Special Feature 1 →シェル力養成
Special Feature 2 →SELECT文をマスターしていますか？

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

July 2017

7

2017年7月18日発行
毎月1回18日発行
通巻387号
（発刊321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

bashをもっと使いこなしませんか？

Special Feature 1

理論&応用で「シェル力」の幅を広げる

基礎の基礎からベテランも知らない技までガッツンと習得

データの抽出・加工に強くなる！

Special Feature 2

MySQL［SELECT文］集中講座

MySQLの始め方／
ORDERとGROUPでデータ抽出／
JOINとサブクエリでデータ加工

Extra Feature 1

ハッシュ関数とは何か？（後編）

Extra Feature 2

Windows Server 2016
ファイルサーバ構築（後編）

Extra Feature 3

またまた帰ってきた
Jamesのセキュリティレッスン

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
July, 2017
7

Special Feature 1 → 第1特集 | Page

Contents

Column

Development

OS/Network

[ロゴデザイン]
デザイン集合ゼブラ+坂井 哲也
[表紙デザイン]
藤井 耕志(Re:D Co.)
[表紙写真]
kasto / AdobeStock
[イラスト]
フクモトミホ
[本文デザイン]
*安達 恵美子
*石田 昌治(マップス)
*岩井 栄子
*ごぽうデザイン事務所
*近藤 しのぶ
*SeaGrape
*轟木 亜紀子、阿保 裕美、
佐藤 みどり、徳田 久美
(トップスタジオデザイン室)
*伊勢 歩、横山 慎昌(BUCH+)
*藤井 耕志、萩村 美和(Re:D Co.)
*森井 一三

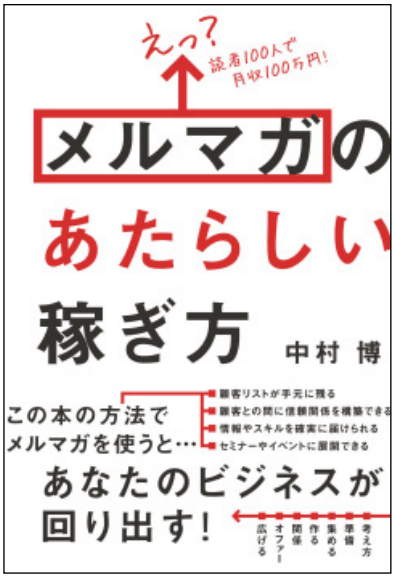
この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

BOOKS INFOMATION

えっ？ 読者100人で 月収100万円！ メルマガの あたらしい 稼ぎ方

中村博 著

A5判／272ページ
定価（本体1480円+税）



ISBN978-4-7741-8961-1

従来、メルマガで行う集客、販促といえば、膨大な数の読者を獲得し、見込み客リストを充実させることが推奨されてきました。しかし、多くの個人事業主や小さな会社にとって、多数の読者の獲得にコストや労力はかけられません。本書は、こうした悩みを抱える読者のために、少ない読者から効率的に顧客を獲得し、無理なく販促活動につなげる方法を伝授します。ポイントは、読者との信頼関係を築くこと。1対1の関係から、個別メール、オファー、成約へと結び付けていきます。少ない読者で最大の利益を得る、あたらしいメルマガの稼ぎ方を、いますぐ学んでみてください！

越境EC& 海外販売 攻略ガイドブック

佐藤亘 著

ISBN978-4-7741-8966-6

A5判／320ページ　定価（本体2280円+税）

今、「越境EC」というキーワードが注目を集めています。
越境ECとは、経済産業省が提唱したキーワードで、アジアや欧米向けに自社ECサイトを設立し、そこで様々な商品を販売する形態のことを指します。
従来日本国内のサイトでしか販売していなかった事業者はもちろん、海外AmazonやeBayでしか販売を行ってこなかった事業者にとって、独立型ECサイトについてのノウハウは少なく、今、情報が求められています。
本書では、越境EC サイト制作をはじめて制作する担当者、これから海外販売を始めたいと考えている経営者、国内Web 制作会社の担当者向けに、必要な知識を最速で学習できるようにまとめた、越境ECの準備・構築・決済・発送・運営に役立つ実務書です。

今すぐ使えるかんたんEx

YouTube 投稿&集客 プロ技セレクション

リンクアップ 著

ISBN978-4-7741-8963-5

A5判／256ページ　定価（本体1680円+税）

この1冊でYouTubeのすべてが分かる！
本書はYouTubeの基本操作から、動画を投稿・編集、チャンネルの管理や「動画が見られているか」の動画分析＆改善方法までを紹介しています。
もちろん、動画から広告収入を得るための設定も解説。個人で動画を投稿してお金にしたい方、仕事で動画を販促に使いたい方におすすめの書籍です。

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

Pocket Reference

小さくても、中身充実！

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

最新刊！

朝井淳 著
四六判／656ページ
定価（本体1980円+税）
ISBN978-4-7741-8732-7

土井毅・高江賢・飯島聡 著／山田祥寛 監修
四六判／512ページ
定価（本体2640円+税）
ISBN978-4-7741-9030-3

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

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

岡本隆史・武田健太郎・相良幸範 著
四六判／384ページ
定価（本体2780円+税）
ISBN978-4-7741-8593-4

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

髙江賢 著／山田祥寛 監修
四六判／576ページ
定価（本体2680円+税）
ISBN978-4-7741-8030-4

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

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

山近慶一 著
四六判／688ページ
定価（本体2980円+税）
ISBN978-4-7741-8001-4

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

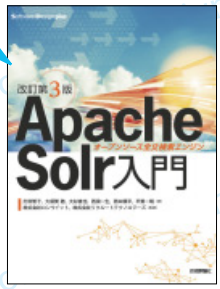
1 >> ／˜＼Fujisan.co.jpクイックアクセス
http://www.fujisan.co.jp/sd/

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）

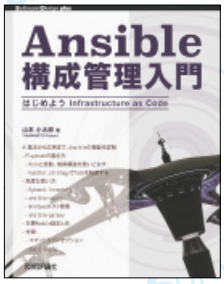
Software Design plus

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

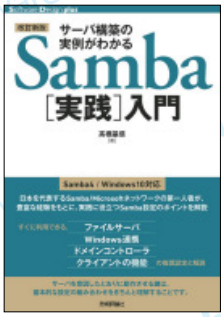


山本小太郎 著
B5変形判・176ページ
定価 2,480円（本体）+税
ISBN 978-4-7741-8885-0

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

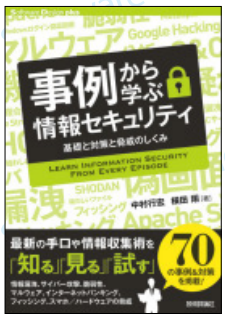
養成読本編集部 編
B5判・160ページ
定価2,180円(本体)+税
ISBN 978-4-7741-8895-9

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

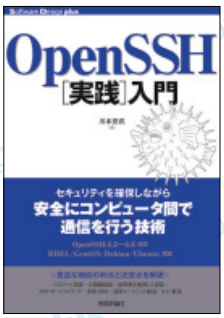
香山哲司、小野寺匠 著
A5判・176ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-8815-7

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

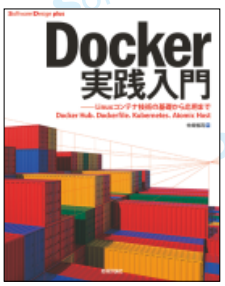


中村行宏、横田翔 著
A5判・320ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-7114-2



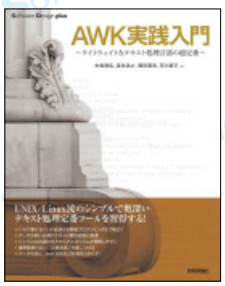
川本安武 著
A5判・400ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-6807-4

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

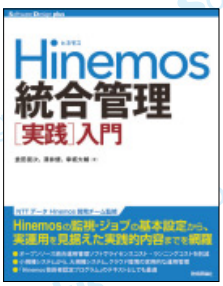


中井悦司 著
B5変形判・200ページ
定価 2,680円(本体)+税
ISBN 978-4-7741-7654-3

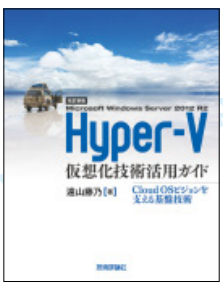
上田隆一 著
USP研究所 監修
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7344-3



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103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。

Moby Project

コンテナシステムのための新プロジェクト

「Moby Project」は、2017年4月にDocker社が発表したコンテナシステムのための新しいオープンソースプロジェクトです。Docker社では、これまでにもコンテナベースのシステムを構築するためのさまざまなコンポーネントを開発・提供してきました。Moby Projectでは、これらのコンポーネントを他のコンポーネントと任意に組み合わせることによって、自由にカスタムのコンテナシステムを作り上げられるしくみが提供されます。

具体的には、Mobyは次のような要素で構成されます。

- コンテナ化されたバックエンドコンポーネントのライブラリ。たとえばローレベルのビルダやロギング機能、ボリューム管理機能、ネットワーク機能、イメージ管理機能、コンテナランタイム、オーケストレーションツールなど
- コンポーネントを標準的なコンテナ・プラットフォームで組み立てるためのフレームワーク、および構築、テスト、デプロイメントのためのツール群
- Moby Originと呼ばれる、コンポーネントの組み立て（アセンブリ）の参照実装

コンポーネントやアセンブリはオープンな共有モデルによって提供され、サードパーティ製のコンポーネントも含めて自由に入れ替えることができます。Moby Projectは、コンポーネントおよびアセンブリの開発におけるコラボレーション基盤として機能するとのことです。

関連ツールのコンポーネント化を進めてきたDocker

Moby Projectの公開に至る以前から、Docker社ではコンテナ関連技術のコンポーネント化を積極的に進めていました。その中には、コンテナベースシステムの構築や管理のためのツールはもちろんのこと、既存のDockerエンジンの機能の一部を独立したツールとして提供したケースもあります。現在提供されている主要なコンポーネントとしては次のようなものが挙げられます。

- **VPNKit**……仮想ネットワークを実現するためのライブラリツールキット
- **DataKit**……分散コンポーネント向けのパイプラインフレームワーク
- **HyperKit**……macOSのHypervisor Frameworkに対応した軽量な仮想化ツール
- **libnetwork**……Dockerのネットワーキング機能を提供するコンポーネント
- **Notary**……安全なコンテンツの公開やコンテンツの内容の検証などを行うためのユーティリティ
- **InfraKit**……クラウドやオンプレミスにおいて、自己修正型のインフラを構築・作成できるツール
- **LinuxKit**……containerdが動作することに特化した最小限のLinuxをビルドするためのツールキット
- **SwarmKit**……マルチホスト環境に対応したDockerコンテナのオーケストレーションツール
- **runC**……コンテナ標準であるOCIに準拠したコンテナランタイム
- **containerd**……Dockerエンジンのコアとなるコンテナランタイム。コンテナ実行レイヤの実装にはrunCを利用している
- **Docker Compose**……複数のDockerコンテナからなるサービスの構築や実行を自動化するツール

このうち、LinuxKitはMoby Projectの発表と同時にOSSとして公開されました。これまでのDockerコンテナはホストOSの上で動作するプロセスの1つでしたが、LinuxKitを使うことで、OS機能ごとコンテナとしてパッケージングして配布できるようになります。

Docker社では、プラガブルで柔軟なコンテナシステムを実現するために、長い時間をかけてこのようなコンポーネント化に挑んできました。Mobyでは、これらのコンポーネントを基本としつつ、必要に応じてコンポーネント単位で組み合わせを変えることによって、特定用途向けのコンテナシステムを簡単に構築することができます。コンテナがさまざまな分野に普及していくうえで、Moby Projectによるエコシステムの形成は大きな役割を果たすことになりそうです。SD

Moby Project
https://mobyproject.org/

DIGITAL GADGET vol.223

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

幼児のためのデジタルガジェット

新しい世代の子供たち

現在、床をハイハイしている赤ちゃんたちは、ごくごく普通に生まれたときからネットはもちろん、スマートフォンやタブレットなどのタッチパネル端末が存在します。そういうタッチすると何か反応することを覚えた子供は、紙の雑誌やテレビ画面さえも、なんでもかんでも触って自分の思いどおりに動かそうとするわけです。

小さな子供には、大人の身体や特徴とは異なるさまざまな要素が存在します。

音に敏感に反応したり、指がまだ正確に使えず、グーパーと握るだけだったり、指も細く、小さすぎて何かをうまくつかんだり、タッチパネルを的確に操作したりできません。そもそもシングルタッチと呼ばれる人差し指だけでボタンをタッチするような操作は不得意です。

また興味を持って何かを触るときも、どこを触るのかわからず、飽きっぽくて、何かに集中していられる時間が短いかと思えば、気に入ったことがあると、何度も何度も飽きずに繰り返します。ティッシュを引き出すことができるとわかったら、もう延々とティシュを箱から引き出してみたり、大人の常識は子供には通じません。

けれども「赤ちゃんだから」と割り切って考えるかもしれませんが、飽きやすい、待たない、慣れないと的確に扱えないといったことの本質は、子供だけでなく、大人や高齢者も含め、すべての人たちに当てはまる事柄でもあるのです。家庭に乳幼児がいると、大人の理屈では考えられないほどの、さまざまな面倒な出来事や、手間、作業が数多く予想できないタイミングで発生します。

生まれたばかりの赤ちゃんは、目もまだまだ発達しきっておらず、大人と同じように物事を見たり判別したりすることができません。赤ちゃんの見え方を試すことのできるサービス「Tiny Eyes（http://tinyeyes.com/）」では、画像をアップロードすると、赤ちゃんの見えている視点ふうにぼんやりした画像を生成してくれます。赤ちゃん向けの絵本がパッキリくっきりとした絵や色合いなのも納得です。Tiny Eyesでは、生まれたばかりの赤ちゃん、1ヵ月から、2、3、6ヵ月の見え方を示してくれます。

また、赤ちゃんがボンヤリとしか見えないことを考慮し、赤ちゃんの視線を集める単純な模様が描かれた「Shutter Buddy」（**pic.1**）というカメラグッズもあります。そういった赤ちゃんならではの特性をとらえたさまざまなグッズが存在しているので、今回はそれらを紹介していきましょう。

赤ちゃんならではのデジタルグッズ

赤ちゃん本人が使うものと、赤ちゃんを持った親や大人たちが活用するデジタルグッズの両面があります。赤ちゃんや幼児は、何かがちょっと光ったり動いたりするだけでとても喜んでくれます。色や形も単純なものしか理解できず、何か深く考えるまえに、まずは触ってみることから始めます。大人の手間や作業を少しでも削減し、子供たちをあやしたり、楽しんだりするデジタルデバイスをいくつか紹介します。

Rayiot

http://www.rayiot.org/

産婦人科の病院で使うような、赤ちゃんの呼吸を監視するモニターの家庭版（**pic.2**）。呼吸、睡眠、寝返り、目を覚ましたことをスマホに知らせてくれます。非常用バッテリーも備

指のサイズが小さい赤ちゃんの手

※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。

え、停電時も3時間ほど動作するそうです。赤ちゃん監視用のカメラとしてはほかにも「Evoz」(**pic.3**)や、スマートフォン端末のカメラを活用する「Dormi」(**pic.4**)などがあります。

Nanit
https://www.nanit.com/

赤ちゃんの睡眠モニター(**pic.5**)。オプションを追加すると赤ちゃんの睡眠パターンの分析が行え、赤ちゃんの就寝、寝起きのパターンや睡眠時間をスマートフォンで把握することができます。何時ごろ眠ると朝まで寝ているとか、単にグズっているだけですぐに寝付くとか、何時ごろ眠ると、夜中何時ごろ起きるといったパターンがわかるわけです。また自然の音を流してくれる機能もあり、さまざまな方法で赤ちゃんの就寝を手助けしてくれます。

SNOO Smart Sleeper
https://www.happiestbaby.com/pages/snoo

自動ゆりかご(**pic.6**)。赤ちゃんの泣き声を検知して、自動的に揺れ始めます。また、単にあやすだけでなく、赤ちゃんが眠りに入りやすいよう、ゆっくり揺らしたり、多くの赤ちゃんでユーザテストした結果から最適な揺れを提供してくれるそうです。1,160ドルで販売中。そのほかにも既存のゆりかごの足に装着して自動で揺れるゆりかご化する「Lullabub」(**pic.7**)という商品もあります。

未来の世代のためのデジタルガジェット

これから育っていくであろう新しい世代は、物事のとらえ方が異なってくるでしょう。たとえば……

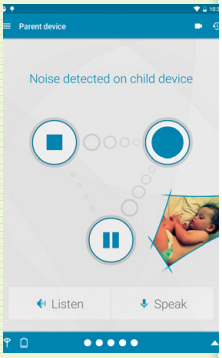
- 高速ネットワークは当然常時つながっているのがあたりまえ
- データの転送量やファイルのサイズなどを気にせずコンテンツを扱う
- コンピュータやネットの負荷などは気にしない
- クラウドにある、ローカルにある、などとデータの置き場所を意識しない
- ちょっとした短い時間のタイムラグやズレ、遅れも気にする
- モノよりコト、何でも手に入る時代なので所有欲やコレクション欲が薄れる

たとえば、現在家や部屋にテレビがない、テレビを見ないという人がいるかもしれません。ゲーム専用のテレビや、コンピュータ用のディスプレイや映画鑑賞用のプロジェクタはあっても、地上波のテレビを録画ではなくリアルタイムで見ることは減っているかもしれません。ある年代以上であれば、たいていの場合、子供のころ、家にテレビが存在したと思います。ですから、現時点で家にテレビがなくても、テレビがある風景というものは想像がつきま

pic.1
Shutter Buddy(左)
Shutter Huggers(右)

pic.2
Rayiot

pic.3
Evoz
https://myevoz.com/



pic.4
Dormi
http://dormi.sleekbit.com/

pic.5
Nanit

pic.6
SNOO Smart Sleeper

pic.7
Lullabub
http://www.babyhugs.com.au/

す。けれど、もう一世代巡り巡って、生まれたときから家にテレビがない世代というのも増えてくるのではないでしょうか。手間をかけずに、YouTubeやテレビ局のアプリで見たい番組が見られる時代が特別なことではなく、ごく普通のことだと子供たちに浸透していっています。

一方、スマートフォンでなんでも撮影でき、音楽も聴き放題のこの時代に、一部の人たちに昔流行った使い捨てのフィルムカメラやカセットテープが見直されつつあります。フィルムカメラは、現像やプリントしなければできあがった写真を見ることができず、面倒な手間と、貴重な時間がかかります。また撮影時に失敗したときも、あとからでないとわかりません。カセットテープもわざわざ録音したり、巻き戻したり、音楽を聴くための準備も、聴く方法もスマートフォンと比較すると面倒であることは否めません。

そういった、一見無駄とも思える時間と手間ひまをかけ、そうやって得られたコンテンツだからこそ、貴重なものとして、特別な感情を抱くことができるのかもしれません。これからは、デジタルデバイス、デジタルなサービスにおいても、必要な手間や、単に素早いだけではない時間の流れ、愛着といったアナログ的な要素を意図的に組み込んでいく必要があるのかもしれません。

最近では、赤ちゃん向けにデジタル技術を活用したBabyTechという分野も広がってきており、BabyTech Awards（http://awards.babytechsummit.com/）という賞が開設されました。この記事でも紹介した商品を含め、数々のグッズがエントリーされ、執筆時現在審査中です。

分野としては、赤ちゃんの飲み食い、遊びや教育、赤ちゃんの移動や乗り物、安全のための見守り、健康状態のチェックなど、ありとあらゆる分野に広がっています。テクノロジの力を借りて、子育てがより便利になり、赤ちゃんたちが元気に成長していってほしいものです。SD

Gadget 1

Pechat

https://pechat.jp/

ぬいぐるみを おしゃべりにするボタン

Pechatは、一般のぬいぐるみの胸にボタンとして取り付けて利用する、Bluetoothスピーカーマイクデバイスです。専用アプリから音声を出すことで、あたかもぬいぐるみがしゃべっているかのように、扱うことができます。あらかじめ登録された言葉や、歌、お話しのほかにも、周囲の声に反応して対話やおうむ返しをするモードもあります。充電式で、3時間の充電で連続1.2時間利用できます。見通し距離でスマートフォンと約10メートル以内のところで使えます。

Gadget 2

Baby Gigl

http://www.slowcontrol.com/en/baby-gigl/

スマート哺乳瓶

Baby Giglはミルク量を計測してくれる、賢いスマート哺乳瓶です。哺乳瓶の本体カバーに重量センサーと傾きセンサーを搭載し、赤ちゃんにミルクをあげた時間や量を計測できます。哺乳瓶を傾けすぎないように警告のランプが点いたり、警告音で知らせてくれたりします。センサーで得られたデータはスマートフォンと連携して記録できます。数時間ごとにミルクを飲む赤ちゃん（と赤ちゃんに飲ませる親）にとっては便利なグッズです。現在開発中で100ドルほどで販売予定とのこと。

Gadget 3

Starling

https://www.versame.com/starling/

言葉の数を 記録してくれるデバイス

Starlingは赤ちゃんに話しかけた言葉の数を記録してくれる星形のウェアラブルデバイス。赤ちゃんの言語能力は幼いころにどれだけ話しかけられたかによって決まるらしく、周りの大人がどのくらい話しかけているのかを計測し、その量を意識するためのデバイスです。また話しかけられる言葉だけでなく、赤ちゃん本人が発した言葉もカウントしてくれます。バッテリーは一度の充電で5日間持ち、防水機能あり。149ドルで販売中。スマートフォンと連携して利用できます。計測はネットなしでも可能です。

Gadget 4

TempTraq

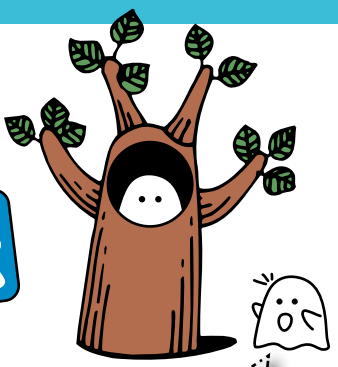
https://www.temptraq.com/

赤ちゃんの 体温計測デバイス

TempTraqは赤ちゃんの体温を計測し、スマホに転送できるウェアラブルデバイスです。子供が高熱を出したときに脇に貼って様子を見ることができます。大型の絆創膏のようなシール状になっており、使い捨てで24時間利用できます。就寝中にあらかじめ指定した度数を越えたら、親に通知したり、2人の子供が同時に高熱を出したときにも見分けることができたり、さまざまな状況に配慮されています。米国では一般的な薬店で定価19.99ドル、実売価格10ドル程度で販売中。米国ではFDA（食品医薬品局）の認可済みですが、日本販売は未定です。

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

結城浩の再発見の発想法 50



Branch ── ブランチ

ブランチとは

ブランチ(Branch)とは、**バージョン管理システムで管理される、プログラムの別バージョン**のことです。ブランチという言葉は「幹から分かれた枝」のことで、大本になっているプログラムを木の幹に見立て、別バージョンを幹から分かれた枝にたとえた用語です。

ブランチの呼び名は開発プロジェクトごとにさまざまですが、実際に運用しているブランチを**マスターブランチ**と呼び、開発のために使うブランチを**開発ブランチ**と呼ぶことがあります(**図1**)。この場合、マスターブランチはいわば木の幹にあたりますが、幹と枝の区別を付けずにすべてブランチとして扱っていることになります。

わざわざブランチという形でプログラムの別バージョンを作るのは、動作しているプログラムを修正によって壊さないようにするためです。プログラムは複雑な構造物ですから、修正するときにはほかの部分へ影響が起きないように注意する必要があります。たとえば、多くのユーザがいるプログラムで修正ミスが入り込んだら、たいへんなトラブルになってしまいます。

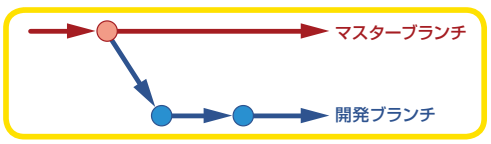
マスターブランチと開発ブランチを分離するのは、そのようなトラブルを避けるためです。開発者は、マスターブランチを直接修正するのではなく、いったんマスターブランチとまったく同じ開発ブランチを作成します。そしてその開発ブランチのほうに修正を加えていくのです。

開発ブランチ上で作業している間、マスターブランチにはいっさい変更ありません。開発ブランチ上での作業が終わり、十分にテストを行ってからマスターブランチに修正点を反映(マージ)することになります(**図2**)。修正点は、開発ブランチが枝分かれした時点から現在までの差分で判定します。このように、マスターブランチと開発ブランチとを分離して、**安定した運用を保ちつつ修正を加えていく**のは、ブランチを作る目的の1つです。

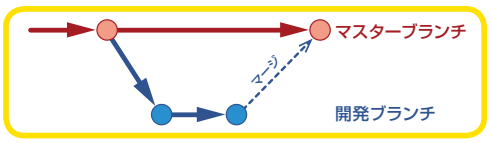
プログラムを開発するときのブランチは2本とは限りません。プログラムに新しく機能を追加するとき、**機能ブランチ**と呼ばれるブランチを作ることがあります(**図3**)。複数の開発者が実装に取り組む過程で、たくさんの機能ブランチが作られるのはめずらしくありません。**複数の開発者が並行して作業を進める**のも、ブランチを作る目的の1つです。

機能ブランチごとに開発された新機能は、完成するごとに開発ブランチにマージされ、さらにテストが済んだところでマスターブランチに

▼図1　マスターブランチと開発ブランチ



▼図2　マスターブランチに開発ブランチをマージする



マージされます。

バージョン管理システムとブランチを使ったプログラムの開発は、作業が進行していくにつれて枝から新しい枝が伸び、そして枝の先がほかの枝に戻っていくような姿になるでしょう(**図3**)。

独立性

「マスターブランチを壊さないようにするため開発ブランチを作る」という目的を考えると、ブランチごとの**独立性**が重要であることがわかります。つまり、あるブランチにいくら修正を加えても、マージするまではほかのブランチにいっさい影響がないということです。ブランチの独立性が守られているため、開発者は失敗やミスを恐れずに安心して開発を行うことができるのです。

「複数の開発者が並行して作業を進める」という目的にも独立性が効いてきます。1つのプログラムに対して複数の開発者が修正を行うのはよくあることです。しかし、1人の修正がほかの人に影響を及ぼすなら、作業効率は落ちてしまいます。機能ブランチが独立していることで、ほかの人の修正に影響を受けず、落ち着いて作業を進めることができるのです。

日常生活とブランチ

日常生活でブランチに似たものはあるでしょうか。プログラムは複製や差分の判定が低コストで行えるので、ブランチを作るのは難しくありません。しかし、日常生活の中では文字どおりのブランチを作ることは難しくなります。それは物理的なものを複製するのは難しいからです。しかし、独立性という観点で考えると、ブランチに似たものは見つかります。

どんな会社も、安定している事業を失敗させたくはありません。しかしその一方で、新しい取り組みを行わなければ時代遅れになってしまうでしょう。ここに見られるのは、運用中のプログラムを壊さないようにしつつ新機能を入れたいという要求と同じです。

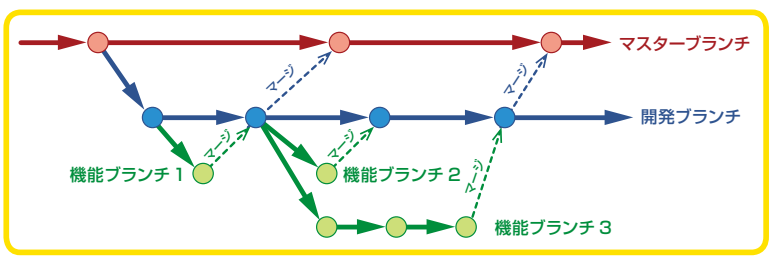
また、いいアイデアを見つけた意欲のある社員は新事業を興したいと思うかもしれません。しかし、会社に属しているという自分の環境を捨ててゼロから起業するのは難しいでしょう。ここにあるのは、自分の環境を壊さないように保ちつつ新しいチャレンジをしたいという要求と言えます。

社内ベンチャーという制度を採用している企業があります。この制度は、社員が得た事業のアイデアを会社の支援を受けつつ実現するというしくみです。社内ベンチャーを利用するというのは、社員であるという自分の環境をそのまま保ちつつ、新たな事業に取り組むわけなので、ちょうど、マスターブランチから新たなブランチを作成したようなものです。たとえ新たなブランチで失敗があったとしても、元の環境に戻ることができるので、安心してチャレンジができるでしょう。

あなたの周りを見回して、安定したものを守りつつも改善を加えたいという要求はないでしょうか。改善はしたいが安定しているものを壊したくはないというのは一見矛盾した要求です。そのときに、独立性を保証する「ブランチ」に相当するものは作れないでしょうか。失敗やミスがあっても安定しているものを壊さないような環境や、「お試し」ができる場を作ることはできないでしょうか。

ぜひ、考えてみてください。SD

▼図3　機能ブランチ



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9回 リーンキャンバス

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

本誌を読まれている方ならば、リーンスタートアップという言葉を聞いたことがあるでしょう。スタートアップのバイブルとも言われるエリック・リースの同名の書籍『リーンスタートアップ』[注1]で一躍有名になった新規事業立ち上げや起業の方法論です。

スタートアップという言葉から、起業家にだけ関係する方法論と誤解されるかもしれませんが、書籍の中でも「スタートアップとは、とてつもなく不確実な状態で新しい製品やサービスを創り出さなければならない人的組織である」とされているように、実は起業家や（いわゆる）スタートアップで働く人だけでなく、すべての人に有効な方法論です。

今回はこのリーンスタートアップで用いられるリーンキャンバスについて解説します。

リーンスタートアップ

さて、あなたが優れた（と信じる）アイデアをもとに起業した起業家だとしましょう。起業家のあなたにとってもっとも恐れるべき事態は何でしょうか？　アイデアを具現化した製品やサービスが使われないことでしょうか？

もちろん、そのとおりです。開発費を投入した製品やサービスが使われないと資金回収できずに、会社は立ち行かなくなってしまいます。

しかし、製品やサービスが最初からうまくいくことは稀（まれ）です。もっとストレートに言うならば、最初に思いついたアイデアは良いものでないことのほうが多いのです。ですので、起業家にとってもっとも恐れなければいけないことは、最善のソリューションである製品やサービスを顧客に届けることができるようになるまでに、リソースが枯渇（こかつ）することなのです。

ここで言うリソースとは、おもに運転資金ですが、エンジニアリングリソースなども含みます。あなたは起業家として初期の運転資金を調達していたとします。最初の資金調達は額も少額で、この資金で最初の製品を世に出す必要があり、それが市場に受け入れられると次のラウンドの資金調達へと進めることになります。

最初に考えたアイデアが当たればそれにこしたことはありませんが、多くの場合はそのアイデアがそのままで通用することはありません。何度もの変更を加えながら、世の中に受け入れられるものを見つけ出していくのです。

ここではあえて起業家としましたが、ここで言う起業家はまさにスタートアップという組織そのもののことです。冒頭で書いたスタートアップの定義からもわかるように、起業家だけでなく、新規事業にかかわるすべての人、中でも製品を成功に結びつける責任を負うPM（Product Manager）がまさに意識すべき考えがリーンスタートアップなのです。

それでは、顧客の求める製品にたどり着くにはどうすれば良いでしょう？　それはひたすら、

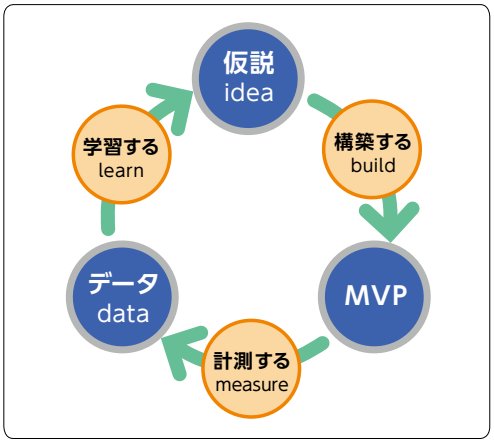
注1）エリック・リース 著、井口耕二 訳、伊藤穣一（MITメディアラボ所長）解説／日経BP社刊／ISBN978-4-8222-4897-0

仮説→検証→修正の反復を繰り返すことです。仮説をもとにMVP[注2](Minimum Viable Product)を作り、その成果を計測し、計測データに基づいた学習から修正を加えること。これを何度も何度も繰り返すことが必要となります。書籍では、これは構築→計測→学習という反復で紹介されています(**図1**)。

整理すると、スタートアップの最大のリスクは誰も求めていないものを作ることであり、成功するスタートアップはリソースを使い切る前に十分な反復を行うスタートアップのことなのです。

注2) MVPとは検証に必要な最低限の機能を持った製品のこと。

▼図1 リーンスタートアップの提唱する構築→計測→学習という反復(書籍『リーンスタートアップ』より)



リーンキャンバス

このように、書籍『リーンスタートアップ』では「構築－計測－学習」という反復を高速に回すことで事業を成功に導く手法が紹介されています。しかし、この書籍で紹介されている内容は理論的な話が中心で、実際にどのように進めれば良いかまではあまり書かれていません。それが紹介されているのが、アッシュ・マウリャ著の『Running Lean ――実践リーンスタートアップ』[注3]です。

この中で著者のマウリャはスタートアップの最大のリスクである「誰も欲しくないものを作ってしまうこと」を避けるために、製品開発の最初のフェーズでまず課題とソリューションのフィットを確認し、課題とソリューションがフィットしたことが確認されたあとに、製品と市場のフィットを確認することが重要と解説しています。

この課題とソリューションのフィットと製品と市場のフィットを確認するために用いられるフレームワークがリーンキャンバスです(**図2**)。

リーンキャンバスは事業計画書相当の内容を1枚のキャンバスに収めたものです。

注3) アッシュ・マウリャ 著、角征典 訳、渡辺千賀 解説、エリック・リース シリーズエディタ／オライリー・ジャパン 刊／ISBN978-4-87311-591-7

▼図2 リーン開発で用いるリーンキャンバス。1から9の順番に埋めていく(書籍『Running Lean』より)

課題 Problems 1	**ソリューション** Solution 4	**独自の価値提案** Unique Value Proposition (UVP) 3	**圧倒的な優位性** Unfair Advantage 9	**顧客セグメント** Customer Segment 2
既存の代替品 Existing Alternatives	**主要指標** Key Metrics 8	**ハイレベルコンセプト** High Level Concept	**チャネル** Channels 5	**アーリーアダプター** Early Adopters
コスト構造 Cost Structure 7		**収益の流れ** Revenue Streams 6		

ここで「事業」という言葉が出てきましたが、製品と事業は表裏一体です。PMと事業責任者の境界がPMの役割を論じる中で永遠のテーマに近い論点であるのと同様、製品と事業の境もまたあいまいです。しかし、作った製品が市場に受け入れられ、きちんと事業として収益をあげられるものでなければ、継続性を持った製品として顧客への説明責任は果たせません。製品の成功は事業の成功でもあるとPMは認識しなければなりません。

リーンキャンバスの各要素

リーンキャンバスは次の9つの要素から構成されています。

1. 課題
2. 顧客セグメント
3. 独自の価値提案
4. ソリューション
5. チャネル
6. 収益の流れ
7. コスト構造
8. 主要指標
9. 圧倒的な優位性

リーンキャンバスは短い時間で一気にこの9つのマスを埋めてしまうことが勧められています。埋められないマスがあっても、気にせずに埋められる範囲で埋め、全体を俯瞰（ふかん）し、埋められる段階になったら埋めていくことで進めます。

❶課題と❷顧客セグメント

製品に期待されることは顧客の課題の解決、および新しい価値の提供です。価値も課題の解決の形で創出されることがほとんどですので、課題抽出こそが重要となります。

書籍『Running Lean ——実践リーンスタートアップ』の中で、「あなたの『製品』は製品では『ない』」というフレーズが出てきますが、顧客にとって重要なのは製品ではなく、自分の課題を解決してくれるものかどうかです。課題が存在しないのに、その課題を解決する製品を作っても誰も利用しません。課題がまず最初に考えるべき要素なのはそのような理由です[注4]。

このマスには1つから3つ程度の課題を書き、また現時点でその課題を解決するために用いられている代替手段があったならばそれを記述します。

また、顧客セグメントにはその課題を持つ顧客を具体的に記述します。その中でも、製品が市場に出た際に真っ先に飛びついてくれそうなアーリーアダプターを明確にします。

❸独自の価値提案

独自の価値提案とは、「差別化要因と注目に値する価値を説明した単一で明確な説得力のあるメッセージ」とされており、デーン・マクスウェルの公式と呼ばれる次の公式で考えることが推奨されています。

即効性のある
明快な見出し ＝ 顧客が望む成果
＋ 明確な期限
＋ それが達成されなかった場合の代替案

少しわかりにくいかもしれませんが、よく出されるドミノピザの広告の例を見れば理解できるでしょう。ドミノピザは「時間内に配達できないと無料」という広告を打ち出していました。これは、顧客が望む成果としての「ピザが配達されること」に、明確な期限としての「時間内」とそれが達成されなかった場合の代替案としての「無料」提供というのが組み合わさったものとなっています。ここまで明快にこのフォーマットに沿う必要はありませんが、顧客、その中でも先ほど定義したアーリーアダプターが望む成果がはっきりと表現されているようにします。

また、このマスにはハイレベルコンセプトというものも書くように言われています。これは

注4）そしてまた、製品は事業を成功に結びつけるビジネスモデルそのものであるため、「あなたの『製品』は製品では『ない』」と言われています。

広い視点で見た場合に、どのように位置付けられるかを示すもので、短い文で、すでに存在する別製品のメタファで表すことが期待されます。たとえば、YouTubeのことを「動画のFlickr[注5]」と例えることができます。

❹ソリューション

ソリューションは課題を解決する機能を書くことになります。最初にキャンバスを作成するときは、この部分はラフなアイデアレベルで十分です。課題の検証が済んでいない段階で、その解決案について練ってもしかたないためです。

❺チャネル

チャネルは製品をターゲットとなる顧客へ届けるための手段です。Web系のサービスであれば、リスティング広告やSEO、ソーシャル経由での流入などが考えられると思います。BtoB製品であれば代理店や関連メディアなども含まれます。

❻収益の流れと❼コスト構造

リーンキャンバスでいう製品とはビジネスモデルそのものですが、そのビジネスモデルには事業としての収益とコストの検討が不可欠です。

収益モデルにはいくつもの形態がありますが、どのような収益モデルを取るのかなどを検討します。また、事業を成り立たせるための主要コストを概算しておきます。

❽主要指標

製品(事業)の成否を判断するために見るべき指標を決めておきます。Web系のサービスであれば、DAU(日次のアクティブユーザ数)かもしれませんし、直帰率かもしれません。

書籍『Running Lean ——実践リーンスタートアップ』では、海賊指標(AARRR)と呼ばれる指標を使うことを推奨しています。これはグロースハックでも用いられる指標で、獲得(Acquisition)、活性化(Activation)、定着(Retention)、収益(Revenue)、紹介(Referral)のそれぞれの段階の頭文字を取って、AARRRと呼ばれています。

製品によって、どの指標を見るべきかは異なりますが、それよりもどの段階かによっても大きく異なります。そのため、もし複数の指標を想定する場合は、どの段階かもあわせて明確にしておくほうが良いでしょう。

❾圧倒的な優位性

圧倒的な優位性では、競合が簡単にはコピーや購入ができないものを挙げます。

検証プロセス

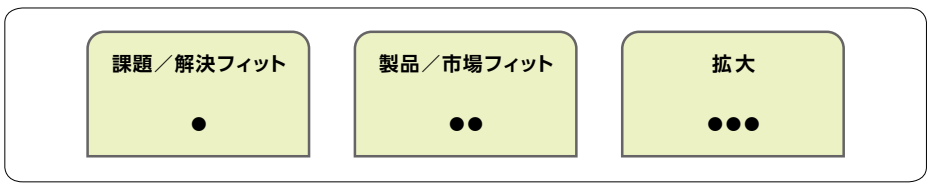
以上がリーンキャンバスの各要素となりますが、これを作成し、何度も書き直すことで、課題と解決のフィット、そして製品と市場のフィットを検証していくことが可能となります(**図3**)。

以上がリーンキャンバスの概要となります。

このリーンキャンバスは顧客が必要とする製品を開発するためにたいへん役立つフレームワークです。次回は、さらにこの中で課題とソリューションのフィットを確認するためのフレームワークである、バリュープロポジションキャンバスについて説明しましょう。SD

注5) FlickrはYouTubeより以前から存在している、写真共有サービス。

▼図3　スタートアップに必要とされる3つのステージ(書籍『Running Lean』より)



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。現在はいくつかのスタートアップのプロダクトマネージメントをサポートしている。

宮原徹の

オープンソース放浪記

第17回 ラズパイオーディオの会でコミュニティ活動

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

コミュニティ活動とは

普段、OSCの事務局のような裏方で忙しく活動しているため、私自身は特定のコミュニティでの活動はしていませんでした。ですが最近、Raspberry Piで音楽を聴く「ラズパイオーディオの会(@RasPiAudio)」というコミュニティを立ち上げて、OSCだけでなく外部のイベントにも出展して普及・啓蒙活動を開始しました。今回は「コミュニティ活動とはこんな感じ」というお話をしたいと思います。

ラズパイオーディオとは?

Raspberry Piはみなさんお馴染みだと思いますが、Raspberry Piでオーディオ、となると「?」な人が多いのではないでしょうか。

パソコンで本格的にオーディオを聴くには、オーディオインターフェースをUSBで接続するのが一般的な方法です。実はRaspberry Pi上で動作するLinuxも、このUSBオーディオインターフェースがあっさりと動作します。そして、Linuxカーネルのオーディオのしくみである ALSA(Advanced Linux Sound Architecture)や、音楽再生を担当するMID(Music Player Daemon)などを組み合わせれば、MP3やFLAC(Free Lossless Audio Codec)などの楽曲データを高音質で再生できるというわけです。もちろん、ハイレゾにも対応しています。

Raspberry Piで動作するオーディオ専用のディストリビューションも各種用意されています。よく使われるのは「Volumio 2[注1]」です。Webアプリでの操作だけでなく、専用のアプリでスマホをリモコンとして使うこともできます。楽曲データはNASに入れたりと、エンジニアスキルを活かせるのもおもしろいところです。

ラズパイオーディオの会としての活動

ラズパイオーディオの会は、2016年11月開催のOSC福岡から活動を開始しました。当時、佐賀在住の座布団1枚会長(@zabuton1mai)がOSC福岡に出展、セミナーを開催したところから結成されました。そのときの様子は本連載第12回(2017年2月号)でも書いています。

コミュニティ活動といっても、各人がTwitterなどで情報発信したり、メーリングリストでやりとりしたりするのが中心です。また、オーディオ系のイベントにも出展してラズパイオーディオをアピールしたりもしていますので、そのときの様子を紹介したいと思います。

注1) https://github.com/volumio/Volumio2

ヘッドフォン祭りに出展

まず、4月29日(土)、30日(日)に中野サンプラザで開催された「ヘッドフォン祭り」に出展しました。活動を支援していただいている出版社「ステレオサウンド」のブースを一部間借りしました。

イベント名称のとおりヘッドフォン好きの人が集まるので、Raspberry Piが小さくて持ち運びもできる、モバイルバッテリで駆動できるところをアピール。さらに試作品ですが、Raspberry Pi Zeroで使用する「SabreberryDAC ZERO」も展示させていただきました(**写真1**)。開発者のたかじんさん[注2]、ありがとうございました。

また、会長と私しかいないラズパイオーディオの会だけでは心許ないので、この連載では常連となりつつ

注2) http://nw-electric.way-nifty.com/blog/

▼写真1 「SabreberryDAC ZERO」を先行展示。わざわざこれの音を聴きに来てくれる人もいました

▼写真2　IoT女子のみなさん。事前の勉強会でラズパイオーディオの予習もバッチリです

▼写真3　OTOTENでの出展の様子。エレベーターの前なので、たくさんの人が立ち寄ってくれました

あるIoT女子(#IoT女子)のみなさんに応援に来てもらいました(**写真2**)。ブースがグッと華やかになりますね。

やはりIT系とは来場者層が異なるため、「Raspberry Piはなんとなく知っている」「興味はあるけどどう始めていいかわからない」という人が圧倒的に多く、かなり開拓の余地がありそうです。

Raspberry Piを使ったオーディオ機器を開発するメーカーのコンソーシアムが結成されており、このイベントでケースの試作品も展示されていましたから、オーディオ業界でRaspberry Piが広まる日はそう遠くないかもしれません。

OTOTENに出展

続いて、5月13日(土)、14日(日)に東京国際フォーラムで開催された「OTOTEN(音展)」にも出展しました。同様にステレオサウンドさんのブースを間借りしましたが、こちらのイベントはハイエンドオーディオ中心のイベントですので、Raspberry PiにUSB接続のDAC＋アンプをつないで、スピーカーで音楽を鳴らすデモを実施しました(**写真3**)。

来場者層もアナログオーディオ愛好家が中心のため、デジタルオーディオにはあまり詳しくない方が多く、基本的なところから紹介することが多かったように思います。ハイエンド市場にRaspberry Piが食い込んで行くには、もう少し時間がかかりそうです。

こんな感じで、ラズパイオーディオの会の活動を行っています。いろいろな人と趣味の話をするのは楽しいですから、まずは好きな何かでコミュニティ活動を始めてみてはいかがでしょうか？ SD

Report

京都奈良で日本酒三昧

ヘッドフォン祭りは2日間でしたが、筆者は初日だけの参加で、翌日から京都・奈良への旅に出かけました。テーマは「日本酒」です。

まず向かったのは伏見。軽く散策しつつ酒蔵さんに立ち寄ったり、酒屋さんで伏見のお酒を飲み比べたりしました。次に向かったのが奈良の大神(おおみわ)神社です。酒造りの神様として信仰されています。そのあと、昼間は室生寺や吉野山などに足を運び、夜は奈良の日本酒が堪能できる酒屋「なら泉勇斎」であれこれと飲み比べしてきました。奈良の日本酒、美味しいですよ。

◀大神神社の境内には奈良県内から奉納された酒樽が並べられています

▼活日(いくひ)神社。杜氏の祖先神 高橋活日命(いくひのみこと)を奉っています

▲酒屋「なら泉勇斎」。奈良県内のお酒を100種類以上試飲・購入できます

ツボイのなんでもネットにつなげちまえ道場

第25回

EnOceanを使ってみる

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

はじめに

今回は、EnOcean（エンオーシャン）という無線規格を使ってみたいと思います。EnOceanは、ドイツのEnOcean GmbHという会社が提供している技術で、「電池レス」というのが特徴です。どうやって電池を使わずにセンサや無線が動くのかというと、センサの周辺の環境から電力（エナジー）を収穫（ハーベスト）、つまり集めて、動きます。こういった技術は、エナジーハーベスティングだとか、環境発電と呼ばれたりしています。

エナジーハーベスティングと急に言われてもピンと来ないかもしれませんが、EnOceanで使われる発電手段の1つに太陽電池があります。太陽電池も、周辺からの光を電力に変換して機器を動かす、エナジーハーベスティングの代表例です。無線通信というのは意外と電力を消費するものですが、EnOceanは、少ない電力で無線通信を実現する技術と、エナジーハーベスティングが特徴です。

どうしてエナジーハーベスティングを行うと便利なのかというと、たとえば最近はトイレの使用状況をIoT化するのが一部のIT企業の間で流行っていますが、トイレのドアに取り付ける開閉センサの電源というのは意外と悩ましいものです。無線通信を行うのにもかかわらず、トイレのドアまで電源の線を引っ張るのもステキではありませんし、電池を使って電池の残量を気にするのもステキだとは言えませんよね。ここでEnOceanの開閉センサ（マグネットコンタクトセンサ）を使えば、トイレの中の照明から電力を得て、無線でデータを送るということも実現できます。

EnOceanの無線

EnOceanの無線通信はサブギガ（1GHzよりも低い周波数帯）で通信を行うため、Bluetoothや無線LANなどで多く使われている2.4GHz帯の影響を受けずに使うことができます。たとえば展示会などでIoTのデモを行う場合、Bluetoothや無線LANを使うと2.4GHz帯は混み合うため当日に思ったように通信ができないことが多くあります。一方でサブギガは、国や地域によって使用できる周波数帯が異なるために、通信モジュールも異なるものを使わなければなりません。EnOceanの国や地域に応じた周波数帯や、通信モジュールの型番の一例を**表1**に示します。

EnOceanの無線通信プロトコルは、ERP（EnOcean Radio Protocol）[注1]というEnOcean独自のものが使われています。EnOceanの送受信機とマイコンの間は非同期シリアル（UART）で接続するのですが、ここではESP（EnOcean Serial Protocol）[注2]が定められています。また、機器ごとのプロファイル（センサの値とデータの

注1） http://www.enocean.com/erp1/
注2） http://www.enocean.com/esp

▼表1 国や地域に応じた周波数帯や、通信モジュールの型番

地域・国	周波数帯	USB受信モジュール	温度センサモジュール
アメリカ合衆国 カナダ	902MHz	USB300U	STM332U
日本	928MHz	USB400J	STM431J
ヨーロッパ 中国	868MHz	USB300	STM330

関係)は、EEP(EnOcean Equipment Profile)[注3]に記されています。

エナジーハーベスティング

次は、電池の代わりにEnOceanのセンサや無線に電力を供給する、エナジーハーベスティングの方法です。先ほど紹介したように太陽電池が最もわかりやすい方法ですが、EnOceanにはスイッチを押すときの動きで発電(電磁誘導と言います)をして操作を伝えるスイッチもあります。ほかにもエナジーハーベスティングには、ペルチェ素子を用いて温度差で発電する方法や、ピエゾ素子を用いて振動から発電をする方法などもありますが、EnOceanの既存のモジュールとしては販売されていません。

こういった方法で収穫できるエネルギーの量は限られているため、無線通信を行えるだけのエネルギー(電力)が一度に収穫できるとは限りません。常に無線通信を行うのに十分な電力を収穫し続けるには、たとえば太陽電池のパネルを大きくするなど、発電するデバイスを大きくして収穫量を増やさなければならないのです。このため、エナジーハーベスティングでは、少しずつ収穫できるエネルギーを蓄えておき、たまに無線通信を行うといったことをする必要があったりします。

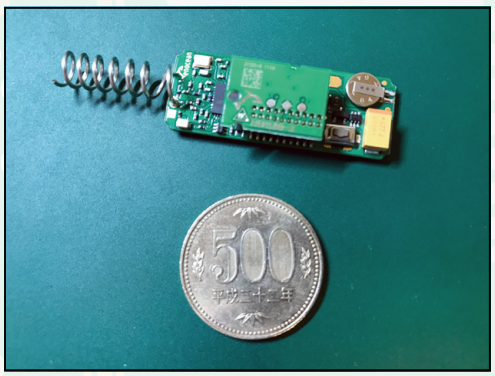
注3) http://www.enocean-alliance.org/eep/

使ってみる

では、今回はEnOceanの温度センサと湿度センサを使って測った温湿度を、mbedで受信してみたいと思います。今回は、温度センサモジュール(STM431J)に湿度センサモジュール(HSM100)を組み付け(**写真1**)樹脂ケースに納めたものを、ローム株式会社から借用しました(**写真2**)。ローム[注4]は、EnOcean規格の推進団体であるEnOcean Allianceのプロモーターです。このセンサモジュールから送信されたデータを、USB 400JというUSB接続タイプのモジュールを使って受信してみました(**写真3**)。USB400Jは、USBホスト機能のあるmbed LPC1768とApplication boardを使って接続します。mbed

注4) http://www.rohm.co.jp/web/japan/enocean

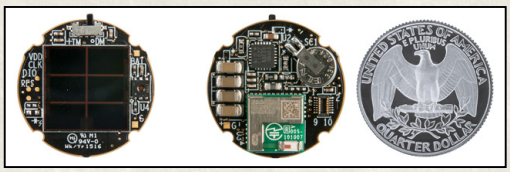
▼写真1 STM431JとHSM100



コラム エナジーハーベスティングでBLE

EnOcean GmbHもサブギガだけでなく、Bluetooth Low Energy(BLE)など2.4GHzの無線を使う機器を販売しています。一方で、エナジーハーベスティングを行える機器は他社も販売していて、たとえばアメリカのCypress Semiconductor Corp.は、コインサイズの電池レスBLE温湿度センサを「CYALKIT-E02 Solar-Powered BLE Sensor Beacon Reference Design Kit」として発売しています(**写真A**)。これは同社のBLEマイコンと、エナジーハーベスティング用のPMIC(電源管理IC)を組み合わせた製品です。こんな小さな太陽電池で集めた電力で、BLEのビーコンとして動作をさせることができます。

▼写真A CYALKIT-E02 Solar-Powered BLE Sensor Beacon Reference Design Kit



▼写真2　温度センサモジュールと湿度センサモジュールを樹脂ケースに納めたもの

LPC1768にはEthernetがありますので、これを使ってMQTTなどでクラウドにデータ送信をすることもできますが、今回は受信したデータをApplication boardについている液晶に表示してみました(**写真4**)。

USB400Jを選択したのは、mbedにはすでにUSB400Serial[注5]というドライバがあるからです。このドライバを使って受信したEnOceanのシリアル信号の構造は、ESPのドキュメントに記載されています。ドキュメント「EnOcean Serial Protocol 3 (ESP3) V1.30[注6]」の13ページ、Table 2に記載されていました(**表2**)。これを参照し、受信したパケットを読み解くと、**図1**のような構造であることがわかります。Packet

注5)　https://developer.mbed.org/users/nanashino/code/USB400Serial/

注6)　https://www.enocean.com/fileadmin/redaktion/pdf/tec_docs/EnOceanSerialProtocol3.pdf

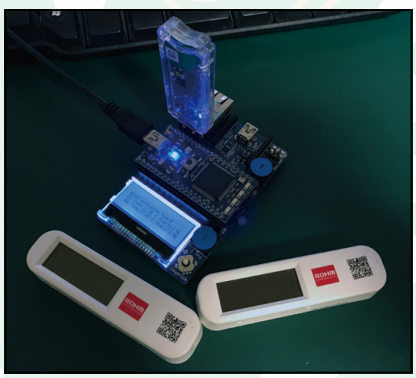
▼写真3　USB400J

Typeが0x0Aですので、ESPのドキュメントの76ページに「Packet Type 10: RADIO_ERP2」として記載されていることがわかりました。掲載されている図(**表3**)によると、Optional Dataには電波の強度が入っているようです。

さて、「Data Length」も0x0A、つまり10byteですので、ERPのドキュメントの「Data contents for Length > 6 Bytes」の項を参照しました。「Data」は0x22から始まりますので、ここに掲載されている表を使って、0x22、つまり0b0010 0010を解釈すると、「001: Originator-ID 32 bit; no Desti nation-ID」、「0: No extended header」、「0010: 4BS telegram(0xA5)」であることがわかりました。

