすることで、Wear側にアプリがインストールされるしくみでした。

この状況は、Wearを使用していないアプリ利用者からすれば、無駄にアプリのダウンロードサイズが大きくなるというデメリットを生んでいました。

Wear 2.0から開発者は、電話用のプログラムとWear用のプログラムをビルドしたら、別々のapkとしてPlay Storeへアップロードすることになります(Wear端末にもPlay Storeが提供され、Wear端末だけでインストールができます)。ただし、1.xのWear OSをサポートしたい場合は、引き続き電話側のapkにWear用のapkを含める必要があります[注2]。

なお2.0では、単独で動作するアプリ(電話に依存しない)についてはStandaloneアプリであることを明記するために、**リスト2**のようにマニフェストでパラメータをtrueにする必要があります。

②Wear単体でネットワークアクセスが可能になった

1.xの環境では、開発者はWear用アプリからネットワークアクセスをすることはできませんでした(Wi-Fi搭載端末であっても)。そのため、外部から何かの情報を取得したいときは、Message APIまたはData Layer APIを利用し、電話をプロキシのようにして実装する必要がありました(**図2**)。

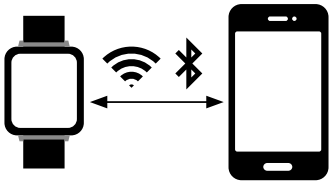
注1) Androidのアプリケーション配布形式。

注2) 両方をサポートする場合は、バージョンコードの管理などが非常に複雑になります。詳細はAPI Overview内のApp Distributionを参照してください。https://developer.android.com/wear/preview/features/app-distribution.html

▼リスト1 phoneアプリのbuild.gradleへwear用のプロジェクトを包括するように記述

```
dependencies {
    //wearable用のモジュール名をphoneのbuild.gradleへ追記
    wearApp project(":wearable")
}
```

▼図1 電話と時計でペアリングして使うことが前提(1.xの場合)



▼リスト2 単独動作のWearアプリであることをAndroidManifestに記述

```
<application>
 ...略...
  <meta-data
    android:name="com.google.android.wearable.standalone"
    android:value="true" />
 ...略...
```

2.0ではこれが大きく緩和され、Wear単体で外部へ通信できるようになりました。外部への通信については、通常のAndroidと同様に利用することができます。ただし、Wear端末は電話よりもCPUパワーが弱く、なおかつバッテリー容量も非常に少ないため、可能な限り通信部分はまとめ、サーバ側でできることはWear端末で処理させないようにし、極力バッテリーを使用しない実装をするように心がけましょう。

これによりMessage APIやData Layer APIを介してネットワーク通信をする必要はなくなりましたが、引き続き電話側とのデータ同期が必要な場合などはこれらのAPIを使用できます。

Wear側から直接ネットワークが利用できるようになったため、できることが増えています。その1つがFCM[注3]を利用できる点です。

1.x系の場合は、電話側で端末識別用のトークンを作成し、電話にプッシュ通知を発行していました。その通知をWearに同期する形でWearへの通知を実装できましたが、このような複雑な実装は2.0では必要ありません。これからはWear単体でプッシュ通知を受け取ることが可能となります[注4](**図3**)。

③Watch Face Complications APIが追加された

Watch Face(時計盤の待ち受けに当たります。**図4**)にも大きなAPIの追加があります。

1.xのWatch Faceは、Watch Faceアプリの中に、スタイル、データの表示、データの取得、データタップ時の動作などすべてを詰め込んで実装する必要がありました(データとは天気の情報や、メールの未読件数などです)。この仕様は、Watch Faceを作ろうとすると前述のすべての項目を作成する必要があり、開発者にとってはハードルの高いものでした。またユーザとしては、気に入った見た目のWatch Faceでも期待するデータ表示がない場合には、別のWatch Faceを探す必要があるという、けっして利便性の良いものではありませんでした。

2.0で追加されたWatch Face Complications APIによって、Watch Faceアプリとプロバイダアプリ(データ提供側)に分離して開発し、組み合わせて使えるようになったことで、これらの問題が解消されます(**図5**)。

プロバイダ側では大きく分けて次の処理を実装します。

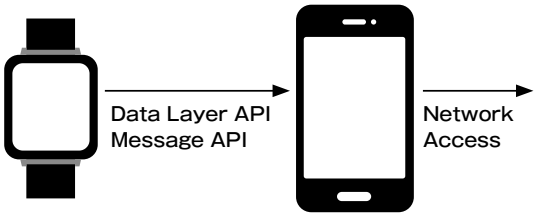
①Watch Faceへどの情報を提供するのか(TEXTなのか数字なのか)
②データの更新時間はどうするのか
③更新時間が来たときの処理

以上の実装を行ったWearアプリをWear 2.0のエミュレータもしくは端末にインストールしてみてください。すると**図6**のようにWatch FaceからWearSampleアプリのデータプロバイダが選択できるようになり、選択後は**図7**のように値が表示されるはずです。

固定の情報を表示させるプロバイダをサンプルで簡単に作りましたので、興味のある方は筆

注3) Firebase Cloud Messaging。サーバから端末へプッシュ通知を送るしくみです。
注4) Google Cloud Messagingについてはサポートされませんので注意してください。

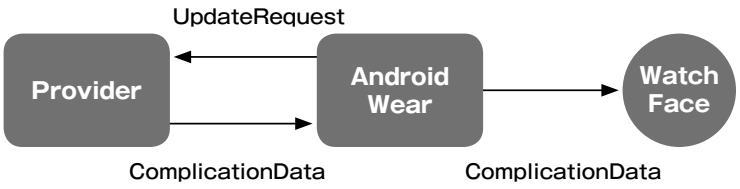
▼図2　一度電話を経由して外部へアクセス(1.xの場合)



**▼図3
エミュレータ環境でも問題なくFCM通知を受け取れた**

▼図4
Android Wear 2.0の
Watch Face

▼図5 Android Wear 2.0のComplicationDataの受け渡し



▼図6
データの選択アプリの中に、作成したプロバイダアプリが表示される

▼図7
選択をすると、ComplicationDataの情報(サンプルでは短文の部分)が表示される

者のGitHub[注5]から使ってみてください。実際のケースになると、たとえば天気予報だった場合は取得したい都道府県を設定する画面が必要でしょう。自社サービスの情報を提供するプロバイダを作成したい場合は、IDの入力などが必要かもしれません。そのため、プロバイダサービスと設定画面のActivityを含めて、プロバイダ用のapkを作成する必要があります。また、ComplicationDataにはタップしたときに実行するIntentを設定することもできます。データをタップしたら、設定画面を開くなどの動作もプロバイダアプリ側の責務となります。

このプロバイダを実装したapkをPlay Storeに提供することで、ユーザは自分の使いたいWatch Faceアプリをダウンロードし、欲しい情報のプロバイダアプリをダウンロードして組み合わせることが可能となります。サービスプロバイダは、Watch Faceを作成することなく自社サービスにWearからの導線を張ることができるようになります。従来Watch Faceのスタイルを作成していた開発者は、スタイルとデータの表示枠を実装するだけで、表示内容や動作は気にせずリリースすることができます。

まとめ

ざっと、Android Wear 2.0においての主要な変更機能を紹介しました。

電話側も作らないといけないと諦めていた開発者の方、また情報を提供したいと考えていたがWatch Faceを作るコストが必要だったサービスプロバイダ。これらの開発者の人たちは、ぜひ新しいAndroid Wear 2.0のSDKを触ってみてください。今まで以上にWear用のアプリが作りやすくなっています。

本件で紹介したいくつかの機能はあくまで筆者が抜粋した機能です。そのほかにもたくさんの機能やAPIが追加されています。この新しいAPIを利用することで、開発者はAndroid Wearのアプリを作りやすくなり、ユーザは、今まで以上にさまざまなアプリを選択できる余地が増えることを期待しています。その結果、Android Wearを搭載した時計を身に着けている人が、今まで以上に増えることを楽しみにしています。SD

注5) このサンプルプロジェクトは、筆者のGitHubで公開しています。https://github.com/katsuki-nakatani/WearProviderSample

書いて覚える Swift 入門

第26回 等しさと文字列と型の強さと

Author 小飼弾（こがい だん）　twitter @dankogai

APFSの秘密を解く

前回予告で今回は「総称型に焦点を当てる」と宣言しましたが、その前にやっておくべきことがありました。Swiftにおける文字の扱いです。なぜ今なのか。こちらをご覧ください（**図1**）。

なぜこうなってしまうのか。その秘密はファイル名にあります。

Perlで書かれた**リスト1**の検証コードでHFS+の空ディレクトリを指定して実行すると、こうなります。

```
食べないでくださーい

食べないでくださーい
食べないよー

かばん.txt:\x{304b}\x{306f}\x{3099}\x{3093}.txt
```

ところがAPFSだとこうなるのです。

```
/Volumes/apfs/utf8test/test.d/かばん.txt:No such file or directory at test.pl line 31.

食べないでくださーい

かばん.txt:\x{304b}\x{306f}\x{3099}\x{3093}.txt
かばん.txt:\x{304b}\x{3070}\x{3093}.txt
```

いったい何が起きているのでしょう？

HFS+では\x{304b}\x{3070}\x{3093}.txtと \x{304b}\x{306f}\x{3099}\x{3093}.txtは同じファイル名として扱われていますが、APFSでは異なるファイル名として扱われているのです。

そしてSwiftも、\u{304b}\u{3070}\u{3093}と\u{304b}\u{306f}\u{3099}\u{3093}を等しいとみなしているのです。

```
1> "\u{304b}\u{3070}\u{3093}" == "\u{304b}\u{306f}\u{3099}\u{3093}"
$R0: Bool = true
```

文字列が単なるバイト列だった前世紀には驚きの結果ですが、Unicodeが普及した今世紀にはむしろUnicode正規化[注1]を前提に比較するというのはUnicodeの理念からすると実は正しい。しかしUnicode的に「正しく」==を実装しているのは、筆者の知る限り現時点においてSwiftぐらいのものでしょう。

Swiftの Unicode原理主義ぶりは、==にとどまりません。先のPerlスクリプトをFoundation

▼図1　APFS-filename（https://twitter.com/Tats_y/status/846899701425201152）

注1）URL https://ja.wikipedia.org/wiki/Unicode正規化

を使って移植した検証コード(**リスト2**)では、APFSでもHFS+と同じ結果が出るのです。

ただしそうなるのはmacOSの場合。Linux上ではPerlと同じように振る舞います。

てんでんばらばらちんぷんかんぷんまとまらない?

なんともややこしそうですが、ベストプラクティスはすでに存在します。NFC(Normalization Form Canonical Compression)にしてしまうのです。

- IMなどで文字列入力した場合、ほぼ100%NFCになる。つまりソースコード中のUnicode文字列は当初からNFC
- NFCのほうがバイト数が少ない
- HFS+は双方受け付けるので、あえてNFDにする必要はない

SwiftでStringをNFCにするには、import Foundationしてから.precomposedStringWithCanonicalMappingにアクセスするだけです。Swift 3.1ならLinuxもサポートしています。こんな長いの覚えられないというのであれば、次のようにしても良いでしょう。

```
import Foundation
extension String {
    var nfc:String {
        return self.precomposedStringWith
CanonicalMapping
    }
    var nfd:String {
        return self.decomposedStringWith
CanonicalMapping
    }
    var nfkc:String {
        return self.precomposedStringWith
CompatibilityMapping
    }
    var nfkd:String {
        return self.decomposedStringWith
CompatibilityMapping
    }
}
```

▼リスト1　Perlによる検証コード

```
#!/usr/bin/env perl
use strict;
use warnings;
use Encode;
use feature 'say';
use utf8;
binmode STDOUT, ':utf8';

sub listdir {
    my $dn = shift;
    opendir my $dh, $dn or die "$dn: $!";
    my @fn = map { decode_utf8($_) }
      grep !/\A\./, readdir $dh;
    closedir $dh;
    @fn;
}

my $dir    = shift or die "usage: $0 dirname";
my $fn_nfc = "\x{304b}\x{3070}\x{3093}.txt";
 # かばん
my $fn_nfd = "\x{304b}\x{306f}\x{3099}\x{3093}
.txt";
eval {
    my $path = "$dir/$fn_nfc";
    open my $fh, '>:utf8', $path
      or die "$path:$!";
    say $fh "食べないでくださーい";
    close $fh;
};
say $@ if $@;
eval {
    my $path = "$dir/$fn_nfd";
    open my $fh, '<:utf8', $path
      or die "$path:$!";
    say <$fh>;
    close $fh;
};
say $@ if $@;
eval {
    my $path = "$dir/$fn_nfd";
    open my $fh, '>>:utf8', $path
      or die "$path:$!";
    say $fh "食べないよー";
    close $fh;
};
say $@ if $@;
eval {
    my $path = "$dir/$fn_nfc";
    open my $fh, '<:utf8', $path
      or die "$path:$!";
    say <$fh>;
    close $fh;
};
say $@ if $@;
for (listdir($dir)) {
    say "$_:", encode 'ascii', $_, Encode::FB
_PERLQQ;
}
unlink "$dir/$fn_nfc","$dir/$fn_nfd"
```

▼リスト2　Swiftによる検証コード

```
#!/usr/bin/env swift
// utility functions and methods
#if os(Linux)
    import Glibc
#else
    import Darwin
#endif
import Foundation
func listDir(_ path:String)->[String] {
    let fm = FileManager.default
    do {
        let items = try fm
            .contentsOfDirectory(atPath:path)
        return items
    } catch let e {
        fatalError("\(e)")
    }
}
func slurpFile(_ path:String) throws ->String {
    do {
        let str = try String(
            contentsOfFile:path,
            encoding:.utf8
        )
        return str
    } catch let e {
        throw e
    }
}
extension String {
    func append(path: String) throws {
        let fm = FileManager.default
        if fm.isWritableFile(atPath:path) {
            do {
                let url = URL(fileURLWith➥
Path:path)
                let fh = try FileHandle(
                    forWritingTo: url
                )
                _ = fh.seekToEndOfFile()
                _ = fh.write(self.data(using:➥
.utf8)!)
            } catch let e {
                throw e
            }
        } else {
            let created = fm.createFile(
                atPath:path,
                contents:self.data(using:.➥
utf8)!,
                attributes:nil
            )
            if !created {
                throw NSError(
                    domain:
                    "failed to create \"\➥
(path)\"",
                    code:500
                )
            }
        }
    }
}
// main part
let args = CommandLine.arguments
if args.count < 2 {
    fatalError("\(args[0]) directory")
}
let dir = args[1]
let fn_nfc = "\u{304b}\u{3070}\u{3093}.txt" ➥
// かばん
let fn_nfd = "\u{304b}\u{306f}\u{3099}\u{3093}➥
.txt"
do {
    let path = "\(dir)/\(fn_nfc)"
    try "食べないでくださーい\n".append(path:➥
path)
} catch let e {
    print("\(#line):\(e)")
}
do {
    let path = "\(dir)/\(fn_nfd)"
    try print(slurpFile(path))
} catch let e {
    print("\(#line):\(e)")
}
do {
    let path = "\(dir)/\(fn_nfd)"
    try "食べないよー\n".append(path:path)
} catch let e {
    print("\(#line):\(e)")
}
do {
    let path = "\(dir)/\(fn_nfc)"
    try print(slurpFile(path))
} catch let e {
    print("\(#line):\(e)")
}
for item in listDir(dir) {
    print("\(item):", Array(item.unicode➥
Scalars))
}
let fm = FileManager.default
for item in [fn_nfc, fn_nfd] {
    let path = "\(dir)/\(item)"
    if fm.fileExists(atPath:path) {
        try fm.removeItem(atPath:path)
    }
}
```

これで"\u{304b}\u{306f}\u{3099}\u{3093}".nfcとすればかばんは3文字で収まります。

もうひとつ念のために、**同等**な比較に加えて**同値**な比較も用意しておくのもよいでしょう。たとえば===を次のように定義しておくのです。

```
func ===(_ lhs:String, _ rhs:String) -> Bool {
return Array(lhs.utf8) == Array(rhs.utf8)
}
```

そうすると、次のようになります。

```
let ue9 = "\u{E9}"            //
let u65u301 = "\u{65}\u{301}" // é + ´

ue9 == u65u301      // true
ue9 === u65u301     // false
ue9.nfd === u65u301 // true
ue9 === u64u301.nfc // true
```

Perlに慣れた筆者から見ると、Swiftの文字列処理はまだまだひよっこもいいところなのですが、はじめからUnicodeをきちんと扱っているのはとてもうらやましい。Our goal is to be better at string processing than Perl[注2]という公約が実現するかどうかはさておき、PerlやPHPやPythonやRubyはUnicodeが後付けなおかげで今でも苦労しているのですから。

次号予告

ところで==は、どんな型にも必ず存在するわけではありません。

```
let a:Any = 1
let b:Any = 1
```

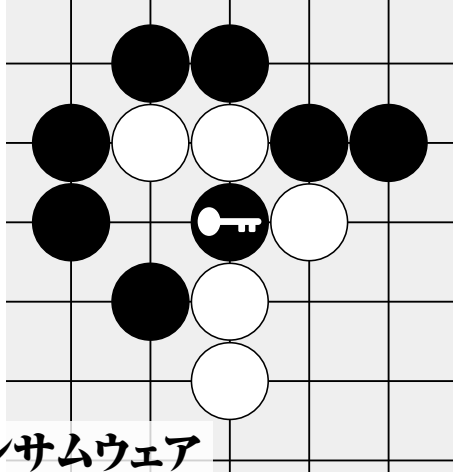
ここでa == bとしても、error: binary operator '==' cannot be applied to two 'Any' operandsと怒られてしまいます。Equatableプロトコルに準拠している型だけが等しさを享受できるのです。

プロトコル？　準拠？　次回はいよいよSwift最大の特長であるプロトコルを、総称型と絡めつつ紹介します。SD

注2）URL https://lists.swift.org/pipermail/swift-evolution/Week-of-Mon-20160725/025676.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

【第四四回】身代金被害だけで済まないランサムウェア

ランサムウェアとは、コンピュータシステムに感染／侵入／伝搬（でんぱん）し、ファイルを暗号化して開けなくするなどの手段で情報資源を人質とし、身代金を要求するマルウェアです。ランサムウェアのことはすでに本連載第24回で解説しましたが、今回は最近のランサムウェアが引き起こしたトラブルと、そこから見えてくる一歩踏み込んだ問題について考えます。

ボットネットとランサムウェア

今日のランサムウェアは、マルウェアとして単独で動作するのではなく、ネットワークを介してサーバと通信を行うシステム、つまりボットネットと同じ構造で動作しています。これはコンピュータ上の情報を人質にし、身代金を要求する性質上、ランサムウェアの被害先を管理する必要があるのと、さらには単機能の能力では達成できないランサムウェアの複雑さがあるからです。

ランサムウェアは、近年の高度化したマルウェアがそうであるように、最初にコンピュータの中に入り込むモジュールはC&C（Command and Control）サーバから必要なモジュールをダウンロードする役目を持っています。

最初にコンピュータに入り込む方法は、PDFやFlashの脆弱性を狙い任意のコマンドを実行できるように細工をしたファイルをネットワーク上のサーバに用意しておくケース、ネットワークのサービスの脆弱性をついて外部から任意のコマンドをリモート実行するようなケース、標的型攻撃のようにメールに添付している実行ファイルをユーザの錯誤により実行させるケースなどさまざまです。まずは外部からイニシャライズするためのモジュール（マルウェアのインストーラ）をダウンロードするだけの単機能で十分役目を果たします。

脆弱性は次々と現れ、またセキュリティアップデートで消えていきます。そのため、侵入の役割を持つモジュールは、どんどんアップデートされていきます。もぐら叩きといった状況ですが、外部のサーバからイニシャライズのためのモジュールのみを取り込むだけの単機能に絞ることで、新規部分の開発量が小さくなり、次々に発見される脆弱性に対応する開発速度も上がるという利点があります。現在のマルウェアは非常にモジュラリティが高くなっており、皮肉なことにソフトウェア工学で習う、ソフトウェアのモジュラリティと可用性の関係性をうまく使っています。

C&Cサーバと通信を行って必要なモジュールをダウンロードして展開し、インストールしてしまえば、そこで初期の段階は終了です。利用するボットネットは、必ずしもランサムウェア独自のインフラを持つ必要はありません。2013年に登場したCryptoLockerは、Gemeover ZeuSのボットネットに相乗りしていました。

次に本格的にマルウェアが動作する段階に入ります。まず侵入したコンピュータがどんな使われ方をしているのかを調べ、コンピュータが接続されているLANを探索し、より感染を広げるような戦略をたてます。これはC&Cサーバとやりとりするマルウェア一般に言えることです。なるべく感染を広げるため、LAN内部に接続されているコンピュータに脆弱性があるかスキャンをかけます。またファイ

ルサーバなどに接続していないかを調べます。

ランサムウェアは、かたっぱしからファイルを暗号化して開けなくするわけではありません。WordやExcelのファイルやデータベースで使っているファイルを狙うなど、効果的に被害を与えようとします。最初に侵入したコンピュータがあまり利用されていなくても、侵入先の組織内で価値のあるものを探します。そして、1つのコンピュータだけではなく、なるべく多くのコンピュータに被害を与えようとします。情報の人質を取って身代金をゆすり取るのが目的であるため、深刻な状況になればなるほどお金を得られる可能性が高くなるからです。

相手の弱みにつけこんで最大限の譲歩を引き出すのがランサムウェアの目的です。相手からどれだけ取れるかしか考えていないと言ってもいいでしょう。ですが、金額はとにかく大きければいいというものではありません。家庭で使っているPCの身代金として10万ドルを要求しても、非現実的な金額ですから無視されるはずです。せいぜい300～500ドルなどの現実的な金額を要求することでしょう。

一方で、企業の業務が滞るような場合や大きな損害になるような場合は、10万ドルでも30万ドルでも高額な金額をふっかけることでしょう。犯人側にとってはその金額が手に入ればラッキー、入らなければ次のターゲットを探せばいいというくらいで、さしたる損はありません。