「Data」の中身については、EEPを参照します。A5ですので、EEPのドキュメントのうち4BS Telegramの温湿度センサ(04)には、01〜03の3種があることがわかります。3種のうちどれだろうと思いSTM431Jのデータシートを見ると、HSM100と併用した場合、A5-04-01、

▼写真4　動かしてみたところ



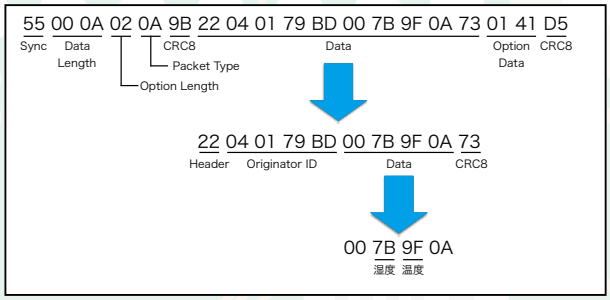
▼表2　パケットの構造(ESP3 V1.30 P.13より引用)

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	Serial synchronization byte; always set to 0x55
Header	1	2	Data Length	0xnnnn	Specifies how many bytes in DATA must be interpreted
	3	1	Optional Length	0xnn	Specifies how many bytes in OPTIONAL_DATA must be interpreted
	4	1	Packet Type	0xnn	Specifies the packet type of DATA, respectively OPTIONAL_DATA
-	5	1	CRC8H	0xnn	CRC8 Header byte; calculated checksum for bytes: DATA_LENGTH, OPTIONAL_LENGTH and TYPE
Data	6	x	... ...	... ...	Contains the actual data payload with topics: - RawData (e.g. 1:1 radio telegram) - Function codes + optional parameters - Return codes + optional parameters - Event codes x = variable length of DATA / byte number
Optional Data	6+x	y	... ...	... ...	Contains additional data that extends the field DATA; y = variable length of OPTIONAL_DATA
-	6+x+y	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

A5-10-10、A5-10-12の3種が書いてあります。このうち、今回はA5-04-01であることがわかります。EEPによると、湿度は0～100%を0～250で表すため、得た値に0.4を乗じて値を取りだし、温度は0～100℃を0～250で表すので0.16を乗じました。パケットの構造さえわかってしまえばEnOceanは簡単に使えます。

このドライバを使って書いてみたコード[注7]のうち、main.cpp(**リスト1**)の118行目や123行目で、表示をするセンサのIDを限定しています。これはOriginator-IDという4byteのセンサのIDの最終byteを使用しています。4byteまるごと確認したほうがよいのでしょうが、手元にあったセンサのIDの最終byteが重複していなかったため、簡易的にこうしました。

▼図1　EnOceanのパケットを読んでみた



▼表3　Packet Type 10の構造(ESP3 V1.30 P.76より引用)

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	Variable length of radio telegram
	3	1	Optional Length	0x02	2 fields fixed
	4	1	Packet Type	0x0A	RADIO_ERP2 = 10
-	5	1	CRC8H	0xnn	
Data	6	x	Raw data	... ...	ERP2 radio protocol telegram without the first Length byte. For sending the ERP2 protocol CRC8 byte can be set to any value. x = Data Length
Optional Data	6+x	1	SubTelNum	0xnn	Number of sub telegram; Send: 3 / receive: 1 ... y
	7+x	1	dBm	0xnn	Send case: FF Receive case: best RSSI value of all received sub telegrams (value decimal without minus)
-	8+x	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

まとめ

無線通信では、みなさんの関心はやはり通信可能な距離に集中するでしょう。EnOceanのWebサイト[注8]には、「通信距離：建物内では30m、自由空間では300mまでカバーします。」と記されています。筆者は2015年のMaker Faire TokyoでEnOceanを使ったデモを行ってみたのですが、このとき会場内の約60m離れた場所に設置した温度センサモジュール(STM431J)からのデータをUSB400Jで受信できました。Maker Faire Tokyoの会場はとても混み合い、2.4GHz帯も混み合い、無線LANやBluetoothの通信に支障が出たりしますが、こうした混雑する会場内で60mも離れて通信ができれば十分だと言えるでしょう。まだ筆者はEnOceanのスイッチや開閉センサを使っていないのですが、これらも試用して経験を積んでみたいと思います。SD

注7)　https://developer.mbed.org/users/ytsuboi/code/USB400J_app_board_demo/

注8)　https://www.enocean.com/jp/technology/radio-technology/

▼リスト1　main.cpp(抜粋)

```
(..前略..)
118     if (sensor == 0x34) {
119         lcd.locate(0,12);
120         lcd.printf("%02X Temp:%2.2f C Hum:%3.1f %%\r\n", sensor, temperature*0.16,
 humidity*0.4);
121     }
122
123     if (sensor == 0x52) {
124         lcd.locate(0,21);
125         lcd.printf("%02X Temp:%2.2f C Hum:%3.1f %%\r\n", sensor, temperature*0.16,
 humidity*0.4);
126     }
(..後略..)
```

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7月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートの回答は誌面作りのために集計いたしますが、それ以外の目的ではいっさい使用いたしません。記入いただいた個人情報は、作業終了後に責任を持って破棄いたします。

01 Wi-Fiホームルータ「Aterm WG1200HP2」 1名

高機能Wi-Fiホームルータ。以前使っていたルータから設定を引き継げる“Wi-Fi設定引越し”機能、スマホのアプリからNFCを使った初期設定ができる“らくらく「かざして」スタート”機能を搭載しています。無線LANの規格はIEEE802.11ac/n/a（5GHz帯）と11n/g/b（2.4GHz帯）。5GHz帯での実効スループットは約605Mbps（UDP）／約520Mbps（TCP）となっています。

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

02 USBメモリ「600-3TC16GN」 2名

「USB Aコネクタ（USB3.0対応）」「USB Cコネクタ（USB3.1/Gen 1対応）」を両方搭載し、パソコンやスマホに合わせて切り替えられるフラッシュメモリ（16GB）です。本体5g・ストラップホール付き・ネックストラップ付属と、持ち運びに便利。

提供元 サンワサプライ　https://direct.sanwa.co.jp/

03 すごーい！オライリーオリジナル サーバルTシャツ 3名

『Javaパフォーマンスチューニング第2版』のカバーを飾った動物「サーバル」のイラストがプリントされたTシャツ。サーバルをはじめとした野生動物が活躍するアニメーション作品の人気急騰を受け、イベントなどで販売されました。MまたはLサイズをプレゼント。

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

04 プログラミング道への招待

竹内 郁雄 著

プログラミングに興味のある学生、初心を忘れてしまった大人を対象にした、コンピュータとプログラミングのしくみを解説する読み物。「プログラミングを楽しむ」ことを念頭に、ユーモラスに語ります。

提供元 丸善出版
http://pub.maruzen.co.jp　2名

05 純粋関数型データ構造

Chris Okasaki 著

関数型言語で書かれたプログラムにとって効率的なデータ構造について学ぶ1冊。「Standard ML」を使い、データの永続性や遅延評価といった関数型言語の特徴に合ったデータ構造について考察します。

提供元 アスキードワンゴ
http://asciidwango.jp　2名

06 Java本格入門

谷本 心 ほか 著

保守性・堅牢性・性能・開発効率といった現場でのJava開発で大切なことに重点を置き、文法からオブジェクト指向やデザインパターン、ドキュメンテーション、品質への配慮まで解説します。Java 8対応。

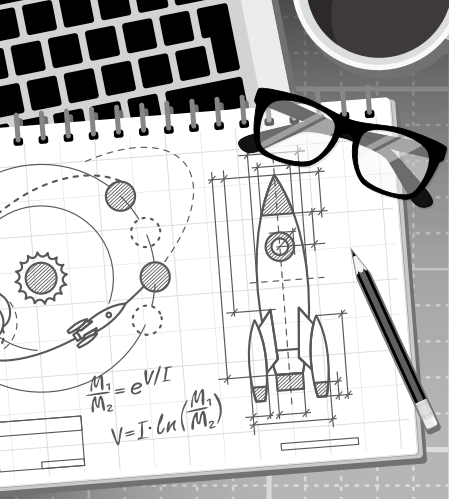
提供元 技術評論社
http://gihyo.jp　2名

07 Intel Edisonマスターブック

北神 雄太 著

乾電池で動作する省電力性ながら、無線LAN・Bluetooth・高性能プロセッサを搭載したIntel製のコンピュータモジュールEdisonについて、Lチカからセンサーデータの活用方法までカバーする1冊です。

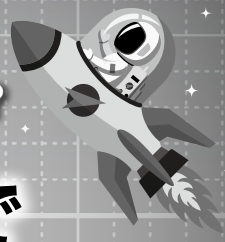
提供元 技術評論社
http://gihyo.jp　2名



第1特集

もっとbashを使いこなしませんか?

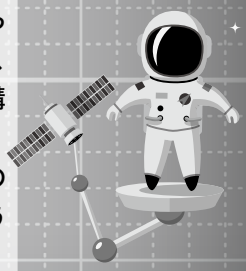
理論&応用でシェル力(りょく)の幅を広げる

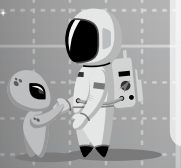


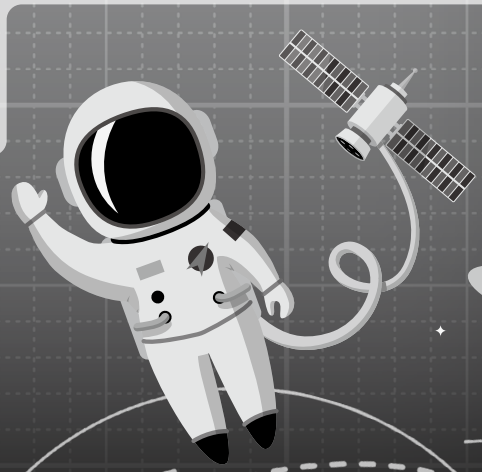
開発でもインフラ管理でも運用でも、シェル上で行う作業はたくさんあります。だからこそ、シェルの操作が少し上達しただけで、いろいろな作業が速く、楽にできるようになります。本特集は、初心者もベテランも、ご自身のレベルに合わせて実力アップを図れるように、2つのパートで構成しています。

理論編(第1〜3章)では、設計思想やしくみをふまえながらシェルの使い方や機能を解説します。今までシェルをなんとなく使ってきたという人は、まずはここでおさらいを。

応用編(第4〜5章)では、熟練者がさらにシェルの活用範囲を広げられる知見を紹介します。







第1章 理論編 … 1

シェル初心者から中級者への次の一歩

シェルのもっともシンプルな使い方である単純コマンドを使いこなすために、リダイレクト、パイプ、環境変数の影響範囲はきちんと理解しておきましょう。そしてシェルスクリプトを書くための準備として「コマンド→パイプライン→リストの循環」という構造を知り、ifやforなどを使った複合コマンドの基本を習得してください。

Author 山森 丈範（やまもり たけのり）
Mail yamamori@kt.rim.or.jp

基本のおさらい

シェルとは

UNIX系OSにテキストモードでログインしたとき、xtermなどの端末エミュレータを開いたとき、またはほかのホストからsshなどでリモートログインしたときに目にするのがシェルのプロンプトです。ユーザはシェルのプロンプトが表示されたコマンドラインにコマンドをタイプします。シェルは、ユーザが入力したコマンドを解釈し、該当するコマンドを実行します。

このようにシェルはユーザがコマンド入力するために必須の存在ですが、シェルにはシェルスクリプトを実行するというもう1つの役割があります。つまりシェルには、ユーザ入力のコマンドを逐次実行するインタラクティブ（会話）モードと、シェルスクリプトを実行するノンインタラクティブ（非会話）モードの2つの使用法があることになります。

シェルの会話モードも非会話モードも基本的には同じです。ユーザがシェルのプロンプト上にタイプしている文字列をそのままファイルに保存すれば、それだけでもシェルスクリプトになります。ただし、シェルスクリプト実行時にはプロンプトは表示されず、ジョブコントロールは行われず、Tabキーなどによるファイル名の補完も行われないなどの違いはあります。

シェルの種類

シェルには、bashなどのBシェル（Bourne Shell）系のシェルのほかに、csh（tcsh）というCシェル系が存在します。history、alias、ジョブコントロールなどのコマンドライン上で便利な機能はもともとcshで導入されたものでした。このため、csh（tcsh）をログインシェルとして使用していたユーザも多かったと思います。

cshでもシェルスクリプトは記述できますが、cshのシェルスクリプトは不都合な点が多いため推奨されていません。コマンドライン上でcshを使用しているユーザは、cshのコマンドラインをそのままファイルに保存しても、文法の違いで/bin/shのシェルスクリプトにはなりません。シェルスクリプト記述時にはあらためて/bin/shを使うように頭を切り替える必要があります。

しかし、bashの登場によりhistory、alias、ジョブコントロールなどの機能がコマンドライン上で使えるようになり、ログインシェルもシェルスクリプト記述も、どちらもbashだけで行えるようになりました。Linuxの多くのディストリビューションやmacOSの標準シェルがbashになっているため、新規のユーザはそのまま

bashユーザになることが多いでしょう。また、bash以外の高機能シェルとしてzshを使用しているユーザも少なくないことと思いますが、zshもBシェル系のシェルであり、bashもzshも含めて、コマンドライン上の文法は現在ではBシェル系に統一されたと言っていいでしょう。

もとcsh系ユーザが戸惑う例の1つは環境変数の設定だと思います。Bシェル系にはsetenvコマンドはないため、**図1**のようにシェル変数をexportする形で環境変数を設定します。bashなどの多くのシェルでは変数の代入とexportを同時に行えますが、従来のBシェル系シェルとの互換性を重視する場合は**図1-a**のように変数の代入とexportを別々に行ったほうがいいでしょう。

単純コマンドとは、リダイレクトやパイプの使い方

単純コマンド

いわゆる普通のコマンドのことをシェルの文法上では単純コマンドと呼びます。ls、cp、rmなどの/binや/usr/binなどにインストールされた外部コマンドはもちろん、cd、export、read、shiftなどのシェル組み込みコマンド（ビルトインコマンド）も単純コマンドです。単純コマンドは、コマンド名と任意の個数の引数の並びで構成され、コマンド名と各引数はスペースまたはTABで区切られます（bashなどの場合はTABはファイル名補完用に利用するため、区切りにはスペースを使います）。

リダイレクト

単純コマンドと、後述する構文などの複合コマンドを併せた「コマンド」は、その標準入力、標準出力、標準エラー出力、さらに任意のファイル記述子をリダイレクトできます。

リダイレクトの書式は、標準入力にファイルを入力する場合は< fileで、標準出力をファイルに書き込む場合は> file、標準エラー出力の場合は2> fileとなります。このほか、すでに存在するファイル内容を削除せずに追記（アペンド）したい場合は>> fileや2>> fileが使えます。さらに、3番以降の任意のファイル記述子も、3> fileや4< fileなどのように使うことができます。

図2のように、1行程度の簡単なファイルを作成する場合は、テキストエディタを開かなくても、echoコマンドの標準出力をファイルにリダイレクトすることで作成できます。サイズゼロのファイルを作成したい場合は、コマンドを省略し、単にリダイレクトだけを実行すればOKです。

▼図1　環境変数CCをgccに設定する例

```
$ setenv CC gcc                          ←setenvコマンドを使おうとすると
bash: setenv: command not found          ←エラーになる
$ export CC=gcc                          ←bashなどでの環境変数の設定方法
$ CC=gcc; export                         ←従来のBシェル系シェルでも使える方法（a）
$ printenv CC                            ←環境変数CCの値を確認
gcc                                      ←gccに設定されている
```

▼図2　リダイレクトを使ったファイル作成

```
$ echo Hello > file                      ←echoコマンドをリダイレクトしてファイル作成
$ cat file                               ←作成したファイルの中身を表示
Hello                                    ←中身を確認
$ > file                                 ←コマンドなしのリダイレクトでサイズゼロのファイル作成
$ cat file                               ←作成したファイルの中身を表示
$                                        ←何も表示されないがファイルは存在している
```

▼図3　catコマンドの引数の途中にリダイレクトを記述

```
cat file1 < file2 file3 - file4          ← < file2の部分がリダイレクト
```

リダイレクト記号の記述位置

リダイレクト記号はコマンドの右端に記述するのが普通ですが、単純コマンドの場合は実はどこに記述してもよく、引数と引数の間に記述したり、さらにはコマンド名の左側に記述するのもOKです。**図3**のcatコマンドの例では、< file2の部分が標準入力のリダイレクトで、残りの「file1、file3、-、file4」がcatコマンドに引数として渡されます。catコマンド自身にはfile2のファイル名は渡されま

せん。catコマンドの引数の-は標準入力を表すため、結局、file1、file3、file2、file4の順で連結されて標準出力に出力されることになります。

ファイル記述子のリダイレクト

リダイレクトはファイルに対してだけでなく、ファイル記述子に対しても行えます。よく使うのが2>&1です。makeコマンドのログを、標準出力も標準エラー出力もまとめてファイルに書き込みたい場合はmake > logfile 2>&1とします。ここで、2>&1は、標準出力のファイル記述子1番を、標準エラー出力のファイル記述子2番として複製する、という動作になり、> logfileよりも右側に記述する必要があります。この記述は感覚的に左右の順番を逆に考えやすいため注意してください。

ほかによく使うのが1>&2です。シェルスクリプト中でエラーメッセージを標準エラー出力に出力したい場合、echo Error 1>&2のように記述します。ここでは標準出力の1の記述は省略できるため、echo Error >&2としてもかまいません。

パイプ

単純コマンドはもちろん、構文などの複合コマンドも併せた「コマンド」はパイプを使うことができます。パイプは、テンポラリファイルを作らずに、あるコマンドの標準出力を別のコマンドの標準入力に接続できる効率の良いしくみです。

何らかのコマンドの出力が長過ぎて、画面表示が流れてしまって読めない場合には、コマンド | lessをよく使います。標準エラー出力も同時にパイプに通したい場合はコマンド2>&1 | lessとします。

図4はカレントディレクトリのファイル数を数えている例です。lsコマンドの出力をwcコマンドで数え、-lオプションでその行数だけを表示させています。lsコマンドには、ドット(.)で始まるファイルも含めて表示させるため、-Aオプションを付けています(.と..を除くため、-aではなく-Aとしています。さらに、ファイル名の中に改行が含まれているような場合にも対応するため、-qオプションを付けています。

ところで、lsコマンドの出力をパイプに通してls | lessを実行すると、表示が1行1ファイルのみの形式に変わってしまいます(**図5**、**図6**)。これは、lsコマンド自身がその標準出力が端末か端末以外(パイプまたはファイル)かを判断して、出力形式を変えるようになっているためです。画面出力と同じ表示をそのままパイプに通したい場合は**図7**のようにls -C | lessとします。GNU版lsでカラー表示を行っている場合に、パイプとlessコマンドを通してカラー表示をするには、ls -C --color | less -rのようにオプションを追加するといいでしょう。lessの-rオプションは、カラー表示のエスケープシーケンスをそのまま通すために必要です。

環境変数の一時変更

単純コマンドは、そのコマンドのみ環境変数を一時的に変更して実行できます。これは構文

▼図4　カレントディレクトリのファイル数を数える

```
$ ls -Aq | wc -l   ←パイプでwcコマンドにつないでlsの出力の行数を数える
167                ←ファイル数が表示される
```

▼図5　lsコマンドを単独で実行

```
$ ls
bin   cdrom  etc   lib    local  mnt  proc  run   srv   sys  usr
boot  dev    home  lib64  media  opt  root  sbin  swap  tmp  var
```

などの複合コマンドではできない文法です。シェルによってはシェル関数の環境変数の一時変更も可能ですが、うまく動作しない場合があります。

環境変数の一時変更は、単純コマンドの左側に、そのまま環境変数の代入を記述します。たとえば環境変数LANGをja_JP.UTF-8などの日本語環境に設定している人が、一時的に英語環境でコマンドを実行したい場合には、コマンド名の直前にLANG=Cを付加して実行します。LANG=C dateなら英語での日付時刻表示、LANG=C man lsなら英語でのマニュアル表示になります[注1]。

図8のように、単純コマンドの実行後に環境変数を表示してみると、もとの値のまま変化していないことがわかります（**図8-a**）。同様のことは、シェルの文法を使わずに外部コマンドのenvを使って行うこともできます（**図8-b**）。複数の環境変数を同時に一時変更することも可能で、その場合は環境変数の代入文を横に並べて記述します（**図8-c**）。いずれの場合も、環境変数の代入文のあとにセミコロン（;）を入れないように注意してください。

注1）言語環境の環境変数はLANG以外のLC_ALL、LC_CTYPE、LC_MESSAGES、LC_TIMEなども関係するため、これらを設定している場合は適宜対応が必要。

コマンドを組み合わせるための多彩な方法

コマンド→パイプライン→リストの循環

シェルスクリプトの文法には、コマンド→パイプライン→リスト→コマンドという循環構造があります。単純コマンドと複合コマンドを併せた「コマンド」は、パイプでつないでパイプラインを作り、複数のパイプラインは「リスト」を構成します。この「リスト」は複合コマンドの中で使うと「コマンド」になります。

「リスト」というのは、たとえばif文のifとthenの間とか、thenとfiの間などに記述するもののことです。実際には、if [$# -lt 3]; then ……のようにtestコマンドの別名の[コマンドを記述することが多いですが、文法上はifとthenの間に記述できるのはリストです。リ

▼図6　lsの出力をパイプに通すと1行1ファイルの表示に変わってしまう

```
$ ls | less
bin
boot
cdrom
dev
etc
home
lib
lib64
local
media
mnt
opt
proc
root
run
sbin
srv
swap
sys
tmp
usr
var
(END)  ←lessのプロンプト
```

▼図7　ls -Cにするとパイプ経由でも画面と同様の表示

```
$ ls -C | less
bin   cdrom  etc   lib    local  mnt  proc  run   srv   sys  usr
boot  dev    home  lib64  media  opt  root  sbin  swap  tmp  var
(END)  ←lessのプロンプト
```

▼図8　環境変数を一時的に変更

```
$ printenv LANG                    ←環境変数LANGを表示
ja_JP.UTF-8                        ←UTF-8の日本語環境
$ LANG=C printenv LANG             ←LANG=Cに一時変更してみる
C                                  ←Cに一時変更されている
$ printenv LANG                    ←再度LANGの値を確認
ja_JP.UTF-8                        ←もとのまま変更されていない(a)
$ env LANG=C printenv LANG         ←envコマンドを使う方法もある(b)
C                                  ←Cに一時変更されている
$ printenv LANG                    ←再度LANGの値を確認
ja_JP.UTF-8                        ←もとのまま変更されていない

↓複数の環境変数を一時変更(c)
$ LANG=C HOME=/home/guest printenv LANG HOME
C
/home/guest
```

ストにはパイプラインも単純コマンドも含むため、単純コマンドの1つである[コマンドを記述することができるわけです。そして、if文全体が1つの複合コマンドになるため、このif文をほかのfor文、while文やcase文、あるいは別のif文の中のリストとして記述し、構文をネスティングすることができるのです。

以上をまとめると**図9**のようになります。

&&リスト、||リスト

リストには、パイプライン同士でANDやORの論理演算を行う&&リスト、||リストと呼ばれるものがあります。「&」が1個のバックグラウンドや「|」が1個のパイプとは違うので注意してください。

&&リスト

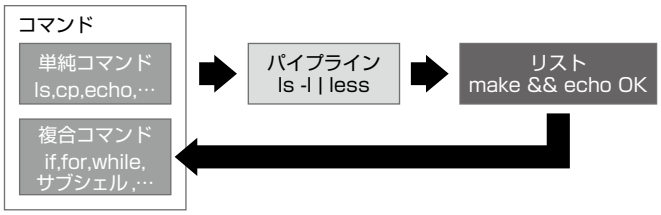
&&リストはパイプライン1 && パイプライン2のように記述し、パイプライン1とパイプライン2の両方が真（終了ステータス0）の場合に全体の真偽値が真になります。実際にはパイプライン1が先に実行され、この終了ステータスが0以外（偽）の場合はその時点で全体の真偽値が偽であると確定してしまうため、パイプライン2は実行されません。この性質を利用し、&&リストは簡単なif文の代わりに利用されます。

図10はcmpコマンドで2つのファイルを比較し、結果が真（ファイル一致）の場合にOKを表示させるというものです（cmpコマンド単独ではファイル一致時にはメッセージは表示されません）。この例のファイル名を見てわかるとおり、ISOイメージファイルとCD-ROM（またはDVD）のデバイスファイルを比較し、ディスクのベリファイを行う場合などに便利です。

||リスト

||リストはパイプライン1 || パイプライン2のように記述し、パイプライン1とパイプライン2のどちらかが真（終了ステータス0）の

▼図9　コマンド→パイプライン→リストの循環



●コマンド：
単純コマンドと複合コマンドを併せてコマンドと言う。コマンドはリダイレクトすることができる。

●単純コマンド：
ls、cp、rmなどの外部コマンド、echo、cd、testなどの組み込みコマンド。引数やリダイレクトが付いていてもよく、環境変数の一時変更も可能。

●パイプライン：
`ls -l | less`のように、1個以上のコマンドを0個以上のパイプ記号|でつないだもの。コマンド1個だけでパイプがなくてもパイプラインと言う。

●リスト：
`zcat file.gz | grep -q word && echo OK`のように、1個以上のパイプラインを0個以上の「;、&、&&、||」の記号でつないだもの。リストの最後は通常、「;、&、改行」のどれかで区切る。パイプライン1個だけでも、パイプなしのコマンドを&&などでつないだものでも、&&などもなしでコマンド1個だけでもリストと言う。

●複合コマンド：
if文、case文、for文、while文などの構文、サブシェル、グループコマンドなど。複合コマンドの中でリストを記述できる。複合コマンドと単純コマンドと併せて「コマンド」になる。

場合に全体の真偽値が真になります。実際にはパイプライン1が先に実行され、この終了ステータスが0（真）の場合はその時点で全体の真偽値が真であると確定してしまうため、パイプライン2は実行されません。つまり&&リストとは逆で、パイプライン1が偽の場合のみパイプライン2が実行されます。この性質を利用し、||リストも&&リストと同様に簡単なif文の代わりに利用されます。

シェルスクリプトの行頭を#!/bin/sh -eと記述して起動している場合、またはシェルスクリプト中でset -eコマンドを実行している場合、シェルスクリプト中の各リストの終了ステータスが0以外になるとその時点でシェルスクリプトを終了するようになります。この-eオプションは、シェルスクリプト中に記述したコマンドのどれか1つでもエラーになればすぐに実行を打ち切りたい場合に便利です。この状態で、あるコマンドだけはエラーを無視し、たとえエラーになっても処理を続行したい場合に||リストが便利です。

リスト1は、rmdirコマンドを使ってディレクトリを削除していますが、たとえ削除できなかった（もともとディレクトリが存在していなかった）場合でもエラーにしたくない場合の例です。コマンドの後ろに|| trueを付けることにより、このリスト全体の終了ステータスは常に0になり、エラーとしては扱われません。なお、trueの代わりに:と記述することもできます。:は何もしないヌルコマンドで、0の終了ステータスを返すため、trueコマンドと同じように使うことができます。

この方法はMakefile中に記述するコマンドのエラーを無視させたい場合にも使えます。

複合コマンド（構文、サブシェル、グループコマンド、シェル関数）

if文

シェルスクリプトのif文は、ifの直後のリストとしておもにtestコマンドの別名の[コマンドを記述し、その真偽によってthenの直後またはelseの直後のリストを実行します。if文の終了はfiです。条件判断が複数あってelse ifの形になる場合はelifのキーワードを使うことができます。

リスト2は、シェルスクリプトが起動されたときの引数の個数を特殊パラメータ$#を参照してチェックし、引数が2個未満の場合はUsage（使用方法）を表示して終了する例です。この例のexitの直後のelseは最適化が可能で、elseの代わりにfiでif文を終了し、そのあとに実際の処理を続けたほうが効率がいいでしょう。

case文

case文は、caseの直後に記述された文字列を、パターンごとに場合分けして、各パターンに対応するリストを実行します。caseの直後には、シェル変数またはコマンド置換を記述するのが普通です。各パターンはその右側に閉じる括弧）を付け、各パターンに対応するリストはその最後にセミコロン2つ;;を付けます。case文の

▼図10　cmpコマンドの結果が真ならOKを表示

```
$ cmp file.iso /dev/cdrom && echo OK   ←ファイル一致を&&リストで判定
OK                                     ←ファイルアクセス後、OKが表示される
```

▼リスト1　||リストを使ってエラーを無視する方法

```
#!/bin/sh -e                    ← -eオプション付きのシェルスクリプト

rmdir ディレクトリ || true     ←ディレクトリを削除できなくてもエラーにしたくない
または
rmdir ディレクトリ || :        ←trueコマンドの代わりに「:」コマンドも使える
```

最後はcaseのスペルを逆にしたesacを記述して終了します。

リスト3はOSの種類ごとに場合分けし、各OSのlsコマンドでファイルのタイムスタンプを秒以下（または秒単位まで）まで詳細に表示する例です（lsコマンドのオプションはOSによって異なります）。ここではcaseの直後にunameコマンドをコマンド置換としてバックォート｀｀で囲んで記述しており、これを各OS名のパターンにマッチさせます。Solarisの場合、uname -sの出力がSunOSになるため、バージョン名も含まれるようにuname -srの出力を利用します。「SunOS 5.11」のような出力になればSolarisです。FreeBSD、NetBSD、OpenBSDについては、パターンを|記号でつないで記述し、OR条件を表現しています。パターン上のアスタリスク*は任意の文字列に一致するため、Linux、FreeBSDなどのバージョン数値は何であっても一致し、Solarisの場合はメジャーバージョン5のみチェックしていることになります。SunOSと5の間にスペースがあるため、パターン上ではシングルクォート' 'で囲みます。

なお、シェルスクリプトの場合分けの処理は、if文でもcase文でも記述できることがありますが、文字列の場合分けについてはcase文を使うほうがパターンにOR条件やアスタリスクなどが使えるため効率がいいでしょう。

for文

シェルスクリプトのfor文は、そのループ変数に代入する値が規則的な数列でなくてもかまわないのが特徴です。ループ変数に代入したい文字列を羅列しておけば、これらが順番に代入されながらfor文のdo～doneのループが実行されます。代入する文字列としてパス名（ファイル名）に展開される*などを使うことができ、for file in *と記述すればカレントディレクトリに存在する各ファイル（ただしファイル名がドットで始まるファイルを除く）ごとにループすることができます。**リスト4**は、do～doneのリストとしてcpコマンドを実行し、各ファ

▼リスト2　if文でシェルスクリプトの引数の個数をチェック

```
#!/bin/sh

if [ $# -lt 2 ]                          ←引数の個数が2個未満の場合
then
  echo "Usage: $0 file1 file2" 1>&2       ←Usage:を標準エラー出力に出力
  exit 1                                  ←終了ステータス1で終了
else
  …実際の処理…                           ←ここに実際の処理を記述
  :
fi
```

▼リスト3　case文でOSごとに場合分けし、lsで詳細タイムスタンプを表示

```
#!/bin/sh

case `uname -sr` in                   ←unameコマンドの出力文字列で場合分け
  Linux*)                             ←Linuxの場合（バージョン数値は任意）
    ls -l --full-time "$@";;          ←Linux（GNU）版のlsコマンド
  FreeBSD*|NetBSD*|OpenBSD*)          ←FreeBSDなどの場合（OR条件使用、バージョン任意）
    ls -lT "$@";;                     ←BSD版のlsコマンド
  SunOS' '5.*)                        ←Solarisの場合（SunOS 5.*として判定）
    ls -E "$@";;                      ←Solaris版のlsコマンド
  *)                                  ←それ以外の場合
    echo unknown OS 1>&2;;            ←エラーメッセージを表示
esac
```

▼リスト4　カレントディレクトリの各ファイルをバックアップ

```
#!/bin/sh

for file in *          ←カレントディレクトリのファイル名でループ
do
  cp -p "$file" "$file".bak  ←各ファイルをバックアップ
done
```

イルをバックアップする例です。

このほかfor arg in "$@"という記述もよく用いられ、シェルスクリプト起動時に付けられた各引数（位置パラメータ）ごとにループすることができます。なお、このin "$@"という記述は実は省略でき、単にfor arg; do～doneと書けば各引数ごとにループするfor文になります。

for文でループ変数を規則的な数列にしたい場合、たとえば単純に変数に1から10までの数値を代入してループしたい場合は、bashやzshの場合は連番のブレース展開を使って**リスト5**のようにするのが簡単でしょう。連番のブレース展開が使えないシェルの場合、Linuxなどseqコマンドが使える環境では**リスト6**のようにseqを利用できます。FreeBSDなど、jotコマンドが使える環境では**リスト7**のようになります。seqもjotもなく、シェルが従来の/bin/shの場合、少々技巧的になりますが**リスト8**のような方法もあります。ここではバックォート｀｀の中のコマンド置換でsetコマンドを使って位置パラメータの個数（$#）をカウンタ代わりに利用していますが、コマンド置換

▼リスト7　jotコマンドを使った数列ループ（FreeBSDなど）

```
for i in `jot 10`  ←jotを使う（`jot - 1 10`でも良い）
do
  echo $i          ←試しにループ変数を表示
  :                ←ここに処理本体を記述する
done
```

▼リスト5　連番のブレース展開を使った数列ループ（bash、zsh用）

```
↓連番のブレース展開で1から10までループ
for i in {1..10}
do
  echo $i          ←試しにループ変数を表示
  :                ←ここに処理本体を記述する
done
```

▼リスト6　seqコマンドを使った数列ループ（Linuxなど）

```
↓seqを使う（`seq 10`でも良い）
for i in `seq 1 10`
do
  echo $i          ←試しにループ変数を表示
  :                ←ここに処理本体を記述する
done
```

全体が暗黙のサブシェルになるため、シェル本体の位置パラメータや$#の値は変化しないので安心です。

while文

while文は、whileの直後に記述したリストが真である限り、do～doneのループを実行します。whileの直後のリストとしては、if文と同様にtestコマンドの別名の[コマンドがよく使われますが、ほかに、readコマンドを使うことも多いです。

リスト9は、/etc/resolv.confを読み込み、その中のnameserverの行に記述されたIPアドレスのみを取り出して表示する例です。readコマンドは標準入力から1行を読み込み、単語に分割して変数にセットしますが、標準入力がEOF（End Of File：ファイル終端）になった場合は終了ステータスが偽になります。このため、while read ……と記述すると標準入力から1行ずつ読み込んでEOFまでループする

▼リスト8　従来の/bin/shの内部コマンドだけで数列ループ（Solaris10以前など）

```
for i in `set X;while [ $# -le 10 ];do echo $#;set "$@" X;done`
do                    ↑setコマンドで$#の値を利用した技巧的な方法
  echo $i             ←試しにループ変数を表示
  :                   ←ここに処理本体を記述する
done
```

ことができます。

ここでは標準入力に/etc/resolv.confファイルをリダイレクトしますが、リダイレクトの記述はwhile文全体のあと、つまりdoneのあとに記述します。

なお、同様の処理はほかのスクリプト言語を使って行うこともでき、たとえばawkを使えば**リスト10**のようになります。しかし、外部コマンドを呼び出さずにシェル内部のwhile文とreadコマンドなどだけを使ってやる方法もあることを覚えておくと応用範囲が広がるでしょう。

サブシェル

リストを( )で囲むとサブシェルになり、サブシェルはもとのシェルとは別扱いで実行されます。サブシェルの中でシェル変数や環境変数を変更したり、カレントディレクトリやumask値などを変更しても、サブシェルを抜けるともとに戻ります。そこで、シェル変数や位置パラメータなどを変化させて何らかの処理を行い、必要な処理が終わったらもとに戻したい場合にサブシェルを使うと便利です。もちろん、サブシェル全体をリダイレクトしたり、パイプに接続したりすることも可能です。

リスト11は、環境変数PATHに設定されているコロン(:)で区切られた複数のディレクトリ名のうち、3番目に設定されているディレクトリ名だけを表示するシェルスクリプトです。シェル変数IFSを:に変更し、:を区切り文字とした状態でsetコマンドを実行すると、:は削除されて、PATHの各ディレクトリが順番に$1、$2、$3、……の位置パラメータにセットされます。ここで$3をechoすれば目的が達成できます。たとえばPATH=/home/guest/bin:/usr/local/bin:/usr/bin:/usr/sbinと設定されていたとすると、3番目の/usr/binが表示されるはずです。サブシェルの中で変更されたIFSと位置パラメータはもちろんもとに戻ります。

サブシェルを1行で記述する場合、次項のグループコマンドとは違って、(の直後にスペースを空ける必要はなく、また、リストの最後はセミコロン(;)なしで括弧を閉じることができます。

サブシェルの中でexitコマンドを実行すると、もとのシェルを終了するのではなく、サブシェルだけが終了になります。exitの引数で終了ステータスを返せるので、これを利用して、1の終了ステータスを返すfalseコマンドと同じコマンドを作ることができます。さらに1だけでなく、0～255までの任意の終了ステータスを返すコマンドをサブシェルで作れます(**図11**)。

▼リスト9　resolv.confのnameserverのIPアドレスを抜き出すスクリプト

```
#!/bin/sh

while read name val comment                    ←1行ずつ読み込んでループ
do
  case $name in nameserver) echo "$val";; esac ←nameserverの行でechoを実行
done < /etc/resolv.conf                        ←リダイレクトはdoneのあとに
```

▼リスト10　awkを使ってnameserverのIPアドレスを抜き出すスクリプト

```
#!/bin/sh

awk '$1 == "nameserver"{print $2}' ➥
< /etc/resolv.conf
```

▼リスト11　PATHの3番目に設定されたディレクトリ名を表示

```
#!/bin/sh

(                  ←サブシェル開始
  IFS=:            ←区切り文字をコロン(:)とする
  set $PATH        ←コロンで区切って位置パラメータにセット
  echo $3          ←3番目の位置パラメータを表示
)                  ←サブシェル終了
```

グループコマンド

リストを{ }で囲むと、グループコマンドと呼ばれる複合コマンドの一種になります。グループコマンド全体が1つのコマンドですので、グループコマンドとしてまとめてリダイレクトしたり、パイプに接続したり、また次項のシェル関数の本体として使用したりすることができます。サブシェルとは違ってグループコマンド内のリストはもとのシェル上で実行され、変数などの変更はすべてグループコマンドを抜けたあとにも影響します。

リスト12は、3個のコマンドの出力を同じlogfileにまとめてリダイレクトしている例です。グループコマンドを使わずに個々のコマンドを個別にリダイレクトする（その場合はアペンドモードで>> logfileとする）よりも記述が簡便になります。

グループコマンドを1行で記述する場合、サブシェルの記述とは違って{ リスト;}のように、{の直後にスペースが必要で、}の前にはセミコロンが必要です。サブシェルとグループコマンドの違いが問題にならない場合は、グループコマンドの代わりにサブシェルを使って記述してもかまいません。

シェル関数

一定の処理を行うコマンドをまとめてシェル関数を定義することができます。シェル関数はシェルスクリプトの中で使えるのはもちろん、コマンドラインで使うこともできます。**リスト13**は、ls -lを実行するようなシェル関数を定義している例で、シェル関数を使ってaliasと同様のことができます。ただし、aliasとは違って引数の受け渡しも明示的に定義する必要があります。シェル関数に対する引数は、位置パラメータ（$1、$2、……）に一時的にセットされ、この値はシェル関数内だけで有効になります。引数をそのまま全部引き継ぐには、"$@"を使います。

このように、シェル関数は関数名() { リスト;}の形で定義します。シェル関数内の変数は、引数渡しのための位置パラメータを除いてグローバル変数のように扱われ、関数内で変更した変数の値は関数からリターンしたあとでも変化したままになります。

関数内の変数をローカル変数のように扱いたい場合は、関数本体の{ }を、( )を使ったサブシェルに変更し、関数名() (リスト)の形で記述すると良いでしょう。実は、関数本体部分は何らかの複合コマンドを1つ記述すればよく、たとえばif文を1つ使って、関数名() if :;then リスト;fiのような変わった定義も文法的には可能です。

シェル関数の終了ステータスは、シェル関数内で最後に実行されたリスト（コマンド）の終了ステータスになりますが、returnコマンドを使って終了ステータスを明示的に返すこともできます。SD

▼図11　サブシェルで任意の終了ステータスを返すコマンドを作る

```
$ (exit 1)          ←サブシェルの中で引数1でexitを実行
$ echo $?           ←終了ステータスを表示
1                   ←終了ステータスは1、falseコマンドと同じ
$ (exit 123)        ←0～255までの任意の終了ステータスを返せる
$ echo $?           ←終了ステータスを表示
123                 ←終了ステータスは123
```

▼リスト12　3個のコマンドの出力をまとめてファイルにリダイレクト

```
#!/bin/sh

{                   ←グループコマンド開始
  hostname          ←ホスト名を出力
  date              ←日時を出力
  who               ←ログインユーザ名を出力
} > logfile         ←以上の出力をまとめてファイルにリダイレクト
```

▼リスト13　シェル関数でllという関数を定義

```
#!/bin/sh

ll()                ←llという関数を定義
{                   ←関数本体はグループコマンドとして記述
  ls -l "$@"        ←ls -lの引数は"$@"ですべて引き継ぐ
}                   ←関数本体の終了
```

第2章 理論編 … 2

シェルスクリプト初心者から中級者への次の一歩

シェルはコマンドラインインターフェースで、インタラクティブ（対話的）に使うだけではありません。コマンドなどをファイルに記述すればシェルスクリプトとして実行できます。ここで意識したいのが、UNIX哲学。この考えに則ってコードを書けば、ほかのコマンドとの連携もできる自作コマンドが作れます。

Author 石山 将来（いしやま まさき）
Twitter @b4b4r07
株式会社メルカリ

基本のおさらい

シェルスクリプトとは

シェルスクリプトとは、普段シェルから打ち込んでいるようなコマンドなどを1つのテキストファイルにまとめたものを指します。

たとえば、ビルドなどの目的に応じた手順（入力する一連のコマンドなど）をスクリプトとして保存し、そのスクリプトを実行することで、上から順番にコマンドがシェルに解釈され実行されていくため、わざわざ1つずつシェルからコマンドを打ちこむ必要がなくなります。

ゆえに、更新作業やビルド処理など定期的に実行する必要がある複数のコマンドセットをひとまとめにシェルスクリプトとして用意しておくなどがよくあるユースケースです。

リスト1に示した例は、

▼リスト1　Go言語をインストールするシェルスクリプト（install-go18.sh）

```
#!/bin/bash
wget "https://storage.googleapis.com/ｱ
golang/go1.8.darwin-amd64.tar.gz"
tar zxf "go1.8.darwin-amd64.tar.gz"
mkdir -p ~/bin
install -m 755 go/bin/go ~/bin
~/bin/go version
```

・Go 1.8の圧縮ファイルをダウンロード
・ダウンロードした圧縮ファイルを展開
・指定したディレクトリを作成
・インストール
・実行（バージョン情報を出力）

を1つにまとめたシェルスクリプトの例です。

このシェルスクリプトをinstall-go18.shという名前で保存して実行してみましょう。

```
$ bash install-go18.sh
```

次のように表示されると成功です。

```
go version go1.8 darwin/amd64
```

だいたいのシェルはシェル自体がif、forといった制御文などを解釈する機能を持っているので、これらを駆使すると、ほかのプログラミング言語で行うようなプログラムを書くこともできます（後述）。しかし、シェルスクリプトの基本は**リスト1**のようなコマンドの羅列をまとめたものだ、と覚えておくことをお勧めします。

シェルの種類とシェバンについて

前項にて、「だいたいのシェルは」と説明しました。一口にシェルと言っても実はさまざま

な実装があり、よく利用されているものとしてはbashやzsh、fishなどが挙げられます。

多くのLinuxディストリビューションでは、bashがデファクトスタンダードとしてログインシェルに指定されています。ログインシェルとは、ログイン時に起動されるシェルのことで、環境変数$SHELLを参照すると確認できます。

```
$ echo $SHELL
/bin/bash
```

リスト1では、スクリプトの1行目に#!/bin/bashと記述しました。これはシェバン（shebang／シバン）と呼ばれるもので、実行形式で呼び出したときにどのインタプリタで実行されるべきかを示したものです。インタプリタ言語であるRubyやPythonでスクリプトを書いたことがある方には馴染みのあるものかもしれません。

シェバンを指定して実行することで、**図1**のように実行することも可能です。

どのシェルでシェルスクリプトを書くべきか

基本的にどのシェルを使ってシェルスクリプトを書いても問題ありません。

ただし、ほとんどのシェル環境で動くシェルスクリプトを求められる場合には、POSIXという規格[注1]に沿って書くことが望ましいです。

しかし、POSIX shに則ってシェルスクリプトを書くとなると、ログインシェルとして普段コマンドラインから利用するbashの機能や構文は使えませんし、一部POSIXでないコマンドやオプションが使えなくなってしまいます。要件やそのシェルスクリプトを書くことで何を解決できれば正とするのかなどと照らし合わせて、条件にあったシェルをシェバンに採用すると良いでしょう。

加えて、シェルの中でも、zshやfishなどはデフォルトでインストールされていないことが多いため、採用には注意が必要です。

ちなみに本稿で記載するシェルスクリプトのコードはbashで書いています。

シェルスクリプトをコマンド化する

Linuxコマンドの多くはC言語などで書かれていることも多いですが、シェルスクリプト（/bin/bash、/bin/zshなど）でも同様にコマンドを作ることができます。**図1**では作成したシェルスクリプトに実行権限を与えると、相対パスで実行することが可能になりました。これをどのディレクトリにいてもコマンド名（ファイル名）だけで実行できるようにするには、環境変数$PATHで指定されている（パスの通った）ディレクトリにコピーまたは移動します（**図2**）。

あるいは、次のように自分で用意したディレクトリにパスを通してもいいでしょう。

```
$ mkdir ~/bin
$ export PATH="$PATH:$HOME/bin"
```

注1）http://pubs.opengroup.org/onlinepubs/9699919799/

▼図1　シェバンを指定すれば、スクリプトの相対パスとファイル名だけでも実行可能

```
$ chmod 755 install-go18.sh   ←実行権限を付与する
$ ./install-go18.sh           ←シェバンに記したインタプリタで呼び出される
```

▼図2　パスの通ったディレクトリに入れることで、コマンド名だけで実行できる

```
$ mv install-go18.sh install-go18          ←コマンドらしい名前に変更
$ echo $PATH                               ←$PATHの内容を確認
/bin:/usr/bin:/usr/local/bin
$ sudo cp install-go18 /usr/local/bin      ←パスの通ったディレクトリにコピー
$ install-go18                             ←コマンド名だけで実行できる
go version go1.8 darwin/amd64
```

シェルスクリプトとパイプライン

シェルにおけるパイプラインとは、「あるコマンドの出力結果を、別のコマンドの入力として接続して、処理を継続する」というたいへん便利な機能です（**図3**）が、その本質はコマンドの並列処理にあります。パイプラインでは、つなげられたコマンドが順番にではなく同時に動作することで、共同で1つのまとまった仕事を処理します。

パイプ処理の多用は遅いと思われがちですが、この性質を鑑（かんが）みるとむしろ高速化が期待できるポイントになっています。なぜならパイプ先で生成されるプロセスはデータが流れたときに作られるのではなく、一斉に作られ常にデータ待ちをしていて、データが流れてきた瞬間から逐次処理をしているためです。

また、フィルタコマンド（標準入力からデータを読み込み、処理結果を標準出力に出力するコマンド）でなくとも、xargsコマンドを通して引数を渡してパイプをつなげ続けることもできます。少し冗長ですが、**図4**はトピックブランチに加えられた変更をmasterの状態に戻す（git revertではない）ようなコマンドです。

このパイプ処理をシェルスクリプトにおいて考えてみましょう。シェルスクリプトは基本的にコマンドラインの写しです。パイプをたくさんにつなげて処理を書くことはシェルスクリプトにおいても有用です。

拙作ですが、enhancdというCLI（コマンドラインインターフェース）アプリをシェルスクリプトで作っており、その中でパイプを多用している実際の例を見ることができます[注2]。

実践的なシェルスクリプトを書くための7つのポイント

ここまでは、おもに手順書としてのシェルスクリプトについて述べてきました。シェルスクリプトとはコマンドの羅列をまとめたテキストファイルに過ぎないので、バッチ処理や定期実行されるLinuxコマンドのセットを書き記したものであるべきです。

しかしここからは、手順書としてのシェルスクリプトというだけでなく、CLIアプリケーションとしてのシェルスクリプトという観点で、書く際に必要なことを説明します。

1. 成否は終了ステータスで返す

終了ステータス（status code/exit code）とは、コマンドの処理が終了したときの成否を表す値のことです。正常終了をゼロ値（0）、異常終了を非ゼロ値（1〜255）として扱います。直前に実行したコマンドの終了ステータスはシェル変数$?で確認できます。

```
$ cp a.sh b.sh
$ echo $?
0
```

たとえば、ここでa.shが存在しないなどの理

注2）https://github.com/b4b4r07/enhancd

▼図3　パイプを使うと、複数のコマンドをつなげて処理ができる

```
$ cat example.csv \                 ←あるCSVデータを
    | awk -F, '{print $1}' \        ←カンマ区切りの1列目を取り出し
    | sort \                        ←昇順ソートして
    | uniq -c \                     ←重複行をまとめてカウントし
    | sort -nr \                    ←カウントした順位で降順ソートし
    | head                          ←上位10個を表示する
```

▼図4　フィルタコマンドではないコマンドでも、xargsコマンドを使えばコマンドをつなげられる

```
$ git diff --name-only master... \  ←masterとの差分のあるファイルを一覧して
    | peco \                         ←インタラクティブに選んで
    | xargs git diff master... \     ←選んだデータをgit diffの引数として渡して
    | patch -p1 -R                   ←リバースパッチを当てて元に戻す
```

由で、cpコマンドが実行できなかった場合は異常終了として1が返ってきます。

```
$ cp a.sh b.sh
cp: a.sh: No such file or directory
$ echo $?
1
```

終了ステータスを用いてコマンド実行の成否をハンドリングすることは、UNIXコマンドでは一般的な手法です。シェルスクリプトでも、これらの終了ステータスを使って条件分岐するパターンが非常に多いです。**リスト2**の例で見てみましょう。

ちなみに、引数の数字を与えず単にexitとだけすると、exitの直前に実行したコマンドの$?を使ってexitされるので、意図しないステータスの場合があり、注意が必要です。

また、この終了ステータスの値（数字）には意味があります（**表1**）。自分で書いたシェルスクリプトやコマンドが0以外の終了ステータスを返す場合は、この規則に則って書くことが望ましいです。自分が示したいエラー症状に応じた終了ステータスを返しましょう。また、これらはシェルスクリプトだけではなく、CLIツールを書く際に使える知識なので覚えておくと良いでしょう。

ただし、例外もあって、wget、diffなどは**表1**にあるケースにかかわらず、そのコマンド内で定義した意味を持つ終了ステータスを返してきます。自分のシェルスクリプト内で終了ステー

▼リスト2　関数is_existの終了ステータスを使って条件分岐するスクリプト

```
#!/bin/bash

# コマンドにPATHが通っているかどうか調べる関数
function is_exist() {
    if [[ -z $1 ]]; then
        echo "too few arguments" 1>&2
        return 1
    fi
    # typeコマンド自体の出力はいらないので
    # 標準出力・標準エラー出力ともに捨てる
    type "$1" &>/dev/null
    # typeコマンドの終了ステータスを返す
    return $?
}

# 変数定義
cmd="go"

# ifでは真偽値で分岐される
# シェルスクリプトでは0がtrueで非0がfalse
# として解釈される
if is_exist "$cmd"; then
    # コマンドが存在する（ここでは何もしない）
    :
else
    # エラーメッセージは標準エラー出力に出す
    echo "$cmd: no such command" 1>&2
    # 異常終了なので0以外で終了する
    exit 1
fi
```

注3）http://tldp.org/LDP/abs/html/exitcodes.html

▼表1　終了ステータスの意味[注3]

値	意味	例	説明
1	一般的なエラー全般	`$ let "var 1 = 1 / 0"`	ゼロ除算などのコマンドを継続できないエラー
2	ビルトイン機能の誤用	`$ empty_func(){}`	キーワード（この例では「;」）の付け忘れや権限などの問題
126	呼び出したコマンドが実行できなかった	`$ /dev/null`	パーミッションの問題か、コマンドがexecutable（実行可能）でない
127	コマンドが見つからない	`$ illegal_command`	$PATHがおかしいときや入力ミスしたときに起こる
128	exitコマンドに不正な引数を渡した	`$ exit 3.14159`	exitコマンドは0〜255の整数だけを引数に取る
128＋n	シグナルnで致命的なエラー	`$ kill -9 $PPID`	例では、$?は137（128＋9）を返す
130	スクリプトがCtrl＋Cで終了	Ctrl＋C	Ctrl＋Cはシグナル2で終了する（128＋2＝130）
255	範囲外の終了ステータス	`$ exit -1`	exitコマンドは0〜255の整数だけを引数に取る

タスを用いて条件分岐するときは、これらのコマンドに注意してください。

2. 成功したときは何も表示しない

簡単なコマンドmkdirを例に説明します。これはディレクトリを作るときに使うコマンドですが、正常にディレクトリの作成が行われれば、ターミナル画面上には何も表示しません。

```
$ mkdir a
$
```

一見、これだと正常に終了したのかの判断がつきにくいですが、これはUNIXなりの親切さゆえの設計なのです。UNIXにおいて一般的なデータの受け渡しは標準入出力です。コマンドが実行されると標準入力を期待して待機し、標準入力から得られたデータを処理して標準出力に結果を出す、という挙動をすることが多いです。これらの振る舞いをするUNIXコマンドを「フィルタコマンド」と言います。フィルタコマンドは標準入力からデータが渡されるのを期待するため、データ以外の情報（コマンドの成功／エラーのメッセージなど）を標準出力に出すと、パイプを使ったデータの連携がやりにくくなります。これらの情報を表示する手段がほしい場合は、--helpオプションや標準エラー出力に表示するべきです。

パイプでつなげられることを前提としていないコマンドなどは、その限りではありません。とはいえ、前述のmkdirはフィルタコマンドではありませんが、出力は端的です。フィルタコマンドでなくとも成否のメッセージは端的であるのがUNIXコマンドらしさです。

もちろん、これはUNIXコマンドやCLIアプリとしてシェルスクリプトを書いている場合には当てはまることです。しかし、たとえばバッチ処理用のスクリプトを書いているときなどは、verbose（冗長に）に表示したほうがいい場合もあります。これは次の「3. 失敗したときは静かなエラー」においても同様です。

3. 失敗したときは静かなエラー

UNIXのエラー出力は静かです。静かという表現は、端的にその原因を告げることを意味します。それ以降の操作についてUNIXは言及しません。

```
$ rm dir
rm: dir: is a directory
```