このように、被害者の組織規模に合わせた身代金を選択しなければ効率は悪いはずです。

身代金の相場

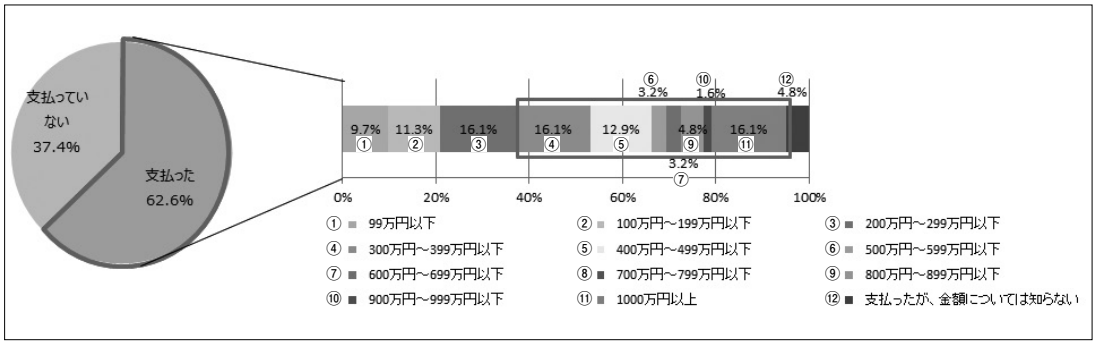
2016年にトレンドマイクロ社が公開した調査に「企業がランサムウェアにいくら払ったか」という興味深いデータがあります(**図1**)。それを見ると6割の被害企業が身代金を払っています。さすがにこの数字には驚きました。筆者は10～20％ぐらいと思っていたからです。さらに、1,000万円以上払っている被害者が約16％もいます。このデータを見ると、ランサムウェアは本当にお金になっていることがわかります。

図1のように被害金額が広く分布しているのは、ランサムウェアのターゲットを選んでいる時点で金額はある程度は考えているのでしょうが、攻撃したあとに攻撃規模や相手の被害を見ながら被害者が妥協する金額を決定しているのではないかと筆者は考えています。収益を最大化する(悪い意味で)努力をしていると考えるのは、そんなにおかしなことではありません。

海外でのケーススタディ

最近は日本でも海外(英語版)のランサムウェアに巻き込まれるケースを聞くようになりました。しかし、特定の国内企業を狙って(日本語で)攻撃をしかけてくるようなあからさまなケースは聞かれません。ただ、本誌がみなさんの手元に届くころに新聞

◆**図1　身代金を払った人の割合と、払った金額の割合**[注1]



注1) 出典：トレンドマイクロ社「企業におけるランサムウェア実態調査 2016」（※図中の丸数字は編集部にて補足した）
http://www.trendmicro.co.jp/jp/about-us/press-releases/articles/20160727064652.html

沙汰になっていても驚きはしません。次に、海外でのランサムウェアのインシデントで非常に興味深いものをいくつかピックアップしたいと思います。

サンフランシスコ市営交通局

2016年11月25日（金）、サンフランシスコ市営交通局の事務系スタッフのメールアドレスに対して網羅的に、ランサムウェアが添付されたメールが送りつけられました[注2]。ターゲットとなったのは事務で使われている900台のPCです。攻撃者はどこかで交通局の事務系スタッフのメールアドレスの一覧を入手したのでしょう。

交通局は対応のためにメールシステムをシャットダウンさせ、内部ネットワークにつながっているPCなどの感染被害を調査し、感染防止の処置を行いました。市内の地下鉄や近郊をつなぐコミュニティ電車などの交通運輸システムはインターネット側の影響を受ける構造にはなっていないようで、システムは安定して稼動していました。実際にランサムウェアの影響を受けたのは、運行には関係ない労務システムなどごく一部でした。

しかし、市営交通局は2016年11月25日から翌日いっぱいまで市内の地下鉄および路面電車（SF Muni）で改札を開放する処置を取りました。感染範囲だけを考えると小さいですが、900台すべてのPCやメールシステムを調査／対処するには、それなりの時間が必要です。その間、実質的に事務作業は麻痺（まひ）しています。そのときに外部から販売トラブルなどがあっても事務系スタッフは対処できません。ゲートをオープンにして乗車カードなどは必要ないようにして対応したとのことです。11月27日（日）からは正常運行に戻ったので、対処の規模を考えると、かかった時間は妥当なところではないでしょうか。

サンフランシスコは観光の街なので、週末に市内交通が麻痺すると多大な経済的二次被害が発生する可能性があります。それゆえかランサムウェアは100ビットコイン（約7万3,000ドル相当／約832万円）を要求したそうです。しかし、アメリカの政府当局はテロリストの要求を表面上は無視するポリシーを貫いていますので、その伝統（？）で当局は犯人に身代金を払うようなことはしませんでした。その後の発表では、本来受けとっていたはずの運賃などの被害総額は5万ドル（約570万円）とのことでした。直接的な被害総額として多いか少ないかの判断は難しいところです。もし、まごついていれば市内鉄道交通が麻痺するわけですから、そこから波及する被害は5万ドルで収まるとは到底思えません。

当局の広報は「今回迅速に対処することができたのは、サイバー攻撃に対して対応シナリオを用意し準備していたから」と答えています。

身代金を支払ったリゾートホテル

2017年1月、オーストリアのウィンタースポーツのリゾート地として有名な地域にある老舗高級ホテルRomantik Seehotel Jaegerwirtの業務で使っている複数のコンピュータがランサムウェアに感染しました[注3]。

冬のシーズン真っ盛りの稼ぎどきの出来事で、予約システムや現金管理などのコンピュータが使えないため混乱しました。とはいえ、そのあたりは人海戦術でどうにかなったようです。致命的だったのは、カードキーを管理するコンピュータが使えず、新たにカードキーを発行できなくなって新しい宿泊客を部屋に案内できなかったことでした（すでに発行しているカードキーはそのまま使えたようです）。

どうしようもないので、犯人に身代金1,500ユーロ（約18万円）を支払い、システムを復旧させたとのことです。ニュースの最後には、今後はカードキーはやめて、機械式のキーに戻すとありました。

このとき、感染したランサムウェアはSatanという名前で、フィッシングや悪意のあるリンクを経由して感染を広げ、感染後はRSA-2048/AES256で

注2）"Hacked Muni refused $73,000 ransom demand; computers restored"
http://www.sfgate.com/bayarea/article/Hacked-Muni-refused-73-000-ransom-demand-10641070.php

注3）"Hotel ransomed by hackers as guests locked out of rooms"
http://www.thelocal.at/20170128/hotel-ransomed-by-hackers-as-guests-locked-in-rooms

ファイルを暗号化するタイプのものでした。暗号化したあと、ユーザに匿名化通信用のTorブラウザをインストールさせ、そのTorを使い.onionドメイン空間（つまりダークネットです）にあるSatanの支払いページに移動させてビットコインで支払いをさせます。そうすると復号のための鍵が手に入ります。身代金を払ったのに復号鍵が手に入らないというケースが多発すると、今後は誰も身代金を払わなくなります。そのため、身代金ビジネスを長い目で見たうえで、きちんと復帰できるような手はずになっているようです。このホテルのコンピュータも元に戻ったとの結末がニュースに書かれていました。

Satanとは、ランサムウェアそのものと、支払うためのダークネット上のサーバなどのインフラのすべてが提供されるRaaS（Ransomware as a Service）です。このときの支払い金額はRaaSユーザ（ランサムウェアをしかける犯人）が決めます。RaaSを提供する側は20～30％の手数料を取り、犯人側は70～80％を受け取ります。しかける犯人は技術的なことは心配せず、ターゲットを選別し、どの程度の金額を要求するかだけに集中できます。攻撃手法は新たな段階に入ったと言えるでしょう。

Loss of View / Loss of Control

LoV（Loss of View）、LoC（Loss of Control）とは、ICS（Industrial Control System）分野のセキュリティで使われる用語です。最後のモニター部分やコントロール部分が麻痺（ロス）することで全体が麻痺してしまうケースなどに、この用語が使われます。

ICSは工場など特殊な環境で動くコンピュータです。ネットワーク的に隔絶されており、世間一般のPCが感染するようなマルウェアはそうそう入り込まない、とされています。まずそこは否定せずに、そうだとしましょう。

一方で、その状況をモニタリングする端末に普通のPCが使われることが、最近では増えてきました。ブラウザを使ってWebベースで、監視画面を見たり、コントロールができたりと、便利な機能を持つものも少なくありません。

しかし、そのとき最も弱いのはそのモニタリング用のPCです。世間一般のマルウェアでも十分に麻痺します。PCがマルウェアに感染し麻痺したところで、工場側のネットワークには（マルウェアの感染という意味で）まったく影響がなかったとしましょう。それでもPC側は、感染した可能性のある機器をすべて洗い出し、信頼できない機器はすべてクリーンインストール、あるいは排除し、新しい機器を入れたり、関連するネットワーク上にマルウェアがいないかなどの検査をしたりしなければいけません。というのも、1台のPCが故障したときのような対処法とは違い、マルウェアの性質から、ネットワークにつながれたすべてのPCが感染している可能性も考えて対処しなくてはいけないからです。

さすがにこれは数分で終わるようなものではありません。稼働している工場ラインをモニタリングするPCが使えなくなり、1時間も2時間も状況を監視できないならば、いったん工場ラインを止めなければいけなくなります。かくして、たった数台のPCの問題であったはずが、工場全体、プラント全体の問題に拡大してしまいます。

サンフランシスコ市営交通局のケースや、リゾートホテルのケースはLoV/LoCの形を変えたものだと言えるでしょう。ランサムウェアは、相手に最大限の被害を与え、金銭をゆする悪意のある攻撃です。多くの人は「書類がダメになってしまう」くらいの認識かもしれません。しかし、ランサムウェアは工場全体、会社全体、システム全体を止めてしまうポテンシャルのある攻撃であり、実際に今回紹介したようなケースが起こっています。

ランサムウェアの対策は通常のマルウェア対策と基本的に同じです。しかし、その対策で被害にあう可能性が小さくなったとしても、完全に安全というわけではありません。最悪のケースが起きた場合に、サンフランシスコ市営交通局のようにパフォーマンスを落としても事業継続させるか、あるいは、安全な状態に移行して停止させるといったプランを考えておかなければいけません。最悪ケースを想定したシナリオを作っておくことをお勧めします。SD

SOURCES

レッドハット系ソフトウェア最新解説

第9回

IT資産可視化の第一歩

企業のシステムに関するサブスクリプション情報や既知の問題などを、自動的に検出するRed Hat系のサービスを紹介します。

Author 小島 啓史(こじまひろふみ)
mail：hkojima@redhat.com
レッドハット㈱ テクニカルセールス本部 ソリューションアーキテクト

Red Hatの可視化ツール

ハードウェア、ソフトウェア、サブスクリプション(またはライセンス)といったIT資産の適切な管理については、セキュリティやコンプライアンスなどの観点からかなり重要視されていることが多いです。こうした管理業務を効率化するためには、ハードウェア情報、利用者情報、ソフトウェア構成、サポート契約期間などを可視化しておく必要があります。そのような可視化のための製品をさまざまな企業が提供していますが、当社でもRed Hat系のソフトウェア資産を可視化するための製品をいくつか提供しています。本記事ではそれらの製品のうち、オンプレミスのサブスクリプション管理ツールであるSAM(Red Hat Subscription Asset Manager)と、システム分析サービスであるRed Hat Insightsを紹介します。

Red Hat Subscription Asset Manager(SAM)

SAMはサブスクリプション情報やソフトウェア更新を管理するためのアプリケーションです。SAMを利用することで、外部接続ができないRHEL(Red Hat Enterprise Linux)システムに対するソフトウェア更新や、企業が持つサブスクリプション情報を一元管理できるようになります。このときの利用イメージは**図1**のようになります。

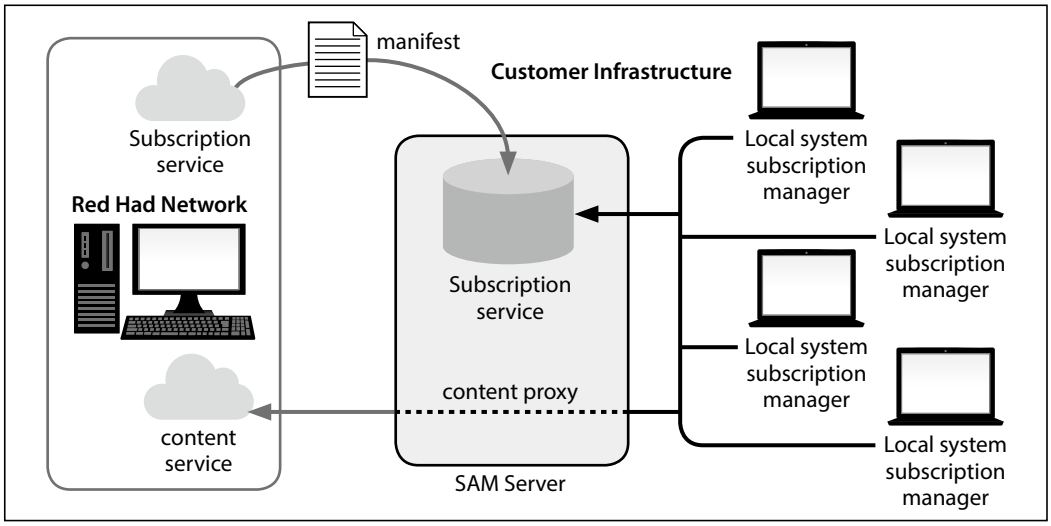
図1では、SAMに一定数のサブスクリプション(図中ではmanifestと記載)を付与してローカルのRHELシステムと紐付けているほか、SAMをコンテンツプロキシとして利用することで、SAM経由でのRHELシステムのソフトウェア更新を可能にしていることを表しています。以降では、こうした環境のセットアップ手順の概略を紹介していきます。

SAMのセットアップ手順概略

最初に、当社のカスタマーポータル[注1]にあるサブスクリプション管理サービス[注2](Red Hat Subscription Manager)で、SAMにサブスクリプション情報を付与するためのマニフェスト(サブスクリプションを利用するための証明書や、数量／種類が記載されたファイルなどが入っているパッケージ)を作成する必要があります。カスタマーポータルでのGUIによる操作を行いますが、作成手順の詳細はこちら[注3]をご参照くだ

注1) URL https://access.redhat.com/
注2) URL https://access.redhat.com/management/
注3) URL http://red.ht/2mMfE33

▼図1　SAMの利用イメージ



さい。作成したマニフェストのダウンロード画面イメージは**図2**のようになります。

▼図2　マニフェストのダウンロード画面イメージ

次にSAMをRHELシステムにインストールします。SAMはRHELのサブスクリプションを購入していれば、無料で利用できます。ただし、執筆時点ではSAMはRHEL6.6以降のシステムにインストールする必要があり、RHEL7系のシステムにはインストールできませんので注意してください。

あらかじめ用意した1台のRHEL6システムをサブスクリプション管理サービスに登録し、SAMをインストールするために必要なリポジトリだけを有効化します。

```
# subscription-manager register --auto-attach
# subscription-manager repos --disable="*"
# subscription-manager repos --enable=rhel-6-server-rpms --enable=rhel-6-server-sam-rpms
```

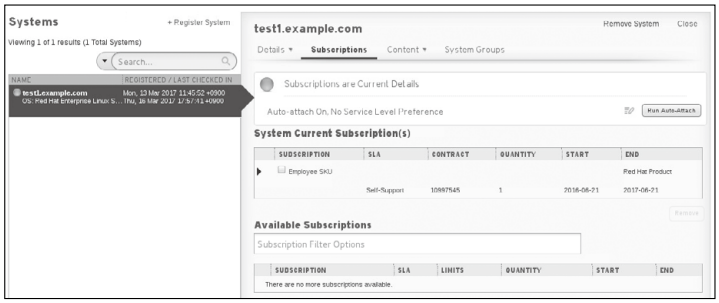
そしてSAMを構成するためのパッケージ群をインストールし、SAMのセットアップコマンド(katello-configure)を実行します。

```
# yum -y install katello-headpin-all
# katello-configure --deployment=sam
```

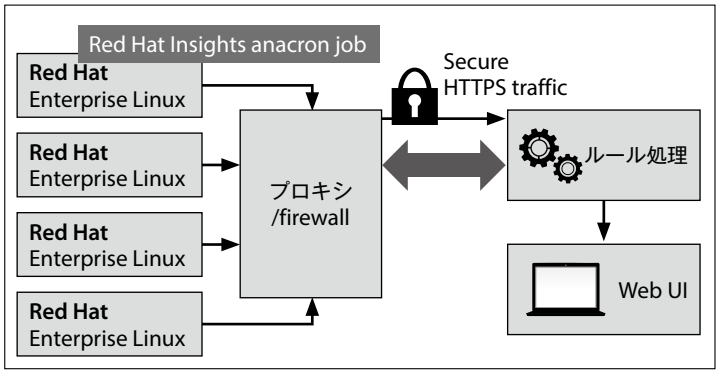
これで、Webブラウザ(サポート対象のWebブラウザはFirefoxとInternet Explorer)からhttps://SAM_FQDN/samにアクセスし、デフォルト設定のユーザとパスワード(ともにadmin)でSAMにログインできるようになります。

最後にダウンロードしておいたマニフェストをSAMにインポートします。インポート作業はSAMのGUIで実行できますが、katelloコマンドでも実行できます。

▼図3　RHELシステムの登録イメージ



▼図4　Red Hat Insightsの利用イメージ




```
# katello -u admin -p admin provider import_manifest --name="Red Hat" --org=ACME_Corporation --file=manifest.zip
```

このコマンドでローカルディレクトリにあるマニフェスト(manifest.zip)をSAMにインポートしています。-u/-pオプションで認証情報を指定し、--nameオプションでマニフェスト提供元の企業名(SAMを利用する場合は「Red Hat」だけ指定可能)、--orgオプションでデフォルトで作成される組織名(ACME_Corporation)を指定しています。

これでSAMの基本的なセットアップ作業は完了です。http://SAM_FQDN/pubからダウンロードできるRPMパッケージをRHELシステム(RHEL7系でもOKです)にインストールし、subscription-managerコマンドでSAMにシステムを登録します。登録時に指定する認証情報はSAMサーバのもの(デフォルトではadminユーザ)を利用してください。システムの登録とサブスクリプションの紐付けが無事完了すると、**図3**のような画面からシステムの情報(サブスクリプション情報やハードウェア情報など)を確認できるようになります。

ほかにも部署に対応する組織(Organization)の作成、各組織での異なるマニフェストの利用(各部署へのサブスクリプションの分配)、ローカルユーザの作成、OpenLDAP/Active Directoryとの連携、Activation KeyによるRHELシステム登録時の特定サブスクリプションの自動紐付けなどがSAMにより可能になります。これらの手順の詳細については、公式ドキュメント[注4]を参照ください。

Red Hat Insights

SAMではRHELシステムに紐付けられたサブスクリプション情報を可視化できますが、既知の脆弱性などの問題を検出できません。ただし、オンラインのシステム分析サービスであるRed Hat Insightsを利用することで、レッドハットのナレッジベースを基に既知の脆弱性／問題の検出や障害の事前予防策をまとめたレポート[注5]を自動的に作成できるようになります。SAMとは違ってRed Hat Insightsは有償のサービスですが、RHEL6以降で提供されるredhat-access-insightsパッケージをインストールしてRHELシステムを登録するだけ[注6]でシステム監視を開始できますので、導入ハードルはそれほど高く

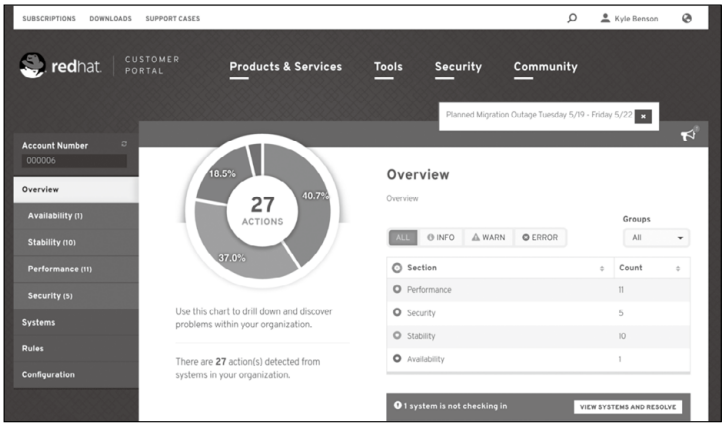
注4）URL http://red.ht/2mMx1Ro
注5）すべての脆弱性／問題を検出することはできませんので注意してください。
注6）URL https://access.redhat.com/products/red-hat-insights#getstarted

ないサービスとなっています。そのため、国内では東京海洋大学でRHEL環境の監視に利用される[注7]など、徐々に利用者が増えてきています。

Red Hat Insightsの利用イメージは**図4**のようになります。外部接続できないRHELシステムでも、プロキシを経由してRed Hat Insightsに登録できます。ただし、すでにSAMに登録しているRHELシステムは、Red Hat Insightsに登録できませんのでご注意ください。そのため、SAMを利用している場合は、SAMに登録していない同一構成のRHELシステムをRed Hat Insights用に別途用意するなど、一工夫必要になります。

Red Hat Insightsにより作成されるレポートは、`https://access.redhat.com/insights/`にアクセスすることで確認できます。このレポートは、Red Hat Insightsにanacronで定期的に送信[注8]される構成情報や動作サービス[注9]をもとに作成されます。また、セキュリティの観点で送信したくない情報[注10]を明示的に指定することもできます。レポートを見ることで、登録されているシステムに関する一部の(重要な)脆弱性／設定の問題を修正するための対応策や、関連するレッドハットのナレッジベースを確認できます(**図5**)。

▼図5　Red Hat Insightsで作成されるレポートのイメージ



▼図6　Red Hat InsightsによるPlaybook自動作成画面のイメージ

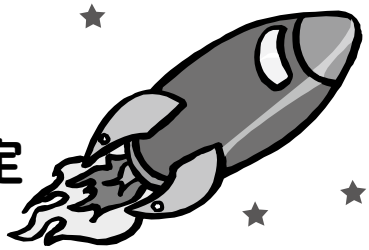
また、執筆時点ではベータ版のサービスとなりますが、**図6**のように対応策を実施するためのAnsible Playbookを自動的に作成することもできます。そのため、Red Hat InsightsのAPI[注11]によるレポート／Playbook作成機能を活用することで、システムの構成管理だけでなく、問題検出と修正の自動化をAnsibleで実現することも可能になります。Red Hat Insightsは評価版[注12]もありますので、興味がありましたらぜひ試してみてください。SD

注7) URL http://jp-redhat.com/openeye_online/case study/4852/
注8) デフォルトでは日次。週次設定も可能。
注9) URL https://access.redhat.com/articles/1598863
注10) URL https://access.redhat.com/articles/2025273
注11) URL https://access.redhat.com/r/insights/docs/
注12) URL https://access.redhat.com/insights/evaluation

47

Debian Developer　やまねひでき　henrich@debian.org

Debian 9開発は終盤へ、新プロジェクトリーダーも決定



Debian Hot Topics

Debian 9リリース進捗続報

開発がフリーズに入ってからすでに数ヵ月が経ち、Debian 9“Stretch”のリリース作業の進捗にやきもきしている人も多くいることでしょう。これに対し、4月16日にリリースチームから進捗報告がありました[注1]。

- キーとなるパッケージのRCバグ数は2月頭の150個から40個へ減らした
- OpenSSL transition[注2]とMySQLからMariaDBへのtransitionが完了。gcc-5も削除
- 「stretch-backports」[注3]を追加完了
- セキュアブートのサポート[注4]はshimブートローダーにMicrosoftから署名を得たが、リポジトリサーバの管理ソフトウェア「DAK（Debian Archive Kit）」でのサポート待ち（Bug#821051参照）
- パッケージのビルドサーバはarmelとarmhfアーキテクチャを除いてStretchにアップグレード済み。armelとarmhfはブートしないため調査中
- Debian 8“Jessie”からのアップグレード時のバグで未解決のものがまだ残っている
- binNMU[注5]については「Built-Using」ヘッダを使うように更新。まだすべて実施完了していない

この報告を見る限り終わりが見えてきたので、RCバグを0にしてリリースになる日も近そうです……とは言っても「リリースの障害と見なされるリリースクリティカルバグ（RCバグ）を0にする」というのは、残念ながら**RCバグをすべて修正することではない**のでご注意ください。修正されないRCバグというのもあります。リリース作業の最後の段階になると、

- stretch-ignoredなどのタグを付けて「今回は無視しても良いことにする」という形にして、修正の優先度を落とす
- 最後はバグのseverity（重要度）を落としてRCとは見なさないようにする（そして、リリース後に再度severityを上げる）

などの力技が繰り出されます（そのため、RC数カウントのグラフではリリース直前のRCバグの減り方がとても急激で、リリース後に急激に伸びます）。

どのようなRCバグがこのような対応を受けるかと言うと、「ライセンス変更をUpstream

注1）URL http://deb.li/i2kQh
注2）OpenSSL 1.1への移行作業ですが、あらゆるパッケージが1.1を使うようになったわけではなく、一部は1.0.2との併存となったようです。
注3）Stretchに対して、より新しいバージョンのパッケージを、リリース後にベストエフォートで提供するリポジトリ。testingにあるソースをStretch向けにビルドしなおしたパッケージが入れられます。
注4）2017年1月号の本連載を参照のこと。
注5）binary Non-Maintainer Uploadの略。ソースをいっさい変えず、ビルド環境を更新してビルドしなおしたパッケージのアップロードを指します。たとえば、依存関係にあるライブラリが更新された場合や、あとからコンパイラのhardeningフラグ（PIE（position independent executables）など）をデフォルト有効にしたうえでパッケージに一律適用して強化を実施する場合などに使っています。

と交渉中でいい反応はもらっているのだが、Debian安定版リリースのタイミングでの修正は困難」などのケースが挙げられます。もちろん、重大なバグであれば、その後のポイントリリースで修正を「押し込む」こと[注6]も可能ですので、気になるバグを見つけたらSubscribe(購読)しておくことをお勧めします[注7]。

参考に、過去のリリーススケジュールを**表1**に載せておきます。ここからもDebian 9のリリースが近いことがわかるかと思います。

Debian Installer Stretch RC 3とDebian 8.8リリース

Debian Installer Stretch RC 3が4月10日にリリースされました[注8]。さすがにこの段階になると、ほとんどがバグ修正でサポート機材の追加も大きくは行われていません。新しいインストーラをお試しいただける方はサイト[注9]からどうぞ。選挙のスローガンっぽいですが、「あなたの報告が改善につながる」ので、ぜひお願いします。

それからDebian 8"Jessie"のポイントリリース[注10]、Debian 8.8が5月6日にリリースされました。今回もバグ修正と、それまでにリリースされたセキュリティ修正を含むもので特段大きな変更はありません。

owncloudパッケージがDebianから完全に削除

2016年7月号の本連載で取り上げたowncloudパッケージですが、セキュリティアップデートなどのメンテナンスも無理である、としてついにDebianから完全に削除されました[注11]。

先日、パッケージメンテナであったDavid Prevotさんが南太平洋の遠方からはるばる来日した際に、一緒にビールを呑む機会があったのでこの件について聞きました。「upstreamとはいろいろと厄介なことがあったけど、やりあった人たちはownCloudからフォークしたNext Cloud[注12]に移動したから、今のownCloudは前よりは一緒に作業しやすくなっているかもしれない。でも、自分はもう作業する気力はなくなってしまったけどね」とのことでした。残念ではありますが、upstreamの協力を得られないソフトウェアの場合、このような不幸な結末にならざるをえない場合もあります。

Debianプロジェクトリーダー選挙

Debianプロジェクトでは1年に1回、プロジェクトリーダーを公式Debian開発者の投票によって決定します。今回はChris Lambさんと現職のMehdi Dogguyさんの候補者2名によって争われ、この結果Chris Lambさんが当選しました。

2人の所信表明は、「http://deb.li/

注6) 安定版リリースチームとの交渉が必要になるので、筆者的にはどうしてもこういう表現をしたくなります(どんな重大な問題も、基本的に黙っていては修正が入るものではありません。がんばってなんとかする「誰か」がいるのです……)。

注7) Subscribeのやり方は簡単です。メールで、「<バグ番号>-subscribe@bugs.debian.org」宛にメールを送り、確認メールに返信するだけで、該当のバグ関連メールがすべて送られてきます。たとえば、上記のセキュアブートのアーカイブサーバでのサポートの問題はBug#821051ですので、「821051-subscribe@bugs.debian.org」宛にメールします。

注8) URL http://deb.li/mGVR

注9) URL https://www.debian.org/devel/debian-installer/

▼表1 これまでのリリース関連スケジュール

時期	イベント
2010年 8月 6日	"Squeeze"フリーズ開始
⋮	⋮
2011年 2月 6日	Debian 6.0"Squeeze"リリース(～約6ヵ月)
2012年 7月30日	"Wheezy"フリーズ開始
2013年 3月20日	RCバグ残り100個
2013年 5月 4日	Debian 7"Wheezy"リリース(～約10ヵ月)
2014年11月 5日	"Jessie"フリーズ開始
2015年 3月 8日	RCバグ残り55個
2015年 4月25日	Debian 8"Jessie"リリース(～約5ヵ月)
2016年11月 5日	"Stretch"フリーズ開始
2017年 4月15日	RCバグ残り40個
2017年 ?月 ?日	Debian 9"Stretch"リリース(これまでの傾向からすると5月か6月?)

注10) このようなポイントリリースはおおよそ3ヵ月に一度程度のペースでリリースされます。

注11) URL https://bugs.debian.org/822681

注12) URL https://nextcloud.com/

UddD」で参照できますが、ここではざっくりとChrisさんの所信表明を眺めてみましょう。

彼は現状のフリーソフトウェアの世界におけるDebianの役割の状況について、Debianではなくほかの派生ディストリビューションがその地位を奪っているなどのいくつか悲観的な問いかけを投げかけました。しかし、そのすぐ後に「そうは言うものの、Debianは**非常に**よく回っている」と続けます。

よく回っているとは言うものの、最初に述べたように問題があることは間違いありません。Chrisさんは、Debianの現状の問題点は大きく2つ「外界とのコミュニケーションと、外の世界に対する認識が欠けている」点から来ているものであると指摘し、次のような説明をしています。

- 自身のスタートアップ業界での経験から、どんなプロジェクトでも「魅力的に見えるかどうか」というのは根源的な問題だと思う。しかし、Debianプロジェクトのサイトはまるで魅力がない(Debianはフレッシュさに欠けるというイメージがあるが、これに凝縮されている)。最も基本的なマーケティングを(意図的に?)無視していることで、潜在的な未来のユーザと開発者が被害を被っている
- Debianに詳しくないユーザに対してのサポートも欠けている。サポートチャンネルには「どのディストリビューションが最も良いか」「インストールにはnon-freeなファームウェアが必要だ」という話題などが繰り返しあふれていて、適切な対応ができていない
- contribute(開発への協力)に興味を持つだろう外部の人間は「Debianはどのように動いているのか」「やっていい作業はどのように決まっているのか」を尋ねてくる。これらが明確ではないところが内部の人には魅力的な点である一方、外部の人には把握が困難でハードルが高くて障害が多いとネガティブにとらえられる
- 実際のところはユーザも派生ディストリビューションも増加の一途なのに、このようなギャップは悲劇的だ。DebianはReproducible Buildsなどの技術的・社会的活動のさまざまなイニシアティブを先導してきたが、これからは単に技術的な面だけではなく、自身のとらえられ方や印象の点を強化しようとするべきだろう。我々は「単なる高品質なパッケージリポジトリ」でありたくはないのだから

ここで内部と外部との「温度差」による問題を明確に指摘したところはさすがです。問題を克服するにあたって、これまでのプロジェクトリーダーは「透明性」「パッケージング以外の活動の推奨」などを強調してきましたが、Chrisさんは次の項目を挙げています。

- **より多くのミーティングを開く**

  自身のこれまでの経験からすると、イベントはソーシャル面の重要性だけではなく、非常に生産性が高くDebianプロジェクトにとって有用なものである[注13]。また、単に新しいイベントを推奨・補助するだけではなく、参加者数と参加する人のダイバーシティを高めることを目指す

- **参加するためのプロセスを改善する**

  現状のWebサイトは、新規ユーザにとって適切な情報が配置されていないのに、不必要に細かい説明があるなどアンバランスな状態なので、ユーザテストを行うなどしてボトルネックを探すつもり

- **Debian独自の「outreach」イニシアティブを作る**

  大成功しているGNOMEプロジェクトの「Outreachy」[注14]にならって、より幅広く潜

注13) 日本の企業のように「会議のための会議」が多いために生産性が落ちているのと比べ、基本オンラインでの散発的な非同期コミュニケーションが多いDebianプロジェクトの場合は、「直接Face to Faceでの作業は集中して議論ができるので非常に効果的だ」ということですね。

注14) URL https://www.gnome.org/outreachy/

在的なContributor層に訴えるのと同時にマーケティングを行う

- **Debianでの活動の障害を積極的に排除する**
  たとえば、Debianには十二分な資金があるのでこれを活用する。改善するのに機材のマシンパワーが必要な場合、これまではあまり積極的に資金を使うことはない文化だったが、これを変える。ソーシャル・属人的な問題にも積極的に関与していく

これまでになく前のめりな方針です。Chrisさんは水泳3.9km・バイク180.2km・マラソン42.195kmで構成されるアイアンマンレースをこなすタフガイでもあるので、問題解決に向けて精力的に活動をこなしてくれることでしょう。

ロシア当局によるDebianメンテナの逮捕と勾留

なかなか衝撃的なニュースが飛び込んできました。Debianプロジェクトは、ロシアのDebianメンテナDmitry Bogatovさんが当局に逮捕勾留されたため、一時的な処置として彼のGPG鍵を、悪用を防ぐためにDebianのキーリングから削除したことを発表しました[注15]。

現在のところ、逮捕の明確な理由は不明です。憶測ではDmitryさんが試験的にTor(匿名で通信するためのソフトウェア)の出口ノードを運用していて、そのノードのIPアドレスを使って当局が目をつけるような活動をした人がいたのではないか？という意見が出ています。一方、Torプロジェクトはこの件について「現在のところ、Web上で参照できる情報以上のものはない。Torプロジェクトはユーザ情報は収集していない」という声明を出しています[注16]。

この件は続報が出次第、お伝えしたいと思います。SD

注15) URL http://deb.li/Esf7　押収されたGPG鍵が当局に悪用される可能性もありますので、妥当な処置でしょう。
注16) URL http://deb.li/31DCz

Ubuntuの方針転換とWeb翻訳混入事件

Ubuntu Japanese Team　あわしろいくや

今回はUbuntuの大きな方向転換について解説します。
同時にUbuntu 17.04開発中に発覚した事件とその対処についてもお知らせします。

Ubuntuの方向転換

さようならConvergence

Ubuntuの創始者にしてUbuntuサポート企業Canonicalの創業者であるMark Shuttleworthは、現地時間4月5日（日本時間では4月6日早朝）に、これまでの方針を転換する発表を行いました[注1]。Ubuntuは2013年から、デスクトップやスマートデバイスで同じ操作感やユーザインターフェースを提供するConvergenceに取り組んできました。それは以前より開発が継続しているUnityを発展させ、Unity 8で結実することになっていました。しかし、そのConvergenceへの投資を取りやめ、経営資源をクラウドやIoTに集中させるというのが主な内容です。

注1） https://insights.ubuntu.com/?p=65030

図1　Ubuntu 11.04のUnity。このころは今ほど多機能ではなかった

Ubuntuはタブレット端末やスマートフォン向けのUbuntu Touchを終了し、デスクトップ版のUnity 8への移行も見送ります。Unity 8までのつなぎであったUnity 7も終息となります。UnityはUbuntu 11.04からデフォルトのデスクトップシェルになったので（**図1**）、長いこと主役の座にあり続けたことになります。

Ubuntu Touchは残念ながら成功を収めたとはいい難く、ユーザもあまり多くないので、衝撃はともかく影響はさほど大きくありません。しかしUbuntuがUnityをやめてしまったら、デスクトップ環境はどうなってしまうのかという心配が出てきますが、これはUbuntu 18.04 LTSのリリースまでにGNOMEに回帰するという発表がありました。

この発表はあくまで方針の発表であり、具体的なことについては不明な点が多いのですが、その後Mark ShuttleworthのGoogle+アカウント[注2]でいくつかのことが明らかになっています。

Ubuntuの今後

UbuntuにはUbuntu GNOMEというフレーバー（公式派生版）があり、両者がどのような関係になるのかですが、UbuntuはUbuntu GNOMEをベースにするということです。Ubuntu GNOMEのWebサイト[注3]には“Ubuntu GNOME（中略）is a mostly pure GNOME desktop experience built from the Ubuntu reposi

注2） https://plus.google.com/+MarkShuttleworthCanonical/posts/7LYubpaHUHH
注3） https://ubuntugnome.org/

tories."とあり、そのとおりUbuntu GNOMEのために特別なカスタマイズをしておらず、純粋なGNOMEデスクトップ体験を提供しています。UbuntuがGNOMEを採用することにより、純粋なGNOMEデスクトップ体験とは異なるものになるのではないかという危惧は、これで否定されたことになります。同時に、GNOME ShellにUnity 7のルック&フィールが構築されることもありません。

Unity 7はどうなるのかというと、デスクトップ環境をGNOMEに移行したあとはuniverseリポジトリで提供するつもりとのことです。ということは、Canonicalによるサポートがなくなるものの、継続して使用すること自体は可能になる見込みです。Unity 7は独自の機能も多く、愛用者が多いため(もちろんそれを嫌う人も多いのですが)、すぐに使えなくなるということはなさそうです。

UnityとMirの今後

Unity 8はディスプレイサーバであるMir[注4]とともに開発されており、Convergenceを実現するキーコンポーネントです。Mirの去就も注目されましたが、IoTデバイスで使用するために開発は継続するとのことです。Mirの開発が発表された際に、同じく新ディスプレイサーバであるWaylandとの対比が話題になったりしましたが、MirはIoT用として目立たないところで活用されていくことになるため、メジャーという点ではWaylandに軍配が上がったということであり、GNOMEに移行するUbuntuでもいずれはWaylandがデフォルトになることでしょう。

投資を取りやめる、とサラリと書いていますが、これには重要な意味があります。Canonicalが投資をやめても現在のソースコードはすべて公開されており、これを元にUnity 8のソースコードをフォークして独自にメンテナンスしていこうという動きがあります。今のところは2つのプロジェクトが別々に活動していますが、そのまま行くのかは今後の流れしだいのようです。そのうちの1つはYunit[注5]で、こちらはデスクトップ寄りにフォーカスするようです。Mirをメンテナンスするつもりはなく、Waylandへの移行を目論んでいるのは興味深いです。もう1つはUBports[注6]です。コミュニティとしては以前から活動しており、その名のとおりUbuntu Touchのポーティングをやっていたのですが、今回の動きを受けてフォークも決断したとのことです。Yunitとは違ってUnity 8の基本方針を維持する方針ではあるものの、やはりMirのメンテナンスは厳しいと認識しているようです。

一方、Unity 7はこのままひっそり姿を消していくものと思われます。Unity 7に慣れたユーザがスムーズにGNOME Shellに移行できるかは非常に疑問があるものの、前述のとおり純粋なGNOMEデスクトップ体験を提供するためにはUnity 7に似せることはできません。また長生きさせるためにはUnity 7をWaylandに対応させる必要があるものと思われますが、技術的にはかなり困難でしょうからこれも行われる見込みは低いです。

17.04のUnity 8

発表から間もない段階で執筆しているため、今後不明点が明らかになっていくと思われます。速報が必要であればgihyo.jpの連載であるUbuntu Weekly Topics[注7]をご覧ください。

先月号でも触れましたが、Ubuntu 17.04にはUnity 8のセッションがインストールされており、環境は選ぶものの誰でも試せます。16.10とは違い、日本語を入力しなければという条件はつきますが、本当にあともう一歩のところまで来ていたことがわかります。これを機会に一度試してみてください。

Web翻訳混入事件

概要

2017年2月にUbuntuやその他FLOSSプロジェクトに、当該者[注8]がWeb翻訳サービスの翻訳結果を

注4) https://Wiki.ubuntu.com/Mir/
注5) https://yunit.io/
注6) https://ubports.com/
注7) http://gihyo.jp/admin/clip/01/ubuntu-topics
注8) ここではこう表記することにします。

適用していたことが明らかになり、その対応を行う必要があった、ということがありました。対応は完了したわけではなく、現在も進行しています。Ubuntu本体は17.04での対応が完了しています。Ubuntu Budgieは対応できていません。Ubuntのリポジトリにあるパッケージの中には未対応のものも含まれている、といったところです。

筆者も当事者の1人であり、可能な限り客観的な記述に努めるものの、そのようには解釈されない部分もあるかもしれませんが、事情を汲んでいただけると幸いです。

前提知識

この問題を理解するには前提となるライセンスの知識が必要です。Web翻訳サービスはいろいろとありますが、当然のことながらそれぞれ独自の利用許諾があります。一般的には個人用途に限定されるなど厳しいものです。そしてFLOSSプロジェクトにもライセンス[注9]があります。UbuntuのLaunchpadなど、Webインターフェースで翻訳が行えるものはさらに別のもっと緩いライセンスが適用されるものもあります。厳しい利用許諾のものを緩いライセンスに入れることはできないので、ライセンス違反になるというわけです。もちろん厳密に法解釈すれば、またWeb翻訳サービスの利用許諾によっては問題ないものもあるかもしれませんが、当該者の対応や対処に問題があり、個別の対処ができなくなりました。

また、Ubuntuの翻訳方法は若干特殊です。Ubuntuではuniverseリポジトリにあるパッケージは翻訳をそのまま使用していますが、Ubuntu独自パッケージやmainリポジトリにあるパッケージの翻訳はLaunchpadのWeb翻訳インターフェースで翻訳し[注10]、その結果を言語やデスクトップ環境などの基準で分類してパッケージ化しています。そのWeb翻訳インターフェースで翻訳する際には緩いライセンスである3条項BSDライセンス（The 3-Clause BSD License）であることを求められます。

Ubuntuのコミュニティで活動する場合、行動規範（Code of Conduct）[注11]を遵守する必要があります。一定以上の貢献をする場合は電子署名ではあるものの署名をする必要もあります。

mainリポジトリにあるパッケージを和訳するために、Ubuntu Japanese Translatorsチームがあります。Launchpadのアカウントがあれば誰でも翻訳できますが、適用には特別な権限が必要です。また、翻訳した人と適用する人は別にするというローカルルールがあります。筆者はUbuntu Japanese Translatorsチームには属していませんが、Ubuntu Japanese TeamメンバーにはUbuntu Japanese Translatorsチームの権限が与えられており、翻訳を適用できます。

問題が発覚した経緯

発端は2月21日、「Japanese team (reviewer) is not working」[注12]と題するメールが、当該者によってグローバルな翻訳のメーリングリストに投稿されたことです。当該者はとくにUbuntuフレーバーのインストーラの説明の翻訳を行ったにもかかわらず、提案状態のままで適用する人がいないことが動機だったと語っています。しかし、この内容であればもっとふさわしい日本語でのメーリングリストに投稿されるべきであり、そのように忠告した人もいましたが、場の移動への同意はありませんでした。

続いてUbuntu Japanese Team/Translatorsの吉田史氏が現状を報告します[注13]。これは長文かつ、ほかの購読者に対する説明という意味合いが強いものでした。内容は「過去に著しい低品質な翻訳やライセンス的に問題があると思われるWeb翻訳サービスの翻訳が適用され、それを削除したことから厳格なプロセスを採用しており、適用に時間がかかるのはある程度はやむを得えず、また、適切な連絡手段を用いるよう依頼する」というものでした。それに対し

注9) 通常ライセンスと利用許諾は同じものですが、FLOSSソフトウェアのライセンスは通常利用許諾とは言わず、またWeb翻訳サービスの利用許諾と区別が容易ですので、分けて表記することにします。

注10) 話の本筋から逸れるので詳しく解説はしませんが、poファイルをアップロードすることによって翻訳することも可能ではあります。

注11) https://launchpad.net/codeofconduct/

注12) https://lists.ubuntu.com/archives/ubuntu-translators/2017-February/007309.html

注13) https://lists.ubuntu.com/archives/ubuntu-translators/2017-February/007314.html

当該者はなぜか「Web翻訳の精度は向上しているので無碍(むげ)に否定するものではない」という旨の投稿を行います[注14]。

これには「問題は精度ではなくライセンスである」ということとともに、「ライセンス問題がある翻訳を提案していないか」という質問が投げかけられますが[注15]、当該者はこれには答えず「翻訳プロセスが止まっていることが問題だ」という主張を続け、堂々巡りが発生することになります。とてつもない疑惑が発生した瞬間です。