上の例では、rmでディレクトリを削除しようとしました。もちろん、この場合rm -rとして再帰的な削除を命令する必要があります。しかし、UNIXはそのやり方を指示したりはしません。「rm -rとしなさい」とも「空のディレクトリならrmdirを使いなさい」とも言いません。ただ単に「それはディレクトリです」と標準エラー出力に表示するだけです。暗示的に「rmではディレクトリは削除できませんよ」と示しているだけなのです。

これはUNIXコマンドを書くときも同様です。エラーを表示するときに必要のない情報まで出力する必要はありません。必要のない情報とは、Usage（使用法）やオプションの使い方を示すようなヘルプメッセージなどです。この無駄な出力によって、パイプによるフィルタリングが難しくなったり、大事な出力が埋れたりするのは良いCLIとは言えません。

4. エラーは標準エラー出力に出力する

「2. 成功したときは何も表示しない」や「3. 失敗したときは静かなエラー」にも述べましたが、エラーは標準エラー出力に出力させましょう。次のようにして、エラーを標準エラー出力に出力することで、フィルタやコマンドとしての振る舞いがとてもやりやすくなります。

```
$ echo "An error occurred " 1>&2
```

また、純粋なコマンドのアウトプットだけを捨てたいとき、またはエラーメッセージだけを捨てたいとき、出力先を分けておくことで、これらの分離がやりやすくなります。

```
./output.sh
#!/bin/bash
echo "Expect stdout"
echo "Expect stderr" 1>&2
```

```
$ ./output.sh >/dev/null
Expect stderr ←標準エラー出力だけ表示
$ ./output.sh 2>/dev/null
Expect stdout ←標準出力だけ表示
```

これを一緒くたに同じ標準出力に出力してしまうと、>/dev/nullにリダイレクトしたとき、すべての出力が無に流れてしまいます。

5. フィルタとパイプを意識する

標準入出力はシェルスクリプトにおいて一般的なデータの受け渡し方法であると述べました。これに関して、パイプの開発者M.D.マキルロイも次のように要約しています。

> これがUNIXの哲学である。
> 一つのことを行い、またそれをうまくやるプログラムを書け。
> 協調して動くプログラムを書け。
> 標準入出力（テキスト・ストリーム）を扱うプログラムを書け。標準入出力は普遍的インターフェースなのだ。
> ――ダグラス・マキルロイ、UNIXの四半世紀

古来から存在するUNIXコマンドの多くは、この思想をもとにフィルタコマンドとして設計されています。私たちが作成するCLIアプリもこの慣習にならい、データ入力には標準入力（stdin）を使用し、データ出力には標準出力（stdout）を使用するべきです。こうすることで、パイプによるフィルタリング処理がしやすくなります。パイプでフィルタしたデータを受け渡しできるようになれば、ほかのツールとの連携がとても簡単になります。

自作するシェルスクリプトでも標準入力を受け付けられるようにする場合、デバイスファイルである/dev/stdinと、test -pなどを使うことで標準入力の有無を判断できます[注4]。

```
$ test -p /dev/stdin
$ echo $?
1  ←標準入力がないため異常終了になる
$ echo "hogehoge" | test -p /dev/stdin
$ echo $?
0  ←標準入力があるため正常終了になる
```

標準入力が確認できる場合、test -pはtrue（0）を返していることが確認できます。ちなみに、testコマンドは[コマンド（対応する位置に]が必要）と同等の機能を持ち、bashなどのシェルではさらに多くの機能を持つ[[（対応する位置に]]が必要）がビルトインコマンド（シェル自体が持っている機能やコマンドのこと）として実装されています。これを使って標準入力を判断するスクリプトを書いてみましょう。**リスト3**は、標準入力からパッケージ名を得て、そ

注4）http://qiita.com/b4b4r07/items/77c589f21a99db8bb682

▼リスト3　フィルタコマンドとしてのシェルスクリプト（goget.sh）

```
#!/bin/bash

# 標準入力がない場合は何もしない
if [[ ! -p /dev/stdin ]]; then
    echo "no stdin data" 1>&2
    exit 1
fi

count=0

# readコマンドは標準入力からデータを変数に格納できる
# whileと組み合わせることでEOF(^D)があるまで読み続ける
while read line
do
    echo "Installing $line"
    # バックグラウンドで処理（サブプロセス化）
    go get -u "$line" &
    # 同時に立ち上げるプロセスの数を16個に制限する
    (( (count += 1) % 16 == 0 )) && wait
done
# すべてのプロセスの終了を待つ
wait
```

れぞれのパッケージを導入するスクリプトです。

実行すると図5のようになります。このようにパイプを捕捉し、フィルタコマンドとしての役割を担うことができるようになります。

6. bashに依存しているのに`#!/bin/sh`と書かない

/bin/shの実態は環境によってまちまちです。さまざまなシェルのシンボリックリンクになっていることが多いです。そのため、予期せぬエラーや動作しないといったことが起きる場合があります。

bashやその他シェルに依存したスクリプトを書くのなら、そのシェル上での動作を前提にしているので、シェバンには#!/bin/bashなどと書いたほうが良いです。

Linuxディストリビューションのデファクトスタンダートと化しているUbuntuでは、/bin/shは/bin/dashになっています。つまり、Ubuntuで動かすスクリプトにbashの文法を書いておきながらシェバンを#!/bin/shとしていると、dash上で実行されてエラーで動かないという事態も起きかねません。

```
./list.sh
#!/bin/sh
list=( $(ls) )
echo "$list"
```

Ubuntu環境だと/bin/shの実態はdashになるので、次のようにエラーとなります。

```
$ ./list.sh
./list.sh: 2: ./list.sh: Syntax error: ➥
"(" unexpected
```

これはdashが(...)の表記をサポートしていないためです。

ほかにも、if [...の書き方(testコマンドの代用である[)などもサポートしていません。これは大きくスクリプトの動作を変えてしまいます。

7. 移植性、ポータビリティ

シェルスクリプトの魅力の1つはポータビリティです。POSIX shで記述するスクリプトは、理論上最も汎用性の高いシェルスクリプトとして、ほとんど改変や手直しを加えることなく多くの環境で使いまわせます。ただし、これにはPOSIXという規格で定められた範囲内の文法やコマンドで記述することが必要となります。

そのスクリプトがどんなケースで実行されることを想定しているかによって、使用するシェルを選択すると良いでしょう。

まとめ

シェルスクリプトとは何なのか、またそれを書くうえで想定される2つのユースケースについてまとめました。

・手順書としてのシェルスクリプト
・CLIアプリとしてのシェルスクリプト

今回紹介したことは、シェルスクリプトにおいては基本的なことです。とくに後者に関しては、ほかのスクリプト言語やGo言語で書く際にも通ずる話です。

ぜひこの機会に、シェルスクリプトを通してその使い方と、CLIアプリ(UNIXコマンド)のお作法について触れてみてください。SD

▼図5　goget.shの実行結果

```
$ cat pkg.txt
github.com/b4b4r07/gist
github.com/heppu/gkill
github.com/peco/peco
github.com/davecheney/httpstat
(..省略..)

$ cat pkg.txt | ./goget.sh
Installing github.com/b4b4r07/gist
(..省略..)
```

第3章 理論編 …3

しくみを知れば、bashは怖くない

第1、2章で基本を学んでも、プログラミング初心者にはまだbashはどこか異質な存在という印象があるのではないでしょうか？　そのイメージを一新するため、本章ではbashをプログラミング言語の一種と捉えつつ、その本質としくみを解説します。そして、より発展的に使うための基礎を固めていきます[注1]。

Author 田島 優也（たじま ゆうや）
Twitter @tajima_taso
株式会社オールアバウト

bashはプログラミング言語処理系

そもそもbashとは何なのでしょうか？　この問いは広い意味では「シェルとは何なのでしょうか？」という問いにつながります。

プログラミング言語を学び始めると、ひとまず細かい説明は抜きでテキストファイルにHello, World!と命令文を書いて実行し、画面に表示させるということから始めるケースが多いと思います。そこから、徐々に発展的なプログラミングを行っていくのですが、特定のプログラミング言語についての理解が深まっていく一方で、ベースのインターフェースとなっているシェルについては、ほとんど意識せずに進んでしまっている人もいるのではないでしょうか？

そこで、この章では最初にHello, World!を書くときの気持ちに立ち戻り、「シェルとは何か？」というところから始めてみましょう。シェルについてインターネットで検索して調べてみると、いろいろな説明が出てきますが、それらをまとめて次のように定義します。

シェルとは、ユーザとOSの中間に位置し、ユーザのリクエストした命令を解釈してそれに対する処理を実行するソフトウェアである。

その定義だけに注目して、まずはシェルとしてbash、プログラミング言語としてPHPを取り上げて両者の共通点を見ていきましょう。

なおバージョンに関してPHPは7.0.18、その他ソフトウェアやライブラリはCentOS 6.9のデフォルトリポジトリに準じます。

PHPとbashを比較する

PHPでプログラムを実行する

PHPでHello, World!を画面に表示するプログラムを書いてみます。

```
<?php
echo "Hello, World!\n";
```

これをexample.phpとして保存し、シェルからphpコマンドの引数として実行します（**図1**）。

▼図1　シェルからexample.phpを実行

```
$ ls -l example.php  ←メタ情報確認
-rw-rw-r--  1 tajima tajima     30  5月  1 13:37 2017 example.php
$ php example.php  ←実行
Hello, World!
```

注1）なお本章でシェルという言葉を用いる場合、それは基本的にCLI（Command Line Interface）のシェルのことであり、ソフトウェアとしてはbashを指すこととします。

bashでプログラムを実行する

続いてbashで同じプログラムを書いてみます。

```
echo "Hello, World!"
```

これをexample.shとして保存し、シェルからbashコマンドの引数として実行します（**図2**）。

気づいた人もいるかもしれませんが、example.shの先頭行にはシェバン（#!/bin/bashなど）を書いていませんし、ファイルのパーミッションに対して実行権も付与していません。必要なのは読み取りの権限だけです。なぜならこの場合、example.shはbashコマンドの引数と指定されているので、インタプリタがbashであることは明確であり、実行権はbashコマンドに付いてさえいれば良いからです。

おそらく、PHPの例では「example.phpの先頭行にシェバンが必要なのでは？」あるいは、「example.phpに実行権を付与しなくてはいけないのでは？」と思った人はいないと思いますが、bashの場合はそのあたりを気にしてしまった方がいるのではないでしょうか？

実は、シェバンはシェル（bash）のスクリプト固有のものではなく、あくまでexecveシステムコールに対してインタプリタとして何を利用するかを指定するものなのです。

さて、bashとPHPにおいて、テキストファイルの文字列から命令を実行するしくみは同じであることがわかりました。次はログインシェルとして利用するケースから両者を比較しましょう。

bashをログインシェルとして使う

bashをとあるユーザのログインシェルに設定してみます。ログインシェルとは、ユーザがシステムにログインしたときに設定されるシェルのことです。ログインしたユーザはログインシェルをインターフェースとして、システムの機能を利用します。

ログインシェルとして動作するシェルは通常、インタラクティブシェルとして実行されます[注2]。

ほとんどのLinuxのディストリビューションのデフォルトで、bashがログインシェルとして設定されていると思いますが、あえて再度bashに設定してみます（**図3**）。

これで、tajimaのログインシェルがbashに変更されました。ログアウトして、再度ログインしてみてください。なお、ユーザがシステムにログインする場合はsshdを経由してログインすることを前提とします。psコマンドで状況を確認してみます（**図4**）。

▼図2　シェルからexample.shを実行

```
$ ls -l example.sh  ←メタ情報確認
-rw-rw-r-- 1 tajima tajima 21  5月  1 13:54 2017 example.sh
$ bash example.sh  ←実行
Hello, World!
```

▼図3　ログインシェルをbashに設定する

```
$ cat /etc/shells  | grep '/bin/bash'  ←/bin/bashが/etc/shellsに記述されているか確認する
/bin/bash  ←記述されている
$ sudo chsh -s /bin/bash tajima  ←chshコマンドでtajimaのログインシェルを/bin/bashに変更
tajima のシェルを変更します。
シェルを変更しました
```

▼図4　psコマンドの実行結果（ログインシェルをbashにした場合）

```
tajima    28035  0.0  0.0 102560  1868 ?        S    14:34   0:00      \_ sshd: tajima@pts/1
tajima    28036  0.0  0.0 108364  1812 pts/1    Ss+  14:34   0:00          \_ -bash
```

注2）命令の入力と実行をユーザが繰り返し行える状態であることをインタラクティブであると言いますが、こういった性質を持つシェルのことをインタラクティブシェルと言います。bashにおいて厳密にログインシェルの定義の話をすると、必ずしもインタラクティブである必要はないのですが、現実的な利用ケースを考慮して、ここではインタラクティブなものとして話を進めます。

ユーザtajimaがログイン後に実行しているプロセスがbashであることが確認できました。ログインしたユーザプロセスの環境変数には、SHELLが設定されているはずなので確認してみます。

```
$ sudo strings /proc/28036/environ | grep SHELL
SHELL=/bin/bash
```

では、bashで次の操作を行ってみましょう。

①カレントディレクトリの確認
②ディレクトリの作成
③ディレクトリへの移動
④ファイルの作成
⑤ディレクトリの中身を確認する

bashでカレントディレクトリの確認は、bashのビルトインコマンドpwdで行えます。

```
$ pwd
/home/tajima
```

ディレクトリの作成は、外部コマンドmkdirで行えます。

```
$ mkdir foo
```

ディレクトリへの移動は、bashのビルトインコマンドcdで行えます。

```
$ cd foo
$ pwd
/home/tajima/foo
```

ファイルの作成は、外部コマンドtouchで行えます。

```
$ touch hoge
```

ディレクトリの中身を確認するには、外部コマンドlsで行えます。

```
$ ls
hoge
```

いずれもLinuxの新人研修で行うような基本操作ですので、とくに説明不要かと思います。ただ、外部コマンド、ビルトインコマンドの違いについては、のちほど補足します。

PHPをログインシェルとして使う

PHPもbashもソースコードから実行できることは先ほど見たとおりですが、次はPHPをログインシェルとして使ってみます。もともとPHPは-aオプションを付けて起動すると、インタラクティブに起動することができます[注3]。php >がプロンプトです。bashにおける$に相当します。

```
$ php -a
Interactive shell

php > echo "Hello, World!\n";
Hello, World!
php >
```

この機能を利用して、PHPをログインシェルとして起動します。そのための準備として、C言語で**リスト1**のソースコードを作成します。

細かい説明は省きますが、要するにPHPのプロセスを-aオプション付きで起動する実行

注3）http://php.net/manual/ja/features.commandline.interactive.php

▼リスト1　PHPをインタラクティブに起動するためのソースコード

```
#include <unistd.h>

extern char **environ;

int main (void) {

  char *const argv[] = {"/usr/bin/php", "-a", NULL};

  execve("/usr/bin/php", argv, environ);

  return 1;
}
```

ファイルを作成するコードです。これをコンパイルすると、引数なしでPHPをインタラクティブに起動できる実行ファイルが作成されるので、それをログインシェルとして設定します。

```
↓実行ファイル名をphp2とする
$ gcc php.c -o php2
↓実行ファイルを/usr/binに移動
$ sudo mv php2 /usr/bin/php2
$ ls /usr/bin/php2
/usr/bin/php2
```

/etc/shellsに実行ファイルのパスを追記します。

```
$ sudo vim /etc/shells
↓vimが起動したら行末に以下を追記し保存
/usr/bin/php2
```

ここまで準備できたら、あとはbashのときと同じようにchshコマンドに/usr/bin/php2を指定してtajimaのログインシェルを変更して完了です。sshdを経由してtajimaユーザでログインしてみましょう。

ログインできたら、別のユーザの端末でpsコマンドを実行して確認します（**図5**）。tajimaユーザの実行しているプロセスがphp -aであることがわかります。

ログインしたユーザプロセスの環境変数には、SHELLが設定されているはずなので確認してみます。

```
$ sudo strings /proc/28232/environ | grep SHELL
SHELL=/usr/bin/php2
```

また、tajimaユーザの画面には、システムへのログイン時に次のようなメッセージが表示され、PHPがログインシェルとして動作していることがわかります。

```
Last login: Mon May  1 14:34:31 2017
from xxx.xxx.xxx.xxx
Welcome to xxxx

Interactive shell

php >
```

それでは、bashのときに行った操作をPHPで行ってみましょう。当然、シェルとしてbashを使用して解釈されていた文字列は、PHPのシェルでは解釈できません。

```
php > pwd;
PHP Notice:  Use of undefined constant
pwd - assumed 'pwd' in php shell code
on line 1
php >
```

このように、あくまでPHPで定められている文法に沿わないと、目的のコマンドを実行できません。bashでは当たり前のように解釈・実行されていた、パイプ（|）やリダイレクト（>）のようなユーティリティコマンドもインタプリタに認識されません。

これはPHP以外のプログラミング言語で書かれたソースコード（たとえばC言語）が、PHPで実行できないことと同じです。

以上をふまえて、PHPでbashと同じこと（前述の①～⑤）を実現してみます。

PHPでカレントディレクトリの確認は、PHPのビルトイン関数getcwdで行えます。また、PHPの関数では、bashのようにコマンドの実行結果を標準出力に吐き出す機能が基本的には組み込まれていないので、結果の出力を確認したい場合はechoを指定します。命令の終端には;と改行を指定します。

▼図5　psコマンドの実行結果（ログインシェルをPHPにした場合）

```
tajima     28231  0.0  0.0 102560  1864 ?        S    16:02   0:00      \_ sshd: tajima@pts/1
tajima     28232  0.0  0.5 314496 10868 pts/1    Ss+  16:02   0:00          \_ /usr/bin/php -a
```

```
php > echo getcwd();
/home/tajima
```

ディレクトリの作成は、PHPのビルトイン関数mkdirで行えます。

```
php > mkdir("./foo");
```

ディレクトリへの移動は、PHPのビルトイン関数chdirで行えます。

```
php > chdir("foo");
php > echo getcwd();
/home/tajima/foo
```

ファイルの作成は、PHPのビルトイン関数touchで行えます。

```
php > touch("hoge");
```

ディレクトリの中身を確認するには、PHPのビルトイン関数scandirで行えます。結果をきれいにフォーマットして出力させるには、少しプログラムを書く必要があるので、今回は結果をvar_dumpで表示するにとどめます。

```
php > var_dump(scandir("."));
array(3) {
  [0]=>
  string(1) "."
  [1]=>
  string(2) ".."
  [2]=>
  string(4) "hoge"
}
```

いかがでしょうか？　bashとはだいぶインターフェースが異なりますが、PHPのシェルもユーザのリクエストに対する命令を実行できています。この挙動はシェルそのものです。

bashのほうがシェルとしての機能と利便性に特化しているというだけで、数あるプログラミング言語処理系の一部というイメージを少しでも持てたとすれば、だいぶbashとの距離感が近づいてきたと言えるでしょう。

bashはリアルタイム性が要求され、実行している環境のファイルシステムの構成などに影響を与えてしまう命令を処理する機会が多いのに対して、通常のプログラミング言語は好きなテキストエディタやIDEの環境でじっくり編集でき、bashほどシステムに影響を与える危険性が少ないかもしれません。

ですが、bashもみなさんが日常的に触れているプログラミング言語と本質的に同じものです。初めて触れるプログラミング言語がbashで、bashで初めてのHello, World!を表示するプログラムを書くケースがあってもいいのです。

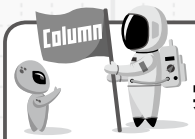
bashで知っておくべきこと

この節ではbashの機能や決まりごとについて見ていきます。bashをプログラミング言語として捉えたうえで、これらを理解しておくとbashを応用的に使えるようになるでしょう。

bashの設定ファイル

bashをプログラミング言語と捉えると、bashの実行の流れは次のように考えられます。

設定ファイルの読み込み → 各種変数や関数の定義 → ユーザプログラムの実行



PHPの外部コマンド実行関数

PHPにはsystemやexecといった外部コマンドを実行するためのビルトイン関数がありますが、今回は使用していません。これらの関数は内部的に/bin/sh -c 引数の形式でコマンドを実行しており、bashを利用してしまうからです。実際にプログラミングを行う過程では、目的に応じてこれらの関数を利用すると便利な場合がありますので、適時利用しましょう。ビルトインと外部コマンドの違いについては、のちほど見ていきます。

設定ファイルをどう読み込むかについては、bashの起動コンテキストによって違いがありますので、その点について説明します。

これまでインタラクティブシェルとログインシェルという言葉をとくに区別せずに使ってきました。ここでその違いを明確にしておきます。

ログインシェル
　グローバルな設定情報を読み込む
インタラクティブシェル
　ローカルな設定情報を読み込む

上記のような違いがあります。ただし、これらは厳密には独立したものではなく、ORで結ばれる関係にあります。

グローバルな設定情報としてはbashはデフォルトで、/etc/profile、~/.bash_profileを対象にして読み込みます。なお、~/.bash_profileが存在しない場合は、~/.bash_login、~/.profileを順に探していき、読み込めた場合はそこで設定ファイルの読み込みをストップします。

ローカルな設定情報としては、~/.bashrcを対象として読み込みます。

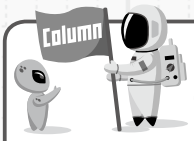
psコマンドでシステムログイン後のtajimaユーザのプロセス状況を確認すると、**図6**のように表示されています。bashでは0番目の引数の最初の文字が-である場合、ログインシェルです[注4]。したがって、設定ファイルとして/etc/profile、~/.bash_profileを読み込みます。

この状態で、bashを起動してみます。

```
$ bash
```

するとログインシェルのbashから非ログインシェルのbashがインタラクティブシェルとして新たに起動されます（**図7**）。したがって、この場合は~/.bashrcが新たに読み込まれます。

もしも~/.bash_profile内でbashのビルトインコマンドexportを使って変数を定義していた場合、新たに起動したbash内でもその変数にアクセスすることが可能です。新たに起動し



ビルトインコマンドexport

シェルから新たに起動したシェルは、親とアドレス空間が異なる別プロセスなので、特別な手法を用いないと親プロセス内の情報を引き継げません。

そこでbashでは親プロセスで定義した変数を子プロセスでも参照するために、環境変数を子プロセスにコピーするという手法で変数の参照を可能にしています。その処理を行ってくれるのが、bashのビルトインコマンドexportです。exportコマンドは子プロセスに引き継ぎたいシェル変数にフラグを付けておき、execveシステムコール呼び出し時の引数にその情報を渡すことにより、生成された子プロセスに対して環境変数を引き継いでいます。

プログラミング言語におけるオブジェクト指向の文脈で語るなら、ログインシェルは親クラス、そこから起動されたシェルは子クラスのようなイメージを持っておくと、各設定ファイルに適切な内容を設定できると思います。

注4）bash --loginやbash -l、またはsu -コマンドやsshd経由で実行された場合もログインシェルと定義されます。

▼図6　ログイン直後は、ログインシェルとしてbashが起動している

```
tajima   10648  0.0  0.1 104264  3652 ?        S    13:01   0:00  |   \_ sshd: tajima@pts/4
tajima   10649  0.0  0.0 108500  1880 pts/4    Ss   13:01   0:00  |       \_ -bash
```

▼図7　さらにbashを起動すると、ログインシェルのbashから非ログインシェルのbashが起動される

```
tajima   10648  0.0  0.1 104264  3652 ?        S    13:01   0:00  |   \_ sshd: tajima@pts/4
tajima   10649  0.0  0.0 108500  1880 pts/4    Ss   13:01   0:00  |       \_ -bash
tajima   11982  0.0  0.0 108500  1864 pts/4    S+   14:38   0:00  |           \_ bash
```

たbashの先頭には-がないので、ログインシェルではありません。

ビルトインコマンドと外部コマンド

ここまでbashの話をしてきた中で、ビルトインコマンドと外部コマンドという話が出てきました。その違いを理解していないと、思わぬ不具合に見舞われることがあるのでその違いをはっきりさせておきます。

ビルトインコマンドも外部コマンドも、コマンドという名前からわかるように実行可能な命令で、ビルトインコマンドはbashの実行ファイルに組み込まれているコマンドを指し、外部コマンドはbashの実行ファイルに組み込まれていないコマンドのことを指します。

違いに関して、より具体的なイメージをつかむために、psコマンドを使って確認してみましょう。

/usr/bin/timeは外部コマンドです。このコマンドは引数として指定したコマンドの実行時間を計測する処理を行います。そこで、/usr/bin/time/の引数としてsleepを指定したあと、ほかの端末からpsコマンドでプロセスの状態を確認してみます。

```
$ /usr/bin/time sleep 10
```

bashから新たに/usr/bin/timeが子プロセスとして起動され、そこからさらにsleepが子プロセスとして起動しています（**図8**）。

次にbashのビルトインコマンドtimeを使って、同様にsleepの実行時間を計測して、ほかの端末からpsコマンドで状態を確認します。

```
$ time sleep 10
```

timeのプロセスが存在せず、bashから直接sleepが起動されて、実行時間を計測しています（**図9**）。これは、timeの処理がbashのプロセスの中で動作していることを意味します。

このように、ビルトインコマンドは、新たにプロセスを生成せずにbashのプロセスの中で実行され、外部コマンドは、新たに生成されたプロセスの中で実行されていることがわかります。ビルトインコマンドのメリットとしては、bashに組み込まれているぶん高速に実行できることと、内部で完結するため外的な要因に依存しない実行結果が保証されることです。

紛らわしいのは（先ほどのtimeコマンドもそうですが）、ビルトインコマンドと外部コマンドで同名のコマンドが存在しているパターンがあることです。指定されたパスが特定するものが、bashのビルトインコマンドなのか外部コマンドなのか判断するには、bashのビルトインコマンドtypeを利用します（**図10**）。

bashのビルトインコマンドの場合は、shell

▼図10　typeコマンドでビルトインコマンド／外部コマンドの判断ができる

```
$ type time
time is a shell keyword
$ type /usr/bin/time
/usr/bin/time is /usr/bin/time
```

▼図8　外部コマンド/usr/bin/timeを実行した場合（bashからtimeが起動、timeからsleepが起動されている）

```
tajima  13081  0.0  0.0 108500  1968 pts/0   Ss   17:50   0:00  |   \_ -bash
tajima  13264  0.0  0.0   3936   300 pts/0   S+   18:09   0:00  |      \_ /usr/bin/time sleep 10
tajima  13265  0.0  0.0 100924   576 pts/0   S+   18:09   0:00  |         \_ sleep 10
```

▼図9　ビルトインコマンドtimeを実行した場合（bashから直接sleepが起動されている）

```
tajima   13081  0.0  0.0 108500  1968 pts/0    Ss   17:50   0:00  |      \_ -bash
tajima   13315  0.0  0.0 100924   572 pts/0    S+   18:14   0:00  |         \_ sleep 10
```

builtinやshell keywordというメッセージが表示され、外部コマンドの場合はそのコマンドまでの絶対パスやハッシュ化して保存されているという旨の表示がされます。

これまでに学んだことを応用してみる

それでは、最後にこれまでに学んだことから、実践的に応用するケースを見ていきましょう。

bash以外のインタプリタでシェルスクリプトを実行したい

PHPとbashの比較の項で、シェルスクリプトはシェル（bash）固有のものではないと説明しました。よって、PHPでシェルスクリプトを実行することも可能です。シェバンにはPHPのインタプリタまでのフルパスを指定します。

```
#!/usr/bin/php
<?php

echo "Hello, World!\n";
```

このファイルをexample_php.shとして保存し、実行権を付与することでそのまま実行可能です（**図11**）。

bashで定義した変数に、bash以外の環境からアクセスしたい

bashの設定ファイルの項で、ビルトインコマンドexportを使って変数を定義すると、新たに起動した子プロセスのbashでもその変数にアクセスできると説明しました。そして、コラム「ビルトインコマンドexport」の中で説明したとおり、変数は環境変数として引き継がれているため、インターフェースさえあれば、bash以外のプロセスからも定義済み変数にアクセスできます。

```
↓bashのexportで変数を定義
$ export hoge=foo
↓PHPをインタラクティブモードで起動
$ php -a
Interactive shell

↓環境変数の値を取得するPHPのビルトイン関数
php > echo getenv('hoge');
foo
```

sudo cdを行いたい

UNIXライクなほとんどの環境において、現在のユーザとは別のユーザの権限でコマンドを実行できるsudoというユーティリティコマンドが使えます。sudoコマンドを利用することによって、一時的に権限レベルの高いユーザの権限でコマンドを実行して、システムに関わる設定変更を行えます。

sudoコマンドは引数として与えたコマンドを、新たなユーザ権限のもと別プロセスで起動するコマンドですので、たとえばsudo sleep 10を実行時のプロセスの状態は**図12**のようになります。

▼図11　PHPで書いたシェルスクリプトを実行する

```
$ chmod u+x example_php.sh  ←実行権を付与
$ ls -l example_php.sh  ←メタ情報確認
-rwxrw-r-- 1 tajima tajima 45  5月  7 19:28 2017 example_php.sh
$ ./example_php.sh  ←実行
Hello, World!
```

▼図12　sudo sleep 10を実行した場合のプロセスの状態

```
tajima  30646  0.0  0.0 108504  1864 pts/1    Ss   12:47   0:00          \_ -bash
root    30715  0.6  0.2 196460  4256 pts/1    S+   12:56   0:00             \_ sudo sleep 10
root    30716  0.0  0.0 100924   576 pts/1    S+   12:56   0:00                \_ sleep 10
```

では、cdコマンドをsudoコマンドから実行してみましょう。

```
$ cd /var/spool/cron
-bash: pushd: /var/spool/cron: 許可がありません
$ sudo cd /var/spool/cron
sudo: cd: コマンドが見つかりません
```

cdコマンドが見つからないというエラーが出てしまいました。外部コマンドとビルトインコマンドの説明を思い出してみてください。cdコマンドを新たなプロセスとして起動しようとしましたが、cdはbashの中に組み込まれているビルトインコマンドなので直接実行できません。また、/bin/cdというような外部コマンドも存在しません。

したがいまして、sudoからcdを実行したい場合はbashそのものを新たに起動しなくてはなりません。

```
↓sudoの-Eオプションで環境変数を引き継ぐ
$ sudo -E bash
$ cd /var/spool/cron
$ pwd
/var/spool/cron
```

このときのプロセスの状態は**図13**のようになり、ログインシェルとして起動していないので読み込まれるのはrootの~/.bashrcのみとなります。この場合グローバルな設定はあくまでtajimaユーザに定義されている想定で、イメージとしてはbashの設定ファイルの項で説明したとおり、親クラスと子クラスの関係です。

あるいは、bashを現在のユーザと異なるユーザの権限で起動しなおすほど大げさなことをしなくても、一度限りのコマンドを実行したい場合は**図14**のような方法もあります。ただし、このとき実行されるコマンドは子プロセスとして実行されますので、親子で共有していない情報の変更についてはカレントプロセスに戻ったときに失われてしまうので注意です。

まとめ

いかがでしたでしょうか？　シェルは黒い画面と呼ばれ、非エンジニアはもちろん初心者のエンジニアにも苦手意識がある方がいるようです。それはシェルに関して知らないことや、理解していないことが多いということだと思いますが、冒頭で述べたようにプログラミング言語を学ぶ過程でそうならざるを得ないという側面があると思います。

そして、いざシェルに関して学びたいと思っても、関連するシェルの情報はオペレーションやTIPSに関するものが多く、しくみの部分に関して説明したものはbashの歴史のわりには少ないように思えます。ですので、本章によって、シェルがみなさんにとって少しでも身近なものになってくれていたとしたら幸いです。

なお、bashのことをもっと深く詳しく知りたいという方は、筆者がQiitaで書いたエント

▼図13　sudoでbashを新たに起動した場合のプロセスの状態

```
tajima  30646  0.0  0.0 108504  1912 pts/1   Ss   12:47   0:00      \_ -bash  ←親クラス
root    31490  0.0  0.1 189396  3012 pts/1   S    14:50   0:00         \_ sudo -E bash
root    31491  0.0  0.0 108360  1836 pts/1   S    14:50   0:00            \_ bash  ←子クラス
```

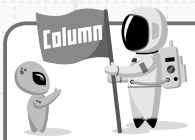
▼図14　一度限りのコマンドであれば、bashの-cオプションを使えば実行できる

```
$ pwd
/home/tajima
$ sudo bash -c "cd /var/spool/cron; pwd"
/var/spool/cron   ←cdコマンドによって子プロセスのカレントディレクトリが変更されている
$ pwd
/home/tajima   ←親プロセスではカレントディレクトリが変更されていない
```

リ[注5]を参照してみてください。

また、筆者のQiitaでは、bashに限らずほかのさまざまなソフトウェアについても調べた結果をアウトプットしていく予定ですので、よろしければフォローしていただけますと励みになります。SD

注5）http://qiita.com/tajima_taso/items/149ca77a2401bf9bf026



同名の外部コマンドとビルトインコマンドが存在する場合

現在のプロセスのカレントディレクトリを表示するコマンドとして、pwdコマンドがありますが、このコマンドはbashのビルトインコマンドとして実装されていることに加え、外部コマンドとしても存在している環境があります。この場合、pwdと打った場合はbashのビルトインコマンドが優先され、環境変数PATH上にpwdコマンドが存在したとしてもそちらは実行されません。

ただし、sudo pwdのようにビルトインコマンドを実行できない状況では、外部コマンドをPATH上のディレクトリから検索して/bin/pwdを見つけるので、こちらを実行します。

```
$ sudo pwd ←/bin/pwdが実行されている
/home/tajima
```

外部コマンドと内部コマンドで実行結果が同じように見えても、指定できる引数や内部処理が異なる場合があるので注意しましょう。

第4章　応用編 … 1

じつはこんな機能があった！ bashの新機能、便利機能

本稿ではバージョン4.0以降に加わった機能をいくつか紹介します。シェルスクリプトの互換性などを考えると、新しい機能を取り入れるのは慎重にならざるを得ませんが、使わないからといって知らなくて良いわけでもありません。どんなことを解決するために追加された機能なのかを考えながら見ていきましょう。

Author 上田 隆一（うえだ りゅういち）
千葉工業大学

2016年9月にbashのバージョンが4.4になり、いくつかの新機能が追加されました。しかし、bashのバージョンアップは毎度盛り上がりに欠けます。なぜかと理由を考えてみると、たとえば仕事でシェルスクリプトを書いていてPOSIXに合わせなければならない場合、bash[注1]の新機能は危険な落とし穴でしかありません。逆にシェル芸的にコマンドを組み合わせるときも、使う機能はせいぜいパイプとリダイレクトくらいです。

「bashに新機能ついたよ！」「お、おぅ……」という微妙なムードには、だいたいこの2つの背景があるような気がします。

たぶん、新機能の追加にはそれなりの理由が

とは言っても、新機能が追加されるということは、それが必要だと思った人と、追加を許可した人がいたということです。また、bashの裏側では、普段ユーザが目にしない設定のためのデータやフラグ類がbashの文法やしくみで管理されており、そのために設けられたと思われる機能もいくつか見受けられます。余計な機能を普段は使わないにしても、それらを知ることで、シェルのしくみを垣間見ることができます。

そこで本稿は、bashのバージョン4.0以後に加わった新機能[注2]を中心に、「なんでこんな機能があるの？」「なにが便利なの？」という視点で探って、読者のみなさんとキャッチアップしていくという趣旨でお送りすることにします。先ほど述べたように筆者はあまりbashの細かい機能に頓着（とんちゃく）していないのですが、それでもbashが不便だなと思うことは多々ありますので、新機能がそれをどう解決するのかということを説明していこうかと思います。

bash 4.4の準備

bashのコードは、

https://ftp.gnu.org/gnu/bash/

にありますので、バージョン4.4をダウンロードして使いましょう。多くのUNIX系OS環境では、**図1**の手順でインストールできます。他の古いバージョンも同様にセットアップできますので、自身で新旧バージョンの比較を行う場合は**図1**のtar.gzファイルを変更していくつかbashをインストールしてみましょう。筆者が本稿で検証に使った環境は、Ubuntu 16.04

注1）shで書けという話ですが、shのふりをしてbashが動いている環境もあるので事態はそんなに単純ではありません。

注2）と言ってもバージョン4.0もすでに8年経っているのですが、盛り上がってないので新機能とさせていただきます。

Serverです。

歴代のバージョンで何が変わったかは、tarで展開したディレクトリ直下のCHANGESというファイルに記録があります。興味があればご一読を。

また、バージョンを切り替えながら作業するときは、頻繁に変数BASH_VERSIONを確認しましょう。たとえばバージョン4.4と3.2を**図1**の手順でパスの通ったところに置くと、次のようにバージョンを切り替えられます。

```
$ bash4.4
$ echo $BASH_VERSION
4.4.0(1)-release
 バージョン3.2も図1の手順で作って置いた場合
$ bash3.2
 後述の連想配列の関係でワーニングが出ますが気にしないことにします
$ echo $BASH_VERSION
3.2.57(1)-release
```

標準出力と標準エラー出力をまとめてパイプに出せる「|&」

まずは軽い話題から。バージョン4.0から|&というパイプの表記ができるようになっています。bashを使っていると、たまに標準出力と標準エラー出力をまとめて別のコマンドに渡したいことがありますが、バージョン3.2までは、この場合、

```
$ ls a aa 2>&1 | nl
     1 ls: 'aa' にアクセスできません: 以下略
     2 a
```

というように、2>&1というファイル記述子の操作を明示的に書く必要がありました。

このワンライナーの意味を説明しておくと、lsの標準出力と標準エラー出力を2>&1でまとめてパイプに渡し、nlというコマンドで行番号をつけるというものです。最初にlsのエラー出力、次にlsの標準出力がパイプを通り、nlで番号がつけられてこのような出力になっています。ファイル記述子については、本稿のニッチな性質上、説明を割愛します。

一方、バージョン4.0以降では、そんな小難しいことを考える必要もなく、

```
$ ls a aa |& nl
     1 ls: 'aa' にアクセスできません: 以下略
     2 a
```

で済んでしまいます。ただ、数文字減っただけなので地味な機能追加ではあります。

これがどれだけ便利か、というところですが、次のようなケースを1つの例として挙げておきます。たとえば一般のユーザでディレクトリにfindをかけると、

```
$ find /proc/
/proc/
/proc/fb
(..略..)
find: `/proc/tty/driver': 許可がありません
find: `/proc/1/task/1/fd': 許可がありません
(..略..)
```

というように標準出力にファイルやディレクトリのパス、標準エラー出力にエラーが出てきます。

▼図1　bashのインストール

```
$ wget https://ftp.gnu.org/gnu/bash/bash-4.4.tar.gz
$ tar zxvf bash-4.4.tar.gz
$ cd bash-4.4/
$ ./configure && make -j
$ sudo cp bash /usr/local/bin/bash4.4
$ bash4.4 --version
GNU bash, バージョン 4.4.0(1)-release (x86_64-unknown-linux-gnu)
Copyright (C) 2016 Free Software Foundation, Inc.
ライセンス GPLv3+: GNU GPL バージョン 3 またはそれ以降 <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

findにかかわらず、2つの出力が入り混じるものを端末で見るときには、

```
$ find /proc/ 2> /dev/null
 ↑標準出力だけ見る
$ find /proc/ > /dev/null
 ↑エラーだけ見る
```

あるいは、

```
$ find /proc/ > a 2> b
```

というようにファイルに一度貯めてから眺めてみたりということをします。

こういう場合、|&を使うとちょっと便利になります。たとえば次のようにlessやほかのコマンドと組み合わせると、

```
$ find /proc/ |& less
 ↑両方を眺める
$ find /proc/ |& grep ^find: | less
 ↑エラーだけ眺める
$ find /proc/ |& grep -v ^find: | less
 ↑標準出力だけ眺める
```

というように、aやbなどというゴミファイルを作らなくても出力が選別できるようになります。

|&がないからといってそんなに困ることはなさそうですが、手が覚えてしまったら、使う人は頻繁に使うようになるのではないかと筆者は考えます。1点だけ注意ですが、標準出力と標準エラー出力がパイプに渡る順序は不定です。

関連事項として&>>という記号も紹介しておきます。標準出力と標準エラー出力をまとめてファイルに追記するためのものです。

```
$ ls a b &> result
 ↑2出力をまとめてファイルにリダイレクト
$ ls a b &>> result
 ↑2出力をまとめてファイルに追記
$ cat result
ls: 'a' にアクセスできません: 以下略
b
ls: 'a' にアクセスできません: 以下略
b
```

bashにはshoptという内部コマンドがあり、さまざまな機能を有効化／無効化できます。次のように、

```
$ shopt -s globstar
```

とすると、globstarという機能が使えます。

使い方を示します。たとえば次のようにディレクトリやファイルが存在するディレクトリを考えます（xargsは誌面節約のため使用）。

```
$ find | xargs
. ./c ./a ./b ./b/f ./b/e ./b/h ./b/h/i
 ./b/h/j ./b/g
```

このディレクトリでecho **すると、

```
$ echo **
a b b/e b/f b/g b/h b/h/i b/h/j c
```

というように、findと同様にディレクトリ構成を見ることができます[注3]。**がglobstarの記号です。

また、ディレクトリだけ抽出するとき、findだと、

```
$ find -type d
.
./c
./b
./b/h
```

というようにオプションを駆使する必要があります。一方、**/と書くと、

```
$ echo **/
b/ b/h/ c/
```

で同様の情報が抽出できます。

注3）「.」から始まるディレクトリやファイルは出力されません。

また、次のように組み合わせると、/etc/下の拡張子がconfのファイルを、子、孫のディレクトリまですべて含めてすべて抽出できます。

```
$ echo /etc/**/*.conf | tr ' ' '\n' | head -n 3
/etc/adduser.conf
/etc/apache2/apache2.conf
/etc/apache2/conf-available/charset.conf
```

この機能を常時端末で使うときは、.bashrcや.bash_profileに、

```
shopt -s globstar
```

と書いておきます。

また、そのときだけ使いたいなら端末でshopt -s globstarと打ちます。機能を切るときは-sでなく-uを指定します。

1点、これも注意ですが、findと違って表示順をソートしてしまっているので、ファイル数の多いディレクトリに適用すると処理が遅くなる可能性があります。次の計測出力は、筆者の使い込んだUbuntu 16.04 Serverで、上の例を作ったときのディレクトリと、ファイルシステム全体に対してfindと**を実行したときの時間を計測したものです。何度も実行してキャッシュの効いた状態で計測しています。

```
ファイル数の少ないディレクトリ
$ time find ./ &> /dev/null

real 0m0.007s
user 0m0.000s
sys 0m0.004s
$ time echo ** &> /dev/null

real 0m0.001s
user 0m0.000s
sys 0m0.000s

ファイル数の多いディレクトリ
$ time find / &> /dev/null

real 0m0.963s
user 0m0.368s
sys 0m0.588s
$ time echo /** &> /dev/null

real 0m7.291s
user 0m6.092s
sys 0m1.192s
```

このように、ファイル数が多くなるとfindと**の実行時間は逆転します。ファイル数が少ないときにfindのほうが遅いのは、findが外部コマンドで、echoが内部コマンドであるのが原因と推察されます。

shoptで設定できる他の機能

ついでに、shopt関係でfailglobとautocdも紹介しておきます。ほかにもさまざまな機能があるので、興味のある方はman bashでshoptに関する記述を読んでみることをお勧めします。

failglobはバージョン3.0以降に存在している機能です。たとえば、

```
$ touch *.txt
```

と打って、拡張子がtxtのファイルのタイムスタンプを現在の時刻にする操作を行いたいとします。ただ、今いるディレクトリにそのようなファイルがない場合、

```
$ touch *.txt
$ ls
*.txt
```

というように、「*.txt」という名前のファイルができてしまいます。面倒ですし、特殊な文字が入ったファイルがディレクトリにできると、二次災害が起きる場合もあります。

これを避けたい場合、.bashrcにあらかじめ、

```
shopt -s failglob
```

と書いておくと、次のようにエラーになります。

```
$ touch *.txt
-bash: 一致しません: *.txt
```

個人的には、このオプションは有効にしておくと便利だと思うのですが、残念ながらタブ補完と干渉する場合があるようです。

autocdは、cdを抜いてディレクトリの名前を端末に入力してもcdできるという機能です。例を示します。

```
$ shopt -s autocd
$ /etc/
cd /etc/
$ ~
cd /home/ueda
```

慣れると便利かもしれませんが、これが使えない別の環境に行くと混乱するかもしれません。

連想配列、配列へのデータ読み込み

本誌2017年1月号の特集「シェル30本ノック」でも取り上げましたが、バージョン4.0以降では連想配列が使えます。**図2**に使い方の例を示します。

筆者は普段まったくこの機能を使わないので、何に使うのか例を挙げることが極めて困難ですが、1つだけ、使用例を挙げます。Ubuntu Server 16.04と17.04で確認していますが、

```
$ set -x
```

▼図2　連想配列の使い方

```
$ declare -A tel          ←telという連想配列を作る
$ tel[警察]=110           ←警察というキーに110という値をセット
$ tel[消防]=119           ←消防というキーに119という値をセット
$ echo ${tel[警察]}       ←警察の値を呼び出す
110
$ echo ${tel[時報]}       ←存在しないキーを指定
$                         ←何も出てこない
$ echo ${!tel[@]}a        ←キーを列挙
警察 消防
```

と打ってから、次のように設定を再読み込みしつつ、先ほど覚えた|&でdeclareを検索すると、

```
$ source ~/.bashrc |& grep declare
+ grep --color=auto declare
+ source /home/ueda/.bashrc
+++ declare -A _xspecs
```

というような出力が出てきます。出力の一番下の行を見ると、連想配列_xspecsが作られています。ちなみに、この連想配列は補完用で、

```
$ echo ${_xspecs[latex]}
!*.@(?(la)tex|texi|dtx|ins|ltx|dbj)
```

というデータが入っています。この例では、latexというコマンドに関係する拡張子が入っています。

このように入力された文字列に対して即座に情報を引き出したい場合には、連想配列は最適です[注4]。また、bashの文法で補完のデータが管理されているので、bashのユーザがカスタマイズすることも可能となります。

一方、連想配列をやみくもに使用することは、個人的にはお勧めしません。_xspecsの例のように、レスポンシブにユーザに何かを提示するような場合や、フラグの類を管理するくらいにとどめたほうが良いでしょう。データを加工するための処理には、ファイルを使ったほうが良いというのが個人的な主張です。

たとえば初心者向けに連想配列の例を挙げてくれと頼まれた場合、**図2**の警察、消防の例に続けて、

```
for key in ${!tel[@]} ; do
    #$keyと${tel[$key]}を使った処理の例
done
```

というようなものを挙げてしまいがちです。し

注4） この補完について気になる人は、bash-completionというキーワードで調査をお願いします。

かし、これは例としてはあまりよくありません。普通の言語ならこれで良いのですが、初心者にこれを見せてしまうと、パイプラインを使うという発想がなくなってしまいます。また、これも普通の言語と違って、書き方がカッコだらけで変な記号もあってたいへんぎこちない印象は否めません。連想配列の使用は、どうしようもないときに限るべきだとしつこく主張して、連想配列の紹介を終わります。

もう1つ、これも乱用禁止ですが、ファイルから配列にデータを読み込む機能がバージョン4.0以降には存在します。例を示します。次のようにmapfileというコマンドを使います。

```
$ mapfile -t passwd < /etc/passwd
$ echo ${passwd[0]}  ←最初の要素をecho
root:x:0:0:root:/root:/bin/bash
$ echo ${passwd[-1]}  ←最後の要素をecho
ueda:x:1001:1001:ueda,,,:/home/ueda:/
bin/bash
```

この例は、配列「passwd」に/etc/passwdの中身をコピーしてechoで呼び出したというものです。mapfileのオプション-tは改行をとるという意味です。また、Pythonのリストのように、配列にマイナスのインデックスを与えて後ろから要素を参照する方法も、バージョン4.3から加わった機能です。

mapfileは、ちょっとしたパラメータをbashに取り込んで管理する場合には便利です。ただ、外部コマンドをいっさい使ってはならないというシビアな状況でもない限り、加工用のデータをこれで取り込んでfor文で加工するようなことは避けるべきではないかと考えています。

コプロセス

今度は、バージョン4.0から加わった「コプロセス」(co-process。子プロセスではない)を紹介します。本稿で紹介するものの中で一番ややこしいです。実例を最初に示します。

```
$ coproc awk '{print $1*2;fflush()}'
[1] 10872
$ seq 1 3 >&"${COPROC[1]}"
$ read n <&"${COPROC[0]}" ; echo $n
2
$ read n <&"${COPROC[0]}" ; echo $n
4
$ read n <&"${COPROC[0]}" ; echo $n
6
```

この例は、1行目でcoprocというコマンドにawkのコードを渡しています。このawkのコードは1列目に数字が入っているテキストを標準入力から読んで、1列目に2をかけて標準出力から出すというものです。fflushという関数は、出力のバッファの中身を標準出力に追い出すためのもので、毎行の処理が終わったら確実に標準出力に出すために付け加えています。3行目ではseqで1、2、3と3行出力し、>&"${COPROC[1]}"でどこかにリダイレクトしています。配列COPROCには、

```
$ echo ${COPROC[@]}
63 60
```

というように数字が入っています。ですので、>&"${COPROC[1]}"は実行時には>&60になり、60番のファイル記述子の指す先にseqの出力が行ったことになります。

4、6、8行目は逆にread n <&63と、63番のファイル記述子の指す先から文字を読んで変数nに読み込み、echo $nで出力しています。出力は5、7、9行目のように、seqから出した数字が2倍されたものです。つまりawkに指定した処理が適用されて返ってきたものです。

ということは1行目で書いたawkがどこかで動いているようです。このawkのプロセス番号はCOPROC_PIDという変数に格納されていて、

```
$ kill -KILL $COPROC_PID
[1]+  強制終了              coproc COPROC /
awk '{print $1*2;fflush()}'
```

のように止めることができます。

というのが一番簡単なコプロセスの使い方ですが、最初の印象はただただ「なんじゃこりゃ」でした。が、整理すると、

- coprocに指定したコマンドが立ち上がりっぱなしになる
- コマンドの入出力先がCOPROC変数で管理される
- 立ち上がりっぱなしになったコマンドにいつでも読み書きできる

ということで、コマンドをサーバ化する機能だと言えます。

こんな回りくどいことをして何なのかというところですが、有効な例を1つ示します。**リスト1**は、bashとawkで最大公約数を求めるプログラムです[注5]。もちろんbashだけ、あるいはawkだけでも解けますが、この例ではあえて両方使っています。ここで大切なのは、処理の内容ではなく[注6]、関数subをbashのwhileループの中で繰り返し呼び出していることです。sub内に実装したawkは2つ数字を入力すると2つ数字を出力しますが、bashで出力を再度awkに入力しています。余談ですが、7行目のset --は、引数（bashの位置パラメータ）の$1、$2、……の値を、後ろに書いた文字列で置き換えるための記法です。この場合は$()内のコマンドの出力が2つの数字に置き換わり、$1、$2に代入されます。

このようにコマンドの出力を再度同じコマンドに入力するような処理を実装する場合、シェルスクリプトでは基本的にwhile文で書くことになります。このとき、ループの回数が多いとコマンドを立ち上げる時間（この場合はawkを立ち上げる時間）が無視できないほどかかることがあります。

これを**リスト2**のようにコプロセスを使って書き直すと、この問題は基本的にはなくなります。コプロセスには名前をつけることができ、**リスト2**の1～4行目のように書くと関数と同じような記述ができます。名前はsubになります。また、同じ名前のbash配列にアクセス用のファイル記述子が入ります。このコプロセスを使うときは10、11行目のようにややこしい記述が必要ですが、コプロセスのawkは立ち上がりっぱなしなので、ループの回数だけawkを立ち上げる負荷はなくなります。

計算量を比較してみましょう。**図3**のようにtimeをつけて実行してみます。real（実際にかかった計算時間）を比較すると、4倍強、コプロセスを使ったスクリプトの処理時間が短くなっています。この例ではループの回数はたった9回でしたが、ループの回数がもっと大きいと、処理時間の差はもっと大きくなります。

注5）さまざまなバージョンのbashで実行する関係でシバンはつけていません。

注6）アルゴリズムについては「ユークリッドの互除法」でご調査を。

▼リスト1　gcd.no_coproc.bash

```
01: function sub(){
02:     awk '$1>$2{print $1%$2,$2;fflush()}
03:          $1<=$2{print $2%$1,$1;fflush()}'
04: }
05:
06: while [ "$(($1*$2))" -ne 0 ] ; do
07:     set -- $(echo $1 $2 | sub)
08:     echo ">" $1 $2 >&2
09: done
10:
11: echo "$(($1+$2))"
```

▼リスト2　gcd.coproc.bash

```
01: coproc sub {
02:     awk '$1>$2{print $1%$2,$2;fflush()}
03:          $1<=$2{print $2%$1,$1;fflush()}'
04: }
05:
06: a="$1"
07: b="$2"
08:
09: while [ $((a*b)) -ne 0 ] ; do
10:     echo $a $b >&"${sub[1]}"
11:     read a b <&"${sub[0]}"
12:     echo ">" $a $b >&2
13: done
14:
15: echo $((a+b))
```

以上でコプロセスの話は終わりです。まとめると、「コプロセスはプログラミングが面倒だけど、ループで何度も同じコマンドを呼び出すときは使用を検討できる」ということになるかと考えます。ただ、コプロセスの出力が常に得られる状態でないと、readで読み出すときに止まってしまうので、awkの場合はfflushを付ける、sedの場合は-uを付けるなどの、標準出力のバッファリングを防ぐ注意が必要です。

番外編：pipefail

pipefailはバージョン3.0から存在している機能ですが、案外マイナーですのでここで紹介しておきます。pipefailは、パイプの中でコマンドがエラーを起こしたときにスクリプトを止めるための設定です。よくシェルスクリプトに関して「shやbashでは-eオプションをつけて、エラーがあったら止めるようにしましょう」という記述があるのですが、これが結構曲者です。

たとえば**リスト3**のスクリプトは-eオプションが1行目で設定されており、2行目のfalseがエラーを出します。しかし、実行すると2行目で止まらず、次のように、

```
$ bash set_e.bash
NG
```

と3行目が実行されてしまいます。

falseの行で止めるには、**リスト4**の2行目のように、

```
set -o pipefail
```

と入れておきます。すると、次のように止まります。

```
$ bash set_e_pipefail.bash
（何も表示されない）
$
```

-eについてはほかにも止まらない条件があり、また、bashが使えない場合もあるのでpipefailがあるから万全ということではありません。しかし、-eを使ったbashスクリプトでパイプを避けながらコーディングしている場合には、pipefailを利用すると良いでしょう。

▼図3　計算量の比較