これを打破するために、「プロセスが止まっている」ということへの反証として、アクティブな翻訳者である筆者が「問題となっている個所の適用を行う」意思を表明すると[注16]、なぜか英語圏では罵倒と取れる表現を伴う、批判とみられる投稿が返ってきました[注17]。この投稿に関して、第三者からCoCに気をつけよう[注18]という注意を受けると、当該者は謝罪とともにメーリングリストを退会してしまいます[注19]。ライセンスに関する深刻な疑義は残ったままで、あまりにも困った事態になりました。

そのあと、当該者は何名かの呼びかけによって問題を理解したのか復帰しますが、疑惑について明確な回答を行わず、そのまま再度メーリングリストを退会してしまいました[注20]。ライセンスの問題は疑わしき場合でも対応せざるを得ず、さらに状況としては「疑わしい」をはるかに上回る疑義を構成する状況となり、当該者による協力も見込めないまま、3月8日に当該者以外の手によって削除作業を行うことを発表しました[注21]。

対処

Ubuntu

本来は、当該者による削除作業が行われればよかったのですが、それが見込めない以上、当該者によって翻訳された項目をすべて、提案された項目を一部洗い出し[注22]、削除(revert)あるいは削除と再翻訳を行いました。いわゆるクリーンルーム実装ならぬクリーンルーム翻訳で行われたため、17.04からは問題がなくなっています。

Ubuntu Budgie

Budgieそのもの(budgie-desktop)と端末(terminix)に当該者による翻訳がありましたが、前者は筆者によりクリーンルーム翻訳[注23]が行われ、後者は名称がTilixに変更されてから筆者が削除作業を行いました。いずれも17.04には間に合っていません。

Ubuntu MATE

MATEに関しては、最新版である1.18のリリースまでにMATEの和訳の管理者によって再翻訳が行われており、対処としては完了しています。頭が下がります。

Linux Mint

4月中旬現在でもまだ対応中です。Cinnamonデスクトップ環境はLinux Mintのプロジェクトですが、Ubuntuのリポジトリにもあるので影響を受けていると言えます。

Ubuntu Japanese Transratorsチーム

ルールに脇が甘い部分があったことは否めないため、翻訳の削除と再翻訳と並行して翻訳ガイド[注24]と参加要件[注25]の見直しも行われました。3月に毎週IRCミーティングを行い、議論を行いました。

その他

ほかにも対処が行われたもの、行われないものがありますが、当該者が口をつぐんでしまった以上、全貌が明らかになるのは期待できません。SD

注14) https://lists.ubuntu.com/archives/ubuntu-translators/2017-February/007315.html

注15) https://lists.ubuntu.com/archives/ubuntu-translators/2017-February/007316.html

注16) https://lists.ubuntu.com/archives/ubuntu-translators/2017-February/007330.html

注17) https://lists.ubuntu.com/archives/ubuntu-translators/2017-February/007331.html

注18) https://lists.ubuntu.com/archives/ubuntu-translators/2017-February/007333.html

注19) https://lists.ubuntu.com/archives/ubuntu-translators/2017-February/007334.html

注20) https://lists.ubuntu.com/archives/ubuntu-translators/2017-March/007350.html

注21) https://lists.ubuntu.com/archives/ubuntu-translators/2017-March/007352.html

注22) https://wiki.ubuntulinux.jp/WIP/translation/ReReviewTranslations

注23) 既存の翻訳ファイル(ja_JP.po)から和訳部分を削除するという方法ではなく、翻訳の元となるファイル(budgie-desktop.pot)からすべて翻訳し、新旧の翻訳ファイルの差分を取り、違いがあるものを翻訳を行うWebサービスであるTransifexに入力するという手間がかかっています。

注24) https://wiki.ubuntulinux.jp/Develop/TranslationGuide

注25) https://wiki.ubuntulinux.jp/enroll/translator_candidates

Unixコマンドライン探検隊
Shellを知ればOSを理解できる

Author 中島 雅弘(なかじま まさひろ) ㈱アーヴァイン・システムズ

利用者の視点から一歩踏み込み、OS視点でコマンドの種類、リダイレクトとパイプを眺めることで、コマンドとコマンドの連携時の内部動作に迫ります。

第14回 コマンドの種類とプロセス、パイプ、リダイレクトのしくみ

コマンドの実行

私達がなにげなく普段実行しているコマンドは、独立したプログラムばかりではなく、シェルの機能の一部分であることもあります。まずは、これらの違いを見てみましょう。

外部コマンド

基本的なコマンドは、/bin、/usr/bin、/usr/share/binなどにある実行可能ファイルを起動することです。これらを後に説明するシェルの機能と区別するために、あえて外部コマンドと呼びます。

```
/bin/echoを実行する例
$ /bin/echo abc ↵
abc
```

外部コマンドは、起動したプロセスの状態であれば、psコマンド[注1]で確認できます。これらの大半は、CPUが直接解釈できるバイナリの実行形式です。また、Unixでは、シェルスクリプトやRuby、Perlなどのスクリプト言語のスクリプトであっても、実行権限を付与することで区別なく実行できます。これらはpsコマンドで状況を確認すると、bash、ruby、perlが実行されていることがわかります。

内部コマンド – ビルトインコマンド

pwdなどに代表される、シェルの内部で処理するコマンドを内部コマンド[注2]と呼びます。内部コマンドは、(bashを使っているなら)bashプロセス中で処理されますので、実行中もpsコマンドでは確認することができません。使い方は、bashのhelpで確認できます。

先の、/bin/echoは外部コマンドでしたが、echoは同じ名前の内部コマンドとしても存在しています。

```
$ echo abc ↵
abc
```

whichコマンド[注3]を使っても、内部コマンドかどうかはわかりません。typeコマンド[注4]なら区別できます。

```
$ which echo ↵
/bin/echo
whichでは外部コマンドのみを表示
typeで調べる
$ type echo ↵
echo is a shell builtin
$ type -a echo ↵
echo is a shell builtin
echo is /bin/echo
```

注1) ps(Process Status):2016年7月号の連載第3回で紹介した、プロセスの状態を表示するコマンド。

注2) 英語ではbuiltin commands。

注3) コマンド名を入力して実行する際(同名のプログラムが複数ある場合など)に、パス上のどのプログラムが実行されるか確認する。

注4) コマンドなどの種類を判別するbashの内部コマンド。

alias – エイリアス

よく使うコマンドとオプション・引数の組み合わせを毎回入力するのが面倒なときに便利なのが、**alias**です。aliasとは、別名のことですので、コマンドやコマンドとオプションの組み合わせに対して名前を付けておくしくみです。aliasも、ほかのコマンドとまったく同じように呼び出せます。

aliasの定義は、

```
alias別名 = 実行するコマンド + 引数
```

とします。

aliasを引数なしで呼び出せば、定義されているaliasがすべて表示されます。

```
Ubuntuでの例（デフォルトで定義されているものがある）
$ alias ↵
alias alert='notify-send --urgency=low -i ➘
"$([ $? = 0 ] && echo terminal || echo ➘
error)" "$(history|tail -n1|sed -e ➘
'\''s/^\s*[0-9]\+\s*//;s/[;&|]➘
\s*alert$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

前の例の最後の行を見ると、Ubuntuでは、lsが色付きで表示するようにaliasを定義してあるのがわかりますね。macOSでは、次のように定義すればlsを色付きで表示させられます。

```
alias ls='ls -G'
```

aliasもtypeで調べることができます。

```
Ubuntuでの確認例（macOSでの標準設定では、aliasが何も定義されていない）
$ which l ↵
何も出力されないか、パス内にはないというメッセージ
$ type l ↵
l is aliased to `ls -CF'
```

aliasで定義されているコマンドを、aliasとして解釈させたくないときは、aliasの先頭に\を付けて記述します。

```
Ubuntuでのlsで色を付けて表示したくない場合
$ \ls ↵
...略...
```

function – bashの関数

シェルは、OSと利用者の仲立ちとなって、命令や結果をやりとりするインターフェースである一方、スクリプト言語としての側面もあります。一連の処理をシェルスクリプトとして記述して、コマンドのように呼び出せるのはご存じでしょう。シェルスクリプトには、**function**を定義でき、利用者はfunctionを任意に呼び出すことができます。コマンドラインからfunctionを記述することもできるので、簡単な例を見てみましょう。

```
$ function abc() { echo abc; echo def; } ↵
↑関数の定義
$ abc ↵
abc
def
```

abcがまるで、コマンドのように呼び出しできました。これもtypeで確認できます。

```
Ubuntuでの例
$ which abc ↵
何も出力されないか、パス内にはないというメッセージ
$ type abc ↵
abc is a function
abc ()
{
    echo abc;
    echo def
}
```

declare – DECLARE

シェルは、明示的に定義しなくても変数を使えます。しかし、データ型やスコープを明確にしておく必要がある場合には、**declare**を使います。declareはオプションで、後続の定義の型を指定(-a：配列、-f：関数、-i：整数、-r：読み取り専用、-x：環境変数としてエクスポート)します。

declareは、引数に定義部分を与えなければ、変数、関数の定義を確認できます。

整数で定義されている変数

```
$ declare -i ↵
declare -ir BASHPID
declare -ir EUID="1000"
declare -i HISTCMD
declare -i LINENO
declare -i MAILCHECK="60"
declare -i OPTIND="1"
declare -ir PPID="15095"
declare -i RANDOM
declare -ir UID="1000"
```

また、定義を削除するには、unsetを使います。

```
unset -f 関数名
unset -v 変数名
```

typesetもdeclareとほぼ同じですが、あまり使われなくなってきています。

STEP UP! プロセスの連携 – シェルの視点から

ここからは、シェルの動作の内側を見ていきます。動作原理の要点を理解するために、模式的なC言語のプログラムを掲載します。C言語のプログラムは、すべてを理解する必要はありません。本文で解説されたシステムコールやライブラリファンクションがどのように使われるのか、プログラムの流れを追いかけてみてください。

forkとexec

シェルで外部コマンドを実行する際、シェルは内部で、システムコールforkを実行してからexecを呼び出します。

forkは実行中のプロセスをコピー、新たにクローンプロセスを作成します。**図1**を見てみましょう。forkされたプロセスには新しいPIDが割り当てられ、forkの直後から同様に処理を続けます。

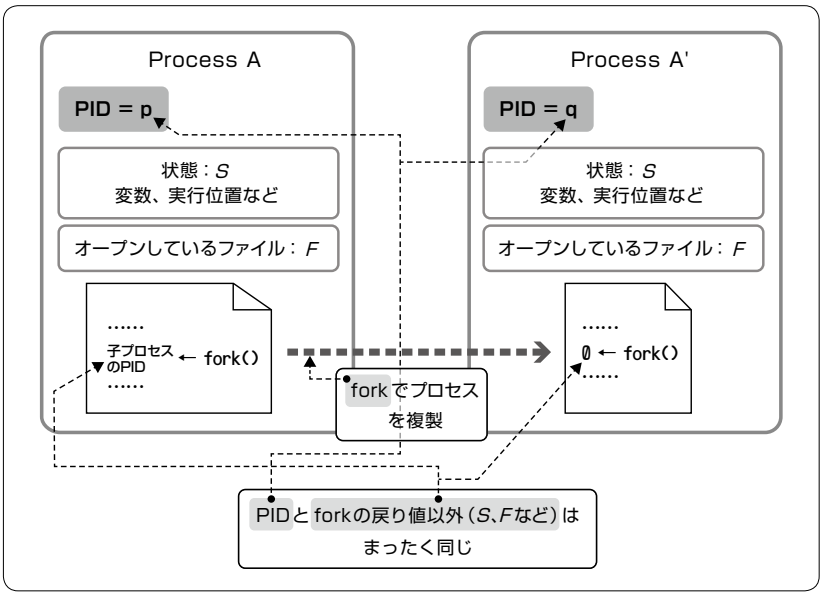
execは、実行中のプロセスを、別のプログラムイメージに置き換えます。**図2**を見てください。exec前と後では、違うプログラムを実行中の状態(プロセス)になっていますが、同じPIDのままです。exec前にオープンしているファイルのファイルディスクリプタ[注5]は、(明示的にexec時にcloseするというフラグが立っていなければ)オープンしたまま引き継がれます[注6]。

リスト1は、外部コマンド実行時の内部処理を抽象化したものです。bashからlsコマンドを実行するということは、bashプロセスをforkして子プロセスを作り(この時点ではbash親プロセスとクローンの子プロセスがあるだけ)、子プロセス側のbashからexecでlsを実行することで、bashのイメージをlsに置き換える、ということです。

注5) 後述「ファイルディスクリプタの話」参照。

注6) execve(2)がシステムコール、ほかのexecl、execlp、execle、execv、execvp、execvpeなどは、内部でexecveを呼び出すライブラリ関数。

▼図1 forkの動き



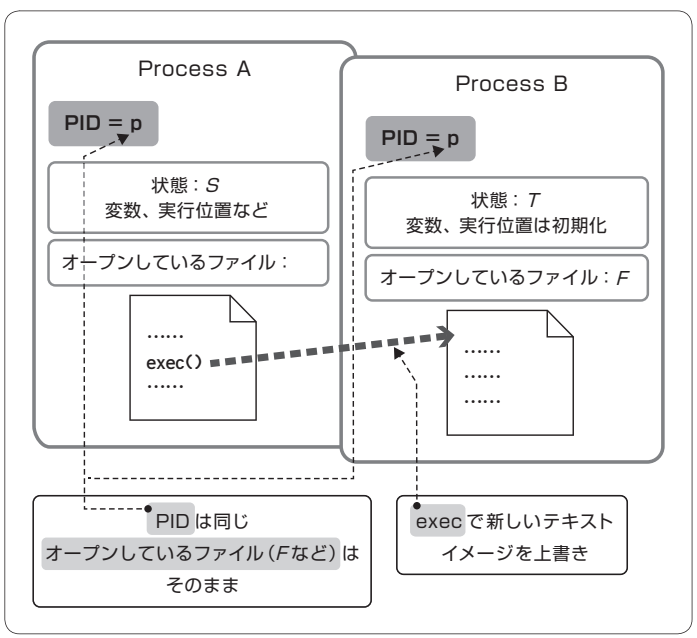
▼リスト1　fork & execのプログラムサンプル（fork_exec.c）

```
 1 #include <unistd.h>
 2 #include <sys/wait.h>
 3 #include <stdlib.h>
 4 #include <stdio.h>

 5 void child(char **);
 6 void parent(int);

 7 int main(int argc, char **argv, char
   **envp)
 8 {
 9     int pid;
10     char **e;

11     for (e = envp; *e != NULL; e++)
12         printf("%s\n",*e);

13     if ((pid = fork()) < 0) {
14         perror("at fork!");
15         exit(1);
16     }

17     if (pid > 0)
18         parent(pid);
19     else
20         child(envp);

21     return 0;
22 }

23 void child(char **envp)
24 {
25     char *argv[6];
```

右段へ続く↗

```
26     printf("I am a child process.
   pid=%d\n", getpid());

27     argv[0] = "/bin/ls";
28     argv[1] = ".";
29     argv[2] = "/tmp";
30     argv[3] = "/bin";
31     argv[4] = "/usr";
32     argv[5] = NULL;
33 #ifdef EXECV  // execvを使う場合
34     puts("execv");
35     execv(argv[0], argv);
36 #else // execl BUG EXISTS on OSX El
   Capitan, BUG FIXED on Sierra
37     puts("execl");
38     execl(argv[0], argv[0], argv[1],
   argv[2], argv[3], argv[4], 0);
39 #endif
40     printf("execl faild\n");
41 }

42 void parent(int pid)
43 {
44     int     status;

45     printf("I am a parent process. The
   child's pid=%d.\n",pid);
46     if (wait(&status) < 0) {
47         perror("at wait!");
48         exit(2);
49     }
50     printf("child process : done.\n");
51 }
```

▼図2　execの動き



open、close、パイプ、リダイレクト、dup、pipe

リダイレクトは連載第5回（2016年9月号）で紹介しました。コマンドの出力先を、標準出力や標準エラー出力から、別のファイルやデバイスに切り替えるしくみです。入力も同様に、標準入力から切り替えることができます。これも内部でどのように処理されているのか見てみましょう。

```
$ ls > ls_output.txt ↵
$ cat < ls_output.txt ↵
...略...
```

ファイルディスクリプタの話

ファイルディスクリプタは、ファイルをオープンすると割り当てられる、各プロセス内で固有の「整数値」です。0、1、2は、どのプロセスでもオープン(下表)してあり、新たに別のファイルをオープンすると(通常は)3、4、……と割り当てられます。

openはシステムコールで成功すればファイルディスクリプタを返します。fopenは、ストリームをオープンするライブラリファンクションで、FILE型の構造体へのポインタが返ります。

対象	FILE型	ファイルディスクリプタ	デフォルトでの向き先
標準入力	stdin	0	tty(キーボード)
標準出力	stdout	1	tty(ディスプレイ)
標準エラー出力	stderr	2	tty(ディスプレイ)

リダイレクト処理は、ファイルの入出力先を結合したり、無用な入出力先を閉じることで実現します。OS視点では、dup：すでにあるファイルディスクリプタを複製する、close：ファイルを閉じる、などを用いています。**図3**は、dupを使ってリダイレクトを模倣しています。

リスト2のソースコードで確認してみると、紛れがありません。

シェル上でのパイプ'|'処理は、前のコマンドの(標準)出力を、後ろのコマンドの(標準)入力として連結します。

パイプによるコマンドの連結
```
$ ls | head ↵
...略...
```

pipeというシステムコールがあります。pipeは、新しい入出力が組になっているファイルディスクリプタを作成します。これを使うとforkしたプロセスとの間で、(望めば)双方向通信ができます。

ちょっと変な感じですがpipeで作ったファイルディスクリプタの出力は、自身の入力に結び付けられます。シェルのパイプ処理によるコマンドの連結は、片方の入力を閉じて、もう一方の出力を閉じるというやり方で、一方通行の入出力ストリームを実現しています。**図4**を、処理の順序をたどってよく見てください。

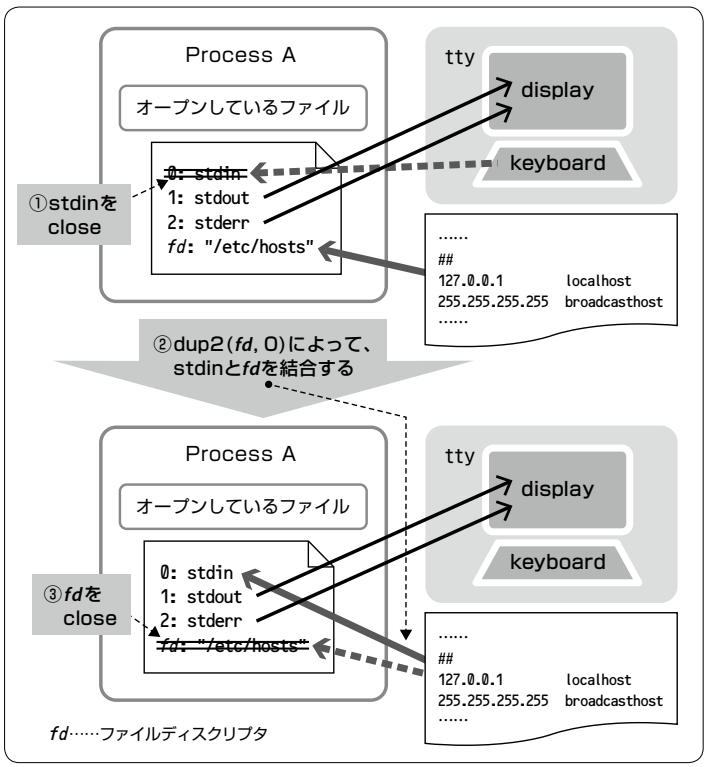
つまり、I/Oを複製して、閉じる。出し入れ口をつなぎ替えていることがイメージできると思います。次ページの**リスト3**のソースコードで、細部まで確認してみてください。

pipe関数はdupを使っても実現できますが、open、closeをする回数がずっと増えて煩雑になります。

高度なリダイレクトとパイプ

リダイレクトやパイプは、入出力先を複製して、無用な出入口を閉じるという2段階、これがイメージできると、複雑なパイプやリダイレクトの連結が使えるようになります(次ページの**表1**)。

▼図3　dupによるリダイレクトの実現

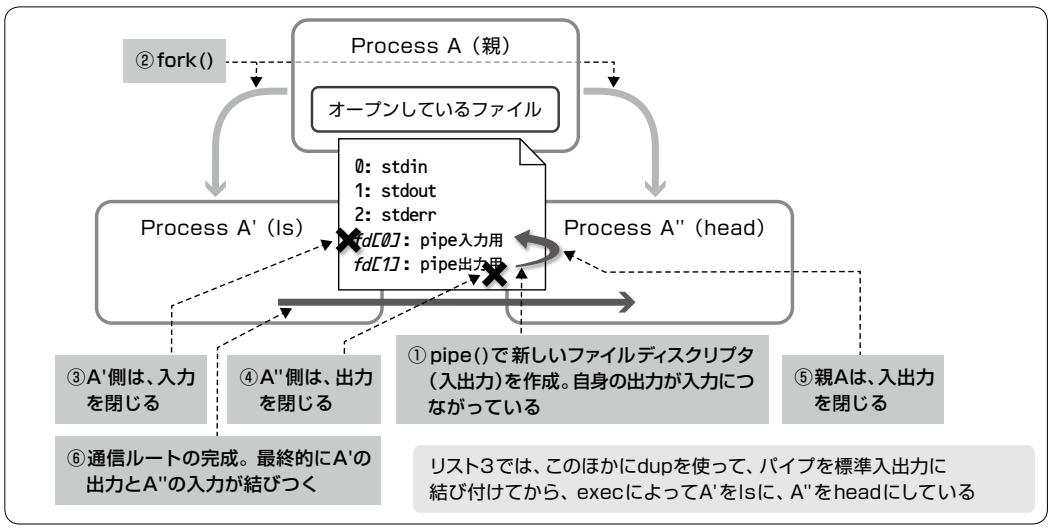


▼リスト2　dup2のプログラムサンプル (do_dup.c)

```
 1 #include <unistd.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <fcntl.h>

 5 #ifndef TARGET
 6 #define TARGET "/etc/hosts"
 7 #endif

 8 void child();
 9 void parent(int);

10 int main(int argc, char *argv[])
11 {
12     int fd;
13     fd = open(TARGET, O_RDONLY);
14     if (fd < 0) {
15         perror("open failed");
16         close(fd);
17         exit(2);
18     }

19     close(0);
20     dup2(fd, 0);
21     close(fd);