```
$ time bash ./gcd.no_coproc.bash 10710 102012
> 5622 10710
略。6回分の出力
> 6 12
> 0 6
6

real 0m0.036s
user 0m0.004s
sys 0m0.000s
$ time bash ./gcd.coproc.bash 10710 102012
略。同じ出力
6

real 0m0.008s
user 0m0.000s
sys 0m0.004s
```

▼リスト3　set_e.bash

```
set -e
false | true
echo "NG"
```

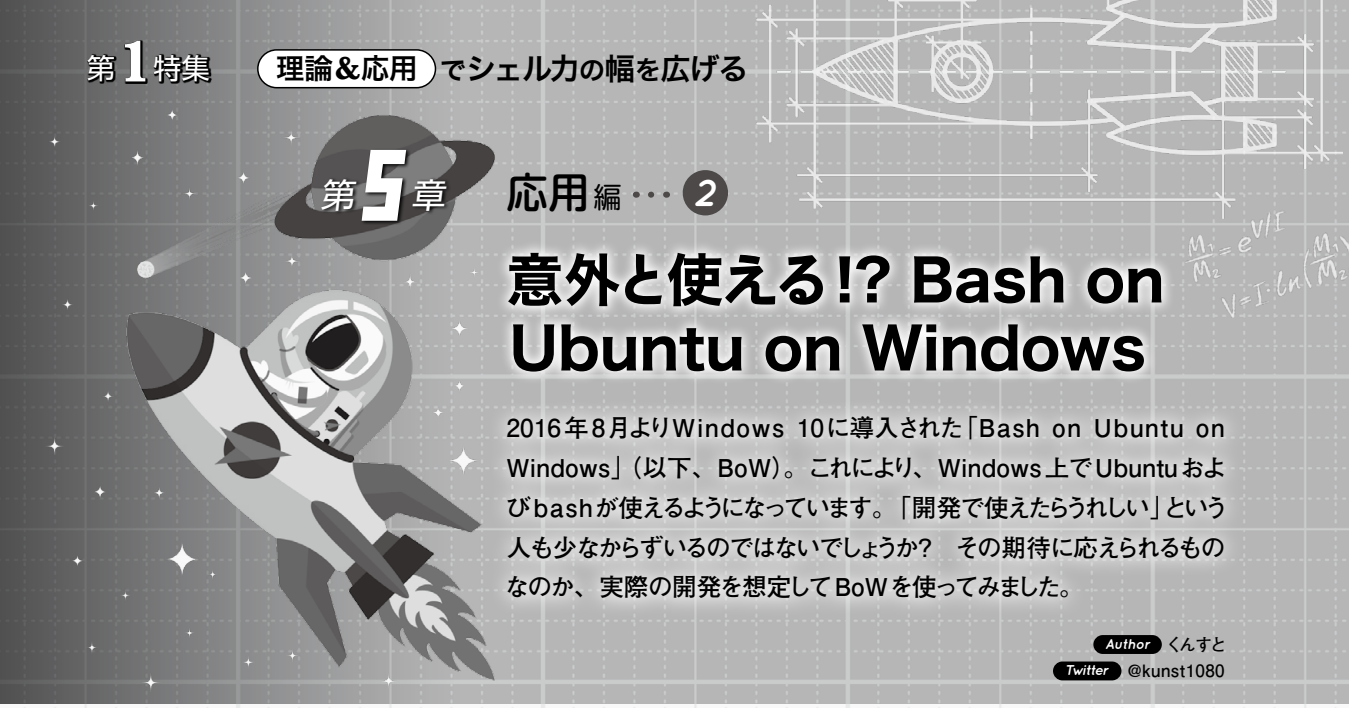
▼リスト4　set_e_pipefail.bash

```
set -e
set -o pipefail
false | true
echo "NG"
```

終わりに

以上、bashの4.0以降の機能を中心に、普段あまりお目にかからないものを紹介しました。このような機能を覚えると使ってみたくなるものですが、とくにスクリプトを書くときは乱用するのではなく、ここぞというところで使えるようになりたいものです。今回の使用例は真面目なものでしたが、おもしろい使い方をいろいろ考えながら[注7]、遊んでみると身につくかと思います。SD

注7）危険シェル芸など。



第5章　応用編 … 2

意外と使える!? Bash on Ubuntu on Windows

2016年8月よりWindows 10に導入された「Bash on Ubuntu on Windows」（以下、BoW）。これにより、Windows上でUbuntuおよびbashが使えるようになっています。「開発で使えたらうれしい」という人も少なからずいるのではないでしょうか？　その期待に応えられるものなのか、実際の開発を想定してBoWを使ってみました。

Author　くんすと
Twitter　@kunst1080

BoWで開発してみる

Windows 10 Anniversary Updateより導入された「Bash on Ubuntu on Windows」は、2017年4月のCreators Updateでより機能が強化され、開発ツールも一通り利用できる状態になってきました。そこで本章では、このBoWを使って実際に開発をしてみることで、本当に「意外と使える」のかを検証します。

環境について

本章での開発・検証は次の環境で行います。

- Windows 10 Pro 64bit、バージョン 1703、ビルド 15063.138（Creators Update）
- 16GB RAM
- Intel Core i7-5500U CPU @ 2.40GHz

そして、BoWはCreators Update後のものを使います（**図1**）。

BoWの導入方法については本稿では省略します。導入の手順はネット上の記事[注2]を参照してください。導入後、スタートメニューより「Bash on Ubuntu on Windows」を選択するか、⊞キー＋Rで表示される「ファイル名を指定して実行」ダイアログにbashと入力することで起動できます。

検証内容について

検証として、実際に次のような簡単なサンプルアプリケーションを開発してみて、使用感を確認していきます[注3]。

- JavaとPlay FrameworkでWeb開発
- Vue.jsとnpmを使ったフロントエンド開発
- Pythonで簡易Twitterクライアント開発

また、上記のほかにも、Mastodonを動かしたりCOBOLの環境を作成したりといった検証も行いました。

▼図1　本章で使うBash on Ubuntu on Windowsの情報 [注1]

```
$ uname -a
Linux DESKTOP-T3CMM04 4.4.0-43-Microsoft #1-Microsoft
Wed Dec 31 14:42:53 PST 2014 x86_64 x86_64 x86_64 GNU/Linux
$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.1 LTS"
```

注1）ターミナルの表記について、とくに記載のない場合はコマンドプロンプトではなくBoWのターミナルを表します。
注2）http://gihyo.jp/admin/clip/01/ubuntu-topics/201608/05
注3）サンプルコードはGitHubにて公開しています。https://github.com/kunst1080/SD201707-sample

それらは、筆者のブログにて紹介していますので、興味がありましたらご覧ください[注4]。

BoWとWindows環境の連携

基本的に、実行環境としてBoWを使用し、エディタやIDEをWindowsで扱うようにします。WindowsからはBoWのフォルダを普通に参照することはできない[注5]ため、Windows上に作業フォルダ（C:¥work）を作成し、このフォルダをWindowsとBoWの両方から参照して開発を行うことにします。BoWからはそのまま使うのは若干面倒ですので、**図2**のように、$HOMEにシンボリックリンクを作成しておくと良いでしょう。

それでは検証していきましょう。

JavaとPlay FrameworkでWeb開発

まずはJavaとビルドツールsbtを使ったPlay FrameworkのWeb開発を行います。以前、個人的に書いたJavaとPlay Frameworkのサンプルアプリケーション[注6]がありますので、本稿でも同様のものを作成します。

OpenJDKとsbtのインストール

BoW環境へ、aptでOpenJDK 8をインストールします。

```
$ sudo apt-get install ↵
openjdk-8-jdk
```

sbtは公式サイトの「Linuxへのsbtのインストール」[注7]の手順に沿ってインストールします（**図3**）。

開発準備（BoW）

BoW環境で、Playアプリケーションのひな形を作成します。ひな形の作成には、sbt newコマンドを使用します[注8]。

```
$ sbt new playframework/play-java-seed.g8
```

このときにいくつか質問をされますが、**図4**のようにname以外は空Enterで大丈夫です。

開発準備（Windows）

BoWでプロジェクトのひな形を作成できましたので、作成されたプロジェクト（C:¥work¥play-java8-sample）をWindowsから開きます。IDEにはIntelliJ IDEAを使用します（IntelliJ IDEAでJavaアプリケーションをコンパイルするには、別途Windows環境にもJDKが必要

▼図2　Windows上の作業フォルダへのシンボリックリンクを作成

```
$ ln -s /mnt/c/work $HOME/work
$ ls -l work
lrwxrwxrwx 1 kunst kunst 11  4月 29 17:58 work -> /mnt/c/work
```

▼図3　sbtをインストール

```
$ echo "deb https://dl.bintray.com/sbt/debian /" | ↵
sudo tee -a /etc/apt/sources.list.d/sbt.list
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 ↵
--recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823
$ sudo apt-get update
$ sudo apt-get install sbt
```

▼図4　ひな形作成時の質問

```
This template generates a Play Java project

name [play-java-seed]: play-java8-sample  ←プロジェクト名を入力
organization [com.example]:  ←空Enter
scala_version [2.11.11]:  ←空Enter
play_version [2.5.14]:  ←空Enter

Template applied in ./play-java8-sample
```

注4）http://www.kunst1080.net/entry/2017/05/26/212415
http://www.kunst1080.net/entry/2017/05/27/001523

注5）http://gihyo.jp/admin/clip/01/linux_dt/201611/21

注6）http://qiita.com/kunst1080/items/51eee39bb1141bd143c4

注7）http://www.scala-sbt.org/release/docs/ja/Installing-sbt-on-Linux.html

注8）https://www.playframework.com/documentation/2.5.x/NewApplication

になります)。

「Import Project」からフォルダを選択し、sbtプロジェクトとしてインポートします(**図5、6**)。インポート後は**図7**の状態になります。

開発

BoWで開発サーバを起動しながらWindowsのIDEでソースコードを編集し開発を進めます。次のコマンドで開発サーバを起動します。

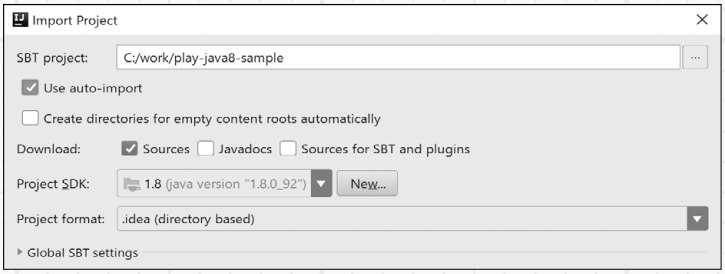
```
$ sbt ~run
```

そして、ブラウザから「localhost:9000」へアクセスします。sbtはホットデプロイに対応しており、IDEでソースを編集し保存すると自動的にコンパイルされ、ブラウザをリロードすると変更後の状態で表示されます(**図8**)。この状態で開発をしていきます。

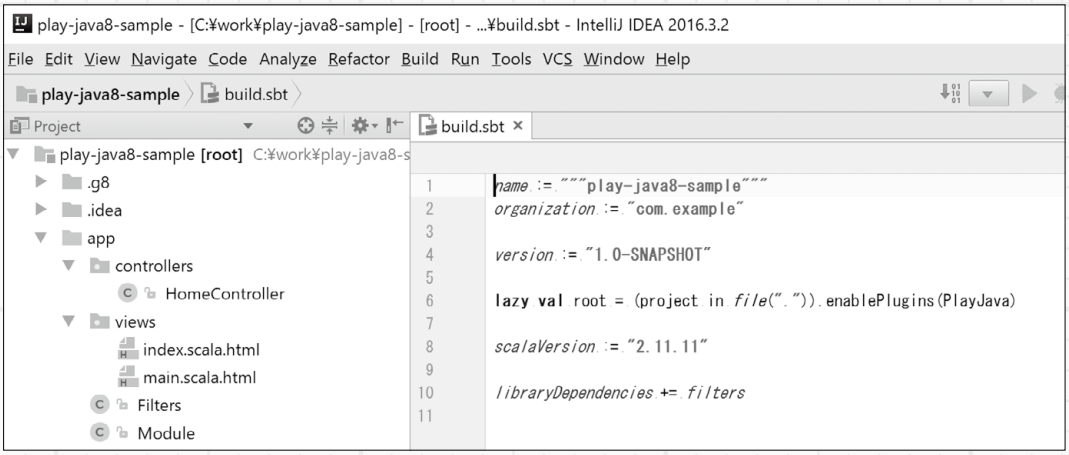
筆者が書き上げたコードはGitHub[注9]にアップしていますので、誌面での掲載は省略します。WindowsでsbtをBoWでsbtを使いましたが、とくに問題もなく書き上げられました。

注9) https://github.com/kunst1080/SD201707-sample/tree/master/play-java8-sample

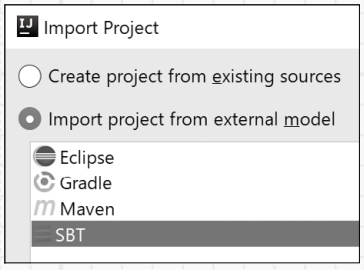
▼図6　IntelliJ IDEAにsbtプロジェクトをインポート(2)



▼図7　インポート直後の状態



▼図5　IntelliJ IDEAにsbtプロジェクトをインポート(1)



▼図8　完成したアプリケーション

Vue.jsとnpmを使ったフロントエンド開発

次に、JavaScriptのフレームワークVue.jsと、パッケージ管理ツールnpmを使ったフロントエンド開発の検証を行います。ここでは、Vue.jsのサイトに紹介されているサンプルのTodo MVC[注10]を作成します。

開発準備（BoW）

BoW環境へ、aptでNode.js（JavaScript実行環境）をインストールします。そのままだと古いバージョンのものがインストールされてしまうため、NodeSource[注11]を使って新しいバージョンのNode.jsがインストールされるようにします。

```
$ wget https://deb.nodesource.com/setup_7.x
$ sudo bash setup_7.x
$ sudo apt-get install nodejs
```

次に、プロジェクトのひな形を作成するため、vue-cli[注12]をインストールします。

```
$ sudo npm install -g vue-cli
```

vue-cliを使ってVue.jsのプロジェクトのひな形を作成します。vue initを実行すると、**図9**のような選択とメッセージが表示されます。

プロジェクトのひな形が作成されたので、npmモジュールをインストールします。

```
$ cd js-sample
$ npm install
```

開発準備（Windows）

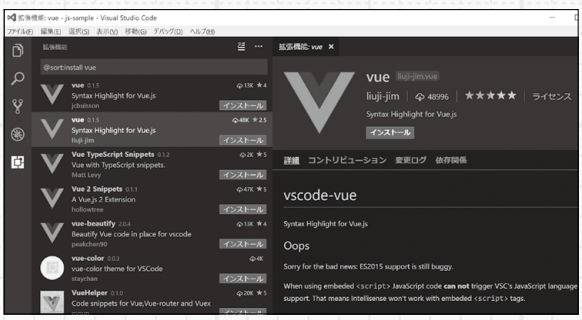
BoWでプロジェクトのひな形を作成できましたので、Windows側でIDEを準備します。今回はIDEにVisual Studio Codeを使用します。そのままではVue.jsで利用する*.vueファイルに対応していないので、プラグインをインストールしましょう（**図10**）。プラグインのインストール後、Visual Studio Codeを再起動すると、*.vueファイルが色付きで表示されるようになります。

IDEの準備ができましたので、BoWで作成したプロジェクト（C:¥work¥js-sample）をIDEで直接開きます。

▼図9　Vue.jsのプロジェクトのひな形を作成する

```
$ vue init webpack js-sample

  This will install Vue 2.x version of the template.

  For Vue 1.x use: vue init webpack#1.0 js-sample

? Project name js-sample
? Project description A Vue.js project
? Author xxxx <xxxx@example.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Setup unit tests with Karma + Mocha? No
? Setup e2e tests with Nightwatch? No

   vue-cli · Generated "js-sample".

   To get started:

     cd js-sample
     npm install
     npm run dev

   Documentation can be found at https://vuejs-templates.github.io/webpack
```

▼図10　vueプラグインのインストール



注10）https://jp.vuejs.org/v2/examples/todomvc.html
注11）https://github.com/nodesource/distributions
注12）https://jp.vuejs.org/v2/guide/installation.html

開発

BoWで開発サーバを起動しながらWindowsのIDEでソースを編集し開発を進めていきます。次のコマンドで開発サーバを起動します。

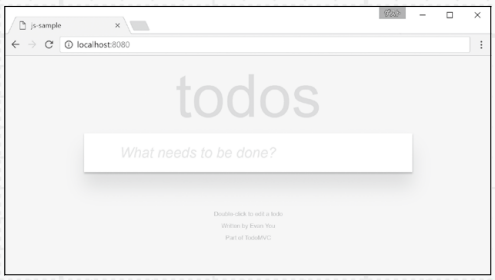
```
$ npm run dev
```

そして、ブラウザから「localhost:8080」へアクセスします。こちらはホットデプロイとホットリロードに対応しており、ソースコードを更新するたびに画面が自動的にリロードされます(**図11**)。この状態で開発をしていきます。

サンプルコードは**注10**のサイトで紹介されているものとほぼ同じため、誌面では省略します。

BoWだけでnpmを利用したため、Windowsへのnpmのインストールはせずに開発できました。BoWでnpmを動かしましたが、とくに問題もなく書き上げられました。

▼図11　完成したアプリケーション



Column ユーティリティ「openコマンド」

2017年4月のCreators Updateによって、BoWから直接Windowsのバイナリを実行できるようになりました。そこで、Macのopenコマンドのように、カレントディレクトリをエクスプローラで開くコマンドを作成しておくと便利です。

「$HOME/bin/open」に**リスト3**のスクリプトを作成し、.bashrcなどで「$HOME/bin」にPATHを通しておくと良いでしょう。これで、openや、open work\js-sampleなどでフォルダを開けます。ただし、BoWの仕様上、Windowsのフォルダ(/mnt/c/以下など)以外のフォルダは開けないので気をつけましょう。

▼リスト3　bin/open

```
#!/bin/bash
if [ $# -eq 0 ]; then
    DIR=.
else
    DIR="$*"
fi
explorer.exe $DIR
```

Column ユーティリティ「パスの変換コマンド」

ターミナルをBoWで、GUIをWindowsで扱っていると、WindowsのパスをBoWで利用したり、その逆を利用したくなったりすることが多々あります。そんなときは**リスト1**、**2**のような変換コマンドを作成しておくと便利です。使用例は次のとおりです。

```
$ echo "/mnt/c/work/python-sample" | u2w
C:\work\python-sample

$ echo "C:\work\python-sample" | w2u
/mnt/c/work/python-sample
```

▼リスト1　WindowsパスをUbuntuパスに変換するコマンド(bin/w2u)

```
#!/bin/bash
sed 's_\\_/_g' | sed -r 's_^(.)(:)_/➥
mnt/\L\1_g'
```

▼リスト2　UbuntuパスをWindowsパスに変換するコマンド(bin/u2w)

```
#!/bin/bash
sed -r 's_^(/mnt/)(.)_\U\2:_g' | sed ➥
's_/_\\_g'
```

Pythonで簡易Twitterクライアント開発

次に、Pythonを使ったTwitterクライアントの開発を行います。開発環境の検証レベルですので、タイムラインの表示のみを行う簡単なCLIクライアントを作成します。

開発準備（BoW）

BoWには標準ではパッケージ管理ツールpipがインストールされていないため、インストールします。

```
$ sudo apt install python3-pip
```

pipを使って、必要なライブラリをあらかじめインストールしておきます。

```
$ pip3 install requests_oauthlib
```

開発準備（Windows）

Pythonの開発検証ではプロジェクトのひな形などは使用しません。「C:¥work¥python-sample」のフォルダをWindowsで作成し、Atomなどのテキストエディタで開きます。

開発

また、今回は開発サーバなども利用しません。Windowsのエディタでソースを編集し、できあがったスクリプトを随時、BoWから実行します。書き上がったコードはGitHub[注13]を参照してください。

では、作成したツールをコマンドラインで実行してみましょう。**図12**のような感じでタイムラインを表示できました。

簡単な検証ではありますが、BoWのPythonからTwitter APIを叩き、データを取得するなどのことができました。

まとめ

Java・JavaScript・Pythonの各種検証を実施し、いずれの環境でも（少しの工夫が必要でしたが）準備をしておけば、普通に開発をしていけました。個人的な使用感としてはWindows上にLinuxの仮想マシンを立ち上げて開発を

注13）https://github.com/kunst1080/SD201707-sample/tree/master/python-sample

Column BoW環境の再構築

BoW環境がうまく動かなかったりして、イチから環境を再構築したい場合の手順を紹介します。

コマンドプロンプトを管理者権限で立ち上げ、次の2つのコマンドを実行します。

```
> lxrun /uninstall
> lxrun /install
```

再構築をしても/rootと/homeフォルダは削除されずに残るため、ユーザを再作成すると.bashrcなどの設定ファイルは元どおり使えます。

▼図12　完成したTwitterクライアント

```
$ ./sample.py
おやすみなさい
これでもう無線のレンジを独占してるテレビ局は正当性の一部を失ったんじゃないの
今世界一ラーメン食べたい自信あるから
RT @miho_karasawa: 影山ヒロノブさん、改めて40周年おめでとうございます！私も影山さんのお声を聴いて育ってきた「影山チルド レン」の一人です。こうしてお祝いをさせていただけるなんて…本当に夢のようです。大先輩方の背中に少しでも近づけるように、これからも頑張ります…
ラーメンの画像を上げるな～～～
```

行っているのとほぼ同じような感じで、十分「使える」という印象です。

Windows上でWeb開発を行う場合、さまざまな開発ツールをインストールする必要があり、環境が汚れてしまう……という問題がありました。実行環境にBoWを使うことで、ある程度環境の分離ができたり、シェルスクリプトによる自動化が行えたりなどのメリットがあるように思います。また、仮想マシンではなくネイティブ環境で実行できるということもあり、今後の選択肢の1つとして考えても良さそうに思います。SD

Visual Studio Codeで無理やりBoWのPythonを使う方法

Pythonの開発検証では、はじめIDEには、Visual Studio Codeを使用する予定でした。しかしながら、Visual Studio CodeからBoWのPythonを直接利用するのは難しく、本稿では使用を見送りました（もちろん、Windows側にPythonを導入しておけば普通に使用は可能でしょうが……）。

しかし、検証の中で一応、Visual Studio CodeからBoWのPythonを無理やり使用できました。ちなみにこの方法ではPylint（Pythonのコードチェッカー）は使えません。

まず、Pythonのプラグインをインストールします（**図13**）。インストール後、Visual Studio Codeを再起動すると、*.pyファイルが色付きで表示されます。

次に、Ctrl＋,でsettings.jsonを開き、編集します（**リスト4**）。そして、settings.jsonに書いた「C:¥work¥python.bat」を**リスト5**の内容で作成します。

設定後Visual Studio Codeでエディタを右クリックし「Run Python File in Terminal」をクリックすると、IDE上の端末で実行されます（**図14**）。

いずれ、Visual Studio CodeでBoWがネイティブサポートされることと思いますので（願望）、そうなったら乗り換えたいですね。

▼図13　Pythonプラグインのインストール

▼リスト4　settings.jsonを編集する

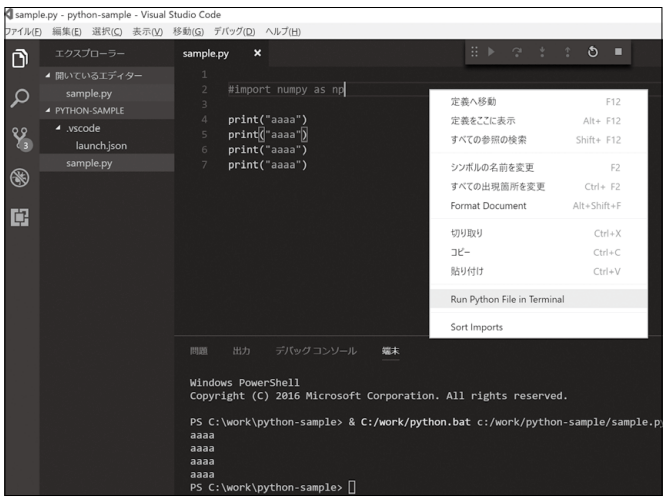
```
// 既定の設定を上書きするには、このファイル内に設定を挿入します
{
    "python.pythonPath": "C:/work/python.bat"
}
```

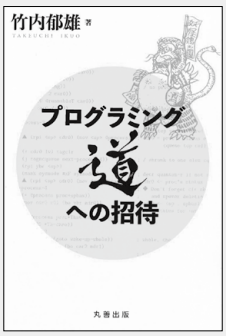
▼リスト5　python.bat

```
@bash.exe -c "echo %* | sed 's_¥¥¥¥_/_g' | ➢
sed -r 's_^(.):_/mnt/¥¥L¥¥1_g' | xargs python3"
```

引数で渡されたPythonスクリプトのパスを、sedでBoWのパスに変換し、bash.exeのpython3コマンドに流し込んでいる

▼図14　Visual Studio Code上で、BoWのPythonを実行

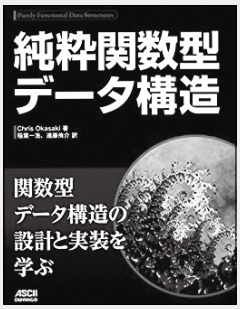




プログラミング道への招待

竹内 郁雄 著
四六判／254ページ
1,800円＋税
丸善出版
ISBN＝978-4-621-30133-3

本書では、オートマトンやフィボナッチ数列を用いたコンピュータのしくみや、"アルゴリズムとは" "プログラムとは" といったプログラミングの基礎が、適所に比喩や余談を交えながらわかりやすく解説されている。教育者でもある著者らしい、これからプログラマへの一歩を踏み出すという方にお勧めの内容だ。しかしこれだけでは、現役プログラマにとっては忘れかけていた記憶が刺激されるくらいだろう。そういった方にも一読してほしいのは、その先に書かれた、プログラミングの「美学」や「道」（老子の示したTao）といった、プログラマの生き様につながる著者の想いだ。たくさんのプログラマに出会い、育ててきた著者の想いにふれることで、自らが初心に返り、より豊かなプログラマ人生を考えるきっかけにしてみてはどうだろうか。

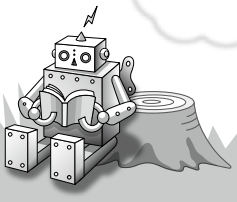


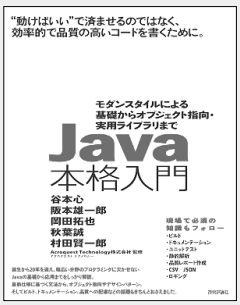
純粋関数型データ構造

Chris Okasaki 著／稲葉 一浩、遠藤 侑介 訳
B5判／216ページ
2,500円＋税
アスキードワンゴ
ISBN＝978-4-04-893056-7

関数型言語で書かれたプログラムの実行速度が遅いというデメリットは、CPUのマルチコア化やコンパイラの性能向上によって緩和されてきたと言われている。加えて、プログラム内で使用するデータ構造を関数型言語に適合したもの（＝関数型データ構造）にすることで実行速度をさらに高める、というのが本書の目的である。内容としては、関数型言語の特徴「破壊的代入が推奨されない」「データが永続性を持つ」「遅延評価」への考察、リンクリスト・木構造・ヒープといった従来のデータ構造の改良、新しい関数型データ構造設計のための技法の紹介と、研究的な側面が強い。本書内のサンプルコードはすべて関数型言語Standard MLで記述されているが、付録として、それらのサンプルコードをHaskellで書き直したものも付いている。

SD BOOK REVIEW





Java本格入門

谷本 心、阪本 雄一郎、岡田 拓也、秋葉 誠、村田 賢一郎 著／Acroquest Technology株式会社 監修
B5変形判／448ページ
2,980円＋税
技術評論社
ISBN＝978-4-7741-8909-3

Javaの入門書ではあるが、文法や機能を堅実に説明していく体裁ではなく、現場での使われ方にフォーカスした1冊となっている。基本文法、型、データ構造などの入門的な章の中にも「名前のつけ方に注意する」「情報共有のために知っておきたい機能」「型にまつわる問題を予防する」といった現場を意識した節があり、チーム開発やその後の保守のヒントになる。ほかのJava本ではそこまで大きく取り上げていないが、本番運用では重要な事柄「日付処理」「スレッドセーフ」を、章を設けて解説しているのも特徴的だ。加えて、ソフトウェアの品質を高めるためのツール群（Maven、Jenkinsなど）とライブラリ（Super CSVやSLF4J・Logbackなど）の使い方も盛り込まれ、読み通すことで入門＋中級者への入り口までカバーできるだろう。

Intel Edison マスターブック

北神 雄太 著
B5変形判／192ページ
2,980円＋税
技術評論社
ISBN＝978-4-7741-8921-5

Intel Edisonは、今話題のIoTプロトタイピングに使えるコンピュータモジュールだ。小型・低消費電力ながら、高性能かつWi-Fi/bluetooth4.0というIoTに必須のインターフェースを備えている。本書の大きな特徴は、Edisonを動かすための電源回路やシリアル通信用のインターフェースの自作に取り組んでいるところだ。Edisonには各種コネクタやArduinoシールドを使うための純正拡張基板が用意されており、必須ではないのだが、イチから自分で組み立てるとできあがったときの喜びもひとしおである。掲載されている作例はLチカなど入門レベルのものから環境測定器まで幅広いレベルがある。実際に試したところ、簡単な配線と設定で通知を送れたのには少し近未来感がありおもしろかった。本書を読んで挑戦してほしい。

第2特集

データの
抽出・加工に強くなる！

MySQL［SELECT文］集中講座

システムやアプリを開発する以上はデータベース、とくにリレーショナルデータベースの扱いを避けては通れず、その中身のデータを操作するためのSQLは、エンジニアの必修科目と言えます。今回はそんなSQLの中で、最も高い頻度で使われるであろう「SELECT文」にスポットを当て、基本構文からORDER BY、GROUP BYを使ったデータの抽出、各種JOINとサブクエリを使ったデータの加工までと、基本から応用までを一気に解説します。本特集では、RDBMSとしてはオープンソースのMySQLを、操作するデータとしては都道府県・市町村の人口・面積データを使います。実務を意識しながら、お手もとの環境でぜひ実践してください。

Author とみたまさひろ

データの抽出・加工に強くなる！
MySQL［SELECT文］集中講座

Part 1

好きな環境ではじめよう

MySQLのインストールとデータベースの用意

このPartではまずMySQLのインストール・初期設定を行います。Ubuntu／CentOS／汎用Linuxバイナリ／Dockerと4つの導入パターンを紹介するので、お好きなものを選んでください。Part1の最後では、Part2、3の演習で使用するデータベースの準備も行います。

Author とみたまさひろ
日本MySQLユーザ会
Twitter @tmtms

データの抽出・加工を担う「SELECT文」は、SQLを扱ううえでもっともよく使う操作ではないでしょうか。本特集で都道府県・市町村に関する実データを操作しながら、SELECT文の基本的・応用的な使い方を身に付けましょう。

はじめようMySQL

まずMySQLをインストールしましょう。いくつかのパターンを示しますので、環境や好みに合ったものを選択してください。

Ubuntu

Ubuntuには標準パッケージとしてMySQLが用意されています。Ubuntu 14.04ではMySQL 5.5または5.6、Ubuntu 16.04以降は5.7が対応しています。インストールするには次のコマンドを実行します。

```
$ sudo apt-get install mysql-server
```

途中でMySQLのrootのパスワード（OSのrootとは異なります）を設定するかどうか尋ねられますが、空のままにしておくこともできます。

起動と停止

Ubuntuでは、インストールされたデーモン系のパッケージは自動的に起動します。mysqldも同様です。OS再起動時にも自動的にmysqldが起動します。手動で起動＆停止したい場合は次のコマンドを実行します。

```
起動
$ sudo service mysql start
停止
$ sudo service mysql stop
```

mysqlコマンドで接続

mysqldに接続するにはmysqlコマンドを使用します。インストール時にパスワードを設定した場合はパスワードの入力が必要です。

```
$ mysql -uroot -p
Enter password: 指定したパスワードを入力
mysql>
```

パスワードを空にしてインストールした場合は、OSのrootユーザからだけ接続できます。

```
$ sudo mysql
mysql>
```

なおUbuntuのmysqlコマンドには、default-character-setオプションが設定されていると日本語が入力できないという問題があるようです。また、LANGやLC_ALL環境変数の値によっても日本語入力ができなくなります。もしmysqlコマンド内で日本語入力ができない場合は、次のように実行してみてください。

```
$ LC_ALL=ja_JP.UTF-8 mysql -uroot -p ➡
--default-character-set=auto
```

▼図1　yumリポジトリの確認

```
$ yum repolist enabled | grep mysql
mysql-connectors-community/x86_64        MySQL Connectors Community          36
mysql-tools-community/x86_64             MySQL Tools Community               47
mysql57-community/x86_64                 MySQL 5.7 Community Server         187
```

▼図2　初期パスワードの確認

```
$ grep 'temporary password' /var/log/mysqld.log
2017-04-30T09:25:50.361916Z 1 [Note] A temporary password is generated for root@localhost: ↵
E-0igVtyZ./2
```

また、次のコマンドで接続を切断します。

```
mysql> quit;
 または
mysql> exit;
```

切断の操作は、ほかの環境でも共通です。

CentOS

CentOS 7には、MySQLはパッケージとして用意されていません。CentOS 6にはありましたが、それもバージョンが5.1と古いのでお勧めしません。

yumで管理したい場合は、Oracleが配布しているrpmパッケージを使用するのが良いでしょう。MySQLの公式ページ[注1]からOSのバージョンに応じたrpmをダウンロードしましょう。このrpmはyumの管理情報が入っているだけですので、ファイルサイズは25KB程度と小さいです。公式ページにはCentOSと書かれたものはありませんが、Red Hat Enterprise Linux / Oracle Linuxと書かれたものが、CentOSでも使用できます。

次のコマンドで、ダウンロードしたrpmをインストールします。

```
$ sudo yum localinstall  ダウンロードしたrpm
```

インストール後はmysql57-communityリポジトリが有効になっています(**図1**)。次のようにインストールします。

```
$ sudo yum install mysql-community-server
```

起動と停止

Ubuntuと異なり、インストールしただけではmysqldは起動しませんが、OSを再起動すると自動起動します。手動で起動・停止したい場合は次のコマンドを実行します。

```
 起動
$ sudo service mysqld start
 停止
$ sudo service mysqld stop
```

mysqlコマンドで接続

CentOSではインストール時にはパスワードの設定は要求されません。そのため、rootユーザに設定された初期パスワードを調べる必要があります。mysqldに接続するための初期パスワードは/var/log/mysqld.logファイルに出力されています(**図2**)。この例ではパスワードは「E-0igVtyZ./2」です。

このパスワードを使うとrootでmysqldに接続できますが、この状態では何のクエリ(命令文)も実行できません。まずパスワードを再設定する必要があります(**図3**)。新しいパスワードは、アルファベットの大文字小文字と数字記号を含み、8文字以上で設定する必要があります。

汎用Linuxバイナリ

Oracleが配布している汎用Linuxバイナリを使用すると、任意のディレクトリにMySQLをインストールできます。このバイナリはOSの

注1) URL https://dev.mysql.com/downloads/repo/yum/

▼図3　mysqlコマンドで接続、パスワードを再設定

```
$ mysql -uroot -p
Enter password: 初期パスワードを入力
mysql> select 123
ERROR 1820 (HY000): You must reset your password using ➥
ALTER USER statement before executing this statement.
mysql> set password='P@ssword1'; 新しいパスワードを設定
mysql> select 123;
+-----+
| 123 |
+-----+
| 123 |
+-----+
```

▼図4　カレントディレクトリにインストール

```
$ tar xf mysql-5.7.18-linux-glibc2.5-x86_64.tar.gz
$ cd mysql-5.7.18-linux-glibc2.5-x86_64
$ ./bin/mysqld --no-defaults --basedir=$(pwd) --initialize
...(略)...
2017-04-26T13:12:59.207693Z 1 [Note] A temporary password➥
 is generated for root@localhost: n++rAyp#k1Uw
```

ディストリビューションに依存しません。自分のホームディレクトリ配下にインストールして、rootではなく自分のユーザ権限で動かすこともできます。

MySQLの公式ページ[注2]からmysql-5.7.18-linux-glibc2.5-x86_64.tar.gz（執筆時点2017年5月の最新版）をダウンロードして、インストールします（**図4**）。最後の行にrootの初期パスワードが出力されるので控えておきます。この例では初期パスワードは「n++rAyp#k1Uw」です。**図4**で使っているオプションの意味は次のとおりです。

- **--no-defaults**：既存のMySQLの設定の影響を受けないようにする
- **--basedir=$(pwd)**：カレントディレクトリ（MySQLを展開したディレクトリ）を基底ディレクトリとする。カレントディレクトリの下のdataディレクトリにデータが作成される
- **--initialize**：データを初期化

注2）URL https://dev.mysql.com/downloads/mysql/

起動と停止

mysqldを起動するには（**図5**）のようにします。

--skip-networkingオプションはTCP/IPを使用しないオプションです。ソケットファイルを使用した接続しか受け付けません。ソケットファイルのパスは--socketオプションで指定します。この例のようにフルパスで指定していない場合は、dataディレクトリからの相対パスとみなされます。

--daemonizeオプションはmysqldがデーモンとしてバックグラウンド動作をする指定です。このオプションを指定しない場合はフォアグラウンドで動作し、mysqldが終了するまでこの端末は使用できなくなります。

--log-errorオプションはエラーログの出力先ファイル名を指定します。--daemonizeオプション指定時には必須です。フルパスで指定しない場合は、--socketオプションと同様にdataディレクトリからの相対パスとみなされます。

mysqldを停止するには、mysqldプロセスに対してTERMシグナルを送ります。mysqldのプロセス番号は「data/ホスト名.pid」というファイルに格納されているため、次のようにしてkillコマンドを実行します。

```
$ kill $(<data/hostname.pid)
```

また、mysqlコマンドからshutdownをクエリとして発行することでもmysqldを停止できます。

```
mysql> shutdown;
```

▼図5　mysqldの起動コマンド

```
$ ./bin/mysqld --no-defaults --basedir=$(pwd) --skip-networking --socket=socket --daemonize➥
 --log-error=error.log
```

▼図6　mysqlコマンドで接続、パスワードを再設定

```
$ ./bin/mysql --no-defaults --socket=data/socket -uroot -p
Enter password: 初期パスワードを入力
mysql> set password='password'; 新しいパスワードを設定
```

▼図7　コンテナ内でmysqldを起動

```
$ docker run --name=mycontainer -e MYSQL_ROOT_PASSWORD=
<初期パスワード> -d mysql/mysql-server:5.7
% docker exec mycontainer yum install -y glibc-common
```

▼図8　mysqldに接続

```
% docker exec -it mycontainer bash -c 'LC_ALL=
ja_JP.UTF-8 mysql -uroot -p --default-character-set=auto'
Enter password: MYSQL_ROOT_PASSWORDで指定した初期パスワード
mysql>
```

mysqlコマンドで接続

インストール時に出力されたパスワードを指定して接続します(**図6**)。なお、--socketオプションはmysqldとは異なり、カレントディレクトリからの相対パスで指定します。

CentOSと同様、パスワードを再設定しないと何のクエリも実行できません。ただしCentOSと違い、新しく設定するパスワードには文字種と文字数の制約はありません。

Docker

もし手元にDockerを実行できる環境があるのなら、試しにMySQLを使ってみるための用途としては最適でしょう。MySQL環境がコンテナ内に閉じているのでホストOSに影響しませんし、不要になった場合は簡単に破棄できます。

OracleのMySQLチームが公開しているDockerイメージ[注3]を使用できます。**図7**のコマンドを実行すると、コンテナ内でmysqldが起動します。初回はイメージをダウンロードするため時間が掛かります。ここでは、--nameオプションによりコンテナに「mycontainer」と名前を付けています。MYSQL_ROOT_PASSWORD環境変数に指定する文字列は、MySQLのrootユーザの初期パスワードです。続くyum installコマンドで日本語入力を可能にする追加パッケージをインストールしています。

起動したmysqldに接続するには、**図8**のコマンドを実行します。LC_ALL=ja_JP.UTF-8と--default-character-set=autoはUbuntuの場合と同様、日本語を入力するための設定です。**図7**のコマンドの実行直後はエラーになることがあります。mysqldが起動するまで十秒程度待ってください。

なお、コンテナを停止するにはdocker stop mycontainer、停止したコンテナを再起動するにはdocker start mycontainerを実行します。コンテナを削除するには、docker rm mycontainerを実行します。

テスト用データベースの用意

ここまでで、mysqlコマンドが実行できる状態になっているかと思います。ここからは、Part2、3で使用するデータベースを用意します。

以降では見やすさのために、クエリの予約語は大文字、それ以外は小文字で記述しますが、MySQLはクエリの大文字小文字は区別しないため、小文字で入力してもかまいません。なお、データベース名とテーブル名は大文字小文字を区別するので注意してください。

また、通常はテーブル名やカラム名に日本語を使用することはないと思いますが、今回は説明のわかりやすさのために日本語にしています。

あらかじめ筆者のリポジトリ[注4]から都道府県データ(prefectures.csv)と市町村データ(cities.csv)をダウンロードし、カレントディレクトリに配置しておいてから、**図9**のクエリを実行します。なお、今回使用するデータは平成27年国

注3) URL https://hub.docker.com/r/mysql/mysql-server/

注4) URL https://github.com/tmtm/sdsample

▼図9　データベースを準備

```
mysql> SET NAMES utf8mb4; 文字コードをUTF-8に設定
mysql> CREATE DATABASE test CHARSET utf8mb4; テスト用データベースを作成
mysql> USE test; テスト用データベースを使用

mysql> CREATE TABLE 都道府県 ( 都道府県テーブルを作成
->     都道府県コード INT NOT NULL,
->     都道府県名 VARCHAR(255) NOT NULL,
->     人口 INT NOT NULL,
->     面積 DOUBLE NOT NULL,
->     PRIMARY KEY (都道府県コード),
->     UNIQUE (都道府県名)
-> );
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE 市町村 ( 市町村テーブルを作成
->     都道府県コード INT NOT NULL,
->     市町村コード INT NOT NULL,
->     区分 INT NOT NULL, -- 1:政令指定都市, 2:市, 3:町村
->     市町村名 VARCHAR(255) NOT NULL,
->     人口 INT NOT NULL,
->     面積 DOUBLE NOT NULL,
->     世帯数 INT NOT NULL,
->     PRIMARY KEY (市町村コード)
-> );
Query OK, 0 rows affected (0.02 sec)

mysql> LOAD DATA LOCAL INFILE 'prefectures.csv' INTO TABLE 都道府県 FIELDS TERMINATED BY ',';
Query OK, 47 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE 'cities.csv' INTO TABLE 市町村 FIELDS TERMINATED BY ',';
Query OK, 1719 rows affected (0.04 sec)
```

勢調査のデータ[注5]を加工したものです。

クエリの説明ですが、CREATE TABLEで作成しているテーブルの形式は**表1、2**のようになっています。

LOAD DATA LOCAL INFILEは、クライアント(mysqlコマンドを実行しているOS)のファイルシステム上に置いてある、タブ区切りテキストが書かれたファイルからテーブルにデータを流し込むための命令です。LOCALを付けずにLOAD DATA INFILEとした場合は、サーバ(mysqldが動作しているOS)のファイルを読み込む指定となります。今回はタブ区切りではなくCSVですので、FIELDS TERMINATED BY ','でカンマ区切りを指定しています。

なお、Dockerを使用している場合はDockerコンテナ内にcsvファイルを置く必要があるため、あらかじめ次のコマンドを実行しておいてください。SD

```
% docker cp prefectures.sql mycontainer:/
% docker cp cities.csv mycontainer:/
```

注5）URL http://www.stat.go.jp/data/kokusei/2015/kekka.htm

▼表1　都道府県テーブル

カラム名	説明
都道府県コード	識別子である整数
都道府県名	文字列
人口	整数
面積	浮動小数点(単位：km^2)

▼表2　市町村テーブル

カラム名	説明
都道府県コード	市町村が属する都道府県の識別子
市町村コード	識別子である整数
区分	1：政令指定都市、2：市、3：町または村
市町村名	文字列
人口	整数
面積	浮動小数点(単位：km^2)
世帯数	整数

データの抽出・加工に強くなる！
MySQL［SELECT文］集中講座

Part 2

SELECTの基本構文から押さえる

データの絞り込み・並び替えをマスターしよう

環境準備とデータの用意が済み、いよいよ実践です。はじめに「テーブルとは何なのか」を押さえたあとは、簡単なSELECT文の使い方からみていきましょう。後半では、単に抽出するだけではなく、並び替えや値の計算、グルーピングによってデータを加工する方法も解説していきます。

Author とみたまさひろ
日本MySQLユーザ会
Twitter @tmtms

本章では、Part1で作ったテーブル「都道府県」（**図1**）を使って、基本的なSELECT文を学んでいきます。

そもそもテーブルとは

SELECTはテーブルから値を抽出するための命令です。SELECTの説明の前にテーブルについて少し説明します。

テーブルは行（ロー、レコード）と列（カラム、フィールド）からなる2次元の表です（**図2**）。ただし、リレーショナル・データベース（以下RDB）でのテーブルは、単純な2次元の表ではありません。行と列は役割が異なり、交換可能ではありません（**図3**）。

RDBのテーブルは、同じ属性項目を持った複数のオブジェクトの集合（**図4**）を2次元の

▼図2　行と列

都道府県名	県庁所在地	人口	面積
北海道	札幌市	5381733	83424
青森県	青森市	1308265	9645
岩手県	盛岡市	1279594	15275
宮城県	仙台市	2333899	7282

行

列

▼図3　行と列は可換ではない

都道府県名	県庁所在地	人口	面積
北海道	札幌市	5381733	83424
青森県	青森市	1308265	9645
岩手県	盛岡市	1279594	15275
宮城県	仙台市	2333899	7282

これはOK

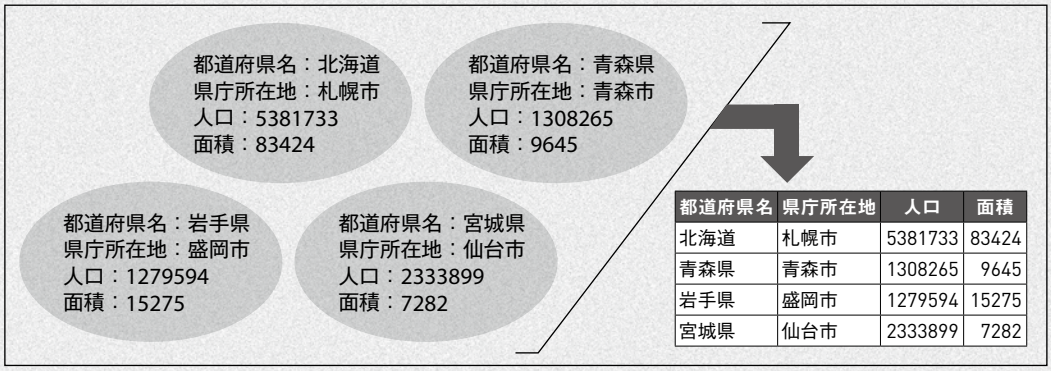
都道府県名	北海道	青森県	岩手県	宮城県
県庁所在地	札幌市	青森市	盛岡市	仙台市
人口	5381733	1308265	1279594	2333899
面積	83424	9645	15275	7282

これはNG

▼図1　都道府県テーブルを表示したところ

```
mysql> SELECT * FROM 都道府県; ←後に解説するSELECT文
+--------------------+----------------+----------+----------+
| 都道府県コード     | 都道府県名     | 人口     | 面積     |
+--------------------+----------------+----------+----------+
|                  1 | 北海道         |  5381733 | 83424.31 |
|                  2 | 青森県         |  1308265 |  9645.59 |
|                  3 | 岩手県         |  1279594 | 15275.01 |
|                  4 | 宮城県         |  2333899 |  7282.22 |
|                  5 | 秋田県         |  1023119 | 11637.54 |
...（略）...
|                 47 | 沖縄県         |  1433566 |  2281.12 |
+--------------------+----------------+----------+----------+
47 rows in set (0.00 sec)
```

▼図4　オブジェクトと表



▼表1　見やすいように並べられたテーブル

都道府県名	県庁所在地	人口	面積
北海道	札幌市	5381733	83424
青森県	青森市	1308265	9645
岩手県	盛岡市	1279594	15275
宮城県	仙台市	2333899	7282
秋田県	秋田市	1023119	11638
山形県	山形市	1123891	9323
福島県	福島市	1914039	13784

▼表2　表1の行・列の順番を入れ替えたもの

面積	都道府県名	人口	県庁所在地
83424	北海道	5381733	札幌市
9323	山形県	1123891	山形市
15275	岩手県	1279594	盛岡市
13784	福島県	1914039	福島市
9645	青森県	1308265	青森市
11638	秋田県	1023119	秋田市
7282	宮城県	2333899	仙台市

表形式で表現したもので、行はオブジェクトを表し、列は属性を表します。

また、行と列には順番はありません。RDBにおいては、**表1**のテーブルと**表2**のテーブルは同じものとなります。列の並びは人間が見やすいように並べられることが多いですが、論理的には順番はないものとして扱われます。

SELECT：カラムの値を取り出す

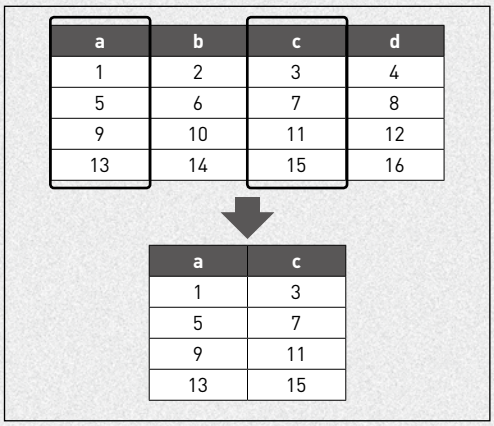
ここからは、実際にSELECTを実行してみましょう。Part1で作成した環境に合わせてmysqlコマンドで接続してください。

SELECTは、指定したカラムの値をテーブルから取り出して返す操作です。結果もテーブルの形式になります(**図5**)。構文としては、SELECTの後ろには取り出すカラム名を並べ、FROMでテーブル名を指定します。

```
SELECT カラム名, ... FROM テーブル名;
```

実行例では次のように改行されているものもありますが、同じ意味となります。

▼図5　SELECTの操作




```
mysql> SELECT カラム名, ...
    -> FROM テーブル名;
```

サンプルデータベースのテーブル「都道府県」から、都道府県名と人口を取り出すには**図6**のように記述します。この例ではテーブルのコードカラムの順にレコードが取り出されていますが、実際には取り出される順番は不定と思って

▼図6　都道府県名と人口を抽出

```
mysql> SELECT 都道府県名, 人口 FROM 都道府県;
+----------------+----------+
| 都道府県名     | 人口     |
+----------------+----------+
| 北海道         |  5381733 |
| 青森県         |  1308265 |
| 岩手県         |  1279594 |
| 宮城県         |  2333899 |
| 秋田県         |  1023119 |
| 山形県         |  1123891 |
| 福島県         |  1914039 |
| 茨城県         |  2916976 |
| 栃木県         |  1974255 |
| 群馬県         |  1973115 |
...(略)...
| 熊本県         |  1786170 |
| 大分県         |  1166338 |
| 宮崎県         |  1104069 |
| 鹿児島県       |  1648177 |
| 沖縄県         |  1433566 |
+----------------+----------+
47 rows in set (0.00 sec)
```

▼図8　カラムに依存しない式を記述

```
mysql> SELECT 1+2+3, 都道府県名 FROM 都道府県;
+-------+----------------+
| 1+2+3 | 都道府県名     |
+-------+----------------+
|     6 | 北海道         |
|     6 | 青森県         |
|     6 | 岩手県         |
|     6 | 宮城県         |
|     6 | 秋田県         |
|     6 | 山形県         |
|     6 | 福島県         |
|     6 | 茨城県         |
|     6 | 栃木県         |
|     6 | 群馬県         |
...(略)...
47 rows in set (0.00 sec)
```

おいたほうが良いでしょう。

SELECTの後ろはカラム名だけではなく、関数などを使用した式も指定できます。たとえば**図7**では、人口を万人単位(ROUNDで小数点以下を四捨五入)で取り出しています。また、カラムに依存しない式を記述することもできます。当然ですが、この場合はすべての行で同じ値になります(**図8**)。このように、SELECTに式を使用すると、結果のテーブルのカラムの名前が式そのものになってしまいます。

ASを使用して結果テーブルのカラム名を変更

▼図7　人口を万人単位で抽出

```
mysql> SELECT 都道府県名, ROUND(人口/10000)
    -> FROM 都道府県;
+----------------+--------------------+
| 都道府県名     | ROUND(人口/10000)  |
+----------------+--------------------+
| 北海道         |                538 |
| 青森県         |                131 |
| 岩手県         |                128 |
| 宮城県         |                233 |
| 秋田県         |                102 |
| 山形県         |                112 |
| 福島県         |                191 |
| 茨城県         |                292 |
| 栃木県         |                197 |
| 群馬県         |                197 |
...(略)...
47 rows in set (0.00 sec)
```

▼図9　人口／面積の値を「人口密度」カラムとして抽出

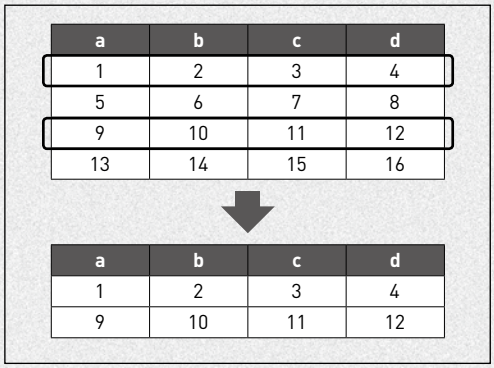
```
mysql> SELECT 都道府県名, ROUND(人口/面積,1)
    -> AS 人口密度
    -> FROM 都道府県;
+----------------+--------------+
| 都道府県名     | 人口密度     |
+----------------+--------------+
| 北海道         |         64.5 |
| 青森県         |        135.6 |
| 岩手県         |         83.8 |
| 宮城県         |        320.5 |
| 秋田県         |         87.9 |
| 山形県         |        120.5 |
| 福島県         |        138.9 |
| 茨城県         |        478.4 |
| 栃木県         |        308.1 |
| 群馬県         |        310.1 |
...(略)...
47 rows in set (0.00 sec)
```

することもできます(**図9**)。なお、ASは省略できます。

カラム名の特別な指定方法として*があります。これはすべてのカラムを意味し、結果のカラムはテーブルのカラムと同じになります(**図1**)。テーブル定義時のカラムの順番が影響する唯一の例外です。

*は、手入力でカラム名をいちいち書くのがめんどくさい場合には便利なのですが、どんなカラムがどの順に返されるのかはSQLからはわからず、本来不要なカラムのデータまで取り出すのは無駄ですので、プログラムから使われることはあまりないと思います。

▼図10　WHEREの動作



a	b	c	d
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

a	b	c	d
1	2	3	4
9	10	11	12

WHERE：対象を絞り込む

WHEREを使えば、SELECTの結果としてテーブル内の全レコードではなく、必要なレコードだけを取り出せます(**図10**)。WHEREの後に指定した式の値が真であるレコードだけが取り出されます。0、FALSE、NULLは偽、それ以外の値は真として扱われます。なおMySQLでは、TRUEは1、FALSEは0を表すシンボルです。構文は次のとおりです。

```
SELECT カラム名, ... FROM テーブル名 WHERE 式;
```

図11では、都道府県テーブルから人口300万人以上の都道府県の名前を抽出しています。

テーブルから値を取り出さないSELECT

SELECTはテーブルから値を取り出すだけでなく、単純に式を評価することだけにも使用できます。FROMを指定しないと式を評価した結果を返します。

```
mysql> SELECT 1+2+3;
+-------+
| 1+2+3 |
+-------+
|     6 |
+-------+
1 row in set (0.00 sec)
```

RDBMSによっては、SELECTにFROMが必須なものがあります。つまりテーブルに関係なく式を評価したいだけであっても、何かしらのテーブルの指定が必要になります。Oracleではそのために、ダミーのテーブルDUALが用意されています。MySQLもDUALに対応しています。

▼図11　人口300万人以上の都道府県の名前を抽出

```
mysql> SELECT 都道府県名 FROM 都道府県
    -> WHERE 人口 >= 3000000;
+-----------------+
| 都道府県名      |
+-----------------+
| 北海道          |
| 埼玉県          |
| 千葉県          |
| 東京都          |
| 神奈川県        |
| 静岡県          |
| 愛知県          |
| 大阪府          |
| 兵庫県          |
| 福岡県          |
+-----------------+
10 rows in set (0.00 sec)
```

```
mysql> SELECT 1+2+3 FROM DUAL;
+-------+
| 1+2+3 |
+-------+
|     6 |
+-------+
1 row in set (0.00 sec)
```

演算子と関数

SELECTやWHEREの式で使用できる演算子や関数は数多くあります。数えたところ、MySQL 5.7では436個ありました。

演算子と関数は、見た目は異なりますがMySQLではどちらもSQLの構文として定義されているので、本質的に違いはありません(ユーザ定義関数を除きます)。よく使われると思われるものをいくつか紹介します。

比較

=(等しい)、<>(等しくない。!=も可)、>(より大きい)、>=(より大きいか等しい)、<(より小さい)、<=(より小さいか等しい)の演算子は、

演算子の左右の2つのオペランド(値、変数)を比較して、真偽値(1または0)を返します(**図12**)。

比較演算子のオペランドの一方または両方にNULLを指定すると結果はNULLになります。NULL同士を比較してもNULLになります。NULLは値ではなく、値が未知であることを示すものです。未知のものと何かを比較しても結果は未知ですので、NULLになるのです。なお、MySQLにはNULLとNULLが等しいとみなす演算子(<=>)もありますが、これは標準SQLにはない演算子です(**図13**)。

BETWEENを使用すると、値が範囲内に入っているかどうかを確認できます。

```
mysql> SELECT 123 BETWEEN 100 AND 200;
+------------------------+
| 123 BETWEEN 100 AND 200 |
+------------------------+
|                      1 |
+------------------------+
1 rows in set (0.00 sec)
```

数値ではなく文字列でも比較ができます。基本的には文字列の先頭から1文字ずつ比較します。なお文字列の比較はコレーション(照合順序)に依存します。MySQLでは通常、アルファベットの大文字と小文字は区別されません。

▼図14　四国の都道府県を抽出

```
mysql> SELECT 都道府県名, 人口 FROM 都道府県
    -> WHERE 都道府県名
    -> IN ('徳島県','香川県','愛媛県','高知県');
+----------------+---------+
| 都道府県名      | 人口    |
+----------------+---------+
| 徳島県          |  755733 |
| 愛媛県          | 1385262 |
| 香川県          |  976263 |
| 高知県          |  728276 |
+----------------+---------+
4 rows in set (0.00 sec)
```

▼図12　比較演算子

```
mysql> SELECT 123=123, 123<>123, 123>123, 123>=123;
+---------+----------+---------+----------+
| 123=123 | 123<>123 | 123>123 | 123>=123 |
+---------+----------+---------+----------+
|       1 |        0 |       0 |        1 |
+---------+----------+---------+----------+
1 rows in set (0.00 sec)
```

▼図13　NULLの挙動

```
mysql> SELECT 123=NULL, NULL=NULL, 123<=>NULL, NULL<=>NULL;
+----------+-----------+------------+-------------+
| 123=NULL | NULL=NULL | 123<=>NULL | NULL<=>NULL |
+----------+-----------+------------+-------------+
|     NULL |      NULL |          0 |           1 |
+----------+-----------+------------+-------------+
1 rows in set (0.00 sec)
```

```
mysql> SELECT 'abc'='ABC';
+-------------+
| 'abc'='ABC' |
+-------------+
|           1 |
+-------------+
1 rows in set (0.00 sec)
```

INを使用すると、値が複数の値のどれかに一致しているかどうかを調べられます(**図14**)。

LIKEで文字列のワイルドカード条件を指定できます。「%」は任意の文字列、「_」は任意の1文字に適合します(**図15**)。

REGEXP(RLIKEでも同様)で正規表現による文字列マッチングができます。ただ日本語などのマルチバイト文字では正常に働きません(**図16**)。

▼図15　名前に「島」という字を含む都道府県を抽出

```
mysql> SELECT 都道府県名 FROM 都道府県
    -> WHERE 都道府県名
    -> LIKE '%島%';
+----------------+
| 都道府県名      |
+----------------+
| 福島県          |
| 島根県          |
| 広島県          |
| 徳島県          |
| 鹿児島県        |
+----------------+
5 rows in set (0.00 sec)
```

数値演算

+、-、*、/で四則演算ができます。また、DIVで整数除算、%(またはMOD)で剰余演算ができます(図17)。

文字列処理

LENGTH()はバイト単位での文字列の長さ、CHAR_LENGTH()で文字単位での文字列の長さを返します。UPPER()、LOWER()は文字列の英字を大文字化、小文字化した文字列を返します。LEFT()、RIGHT()、SUBSTRING()で文字列の先頭、末尾、途中の部分文字を返します(図18)。

論理演算

x AND yは論理積で、xとy

▼図16　正規表現による文字列マッチング(日本語ではうまくいかない)

```
mysql> SELECT 'AAA' REGEXP '^A*$', 'あああ' REGEXP '^あ*$';
+--------------------+----------------------------+
| 'AAA' REGEXP '^A*$' | 'あああ' REGEXP '^あ*$'      |
+--------------------+----------------------------+
|                  1 |                          0 |
+--------------------+----------------------------+
1 rows in set (0.00 sec)
```

▼図17　数値演算

四則演算

```
mysql> SELECT 456+123, 456-123, 456*123, 456/123;
+---------+---------+---------+---------+
| 456+123 | 456-123 | 456*123 | 456/123 |
+---------+---------+---------+---------+
|     579 |     333 |   56088 |  3.7073 |
+---------+---------+---------+---------+
1 rows in set (0.00 sec)
```

整数除算と剰余演算

```
mysql> SELECT 456/123, 456 DIV 123, 456 MOD 123, MOD(456,123);
+---------+-------------+-------------+--------------+
| 456/123 | 456 DIV 123 | 456 MOD 123 | MOD(456,123) |
+---------+-------------+-------------+--------------+
|  3.7073 |           3 |          87 |           87 |
+---------+-------------+-------------+--------------+
1 rows in set (0.00 sec)
```

▼図18　文字列処理

文字列の長さ

```
mysql> SELECT LENGTH('abc'), LENGTH('いろは'), CHAR_LENGTH('いろは');
+---------------+--------------------+-------------------------+
| LENGTH('abc') | LENGTH('いろは')     | CHAR_LENGTH('いろは')     |
+---------------+--------------------+-------------------------+
|             3 |                  9 |                       3 |
+---------------+--------------------+-------------------------+
1 rows in set (0.00 sec)
```

大文字・小文字の変換

```
mysql> SELECT UPPER('aBcD'), LOWER('aBcD');
+---------------+---------------+
| UPPER('aBcD') | LOWER('aBcD') |
+---------------+---------------+
| ABCD          | abcd          |
+---------------+---------------+
1 rows in set (0.00 sec)
```

先頭、末尾、途中の部分文字

```
mysql> SELECT LEFT('あいえうお',2), RIGHT('あいうえお',2), SUBSTRING('あいうえお',3,2);
+-------------------------+--------------------------+-----------------------------------+
| LEFT('あいえうお',2)       | RIGHT('あいうえお',2)       | SUBSTRING('あいうえお',3,2)         |
+-------------------------+--------------------------+-----------------------------------+
| あい                      | えお                       | うえ                                |
+-------------------------+--------------------------+-----------------------------------+
1 rows in set (0.00 sec)
```

の両方が真の場合に1、そうでなければ0を返します。x && yとも記述できます。

```
mysql> SELECT 1 AND 1, 1 AND 0, 0 AND 0;
+---------+---------+---------+
| 1 AND 1 | 1 AND 0 | 0 AND 0 |
+---------+---------+---------+
|       1 |       0 |       0 |
+---------+---------+---------+
1 rows in set (0.00 sec)
```

x OR yは論理和で、xとyのどちらかが真の場合に1、そうでなければ0を返します。x || yとも記述できます。

```
mysql> SELECT 1 OR 1, 1 OR 0, 0 OR 0;
+--------+--------+--------+
| 1 OR 1 | 1 OR 0 | 0 OR 0 |
+--------+--------+--------+
|      1 |      1 |      0 |
+--------+--------+--------+
1 rows in set (0.00 sec)
```

x XOR yは排他的論理和で、xとyのどちらかが真でもう一方が偽の場合に1、そうでなければ0を返します。

```
mysql> SELECT 1 XOR 1, 1 XOR 0, 0 XOR 0;
+---------+---------+---------+
| 1 XOR 1 | 1 XOR 0 | 0 XOR 0 |
+---------+---------+---------+
|       0 |       1 |       0 |
+---------+---------+---------+
1 rows in set (0.00 sec)
```

NOT xは否定で、xが真の場合に0、そうでなければ1を返します。!xとも記述できます。

```
mysql> SELECT NOT 1, NOT 0;
+-------+-------+
| NOT 1 | NOT 0 |
+-------+-------+
|     0 |     1 |
+-------+-------+
1 rows in set (0.00 sec)
```

なお、||はMySQLでは論理和の演算子ですが、標準SQLでは文字列結合の演算子です。MySQLでも、sql_modeシステム変数にPIPES_AS_CONCATを設定することで、文字列結合として働くようになります(**図19**)。

ORDER BYとLIMIT

SELECTにORDER BYを指定することで結果を昇順に並び替えることができます。さらにDESCを指定すると降順に並びます。**図20**は都道府県を面積の大きい順に並び替えた例です。

通常は、WHEREで絞り込んだ数のレコードが返りますが、LIMITを使用すると指定した数のレコードをだけを返すようにできます(**図21**)。LIMITを使用する場合は通常、ORDER BYを指定します。そうでないと、何が返されるか不定になります。

▼図19　設定を変え、||を文字列結合に使う

```
mysql> SELECT 123 || 456;
+-------------+
| 123 || 456 |
+-------------+
|           1 |
+-------------+
1 rows in set (0.00 sec)

mysql> SET sql_mode='PIPES_AS_CONCAT';
以降、||は文字列結合に
mysql> SELECT 123 || 456;
+-------------+
| 123 || 456 |
+-------------+
| 123456      |
+-------------+
1 rows in set (0.00 sec)
```

▼図20　面積の大きい順に都道府県を抽出

```
mysql> SELECT 都道府県名 FROM 都道府県
    -> ORDER BY 面積 DESC;
+-----------------+
| 都道府県名      |
+-----------------+
| 北海道          |
| 岩手県          |
| 福島県          |
| 長野県          |
| 新潟県          |
| 秋田県          |
| 岐阜県          |
| 青森県          |
| 山形県          |
| 鹿児島県        |
...(略)...
47 rows in set (0.00 sec)
```

▼図21　面積の大きい順に4つまで抽出

```
mysql> SELECT 都道府県名  FROM 都道府県
    -> ORDER BY 面積 DESC
    -> LIMIT 4;
+----------------+
| 都道府県名     |
+----------------+
| 北海道         |
| 岩手県         |
| 福島県         |
| 長野県         |
+----------------+
4 rows in set (0.00 sec)
```

集約関数

今まで見てきたように、SELECTはWHEREでレコードを絞り込んだ結果を返すので、結果テーブルのレコード数はWHEREの条件に適合した数になります。しかし、レコードの中身ではなくレコード数だけがほしい場合があります。SELECTにCOUNT(*)を指定すると、結果テーブルのレコード数が返ります(**図22**)。

なお、COUNT(*)は条件に一致するすべてのレコードをカウントしますが、引数に*ではなくカラム名を指定した場合は、カラムの値がNULLでないレコードをカウントします。

このCOUNT()のような関数を集約関数と言います。

集約関数は、形式は通常の関数と同じなのですが、SELECTに集約関数を指定すると、結果テーブルそのものではなく、結果テーブル全体に対する演算結果を返すようにSELECTの動きが変わります。

集約関数はCOUNT()のほかにもいくつかあります。SUM()は合計、AVG()は平均値、MAX()は最大値、MIN()は最小値を計算します(**図23**)。

▼図22　「島」を含む都道府県名とその件数

```
mysql> SELECT 都道府県名 FROM 都道府県
       -> WHERE 都道府県名
       -> LIKE '%島%';
+----------------+
| 都道府県名     |
+----------------+
| 福島県         |
| 島根県         |
| 広島県         |
| 徳島県         |
| 鹿児島県       |
+----------------+
5 rows in set (0.00 sec)
```

レコード数のみ表示

```
mysql> SELECT COUNT(*)  FROM 都道府県
    -> WHERE 都道府県名
    -> LIKE '%島%';
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```

GROUP BYとHAVING

集約関数はテーブル全体ではなく、ある条件でグルーピングしたグループごとに適用できます。たとえば、都、道、府、県のそれぞれの数を数えるには、都道府県名の右端1文字でグループングして、グループ内のレコード数を数えれば良いです(**図24**)。SELECTとGROUP BYの両方にRIGHT(**都道府県名**,1)を記述していますが、ASで別名を付けると簡単に記述できます(**図25**)。

もっと簡単に、SELECTに記述した式の順番をGROUP BYに指定することもできます。今回は1番目の値でGROUP BYしているので、

▼図23　都道府県の人口の合計、平均値、最小値、最大値

```
mysql> SELECT SUM(人口), AVG(人口), MIN(人口), MAX(人口) FROM 都道府県;
+-------------+--------------+-------------+-------------+
| SUM(人口)   | AVG(人口)    | MIN(人口)   | MAX(人口)   |
+-------------+--------------+-------------+-------------+
|   127094745 | 2704143.5106 |      573441 |    13515271 |
+-------------+--------------+-------------+-------------+
1 rows in set (0.00 sec)
```

▼図24　都・道・府・県でグルーピング、数を集計

```
mysql> SELECT RIGHT(都道府県名,1), COUNT(*)
    -> FROM 都道府県
    -> GROUP BY RIGHT(都道府県名,1);
+-------------------------+----------+
| RIGHT(都道府県名,1)       | COUNT(*) |
+-------------------------+----------+
| 府                      |        2 |
| 県                      |       43 |
| 道                      |        1 |
| 都                      |        1 |
+-------------------------+----------+
4 rows in set (0.00 sec)
```

▼図25　図24をASを使って簡略化

```
mysql> SELECT RIGHT(都道府県名,1)
    -> AS type, COUNT(*)
    -> FROM 都道府県
    -> GROUP BY type;
+------+----------+
| type | COUNT(*) |
+------+----------+
| 府   |        2 |
| 県   |       43 |
| 道   |        1 |
| 都   |        1 |
+------+----------+
4 rows in set (0.00 sec)
```

▼図26　図25のGROUP BY部分を簡略化

```
mysql> SELECT RIGHT(都道府県名,1)
    -> AS type, COUNT(*)
    -> FROM 都道府県
    -> GROUP BY 1;
+------+----------+
| type | COUNT(*) |
+------+----------+
| 府   |        2 |
| 県   |       43 |
| 道   |        1 |
| 都   |        1 |
+------+----------+
4 rows in set (0.00 sec)
```

GROUP BY 1と記述できます(**図26**)。

なおGROUP BY使用時に、SELECTに記述できるのは集約関数とGROUP BYに記述した式のみです。

WHEREはGROUP BYよりも前に評価されます。つまりWHEREで絞り込んだ結果テーブルに対してグルーピングされます。全都道府県ではなく、人口300万人以上の都道府県でグルーピングするには**図27**のようにします。WHERE

▼図27　人口300万人以上の都道府県でグルーピング、数を集計

```
mysql> SELECT RIGHT(都道府県名,1)
    -> AS type, COUNT(*)
    -> FROM 都道府県
    -> WHERE 人口 >= 3000000 GROUP BY 1;
+------+----------+
| type | COUNT(*) |
+------+----------+
| 府   |        1 |
| 県   |        7 |
| 道   |        1 |
| 都   |        1 |
+------+----------+
4 rows in set (0.00 sec)
```

▼図28　カラム「count」の値が2以上で絞り込み

```
mysql> SELECT RIGHT(都道府県名,1)
    -> AS type, COUNT(*) AS count
    -> FROM 都道府県
    -> GROUP BY 1
    -> HAVING count >= 2;
+------+-------+
| type | count |
+------+-------+
| 府   |     2 |
| 県   |    43 |
+------+-------+
2 rows in set (0.00 sec)
```

▼図29　ORDER BYが先に評価され、カラム「count」値の昇順に

```
mysql> SELECT RIGHT(都道府県名,1)
    -> AS type, COUNT(*) AS count
    -> FROM 都道府県
    -> GROUP BY 1
    -> ORDER BY count;
+------+-------+
| type | count |
+------+-------+
| 道   |     1 |
| 都   |     1 |
| 府   |     2 |
| 県   |    43 |
+------+-------+
4 rows in set (0.00 sec)
```

の後にGROUP BYされるということは、グルーピングした結果のテーブルに対してWHEREは使えないということです。

GROUP BYの結果をさらに絞り込みたい場合はHAVINGを使用します(**図28**)。なお、ORDER BY、LIMITはGROUP BYよりも後に評価されます(**図29**)。SD

データの抽出・加工に強くなる！
MySQL［SELECT文］集中講座

Part 3

SELECT文を使いこなせますか？

JOINによる結合と、サブクエリを覚えよう

このPartでは、2つ以上のテーブルを使い、いろいろな種類の結合（JOIN）やサブクエリを学習していきます。また、クエリを使った簡単なデータ分析なども行います。少し難しくなりますが、クエリと実際の出力を照らし合わせてイメージしてみてください。

Author とみたまさひろ
日本MySQLユーザ会
Twitter @tmtms

2つのテーブルを結合してみよう

Part2では都道府県テーブルを操作していましたが、ここからはPart1の最後にLOADコマンドで読み込んだ、市町村テーブル（**図1**）も使って、複数のテーブルを使ったSELECT文を学習しましょう。

まず市町村テーブルを用いて、全国に市町村がいくつあるか数えてみます。該当するレコード数をカウントするには、Part2の集約関数の節で解説したCOUNT(*)を使います。

```
mysql> SELECT COUNT(*) FROM 市町村;
+----------+
| COUNT(*) |
+----------+
|     1719 |
+----------+
1 rows in set (0.01 sec)
```

この結果を見ると、日本全国に全部で1,719個の市町村があるようです。

次に、都道府県ごとの市町村の個数を、多い順に10件取り出してみましょう。市町村テーブルには都道府県を示す都道府県コードカラムがあるので、それを使ってPart2で解説したGROUP BYでグルーピングし、さらにCOUNT(*) **市町村数**の結果をORDER BY DESCで降順に並べ、LIMIT 10で10件取り出します（**図2**）。

市町村テーブルには都道府県コードしかないので、都道府県名がわかりません。都道府県コードと都道府県名の対応は都道府県テーブルにあります。まずは都道府県テーブルからPart2で解説したINを使って直接都道府県コードを数値で指定して、この10件の都道府県が何かを取り出してみます（**図3**）。

このように2つの結果を合わせることによって、1位は北海道の179市町村、2位は長野県の

▼図1　市町村テーブルを表示したところ

```
mysql> SELECT * FROM 市町村;   *はすべての要素を表す
+------------------+----------------+--------+------------+----------+----------+------------+
| 都道府県コード   | 市町村コード   | 区分   | 市町村名   | 人口     | 面積     | 世帯数     |
+------------------+----------------+--------+------------+----------+----------+------------+
|                1 |           1100 |      1 | 札幌市     | 1952356  | 1121.26  |     921837 |
|                1 |           1202 |      2 | 函館市     |  265979  |  677.86  |     123950 |
|                1 |           1203 |      2 | 小樽市     |  121924  |  243.83  |      55466 |
|                1 |           1204 |      2 | 旭川市     |  339605  |  747.66  |     155747 |
|                1 |           1205 |      2 | 室蘭市     |   88564  |   80.88  |      43616 |
...（略）...
|               47 |          47382 |      3 | 与那国町   |    1843  |   28.96  |       1080 |
+------------------+----------------+--------+------------+----------+----------+------------+
1719 rows in set (0.00 sec)
```

77市町村、3位は埼玉県の63市町村……ということがわかります。

ですが、こんな別のテーブルの結果をコマンドに手入力するのは、人間がするべき作業ではありません。もちろんプログラムを作れば同様のことはできますが、ここは当然、MySQLにやらせましょう。

1つのSELECTで2つ以上の複数のテーブルを結合して使用できます。この例ですと、**図4**のように実行できます。

▼図2　都道府県ごとの市町村の個数を多い順に10件取り出す

```
mysql> SELECT 都道府県コード, COUNT(*) 市町村数
    -> FROM 市町村 GROUP BY 都道府県コード  GROUP BYでグルーピングを指定
    -> ORDER BY 市町村数 DESC LIMIT 10;  ORDER BY DESCで降順、LIMITで件数を指定
+--------------------+--------------+
| 都道府県コード     | 市町村数     |
+--------------------+--------------+
|                  1 |          179 |
|                 20 |           77 |
|                 11 |           63 |
|                 40 |           60 |
|                  7 |           59 |
|                 23 |           54 |
|                 12 |           54 |
|                 43 |           45 |
|                  8 |           44 |
|                 27 |           43 |
+--------------------+--------------+
10 rows in set (0.00 sec)
```

▼図3　直接都道府県コードを数値で指定して県名を表示する

```
mysql> SELECT 都道府県コード, 都道府県名 FROM 都道府県
    -> WHERE 都道府県コード IN (1,20,11,40,7,23,12,43,8,27);
+--------------------+----------------+
| 都道府県コード     | 都道府県名     |
+--------------------+----------------+
|                  1 | 北海道         |
|                  7 | 福島県         |
|                  8 | 茨城県         |
|                 11 | 埼玉県         |
|                 12 | 千葉県         |
|                 20 | 長野県         |
|                 23 | 愛知県         |
|                 40 | 福岡県         |
|                 43 | 熊本県         |
|                 27 | 大阪府         |
+--------------------+----------------+
10 rows in set (0.00 sec)
```

▼図4　都道府県テーブルと市町村テーブルを結合して結果を表示

```
mysql>  SELECT 都道府県名, COUNT(*) 市町村数 FROM 都道府県
    ->  JOIN 市町村 USING (都道府県コード)
    ->  GROUP BY 都道府県コード ORDER BY 市町村数 DESC LIMIT 10;
+----------------+--------------+
| 都道府県名     | 市町村数     |
+----------------+--------------+
| 北海道         |          179 |
| 長野県         |           77 |
| 埼玉県         |           63 |
| 福岡県         |           60 |
| 福島県         |           59 |
| 千葉県         |           54 |
| 愛知県         |           54 |
| 熊本県         |           45 |
| 茨城県         |           44 |
| 大阪府         |           43 |
+----------------+--------------+
10 rows in set (0.00 sec)
```

結合（JOIN）の種類

テーブルの結合には次のような種類があります。

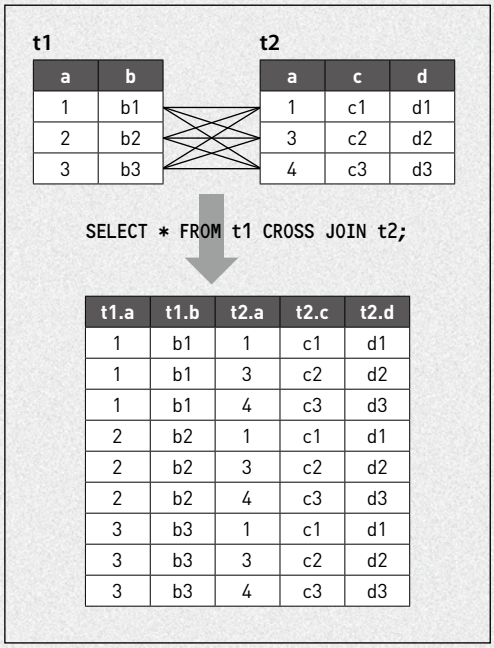
- CROSS JOIN … 交差結合
- NATURAL JOIN … 自然結合
- INNER JOIN …… 内部結合
- OUTER JOIN … 外部結合
- UNION …… 和集合

それぞれを実際に試しながら見ていきましょう。

CROSS JOIN（交差結合）

CROSS JOINを使用すると、2つのテーブル（テーブル1とテーブル2とします）のすべてのレコードの組み合わせで新しいテーブルを作ることができます。テーブルを次のよう

▼図5　CROSS JOIN（交差結合）のイメージ



に指定します。

```
FROM テーブル1 CROSS JOIN テーブル2
```

単純に「,」で連結して次のように記述することもできます。

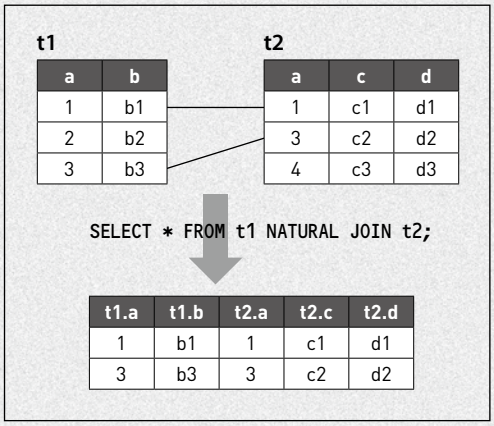
```
FROM テーブル1, テーブル2
```

実際の動作を**図5**に示します。

たとえば1,000行程度のテーブルでも、その2つをCROSS JOINすると論理的には100万レコードもの一時テーブルができあがります。そこからWHEREやGROUP BYを使用して目的のレコードを抽出することになります。これはあくまでも論理的にはそのような操作になるということであって、実際のRDBMSの処理ではテーブルの結合時にもWHEREなどの条件が考慮されるので、必ずしもすべてのレコードを持った巨大な一時テーブルが作成されるわけではありません。

結合したテーブルに対するWHEREで、条件を記述したり最終的に抽出するカラムを選択したりする際、両方のテーブルで同じ名前のカラムがある場合はどちらのテーブルのカラムかわからないので、「テーブル名.カラム名」のように指定します。

▼図6　NATURAL JOIN（自然結合）のイメージ



NATURAL JOIN（自然結合）

NATURAL JOINを使用すると、左右両方のテーブルの同じ名前のカラムが同じ値であるレコードが結合されます。テーブルを次のように指定します。

```
FROM テーブル1 NATURAL JOIN テーブル2
```

実際の動作を**図6**に示します。

INNER JOIN（内部結合）

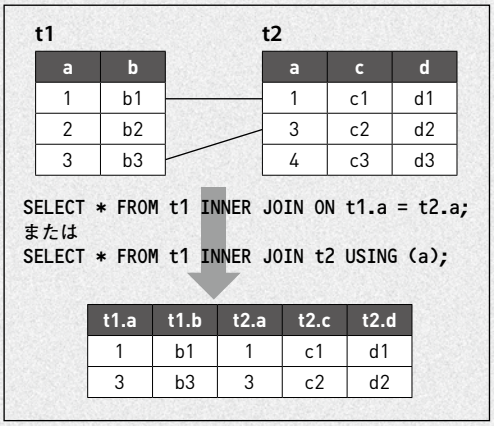
INNER JOINは、条件を満たすレコード同士が結合され、条件を満たさないレコードは捨てられます。条件はUSINGまたはONで指定します。USINGは両方のテーブルに存在するカラムを指定して、カラムが同じ値のレコード同士が結合されます。ONは式の結果が真になるレコード同士が結合されます。テーブルを次のように指定します。

```
FROM テーブル1 INNER JOIN テーブル2
 USING（カラム1, カラム2, ...）
FROM テーブル1 INNER JOIN テーブル2 ON 式
```

※INNERは省略可能です

実際の動作を**図7**に示します。

▼図7　INNER JOIN（内部結合）のイメージ



▼図8　OUTER JOINの種類

```
FROM テーブル1 LEFT OUTER JOIN テーブル2
 {USING または ON}
FROM テーブル1 RIGHT OUTER JOIN テーブル2
 {USING または ON}
FROM テーブル1 FULL OUTER JOIN テーブル2
 {USING または ON}
```

図4の都道府県ごとの市町村数を求めたSELECTでは、このINNER JOINを使用しています。なお、MySQLではCROSS JOINとINNER JOINの区別はありません。

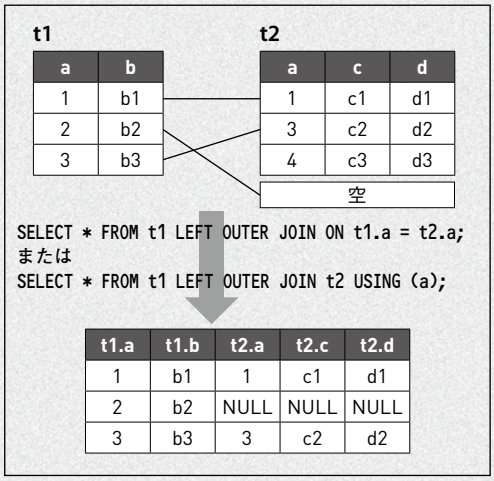
OUTER JOIN（外部結合）

OUTER JOINは、INNER JOINと同様に条件を満たすレコード同士が結合されますが、条件を満たさないレコードも残されます。テーブルを**図8**のように指定します（OUTERは省略できます）。

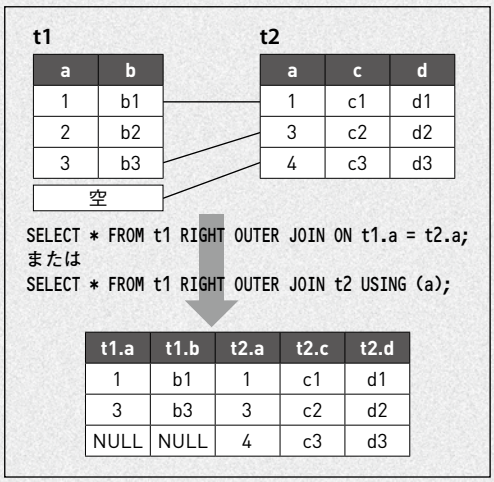
LEFT OUTER JOIN（**図9**）はクエリにおいて左側のテーブルのレコードが残り、RIGHT OUTER JOIN（**図10**）は右側のレコードが残ります。FULL OUTER JOIN（**図11**）は両方のテーブルのレコードが残ります。いずれも対応するレコードがないテーブルのカラムはNULLになります。OUTER JOINを使って村を持たない都道府県を調べてみます（**図12**）。

この例では、都道府県テーブルと市町村テーブルを、都道府県コードが同じでかつ市町村名

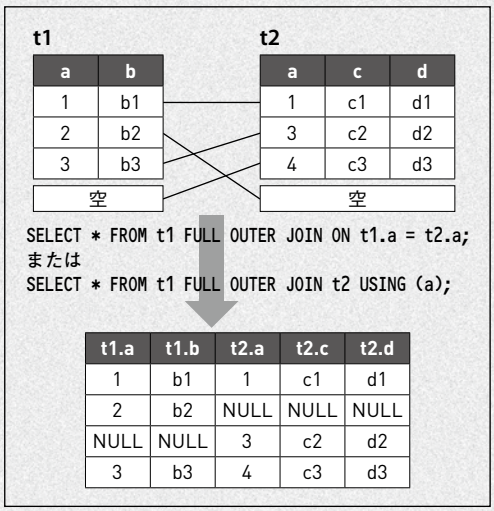
▼図9　LEFT OUTER JOIN（左外部結合）のイメージ



▼図10　RIGHT OUTER JOIN（右外部結合）のイメージ



▼図11　FULL OUTER JOIN（完全外部結合）のイメージ



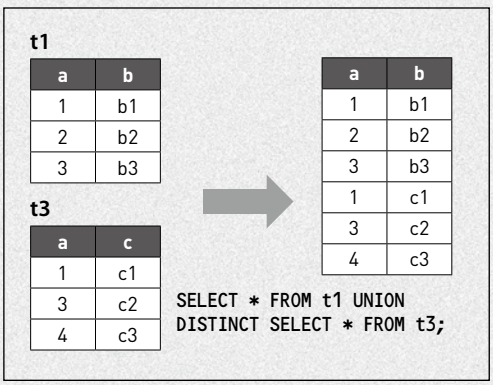
▼図12　OUTER JOINを使った例

```
mysql> SELECT 都道府県名 FROM 都道府県 LEFT JOIN 市町村
    ->  ON 都道府県.都道府県コード=市町村.都道府県コード AND 市町村名
    ->  LIKE '%村' WHERE 市町村コード IS NULL;
+----------------+
| 都道府県名     |
+----------------+
| 三重県         |
| 佐賀県         |
| 兵庫県         |
| 山口県         |
| 広島県         |
| 愛媛県         |
| 栃木県         |
| 滋賀県         |
| 石川県         |
| 福井県         |
| 長崎県         |
| 静岡県         |
| 香川県         |
+----------------+
13 rows in set (0.00 sec)
```

が「村」で終わるという条件で結合しています。LEFT JOINですので条件を満たさない場合でも都道府県テーブルのレコードは残り、市町村テーブルはNULLになります。WHEREで市町村テーブル側のカラムがNULLであるレコードを抽出すれば条件を満たさないレコード、この場合は市町村が「村」で終わるレコードがない県が抽出されるというわけです。

このようにOUTER JOINは、条件を満たさないレコードを抽出するのによく使われます。

▼図13　UNION（和集合）のイメージ



▼図14　異なる県で同じ名前の市町村名を検索

```
mysql> SELECT 市町村名 FROM 市町村
    ->  GROUP BY 市町村名
    ->  HAVING COUNT(*) > 1;
+--------------+
| 市町村名     |
+--------------+
| 伊達市       |
| 南牧村       |
| 南部町       |
| 太子町       |
| 小国町       |
...（略）...
| 那珂川町     |
| 金山町       |
| 高山村       |
| 高森町       |
+--------------+
26 rows in set (0.00 sec)
```

UNION（和集合）

JOINはテーブルとテーブルを横方向に結合してカラムを増やす操作でしたが、UNIONはテーブルとテーブルを縦方向に結合してレコードを増やす操作です。

```
SELECT …… UNION DISTINCT SELECT ……
SELECT …… UNION ALL SELECT ……
```

UNIONの前後に書かれたSELECTの結果を結合します（**図13**）。DISTINCTは結合結果から重複レコードを削除します（DISTINCTは省略可能です）。ALLはすべてのレコードを採用します。結合するテーブルはカラムの数が同じである必要があります。結合結果のカラム名は最初のテーブルのカラムが使われます。

サブクエリ

日本国内には異なる県で同じ名前の市町村名があります。調べてみましょう。市町村テーブルの中から重複が1件よりも多い市町村名を調

べるには**図14**のようにします。

26個の市町村が重複しているようです。この市町村がどの都道府県に所属しているかを調べるにはどうすればいいでしょうか。このクエリの結果の市町村名をINに羅列して**図15**のようなSELECTを実行すれば市町村名と都道府県名のペアが得られます。

実は、INの括弧内には値のリストだけではなくSELECTを記述できます。ですので、今回の場合は**図16**のように記述できます。

このようにSELECTの中にSELECTを含めたクエリをサブクエリといいます。INだけではなくいくつかの種類があります。

▼図15　図14の重複した市町村の都道府県を調べる

```
SELECT 市町村名,都道府県名 FROM 市町村 JOIN 都道府県
 USING（都道府県コード）WHERE 市町村名
 IN（'伊達市','南牧村','南部町','太子町', ……);
```

▼図16　SELECTの中にSELECTを含んだ例（サブクエリ）

```
mysql> SELECT 市町村名,都道府県名 FROM 市町村 JOIN 都道府県
    ->  USING（都道府県コード）WHERE 市町村名
    ->  IN（SELECT 市町村名 FROM 市町村 GROUP BY 市町村名
    ->  HAVING COUNT(*) > 1）ORDER BY 市町村名;
+--------------+----------------+
| 市町村名     | 都道府県名     |
+--------------+----------------+
| 伊達市       | 福島県         |
| 伊達市       | 北海道         |
| 南牧村       | 長野県         |
| 南牧村       | 群馬県         |
...（略）...
| 金山町       | 福島県         |
| 金山町       | 山形県         |
| 高山村       | 長野県         |
| 高山村       | 群馬県         |
| 高森町       | 熊本県         |
| 高森町       | 長野県         |
+--------------+----------------+
59 rows in set（0.01 sec）
```

比較演算子

SELECTが単一のレコードを返す場合、比較演算子で比較することができます。次の式はSELECTの結果のxの値がaと等しい場合に真になります。

```
a = (SELECT x FROM ……)
```

SELECTが複数列を返す場合は括弧で括って比較できます。次の式はSELECTの結果のx,yの値がそれぞれa,bと等しい場合に真になります。

```
(a,b) = (SELECT x,y FROM ……)
```

IN

先ほど示したようにSELECTが複数のレコードを返す場合、INを使用して比較することができます。次の例はSELECTの結果の複数のレコードのxの値のどれかがaと等しい場合に真になります。

```
a IN (SELECT x FROM ……)
```

比較演算子と同様に括弧で括って複数列を返す場合でも記述できます。

```
(a,b) IN (SELECT x,y FROM ……)
```

EXISTS

EXISTSはSELECTが結果を返す場合に真となります。

```
EXISTS (SELECT ……)
```

FROM

テーブルをSELECTした結果もまたテーブルです。通常のテーブルと同様にFROMに指定することができます。ただしASでテーブルに名前（この場合はT）をつける必要があります。

```
SELECT …… FROM (SELECT …… FROM ……)
 AS T WHERE ……
```

この形式のサブクエリを使用して、都道府県

の面積と都道府県内の市町村の面積の合計が同じかどうか調べるクエリを書いてみます。

各都道府県の市町村の面積の合計は次のクエリで求められます。

```
SELECT 都道府県コード,SUM(面積)
 AS 市町村合計面積 FROM 市町村
 GROUP BY 都道府県コード;
```

この結果テーブルと都道府県テーブルを結合して、面積に差があるものを抽出してみます。面積は浮動小数点数なので誤差が発生するために、差の絶対値が1km^{2}以上という条件にしています。

結構差がありました(**図17**)。

今回使用したテーブルは国勢調査のデータから抜き出したものですが、面積はいくつか調整されていて都道府県と市町村の合計が合わないものがあるのです(たとえば北海道には北方領土の面積が含まれているなど)。

面積ではなく人口で同様の処理を行うと、都道府県の人口と市町村の合計人口が等しいことがわかります(**図18**)。

おわりに

単一のSELECT文は、論理的にはJOIN、WHERE、GROUP BY、HAVING、ORDER BY、LIMITの順に評価されます。そのSELECT結果のテーブルをUNIONで連結したり、別のSELECTの一部として使用したりすることもできます。

今まで見てきたように、この過程のそれぞれの演算の入力はテーブルであり、出力もまたテーブルです。つまりSELECTは1つまたは複数のテーブルを元に新たなテーブルを作るという、テーブルを単位とした演算処理なのです。レコードやカラムだけではなくテーブルを意識することで、複雑なSELECT文でも理解しやすくなると思います。SD

▼図17　都道府県の面積と都道府県内の市町村の面積の合計が同じかどうか調べる

```
mysql> SELECT 都道府県名,ROUND(ABS(面積-市町村合計面積),1)
    ->  AS 面積の差 FROM 都道府県 JOIN
    ->  (SELECT 都道府県コード,SUM(面積) AS 市町村合計面積 FROM 市町村
    ->  GROUP BY 都道府県コード)
    ->  AS t USING (都道府県コード)
    ->  WHERE ABS(面積-市町村合計面積) > 1;
+----------------+--------------+
| 都道府県名     | 面積の差     |
+----------------+--------------+
| 北海道         |       4970.8 |
| 秋田県         |         22.0 |
| 東京都         |          4.8 |
| 山梨県         |          4.7 |
| 愛知県         |          3.1 |
| 岡山県         |          7.0 |
+----------------+--------------+
6 rows in set (0.00 sec)
```

▼図18　面積ではなく人口で比較した

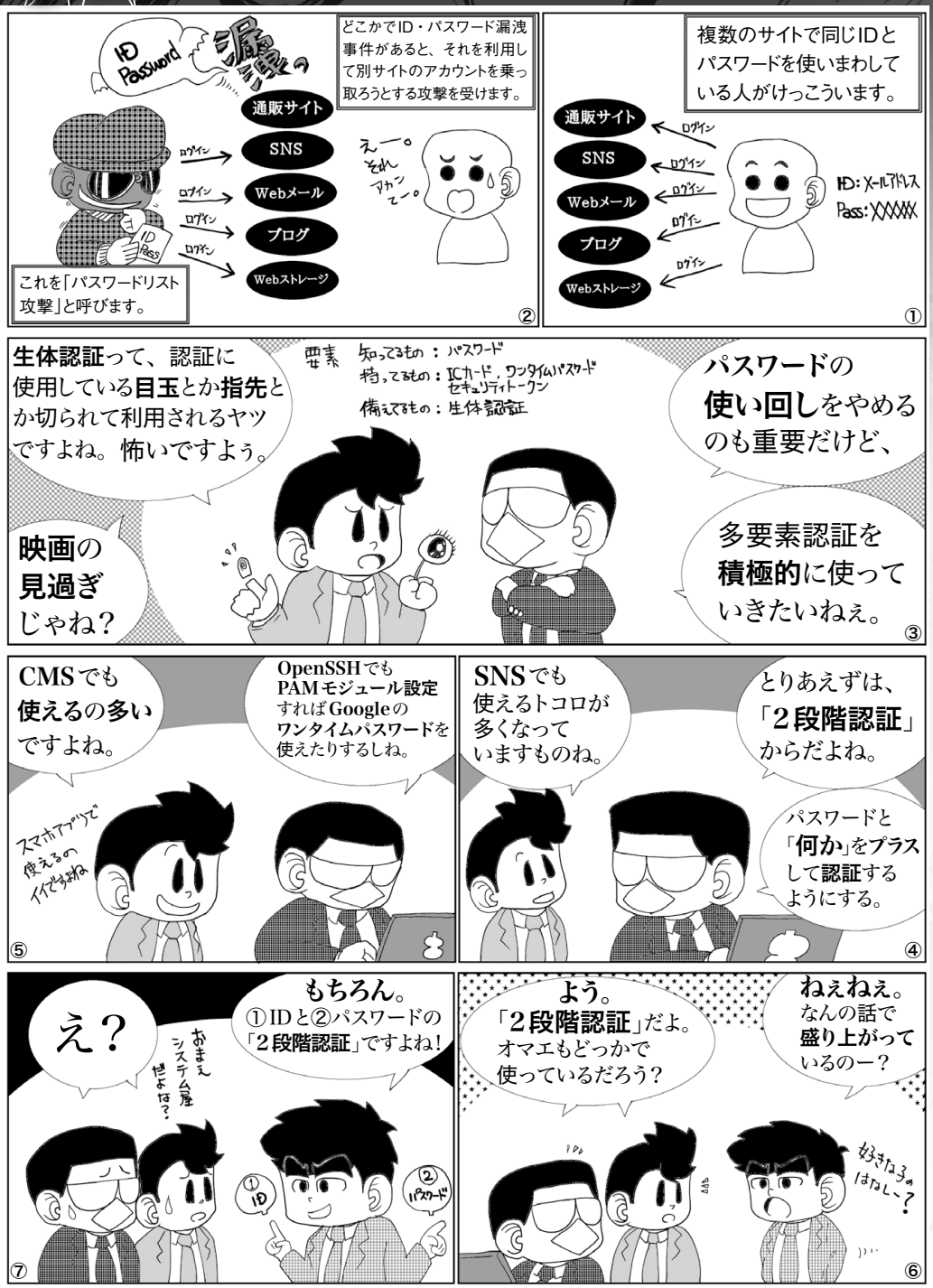
```
mysql> SELECT 都道府県名 FROM 都道府県 JOIN
    ->  (SELECT 都道府県コード,SUM(人口) AS 市町村合計人口
    ->  FROM 市町村 GROUP BY 都道府県コード)
    ->  AS t USING (都道府県コード)
    ->  WHERE 人口 <> 市町村合計人口;
Empty set (0.00 sec)
```

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第41回 2段階認証

GWがあっという間に過ぎて茫然自失のまま日々を過ごす担当のまわら、日々ジョギングにいそしみリア充を極めるくつな先生のマンガが読めるのは本誌だけ！



to be Continued

パスワードリスト攻撃の被害が後を絶ちません。防衛策は、パスワードの使い回しをやめること、多段認証を使うこと。OpenSSHサーバでもGoogleワンタイムが使えるのを最近試しました。SNSやCMS、Webストレージ等でも利用できるので積極的に使いたいですね。いつぞやの芸能人チョメチョメ画像漏洩もパスワードリスト型攻撃が原因だったと聞きます。ちゃんと対策しないと、次はアナタのチョメチョメ画像が世に放たれるかもしれませんよ。「"チョメチョメ"が古い」と指摘している前に、アナタのアカウントが本当に大丈夫か、パスワードを使いまわしてないかちゃんとチェックしましょうね！　自分のアカウントは自分で守る！　ダディとの約束だぞ！

特別企画

さまざまなシステムで使用されている ハッシュ関数を使いこなしていますか？ 後編

ソフトウェア開発での実装ポイント

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。この特集では前編に引き続き、ハッシュ関数を実際に利用する際に注意すべき点や、OSSではどのように実装されているかなど、もう少し掘り下げてハッシュ関数について解説します。

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

前回のおさらい

前編ではMD5やSHA-1など、ハッシュ関数とは何か、ハッシュ関数が持つべき特性はどのようなものかを解説しました。後編となる今回は前編の内容をさらに深掘りする内容となりますので、簡単に前編の復習をしましょう。

ハッシュ関数とは

あるデータが与えられ、そのデータからあるアルゴリズムで小さな値を算出したとします。このとき、「あるアルゴリズム」のことをハッシュ関数、「小さな値」のことをハッシュ値と呼びます。また、データからハッシュ値を計算することを「ハッシュ化」と呼ぶこともあります。MD5、SHA-1、SHA-512などがコンピュータプログラムでよく利用されるハッシュ関数の例です。

ハッシュ関数の持つ特性とそれを活かした利用例

ハッシュ関数はいくつかの特性を持ち、その特性を活かしてコンピュータプログラムで利用されています。

「ハッシュ関数が持つ特性」

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

また、ハッシュ関数を利用する場合の重要な留意事項として、ハッシュ値は衝突することがあることを解説しました。つまり、ハッシュ関数は別のデータに対して同じハッシュ値を出力することがあります。これは後編の内容にも深くかかわってきますので頭の片隅に置いておいてください。

そして、本誌6月号前編の「ハッシュアルゴリズムの脆弱性」の節で説明に誤りがありました。「**第2原像計算困難性**が破られた」と記述しましたが、正しくは「**衝突困難性**が破られた」でした。この場合の計算量は2^{160}ではなく2^{80}です。お詫びして訂正します。Googleが発見した方法では、これを2^{63}回の計算で実現したようです。そしてその計算に利用した計算資源は1つのCPUで6,500年換算とのことです。

ハッシュ関数使用時の注意

ハッシュ関数はとても便利に使うことができます。しかし、その特性を理解したうえで正しく使わないと問題が発生することがあります。ここでは前編で述べた3つの利用例について、その注意点を紹介します。

ファイルの同一性チェックでの注意点

入手したファイルから計算したハッシュ値と公式の配布元が公開しているハッシュ値を比較して、自分が入手したファイルが間違いなく公式の配布元が配布しているファイルと同じことを確認する、というのがこの利用法でした(**図1**)。

この利用方法では、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」という性質を利用していましたが、その後者について前述のとおり、ハッシュ値は衝突する可能性があります。

つまり、次のようなストーリーで、ファイルの同一性チェックが無効化されてしまいます。

①あるソフトウェアAの公式サイトにソフトウェアAのファイルとハッシュ値が公開される
②攻撃者が悪意のあるソフトウェアBを作成し、そのハッシュ値がソフトウェアAのハッシュ値と同じになるようにファイルを作成する
③攻撃者はソフトウェアAと称してソフトウェアBを配布する
④ソフトウェアAのつもりでソフトウェアBを入手したユーザは、ソフトウェアAの公式サイトにアクセスし自分が入手したファイルから計算したハッシュ値とソフトウェアAのハッシュ値が同じことを確認し、実行してしまう

このような攻撃に対しての対策は、ハッシュアルゴリズム単体ではとくになく、第2原像計算困難性の高いハッシュアルゴリズムを使うしかありません。ハッシュアルゴリズム単体でなければ、2つのアルゴリズムで計算したハッシュ値を両方比較する、などの工夫はできます。前述のUbuntuの公式サイトではMD5とSHA-1、SHA-256が公開されていますので、入手したファイルについてその3つのハッシュ値を計算して比較して一致すればまずそのファイルの内容は公式サイトで配布されているファイルと同じものであると言えるでしょう。もちろん、これでも3つとも衝突させている可能性もゼロではありません。しかし、ハッシュアルゴリズム1つに対する場合でも非常に難しい攻撃ですので、3つとも衝突させている可能性はゼロとみなして良いでしょう。

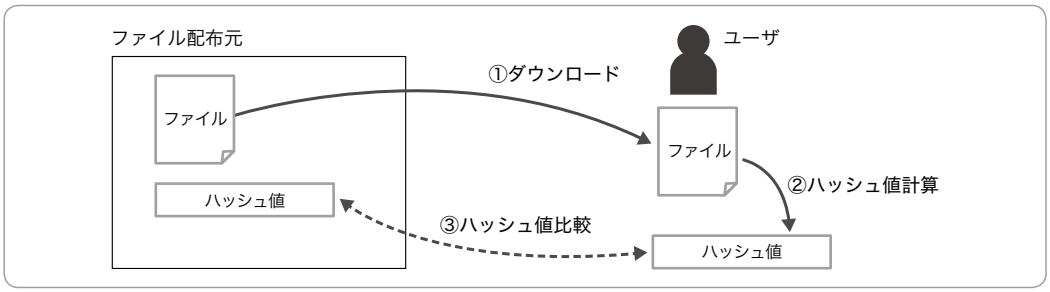
推測しにくいURLの生成での注意点

Ruby on RailsやCakePHP、Drupalなど、連番の数字をテーブルのキーとしてデータベースに持つ設計のもとで、URLのパラメータにレコードを指定したい場合に、指定するパラメータをテーブルのキーをハッシュ化したものとすることでURLを推測しにくいようにする、というのがこの利用法でした(**図2**)。

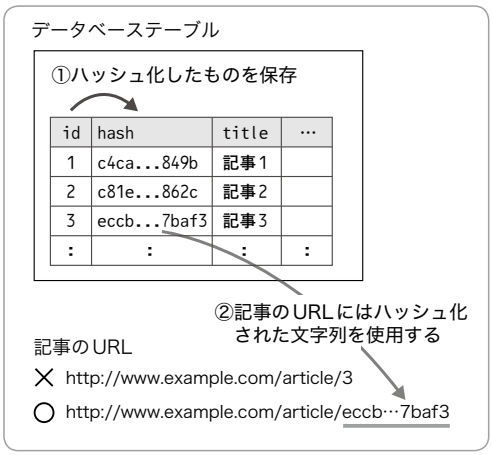
この利用法ではハッシュ関数の「どんなデータを入力しても決まった長さのハッシュ値を出力する」「別のデータに対してはほぼ別のハッシュ値を出力する」という性質を利用していますが、ハッシュ関数は「同じデータを入力すると必ず同じハッシュ値を出力する」という性質も持っています。

そのため、次のようなストーリーでURLを

▼図1　ファイルの同一性チェック



▼図2　推測しにくいURLの生成



推測されてしまう可能性があります。

①システム開発者は記事のURLを、/article/＋データベースのキーの数字をハッシュ化したものとしてシステムを設計する
②攻撃者は、記事URLを見て数字をハッシュ化したものと推測し、1から順番にカウントアップしながらいくつかのアルゴリズムでハッシュ値を計算して記事URLの文字列と比較する
③一致するものがあれば、攻撃者はすべてのURLが推測可能となる

このような攻撃には非常に有効な対策があります。それは、上記ストーリーの①で「データベースのキーの数字をハッシュ化したもの」としているところを、「データベースのキーの数字と任意の長い文字列を結合してハッシュ化したもの」とすれば良いのです。この使い方ではデータベースのキーとして振られる連番の数字から推測しにくく重複しにくい文字列を得られれば良いので、ハッシュ関数に与えるデータが「データベースのキーの数字」でも「データベースのキーの数字と任意の長い文字列を結合したもの」でもその要件を満たせるのです。

この対策をすることにより、攻撃者はシステム開発者が設定した「任意の長い文字列」を入手しない限り、URLの推測が非常に困難になります。攻撃者がこのような設計のもとでURLの推測をするためには、総当たりでデータベースのキーと任意の長い文字列を探す必要があります。

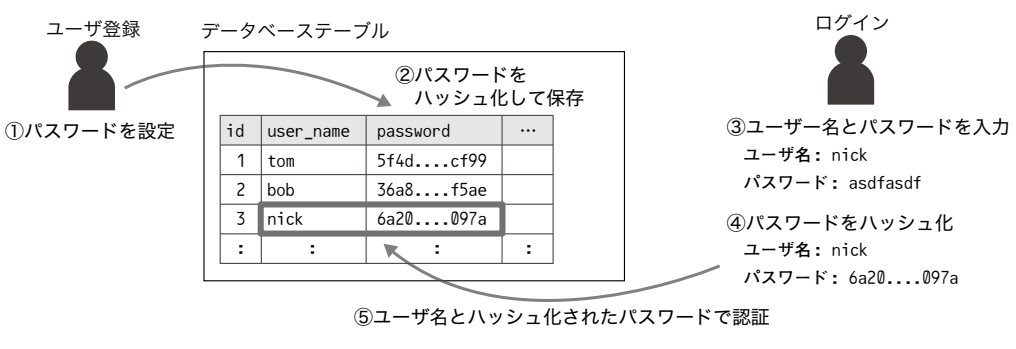
ここで使った「任意の長い文字列」のことを、「ソルト(salt)」と呼びます。この用途の場合、ソルトはシステム設定として持って、すべて同じソルトを使うのがよくあるパターンでしょう。