22     execlp("head", "head", NULL);
23     return 0;
24 }
```

右段へ続く↗

▼図4　pipeのイメージ



シェルのリダイレクト記号‘>’とパイプ記号‘|’の直前、直後には、ファイルディスクリプタを記述できます。省略すると記号の左側は、標準出力(1)を指定したということです。

コマンドラインから、ほかにもopen、dup、closeに対応した指定がさまざまできます。次ページの**表2**でもう少し詳しく見てみましょう。

```
$ cd /var
↑一般ユーザで、varディレクトリの下を見る例。権限がないところは見られないので、エラーが出ることを期待
$ ls -R > /dev/null 2>&1
↑結果もエラーも捨ててしまう場合
$ ls -R 2>&1 > /dev/null
↑これでは、うまくいかない
```

なんだか複雑に見えるかもしれませんが、*n*>&*o*は、dup2(*o*,*n*)しているのです。シェル操作で、リダイレクト、パイプ処理の記述は数字や記号が入り乱れて、読みづらかったり、各順序が混乱したりということがありますが、このように内部的な処理(fork、exec、dup、pipe、closeなど)がイメージできれば、正しく記述できるようになるはずです。

例に戻って、動きを確認してみます。

1つめは、どうして標準出力・エラー出力が両方/dev/nullに捨てられるのでしょうか。

①標準出力を/dev/nullに変更する
②標準エラー出力を、標準出力の出力先と同じにする：つまりエラー出力も/dev/nullになる

▼リスト3　pipeのプログラムサンプル (do_pipe.c)

```
 1 #include <unistd.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <fcntl.h>

 5 #ifndef TARGET
 6 #define TARGET "/etc/hosts"
 7 #endif

 8 void child(int);
 9 void parent(int);

10 int fd[2];
11 int pid[2];

12 char *command[] = {"ls", "head"};

13 int main(int argc, char *argv[])
14 {
15
16     int i, cp;
17     int status;

18     if (pipe(fd) < 0) {
19         perror("pipe creation failed");
20         exit(1);
21     }

22     for (cp=0; cp < 2; cp++) {
23         if ((pid[cp] = fork()) < 0) {
24             fprintf(stderr, "fork ➥
   failed (cp=%d)\n", cp);
25             exit(1);
26         }

27         if (pid[cp] == 0) {
28             child(cp);
29         }
30     }

31     // 親プロセスではpipeは不要。入出力と➥
   も必ず閉じる
32     for (i=0; i<2; i++) close(fd[i]);

33     for (cp=0; cp<2; cp++) {
34         waitpid(pid[cp], &status, 0);
35         printf("%s (cp=%d, pid=%d) : ➥
   done\n", command[cp], cp, pid[cp]);
36     }

37     return 0;
38
39 }

40 void child(int cp)
41 {
42     fprintf(stderr, "I am going to be ➥
   '%s'. pid=%d, cp=%d\n", command[cp], ➥
   getpid(), cp);

43     close(fd[cp]); // 先に実行pipeの入力➥
   をclose、後に実行pipeの出力をclose

44     close(cp^1);     // 先に実行stdoutを➥
   close、後に実行stdinをclose
45     dup2(fd[cp^1], cp^1);  // パイプを➥
   stdioに結合
46     close(fd[cp^1]);       // 結合できた➥
   らpipeは不要
47     execlp(command[cp], command[cp], ➥
   NULL);
48     fprintf(stderr, "NEVER REACH ➥
   HERE... child %d : done\n", cp);
49 }
```

右段へ続く↗

▼表1　リダイレクトとパイプのいろいろ(基本形)

書式	実行内容
cmd1 | cmd2	ご存じパイプ、cmd1の標準出力をcmd2の標準入力へ連結
> file	標準出力をfileにする
< file	標準入力をfileにする
>> file	標準出力をfileに追記する

▼表2　リダイレクトとパイプのいろいろ(発展形)

書式	実行内容
n> file	ファイルディスクリプタ*n*をfileに出力する
n< file	ファイルディスクリプタ*n*としてfileをオープンする
n>&*o*	ファイルディスクリプタ*n*を(出力)ファイルディスクリプタ*o*のコピーにする(*n*は省略すれば1)
n<&*o*	ファイルディスクリプタ*n*を(入力)ファイルディスクリプタ*o*のコピーにする(*n*は省略すれば0)
&> file	標準出力と標準エラー出力の両方をfileに出力する
<&-	標準入力をcloseする
>&-	標準出力をcloseする
n<&-	入力ファイルディスクリプタ*n*をcloseする
n>&-	出力ファイルディスクリプタ*n*をcloseする

2つめのlsのエラー出力が/dev/nullに捨てられないで、標準出力に出ている理由も、よく考えてみましょう。

① エラー出力の出力先を標準出力と同じにする：これは、エラー出力も標準出力もともとttyへの出力だから、意味がない

② 標準出力を/dev/nullにする：標準出力は/dev/nullだが、エラー出力はttyのまま

ちなみに次のようにすれば、出力とエラー出力を別の出力先にして保存

することもできます。ファイルディスクリプタを操作の対象にしていることを意識しましょう。

```
結果をnormal.txtに、エラー出力があればerror.txtに
$ ls -R > normal.txt 2> error.txt
```

ヒアドキュメントとヒアストリング

bashには、一風変わっていますが便利なリダイレクトに、ヒアドキュメントとヒアストリングがあります。

ヒアドキュメント(<<)は、PerlやRubyなどのメジャーなスクリプト言語にもあります。コマンドに対する入力を、スクリプト中の(改行も含めた)一連の文字列にすることができます。データと処理ロジックを一緒に書いておけるので、プログラムの動作の意味が明瞭になります。

```
<< 識別用文字列
```

とすると、次に**識別用文字列**が現れるまでを、処理ロジックではなく、データとしてリダイレクトします。

```
ヒアドキュメントの例(END_OF_TEXTが出現するまで、プロンプトが'> 'になって、データの入力ができる
$ cat <<END_OF_TEXT ↵
> I have a pen. ↵
> I have an apple. ↵
> END_OF_TEXT ↵
```

変数の内容をコマンドにリダイレクトできるbashの**ヒアストリング(<<<)**は、とても便利です。

```
<<< ${val}
```

第12回(2017年4月号)のhangmanをbashスクリプト(bangban)で実現したときにも活躍しました。これを使うと、一時ファイルに結果を出力しなくてもよいなど、スクリプトが簡潔になります。

```
bangbanでの活用例:while処理に$word変数をリダイレクトする
while read -n 1 c; do if [ "$c" != "" ] ; ↵
then w+=("$c") ans+=("-"); fi; done <<< $word
```

bashのexec

bashには、内部コマンドとして**exec**があります。シェルの代わりに別のコマンドを実行したり、execするコマンドを指定しなければ新しいbashが起動され、以降すべての処理結果のリダイレクト先を指定したりできます。

```
エラー出力は、すべてerror_output.txtに
$ exec 2> error_output.txt
```

今回の技術が活躍するところ

コマンドには、外部コマンド、内部コマンドがあり、ほかにもシェルの関数やaliasなどがコマンドのように呼び出せることを解説しました。また、少し高度なパイプやリダイレクト処理を紹介しました。

外部コマンドの実行、パイプやリダイレクト実現のメカニズムがどのようにOS上で実現されているかと、Unixがマルチユーザ・マルチプロセスであることをきちんと理解することで、複雑でわかりにくそうなシェル独特の記法も問題なく、正確な操作や効率的なスクリプティングができるようになります。

次回について

次回は、sshを使って、リモートアクセスを体験します。SD

今回の確認コマンド

【manで調べるもの
(括弧内はセクション番号)】
which(1), bash(1), csh(1), fork(2), execve(2), exec(3), open(2), close(2), dup(2), pipe(2), fopen(3), fprintf(3)
【以下はbashのhelpコマンドを使って確認】
alias, type, declare, typeset, local, unset, exec

第62回

SCSIのPersistent Reserveを一般に使えるようにするioctl()

Text：青田 直大　AOTA Naohiro

2017年5月

4月16日にLinux 4.11-rc7がリリースされています。いろいろRevertもありつつ(筆者もNICのドライバがおかしかったのがRevertされて助かりました)、RC7ということでそろそろLinux 4.11がリリースされるでしょう。

今月はSCSIの機能の1つであるPersistent Reserveをより一般に使うことを可能にする新しいioctl()について紹介します。

Persistent Reserveとは

クラスタ環境などでは、複数のマシンが使うストレージを一ヵ所にまとめておく構成をとることがあります。たとえばiSCSIを使うことで、複数のマシン間で1つのストレージを共有して使うことができます。Persistent Reserveとは、こうした環境でストレージの排他制御を行うためのSCSIの機能です。Persistent Reserveを獲得することで、ほかのマシンからのアクセスを禁止できます。

Persistent Reserveの種類

Persistent Reserveで排他制御を行う際には、禁止される操作と、アクセス範囲とがそれぞれ異なる6種類のタイプのいずれかを指定します。操作については、書き込みだけを禁止するか、読み出しを含めたすべての操作を禁止するかの2種類です。アクセス範囲には、1つはPersistent Reserveを取得したマシンにだけアクセスを許可するモードであるExclusiveと、Persistent Reserveを取得したマシン以外にも「登録した」マシンからのアクセスも許可するモードであるExclusive Registrants Only、およびExclusive All Registrantsの合わせて3つがあります(**図1**)。

Exclusive Registrants Onlyと、Exclusive All Registrantsとの違いは、"Reservation Holder"の扱いにあります。Reservation Holderとは、いわばReservationの管理者で、Reservationを解除する権限を持ちます。Exclusive Registrants Onlyは、Reservationを登録したマシンのみがReservation Holderとなり、Exclusive All Registrantsではすべての登録マシンがReservation Holderとなります。すなわち、Exclusive All

Registrantsでは、Reservationを登録したマシン以外のマシンから、登録を解除できるということになります。クラスタで使うような例でいえば、Exclusive Registrants Onlyは、マスタとなるマシンが排他制御を管理する場合に使い、Exclusive All Registrantsはすべてのマシンが対等な場合に使うことになります。

sg3_utilsでのPersistent Reserveの設定

sg3_utils[注1]の**sg_persist**コマンドを使うことで、Persistent Reserveを操作できます。それでは実際に、2つのマシンから接続されたiSCSIディスク(target)に対して、Persistent Reserveの操作を行ってみましょう。

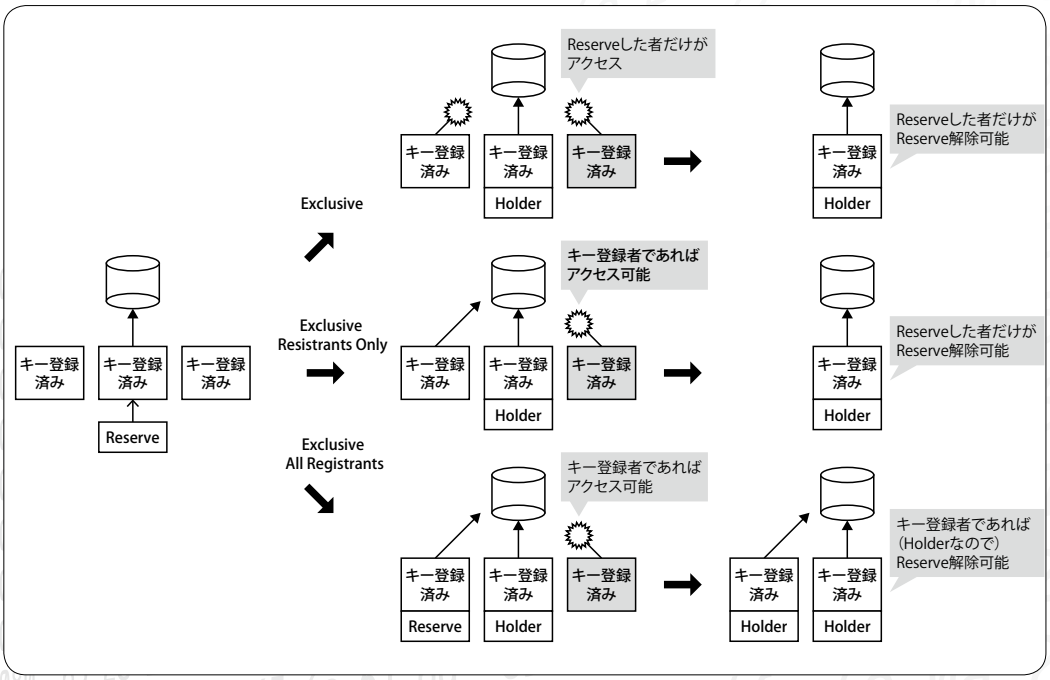
LVM上のディスクをバックエンドとしてiSCSIのtargetを作成し、両方のマシンからlun0にアクセスできるようにACLを設定します。どちらのマシンからも、**/dev/sdb**にiSCSIディスクを接続します。HostAは"**iqn.2005-03.org.open-iscsi:a13e4179ff7**"で接続し、HostBは"**iqn.2005-03.org.open-iscsi:260dc255081**"で接続しています。

Persistent Reserveの設定を行うには、まずキーを登録します。始めに**sg_persist -s**コマンドで初期状態を確認しておきましょう。**図2**のように、Persistent Reserveのgeneration以外の情報は返ってきていません。キーの登録は**sg_persist -o -G -S <KEY>**コマンドで行います(**図2**の「キーの登録」以降)。**-o**はディスク側の設定に変更を加えるコマンドの場合に指定します。また、**-G**が登録を行うことを指定し、**-S**とその引数はキーを指定しています。ここでは、HostAに"**12345678**"というキーを、HostBに"**9ABCDEF0**"というキーを設定しています。

キーを設定して、もう一度状態を見てみます(**図2**の「登録されたキーの確認」以降)。今度はそれぞれのホストが登録したキーが表示されています。また、どちらのキーもReserveをしていないので、Reservation Holderにはなっていません。

注1) http://sg.danny.cz/sg/

▼図1　Persistent Reserveのアクセス範囲



▼図2　キーの登録

```
Persistent Reserveの確認: KeyもReservationも登録されていない
HostA $ sudo sg_persist -s /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
  PR generation=0xa5
  No full status descriptors
キーの登録
HostA $ sudo sg_persist -o -G -S 12345678 /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
HostB $ sudo sg_persist -o -G -S 9ABCDEF0 /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
登録されたキーの確認
HostA $ sudo sg_persist -s /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
  PR generation=0xa7
    Key=0x12345678
      All target ports bit clear
      Relative port address: 0x1
      not reservation holder
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:a13e4179ff7
    Key=0x9abcdef0
      All target ports bit clear
      Relative port address: 0x1
      not reservation holder
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:260dc255081
```

▼図3　Write Exclusiveの設定

```
Reserveの設定
HostA $ sudo sg_persist -o -R -K 12345678 -T 1 /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
HostA $ sudo sg_persist -s /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
  PR generation=0xb2
    Key=0x12345678
      All target ports bit clear
      Relative port address: 0x1
      << Reservation holder >>
      scope: LU_SCOPE,  type: Write Exclusive   ←typeがWrite Exclusiveになっている
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:a13e4179ff7
    Key=0x9abcdef0
      All target ports bit clear
      Relative port address: 0x1
      not reservation holder
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:260dc255081
HostA以外からは書き込めなくなる
HostA $ sudo dd if=/dev/zero of=/dev/sdb bs=4096 count=1 oflag=sync
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.0202637 s, 202 kB/s
HostB $ sudo dd if=/dev/zero of=/dev/sdb bs=4096 count=1 oflag=sync
dd: error writing '/dev/sdb': Input/output error
1+0 records in
0+0 records out
```

次ページに続く↗

```
0 bytes copied, 0.0765853 s, 0.0 kB/s
HostB $ dmesg
dmesg
[101050.271759] sd 30:0:0:0: reservation conflict  ←Reservationが原因でWriteが失敗している
[101050.280662] sd 30:0:0:0: [sdb] tag#46 FAILED Result: hostbyte=DID_OK driverbyte=DRIVER_
OK
[101050.298433] sd 30:0:0:0: [sdb] tag#46 CDB: Write(10) 2a 00 00 00 00 00 00 00 08 00
[101050.316731] blk_update_request: critical nexus error, dev sdb, sector 0
[101050.326244] Buffer I/O error on dev sdb, logical block 0, lost async page write
[101050.344757] VFS: Dirty inode writeback failed for block device sdb (err=-5).
```

次に、HostAに完全な排他アクセス権(Exclusive)を設定してみましょう。Reserveの設定には**-R**オプションを使い、**-T**でReserveの種類を指定します(**図3**)。ここではWrite Exclusiveを意味する"1"を指定します。再度、状態を読み取ると、HostAのキーである"12345678"側が、Reservation Holderとなり、そのtypeがWrite Exclusiveになっていることが確認できます。この状態で両方のマシンからddで書き込みをしてみましょう(**図3**の「HostA以外からは書き込めなくなる」以降)。HostAから問題なく、HostBからの書き込みはエラーになっています。dmesgを見ると、"reservation conflict"とReservationが原因でWriteが失敗していることがわかります。

次に、ReserveのtypeをExclusive Registrants Onlyに切り替えてみましょう(**図4**)。まず、**-L**オプションを使ってReserveを解放し、ふたたび**-R**でExclusive Registrants Only(type=5)に切り替えます。

今度もddで書き込んでみると、どちらのホストもキーを登録しているので、書き込みが成功します。ここでさらに、HostBの登録を解除してみましょう。すると、ステータスからHostBのキーが消え、ddの書き込みが失敗するようになります。このように、Exclusive Registrants Onlyは、キーを登録しているホストにのみアクセス権を与えます。

最後にExclusive All Registrantsの場合を見ていきましょう。HostAでExclusive All Registrants(type=8)の設定を行い、ステータスを見てみます(**図5**)。すると、先ほどと違ってどちらのキーにも"Reservation holder"がついていることがわかります。ということで、HostBから

▼図4　Exclusive Registrants Onlyの設定

```
現在のReserveを解放
HostA $ sudo sg_persist -o -L -K 12345678 -T 1 /dev/sdb
Exclusive Registrants Onlyの設定
HostA $ sudo sg_persist -o -R -K 12345678 -T 5 /dev/sdb
  LIO-ORG   target           4.0
  Peripheral device type: disk
HostA $ sudo sg_persist -s /dev/sdb
  LIO-ORG   target           4.0
  Peripheral device type: disk
  PR generation=0xbc
    Key=0x12345678
      All target ports bit clear
      Relative port address: 0x1
      << Reservation holder >>
      scope: LU_SCOPE,  type: Write Exclusive, registrants only
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:a13e4179ff7
    Key=0x9abcdef0
      All target ports bit clear
      Relative port address: 0x1
      not reservation holder
      Transport Id of initiator:
```

次ページに続く↗

```
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:260dc255081
HostA $ sudo dd if=/dev/zero of=/dev/sdb bs=4096 count=1 oflag=sync
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.0283303 s, 145 kB/s   ←書き込みは成功
HostB $ sudo dd if=/dev/zero of=/dev/sdb bs=4096 count=1 oflag=sync
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.00983151 s, 417 kB/s   ←書き込みは成功
HostBの登録を外す
HostB $ sudo sg_persist -o -G -K 9ABCDEF0 /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
HostA $ sudo sg_persist -s /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
  PR generation=0xc2
    Key=0x12345678
      All target ports bit clear
      Relative port address: 0x1
      << Reservation holder >>
      scope: LU_SCOPE,  type: Write Exclusive, registrants only
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:a13e4179ff7
   ここにあったHostBのキーが消えた
HostB $ sudo dd if=/dev/zero of=/dev/sdb bs=4096 count=1 oflag=sync
dd: error writing '/dev/sdb': Input/output error   ←書き込みは失敗
1+0 records in
0+0 records out
0 bytes copied, 0.0885822 s, 0.0 kB/s
```

▼図5　Exclusive All Registrantsの設定

```
HostA $ sudo sg_persist -o -R -K 12345678 -T 8 /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
HostA $ sudo sg_persist -s /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
  PR generation=0xc6
    Key=0x12345678
      All target ports bit clear
      Relative port address: 0x1
      << Reservation holder >>   ←“Reservation holder”がついている
      scope: LU_SCOPE,  type: Exclusive Access, all registrants
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:a13e4179ff7
    Key=0x9abcdef0
      All target ports bit clear
      Relative port address: 0x1
      << Reservation holder >>   ←“Reservation holder”がついている
      scope: LU_SCOPE,  type: Exclusive Access, all registrants
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:260dc255081
HostBからReserveを解除
HostB $ sudo sg_persist -o -L -K 9ABCDEF0 -T 8 /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
HostA $ sudo sg_persist -s /dev/sdb
  LIO-ORG   target            4.0
  Peripheral device type: disk
  PR generation=0xc6
    Key=0x12345678
      All target ports bit clear
```

次ページに続く↗

```
      Relative port address: 0x1
      not reservation holder   ← "Reservation holder"が外れている
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:a13e4179ff7
    Key=0x9abcdef0
      All target ports bit clear
      Relative port address: 0x1
      not reservation holder   ← "Reservation holder"が外れている
      Transport Id of initiator:
        iSCSI name and session id: iqn.2005-03.org.open-iscsi:260dc255081
```

Reserveの解除を行ってみましょう。すると、どちらのキーからも"Reservation holder"が正しく外れていることがわかります。

Linuxにおける Persistent Reserveのサポート

さて、**sg_persist**をstraceしてみるとわかりますが、このコマンドは**ioctl(SG_IO)**を使って直接SCSIを使ってPersistent Reserveを扱っています。直接SCSIデバイスだけを相手にしている場合は、これでもかまいませんが、その上にLVMで論理ボリュームを構築すると、論理ボリュームにどの物理デバイスが対応しているかなど扱いが面倒になります。また、NVMeにもおいてもSCSIのPersistent Reserveと互換性のあるReserveを行えるようになっています。そこで、SCSI、LVM、NVMeのどれに対しても動く**ioctl(IOC_PR_*)**が追加されています。

簡単にキーの登録とReserveを**ioctl(IOC_PR_*)**を用いて行うプログラムを見ていきましょう(**リスト1**)。

▼リスト1　ioctl(IOC_PR_*)を使ったPersistent Reserveの登録(エラー処理は省略)

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <linux/types.h>
#include <linux/pr.h>
#include <sys/ioctl.h>

#define register_key(fd, k) _register_key(fd, 0, k)
#define unregister_key(fd, k) _register_key(fd, k, 0)

int _register_key(int fd, int old_key, int new_key) {
  struct pr_registration reg;
  memset(&reg, 0, sizeof(reg));
  reg.old_key = old_key;
  reg.new_key = new_key;
  return ioctl(fd, IOC_PR_REGISTER, &reg);
}

#define DEV "/dev/sdb"
#define KEY 0x12345678

int main(int argc, char *argv[]) {
  int fd = open(DEV, O_RDWR);

  printf("register key: 0x%x\n", KEY);
  register_key(fd, KEY);
  struct pr_reservation reserve;
  reserve.key = KEY;
  reserve.type = PR_WRITE_EXCLUSIVE;
  ioctl(fd, IOC_PR_RESERVE, &reserve);
  // unregister_key(fd, KEY);

  close(fd);
  return 0;
}
```

キーの登録は、対象デバイスのファイルデスクリプタに**ioctl(IOC_PR_REGISTER)**を発行することで行います(**_register_key**関数)。ここで**struct pr_registration**の**old_key**に0を入れ、**new_key**にキーを入れると新しくキーが登録され、逆に**old_key**を既存のキー、**new_key**を0とするとキーが削除されます。Reserveの登録には**ioctl(IOC_PR_RESERVE)**を使います。**struct pr_reservation**の**key**に対象とするキーを入れ、**type**に獲得したいReserveのtype(ここでは**PR_WRITE_EXCLUSIVE**)を入れます。このように**ioctl(IOC_PR_*)**を用いることで、デバイスの種類に依存せずにPersistent Reserveの機能を使うことができます。SD

June 2017

NO.68

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
榎 真治　ENOKI Shinji　enoki-s@imail.plala.or.jp

Unix系コミュニティ運営の光と影

今回は3月に東京で行った研究会について報告します。

jus研究会 東京大会

■ITコミュニティの運営を考える
【講師】蛯原 純
（日本NetBSDユーザーグループ、㈱創夢）
羽鳥 健太郎（小江戸らぐ）
上田 隆一（USP友の会、千葉工業大学）
榎 真治（日本UNIXユーザ会、LibreOffice日本語チーム）
【日時】2017年2月11日（土）10:00～18:00
【会場】明星大学 日野キャンパス 26号館

■はじめに

jusでは各地で「ITコミュニティの運営を考える」をテーマにパネルディスカッションを行っています。jus研究会東京大会では、日本NetBSDユーザーグループの蛯原さん、小江戸らぐの羽鳥さん、USP友の会の上田さんをお迎えして行いました（**写真1**）。参加者は25名でした。

日本NetBSDユーザーグループ（以下、JNUG）は、NetBSDにかかわる人の集まりで、古くから活動されており、各地でのオープンソースカンファレンス（以下、OSC）にも最も積極的に参加されているコミュニティです。小江戸らぐは、埼玉県川越市を活動の中心としたLinuxの地域ユーザーズグループで、気軽に相談できるゆるい場の提供を目的に、毎月のオフ会など十数年活動されています。USP友の会は2ヵ月に1回、シェルのワンライナーの使い方を学ぶシェル芸勉強会を開催され、参加者全員が各自でお題を解いていきます。大阪や福岡でのサテライト会場もあります。

■場を荒らす参加者をどうするか

はじめに上田さんから「コミュニティの場を荒らす参加者をどうするか？」というお題が出されました。小江戸らぐでは、今までの活動の中で一度だけ、どうしても信頼関係を築けない方がいたそうですが、そのときは来ないでくださいとはっきりお断りされたとのことです。訴えるとまで言われたので、法律無料相談にも行かれたそうです。まずいなというときは自分を信じて徹底的にやるのが良い、というのが教訓とのことでした。

USP友の会では、懇親会の場で、一方的に話をされて噛み合わず困ったことがあったそうです。羽

写真1　講師のみなさん
（左から上田さん、羽鳥さん、筆者（榎）、蛯原さん）

鳥さんからは、発表時なら時間だからと区切る、「そこまで語れるなら本にしてみようよ」あるいは「ほかのお題もやってみない？」と提案してみてはどうか、というコメントがありました。上田さんが以前トラブルになったケースでは、良い悪いを議論するのではなく「僕が一緒にやれないのでごめんなさい」という言い方をしたことはあったそうです。

さらに、USP友の会で困ったパターンとして、セクハラもあったそうです。蛯原さんからは「コミュニティが男性だけ、あるいは女性だけになってしまっていて、違う性の人が集まっているのはすごい」というコメントがあり、羽鳥さんからの「インドネシアのLinuxコミュニティが婚活の場になっている」という話に展開しました。イスラム教国であり男女の出会いが少ないという事情もあるそうですが、女性が参加したときに「次に参加するときに友達を1人連れてきてください」とお願いして地道に変わっていったそうです。

■コミュニティをどうやって次の世代に渡していくか

続いて羽鳥さんから「コミュニティをどうやって次の世代に渡していくか？」というお題が出されました。これまでは信頼関係を築くことを重視しつつやってきて、新しい情報を知りたいという気持ちはあるけれど、年をとって疎くなっていく面も感じられているそうです。次の展開としてコワーキングスペースをやってみたいという思いもあるそうです。

蛯原さんからは「やる人は勝手にやるのではないか？」というコメントがありました。JNUGでは、コードなど成果物でしかわからない部分があること、コミュニティ活動に疲れてしまった人が多いことから、あえて何も考えていないそうです。「Unixとしてどこまでやれるのかを追求し、Unixのやることがなくなったら、コミュニティも消滅したらいい」とのことです。

USP友の会では余計な資産を解体して、運営を軽くしようとされているそうです。メーリングリストを廃止してFacebookへの移行を検討していたり、勉強会も2名で運営できるように懇親会をビアバッシュにして事前の人数予測を不要にしたり、定員をあえて40人に絞ったりといった具合です。サテライト会場も任せきりで口出ししないそうです。

■運用しやすい人数は

3つめとして、蛯原さんから「運用しやすい人数はどれくらいか？」というお題がありました。「60人は多くて10人は少ないのではないか？」「いい数字をキープしているときれいに回るのではないか？」というテーマです。小江戸らぐではオフ会の参加者は20人前後で、すごく回しやすいそうです。コミュニティのことに詳しい人ばかりでもなく、お酒を飲みに来る人もいて、興味はバラバラだということでした。

LibreOfficeの関西勉強会は十数名程度であり、ゆるくやりやすい人数です。JNUGの毎月の集まりでは数名程度で、年末は少し増えるそうです。小江戸らぐも初期のころはそれくらいの人数だったそうですが、オフ会での発表をまとめて同人誌を作り、年2回のコミケや各地のOSCで販売していると、新しい人も来てくれるようになったとのことです。USP友の会は人の入れ替わりが激しく、常連さんでも一時来なくなって、また2～3年したら来るようになることもあるそうです。これは小江戸らぐでも同様だそうです。

またUSP友の会ではがんばる要素を消しているとのことです、羽鳥さんは「がんばらないことには本当に同意で、楽しむ、おもしろがっていきたい」とコメントがありました。JNUGでは、息を吸うのと同じように活動されているそうです。

■終わりに

今回は、特色がそれぞれ際立っている3つのUnix系コミュニティの方に出演いただきました。話がさまざまな方向に展開して、どうなることかと思いましたが、コミュニティ運営についてそれぞれの苦労されているところや工夫について、少し突っ込んだお話を伺うことができました。SD

Hack For Japan

エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha

第66回

CIVIC TECH FORUM 2017で社会の問題とテクノロジをつなげる

読者のみなさんはシビックテック（Civic Tech）という言葉をご存じでしょうか？　近年、注目を集めているシビックテックについて議論するイベント「CIVIC TECH FORUM 2017」のレポートを通じて、その言葉の意味とともに新しいテクノロジとデザインが適用されていく世界をHack For Japanスタッフの鎌田がお伝えしたいと思います。

シビックテックとは

シビックテックを語るうえで大きく分けて3つの立場があります。1つは税金をもとに社会にサービスを提供する「行政」、2つめは資本主義にのっとり社会にサービスを提供する「民間企業」、最後はそのどちらでもない非営利の市民コミュニティである「市民」の3つです。社会が抱える課題は広範におよびますが、その影響を受けるのは3つのどのような立場にあっても、最終的には私たち「市民」にほかなりません。そうした構造からそれぞれの立場を超えて力を合わせ、市民としてテクノロジやデザインを用い、社会的な課題を効率よく、安価に解決していくムーブメントをシビックテックと呼んでいます。東日本大震災以降、復興支援の取り組みの影響を受けてか、徐々にこのムーブメントも盛り上がりを見せています。

今回レポートする「CIVIC TECH FORUM 2017[注1]」も2015年から数えて3回目となります。参加人数も増え、先着200名のイベント申し込みも、あっという間に満員御礼となり、注目度の高さがうかがえます。

CIVIC TECH FORUM 2017の構成

CIVIC TECH FORUM 2017は2017年3月25日の朝から、2017年2月24日にオープンしたばかりのNagatacho GRIDでの開催です。基調講演やパネルディスカッションを中心とした「メインステージ」と、シビックテックに関するテーマで来場者とディスカッションする「インタラクティブステージ」の2本立ての構成で実施されました。

ここで発表されたものの中から、本誌をご覧の開発者のみなさんも興味をもたれそうなものをいくつかピックアップして紹介します。

社会の課題とテクノロジのギャップ

最初に紹介する講演はメインステージから、総務大臣補佐官の太田直樹さんとリクルートメディアテクノロジーラボ室長の麻生要一さんによる「社会の課題とテクノロジーのギャップ――テクノロジーは課題を解決できるのか」というテーマの対談です（**写真1**）。

総務大臣補佐官として地方創生、ICT/IoTの政策立案・実行を補佐する太田さんは、IoTやAIなどのテクノロジの力で劇的に変化していく世の中に対して、その未来を待つ姿勢よりも、自ら作りにいった

◆写真1　総務大臣補佐官の太田さんとリクルートの麻生さん

注1　http://civictechforum.jp/

Column

Hack For Japan連載の意義について

2011年3月11日の東日本大震災発生のあとにHack For Japanは発足し、この連載記事は2011年の11月号から続けさせていただいています。今号で66回目となり、復興支援のためのハッカソンなど、エンジニアの“何か役に立ちたい”という想いから始まった活動についてこれまでレポートしてきました(hack4.jpのサイトには過去記事のアーカイブもあります)。

あれから6年が経過し、復興が進んでいる地域もある一方、まだこれからという地域もあります。私たちの活動も変化して、直接Hack For Japanとして主催するイベントの数は減りましたが、今後発生し得る災害を考えた場合、エンジニアがつながり続けるコミュニティは必要と考えて存続しています。また、スタッフは復興支援にとどまらず、今号でも取り上げているCivic Tech、防災／減災、東北でのプログラミング教育などさまざまな活動に関わっています。2017年度も引き続きそういった「エンジニアができる社会貢献」をテーマにした記事をお届けしていきますので、どうぞよろしくお願いします。

(Hack For Japan　高橋憲一)

```
While (Japan.recovering) {
        we.hack();
}
```

ほうが確実性があると主張。そうしたテクノロジの背景にあるデータの取得に関して、日本の人々はマイナスな反応を示しやすい風潮を課題として、丁寧に進めないとテクノロジの導入自体が諸外国と比較して難しくなるだろうと見立てました。また、神戸市で市民参加型のお子さん見守りサービスの成功事例を引き合いに出し、参加者が自らデータの提供、利用を意識的にできる設計の大切さを説きました。

リクルートの麻生さんは、高知県や長野県塩尻市などの行政と包括提携を行い、リクルートの事業開発部門を行政の中に置いている事例と、過疎地での自動車相乗りサービス「あいあい自動車」が乗り越えた地域の課題を紹介してくれました。どんなに優れたテクノロジであっても、主体は地域に住んでいる人に軸がないと地域課題は解決できないとし、高齢化が進む過疎地では高齢者のリテラシーでも操作が可能なくらいまでシンプルでわかりやすいインターフェースを用意しないと、優れたサービスも利用につながらないという実体験から得た知見を語ってくれました。また、地域に寄り添った活動にするため、急がずに時間をかけていく姿勢も重要だとされました。

お二人の話は、システムを開発する我々のような立場の人間にも非常に参考になるお話でした。

OSSに学ぶコミュニティ運営

続いてもメインステージから、Code for Kanazawaの代表を務める福島健一郎さん、日本UNIXユーザ会幹事の法林浩之さん、国立研究開発法人情報通信研究機構 研究員の傍ら、さまざまなコミュニティの立ち上げ、運営に関わる湯村翼さんら3人による「社会インフラとなったオープンソースコミュニティに学ぶコミュニティ運営のコツ」と題した講演からの報告です(**写真2**)。オープンソースが成功したコミュニティ運営の事例を参考に、まだまだ課題のあるシビックテックコミュニティの運営に活かすことを目的とした対談です。

最初にシビックテックコミュニティを運営する福島さんから、このセッションの2つの目的が紹介されました。1つは「オープンソースコミュニティがどうして成功できたのか、そこからシビックテックコミュニティの未来を考えられないか？」という点と、2つめは「現在のOSSがインターネットを支える基盤ソフトを作り上げているように、シビックテックコミュニティが作るソフトウェアが将来の社会を支えるという可能性もあるのではないか？」というものです。

その目的を受けて、長年、日本UNIXユーザ会を運営している法林さんは現在のシビックテックの様子が、以前の、Linuxが誕生し、それに合わせてたくさんのLinuxコミュニティが誕生し、そしてなくなっていった様子に似ているとおっしゃいました。なぜ、たくさんあったLinuxコミュニティがなくなったかというその理由として、普及したことによってコミュニティが減ったことを挙げられまし

た。法林さんによれば、参加者自体が少ないときほどコミュニティが誕生し、普及が進み、一般的になればなるほど、その段階でコミュニティは役割を終えていく印象があるそうです。「そういう意味で言うとシビックテックもこれからかもしれない。Linuxも15年かかった」という言葉でまとめられました。

また、シビックテックのような活動は運営にかかる活動資金の面やメンバーの流動性が低いところに課題があるとする福島さんからの問いかけに対し、さまざまなコミュニティ運営に携わる湯村さんは、まずコミュニティ運営にかかる活動資金の準備については大きく分けて2種類あるとしました。1つはスポンサーを募って運営するもの、もう1つはコミュニティメンバーから少額の集金を行い、懇親会などの必要十分な金額を集め運営にあてるものです。前者はスポンサーを募る分、スポンサーの意向をコミュニティの運営に反映させる要素が出てくるため、コミュニティの自由な活動には制約が出てきます。後者については、運営で必要な最低限の金額を自分たちで出すため、非常に自由な活動ができるコミュニティになります。湯村さんはコミュニティの運営にかかる活動資金面では、そのコミュニティの方向性に合わせて、どちらかを上手に選択するのが大切だとしました。

もう1つの、メンバーの流動性が低いという課題については、オープンソースのほうも高齢化しているところは増えているので難しいところもあるが、うまくいっているコミュニティは参加者自体の数が多い場合と、運営側が若い人材に権限を移譲していく場合の2つがあるというお話でした。

福島さんの、どうしてオープンソースコミュニティは成果を出し続けられるのか？といった問いかけに対しては、湯村さんは成果を出し続けられているように見えるかもしれないが、その背後にはたくさんの失敗があるとし、たくさんコミュニティが誕生することが大切と説きました。対して、法林さんはスキルややる気がある人を入りやすくするのと、自由な競争があるほうが、健全に成長して、成果につながっていくことから、シビックテックコミュニティも競争があったほうが良いのではないだろうかという提言をされました。

◆写真2　左から福島さん、法林さん、湯村さん

最後にコミュニティ運営で活動するお二人からシビックテックに対するエールとして、メッセージが送られました。まず、法林さんからはオープンソースがインターネットの力によって場所に縛られない活動を行えたことが成功の一因だったことを挙げ、ふるさと納税のように地域に縛られないシビックテック活動を模索するのはどうかという提言がありました。湯村さんからはまだ知名度が足りていないシビックテックという活動自体をより多くの人に知ってもらうために、身近な人たちに知ってもらう会話などから始めるのが良いのではないかといったメッセージが述べられて、このセッションは締めくくられました。

本当のオープンデータの話をしよう

来場者とのディスカッションを行うインタラクティブステージから、東京大学空間情報科学研究センター特任講師である瀬戸寿一さんと、モデレーターとして再びCode for Kanazawaの福島さんによる、オープンデータをテーマとした「本当のオープンデータの話をしよう」と題するディスカッションの報告です。まず最初に瀬戸さんからのインプットとして、国内外のオープンデータ事情を紹介いただき、適宜インタラクティブな質疑応答を挟みながらこのセッションは進みました（**写真3**）。

あらためて、オープンデータの定義が確認され「目的を問わず、誰でもどこでも自由に利用し共有

できるデータ」という共通認識を会場にいる参加者全体で持つことで、瀬戸さんより紹介される事例に耳を傾けることができました。

最初に紹介されたのは国内のオープンデータを公開する自治体の数からですが、県単位では47都道府県中、34県がオープンデータを公開しています。こう聞くと日本の多くの自治体が公開しているかのように見えますが、これを市町村単位にしてみると1,718市町村の中で、233市町村がオープンデータを公開しています。だいぶ目減りした印象を受けます。さらに国外のオープンデータ件数と比較してみると、日本全体で公開されている数はおよそ2万弱となっていますが、米国ではニューヨーク州だけでもおよそ1万ものオープンデータが公開されているという例が示されました。日本全体と米国の1州で比較できる現状を見ると、オープンデータの取り組みがいかに日本の自治体で進んでいないかがわかります。

我々、Hack For Japanも東日本大震災をきっかけに行政が公開するデータを見てきましたが、PDFやJPEGといった再利用しにくい形式で公開されているために、復興支援、防災、減災の取り組みで用いようとすると大きなハードルがあったことを度々紹介してきました。もちろん、東日本大震災のころと比較するとオープンデータの取り組みは非常に進歩しました。しかし、まだまだ国外の状況と比較すると日本は遅れているといった実態が見えてきました。

一方で、国内の成功事例もいくつか紹介されました。1つは東日本大震災の影響を受けた福島原発に端を発した、放射線問題への対応として誕生したSafecastの活動です。取得した放射線データをオープンにしたほか、ガイガーカウンターの仕様もオープンにしたことで、比較的手軽にガイガーカウンターの準備、設置、計測が行えるようになりました。これによって収集・公開されるデータが増え、放射線の状況を可視化できました。ほかには、市民が行政と共同で街の課題をレポートし、その改善を行っていく「ちばレポ」の取り組みなどが紹介されました。オープンデータを軸とした成功事例は国内でも目に入るようになってきています。OpenGLAMのように図書館、博物館、美術館などもデータのオープン化がされてきている昨今のトレンドなどもふまえ、議論は熱を帯びていきました。

◆写真3　講演する瀬戸さんと会場の様子

そんな中、データの側面から見た多様な参加の可能性として、瀬戸さんよりオープンデータの類型を紹介いただきました。まずは行政が持つ公共データの流通活用を促すトップダウン型、もう1つが市民によるデータ収集への貢献に支えられるボトムアップ型、そして、新しい参加の形として、行政だけでなく市民も協力してデータを収集し、フィードバックしながら進めるミドルアップ&ダウン型といったオープンデータの活動に関する参加の形です。多様な参加方法でデータを使いたい、作りたいと思える社会「参加型データ社会」の重要性をうったえられました。

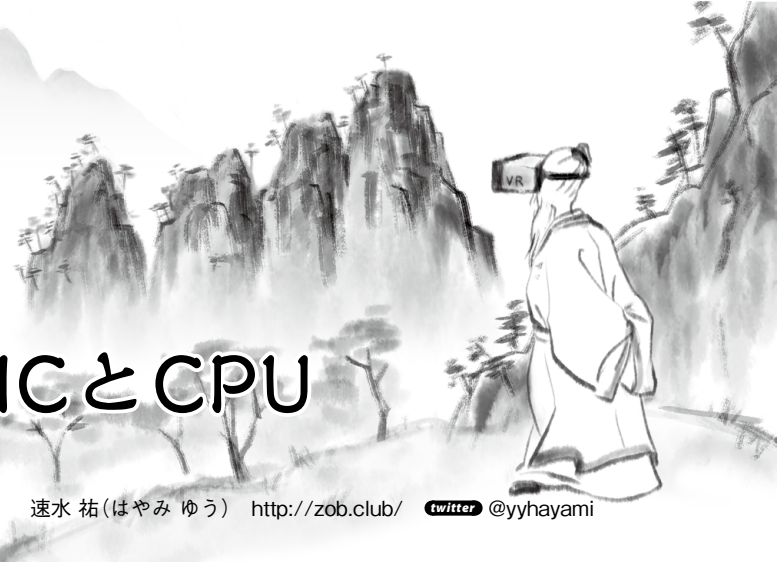
シビックテックの今

さて、CIVIC TECH FORUM 2017のセッションの中から、Hack For Japanのこれまでの活動と近いものや読者のみなさんが興味を持たれそうなものをいくつかピックアップして紹介しました。しかし、このほかにも興味深いシビックテックにまつわるセッションは数多くありました。東日本大震災以降、テクノロジとデザインを活用して、社会課題を解決していこうとする取り組みは急増しています。読者のみなさんも「市民」の一人として、こうしたカンファレンスの参加から社会課題の解決に貢献してみてはいかがでしょうか。SD

温故知新
IT むかしばなし

第66回

汎用ロジックICとCPU



速水 祐（はやみ ゆう） http://zob.club/ twitter @yyhayami

はじめに

1970年代の8bitマイクロプロセッサが登場する前の時代では、汎用大型コンピュータを小さくまとめたミニコンが広く使われていました。ミニコンの代表となるDEC社の16bitのミニコンである「PDP-11」の上では、初期のUNIXが開発され、研究者の間で利用され始めていました。マイクロプロセッサがない時期に、このミニコンのCPUボードは、汎用ロジックICを数多く組み合わせることでそれを実装していたのです。今回は、この汎用ロジックICの話をしましょう。

汎用ロジックIC

汎用ロジックICは、半導体を用いて論理回路を構成する集積回路（IC）であり、SoC[注1]の普及などにより使用頻度は減りましたが、現在でも多くの電子回路で利用されている重要なICです。

1966年、Texas Instruments社からバイポーラ（2極式）汎用ロジックICの74シリーズが発売され、このICファミリーは、複数の他社からのセカンドソースとして、この体系に従い供給が行われ業界の標準になりました。