ログインパスワードの保存での注意点

ユーザのログインを伴うシステムでユーザのパスワードをハッシュ化してデータベースに保存するというのがこの利用法でした(**図3**)。

この利用法でも注意すべき事項はあります。それは「パスワードはユーザが指定する」ことです。その結果、システム中にセキュリティレベルの低いパスワードが含まれていることがあります。このような状況であれば、攻撃者がハッシュ化されたパスワードのリストを入手したときに、ハッシュ化されたパスワードのハッシュ

▼図3　ログインパスワードの保存



値と「世の中でよく使われているパスワード一覧」などのリストなどからハッシュ値を計算したものを比較することで容易にユーザのパスワードを知ることができます。

この攻撃に対する対策はどうしたら良いでしょうか？　ここまでこの特別企画記事を読み進めてきた方であれば想像がつくでしょうか。

答えは「ソルトを追加する」です。ソルトをパスワードに追加してからハッシュ化することで、攻撃者は「世の中でよく使われているパスワード一覧」などのリストを持っていたとしても、それに加えてソルトも探す必要があるので簡単にはパスワードを知ることはできなくなります。

では、どのようにソルトを追加するのが良いでしょうか。さきほどの「推測しにくいURL」の例ではソルトはシステム設定として持って、すべて同じソルトを使うことが多いと解説しました。これと同じ方法でもパスワードをそのままデータベースに保存するよりは良いのですが、複数のユーザが同じパスワードを使っていると、ハッシュ化されたパスワードもすべて同じになってしまいます。そのため、攻撃者がハッシュ化されたパスワードのリストを入手したときに、その中で出現回数の多いものは「世の中でよく使われているパスワード一覧」のそれぞれ上位のものである可能性が高いというヒントを与えることになってしまい、望ましくありません。

パスワードのハッシュ化について、ソルトはユーザごとに計算し、ハッシュ値と合わせてデータベースに保存する実装がよく利用されています。また、ソルトとハッシュ値だけではなく、使用しているハッシュ化アルゴリズムなどもあわせてデータベースに保存している実装もよく見かけます。

推測しにくいURLの例もそうですが、ハッシュ関数を使用するほとんどのケースではソルトを付けることになるでしょう。逆にソルトのないハッシュ化ロジックを見たら疑問に思って良いでしょう。それぐらいハッシュ関数とソルトは切っても切れない存在なのです。

■ Drupal 8での実装

ここで、PHPベースのオープンソースCMS（Contents Management System）であるDrupal 8でのパスワード保存の実装を見てみましょう。

Drupal 8のハッシュ化パスワードは次のような形の文字列です。

例）Drupal 8のハッシュ化パスワード

```
$S$Db4Dmq533CGzI1ZjHJeRLQ69o02FE2wEG./ヲ
t7io6L9DNUdxpNtb1
```

この文字列には、ハッシュアルゴリズム、繰り返し（ストレッチング）の回数[注1]、ソルト、ハッシュ値が含まれています（**表1**）。

ハッシュアルゴリズムに対する攻撃は基本的には総当たりをベースとしたものです。そのため、コンピュータの演算能力が上がればハッシュ化されたパスワードの安全性は下がっていきます。また、研究の結果、総当たりの回数を減らす方法が見つかることもあります。

そのような場合に、上記のDrupal 8の例のようにハッシュアルゴリズムを識別するための情報をハッシュ化されたパスワードの一部として保存しておくと、システムの運用中に使用するハッシュアルゴリズムを変更できます。Drupal 8では、先頭がSの場合はDrupal 7以降標準のSHA-512を使用したもの、HとPの場合は古いDrupal互換のMD5ベースの

▼表1　文字列とハッシュ値・ソルトの対応

S	D	b4Dmq533	CGzI1ZjHJeRLQ69o02FE2wEG./t7io6L9DNUdxpNtb1
ハッシュアルゴリズム	繰り返しの回数	ソルト	ハッシュ値

注1）Drupalではハッシュ値の安全性を高めるために、計算したハッシュ値の後ろにパスワードを結合して再度ハッシュ化する、という操作を複数回繰り返しています。繰り返し回数はパスワードのハッシュ化のたびに決めており、その回数をデータベースに保存するハッシュ化パスワードの中に含めています。

もの、となっています。

このようにすることで、ある時点でパスワードのハッシュ化に使用するアルゴリズムを変更した場合でも、それ以降にパスワードを設定したユーザだけ新しいアルゴリズムを使用し、既存のユーザは古いアルゴリズムを使用してログインできます。さらにユーザが古いアルゴリズムを使用してログインした際に、ユーザが入力したパスワードの正当性を古いアルゴリズムで認証したあとに、新しいアルゴリズムを使ってハッシュ化パスワードを上書き保存することで、それ以降は新しいアルゴリズムを使用できます。

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。

前回解説したとおり、ハッシュアルゴリズムの安全性の指標として「衝突困難性」「原像計算困難性」「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
- 原像計算困難性……ハッシュ値から元のデータを算出することが難しいという性質
- 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

こ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。冒頭でも触れたとおり、先日のGoogleの例は、同じSHA-1ハッシュ値を持ち内容の異なる2つのPDFを現実的な計算量で作成する方法が見つかった、ということで、上記で言うと「衝突困難性」が破られた、ということになります。

巨人の肩の上に乗る

先のDrupal 8の実装の例もそうですが、パスワードのハッシュ化などのように難しいところはたいてい先人が考えてしくみを作り、オープンな環境で議論して鍛え、堅固なものになっています。そのようなしくみはソフトウェアの実装そのものに現れる場合もありますし、APIなどに現れることもあります。

たとえば、Drupal 8ではCryptクラスに`Crypt::hashBase64`というメソッドを持っており、これは`Calculates a base-64 encoded, URL-safe sha-256 hash.`ということで、URLに利用可能なSHA-256ハッシュを計算してくれます。また、より厳密に2つのデータを比較するためのメッセージ認証符号(MAC)のアルゴリズムとして有名なHMACについても`Crypt::hmacBase64`という形で実装されています。Drupal 8を使うのであれば、独自にハッシュ化を実装するのではなく、これらのメソッドを使うのが良いでしょう。

ただし、フレームワークやライブラリが提供する機能を利用する場合は、必ず中身を理解してしかるべきものを使いましょう。前述のように`Crypt::hashBase64`と`Crypt::hmacBase64`がある場合に、自分はどちらを利用すべきか。この判断には、ハッシュとHMACについての理解、そしてそれぞれのメソッドの実装の理解が必要でしょう。正しい理解なしにフレームワークやライブラリが提供する機能を利用してしまうと、意味がないだけでなく、場合によっては本来必要だった要件を満たせなかったり、せっかく考えられたしくみを殺してしまうことにもなりかねません。

ハッシュとよく混同されるハッシュに似た概念

人間にとって意味のあるデータ、たとえば文章などをハッシュ化すると、データが不規則に見えるデータに変換されたように見えます。

一方、世の中にはハッシュ関数以外にも、与えられたデータを不規則に見えるデータに変換する操作があり、ときとしてそれはハッシュと混同されます。

ハッシュについての理解が深まったところで、ハッシュ関数とよく混同される操作を紹介します。

暗号化

もっともよくある勘違いが、暗号化とハッシュ化の混同です。前編の冒頭で紹介したAくんもこの違いをはっきりと認識していませんでした。

暗号化は、デジタル大辞泉では「文章や電子データの情報を一定の規則に従って組み替え、通信途中に第三者に利用されないようにすること。受信者は暗号化された情報に逆の手続きを施して解読する。」と説明されています。

2者間の通信を考えたときに、送信者は鍵と特定のアルゴリズムを使用して元のデータ（平文）を暗号文に変換します。この操作を暗号化と言います。暗号文を受信した受信者は、暗号化に使用したのと同じアルゴリズムと、暗号化に使用したのと同じ鍵、または、暗号化に使用された鍵とペアになる鍵を使用して平文に変換します。この操作を復号と言います。

暗号化は、復号して元のデータに戻すことを前提とした操作です。そのため、元のデータ（平文）が長くなれば、暗号化されたデータ（暗号文）は長くなります。

暗号化の方式として、暗号化に使用した鍵と同じ鍵を使って復号する「共通鍵方式」や、送信者と受信者がそれぞれペアになった対の鍵を使用して暗号化・復号を行う「公開鍵方式」があります。

公開鍵方式は、公開鍵で暗号化した暗号文は秘密鍵で復号し、秘密鍵で暗号化した暗号文は公開鍵で復号するように設計されており、SSHを始め、さまざまなアプリケーションで利用されています。

ただし、公開鍵暗号はその計算の負荷が高く、大きなデータの暗号化・復号には向きません。そのため、実際にやりとりする暗号文は共通鍵で暗号化するが、その共通鍵の配送に公開鍵方式を使うことで公開鍵暗号の秘匿性の高さと共通鍵暗号の計算負荷の低さという「良いところ取り」をした通信方法もあり、身近なところではSSLがこのような実装になっています。

なお、余談ですが、「暗号化」に対応する言葉は「復号化」ではなく、「復号」です。これは、日本語では「復号」が動詞なのに対して「暗号」は名詞なので「化」を付けて動詞化しているためです。英語では暗号化 encrypt に対して複合 decrypt と対称的な単語になっていてうらやましいですね。

難読化

難読化とは、プログラムのソースコードを人間が読みにくいように変換することです。

難読化はハッシュと同様に片方向の変換ですが、難読化されたソースコードはそのまま実行されます。そのため、難読化されていてもプログラムとして解釈可能であり、難読化前と同じ動作をする必要があります。

具体的には、変数名や関数名などを1文字のアルファベットに置換したり、改行やスペースを削除するなどの操作を行います。これらの操作によりファイルサイズが小さくなることも多いため、CSSなどでは同様の操作を「圧縮」と呼ぶこともあります。

暗号化と比較したときに、暗号化は復号するまでは使うことができませんが、難読化ではそのまま使えるところが異なります。

難読化がよく使われてるのはJavaScriptやJavaなどです。

JavaScriptは、ソースコードをそのままユーザの手元のブラウザ上で実行するので、ユーザはソースコードを入手できます。そのため、ユーザにソースコードを読まれたくない場合に難読化が使用されます。さらに、前述のとおり、難読化することでファイルサイズが小さくなることも多いため、インターネットを経由して配布されるJavaScriptにとってはメリットになります。

もう1つの例として、Javaの実行ファイルはバイナリで配布されますが、逆コンパイルすることでユーザがソースコードを入手できるため、難読化されることがあります。Javaの難読化製品としてはAndroidのProGuardが有名です。

エンコード

エンコードは、特定のアルゴリズムでデータを別の形に変換することです。変換されたデータはデコードすることで元のデータに戻すことができます。つまり、暗号化同様、両方向の変換で、元のデータが長くなれば、エンコードされたデータも長くなります。

よく使われるエンコードは、MIMEエンコード、URLエンコード、BASE64などです。どれも1回は聞いたことがあるでしょう。

エンコードは、おもに、特定の文字列しか通せない経路に任意のデータを通す目的で使います。

たとえば、URLエンコードは、URLに使えない文字をURLに入れるために使用しますし、MIMEエンコードは7ビット文字しか使えないE-mailにファイルを添付して送信するために使用します。また、多くの暗号化アルゴリズムでは文字列を暗号化すると暗号文はバイナリで出力されます。これをテキスト形式に戻すためにエンコードが使用されたりもします。

エンコードは暗号化と同様、元の形に戻すことを前提とした双方向の変換ですが、暗号化と異なり、アルゴリズムを知っていれば元の形に戻す(デコードする)ことができます。そのため、一見、不規則に見えるデータに変換されますが、エンコードされた文字列を暗号のつもりで使用してはいけません。

なお、バイナリに対して2回同じ操作をすると元の値に戻る排他的論理和(XOR)もある意味、エンコードと言えます。昔のコンピュータではフロッピーなどに格納するデータやマシン語のプログラムをXORした状態で格納し、実行時にそれをメモリにロードしたあとにXORして元の形に戻してから実行するような実装もありました。これは目的としては難読化に近く、ゲームのセーブデータや、フロッピーディスクのコピープロテクトプログラムなどにそのような処理がよく見られました。また、2017年5月に話題になったランサムウェア「WannaCry」でも、プログラムがXORでエンコードされていたようです。

まとめ

2回にわたってハッシュ関数・ハッシュ値とその周辺技術について解説しましたが、最後に内容をまとめます。

- ハッシュ関数はデータをハッシュ値に1方向に変換する関数であり、ハッシュ値は元のデータに戻すことはできない
- ハッシュ関数は「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」という性質を持っている
- ハッシュ関数を使うときは必ず入力データにソルトを付加し、衝突の可能性を常に考慮に入れる
- パスワードのハッシュ化などはできるだけ既存のしくみを使うのが望ましい
- ハッシュと似た概念として暗号化、難読化、エンコードがあるが、それらはすべて別の概念である

エピローグ

前回の冒頭で上司のBさんからハッシュを使った実装を依頼された新人Aくん、その後どんな設計にしたのでしょうか。

上司Bさん)ログインパスワードの保存の件はどうだ？　順調に進んでいるか？
新人Aくん)はい。いくつかのオープンソース

ソフトウェアの実装を調査し、SHA-512でハッシュ化されたパスワードをソルトと一緒にデータベースに保存することにしました。

上司Bさん）うむ。それで良いだろう。

新人Aくん）それから、1つご提案があります。今回はSHA-512を使用しますが今後SHA-512に脆弱性が発見されたときのために、データベースに保存するハッシュ値にアルゴリズムを指定する値も含めようと思うのですがいかがでしょうか。

上司Bさん）良い観点だ。それで進めてくれ。

新人Aくん）承知しました！

――新人Aくん、だいぶ勉強をしたようですね。今後も今回のことを忘れずに疑問に思ったことは精力的に調べて成長していってほしいですね！SD

Column 「プログラム言語のハッシュへの攻撃」

本記事の前編のコラムでは、プログラム言語で連想配列のことをハッシュと呼ぶ言語がある理由について、内部的な実装にハッシュ関数を利用しているためである、という話を紹介しました。ハッシュ関数を利用した連想配列の実装では、連想配列のキーをハッシュ化しそのハッシュ値のアドレスに値を格納します。ハッシュ値は衝突する可能性がありますが、衝突した場合はハッシュ値のアドレスにリスト形式で複数の値を格納します。この実装では、ハッシュ値が衝突しない限りでは1回のハッシュ値の計算で指定されたキーに対応する値を見つけることができます。ハッシュ値が衝突した場合は1回のハッシュ値の計算＋ハッシュ値が衝突した数だけループをたどって指定されたキーに対応した値を見つけることになります。

さて、このような言語が多く存在する中で、2011年の年末にhashdosというセキュリティ上の問題が発表されました。これは、連想配列の実装に問題を抱えた言語でWebアプリケーションなどを実装した際に、容易にDoS攻撃（計算量を大きく増加させ攻撃対象のサーバのパフォーマンス低下を引き起こす攻撃）ができる、という問題でした。

たとえばPHPでは、HTTPのGETメソッドやPOSTメソッドでアクセスされた場合に、そのパラメータが連想配列に格納された状態でPHPプログラムが実行されます。つまり、連想配列のキーを外部から自由に指定できることになります。ここで大量のパラメータを、キーをハッシュ化したものが同じになるような名前で与えることにより、配列に対するアクセスのパフォーマンスを著しく落とします。

hashdosは、その発表時点でPHPに限らずJavaなど多くの言語で抱える問題でした。この対応は各言語で行われ、ある言語ではプロセスごとに異なるソルトを使うようにして意図的な衝突が起こらないようにし、また別の言語ではHTTPパラメータの数を制限することでDoSとしての効果を下げるという対症療法的な対応をしました。

このように、ハッシュ関数の最大の特徴である「衝突が起こり得る」という特性には常に注意を払う必要があります。

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(後編)

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 http//blogs.technet/microsoft.com/osamut
日本マイクロソフト㈱

本誌2017年6月号では、ファイルサーバとしての機能の解説と構築する際に使えるWindows Server 2016の最新機能にも触れました。さて、実際に試してみたいと思った方が次に考えるのは、その環境をどうやって用意するかではないでしょうか。今回は手軽にWindows Server 2016のファイルサーバ環境を作るべく、クラウド(Microsoft Azure)上に検証用のファイルサーバを構築する方法を解説します。また、クラウドならではのコツなど、知っておくべき、もしくは気をつけるべきポイントにも触れていきます。

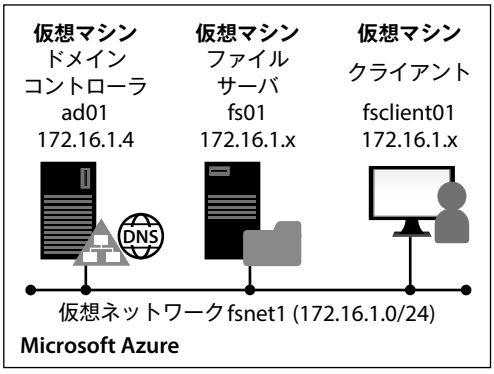
Azure上に検証環境を構築

まずは、図1のような構成を目指します。Azure上に1つのネットワークを作り、そこに認証基盤としてのActive Directoryドメインコントローラと機能検証用のファイルサーバ、そして接続テスト用のクライアントとして3台の仮想マシンを配置します。

①仮想ネットワークfsnet1の作成

テスト環境なので、見た目にもわかりやすいIPアドレスを設定しておくとよいでしょう。また、図2の設定画面に出てくる「リソースグループ」などの用語はAzure特有のため詳細な説明は割愛します。Azureにはネットワークや仮想マシンを束ねて管理するためのグループ名のようなものがあると考えてください。

▼図1 Microsoft Azure上で作るファイルサーバの簡易検証環境(IPサブネットや仮想マシンの名前など、本記事に書かれた内容はあくまでも一例)



②Windows Server 2016 Datacenter テンプレートからad01、fs01、fsclient01という3つの仮想マシンを作成し、fsnet1に配置

ad01を最初に作成しましょう(図3)。そうす

▼図2 仮想ネットワークfsnet1の作成画面

▼図3　Windows Server 仮想マシンの作成画面

ることで、ad01にサブネット内の4番めの番号172.16.1.4というIPアドレスが割り当てられているはずです。ドメインコントローラマシンのIPアドレスを確実に把握しておくことで、このあとのActive Directoryのセットアップ、ファイルサーバやクライアントのドメイン参加などで重要になるDNSサーバのIPアドレス指定がしやすくなります。

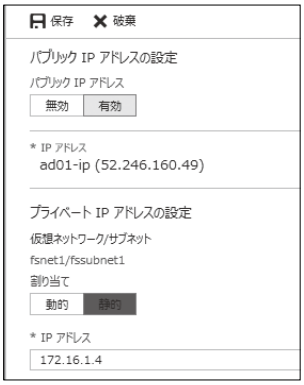
③ad01のネットワークインターフェースプロパティでプライベートIPアドレスを[静的]に変更

Azure上の仮想マシンは、使わないときにシャットダウンをしておくことで余計な課金を止めることができます。ただし、この際IPアドレスも解放し、何もしなければ次の起動時に違うIPアドレスになることがあります。そこで、**図4**の設定を用いて、ドメインコントローラad01のIPアドレスが常に同じになるよう固定します。

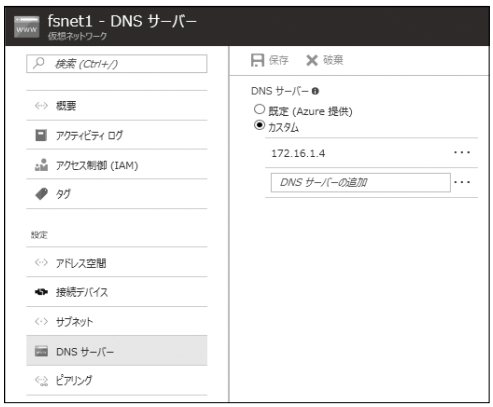
④fsnet1のプロパティでDNSサーバを172.16.1.4(カスタム)に設定

AzureはデフォルトでDNSサービスが動いており、とくにDNSサーバの指定をしなくても仮

▼図4　仮想マシンに自動的に割り当てられたIPアドレスを静的な割り当てに変更(仮想マシンのプロパティはDHCPになっていても、常に同じIPが割り振られる)



▼図5　DNSサーバのIPアドレスを172.16.1.4に固定しているところ

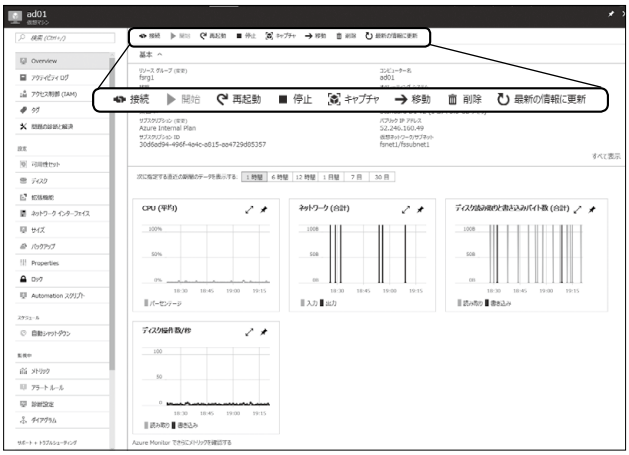


想ネットワーク内の仮想マシンは名前解決ができるようになっています。しかし、Active Directory環境のように独自でDNSを立てることも多くあるでしょう。このような場合、ネットワークに配置された仮想マシンが同じDNSサーバを確認できるようにしなければなりません。**図5**では、仮想マシンそれぞれにDNSサーバの設定をしなくて済むよう、仮想ネットワークのプロパティ設定でDNSサーバのIPアドレスを固定化しています。

⑤④のDNS設定を反映させるため、仮想マシンad01を再起動

この作業によって、ad01という仮想マシンの

▼図6　仮想マシンのプロパティ画面(左のプロパティー覧に加えて、上にも接続や再起動のメニューあり)



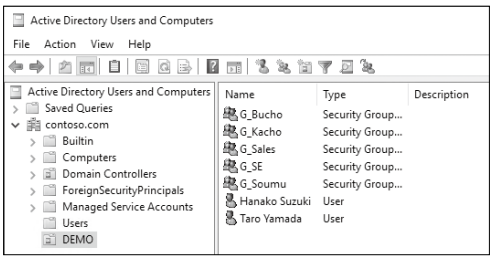
DNSサーバアドレスは172.16.1.4となり、自分自身に向けられることになります。

⑥ad01に接続し、Active Directoryドメインコントローラへと昇格(今回はcontoso.comを使用)

仮想マシンのプロパティ画面にある接続をクリックするとRDPファイルがダウンロードされるため、それを使って仮想マシンにリモートデスクトップで接続します(**図6**)。Active Directoryドメインコントローラへの昇格手順も既知のため細かくは触れませんが、Azureだからといって特別なことはありません。いつもどおりの手順で実施してください。

ドメインコントローラの準備ができたら、検証用のユーザやグループを作成しておくとよいでしょう(**図7**)。画面は英語ですが、これは後述する方法でOSを日本語化できます。

▼図7　DEMOというOUを作成し、ユーザやグループを登録したところ



⑦DNS設定を反映させるためにfs01を再起動

この再起動により、ファイルサーバマシンはActive Directoryとともに動作するDNSサーバを見にいくことになります。ちなみに再起動の作業は意図的に仮想マシンに接続してから実施する必要はなく、Azureの管理ポータルからでもできます。

⑧fs01にリモートデスクトップ接続し、ドメインcontoso.comに参加(再起動)

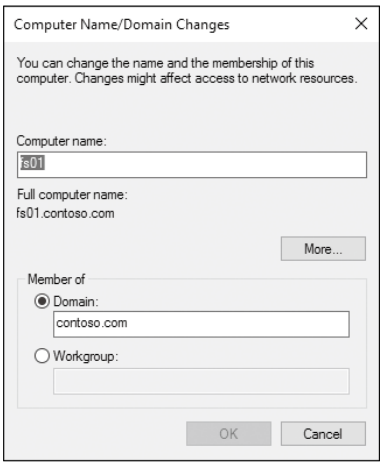
この段階ではすでにDNSサーバのIPアドレスがドメインコントローラに向いているので、ドメインへの参加はスムーズです(**図8**)。

⑨再起動後、fs01にドメインの管理者としてログオンし、fs01を日本語化

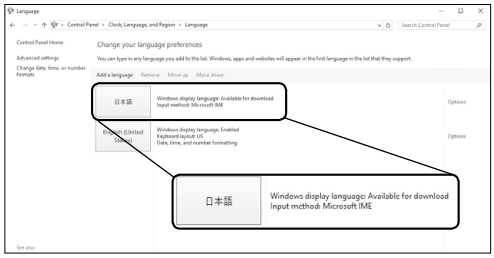
Azureの仮想マシンの、日本語化の手順は次のようになります(**図9**)。日本語化したテンプレートを用意しておき、そこから仮想マシンを作ることもできます[注1]。

注1) 管理ディスク(Managed Disks)の"イメージ"リソースを使用し、仮想マシンを複数台展開する
URL https://blogs.technet.microsoft.com/jpaztech/2017/05/10/deployvmsfrommanagedimage/

▼図8　見慣れたドメイン参加画面。このあと、ユーザ名とパスワード入力画面が表示される



▼図9　日本語言語パックの追加



❶[Control Panel]の[Add a language]
❷[Add a language]で[Japanese(日本語)]を[Add]
❸[Move up]で日本語の優先順位を上に
❹[Options]をクリック――言語パックを自動で探してくれる
❺[Download and install language pack]をクリックして言語パックをインストール
❻[Change date, time, or number formats]をクリックして、ロケールやようこそ画面などを日本語に変更

⑩fsclient01に対しても⑦～⑨を実施

これで、Azure上に認証基盤とファイルサーバ、クライアント環境ができあがります。慣れていれば数十分、いろいろと確認しながらでも1～2時間もあればというところでしょう。ちなみに、Office 365 ProPlusを使える権利をお持ちの場合は「Windows Server Remote Desktop Session Host with Microsoft Office 365 ProPlus」という仮想マシンのテンプレートをAzure Marketplaceで検索し、そこから仮想マシンfsclient01を作成するとよいでしょう。このテンプレートはVDIなどの用途のために用意されていて、Officeが事前にインストールされているため、ファイルサーバ検証にOfficeが必要な場合にも別途用意する手間を省けます。

ファイルサーバとしての環境の準備

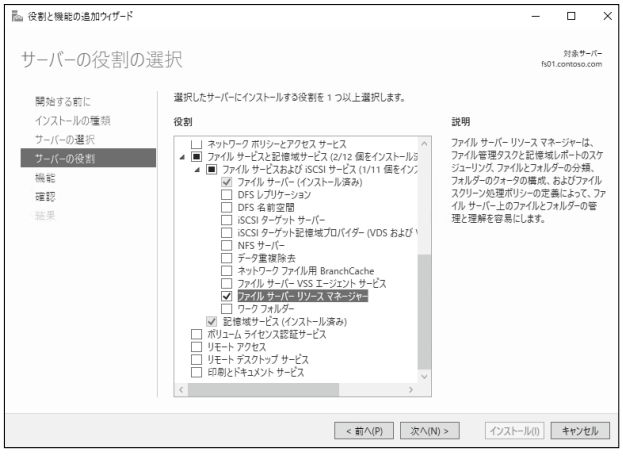
前号の復習も兼ね、ファイルサーバの機能とそれを試すのに必要となる作業を簡単に説明します。**図10**の画面は、Windows Serverが起動する際に立ち上がってくるサーバマネージャというツールに統合されています。またエクスプローラでフォルダを共有した場合でもサーバマネージャ側にも表示してくれるので、どのフォルダを共有したか覚えていないという場合でも頼りになる存在です。

そして、ディスククォータやスクリーニング、レポート、ファイル分類機能などを利用する場合には、Windows Serverの役割と機能の追加からファイルサーバリソースマネージャを追加します(**図11**)。

追加が終わると、各種設定が可能になるので、ファイルサーバ側でクォータなどの設定を行い、それらの設定が正しく機能しているかを

▼図10　サーバマネージャに統合された共有フォルダ管理画面

▼図11　ファイルサーバリソースマネージャを追加しているところ



▼図12　サーバマネージャのツールの一覧にファイルサーバリソースマネージャが追加されている

fsclient01というクライアントから確認していくとよいでしょう(**図12**)。

今回用意したAzure上の各仮想マシンは、この状態でインターネットにも接続できる状態なので、検証用のツールやファイルのダウンロードなどもできます。また、リモートデスクトップで接続していれば、ローカルPCにあるファイルをドラッグ&ドロップで仮想マシンのデスクトップなどにコピーすることも可能になっています。

さらに、Windows Server 2016ベースの環境には、標準でマルウェア対策ソフトウェアのWindows Defenderが組み込まれており、Windows Updateと連動しながら最新の定義ファイルの更新も行われ、スキャンも実行されますので、自社内で利用しているマルウェア対策ツールを入れて検証したい場合を除き、このまま進めてよいでしょう。

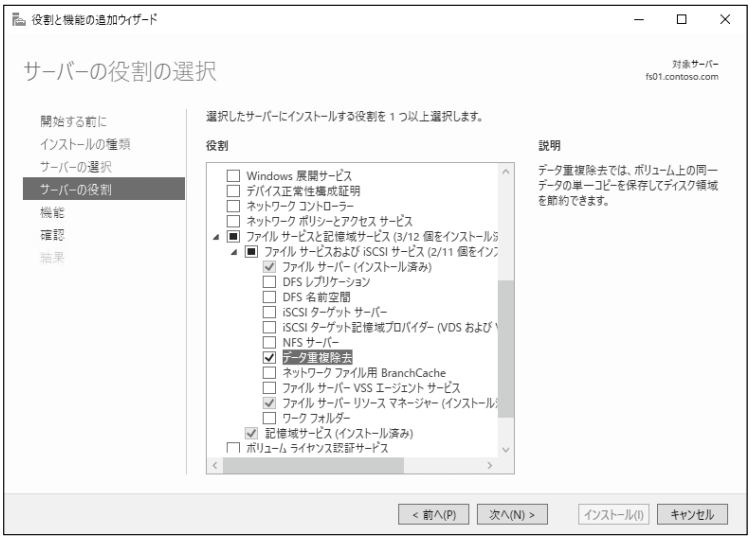
ストレージの重複除去機能を試す

Windows Server 2016のストレージ機能の中には、重複除去の機能が含まれています。ただし、こちらも最初は有効になっていないため、役割と機能の追加という作業を行います。また、記事執筆時点では、更新プログラム[注2]を適用する必要がありますので注意ください(**図13**)。

機能の追加と更新プログラムの適用が完了したら、あとは重複除去の設定を行うだけと言えます。ただし、実際のファイルサーバはデータ用ディスクの追加なども行われているでしょうから、ファイルサーバを模擬すべくAzure上の仮想マシンにもデータ用のディスクを追加してみましょう。

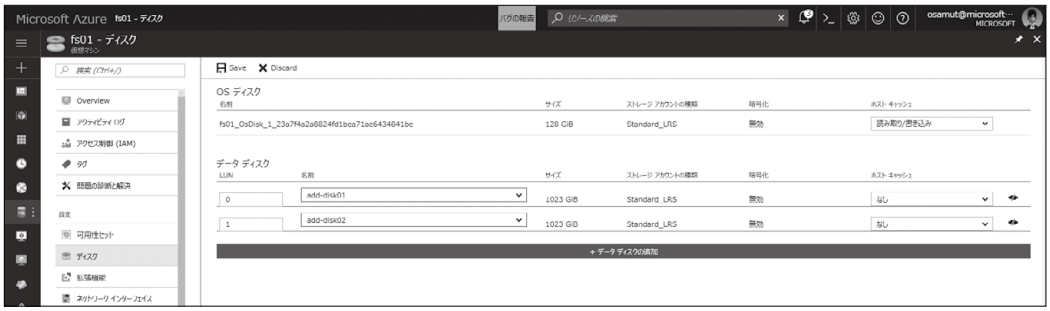
図14のように、仮想マシンのプロパティにあるディスクメニューで「データディスクの追加」という作業を行います。Azureの新しいディスク管理機

▼図13　データ重複除去の機能追加画面



注2) URL https://support.microsoft.com/ja-jp/help/4013429/windows-10-update- kb4013429

▼図14　仮想マシンへのディスク追加画面



構「管理ディスク」では、IOPSが500でスループットの上限が60MB/秒に制限されたStandardと、ディスクあたり最大でIOPS 5000、スループットが200MB/秒まで高速にアクセス可能なPremiumがあります(**図15**)。

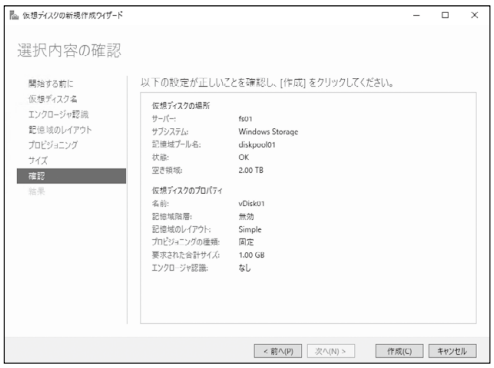
▼図15　記事執筆時点での Premium 管理ディスクのパフォーマンスとコスト(https://azure.microsoft.com/ja-jp/ricing/details/managed-disks/)

	P10	P20	P30
ディスク サイズ	128 GB	512 GB	1,024 GB
月額	¥2,312.34	¥8,588.40	¥15,854.88
ディスクあたりの IOP	500	2,300	5,000
ディスクあたりのスループット	100 MB/秒	150 MB/秒	200 MB/秒

ファイルサーバかどうかに限らず、ディスク(ストレージ)の選択は、スピード、容量、コストのバランスがとても重要になります。もしIOPS値が1つのディスクでは足りない場合は、複数のディスクを追加し、それをWindows Serverのストレージ機能などを使って分散処理させることで高速化を図れます。

さて、Azureのポータルからディスクの追加という作業をすると、サーバマネージャの記憶域プールという画面に物理ディスクとして追加されています。今回は、このディスクをWindows Serverの機能でプール化し、そこから仮想ディスクvDisk01の作成を行います(**図16**)。

▼図16　ディスクをプール化したあと、仮想ディスクとして切り出している画面(「記憶域のレイアウトSimple」とはストライピングのこと)



仮想ディスクとして切り出したあとは、論理ボリュームとしての設定ウィザードが表示されます。このウィザードでは、エクスプローラから見えるドライブ番号(Fドライブとか Hドライブ、ドライブレターとも言う)の設定や、重複除去の設定なども可能になっています。

▼図17　ウィザードでデータ重複除去の設定をしているところ

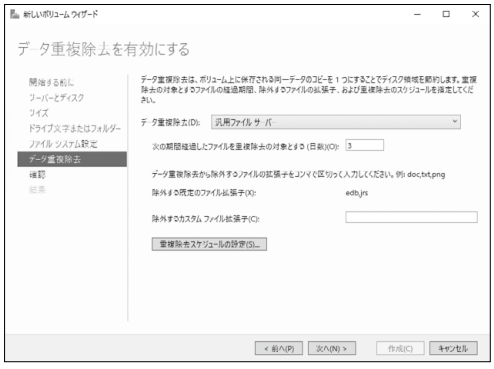


図17はファイルサーバとして重複除去を有効にしているところですが、この設定だけで定期的に重複除去が行われ、ファイルサーバの負荷が高い場合には重複除去の処理を一時停止します。ただ、デフォルトの設定に任せるのではな

▼図18　重複除去(deduplication)用のPowerShellコマンドの一覧を表示したところ

```
Administrator: Windows PowerShell
PS C:\Users\osamut.CONTOSO>
PS C:\Users\osamut.CONTOSO>
PS C:\Users\osamut.CONTOSO> get-command *dedup*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Function        Disable-DedupVolume                                2.0.0.0    Deduplication
Function        Enable-DedupVolume                                 2.0.0.0    Deduplication
Function        Expand-DedupFile                                   2.0.0.0    Deduplication
Function        Get-DedupJob                                       2.0.0.0    Deduplication
Function        Get-DedupMetadata                                  2.0.0.0    Deduplication
Function        Get-DedupProperties                                2.0.0.0    Storage
Function        Get-DedupSchedule                                  2.0.0.0    Deduplication
Function        Get-DedupStatus                                    2.0.0.0    Deduplication
Function        Get-DedupVolume                                    2.0.0.0    Deduplication
Function        Measure-DedupFileMetadata                          2.0.0.0    Deduplication
Function        New-DedupSchedule                                  2.0.0.0    Deduplication
Function        Remove-DedupSchedule                               2.0.0.0    Deduplication
Function        Set-DedupSchedule                                  2.0.0.0    Deduplication
Function        Set-DedupVolume                                    2.0.0.0    Deduplication
Function        Start-DedupJob                                     2.0.0.0    Deduplication
Function        Stop-DedupJob                                      2.0.0.0    Deduplication
Function        Update-DedupStatus                                 2.0.0.0    Deduplication
```

くご自身で制御したい場合には、重複除去スケジュールの設定を行うとよいでしょう。また、Windows Serverには重複除去の処理を制御するPowerShellコマンドも用意されているため、処理中の重複除去ジョブの列挙や強制的な実施などもできます。検証環境では重複除去の即時実行Start-Dedupjobコマンドやジョブの状態チェックGet-Dedupjobをよく使うことになるので覚えておくとよいでしょう(**図18**)。

このように、重複除去は高額な専用ストレージだけのものではなくなり、クラウド上でも利用できるようになりました。さまざまな部門、さまざまな仕事をしている社員が利用するファイルサーバなので、運用していくと意図せずに同じファイルが複数保存されてしまうこともあります。利用者に余計な作業を強いることなく、効果的にファイルサーバ用のストレージを活用するために、重複除去は欠かせない機能の1つと言えます[注3]。

クラウド上でファイルサーバの可用性を高める

これまで、高可用性ファイルサーバを構築するには、複数台のサーバと共有ストレージを用意し、Windows Serverの標準機能であるフェールオーバークラスタを使っていました。この構成はサーバOSが最新に変わった今でもできますが、一般的にパブリッククラウド上ではこのような構成をとりません。なぜなら、オンプレミスで利用される共有ストレージ装置はそれ自身が止まらずにサービスを提供する工夫がなされているのに比べ、クラウド上では仮想マシンがその代わりを担うことになり、共有ストレージとしての仮想マシンそのものが単一障害点になってしまう可能性があるからです。そのため、Azure上のシステムによっては、サイオステクノロジー社のDataKeeperなどを利用してストレージレプリケーションを実現し、可用性を高める構成などがとられています。

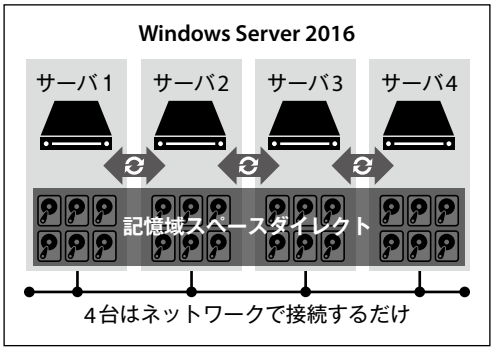
Windows Serverの標準機能で実現する方法はないものか、と思われた方もいるでしょう。その答えとして、ようやく登場したのがWindows Server 2016の記憶域スペースダイレクト(Storage Spaces Direct = S2D)という技術です[注4]。

図19を見てわかるとおり、4台のサーバがネットワークで接続されているだけです。しかし、それぞれのサーバに搭載されたローカルディスクを記憶域スペースダイレクトという技術で1つに束ね、サーバ4台はすべてアクティブに動くことができます。また、この技術は最低2台から構成可能になっているので、たった2台のWindows Server 2016で可用性の高いファイル

注3) 現時点では、Windows Server 2016一押しの新しいファイルシステムReFSは重複除去が効かないため、ファイルサーバのデータ用ボリュームはNTFSを選択する場面も多く出てきます。もちろん、ReFSによる重複除去も将来的には実現の方向で話が進められています。

注4) 記憶域スペースダイレクトの詳細はこちら
URL https://docs.microsoft.com/ja-jp/windows-server/storage/storage-spaces/storage-spaces-direct-overview

▼図19　記憶域スペースダイレクト



サーバを構築可能になっています。

ここから、2台のWindows Server 2016仮想マシンを使った高可用性ファイルサーバの構築手順を見てみましょう。認証基盤などはすでに用意できているので、先ほどの3台の検証環境に追加する形で、**図20**を目指します。

手順は次のとおりです。

①Windows Server 2016 の仮想マシンを2台作成し、fsnet1 へ配置
②2台のマシンそれぞれを Active Directory に参加
③2台のマシンそれぞれにデータ用のディスクを2つ追加(手順確認のため)
④2台でフェールオーバークラスタを構築
⑤フェールオーバークラスタのクォーラムとしてクラウドウィットネスを設定
⑥記憶域スペースダイレクトを有効化するコマンドを実行
⑦記憶域スペースダイレクトを使って2台ずつ追加したディスクをプール化
⑧ディスクプールから仮想ディスクを作成
⑨ファイルサーバとして利用するため、スケールアウトファイルサーバの役割を追加
⑩共有フォルダ作成

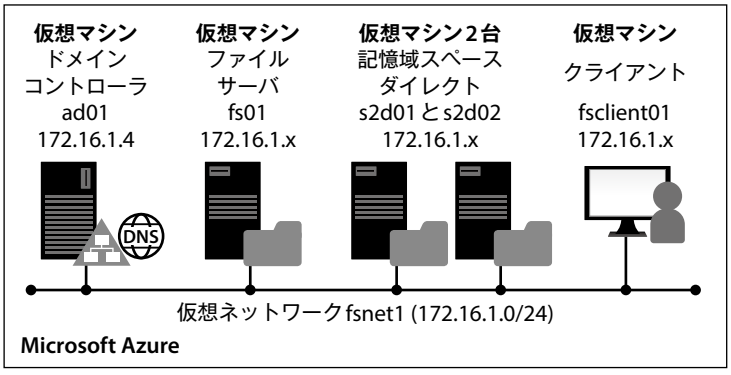
ポイントをいくつか書いておきましょう。まずは⑤のクォーラムの設定ですが、Windows Server 2016では、ウィットネスにAzureストレージを選択できるようになっています。

⑥の手順では、記憶域スペースダイレクトを有効化するために Enable-ClusterS2D コマンドを利用します。このコマンドで、各ノードにバラバラに接続されたディスクがS2Dというしくみに組み込まれます。たとえば、今回の環境では1台のマシンにデータ用のディスクを2つ追加しているため、本来サーバマネージャの管理画面には2つのディスクが見えているはずです。しかし、このコマンドを実行することで、**図21**のように合計4つのディスクが1台のサーバのローカルに存在しているように見えてきます。

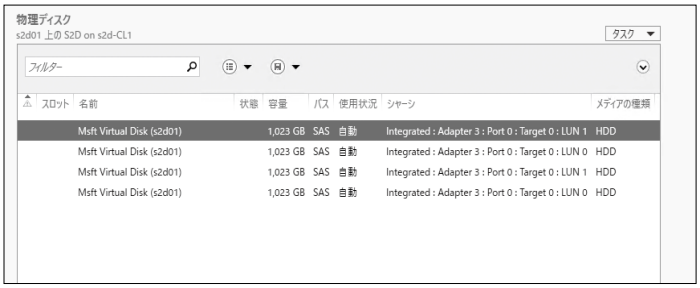
2台にまたがっているディスクが1台のマシンに接続されているように見えるので、次はそれらのディスクをプール化し、さらに仮想ディスクを切り出すことでS2Dの環境構築は完成です。ただし、今回はそれをファイルサーバとして利用したいので、スケールアウトファイルサーバという役割の追加作業を行います。これで、2台のマシン(+クラウドウィットネス)による可用性の高い共有フォルダができあがります。クライアントからも共有フォルダが見えているはずです。

さて、今回の検証環境ではクライアントもクラウド上に作成しているため、ファ

▼図20　Azure上の検証環境にS2Dマシンを2台追加したところ



▼図21　記憶域スペースダイレクトを有効にするコマンドを実行したあとのディスク管理画面

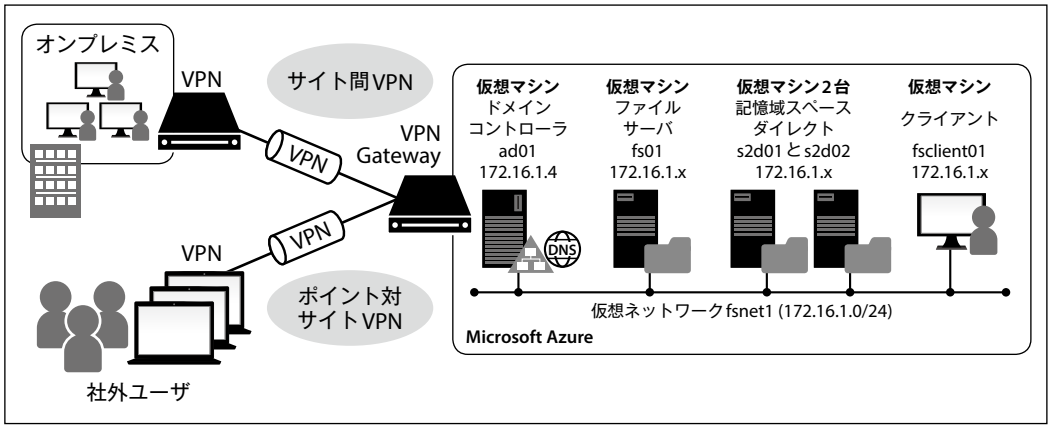


イルサーバも利用者もクラウド上にいることになっています。最近流行り始めたクラウド上のVDI環境であれば、ほぼ同じ構成と言えるでしょう。しかし、利用者のクライアントが社内に設置されたPCであった場合にはどうするか、社外に持ち出すモバイルPCだったらどうするかも検証したいところです。

まずは、比較的容易に試すことができるパターンから紹介しておきましょう。AzureにはVPN Gatewayという機能があり、それを使うことで、**図22**のように企業のネットワークとAzureの仮想ネットワークをサイト間VPN(Site-to-Site VPN)で接続できます。企業側にVPN装置がない場合はWindows Server 2016のリモートアクセス機能を使ってVPNルータ代わりにすることもできます。こうすることで、社内のユーザはファイルサーバがAzure上にあることを意識することなく<\\ファイルサーバ名\共有名>でアクセス可能になります。また、Azure VPN Gatewayにはポイント対サイトVPNという機能も持っていて、PCから直接Azure上の仮想ネットワークにVPN接続できるので、モバイルユーザを想定した検証もできます。ただし、このポイント対サイトVPNには接続の上限値があるので気をつけましょう。

それから、ファイルサーバをクラウドに配置する場合、企業とAzureとを閉域網接続し、通信の安定化を図るパターンが増えてきています。全社員が利用するファイルサーバの利便性を落とすことは社員の生産性に影響を及ぼすからです。AzureにはExpressRouteという閉域網接続サービスがあり、社内とAzureとの通信を安定した品質で構築できます。また、クラウド上のファイルサーバと社内のPCとの距離が空くことによって著しくパフォーマンスが低下する場合は、Windows Server 2016のブランチキャッシュという機能を利用して、社内にファイルサーバ用のキャッシュを置くこともできます。一度アクセスしたことがあるファイルはキャッシュにも保管され、次からはキャッシュサーバとの

▼図22　Microsoft Azure上のファイルサーバへVPNを経由して接続するパターン



やりとりで済みますし、あくまでもキャッシュという位置付けのため、クラウド上のファイルサーバが常に正であるというルールを徹底できます。

クラウド上のファイルサーバのバックアップ

最後にクラウド上でのバックアップをどうするかは考えておく必要があります。クラウドストレージのスナップショット機能でとりあえずデータがなくならないようにしているという強引なものから、仮想マシンにバックアップエージェントを入れ、オンプレミスと同じようにバックアップをとるパターンもあるでしょう。Azureの場合は、Azureバックアップサービスを使うのがシンプルです。もともとAzureバックアップサービスはオンプレミスのデータを保護する目的などで使われてきましたが、最近Azure仮想マシンのサービスとうまく統合されました。

図23のように、Azureの仮想マシンのプロパティにはバックアップという項目が用意されています。バックアップサービス側でバックアップ用のポリシー(どのようなタイミングでバックアップをとるかなど)の定義を事前に作っておけば、仮想マシンの管理者が、仮想マシンのプロパティ設定画面からバックアップを有効化できます。また、仮想マシンの復元やファイルレベルの回復作業というメニューも仮想マシンのプロパティに用意されています。

▼図23　バックアップツールと統合されたAzure仮想マシンのプロパティ

進化の激しいクラウドで起こるファイルサーバ周りの進化の可能性

クラウド上の仮想マシンを使ってファイルサーバを構築するところまで解説してきました。しかし、パブリッククラウドでは、SaaSやPaaSが台頭してきており、仮想マシン依存から脱却できる可能性が出てきています。これはファイルサーバも例外ではなく、Azure上で認証サービスを提供するAzure Active Directoryにはドメインサービスが提供され、ドメインコントローラを仮想マシンで構築する必要がなくなりつつあります。また、Azureのストレージサービスは、BloB StorageやTable Storageなどのサービスとともにファイル共有サービスを提供しています。記事執筆時点で、このサービスは社内のドメインに参加できないため、利用用途は限られますが、今後はドメインに参加可能なサービスになる可能性もあります。

このように、パブリッククラウドは高速に進化し、さまざまなITの要素をサービス化し始めているため、いずれはドメインコントローラもファイルサーバも当たり前のようにサービスとして利用する時代が来るかもしれません。また、クラウド上のサービスはSLAとともに高可用性を意識した構成になっている場合が多いので、記憶域スペースダイレクトなどの技術を使って仮想マシンの可用性を高めるという必要すらなくなるかもしれません。柔軟な容量の増減、APIやコマンドを使った自動化、バックアップサービスとの連携など、今後ファイルサーバの見直しをするエンジニアは、クラウドにも注目しておくとよいでしょう。SD

短期集中連載

Jamesのセキュリティレッスン

パケットキャプチャWiresharkの新展開

第10回 Jamesの挑戦状! Wireshark実践問題

Writer 吉田 英二(Eiji James Yoshida)
合同会社セキュリティ・プロフェッショナルズ・ネットワーク(http://www.sec-pro.net/)

はじめに

みなさん、こんにちは! Eiji James Yoshidaです。毎回洋楽のことばかり書いているけど邦楽は聴かないのかよ、と思われているかもしれませんが、最近は一風堂(IPPU-DO)の曲を聴いていたりします。もちろんラーメン屋ではなく、土屋昌巳率いる80年代のロックバンドで、歌謡曲と80年代英国New Waveの融合っぽい独特な雰囲気の曲が多くて好きだったりします。「すみれ September Love」は一風堂で有名な曲なので聴いたことのある人もいると思いますが、SHAZNAの曲と間違えている人も多いんですよね。筆者はSHAZNAのカバー版より一風堂のオリジナルをお勧めします。

さて、今回は本連載の記念すべき第10回なので、Jamesの挑戦状としてWiresharkについての問題を出します。ぜひ過去の連載も参考にしながら、全問正解目指して頑張ってください!

環境説明

本稿を書く際に使用した環境はWindows 10でWireshark 2.2.6です。

・Wireshark
https://www.wireshark.org/download.html

Wiresharkのインストール方法はお任せしますが、とくにこだわりがない場合はデフォルトでインストールしてください。また、筆者のブログから下記のファイルをダウンロードしてください。

・Software Design 短期集中連載「Jamesのセキュリティレッスン」用キャプチャファイル
http://d.hatena.ne.jp/EijiYoshida/20140907/1410071296
・sd1707.zip
(展開用パスワード:88224646ba)

あとは何かBGMでも流しましょう。筆者は一風堂のベストアルバム「ESSENCE:THE BEST OF IPPU-DO」をBGMにして本稿を書いています。とくに「ふたりのシーズン」や「イミテーション・チャチャ」、「すみれ September Love」はお気に入りでよく聴いていたりします。

いざ、挑戦!

Wiresharkのインストールやダウンロードしたファイルの展開も終わり、問題を解く環境が整ったと思うので、さっそく挑戦しましょう!