74シリーズは、あとに続く数字で機能を定義し、基本となる7400のNANDゲートから始まり**表1**のようにさまざまな論理回路を実現しています。

当初は、+5V単一電源のバイポーラトランジスタ[注2]のTTL[注3]であり、消費電力が大きく大規模な回路をTTLで組むとかなりの電流が流れる回路となってしまいます。そこで、消費電力を小さくした74Lシリーズや、より高速化を目指した74Sシリーズが登場します。

1970年代になると消費電力を低減し、高速化も実現した74LSシリーズが発売され、その使いやすさから広く使われるとともに74に続く数字も増え、多種多様のロジックICが現れました。74LSシリーズのDIPパッケージ[注4]は、現在でも電子部品として見かけることができます。1980年代中盤には互換性を確保したC-MOS版である74HCシリーズが登場し、より低消費電力となり電源の範囲も広がり、出力電流も確保しやすく非常に使いやすいロジックICとなりました。74HCシリーズは30年を経過した現在でも電子工作や教育用途として使われています。

CPUを構成するロジックIC

1980年前半の黎明期のパソコンでは、現在のSoCと違い8bit CPUの周辺に数々のデバイスを配置しなければならず、それらと接続するためにAND/ORなどの論理ゲートではない別のロジックICが数多く使われていました。

マイコンのアドレスバスには、データの安定化のためのバッファ74LS244、データバスには双方向データためのバストランシーバ74LS245がメモリや周辺デバイスの間に必要でした。

80年代中盤になると8bitパソ

▼表1　74シリーズの代表的な論理回路

型番	論理	回路
7400	NANDゲート	4回路2入力
7402	NORゲート	4回路2入力
7404	NOTゲート	6回路
7408	ANDゲート	4回路2入力
7432	ORゲート	4回路2入力
7486	XORゲート	4回路2入力
7474	Dフリップフロップ	2回路

注1）System-on-a-chip

注2）PNPまたはNPNの半導体接合構造で、スイッチングの機能を持つトランジスタ。

注3）Transistor-Transistor-Logic

注4）Dual Inline Package

コンのCPUと周辺デバイス間のロジックICが使われている部分はASIC[注5]に替わり、パソコン内部のICは大幅に減り、機能の割にはシンプルな構成になっていきました。

マイコンが使われる前のCPUボードは、それ自体がロジックICで構成されていました。筆者は、現在、情報教育用のCPUをロジックICを使って設計・製作しています。その設計の過程で、いかに少なく効果的に使えるロジックICを選択して組み合わせるかが重要なテーマとなっています。

1970年代のミニコンのCPUでも同様だったと思われ、1枚のCPU基板[注6]にさまざまなロジックICが美しく整列しています(**写真1**)。CPUボードでは、プログラムが格納されているアドレスを増加させるためのカウンタ74161やデータを一時的に格納するレジスタ74273などが使われています。さらに計算を行う、まさに中央演算装置の核となるALU[注7] 74181が必要になります。**写真1**はPDP-11のCPU基板の一部です。

ALU 74181

ALU 74181は、非常に多機能なロジックICであり、4bitの2つのデータを16種類の論理演算か同じく16種類の算術演算を実行して4bit結果を出力します。論理・演算の32種類を選択するための5bit信号入力があり、全24ピンの大きめのロジックIC(**写真2**)です。これだけ多機能のICですが、その内部構成は職人技で最低限のトランジスタでその機能を実現しています。74181はミニコンのCPUでは必須のICでしたが、現在生産が終了しており、手に入れることは困難を伴います。秋葉原や日本橋を探しても見つけることはほぼ不可能であり、オークションサイトでもたまに74LS181が出品される程度です。筆者は、10年ほど前に1日かけて秋葉原で探しまわり、奇跡的に74HC181を発見し、数個の在庫をすべて購入したことがありました。

生産が中止されたロジックIC

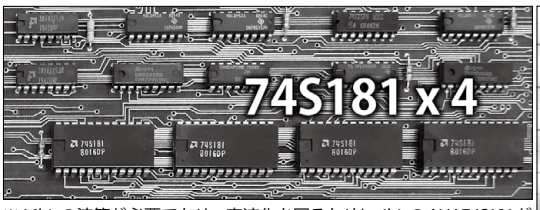
1970年代には、CPUを始めさまざまな回路がロジックICの組み合わせで実現され、多種類ロジックICが製造されましたが、現在では生産中止になっていて、ALU 74181以外にも購入が難しい型番の74シリーズが多くあります。筆者の設計しているCPUでは、16個の8bitインデックスレジスタを使います。これを実現するためには通常なら8bitレジスタICを16個並べなければなりませんが、ロジックデバイス規格表にも載っていない4bit×16のレジスタを実現する74LS219を見つけました。74LS219が2つあれば16個の8bitインデックスレジスタを実装できるのです。しかし、これが74181以上に入手が困難です。海外のサイトをくまなく検索しても、過去に存在したことはわかるのですが、その現物を手に入れることは不可能です。それでも、探し続けていると、なんと近所の古くからある電子部品屋で発見できたのです。

現在のロジックIC

現在でもSOP/SSOPなどのフラットパッケージで広く使われています。東芝が生産しているLCX/LVX/VCXなどの電源電圧が幅広く、低消費電力のさまざまなシリーズも存在します。しかし、74のあとに続く型番の種類はあまり残っていません。昔のように電子部品屋を覗(のぞ)くと、キラリと光る宝石のようなロジックICが見つかるかもしれません。

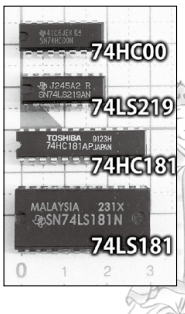
SD

▼写真1　PDP-11のCPU基板の一部



※16bitの演算が必要であり、高速化を図るために4bitのALU 74S181が4つ並んでいる

▼写真2　ALU 74181とほかのロジックICの大きさ比較↓



※この写真は、UECコミュニケーションミュージアムの協力を得て撮影しました

注5) Application Specific Integrated Circuit：特殊用途向け集積回路

注6) マイコンが載った基板ではなく、CPU自体を構成する基板。

注7) Arithmetic Logic Unit：算術論理演算装置

うまくいく チーム開発のツール戦略

第8回 ドキュメントにかかわる時間を効率化すれば生産性はもっと上がる!

Author リックソフト㈱ 青地 芳彦

企業のコミュニケーションは変革を迫られている

IT技術の進歩によって、コミュニケーションの手段が急速に変わってきています。SNSは、友人、知人との間だけでなく、企業内でのコミュニケーションや、企業と個人、たとえば人材採用の場面でも、メールに代わる手段として使われ始めています。企業の情報発信の場も、Webに代わってFacebookやInstagramに作られるケースが増えてきました。また、クラウド化するファイル共有サービスは、個人のコンテンツを保存するだけでなく、業務で手軽にファイルを共有する手段としても使われています。

こうしたコミュニケーションやデータのあり方を変えるデジタルトランスフォーメーションの流れは、書店業界にAmazonが、旅行業界にAirbnbが進出してきたように、ビジネスの様相を大きく変えてしまいます。その潮流にいち早く気づき、適応していかなければ他社との競争に勝つことはできません。ビジネスの変化のスピードに後れをとることのないよう、今こそ業務を見なおし、改革をすべきではないでしょうか?

IDCの調査によると、日本のビジネスワーカーは、メールなどのドキュメントに関するさまざまな課題の対処に多くの時間を費やしている反面、オフィス生産性ツールに依存する割合はほかの国よりも低いという結果が出ています[注1]。

本稿で紹介するコンテンツ管理システム「Alfresco」のユーザ会では、多くのユーザがメールの多さを指摘しており、あるユーザは1日に受ける量のうち、8割が社内のメールだったりするようです。また、いまだに承認、決済をハンコで行っている会社が多く、しかもそうした決済のフォローアップもメールだったりするので、さらにメールが増えるという悩みが聞かれました。

今回はチーム開発の話とは少し違いますが、より大きな視点で業務改善につながる、コンテンツ管理についてのツール戦略を紹介します。

仕事のスタイルを変えるAlfresco

日常業務を効率化し、検索などのドキュメントにかかる時間からビジネスワーカーを解放するのが、今回紹介する「Alfresco」という製品です。Alfrescoは、コミュニケーションとコラボレーションをオープンソースの技術によって改革し、ビジネスを有利に進めるための手段を提供します。Alfrescoには、Alfresco Content ServicesというECM(Enterprise Content Management)製品と、Alfresco Process ServicesというBPM(Business Process Management)製品があり、これらを統合した新しいビジネスプラットフォームとしてAlfresco Digital Business Platformがリリースされています。タスクが中心のプロセスにも、ドキュメントが中心のプロセスにも対応し、セキュリティやコンプライアンスを強化しながら、迅速にシステムを構築できる可用性

注1) 「日本国内のインフォメーションワーカーが抱える生産性ギャップを埋める:IT部門の新たな課題と機会」(IDC, September 2012)

の高いソリューションです。

Alfrescoには次の3つの特徴があります。

シンプル

Web UIのAlfresco Shareは、シンプルなデザインながらAlfrescoの機能を十分に利用でき、ユーザは直感的に操作できます。ブラウジングしながらコンテンツを見つけられるようにするため、サムネイル表示をはじめ、さまざまな表示方法でブラウジングをサポートします(**図1**)。

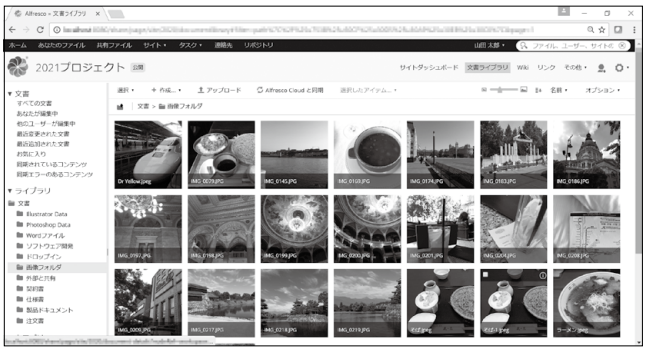
また、オーディオ、ビデオを含む、多くのファイルフォーマットのプレビューが可能なので、管理されているコンテンツをローカルのPCなどにダウンロードせずにブラウザ内で確認できます(**図2**)。

Alfrescoはブラウザでのアクセスのほかに、CIFS、WebDAV、IMAPを通して利用できるので、Windowsのエクスプローラや Macの Finderからも利用できますし、メールクライアントからAlfrescoをブラウジングすることもできます。また、オプションとしてAlfresco Content Connector for MS Outlookを利用することで、Outlookから Alfrescoを操作し、ブラウジングだけでなく、検索やプレビューを行ったり、メールをアーカイブするといったこともできます。ユーザの用途に応じてさまざまなアクセス方法を選択できます。

Alfrescoでは、ビジネスに関わるさまざまなユーザが、いつでも、どこからでも、多様な手段によって最新の情報を共有できます。業界唯一の Cloud Sync は、Alfresco in the cloudとオンプレミスのAlfresco Content Servicesの間で同期を行うことで、すべてのユーザが最新のバージョンをいつでも共有できます。

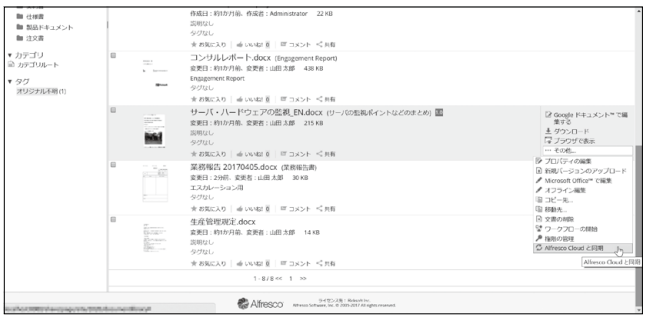
管理者は、外部のユーザとどのようなコンテンツを共有しているのかを一目で把握できるので、クラウド系のストレージサービスに比べてセキュリティに対するガバナンスを高めることができます。また、オンプレミスとクラウドの間はワークフローを利用できるので、コンテンツを共有しながら、外部も巻き込んだプロセス(レビューや承認など)のステータスを把握できます(**図3、4**)。

▼図1 ギャラリービュー

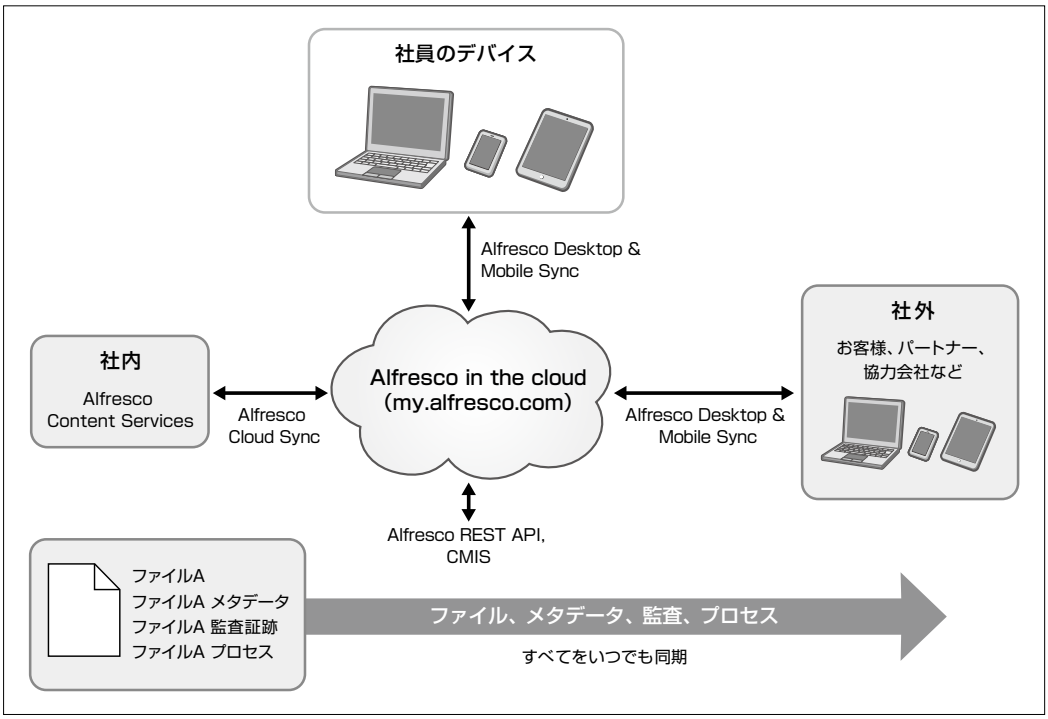


▼図2 ドキュメントのプレビュー

▼図3 Cloud Sync機能



▼図4　Cloud Syncを使った社内／社外の共有のイメージ



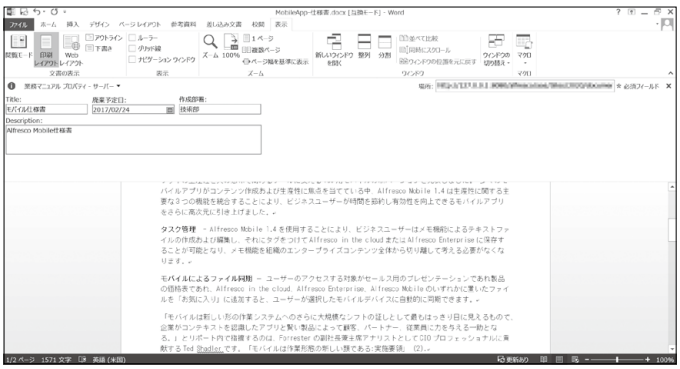
スマート

コンテンツの編集においても、余計な手間をかけることなく編集作業に専念できる工夫がなされています。Alfrescoでは、AOS(Advanced Office Services)を実装しており、Officeドキュメントを直接開いて編集できます。編集をしている間、ドキュメントは自動的にロックされ、ほかのユーザが編集や更新することをブロックします。また、ドキュメントにメタデータ(プロパティ)がある場合にも、Office内で確認や編集ができるので、メタデータも一貫してOfficeアプリケーション内で編集／利用することができます(**図5**)。

ドキュメントを再利用するうえで重要な検索についても、目的のドキュメントをすぐに見つけられる機能が備わっています。ライブサーチ機能では、検索窓にキーワードを入力すると、該当するドキュメントがすぐに一覧表示されます。さらに、検索結果をさまざまなプロパティ条件でフィルタリングすることで、見つけたいドキュメントを絞り込めます(**図6**)。

コンテンツプラットフォームを十分に活用したスマートフォルダ機能は、物理的なフォルダ構造に依存しない形でコンテンツを利用ユーザが再構成できます。たとえば、技術者側にある

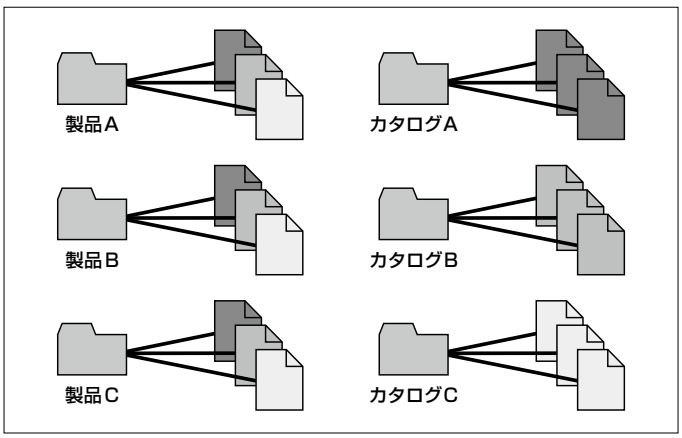
▼図5　AOSでのプロパティ編集



コンテンツが製品ごとにそれぞれフォルダに分類されていて、その中でそれぞれの用途別のドキュメントがサブフォルダになっているようなとき、マーケティングや営業側にいるユーザにとっては、その資料がリリースノートやデータシートごとに再分類されているほうが便利な場合があります。このようなケースでスマートフォルダを利用すると、実際の物理的なフォルダ構造に依存せずに、さまざまな切り口の仮想的なフォルダ構造でブラウジングができます。これは再利用性を向上させる大きな機能です(**図7**)。

▼図6　検索結果のフィルタリングとハイライト(囲み個所)

▼図7　スマートフォルダのイメージ図(同じ模様のドキュメントはそれぞれ同一のドキュメントを指す)



オープン

オープンな技術を利用した、オープンソースのAlfrescoは、外部との接続に対してもCMISやFTPなどの標準的なプロトコルを利用できるので、自社のアプリケーションや外部システムとの連携も容易です。また、Alfrescoのコア機能の分離を徹底したREST API化と、モジュール化により、ユーザは必要な機能だけを使って外部システムと連携したり、Angular.jsをベースにしたAlfresco Development Frameworkを利用して、業務に適したユーザアプリケーションを短時間に作成することができます。こうしたモジュール化がもたらす恩恵は、カスタマイズやインテグレーションだけでなく、システムのスケーラビリティにも及び、12億ドキュメントの管理をAWS上で実証しています[注2]。

注2) URL https://www.alfresco.com/blogs/how-alfresco-powered-a-1-2-billion-document-deployment-on-amazon-web-services/

最後に

シンプル、スマート、オープンなAlfrescoは業務の効率化をもたらし、ビジネスワーカーの時間の使い方を変える大きな力となります。Alfrescoをこの機会にぜひお試しください。Alfresco製品は、Alfresco社のサイトからダウンロードして評価いただくことができます。また、製品に関する質問、問い合わせはリックソフトを含むAlfresco社のパートナーからできますので、お気軽にご相談ください。SD

※本連載の過去記事は技術評論社のWebサイト「gihyo.jp」でもご覧になれます。
http://gihyo.jp/ad/01/atlassian

Special Interview

3年で300%急成長の

本誌で好評連載中の「うまくいくチーム開発のツール戦略」でお馴染みのリックソフト㈱が業務拡大につき各種エンジニアを募集中です。世界40ヵ国54,000社が導入しているAtlassian製品はご存じの読者も多いことでしょう。リックソフトはAPAC（アジア太平洋地域）でナンバー1の販売実績を誇ります。急成長を遂げる同社にエンジニアとして転職した大野智之氏（**写真**）に、リックソフトの魅力を伺いました。（編集部）

リックソフト㈱
開発部 RickCloud Team
大野 智之氏

エンジニアとして挑戦するために転職を決意

――リックソフトに入社する前に勤めていた企業での業務内容を教えてください。

大野氏　茨城県つくば市にある企業でSEとして働いていました。おもに担当していたのはWebやインフラの領域で、自社サービスの構築やサーバの運用管理といった業務になります。受託開発もあり、近隣の自治体や研究機関のシステムの構築や運用保守も行っていました。

――転職した背景には、どのような理由があったのでしょうか。

大野氏　以前の職場では、私以外のエンジニアはWindowsのデスクトップアプリケーションをメインで開発していて、Webやインフラといった領域の経験は豊富とは言えませんでした。そのため、システムやインフラを構築する際は、それらの領域のスキルがなくても運用できるように設計していたのですが、一方で新しい技術やプロダクトにチャレンジできないという不満がありました。たとえばWeb開発でPHPのフレームワークを選択する際、自分が使いたいものを優先するのではなく、ほかの人がメンテナンスすることも考えて一般的に使われているものを選択するといった形です。そういった部分でジレンマを感じていたため、最終的に転職を決めました。

Webやインフラに精通したエンジニアと切磋琢磨できる環境

――転職先としてリックソフトを選んだ理由を教えてください。

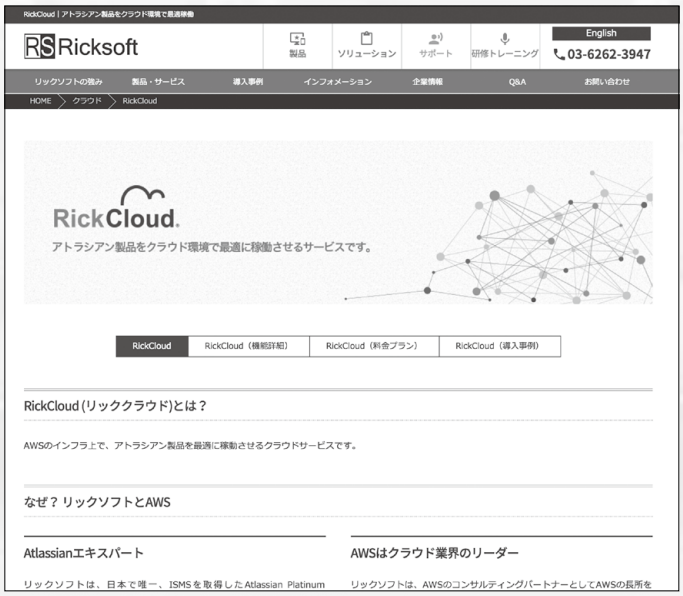
大野氏　リックソフトでは自社で「リッククラウド」というクラウドサービスを提供しています（**図1**）。これは「JIRA」や「Confluence」「Bitbucket」といったAtlassianのプロダクトを提供する、マネージド型のクラウドサービスです。こうしたサービスであれば、Webやインフラに関するこれまでの業務経験を活かせるのではないかと考えたのが大きな理由です。転職活動を始めるまでは、リックソフトについてまったく知りませんでした。調べてみるとAtlassianのプロダクトを扱っていて、日本でトップレベルのパートナーだということがわかりました。実はSourceTreeというフリーのGit/Mercurialクライアントツールを以前から利用していたほか、Git環境を構築できるBitbucketを個人的に使っていたので、Atlassianについては知っていたんです。また面接での感触から、社内の雰囲気もいいのだろうと感じたことから、最終的にリックソフトを転職先に選びました。

――実際に入社してみていかがでしたか。

大野氏　リックソフトにはWebやインフラに精通したエンジニアがたくさんいて、前職で抱えていたような制約にとらわれずに新しい技術

ベンチャーがSE・PGを大募集!!