問題1

「どのようなキャプチャフィルタがインターフェースに設定されていたのか」と「どのインターフェースでキャプチャされたパケットなのか」という情報が記録されるファイル形式はどれでしょうか。選択肢の中から正しいものをすべて選んでください(解答はP.103)。

① ppapファイル形式
② pcap(libpcap)ファイル形式
③ pcap-ngファイル形式

問題2

送信元IPアドレスまたは送信先IPアドレスのどちらか一方が192.0.2.1ではないパケットだけを表示するディスプレイフィルタはどれでしょうか。選択肢の中から正しいものをすべて選んでください(解答はP.105)。

①ip.addr == 192.0.2.1
②ip.addr != 192.0.2.1
③!ip.addr == 192.0.2.1

問題3

SYNパケットだけを表示するディスプレイフィルタはどれでしょうか。選択肢の中から正しいものをすべて選んでください(解答はP.105)。

①syn
②tcp.flags == 2
③tcp.flags == syn
④tcp.flags.syn == 1

問題4

2017年5月5日1:23:45の間にキャプチャされたパケットだけを表示するディスプレイフィルタはどれでしょうか。選択肢の中から正しいものをすべて選んでください(解答はP.106)。

①frame.time == "2017-05-05 01:23:45"
②frame.time >= "2017-05-05 01:23:45"
  && frame.time < "2017-05-05 01:23:46"
③"2017-05-05 01:23:45" <= frame.time
  < "2017-05-05 01:23:46"

問題5

「sd1707.pcapng」に保存されている、「james」という文字列を含むパケットの番号(No.)をすべて答えてください(解答はP.106)。

問題6

「sd1707_portscan.pcapng」には、攻撃者(192.0.2.100)が標的(192.0.2.1)をSYNポートスキャンしたときに送受信されたパケットが保存されています。ポートスキャンで開いていると判断されたポート番号をすべて答えてください(解答はP.107)。

問題7

「sd1707_ftppassauth.pcapng」にはFTPサービスのパスワード認証で送受信されたパケットが保存されています。このキャプチャファイルを参考にしながら、「sd1707_ftpcrack.pcapng」に保存されている、FTPサービスに対するパスワード推測攻撃で認証に成功したユーザ名とそのパスワードをすべて答えてください(解答はP.108)。

問題8

「sd1707_intrusion.pcapng」には、攻撃者(192.0.2.100)が標的(192.0.2.2)のバックドア(4444/tcp)に接続しているときに送受信されたパケットが保存されています。攻撃者がバックドアに接続して実行したWindowsコマンドをすべて答えてください(解答はP.110)。

答え合わせ

それでは答え合わせをしていきましょう。

問題1の解答

答えは③です。

①のppapだとPen Pineapple Apple Penになってしまうのと、そもそもppapというキャ

プチャファイルのファイル形式が現時点では存在しません。

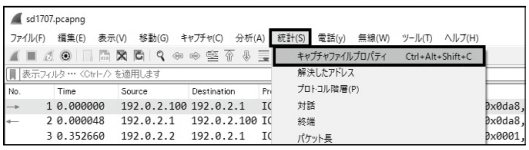
残る②と③については、pcap(libpcap)ファイル形式の「sd1707.pcap」とpcap-ngファイル形式の「sd1707.pcapng」のキャプチャファイルプロパティや、パケット詳細ペインのFrameを展開すれば答えがわかります。

まずはWiresharkで「sd1707.pcapng」を開きましょう。[統計]メニュー(**図1**)にある[キャプチャファイルプロパティ]をクリックして[キャプチャファイルプロパティ]ウィンドウ(**図2**)を表示します。[インターフェース]にある[キャプチャフィルタ]の下を見ると「icmp」とありますが、これがパケットをキャプチャする際にインターフェースに設定したキャプチャフィルタの内容になります。

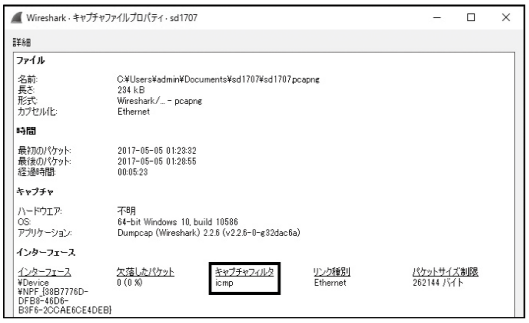
続いて、[キャプチャファイルプロパティ]ウィンドウを[閉じる]ボタンをクリックして閉じたら、Wiresharkの上側に表示されている[パケット一覧]ペインでNo.1のパケットをクリックします。Wiresharkの中央に表示されている[パケット詳細]ペインで一番上にある[Frame 1:(省略)]を展開すると、**図3**のように「Interface id: 0 (\Device\NPF_{38B7776D-DFB8-46D6-B3F6-2CCAE6CE4DEB})」という情報が表示されますが、この「\Device\NPF_{}」で囲まれた値「38B7776D-DFB8-46D6-B3F6-2CCAE6CE4DEB」が、No.1のパケットをキャプチャしたインターフェースのGUIDになります。

インターフェースのGUIDを調べるには、パケットをキャプチャしたマシンのコマンドプロンプトで**図4**のコマンドを実行します。この図は筆者の環境で実行した結果になりますが、パケットをキャプチャしたインターフェースのGUIDとSettingIDの値が同じことから、「Intel(R) PRO/1000 MT Network Connection」でNo.1のパケットをキャプチャしたことがわかります。このように③のpcap-ngファ

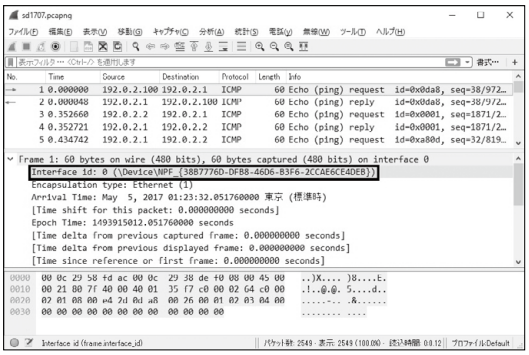
▼図1　[統計]メニュー



▼図2　pcap-ngファイル形式の[キャプチャファイルプロパティ]ウィンドウ



▼図3　pcap-ngファイル形式の[Frame]項目にあるインターフェースID



▼図4　wmicコマンドによるインターフェース情報の表示

```
C:\WINDOWS\system32>wmic nicconfig get description,index,ipaddress,macaddress,settingid
Description                               Index  IPAddress         MACAddress         SettingID
Intel(R) PRO/1000 MT Network Connection   0      {"192.0.2.101"}  00:0C:29:AF:68:39  {38B7776D-DFB8-46D6-B3F6-2CCAE6CE4DEB}

C:\WINDOWS\system32>
```

イル形式なら、「どのようなキャプチャフィルタがインターフェースに設定されていたのか」と「どのインターフェースでキャプチャされたパケットなのか」という情報が記録されます。

今度は、先ほどと同じ手順でpcap(libpcap)ファイル形式である「sd1707.pcap」をWiresharkで開いて、[統計]-[キャプチャファイルプロパティ]ウィンドウにキャプチャフィルタの内容が記録されているか、No.1のパケットの[パケット詳細]ペインにキャプチャしたインターフェースのGUIDが記録されているか確認してください。

「sd1707.pcap」の[キャプチャファイルプロパティ]ウィンドウ(**図5**)を見ると[キャプチャフィルタ]の下が「不明」となっていることから、pcap(libpcap)ファイル形式では記録されないことがわかります。

続いて[キャプチャファイルプロパティ]ウィンドウを閉じたら、No.1のパケットの[パケット詳細]ペインで[Frame 1:(省略)]を展開すると**図6**のようにパケットをキャプチャしたインターフェースの情報が記録されないことがわかります。このように②のpcap(libpcap)ファイル形式では、「どのようなキャプチャフィルタがインターフェースに設定されていたのか」と「どのインターフェースでキャプチャされたパケットなのか」という情報は記録されません。

pcap(libpcap)ファイル形式とpcap-ngファイル形式の詳細については本誌2014年11月号の同連載第1回、12月号の第2回、2015年1月号の第3回、9月号の第4回で解説しています。

問題2の解答

答えは③です。「sd1707_ipaddr.pcapng」で各ディスプレイフィルタを試してください。

③は「送信元IPアドレスまたは送信先IPアドレスのどちらか一方が192.0.2.1ではないパケットだけ」が表示されます。

①は「送信元IPアドレスまたは送信先IPアドレスのどちらか一方が192.0.2.1のパケットだけ」が表示されます。

②は間違えやすく非推奨となっている**!=**を使っていて、「送信元IPアドレスと送信先IPアドレスの両方が192.0.2.1ではないパケットだけ」が表示されます。

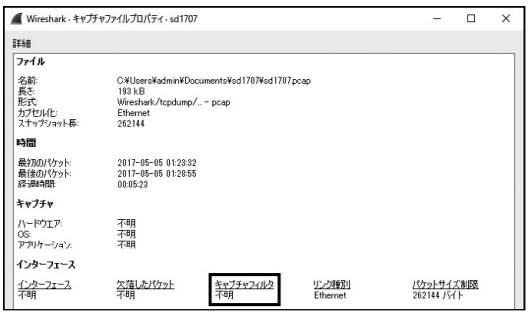
ディスプレイフィルタの使い方については、本誌2015年10月号の第5回で解説しています。

問題3の解答

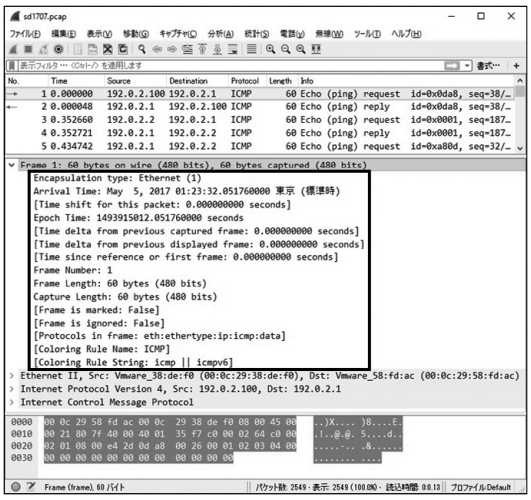
答えは②です。「sd1707_portscan.pcapng」で各ディスプレイフィルタを試してください。

②はTCPヘッダにある制御フラグ・フィールド全体の状態が条件に含まれます。制御フラグ・フィールドのうちSYNビットだけが立っ

▼図5 pcapファイル形式の[キャプチャファイルプロパティ]ウィンドウ



▼図6 pcapファイル形式の[Frame]項目



▼図7　TCP制御フラグ・フィールドの値

URG	ACK	PSH	RST	SYN	FIN
0	0	0	0	1	0

2進数→10進数 ➡ 2

ている状態を2進数で表すと、**図7**のように「000010」となり、これを10進数で表すと「2」になります。結果として、TCPの制御フラグ・フィールド全体の状態が000010のパケットだけを表示するといった意味になるため、SYNパケットだけが表示されます。

①と③はディスプレイフィルタを設定しようとするとフィルタツールバーの色が赤になることから、そもそもディスプレイフィルタとして正しくないことがわかります。

④はSYNビットの状態だけが条件に指定されているので、ほかのACKビットやFINビットなどの状態が条件に含まれないことに注意してください。結果としてSYNパケットのほかにもSYN+ACKパケットが表示されてしまいます。

ディスプレイフィルタの使い方については、本誌2015年10月号の第5回で解説しています(問題4、5、7の解答の際にも参考にしてください)。

問題4の解答

答えは②と③です。「sd1707.pcapng」で各ディスプレイフィルタを試してください。まずはWiresharkの時刻表示形式を日時に変更しましょう。[表示]メニューにある[時刻表示形式]を選んで[日時](**図8**)をクリックします。これで[パケット一覧]ペインの[Time]項目が日時で表示されます。

②はパケットをキャプチャした日時が2017年5月5日1:23:45.000000000以上で、なおかつ2017年5月5日1:23:46.000000000未満という条件になることから、2017年5月5日1:23:45.000000000から2017年5月5日1:23:45.999999999の間にキャプチャされたパケットだけが表示されます。

③は見た目こそ間違えているように見えますが、これでも構文としては正しく、②と同じ結果が表示されます。

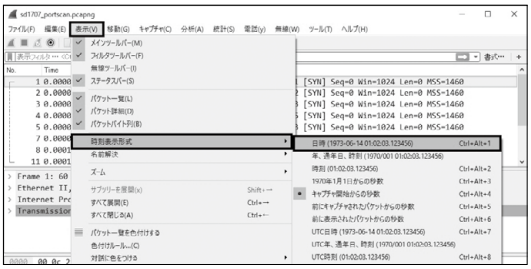
①は2017年5月5日1:23:45.000000000ちょうど(Just)にキャプチャされたパケットだけが表示されます。

問題5の解答

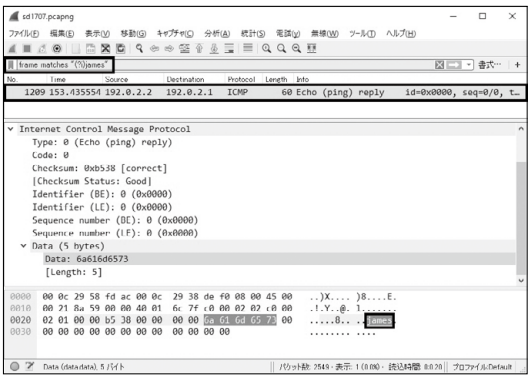
答えはNo.1209です。

パケット内の文字列を条件に検索するには`contains`や`matches`を使いますが、`matches`はPerl互換正規表現(PCRE)を使えるので便利かつ強力なことから、筆者は`matches`の使用をお勧めします。まずはパケットのどの部分に文字列が含まれているのかわからないので、パケット全体を指定する`frame`を使うことにします。あとは「james」が全部小文字かどうかもわからないので、大文字小文字を区別しない

▼図8　[時刻表示形式]を[日時]に変更



▼図9　問題5の解答のパケット



ように(?i)を文字列の頭に付けて次のようなディスプレイフィルタを設定します。

```
frame matches "(?i)james"
```

「sd1707.pcapng」を開いてディスプレイフィルタを設定した結果が図9になります。Wiresharkの下側に表示されている[パケットバイト列]ペインを見ると、確かに「james」という文字列を含んでいることがわかります。

問題6の解答

答えは21/tcp、22/tcp、80/tcp、443/tcpです。

SYNポートスキャンのしくみとしては、開いているか閉じているかわからないポートにSYNパケットを送信して、その反応からポートの開閉を判断します。そのポートからSYN+ACKが返信される場合は開いている、そのポートからRST+ACKが返信される場合は閉じていると判断します。

まずはSYNポートスキャンのスキャン対象となったポート番号を調べます。送信元IPアドレスが攻撃者(192.0.2.100)で送信先IPアドレスが標的(192.0.2.1)のSYNパケット(tcp.flags == 2)だけを表示すれば良いので、次のようなディスプレイフィルタを設定します。

```
ip.src == 192.0.2.100 && ip.dst ==
192.0.2.1 && tcp.flags == 2
```

このディスプレイフィルタを設定した結果が図10になります。SYNパケットの送信先ポート番号がスキャン対象なので、[パケット一覧]ペインの[Info]項目を見ると、21/tcp、22/tcp、23/tcp、25/tcp、53/tcp、80/tcp、139/tcp、443/tcp、445/tcpがSYNポートスキャンされたことがわかります。

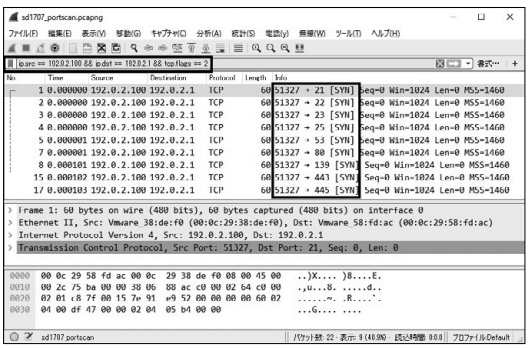
次にSYNポートスキャンで開いていると判断されたポート番号を調べます。SYNパケットに対してSYN+ACKパケットを返信したポート番号が開いていると判断することから、送信元IPアドレスが標的(192.0.2.1)で送信先IPアドレスが攻撃者(192.0.2.100)のSYN+ACKパケット(tcp.flags == 18)だけを表示すれば良いので、次のようなディスプレイフィルタを設定します。

```
ip.src == 192.0.2.1 && ip.dst ==
192.0.2.100 && tcp.flags == 18
```

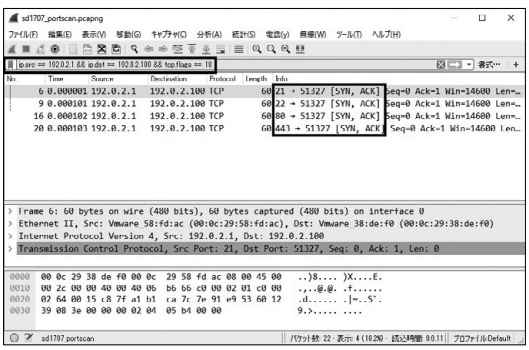
このディスプレイフィルタを設定した結果が図11になります。[パケット一覧]ペインの[Info]項目を見ると、SYNポートスキャンしたポートのうち、21/tcp、22/tcp、80/tcp、443/tcpからSYN+ACKパケットが返信されていることから、これらのポート番号がSYNポートスキャンで開いていると判断されたと推測されます。

ちなみにSYNポートスキャンを行ったポートスキャナの結果が図12になりますが、推測した結果と同じであることがわかります。

▼図10 SYNパケットだけを表示



▼図11 SYN+ACKパケットだけを表示



▼図12　SYNポートスキャンを行ったポートスキャナの結果（Kali Linuxで実行）

```
root@kali:~# nmap -P0 -n -r -sS 192.0.2.1 -p 21,22,23,25,53,80,139,443,445

Starting Nmap 7.40 ( https://nmap.org ) at 2017-05-05 01:31 JST
Nmap scan report for 192.0.2.1
Host is up (0.00013s latency).
PORT    STATE  SERVICE
21/tcp  open   ftp
22/tcp  open   ssh
23/tcp  closed telnet
25/tcp  closed smtp
53/tcp  closed domain
80/tcp  open   http
139/tcp closed netbios-ssn
443/tcp open   https
445/tcp closed microsoft-ds
MAC Address: 00:0C:29:58:FD:AC (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.20 seconds
root@kali:~#
```

キャプチャファイルからポートスキャンの結果を推測する方法については、本誌2015年11月号の第6回で解説しています。

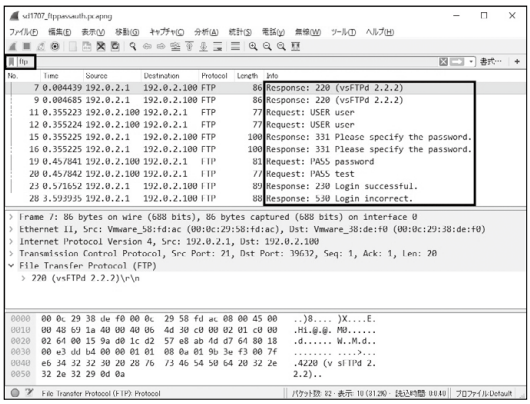
問題7の解答

答えは、ユーザ名「user_d」／パスワード「pass08」と、ユーザ名「user_h」／パスワード「pass04」です。

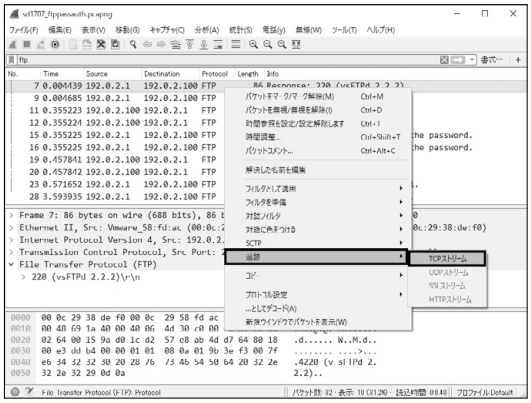
まずはWiresharkで「sd1707_ftppassauth.pcapng」を開いて、FTPサービスのパスワード認証のやりとりを調べましょう。キャプチャファイルを開いたら、**ftp**というディスプレイフィルタを設定すると**図13**のようにFTPのパケットだけが表示されます。［パケット一覧］ペインの［Info］項目を見ると同じような内容が2回ずつ表示されていますが、これはパスワード推測ツールがFTPサービスに対してほぼ同時に2つ接続したからです。パスワード推測ツールの多くは効率化のために複数の接続を行い、それぞれの接続でパスワード推測を行います。それぞれの接続でやりとりされた情報を追いかけるには、パケットをひとつひとつ見るよりストリームを追跡する機能が便利なので使いましょう。

試しに、No.7パケットを含む接続でやりとりされた情報を追いかけるには、［パケット一覧］ペインのNo.7パケットを右クリックしてコン

▼図13　ftpパケットだけを表示



▼図14　［パケット一覧］ペインのコンテキストメニュー



テキストメニューの[追跡]にある[TCPストリーム](**図14**)をクリックします。[TCPストリームを追跡]ウィンドウ(**図15**)が表示され、No.7パケットを含む接続でやりとりされた情報が赤と青で表示されます。赤は接続した側(192.0.2.100)が送信した情報、青は接続された側(192.0.2.1)が送信した情報になります。

内容を見るとFTPサービスのパスワード認証では「USER ユーザ名」でユーザ名を送信して、「PASS パスワード」でそのユーザのパスワードを送信していることがわかります。そして認証に成功した場合は、サーバ側から「230 Login successful.」という返事があることもわかります。

それでは、もう1つの接続も見てみましょう。[TCPストリームを追跡]ウィンドウを閉じたら、再度ディスプレイフィルタとして**ftp**を設定してください。その後はNo.9パケットを右クリックしてから[追跡]にある[TCPストリーム]をクリックして、[TCPストリームを追跡]ウィンドウ(**図16**)を表示します。パスワード認証の結果を見ると、パスワード認証に失敗した場合はサーバ側から「530 Login incorrect.」という返事があることがわかりました。

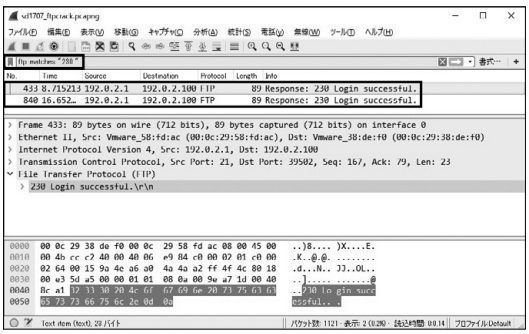
パスワード認証に成功した場合は「230 Login successful.」という返事があることから、この文字列を含むパケットだけが表示されるようにディスプレイフィルタを設定することで、パスワード認証に成功した接続だけが表示されます。このときに文字列の先頭部分にある「230」という成功を表す3桁のステータスコードを活用して、次のようなディスプレイフィルタを設定します。ステータスコードの後ろにある区切り文字のスペース(␣)も含めることで、なるべくステータスコードの部分が条件に当てはまるようにしています。

```
ftp matches "230␣"
```

▼図15　No.7パケットを含む接続の[TCPストリームを追跡]ウィンドウ

▼図16　No.9パケットを含む接続の[TCPストリームを追跡]ウィンドウ

▼図17　「230␣」という文字列を含むパケットだけを表示



Wiresharkで「sd1707_ftpcrack.pcapng」を開いて、このディスプレイフィルタを設定した結果が**図17**になります。2つの接続が表示されているので、[パケット一覧]ペインでNo.433パケットを右クリックしてから[追跡]にある[TCPストリーム]をクリックして、[TCPストリームを追跡]ウィンドウ(**図18**)を表示します。表示された内容を見ると、ユーザ名が「user_d」でパスワードが「pass08」のときに、「230 Login successful.」という返事があることから、「user_d」のパスワードは「pass08」ということが

わかります。

それでは、残るもう1つの接続も見てみましょう。[TCPストリームを追跡]ウィンドウを閉じたら、先ほどのディスプレイフィルタ(**ftp matches "230_"**)を再度設定してください。その後はNo.840パケットを右クリックしてから[追跡]にある[TCPストリーム]をクリックして、[TCPストリームを追跡]ウィンドウ(**図19**)を表示します。表示された内容を見ると、ユーザ名が「user_h」でパスワードが「pass04」のときに、「230 Login successful.」という返事があることから、「user_h」のパスワードは「pass04」ということがわかります。

ちなみにFTPレスポンスのステータスコードを条件にしたディスプレイフィルタは、次のように設定することもできます。

```
ftp.response.code == 230
```

こちらはFTPのパケットから「230」という文字列を探すのではなく、FTPの通信を解析してFTPレスポンスのステータスコードが「230」かどうかを判断しているので、**matches**を使う方法より正確な結果が期待できます。

問題8の解答

答えは次の9つのコマンドです。

```
・ipconfig
・whoami /user
・cd /
・dir
・cd inetpub
・dir
・cd wwwroot
・dir
・echo Hacked by James > index.html
```

Wiresharkで「sd1707_intrusion.pcapng」を開くと、攻撃者(192.0.2.100)が標的(192.0.2.2)のバックドア(4444/tcp)に接続して何か情報をやりとりしていることがわかります。

パケットを順に見ていくとNo.4パケットの[パケットバイト列]ペインにはWindowsのコマンドプロンプトのようなメッセージがあり、No.6パケットの[パケットバイト列]ペインには「ipconfig」というコマンドと思われる文字列があります。このことから、バイナリ形式ではなくテキスト形式の通信である可能性が高いので、試しにストリームを追跡する機能を使ってみましょう。

[パケット一覧]ペインでNo.4パケットを右クリックしてから[追跡]にある[TCPストリーム]をクリックして、[TCPストリームを追跡]ウィンドウ(**図20**)を表示します。前述のとおり、赤は接続した側(192.0.2.100)が送信した情報

▼図18　No.433パケットを含む接続の[TCPストリームを追跡]ウィンドウ

▼図19　No.840パケットを含む接続の[TCPストリームを追跡]ウィンドウ

で、青は接続された側(192.0.2.2)が送信した情報になります。

内容を見るとWiresharkが正しく表示できずに所々文字化けしていますが、攻撃者(192.0.2.100)から標的(192.0.2.2)のバックドア(4444/tcp)宛にさまざまな文字列が送信されていて、その文字列がコマンドとして標的(192.0.2.2)にあるWindowsのコマンドプロンプトで実行されていることがわかります。

あとは[TCPストリームを追跡]ウィンドウの赤で表示されている文字列を調べれば、攻撃者がバックドアに接続して実行したコマンドがわかります。

ちなみに攻撃者が実行したコマンドを順に見ていくと、次のように動いていたことがわかります。

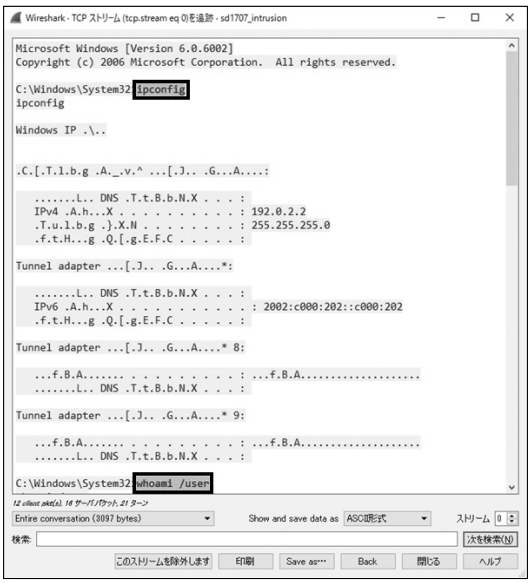
❶`ipconfig`で侵入したサーバが接続されているネットワークを確認
❷`whoami /user`でコマンドを実行する標的側ユーザを確認
❸`cd /`でCドライブ直下に移動
❹`dir`でCドライブ直下にあるファイルやディレクトリを確認
❺`cd inetpub`でinetpubに移動
❻`dir`でinetpub直下にあるファイルやディレクトリを確認
❼`cd wwwroot`でwwwroot(IISのWeb用ディレクトリ)に移動
❽`dir`でwwwroot直下にあるファイルやディレクトリを確認
❾`echo Hacked by James > index.html`でwwwroot直下にあるindex.htmlを改ざん

さらにNo.43のパケットでTCPストリームの追跡を行うと、[TCPストリームを追跡]ウィンドウ(**図21**)の内容から、攻撃者はindex.htmlを改ざんしたあとに標的(192.0.2.2)の80/tcpにWebブラウザで接続して、コンテンツを改ざんできているか確認していることがわかります。

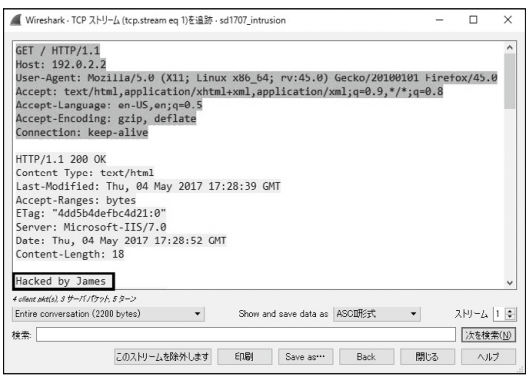
おわりに

今回はJamesの挑戦状ということでWiresharkについての問題に挑戦していただきましたが、いかがでしたでしょうか。本誌2014年11月号から連載をスタートして、ふと気付いたら第10回になりましたが、いまだに紹介できていない機能や使い方がWiresharkにはたくさんあるので、引き続き本連載で取り上げて解説していきたいと思います。SD

▼図20 [TCPストリームを追跡]ウィンドウで攻撃者の行動を追跡



▼図21 コンテンツを改ざんできているかを確認する攻撃者

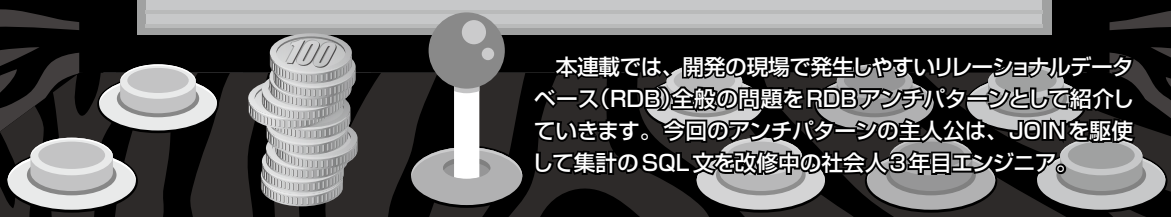


RDBアンチパターン

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3回 やりすぎたJOIN



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、JOINを駆使して集計のSQL文を改修中の社会人3年目エンジニア。

前回は「失われた事実」と題して、保存するデータ設計の重要性を説明しました。今回は「やりすぎたJOIN」として、RDBだからこそ陥るJOINの罠を説明します。

やりすぎたJOIN

RDBは前回でもお話したとおり、リレーショナルモデルに沿って設計・正規化することが大事です。そして、そのように作ったデータベースから正しいデータを取り出すには、必ずJOINが必要になります。しかし、正規化を正しく利用した設計がされていないと、ボトルネックになりやすいのもJOINの特徴の1つです。

JOINの機能はRDBMSによって異なるため、今回はOSSのRDBMSを題材に解説します。

事の始まり

社会人3年目のKさんはSQLが大好き。

Kさん：集計のSQL、今は2個になってるけどこうすれば……1個のSQLにできる！　よーし、この改善で不要なクエリを1つ減らせたぞ！これを今のバッチと差し替える前に本番で実行してみよう。……あれ、終わらない？

実行直後、監視システムからDB負荷に関するアラート通知が飛んできた。それに合わせ、飛んできたかのような勢いで先輩エンジニアのTさんが現れる。

Tさん：今すぐそのクエリを止めるんだ！
Kさん：あっ……すみません。わかりました。

実行中のクエリを止めることで負荷がみるみる下がり、アラートは収まった。

Kさん：すみません、SELECT文なので大丈夫かと思って本番で実行しました。staging環境で動作確認もしましたし……。
Tさん：ちょっとクエリ（**リスト1**）を見せてくれる？　やりたいことは、指定した合計単価以上の会員の組み合わせを取り出したいだけだよね？それには不要なテーブルのJOINが多すぎるな。実行計画を見てみると単価表の会員idにはINDEXがないね。不等号のJOINだからなおさらINDEXが必要だ。WHERE句の前にONで絞り込むことでタスクテーブルは小さくできるね。……ほかにも修正点があるから、ちょっとJOINについて説明しよう。
Kさん：はい、お願いします！

▼リスト1　Kさんが作ったクエリ

```
SELECT
  単価表1.単価 AS 単価1
  , 会員1.会員id AS 会員1id
  , 単価表2.単価 AS 単価2
  , 会員2.会員id AS 会員2 id
FROM
  単価表 AS 単価表1
  INNER JOIN 単価表 AS 単価表2
    ON 単価表1.単価id < 単価表2.単価id
    AND 単価表1.会員id <> 単価表2.会員id
  INNER JOIN 会員 AS 会員1
    ON 単価表1.会員id = 会員1.会員id
  INNER JOIN 会員 AS 会員2
    ON 単価表2.会員id = 会員2.会員id
  INNER JOIN 都道府県 AS 都道府県1
    ON 会員1.出身県id = 都道府県1.県id
  INNER JOIN 都道府県 AS 都道府県2
    ON 会員2.出身県id = 都道府県2.県id
  INNER JOIN 会社 AS 会社1
    ON 会員1.会社id = 会社1.会社id
  INNER JOIN 会社 AS 会社2
    ON 会員2.会社id = 会社2.会社id
WHERE
  （単価表1.単価 + 単価表2.単価）> :合計単価;
AND 会社1.会社名 = '株式会社 そーだい'
AND 会社2.会社名 = '株式会社 そーだい'
```

JOINの特性

代表的なINNER JOIN

アンチパターンの分析の前に、まずはJOINの特性について説明します。JOINはその名のとおり、テーブルとテーブルの結合です。

代表的なJOINであるINNER JOINを使った次のSQLを実行した場合のイメージは、**図1**のとおりです。

・INNER JOIN[注1]

```
SELECT
    会員.名前 AS 会員名
    , 都道府県.名前 AS 出身県
FROM
  会員
  INNER JOIN 都道府県
    ON 会員.出身県id = 都道府県.県id
```

また、JOINは集合の和の結果ですので、ベン図でよく表現されます(**図2**)。このように、重なり部分だけを取得するのがINNER JOINです。

そのほかのJOIN

JOINにはほかにもいくつか種類があります(**図3**)。実務でよく使うのは、LEFT OUTER JOINとRIGHT OUTER JOINの2つです。

・LEFT OUTER JOIN[注2]

```
SELECT
  *
FROM
  会員
  LEFT OUTER JOIN 都道府県
    ON 会員.出身県id = 都道府県.県id
```

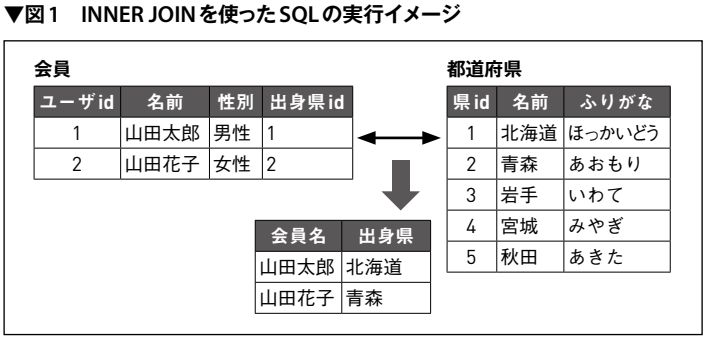
・RIGHT OUTER JOIN[注3]

```
SELECT
  *
FROM
  会員
  RIGHT OUTER JOIN 都道府県
    ON 会員.出身県id = 都道府県.県id
```

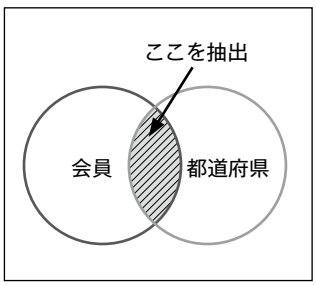
ベン図の内、すべてを取ってくるJOINを

注1）INNERを省略して単にJOINと書くこともできる。
注2）OUTERを省略してLEFT JOINと書くこともできる。
注3）OUTERを省略してRIGHT JOINと書くこともできる。

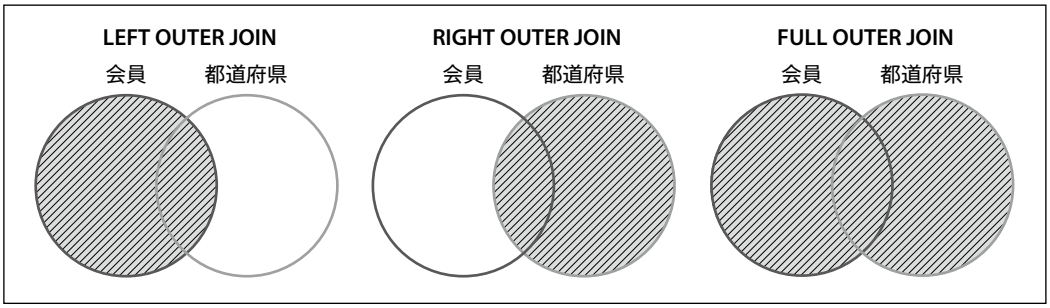
▼図1　INNER JOINを使ったSQLの実行イメージ



▼図2　INNER JOINを使ったSQLをベン図で表現



▼図3　そのほかのJOIN



FULL OUTER JOINといいます。

・FULL OUTER JOIN[注4]

```
SELECT
  *
FROM
  会員
  FULL OUTER JOIN 都道府県
    ON 会員.出身県id = 都道府県.県id
```

MySQLはFULL OUTER JOINをサポートしていないため、次のようにRIGHT JOINの結果をLEFT JOINの結果とUNIONすることで表現します。

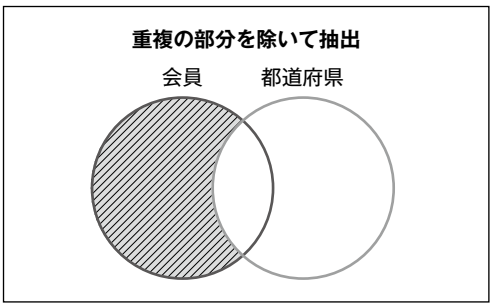
```
SELECT
  *
FROM
  会員
  LEFT OUTER JOIN 都道府県
    ON 会員.出身県id = 都道府県.県id
UNION
SELECT
  *
FROM
  会員
  RIGHT OUTER JOIN 都道府県
    ON 会員.出身県id = 都道府県.県id
```

そのほかにも、次のようにして差の部分を抽出することができます(**図4**)。

```
SELECT
  *
FROM
  会員
  LEFT OUTER JOIN 都道府県
    ON 会員.出身県id = 都道府県.県id
WHERE
  都道府県.県id IS NULL
```

注4) OUTERを省略してFULL JOINと書くこともできる。

▼図4　重複部分を除去



✤ JOINの問題点

ベン図が**図5**のように増えていくとどうでしょう？　3つの重なりを調べる場合は、次のようにテーブルを調べる必要があります。

AとB／AとC／BとC

さらに4つになると、次のとおりです。

AとB／AとC／AとD／BとC／BとD／CとD

このように指数関数的に増加します。JOINの回数が増えると急激に重くなるのがJOINの特徴の1つです。

また、JOINは掛け算と言われます。テーブルスキャンの場合、100行と100行のJOINの場合は10,000行のテーブルスキャン相当ですが、10,000行と10,000行ではなんと100,000,000行です。

このように、JOINはSQLの処理の中でもっとも重い処理の1つと言えるでしょう。しかし

ながら、多くのRDBMSにはこの重い処理を高速に処理するための工夫がたくさんあります。たとえば、100行と「一意なINDEXが貼られた100行」の場合は、100行+(100×1)行となり200行相当です。INDEX1つでこのように大きく計算結果が変わるのです。次は、JOINの高速化についてみてみましょう。

JOINのアルゴリズムの種類

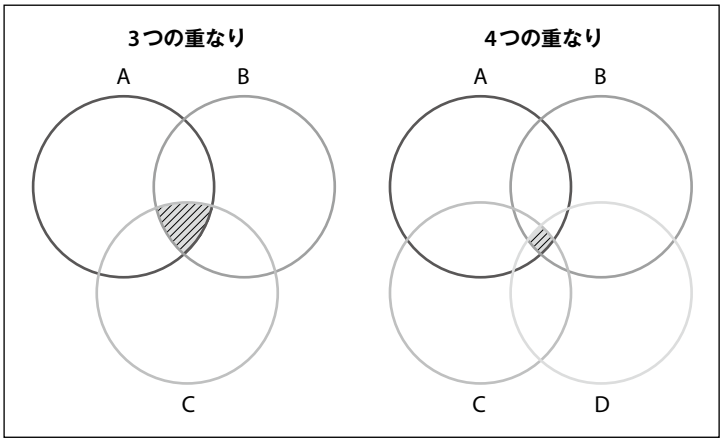
JOINの高速化のコツを知るためには、JOINのアルゴリズムを知る必要があります。RDBで使われるJOINのアルゴリズムには、おもに次の3つがあります。

- Nested Loop Join(NLJ)
- Hash Join
- Sort Merge Join

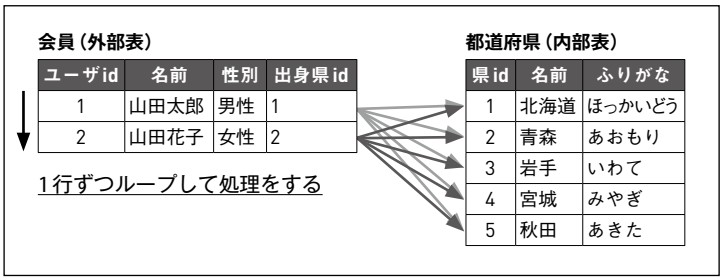
それぞれの動作をまとめると**図6**～**図8**となり、特徴をまとめると**表1**となります。図の見方ですが、ループの元になるテーブルが「外部表」(この場合は会員テーブル)、ループの先になるテーブルが内部表(この場合都道府県テーブル)となります。また外部表は、別名「駆動表」とも呼ばれます。

一言にJOINと言っても、大きく異なる3種類のアルゴリズムがあるのです。また注意点として、PostgreSQLは

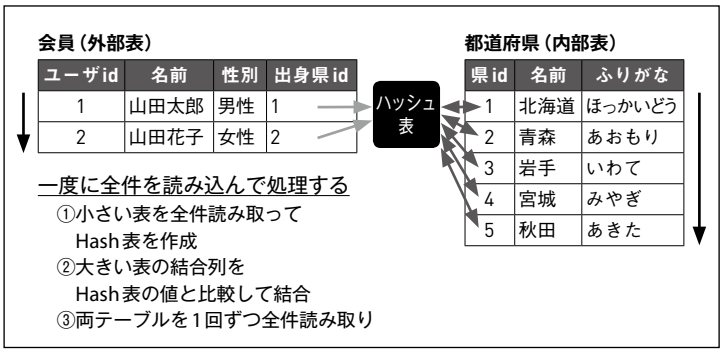
▼図5　増えるベン図



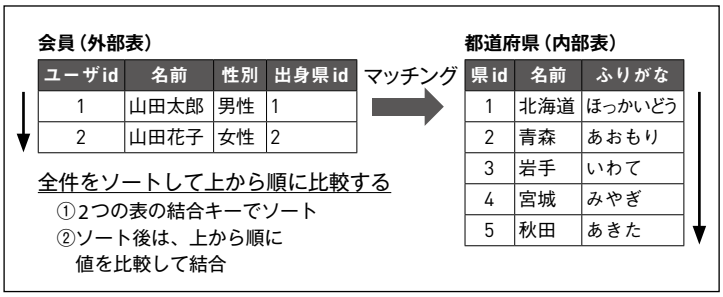
▼図6　NLJの動作



▼図7　Hash Joinの動作



▼図8　Sort Merge Joinの動作



▼表1　JOINの3アルゴリズムの特徴

アルゴリズム	特徴
NLJ	・内部表の結合キーの列に利用できるINDEXがある場合、ループ数を省略できるため外部表が小さいほど高速になる ・内部表の結合キーが一意の場合は内部表対象レコードを絞りこめるため、より高速になる ・1レコードずつ確定するので、確定したレコードはレスポンスとして返すことができる
Hash Join	・「外部表が大きい場合、または内部表の対象件数が多い場合」と「結合条件の索引がなく、テーブルのフルスキャンが必要な場合」ではNLJより有利 ・Hash表を作成さえすれば結合は非常に高速だが、Hash表の作成・保存ができるだけの十分なメモリが必要
Sort Merge Join	・ソートに用いる索引が作成されていると高速化できる ・Hash Joinと同様に表の大部分を結合する場合に有効 ・Hash Joinと違い、等値結合だけでなく不等式(<、>、<=、>=)を使った結合にも利用できる ・INDEXが活用できる場合はHash Joinより速い場合もある

3種類のJOINをサポートしていますが、MySQLはNLJしかサポートしていません。PostgreSQLはMySQLに比べ、大きな2つの表のJOINや不等号を使ったJOINが得意と言えます。

しかし、MySQLはJOINが苦手かと言われれば一概にはそうではなく、表の説明にもあるとおり、内部表に適切なINDEXがあり、小さな外部表をもとに等号で結合する場合は、非常に高速に処理できます。MySQLはNLJの特徴をより活かした設計が得意と言えるでしょう。

問題点

今回のアンチパターンには、ここまで紹介してきたJOINの特性をふまえると次のような問題があります。

・JOINに対する不理解
・多段JOINと不要なJOIN
・JOINの内部表にINDEXがない

これらはパフォーマンスに直結します。staging環境などのデータが小さい場合は問題がなくても、本番環境の大きなデータではサービスに影響を与えるほど大きな負荷になることもあります。また、最初の**リスト1**を見てわかるように複雑なクエリ(スパゲッティクエリ)の要因になります。

このアンチパターンのポイントと対策

今回のアンチパターンは「JOINを知らずに不用意に多段JOINをした」ということになります。

まずJOINの特性を知れば、大きなテーブルのJOINについては注意を払いますし、INDEXの活用場所も見えてきます。Kさんが書いたクエリを書き直した場合の例ですと、**リスト2**のようなクエリになります。このクエリは、条件に関係なかった都道府県テーブルを排除し、単価表の計算を行ったあとに、そのテーブルに対して絞り込んだ会員テーブルをJOINしています。これにより、複数回行っていた会員テーブルや会社テーブルのJOINを最低限にしています。また単価表テーブルの会員idにはINDEX

▼リスト2　リスト1を書きなおし

```
SELECT
  単価表1.単価 AS 単価1
  , 単価表1.会員id AS 会員1id
  , 単価表2.単価 AS 単価2
  , 単価表2.会員id AS 会員2id
FROM
  (
    SELECT
      単価表1.単価 AS 単価1
      , 単価表1.会員id AS 会員1id
      , 単価表2.単価 AS 単価2
      , 単価表2.会員id AS 会員2id
    FROM
      単価表 AS 単価表1
      INNER JOIN 単価表 AS 単価表2
        ON 単価表1.単価id < 単価表2.単価id
        AND 単価表1.会員id <> 単価表2.会員id
    WHERE
      (単価表1.単価 + 単価表2.単価) > :合計単価
  ) AS 単価組み合わせ表
  INNER JOIN 会員
    ON 会員.user_id = 単価組み合わせ表.会員1id
    AND 会員.user_id = (SELECT 会社id FROM
会社 WHERE 会社名 = '株式会社 そーだい')
```

▼リスト3　リスト2をViewを使って書きなおし

```
CREATE VIEW 単価組み合わせ表 AS
SELECT
  単価表1.単価 AS 単価1
  , 単価表1.会員id AS 会員1id
  , 単価表2.単価 AS 単価2
  , 単価表2.会員id AS 会員2id
  , (単価表1.単価 + 単価表2.単価) AS 単価合計
FROM
  単価表 AS 単価表1
  INNER JOIN 単価表 AS 単価表2
    ON 単価表1.単価id < 単価表2.単価id
    AND 単価表1.会員id <> 単価表2.会員id

SELECT
  単価組み合わせ表.単価1 AS 単価1
  , 単価組み合わせ表.会員1id AS 会員1
  , 単価組み合わせ表.単価2 AS 単価2
  , 単価組み合わせ表.会員2id AS 会員2
FROM
  単価組み合わせ表
  INNER JOIN 会員
    ON 会員.user_id = 単価組み合わせ表.会員1id
  INNER JOIN 会社
    ON 会員.会社id = 会社.会社id
    AND 会社.会社名 = '株式会社 そーだい'
WHERE 単価合計 > :合計単価
```

を追加することで、データが大きくなった際にはSort Merge Join に効いてきます。

しかしこのクエリは、わかりやすいクエリと言い難いのが正直なところです。このような場合は、Viewを活用するとシンプルになります（**リスト3**）。JOINの回数を減らして高速化できたうえに、シンプルになりました。この例では単価組み合わせ表を汎用的に使うために単価合計での絞り込みを外に持ってきました。Viewを活用することでスパゲッティクエリを防ぐこともできますので、テクニックとして覚えておきましょう。

さらに、単価表の更新が少ない場合や更新が1日1回で良い場合などは、集計結果を別テーブルとして保存することでViewのようにクエリをシンプルにしつつも、計算結果を実体として持っていますので高速に参照できます。

また、このようにSQLの結果を実体のあるViewにする機能としてマテリアライズド・ビューがあります。マテリアライズド・ビューはクエリの結果のテーブルを作ることと一緒ですが、再作成のときにテーブルの作りなおしが不要で、共有ロックのリフレッシュのみで良い、などのメリットがあります。SQL Serverなどの商用DBでは有料機能ですが、PostgreSQLでは9.3以降から無料で使えますので、PostgreSQLをお使いの方はぜひお試しください。

それでは、今回のアンチパターンの対策ポイントをまとめます。

・JOINは必要最低限
・INDEXを適切に活用する
・JOINするテーブルは小さくしてからJOINする
・複雑なクエリになった場合はViewを活用する

これらの点をふまえて、JOINを有効活用しましょう。読者のみなさんへの注意点として、JOINに対して間違った認識を行い「重い処理だからJOINは禁止！」などのルールを作ってしまうと、逆にN+1問題の温床になったり、無駄なクエリを発行することになります。繰り返しますが、「JOINはRDBを使ううえで必要な機能」ですので、正しく理解したうえで有効活用をしましょう。

またJOINにおけるINDEXの挙動に関しては@yoku0825さんのスライド「Where狙いのキー、order by狙いのキー」[注5]が非常に参考になります。MySQLの話ですが、MySQL使い以外の人にもとてもわかりやすく、応用が利く話ですのでぜひ見てみてください。

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？　JOINの話題の中でも、INDEXの重要性についてご理解いただけたかと思います。次回はそんなRDBの主役の1人、INDEXの話題です。今まで使っていたINDEXが急に効かなくなったなど、RDBのパフォーマンスに直結のテーマです。次回の「効かないINDEX」もお楽しみに！ SD

注5）URL https://www.slideshare.net/yoku0825/where order-by/

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17回 JOINのロックが怖くて飯が食えるか!!

開発現場でたまに目にするJOIN禁止ルール。システムのスケールアウトがしにくくなる、性能を悪化させるなどのデメリットが禁止のおもな理由です。しかし、そのデメリットに根拠はあるのでしょうか？　回避はできないのでしょうか？　技術者ならデメリットにおびえるのではなく、うまく回避してメリットを享受しましょう。

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。

帰ってきたJOIN禁止論

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。

「生島さん、ちょっとお尋ねしたいんですけど！」

と、当連載では久しぶりに登場の五代さんが、私の顔を見るなり大きな声で呼びかけてきました。

「経緯は僕から説明します」という大道君に話を聞いてみると**図1**のようなことでした。

「ははあ、よくいるJOIN禁止論者ですね。率直に言って『a. 原因不明のトラブル』っていうのはアンタの技術力がないだけやろ、という感じですし、b. のほうも単に設計が悪いだけでしょう。c. はコストの考え方が間違っていると思いますね」

ちなみにb. については、本来は「そもそもRDBMSで処理すべきなのか？」から考えるべきです。ドキュメント型やグラフ型など、データ構造が本質的にテーブル型ではないシステムを作るにはそもそもSQLは向いていないので、たとえばブログや画像、動画サイトなどのコンテンツ部分はNoSQLに向いています。最初は手慣れたRDBで試作するけれど将来はNoSQLに移行するといった見込みがあるならJOINなし

▼図1　JOIN禁止の理由

(1)一部の協力会社の技術責任者が、強硬にJOIN禁止の方針を出していて困っている

(2)JOIN禁止の理由は次の3つ
- a. 原因不明のトラブルを起こしやすい
- b. 将来ユーザが増えたときにスケールアウトしにくい
- c. 以上の悪影響によりJOINを使うとコスト高になる

(3)ちょうど折悪しく、OracleからMySQLに移行しようとしているプロジェクトの案件でJOINがらみの部分でトラブルが起きたことも彼の主張を後押ししてしまった

という方針でも良いでしょうが、将来に渡ってRDBMSを使うならJOINを使いながらスケールアウトを効かせる方法はあります(当連載第10回[注1]で紹介)。つまり、きちんと設計しておけばスケールアウトに対応するのは容易で、そのほうがトータルコストは小さくなります。

「やはり生島さんはそう思われますよね。そこで、とくにコストについての考え方を一度きちんと整理しておきたいんですわ」

「なるほど、わかりました。でも、(3)のトラブルっていうのはなんですか？　そこが解決しなければそのJOIN禁止派さんがますますドヤ顔になりそうですね」

「いや、それがまだなんですよ……」

「じゃあ、そっちからやりましょうよ」

MySQLではロックの粒度が粗くなる?