▼図1　リックソフトが自社で展開するクラウドサービス「リッククラウド」



を積極的に採用できるようになりました。また、ほかのエンジニアと意見交換できることは、この会社に入って良かったと感じる部分です。現在はリッククラウドの機能改良を進めるプロジェクトにアサインされていますが、その中でさまざまな技術を使って新しいことに挑戦できているのは楽しいですね。

企業価値、サービスの価値を高められるエンジニアが目標

――*現在の業務内容について教えてください。*

大野氏　現在はリッククラウドの研究開発や構築作業、保守といった業務に携わっています。また、そのほかに社内の情報システムやインフラの管理も行っています。基本的にはインフラ寄りのエンジニアという立ち位置で、若干DevOpsなどといった領域にも手を広げています。

――*リックソフトでどんなことにチャレンジしたいですか。*

大野氏　まずは自社サービスであるリッククラウドをより魅力的なサービスにしていきたいですね。リッククラウドは2017年5月でサービス提供から3年目を迎えます。おかげさまで多くのお客様にご利用いただいていますが、サービスの使い勝手など改善すべき点はまだまだあると感じています。私がユーザであればこういった機能がほしいと思うこともありますし、お客様からのお問い合わせでこういった機能が求められているんだと気づくこともあります。それらを参考にしつつ、お客様にとってより使い勝手の良いサービスとなるように開発を進めていきたいと思います。

――*リックソフトへの転職を考えている方にメッセージをお願いします。*

大野氏　リックソフトはまだ若い会社で、チャレンジすべき領域が社内にたくさん残っています。主体的に仕事に携わりたい人にとっては働きがいのある会社ではないでしょうか。また私も関わることになったリッククラウドも展開しているので、クラウドに深い経験を持っている方やDevOpsに興味があるエンジニア、あるいはそれらの領域でもう一歩踏み出してチャレンジしたいと考えている人にぜひ来ていただきたいですね。

――*最後に、エンジニアとしての目標を伺わせてください。*

大野氏　企業の価値、あるいはサービスの価値を高められるエンジニアになりたいと考えています。それと、さまざまなことに積極的に興味、関心を持っていきたいですね。

――*本日はありがとうございました。*SD

Software

レッドハット、
Ansible Tower 3.1を提供開始

レッドハット㈱は3月9日、構成管理ツールAnsibleのWebユーザインターフェース「Ansible Tower」の最新バージョン「Ansible Tower 3.1」の提供開始を発表した。

Ansible Towerは、AnsibleをWebブラウザからコントロールできるWebユーザインターフェース。Ansibleの実行結果、ユーザの権限／実行履歴などを管理でき、おもに企業での利用が想定されている。最新バージョンでは次のような拡張が行われた。

- マルチプレイブックによるワークフロー：追加のプレイブックを作成することなく、複数の既存Job Template（プレイブックの実行に関わる設定）を連結し単一のJobとして実行可能
- スケールアウトクラスタリング：同時に実行可能な処理数を増強可能
- エンタープライズログ管理プロバイダとの統合：分析およびログ収集機構の一部として、自動処理の結果に対するインデックス付け、監視、および処理が可能
- ユニバーサル検索機能：実行ユーザ、使用されたインベントリごとのJob Templateなど、さまざまな条件でTower全体の検索およびフィルタ設定が可能

CONTACT
レッドハット㈱ URL https://www.redhat.com/ja

Topic

タレスジャパン、
「2017 Data Threat Report 日本エディション」を発表

タレスジャパン㈱は4月19日、データ脅威に関するレポート「2017 Data Threat Report 日本エディション」を発表した。

本レポートは、世界主要国における各業界のシニアエクゼクティブ1,100名（内日本100名）を対象に実施した、データ脅威に関する調査結果をもとにまとめられたもの。レポートのハイライトは次のとおり。

- 日本企業の54%でITセキュリティに対する投資が増加。投資の内訳としては、「コンプライアンス要件への適用」が第1位に（66%）。第2位となったのは「ビジネスパートナーからの要件」（50%）で、第3位となったのは「企業ブランドや企業イメージの保護」（47%）
- 79%の回答者が機密データを新規にマイグレートした環境で使用するとしているが、半数以上（56%）の回答者が新環境の中で機密データが危険な状態で使用されていると考えている
- 保存データのセキュリティ対策は、回答者の63%がデータ保護において最も効果的と評価しているにもかかわらず、ITセキュリティに対する予算の中では最下位に位置

CONTACT
タレスジャパン㈱ URL https://www.thalesgroup.com/ja/

Service

FishMe、
フィッシング対策ソリューションを日本市場で販売開始

FishMe, Inc.は4月12日、日本国内にて同社のフィッシング対策ソリューションをS&J㈱とNRIセキュアテクノロジーズ㈱から販売することを発表した。

FishMeのソリューションは、標的型攻撃やランサムウェアなどのサイバー攻撃に関する従業員教育を促し、従業員自らが不審なメールを報告できるようにするもの。従業員から報告されるメールに対して、社内のセキュリティ担当者が迅速に対応するための製品も含まれる。

- PhishMe Simulator：訓練用に、攻撃手法を再現したメールを送信
- PhishMe Reporter：不審メールを従業員自らが報告するためのツール
- PhishMe Triage：社内のセキュリティ担当者のための分析・ブロックツール
- PhishMe Intelligence：フィッシングに関するさまざまな情報を配信

リリースによると、メールによる攻撃に対して日本では「開かない」対策が一般的だが、欧米では「不審メールを報告し対処する」対策が浸透してきているとのこと。

CONTACT
FishMe, Inc. URL https://phishme.com

Event

「U-22プログラミング・コンテスト2017」、募集開始

「U-22プログラミング・コンテスト2017」の開催が決定、Webサイトがオープンした。

本コンテストは、優れた人材の発掘・育成を目的として開催されている、作品提出型のプログラミングコンテスト。今年のキャッチフレーズは「夢を動かせ。自由な創造力で未来をつかめ！」。昨年に引き続き、「プロダクト」「テクノロジ」「アイデア」の3つの評価ポイントにより審査される。参加資格は日本国内に居住する1995年4月2日以降に生まれた個人やチーム（チーム参加の場合、同じ学校に所属する学生であれば23歳以上のメンバーも参加可）で、募集する作品は、未発表または2016年9月1日以降に発表したオリジナル作品。

また今年から「小学生部門」が設けられ、プログラミングを学び始めた小学生も、コンテストに参加しやすい体制となっている（引き続き、U-22部門に12歳以下の児童が応募することは可能）。

●開催スケジュール

7月3日～8月24日	応募受付
9月上旬～中旬	事前審査、一次審査
10月1日	最終審査会・特別講演・各賞発表

CONTACT
U-22プログラミング・コンテスト2017
URL http://www.u22procon.com

Service

さくらインターネット、「sakura.io」を提供開始

さくらインターネット㈱は4月18日、IoTプラットフォーム「sakura.io」の提供を開始した。

sakura.ioでは、データを送受信するための「さくらの通信モジュール（以下通信モジュール）」「通信環境」「データの保存や連携処理に必要なシステム」を一体で提供するIoTのプラットフォーム。本サービスは2016年から提供されていた「さくらのIoT Platformβ」の正式版で、併せてロゴも刷新された。

▲sakura.ioロゴ

本プラットフォームにおけるIoTデータの流れは、

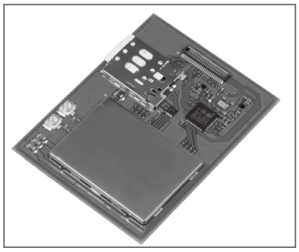
(1) I²C/SPIを使いセンサーから通信モジュールへ
(2) ソフトバンクのLTE閉域網を使い基地局へ
(3) 専用線でさくらインターネットのデータセンターへ
(4) SSL/TLS通信でsakura.ioからJSONデータとして取得

と、非常にセキュアな通信路となっている。また、sakura.ioはIBM Bluemix、myThings、AWS IoTなどのサービスと連携できるようになっており、IoTシステムを簡単に開発できる。

4月18日より販売する通信モジュールは、キャリアのLTE閉域網のみを通って基地局と接続する「単体方式」だが、2.4GHzまたは920MHz（LoRa）の変調方式の通信モジュールから各変調方式に対応するゲートウェイを経由して基地局と接続する「ゲートウェイ方式」も現在開発中とのこと。料金体系は次のとおり（①と②は単体方式のもの。価格はすべて税別）。

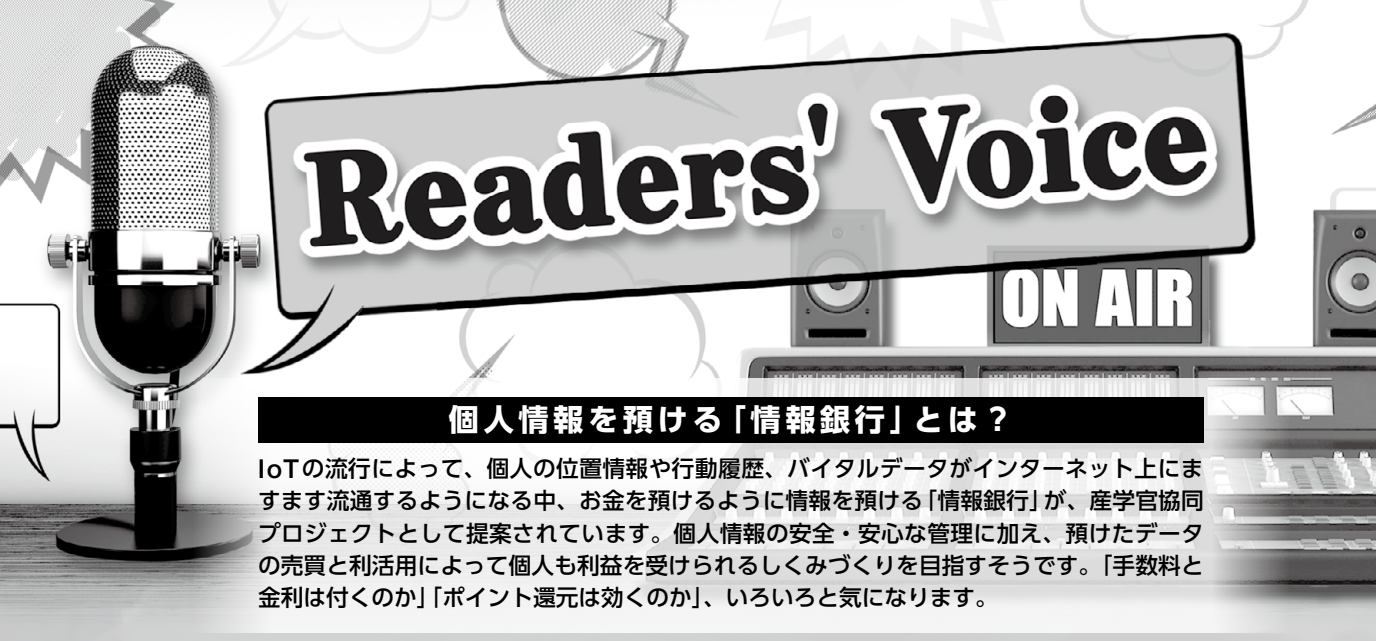
①通信モジュールの販売価格は8,000円／台
②sakura.ioの月額利用料金は通信モジュール1台につき月額60円
③通信モジュールとsakura.io間で通信をする場合に、「sakura.ioポイント」を消費（100回の通信あたり100ポイント消費）。通信モジュールの初回登録時に10,000ポイント／台、またsakura.ioの月額利用料金を支払うと毎月10,000ポイントが付与される。なお、20,000ポイントあたり100円で追加購入も可能

通信モジュールのほかに、検証用のブレイクアウトボード（2,500円／1台）、Arduinoシールド（5,000円／1台）、Raspberry Piシールド（価格未定。6月発売予定）のラインナップがある。加えて「ハードウェアライセンス」「プロトコルライセンス」の販売によって、他企業の自社モジュール／システムの開発もサポートする。



▲㈱さくらの通信モジュール（単体方式）

CONTACT
さくらインターネット㈱ URL https://www.sakura.ad.jp



個人情報を預ける「情報銀行」とは？

IoTの流行によって、個人の位置情報や行動履歴、バイタルデータがインターネット上にますます流通するようになる中、お金を預けるように情報を預ける「情報銀行」が、産学官協同プロジェクトとして提案されています。個人情報の安全・安心な管理に加え、預けたデータの売買と利活用によって個人も利益を受けられるしくみづくりを目指すそうです。「手数料と金利は付くのか」「ポイント還元は効くのか」、いろいろと気になります。

2017年4月号について、たくさんの声が届きました。

第1特集 Linix入門（OS操作編）

新人歓迎企画第1弾。今年入社でLinuxにはじめて触るといった新人を対象に、Linuxの基本知識、CUI端末での操作、Webサーバの立ち上げ方を、先輩が後輩に教える流れで基礎からやさしく解説しました。

新人が自分の下にも来る予定のためタイムリーだった。　須田さん／長野県

新人のときはそうだったなぁ、と思い出されて良い感じでした。知らないあるいは知ったつもりだったことがありました。　ももんがさん／静岡県

自分も今月ちょうど就職したところなので、非常に参考になっています！
若杉達郎さん／東京都

この時期は新人向けの話題になり、毎年初心に帰れて良いですね。
カズさん／千葉県

新人向けかと思ったら、なかなか奥が深かった。　ろくじろうさん／東京都

Linuxサーバの基本を知るために今月号を買いました。専門書になると少し高価なので。初心者向けなので研修を受けている感じで記事を読むことができて良いと思います。　上山さん／石川県

初心者向けに基礎から解説したので、初心者ではない人にとっても振り返りのための良い記事になりました。そのためか、新人さんよりも教える立場のベテランさんからの声が多かったです。

第2特集　はじめてのサーバーレスアーキテクチャ

最新の開発手法「サーバーレスアーキテクチャ」について、登場に至った経緯と最新の状況を押さえたあと、2つのサービス「AWS Lambda」と「Microsoft Azure Functions」を使って実践的に解説しました。

具体的なサービスを挙げて説明されているのが丁寧で良かったです。
村橋さん／北海道

サーバーレス＝スタンドアロンと思ってしまった。
やっぱり紙がすきさん／奈良県

Serverless Framework は Serverless環境の構成管理にも使えますね。勉強になりました。　都築さん／愛知県

サーバーレスという概念にワクワクしています。　にしむさん／東京都

そんな開発手法が出てきているのか、と驚く声が多かったです。クラウドは変化の速い世界ですが、開発の効率化のためにキャッチアップしていきたいですね。

一般記事 mBaaSのしくみ紹介【1】

モバイルアプリのバックエンド開発を省力化できるmBaaS。その1つ「ニフティクラウドmobile backend（NCMB）」のしくみと使い方を解説する短期連載です。第1回ではNCMBの概要と採用事例を紹介。

最初は「クラウドサービスの名称がまた増えたのか……」と思いましたが、わかりやすい説明で興味が持てました。
最強のゆめなさん／神奈川県

ちょうど今開発しているアプリで使おうと思っていたので、良かった。後編にも期待しています。　吉良さん／東京都

クラウドにバックエンドを任せるのは良いなと思いました。
クラウドさん／京都府

4月号のプレゼント当選者は、次の皆さまです

①無線LANギガビットルータ「WRC-1750GHBK-E」
田口保志様（長野県）

②世界巡業「JF-PEACE4」
kinoko様（東京都）、シャケのフレンズ様（大阪府）

③Paragon Hard Disk Manager for Mac
中村昭博様（石川県）、yoshitaka様（神奈川県）

④『Linuxカーネル「ソースコード」を読み解く』
高野俊作様（神奈川県）、杉山貴紀様（山口県）

⑤『プログラミング作法』
山口尚晃様（愛知県）、佃駿介様（三重県）

⑥『進化する銀行システム』
下平学様（東京都）、ぴょうへい様（大阪府）

⑦『プロが教える情報セキュリティの鉄則』
松井吉光様（愛知県）、天野恵子様（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。

モバイル開発だけでなく、IoTの分野で今後はもっと必要になるサービスだと思う。 HOSさん／京都府

mBaaSは「サーバーレスアーキテクチャ」と同じく、開発の大部分をクラウドサービスでまかなうしくみでした。導入によってアプリやサービスの開発に注力できるということで、魅力を感じている読者も多いようです。

一般記事　PG-Stromの構造と機能、そしてその威力とは【後編】

PG-Stromは、GPUを用いて、集計・解析系の処理を非同期・並列に実行できるPostgreSQLのオープンソースの拡張モジュールです。後編では、GROUP BYというSQL文がGPU上でどのように実行されるかを解説しました。

GPUとデータベースの連携で強力なことができるとは知らなかった。
jo7oemさん／山形県

GPUがもっと利用されるようになるとおもしろくなるのでは？と期待します。
とーふやさん／神奈川県

GPUはコスト的に取っつきにくいのでPG-Strom自体にはあまり興味なかったんですが、処理内容の解説がとても良かったです。 九郎さん／大阪府

GPUをDBで使う発想はおもしろいと思った。 浅野さん／神奈川県

GPUでSQLを分散（？）処理させようと思いつくだけでもすごいことですが、それを実装してしまうところがまたすごい。で、結果を残しているんだから、ほんと。 atachibanaさん／東京都

RDBとGPUを組み合わせるという発想に驚いた、という声を多くいただきました。ビッグデータの時代に突入した今、ますます注目される技術になることでしょう。

一般記事　Jamesのセキュリティレッスン【9】

セキュリティ対策・トラブルシューティングに活用できるパケットキャプチャのツール「Wireshark」の使い方を紹介する短期連載です。今回はIPsecの暗号化通信を復号する方法を解説しました。

IPsecのことを理解してないネットワークエンジニアも増えているので、すごく良い記事だと思う。
romeosheartさん／長崎県

自分はネットワークが専門なのですが、WiresharkでIPsecパケットを復号できるというのを初めて知りました。新しい収穫です。 にわとりさん／東京都

この連載を通じセキュリティに対する確固とした考え方や視点が得られ、有意義でした。 オミオさん／宮城県

記事の中のipsec-gw1、ipsec-gw2のOSが、UbuntuではなくWindows OSの場合にもIPsecの復号ができるかどうかに興味があります。
psiさん／東京都

手元でできるセキュリティ対策記事ということで人気の高い本連載。いつも読んでいただいている方は、知識や技術だけでなく、セキュリティに対する意識も高めていただいているようです。

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
July 2017

2017年7月号
定価（本体1,220円+税）
184ページ

6月17日発売

［第1特集］初心者もベテランもさらに実力UP

理論&応用で シェル力の幅を広げる

［第2特集］データの抽出・加工に強くなる!

MySQL［SELECT文］集中講座

MySQLの始め方とデータのあつらえ方／プロ直伝SELECT文のマスター方法

［一般記事］

ハッシュ関数とは何か?（後編）／Windows Server 2016ファイルサーバ構築（後編）／またまた帰ってたJamesのセキュリティレッスン

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「Vimの細道」（第19回）は都合によりお休みさせていただきます。

SD Staff Room

●健診で胃に異常が見つかり、ピロリ菌除菌を勧められた。検査はいたって簡単で尿検査でピロリ菌の存在がわかる。ということで今号制作中に抗生物質を朝晩飲んでいる。結果がわかるのは3ヵ月後だ。おそらく9月号あたり。今度は検便しなければならない。いろいろ体にガタがきているアラフィフ編集じゃった。（本）

●先月のデジカメ写真閲覧環境の続き。iPad Pro 12.9ならば2,732×2,048なのだがお高いので、Huawei PediaPad M3を導入してみた。8インチ2,056×1,600のIPS液晶で発色も良く、SIMフリーで動作も速い。こんなのが40k円以下なんて、すごい世の中になったものだ。（でもiPad mini待ち幕）

●行ってきました「技術書典2」。11時半ごろに着いたら、すでに入場制限で行列が。あいにくの雨の中、傘をさして静かに並ぶ方々に囲まれながら、こんなにまでして技術書を読んで勉強したい人がいるのかとニヨニヨしていたのは私です。主催者、出展者はもちろん、来場者の皆さまもお疲れさまでした!（キ）

●Suica定期券の期限切れに気づかずに電車に乗り続けていると、いずれSuica残高が不足して改札で止められます。私は定期が切れる6月と12月に毎度失敗します。今回は運良く今気づいたので、ここに書いてみました。本誌発売日に自らこの文章を読んだとき、定期券更新を思い出す作戦です。（よし）

●赤ワインをコーラで割ってつくるカクテル「カリモーチョ」にハマり中。ワインの渋さとコーラの甘ったるさが調和して、なんとも不思議な味わいです。調べると、カリモーチョはスペインが起源とのこと。同じくスペインで生まれたサングリアも大好きなので、いつか本場のバルをはしごしてみたいです。（中）

●毎年開催を楽しみにしている『ホビーショー』。ハンドメイド好きにはたまらなく創作意欲を高めてくれる楽しいイベントです。会場はかなりの広さがありブースも多いので一日歩き回っても、全部は見れないくらい時間が足りないのです。今年はどのくらい見れるかな。余計な資材まで買わないように気をつけないと。（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］
03-3513-6179

※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 2017年6月号

発行日
2017年5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。