ということで調べてみたところわかったのは、ECシステムの注文処理時に、「在庫があったら引き当てて注文を確定し、なかったらロールバックする」という処理で性能劣化およびデッドロックが多発していたということでした。そこで使われていたのは**リスト1**のようなSQL文です。

「いわゆる『ぐるぐる系』じゃないです。普通にJOINを使って読んでいるんですが……」

「この-- z.在庫数というコメントが気になりますね。これ、OracleからMySQLに移行したと言っていましたよね？」

「そうです。もともとはOracleでした」

「ははあ、それで読めた。MySQLではロックの粒度が違うことによる問題でしょうね、これは」

「どういうことですか？」

言うまでもなく、ロックというのはシステムの中の1つしかないリソースを多数のプロセス(スレッド)が同時に使おうとしたときに発生するもので、1つのプロセスが占有すると残りはそれが解放されるまで待たされるため、レスポンスタイムの悪化を引き起こします。この種の現象は同時ユーザ数やデータ量が増えたときに起こりやすく、逆に言うと常に再現するわけでも不正なデータが残るわけでもないため、「原因不明のトラブル」として受け止められやすいものです。

「実はOracleとMySQLではこのSQLでのロックの動きが違うんよ。それで、MySQLに移行したら問題が起きたんやろね」

「それはひょっとして-- z.在庫数というコメントが関係あるんですか？」

「そのとおり。実はMySQLではSELECT文でFOR UPDATE z.在庫数という構文が使えない。だから、移行するときにもともと入っていたz.在庫数を削ってコメントにして残しておいたんやろ。それで文法的には通るようになったけれど、ロックで問題が起きたというわけや」

注1) 本誌2016年12月号。

▼リスト1　性能劣化を招いていたSQL文

```
BEGIN

SELECT *
    FROM 在庫数 z
         INNER JOIN 製品マスタ p
              ON z.製品ID = p.ID
             (他、いくつかのマスタ)
    WHERE
         z.製品ID = ?
    FOR UPDATE ;      -- z.在庫数
```

サブクエリ化でロック範囲を制限する

OracleとMySQLでのロックの違いを簡単に書くと**図2**のようになります。OracleではFOR UPDATEのあとにテーブル名.カラム名を指定することでロックの範囲を狭くすることができますが、MySQLではできません。

「じゃあ、これはOracleからMySQLに移行するときに起こりやすい問題なんですか」

「そういうことやね。で、これを回避するにはSELECT文を**リスト2**のようなものに変えてやればいい」

「いったんサブクエリを使うことで、FOR UPDATEの範囲は在庫数テーブルだけだよということを教えてやるんですか」

「そういうこと。こうしても実行計画は素直にJOINしたのと同じになるからパフォーマンスは変わらない。ロック範囲が狭くなるだけ」

そう聞いて実際に実行計画を調べてみる大道君。

「本当ですね、変わらないですね！」

「そんなわけで、下手な使い方をしたら、そりゃあいろいろと問題起こしますけど、それを『a. 原因不明のトラブル』と呼ぶのは単にSQLを知らな過ぎ、技術力がないということだと思いますよ」

デッドロックの発生パターン

「ついでにもうひとつ教えてください。単に同時競合だったら性能は落ちても動くと思うんですけど、デッドロックというのはどうして起こるんですか？」

「そういう場合はこいつを見てくれ」

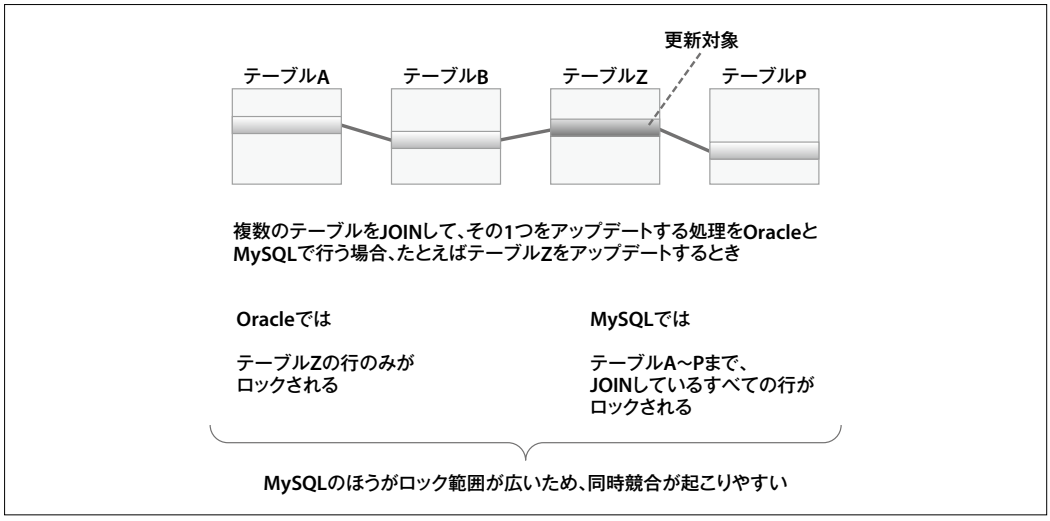
図3が典型的なデッドロックの発生パターンです。プロセスが2つあり、どちらもリソースAとBを使いますが、プロセス1はA→B、2はB→Aとそれぞれ違う順序で排他的ロックをかけようとしています。リソース側の「L」はロックされた状態、「F」は解放された状態を表します。

「ここで実行のタイミングがたまたま『1がAをロック→2がBをロック』という順序で進むとどうなる？」

▼リスト2　FOR UPDATEでのロック範囲限定方法（MySQL向け）

```
SELECT *
FROM
    (SELECT * FROM 在庫数 z WHERE z.製品ID = ? FOR UPDATE) z
    INNER JOIN 製品マスタ p
               ON z.製品ID = p.ID
              (他、いくつかのマスタ);
```

▼図2　OracleとMySQLのSELECT～FOR UPDATE



「あ……その時点でAB両方がロックされた状態になって、その後もうひとつのリソースをロックしようとすると解放されるまで待たされて……お互い相手を待ってて止まってしまうわけですか」

「そういうこと」

デッドロックはタイミング依存の障害ですからテストでは発見しにくく、実運用に入って負荷が上がったところで起きるため、開発者にとってはやっかいな問題です。この1年、よく勉強してきた大道君でも知らなかったように、RDBでの開発経験の長いエンジニアでもわかっていないことは珍しくありません。

「ということは……プロセス1と2がどちらもA→Bの順番でロックをかけるように作っていれば、これは発生しないんですか？」

「そう、それがデッドロック回避の一番の基本やね。それにはDBへのアクセスシーケンスをきっちりと管理する必要があって、そのためにもDB担当とUI担当を分けるAPIファースト開発をしたほうがいいんよ」

回避できるデメリットはデメリットではない

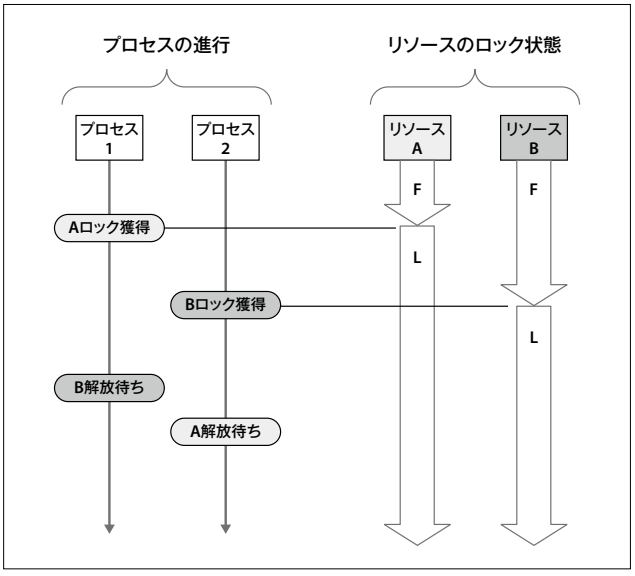
そこで五代さんがおもむろにつぶやきました。

「今回みたいに一応技術リーダーをやっている人間にJOINのデメリットを説明されると、そういうものかと思ってしまいますけど、単に回避する方法を知らないだけなんですかねえ」

「少なくとも、デメリットがあるから使わない、というのはおかしいですよね。新しい技術にはデメリットはあるに決まっています。それを上回るメリットがあるから使われるので、デメリットは防ぐ方法を講じればいいだけです」

新しい問題には新しいテクノロジが作られるものです(**図4**)。何十年もの歴史があるRDBを「新しいテクノロジ」と呼ぶのも変なものですが、現代でも手続き型言語からプログラミングを覚えた人にとっては、集合指向のパラダイムで作られたRDBは「新しいテクノロジ」と言えるでしょう。新しいテクノロジにはそれぞれ特有のメリットもデメリットも複数あるもので、それを活かす方法も防ぐ方法もきちんと

▼図3　典型的なデッドロックの発生パターン



学ばなければ有効には使えません。それをきちんと学んでいない人間はデメリットを過大評価／メリットを過小評価して「新しいテクノロジを使わないことを正当化」していることがあります。JOIN禁止というのもそれに類する主張と思われます。

JOINを使うと高コストになる？

「ところでコストの考え方についてなんですけど」

「コストというのも、パターンがいろいろあるやないですか。定常的にかかる費用がいくらで一時的にかかるのがいくら、そしてそれをどの程度正確に読めるのか？　そのへんの相場観がわからんのですわ」

「一理あるのは、**図1**の(2)のb、『スケールアウトしにくい』というのは必ずしも嘘とは言えない。JOINを使っていると、DBサーバが1台で済まなくなって分散させるときにはちょっと手間がかかる。ただ、最初からそれを想定して設計しておけば大したことはない。つまり知っていれば回避できるわけや！」

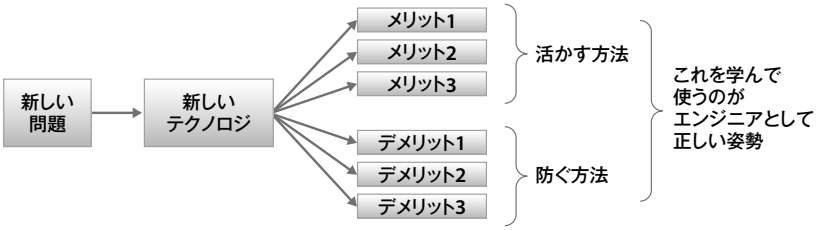
「そういうことなんやね」

「まあ、コストを『大したことない』とか『ちょっと』とか定性的に言うのは難しいけど、傾向としては**図5**のようなイメージになる、私の意見としては」

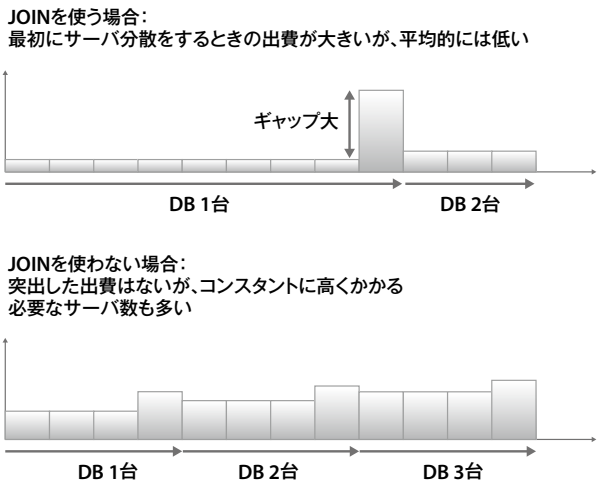
「JOINを使ったほうがサーバは少なく済むんですか？」

「JOINをしようがしまいが、必要なデータは取ってこなきゃいけない。それをJOINして取

▼図4　「デメリットがあるから使わない」というのはエンジニアにとって正しい姿勢ではない



▼図5　JOINを使う場合と使わない場合のコスト・パターン



るとその瞬間は負荷が高くても1発で済むのに対して、分割して取ると低い負荷が何度もかかるので結局トータルではかえって重くなる(DBサーバとは別のキャッシュを使っている場合を除く)」

「JOINしたほうが出費が平均的には低い、というのはサーバ台数が少ないから？」

「プログラムが単純になるので開発／保守工数が低く済むし、もちろん、全般にサーバ負荷も減るから、台数を減らせるという理由もある。JOINを使うほうがDB/Web/APサーバともにスケールアウトを迫られる時期が大幅に遅く(台数は少なく)済むからね。従量制でCPU課金されるようなクラウドを使っている場合、コストの差はもっと顕著に出るね」

「サーバを増やすときのギャップがエライ違いますね？」

「JOINを使っていると、2台以上に増やすときにプログラムの改修が必要になるのでギャップが大きいんですよ。といっても、スケールアウトを予想してJOINを避けるなら、最初からJOINしながらスケールアウトするように設計しておけば回避できます」

この件は当連載第10回で触れた「マスタのコピーを作る」という方法です。

「JOINを使わない場合もサーバを増やすときに少し工数がかかりますか……」

「そこはどうしても、物理サーバでも仮想サーバでも、機材の設置や設定変更、分散範囲の切り分けに多少の作業が発生するからね。プログラムの改修ではないから単純な作業で済むけど」

「ふむ……」

人は知らないことは不安に思うもの

考え込む五代さんに私のほうから聞いてみました。

「もしこれが実際のコスト・パターンだとしたら、経営者の視点ではどちらが望ましいですか？」

「そうですね、お金というのは結局総額でいくらになるかが問題ですから……**図5**で言えば面積が小さいほうが安くつくわけですから、JOINを使うべきやろね……」

「私もそう考えるんですけど、実際にその選択をしようとしたときに何か不安に感じることはありますか？」

「もちろん、ギャップの大きさは気になりますわ。本当にこれで済むのかどうか。実際やってみたらこの5倍かかりましたとか、改修でバグを出してサービスが止まりましたとか、そういう事態が起きへんかという不安は感じますね」

「あ、そうなんですね。実は私は逆で、JOINを使ったときのギャップは最初からそう設計しておくことでこの図の半分とか1/4まで落とせる見込みで考えますし、JOINを使わないほうはプログラムが複雑化しますんで、開発／保守コストがもっと膨らむんじゃないかと、そっちのほうを心配しますね」

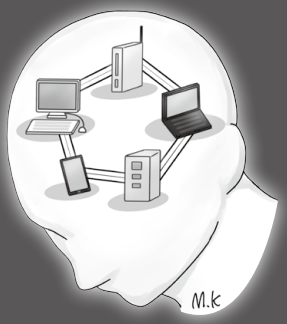
「あ、そっちですか。うーん……人間、知らないことは不安に感じるってことですかね、それは。今回JOIN禁止を言ってきた人も、実はSQLをよく知らないからギャップが大きく見えてるだけなのかなあ……」

「やっぱりちゃんと勉強しなきゃダメなんですね」とつぶやいた大道君のほうを見て、五代さんは少し考え込んだあと、言いました。

「JOINを使うほうが経営的にはメリットがありそうなことはわかりました。ただ、そのメリットを得るためには、ちゃんとSQLがわかる技術責任者を確保しなきゃいけないってことですよね？」

「そうなりますね」

「これはうちの人事戦略にかかわる問題ですな……いつも生島さんを頼れるわけじゃないですから、中途で雇うか社内で育てるか、いずれにしても社内の技術力を上げなきゃいけない。それを会社の方針にして社長レベルからの働きかけで、技術に対して取り組む姿勢を変えていかんといけませんね。ということで……これからもご協力ください！　大道君も、頼むで！」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14回 仮想環境リモート運用管理の実装例

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8回目です。

前回(本誌2017年6月号)は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したが、今回はその具体的な実装例のポイントについて解説します。一般的な利用者管理APIのWebブラウザやセキュリティツールでWebサーバと仮想環境運用管理インフラを経由して、仮想マシンの利用部門がリモート運用管理する具体的な実装例について考えていきます。

実装のしくみ

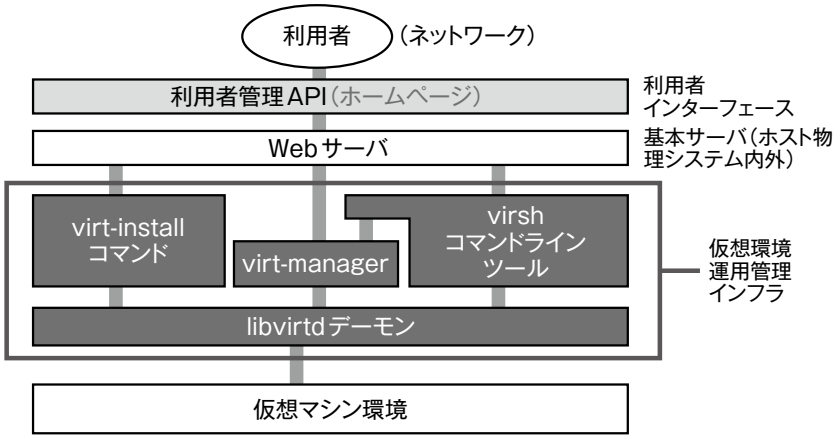
実装のしくみは、前回にも掲載した**図1**のように、「仮想環境運用管理インフラと利用者管理API」の中の、利用者管理API(ホームページ)、Webサーバ、そして、仮想環境運用管理インフラ、を通して仮想マシン環境にアクセスして運用管理するというものです。

「仮想環境運用管理インフラ」内では、virt-installコマンドとvirshコマンドラインツールによりlibvirtdデーモンを制御します。

基本的な考え方

基本的な考え方は、前回までの解説のように、一般的なアプリケーションインターフェースで簡単に利用できるようにすること、仮想マシンの基本的なハードウェア管理(ハードウェア制

▼図1　仮想環境運用管理インフラと利用者管理API(連載第13回 図1再掲)



御など)とシステム管理(電源オン直後からのコンソール操作など)を行うこと、です。

また、リモート運用管理に必要な仮想マシンの情報データベース(DB)が必要です。

一般的なアプリケーションインターフェース

利用部門の技術者が扱いやすい一般的なアプリケーションである、Webブラウザや VNC などを使うのが基本です。これらを通して運用管理操作を行えるようにします。これらは一般的なアプリケーションですので、利用端末はPCばかりでなく、携帯端末でも可能になります。

ハードウェア管理

仮想マシンでのデバイスの着脱やCD/DVDメディアのセット、電源オン／オフ、システム起動／強制停止など、(通常の物理システムと同様の)ハードウェア操作をリモートから行います。

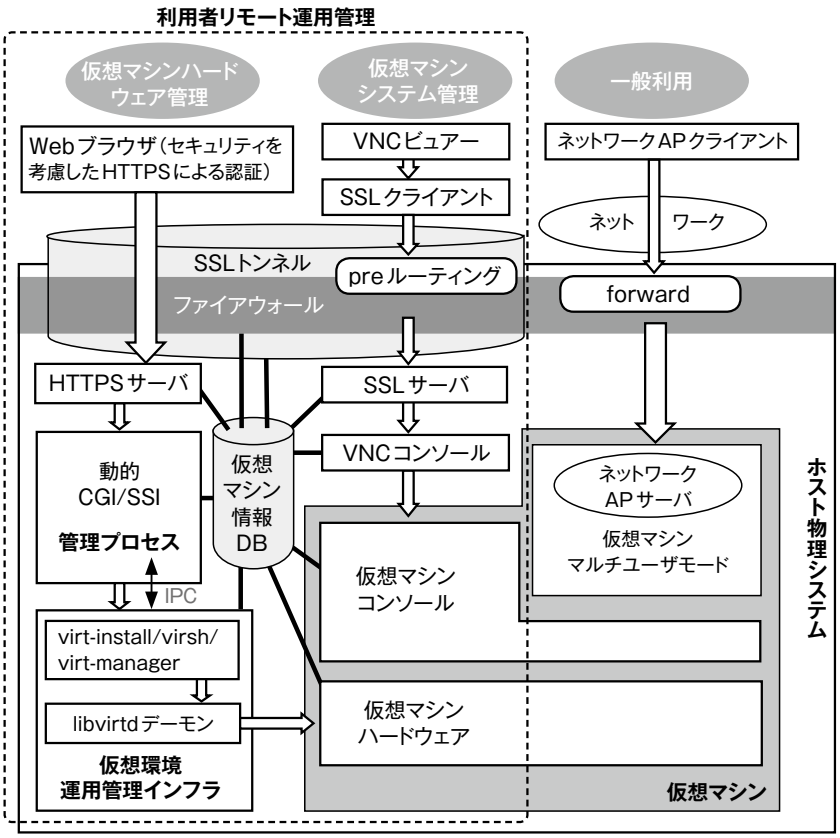
システム管理

OSのインストールや、電源オン直後のBIOS設定やシングルモードなど、システムやネットワークが稼働していない状態でのコンソールもリモートから操作します。

具体的な実装の構造例

実装のしくみをブレークダウンすると、いくつかの考え方がありますが、一般的な構造はたとえば、**図2**のようになります。基本的な操作は、仮想マシンハードウェア管理、仮想マシンシステム管理、そして、一般利用です。

▼図2　利用者による仮想マシンのリモート運用管理の実装図



※IPC：Inter-Process Communication(プロセス間通信)

仮想マシンハードウェア管理

一般的なアプリケーションであるWebブラウザからネットワークを通して、もちろんセキュア通信(HTTPSプロトコル)を使い、Webサーバ経由で仮想環境運用管理インフラを制御・利用する、結果を戻す(逆方向)という流れです。

Webブラウザ(HTTPSプロトコル)

Webブラウザから物理ホストのWebサーバに利用者認証を行ってから接続して仮想マシンのハードウェア制御情報を伝えます。これらはSSLトンネル経由で行います。

リモート運用管理のWebページフローは次のようなものが考えられます。

1ユーザホームページ
- (処理)ポータルページ
  - ①クライアント証明書の認証とユーザ名/パスワードの認証を行う
  - ②正当ユーザのみ受け付け、ユーザ個別ホームページへのリンクを表示
- (下位連携)ユーザ個別ホームページ

2ユーザ個別ホームページ
- (処理)ユーザメイン画面
  - ①ユーザ名に対応したグローバルIPアドレス[注1]で相手認証を行い、正当ユーザのみ受付
  - ②それぞれのcapability(権限能力)に対応した仮想マシンの制御メニューを表示
  - ③ユーザが選択した操作情報を仮想マシン制御デーモン・クライアントに渡す
- (メニュー)状態表示、起動、停止、再起動、OSインストール準備[注2]、利用者情報、利用ガイドやサポート情報
- (上位連携)ユーザホームページからの入りのみ
- (下位連携)仮想マシン制御

利用者認証

利用者認証は一般的なホームページの認証管理「セッション管理」です。いくつかのWebページを渡り歩く利用になるので、そのセッションを唯一認識するようにしておかなければなりません(セッション状態の保持)。

セッション管理としては、セッションIDやCookieによりstateful(状態保持)通信を行う方法が一般的ですが、ここでは認証とセキュリティを併用するために、ユーザID/パスワードと証明書(SSL)により入口から入り、次のページ以降ではその情報+IPアドレスとから制御管理することにします。

そのためには、運用管理サーバ上でSSL証明書を作成します(手順は**図3**)。証明書はサーバと利用クライアントの両方です。

HTTPSサーバ設定ファイルで、ユーザ名/パスワードと証明書チェックの設定をしておきます(**リスト1**)。

▼図3　SSL証明書のサーバ側の設定およびクライアントの設定手順

1. httpd.confのSSL設定

2. SSL証明書の作成
- 2-1. サーバプライベートキーの作成
- 2-2. 自動起動のためのパスフレーズ問い合わせの除去
- 2-3. サーバ証明書作成依頼書の作成(CSR)の作成
- 2-4. 自己認証局(CA)プライベート鍵および証明書作成
- 2-5. 自己認証局CA証明書のPC用x509変換
- 2-6. 自己認証局CA署名によるサーバ証明書作成
- 2-7. クライアントプライベート鍵の作成
- 2-8. クライアント証明書発行要求(CSR)の作成
- 2-9. 自己認証局CA署名によるクライアント証明書の作成
- 2-10. クライアント証明書のPC用PKCS#12変換

3. Windowsでの証明書のインポート

※ここではサーバとクライアントの両方の証明書を自己認証局CAを作成して署名しているが、正式には公的なCAで署名・証明書作成を依頼する(上記、2-4、2-5、2-6、2-9)。

注1) 利用ユーザのユーザ名やグローバルIPアドレスなどのほか、仮想マシン用デバイス、VNCポート番号/パスワード、利用OSなどを制御情報として持つ。

注2) インストールOSのフレーム作成とインストール起動。インストールのオペレーション操作はSSL+VNCで行う。なお、このSSL/VNCはサーバ側はstunnelなどのSSLラッパからVNCサーバ、クライアント側はSSL+VNCで行うようにあらかじめ準備しておく。SSLクライアント証明書も必須。SSL双方向認証によるセキュリティ強化のため。

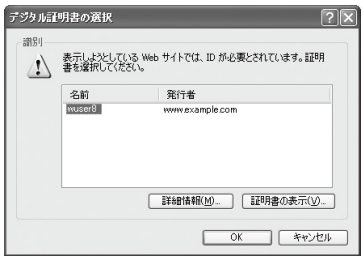
▼リスト1　HTTPSサーバのユーザ名／パスワード設定

ファイル：httpd.conf
```
<Directory "/var/www/shtml">
  AuthType Basic
  AuthName "Members Page"
  AuthUserFile /etc/httpd/conf/passwd
  AuthGroupFile /etc/httpd/conf/group
  Require group ugroup
```

ファイル：group
```
ugroup: wuser10 wuser6 wuser7 wuser8 wuser9
```
許容ユーザ名

ファイル：passwd
```
wuser10:QiZwrpBPUCKEY
wuser6:QxoKThqFQTqLg
wuser7:kQfnIx6XqdjQI
wuser8:aIt3qqh1tiSBE
wuser9:Utj66hax1uk1M
```
ユーザ名とハッシング・パスワード

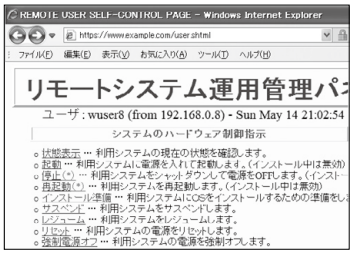
▼図4　証明書選択



▼図5　ユーザ名／パスワード入力認証

▼図6　リモート運用管理ポータルページ

▼図7　リモート運用管理ページ



運用管理ホームページ

最初の入口ページはユーザID／パスワード入力とクライアント証明書確認を行う簡単なユーザホームページです(**図4～6**)。

後続のページは、入力されたユーザID／パスワードと接続端末のIPアドレスから、CGI/SSIで動的に作成して表示し、接続クライアントに対応した運用管理ページになります(**図7**)。そのため、このときに仮想マシン情報DBが重要な役を負います。

ファイアウォール

運用管理のHTTPS接続は物理ホスト上(または、別の物理ホスト上)のHTTPSサーバに接続を許可しますが、個別仮想マシンに対応した(許可された)リモート運用管理端末のIPアドレスに応じてフィルタ許可する必要があります。許可されていないリモートIPアドレスからの着信は許可しません(**リスト2**)。

HTTPSサーバ

受け取った情報から仮想マシン管理プロセスのクライアントを起動します。

クライアント－サーバ間のSSL証明書処理を行うために、証明書の管理・保存を確実にし

ておく必要があります。また、セキュリティを厳しくするために、ユーザID／パスワードおよび証明書の属性情報なども確認チェックが必要です。

管理プロセス

HTTPSサーバプロセスはCGIやSSIから管理プロセスを起動・処理し、その結果を戻します。

管理プロセスは受信した目的の操作情報をvirt-install/virsh/vairt-managerなどの仮想環境運用管理インフラに送信します。管理プロセスは、HTTPSサーバにより動的に起動され、制御処理終了後は自分でexitします。なお管理プロセスは、仮想環境運用管理インフラを操作するために管理者属性がなければなりません。

管理プロセスがクライアント－サーバ接続する仮想環境運用管理インフラは、通常のサーバアプリケーションプロセスのように、口を開けて待っている状態です。また、管理プロセスと仮想環境運用管理インフラは、IPC(Inter-Process Communicaton：プロセス間通信。システム内)で制御処理と結果返送を行います。**図8**は仮想マシンの状態を確認したページです。

仮想環境運用管理インフラ

仮想環境運用管理インフラでは、virt-install/virsh/virt-managerがlibvirtdデーモンに仮想マシンのハードウェアドライバの制御管理を行わせます(**表1**)。

仮想マシンシステム管理

仮想マシンのコンソール処理には、KVMで提供されているVNCを使います。ただし、セキュリティ保持のためには、SSL上で行う必要があります。

▼リスト2　リモート仮想マシン管理のためのファイアウォール設定
※外部から仮想ホストへのリダイレクト(外部から物理ホストへの着信の、仮想ネットワークへのリダイレクト)

```
対象：<iptables/natテーブル：PREROUTING/DNAT>
外部から各利用者がそれぞれの仮想ホストを個別に利用する、個別利用者ホスト－仮想ホストを1対1対応
(物理ホスト：192.168.0.18、外部ホスト：192.168.0.8、192.168.0.12と仮定)

# [192.168.0.8] から[192.168.0.18:20-9999/tcp]への接続は[192.168.122.181]へリダイレクト。
-A PREROUTING -s 192.168.0.8 -i eth0 -p tcp -m tcp --dport 20:9999 -j DNAT --to-destination➥
 192.168.122.181
# [192.168.0.8] から[192.168.0.18:domain/udp]への接続は[192.168.122.181]へリダイレクト。
-A PREROUTING -s 192.168.0.8 -i eth0 -p udp -m udp --dport domain -j DNAT --to-destination➥
 192.168.122.181
# [192.168.0.12]から[192.168.0.18:20-9999/tcp]への接続は[192.168.122.182]へリダイレクト。
-A PREROUTING -s 192.168.0.12 -i eth0 -p tcp -m tcp --dport 20:9999 -j DNAT --to-destination➥
 192.168.122.182
# [192.168.0.12]から[192.168.0.18:domain/udp]への接続は[192.168.122.182]へリダイレクト。
-A PREROUTING -s 192.168.0.12 -i eth0 -p udp -m udp --dport domain -j DNAT --to-destination➥
 192.168.122.182
# 仮想マシンコンソール[192.168.0.18:10021/10022/10023/10080/10443/15801/15901/15911/15912/➥
15913]への接続は[192.168.0.18:21/22/23/80/443/5801/5901/5911/5912/5913]へリダイレクト。
-A PREROUTING -i eth0 -p tcp -m tcp --dport 10021 -j DNAT --to-destination  192.168.0.18:21
-A PREROUTING -i eth0 -p tcp -m tcp --dport 10022 -j DNAT --to-destination  192.168.0.18:22
-A PREROUTING -i eth0 -p tcp -m tcp --dport 10023 -j DNAT --to-destination  192.168.0.18:23
-A PREROUTING -i eth0 -p tcp -m tcp --dport 10080 -j DNAT --to-destination  192.168.0.18:80
-A PREROUTING -i eth0 -p tcp -m tcp --dport 10443 -j DNAT --to-destination  192.168.0.18:443
-A PREROUTING -i eth0 -p tcp -m tcp --dport 15801 -j DNAT --to-destination  192.168.0.18:5801
-A PREROUTING -i eth0 -p tcp -m tcp --dport 15901 -j DNAT --to-destination  192.168.0.18:5901
-A PREROUTING -i eth0 -p tcp -m tcp --dport 15911 -j DNAT --to-destination  192.168.0.18:5911
-A PREROUTING -i eth0 -p tcp -m tcp --dport 15912 -j DNAT --to-destination  192.168.0.18:5912
-A PREROUTING -i eth0 -p tcp -m tcp --dport 15913 -j DNAT --to-destination  192.168.0.18:5913

外部から仮想マシンコンソールへはstunnel経由でSSL復号後、VNCコンソールへリダイレクト
```

▼図8　リモート運用管理例

▼図9　SSL：VNCパスワード認証

▼図10　SSL：VNC接続例

▼表1　仮想マシンのハードウェアドライバの制御管理のコマンド例

コマンド	用途
list	状態、情報などの表示
create	定義からの生成
start	起動
shutdown	シャットダウン
reboot	再起動
install	インストール
suspend	サスペンド
resume	レジューム
reset	リセット
setting	情報DB設定
poweroff	電源停止

WindowsではWindowsのstunnel経由でVNCビュアーを使い、物理ホスト側のSSL + VNCで認証して(**図9**)から接続します(**図10**)。

preルーティング

クライアントシステムからの接続は、一般利用と同じ経路を通るため[注3]、ファイアウォールでは通常、仮想マシンにルーティングされます。

しかし、リモート運用管理のSSL + VNCで仮想マシンをシステム管理する、この場合は、「物理ホスト」側に接続しなければなりません。そのため、この接続は一般の接続とは異なり、物理ホストシステムにpreルーティング(ルーティング処理の前に行き先を変える)する必要があります(**リスト2**)。

ここでは、VNC接続を解説していますが、一般のシステム管理ではtelnetやftp、sshなどの管理端末処理も同様です。

注3)　一般利用(図2の一番右側)では仮想マシンのSSLサーバやVNCサーバに接続する場合はそのまま仮想マシンに転送されます。このとき、IPアドレスが同じとなるので宛先の切り分けが必要です。

ファイアウォール

リモートシステム管理でも対応する仮想マシンに対応した(許可された)リモート運用管理端末のIPアドレスに応じてフィルタ許可する必要があります。

SSLサーバ(物理ホスト)

クライアントからの(preルーティングされた)SSL接続はSSLサーバで復号されて、仮想マシンのVNCコンソールへ平文接続されます。実装例としてはstunnelがあります(**リスト3**)。

VNCコンソール(物理ホスト)

VNCコンソールは物理ホスト上で仮想マシンコンソールとして動作します。物理ホスト上でKVM仮想マシンの一般的な操作端末です。

仮想マシンコンソール

ローカルVNCにより仮想マシンコンソールの操作が可能になり、電源オン後の、あるいは、シングルモードなど仮想マシンのネットワーク機能が動作していないときでも、コンソール端末として操作が可能です。

仮想マシン情報DB

仮想マシンの情報DBには、リモート運用管理に必要な情報すべてが入っていなければなりません。

物理ホストシステム上のユーザアカウントID／パスワード、その証明書情報、リモート運用管理端末のIPアドレス／マスク、仮想マ

▼リスト3　stunnelの関連設定

```
; サーバ証明書／キー
cert = /etc/pki/tls/certs/server.crt
key  = /etc/pki/tls/private/server.key
; SSLプロトコル (all, SSLv2, SSLv3, TLSv1)
sslVersion = all
options = NO_SSLv2
; 証明書の検証を行う
verify = 2
; CA証明書ファイル
CAfile = /etc/pki/CA/cacert.crt
; サービス
[vncs]  ;物理ホスト
accept  = 15901  ;着信ポート
connect = localhost:5901
; ↑復号後の平文接続転送ポート

; ### KVM 仮想マシン宛 SSL console vnc ###
[vnc-s186]
accept  = 15916
connect = localhost:5916

[vnc-s187]
accept  = 15917
connect = localhost:5917

[vnc-s188]
accept  = 15918
connect = localhost:5918

[vnc-s189]
accept  = 15919
connect = localhost:5919

[vnc-s190]
accept  = 15920
connect = localhost:5920
```

シンの割り当てIPアドレス、MACアドレス、VNC接続のためのユーザアカウントID／パスワード、OS情報、メモリ情報、ディスク情報、など多数あります。

このDBはセキュリティを厳しく管理されていなければなりません。そのためには、ファイル属性のみならず、暗号化(ハッシングだけでは平文を読み出しできないので不可)や認証、さらには、アクセス排他制御などの機能も必要です。

また、この情報を利用者が適時に追加・更新などできるようにすること、さらに初期インストールやバックアップなどを行うツールも必要になります。

実装のポイント

仮想マシン環境を実装する場合、その利用用途を十分に考慮したつくりにしなければなりません。つまり、誰が、どのように接続して利用するか、に適切に対応する必要があります。

仮想マシン間あるいは、外部システムと仮想マシンとの間のコミュニケーションが想定されるので、接続する、利用元のシステムと利用先の仮想マシンとの間の、アドレスやプロトコル、ポート(アプリケーション)、フロー(さらにはデバイス)などを考慮することになります。

また、仮想マシンの利用と、(本連載で解説している、仮想マシンの利用者による)管理を個別に考えなければなりません。つまり、利用用途に応じて、利用接続と管理接続の相手を特定し限定します。

ただし、仮想マシンの利用用途にはさまざまなものがあるため、それに応じてセキュリティを含むシステム化、体系化が必要になります。

繰り返しになりますが、仮想マシンにはWebサーバやメールサーバなどインターネットに公開するサーバ、社内などの組織専用で利用するサーバ、開発システムなど一時的システムとしての利用、BYOD[注4]によるクライアントシステムとしての利用など多々あります。

次回、最終回はこれらについて解説していきます。SD

注4) BYOD：Bring Your Own Device、個人電子機器の社内利用。

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

書いて覚える Swift 入門

第27回 静かなること型の如し

Author 小飼弾（こがい だん）　twitter @dankogai

型の完全理解は可能か?

今回はSwift最大の特長であるプロトコル(protocol)を、総称型(generic type)と絡めつつ紹介します。そのためには、型(type)とは何かをまず理解しておく必要があります。型とは何か、なんとも深淵そうな質問で、実際それだけでTAPLこと『型システム入門[注1]』という名著がまるごと1冊書けてしまうほどなのですが、本連載は「書いて覚えるSwift入門」。実際に書いていくことにしましょう。

0 == 0.0 // compile error

REPLで次のとおりに入力してみましょう。

```
var i = 0
i == 0
i == 0.0
```

macOSでは次のようになります。

```
Welcome to Apple Swift version 3.1
(swiftlang-802.0.53 clang-802.0.42). Type
:help for assistance.
  1> var i = 0
i: Int = 0
  2> i == 0
$R0: Bool = true
  3> i == 0.0
error: repl.swift:3:3: error: binary operator
'==' cannot be applied to operands of type
'Int' and 'Double'
i == 0.0

repl.swift:3:3: note: expected an argument
list of type '(Int, Int)'
i == 0.0
```

(Objective)?C(++)?やJavaなどのコンパイル言語に慣れた人にとっては当たり前のこの挙動は、JavaScriptやPerlやPythonやRubyなどのスクリプト言語にとっては驚きの結果です。

・node(JavaScript)

```
> var i = 0
undefined
> i == 0
true
> i == 0.0
true
```

・perl -de 1

```
main::(-e:1):1
  DB<1> my $i = 0

  DB<2> p $i == 0
1
  DB<3> p $i == 0.0
1
```

・python

```
>>> i = 0
>>> i == 0
True
>>> id  == 0.0
True
```

注1） URL https://estore.ohmsha.co.jp/titles/9784274069111P

・irb(ruby)

```
irb(main):001:0> i = 0
=> 0
irb(main):002:0> i == 0
=> true
irb(main):003:0> i == 0.0
=> true
```

スクリプト言語で0 == 0.0が成立する理由は厳密にはそれぞれの言語で異なるのですが、Swiftで0 == 0.0が成立しない理由は明白です。型が一致しないからです。0はIntという型で、0.0はDoubleという型になります。そしてSwiftの型は静的。コンパイルの段階でどの変数(および定数)がどんな型なのかがあらかじめ決まっているので、0 == 0.0は実行すらさせてくれないというわけです。

なぜSwiftでは0と0.0は別々の型なのでしょう?

別の役割が期待されているからです。

たとえば割り算。IntとDoubleでそれぞれ/してみましょう。

```
  1> var i = 42
i: Int = 42
  2> i / 10
$R0: Int = 4
  3> var d = 42.0
d: Double = 42
  4> d / 10
$R1: Double = 4.2000000000000002
```

かたや4、かたや4.2000000000000002。何が違うか。そう、余りです。

```
  5> i % 10
$R2: Int = 2
  6> d % 10
error: repl.swift:6:3: error: '%' is
unavailable: Use truncatingRemainder instead
d % 10

Swift.%:2:13: note: '%' has been explicitly
marked unavailable here
public func %(lhs: Double, rhs: Double) ->
Double
```

整数の範囲で「割り切る」代わりに「余り」も%で出せるのがIntの/で、精度一杯まで「割り続ける」代わりに「余り」を出さないのがDoubleの/。こういった区別がない言語では、==が楽な代わりにほかで苦労しています。たとえばJavaScriptにはDoubleに相当するNumberはあってもIntに相当する型はないので、Swiftの42 / 10に相当する演算は(42 / 10) | 0などとしなければなりません。

引数をそのまま返すだけの簡単なお仕事

ここで、1番目に簡単な関数を考えてみましょう。ちなみに0番目に簡単な関数は**何も引数を取らず何もしない関数**で、Swiftならばこうなります。

```
func noop(){}
```

これが0番目なら、1番目は当然**1つ引数をとってそれをそのまま返す関数**になるでしょう。簡単ですね——動的言語なら。

・JavaScript

```
function id(x){ return x }
// es6 ならもっと簡単に var id = (x)=>x;
```

・Perl

```
sub id { @_ }
```

・Python

```
def id(x):
  return x
```

・Ruby

```
def id(x)
  x
end
```

それではSwiftでは? Swiftは静的型言語(大事なことなので何度も繰り返します)。関数を定義するときには、引数と戻り値を明示しな

ければなりません。ということは……

```
func id(_ x:Int)->Int { return x }
func id(_ x:Double)->Double { return x }
func id(_ x:String)->String { return x }
// ...
```

こういうのを繰り返し書かなければならないということでしょうか？　やってることどころか{}の中身もまったく同じなのに？

ここで颯爽と登場するのが**総称型**(generic type)。次のように書いておけば……

```
func id<T>(_ x:T)->T {
    return x
}
```

何でもござれです。

```
  1> func id<T>(_ x:T)->T {
  2.     return x
  3. }
  4> id(0)
$R0: Int = 0
  5> id(0.0)
$R1: Double = 0
  6> id("")
$R2: String = ""
  7> id([0])
$R3: [Int] = 1 value {
  [0] = 0
}
  8> id([0:""])
$R4: [Int : String] = 1 key/value pair {
  [0] = {
    key = 0
    value = ""
  }
}
```

ここでidは**総称関数**(generic function)、Tは**プレイスホルダー型**(placeholder type)と言います。

得意なことは違うから

この何でもござれぶりを見ると、関数という関数を総称型で書きたくなってきますが、これは以前言った「本当に必要なとき以外Any型は使うべきではない」と同様な意味でムチャ振りです。それを実感するため、今度はl == rに相当するeq(l,r)を同様に書いてみましょう。

```
  1> func eq<T>(_ l:T, _ r:T)->Bool {
  2.     return l == r
  3. }
  4.
error: repl.swift:2:14: error: binary operator
'==' cannot be applied to two 'T' operands
    return l == r

repl.swift:2:14: note: overloads for '=='
exist with these partially matching parameter
lists: (Any.Type?, Any.Type?), (UInt8, UInt8),
/* 中略 */ (UnsafePointer<Pointee>,
UnsafePointer<Pointee>)
    return l == r
```

なんかむちゃくちゃ文句言って来ましたよ。「Any型には==はない」と前回も言いましたが、==演算子はどんな型でもOKとはいかない以上、総称しようがないのです。ということは、

```
func eq(_ l:Int, _ r:Int)->Bool {return l == r}
// ...
```

の時代に戻らなければいけないということでしょうか？

ここでいよいよプロトコルが登場します。要は「==演算子を持つ型」というのを何とか表現できればいいのですよね？　これでどうだ？

```
func eq<T:Equatable>(_ l:T, _ r:T)->Bool {
  return l == r
}
```

今度はうまくいきました！

```
  1> func eq<T:Equatable>(_ l:T, _ r:T)->Bool {
  2.     return l == r
  3. }
  4> eq(0,0)
$R0: Bool = true
  5> eq(0.0,0.0)
$R1: Bool = true
  6> eq("","")
$R2: Bool = true
```

このEquatableのことを**プロトコル**(proto

col)といい、T:Equatableで「Tという型はEquatableというプロトコルに準拠(conform)している」ことを表現します。

めでたし、めでたし？

――ちょっと待った！　これは？

```
  4> [0]==[0]
$R0: Bool = true
  5> eq([0],[0])
error: repl.swift:5:8: error: argument type
'[Int]' does not conform to expected type
'Equatable'
eq([0],[0])
```

なぜ[0]==[0]はうまくいくのにeq([0],[0])はうまくいかないのでしょう？　むしろ[0]==[0]がうまくいくほうが不思議ではありませんか？　Array自体はEquatableではないのに……。

「それ自体はプロトコル準拠ではないけど、中身は準拠している」ということをSwift語で何と言えばいいのでしょうか？

こういうときに便利なのがswiftdoc.org[注2]。ArrayやDictionaryやRangeが共通して準拠しているSequenceをよく見てみるとelementsEqual[注3]なるメソッドが存在するではありませんか。

```
func eq<T:Sequence>(_ l:T, _ r:T)->Bool
    where T.Iterator.Element:Equatable
{
    return l.elementsEqual(r)
}
```

このようにして、実行してみると

```
  4> eq([0],[0])
$R0: Bool = true
  5> eq(0...1,0...1)
$R1: Bool = true
```

うまくいったようですが、これでもまだ完璧ではありません。

```
  7> eq([0:""],[0:""])
error: repl.swift:7:3: error: type '(key: Int,
value: String)' does not conform to protocol
'Equatable'
eq([0:""],[0:""])
```

SequenceとしてのDictionary<K,V>のElementは(K, V)というタプル型で、これがEquatableではない、と。

さすれば……

```
func eq<K: Equatable,V: Equatable>
    (_ l:[K:V], _ r:[K:V])->Bool
{
    return l == r
}
```

これを実行してみると、

```
6> eq([0:""],[0:""])
$R0: Bool = true
```

これでDictionaryもeq()できるようになりました。

オレオレプロトコルの書き方

それでは同様に、Collectionの中身を総和するsumというメソッドを書いてみましょうか。そのためにはCollectionのIterator.Elementが演算+をサポートしていることをSwiftが知っていればよいわけですが、==のEquatableと違って+にAddableというプロトコルは見当たりません。

ならば定義してしまいましょう。

```
protocol Addable {
    static func +(_ l:Self, r:Self)->Self
}
```

これは、次のとおりに読めます。

注2）URL http://swiftdoc.org/
注3）URL http://swiftdoc.org/v3.1/protocol/Sequence/#func-iterator-element_-equatable-elementsequal_

> *プロトコルAddableに準拠している型は、自分自身と同じ型を持つlとrを受けて同じ型の値を返す+という二項演算子を持つ*

IntやDoubleといった数値型のみならずStringもAddableに準拠しているのは明白ですが、残念ながらSwiftは良きに計らってくれません。プロトコルに準拠しているのだということをSwiftに教えるには、空のextensionを用います。

```
extension Int: Addable{}
extension Double: Addable{}
extension String: Addable{}
extension Array: Addable{}
……(中略)……
```

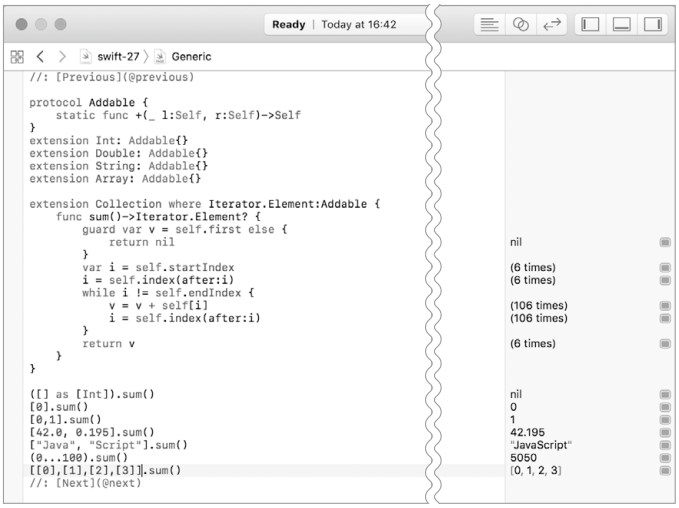
ここでプロトコルに準拠していない型にextensionをかけるとエラーで止まります。

```
extension Dictionary: Addable {}
// error: type 'Dictionary<Key, Value>' does not conform to protocol 'Addable'
```

これで準備は完了。あとはCollectionを拡張するだけです。

```
extension Collection where Iterator.Element:Addable {
    func sum()->Iterator.Element? {
        guard var v = self.first else {
            return nil
        }
        var i = self.startIndex
        i = self.index(after:i)
        while i != self.endIndex {
            v = v + self[i]
            i = self.index(after:i)
        }
        return v
    }
}
```

▼図1　実行画面



要素をイテレートするのにずいぶんまだるっこしい方法を使っていますが、これはArrayやRangeをそのまま拡張するのではなく、Collectionというプロトコルを拡張しているから。たとえばArrayだけであれば、

```
extension Array where Element:Addable {
    func sum()->Element? {
        guard var v = self.first else {
            return nil
        }
        for i in 1..<self.count {
            v = v + self[i]
        }
        return v
    }
}
```

とより簡潔に書けますが、(0...100).sum()のようにRangeまでまとめて拡張することはまなりません。

次回予告

というわけで今回もコード盛りだくさんでお届けしましたが、次回はWWDCの知見をなるべくお伝えしたうえで、次回にまたプロトコルについて続きを話します。SD

presented by
Japan Android Group
http://www.
android-group.jp/

コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

第16回 VR/ARアプリ開発を後押しする空間認識技術Tangoとは

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

三宅 理（みやけ おさむ）
日本Androidの会 運営委員・
日本Androidの会 埼玉支部
合同会社ソニックスタジオ

Tangoとは

Tango（タンゴ）はGoogleが開発した空間認識の技術です。タブレット型の開発者向け端末Tango Developer Kitが販売されたあと、一般向けの「Phab 2 Pro」（Lenovo）が発売されました。

Tangoは「Motion Tracking」「Area Learning」「Depth Perception」の3つの機能から成り立っています。

Motion Trackingは、9軸IMU（Inertial Measurement Unit：3軸電子コンパス、3軸加速度センサー、3軸ジャイロスコープ）と6DOF（Degrees Of Freedom：3次元の直交座標系に沿った動きと3軸各々の周囲の回転）アルゴリズムで取得できるデータを用いて運動情報を解析します。これはデバイスを持って動いたときにどれくらい動いたか検知します。

▼写真1 Phab 2 Pro（左）とZenFone AR（右）

Area Learningは場所を認識します。カメラとMotion Trackingの情報を使い、特徴点を抽出することで空間データとして保存します。同じ場所と認識できる場合、データは最新の環境に合わせて更新されます。これによりある空間のどの部分に端末が存在しているかを判定できるようになります。

Depth Perceptionは奥行きを認識します。これは赤外線ベースのセンサーを使って認識します。床、壁、平面、端面を検知できます。これは写真のような平面な状態を取得するのではなく、立体的な物を立体と認識できるようになります。

このような情報をアプリケーション（以下アプリ）から利用できるのがTangoです。TangoはAndroid上で動作します。

動作機種

Tangoでは特定のハードウェアとAPIを実装したデバイスで動作します。日本国内で購入可能なものとして、レノボより発売中の「Phab 2 Pro[注1]」とASUSより今年夏発売予定の「ZenFone AR[注2]」があります（**写真1**）。どちらもAndroid端末にTangoが搭載されているのでTangoだけではなくAndroidの機能も使えます。

注1）http://www3.lenovo.com/jp/ja/tango/
注2）https://www.asus.com/jp/Phone/ZenFone-AR-ZS571KL/

そのためアプリの開発もAndroidをベースとして開発していくことができます。

エミュレータは現在ありませんので、アプリケーションを開発する場合Tango対応の実機が必要となります。

開発言語

Androidベースということで、Tangoでアプリを開発するにはC/C++、Java、Unity(C#)と、3つの言語が公式でサポートされています。

Unityによる Tango World

今回はTangoの世界を体験するためUnity[注3]でアプリを開発する方法を見ていきたいと思います。

Tangoを利用するうえで必要なものはUnityとTango SDK for Unity[注4]です。Unityはマルチプラットフォームアプリを開発できる開発ツールです。無料で始められ、WindowsとMacで動作します。スマートフォン向けのゲームアプリ開発によく使われています。筆者の環境は、MacBook Pro 13 2016とUnity 5.6.0f3を使ってビルドしました。

注3）http://japan.unity3d.com/
注4）https://developers.google.com/tango/downloads

アプリの動作準備

今回はTanog SDKに含まれるサンプルをビルドして実行してみます。

最初に新しくUnity Projectを作成します。Projectが作成されたらTango SDK for Unityのファイルである「TangoSDK_Gankino_Unity5.unitypackage」をダブルクリックしてプロジェクトにインポートします(**図1**)。

Unityの左下にあるProjectを選択して[Tango SDK]－[Examples]－[Scenes]を選択します。右側にシーンの一覧が表示されるので「Detect TangoCore」を選択します(**図2**)。このシーンは、各シーンを選択するためのメニューとして動作します。

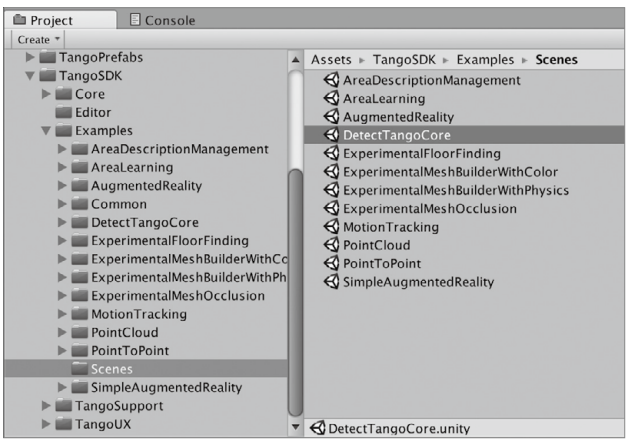
メニューの[File]－[Build Settings]を選択して「Build Settings」ダイアログを表示します。PlatformにはAndroidを選択して「Switch Platform」ボタンを押します(**図3**)。「Player Settings」を押して、PackageNameを入力します。

Projectから**図3**に含まれるシーンをドラッグ&ドロップして登録します。一番上は「Detect TangoCore」になるようにしてください。これで起動時に最初に選択されるシーンになります。

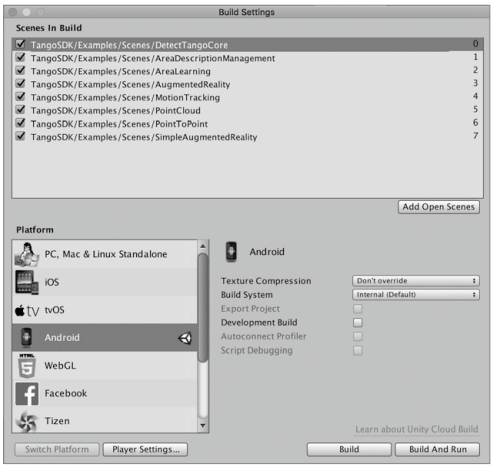
PCにTango端末を接続して「Build And Run」ボタンを押してビルドと実行を行います。ボタ

▼図1　Import画面

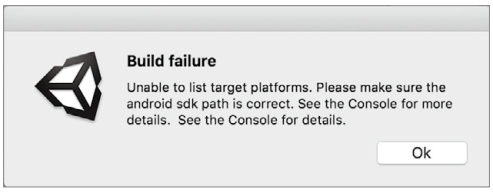
▼図2　シーンの選択



▼図3　Build Settingsダイアログ



▼図4　SDK PATHエラー



ンを押した際に**図4**のようなエラーダイアログが表示されましたか？　開発にAndroid Studio 2.3以上を利用している場合、Unity側が対応できていないのが理由で、別途SDKをインストールする必要があります。Android開発サイト[注5]より「Get just the command line tools」からSDKをダウンロードして解凍、配置したあと、Unityメニューの[Unity]－[Preferences]を選択後、「External Tools」のタブ内にあるAndroid SDKのパスを先ほど配置した場所に変更します。これでビルドができるようになります。

アプリの動作

「Build And Run」ボタンを押してビルドと実行を行うと**図5**のような画面が表示されます。右上のメニューをタップすることでそれぞれのシーンが呼び出されます(**図6**)。

Tangoの今後

ARエンジンであるVuforia[注6]はVuforia Smart TerrainでTangoをサポートすることが発表されました。TangoはVRのようにコンシューマーで普及という流れよりもBtoBで普及していくと思います。専用デバイスを使い自社サービスを提供するということは大いにありそうです。またセンサー部分を活用しておもしろいサービスを作ることもできるかと思います。

日本Androidの会2017年3月定例会では「VR/AR/MRのサービス最前線！」というテーマで定例会を開催しました[注7]。その中でTangoを活用したプロジェクトとして「ミク☆さんぽ[注8]」(KDDI株式会社)やインテリア3Dシミュレーターのtango対応を行った株式会社リビングスタイルがあります。Google I/O 2017では、仮想位置認識サービス(VPS：Virtual Position Service)としてGoogleはTangoを紹介していました。これらにより今後Tangoは普及していくと思います。みなさんもTangoを活用しておもしろいアプリを作ってみませんか？ SD

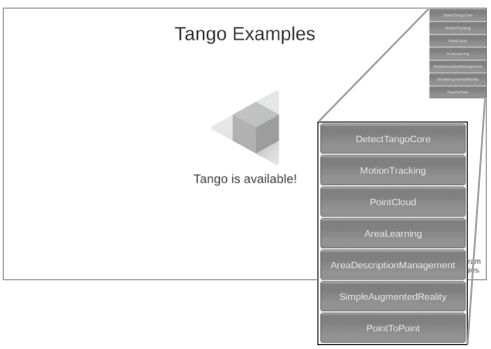
注5）https://developer.android.com/studio/index.html#downloads

注6）https://www.vuforia.com/

注7）https://japan-android-group.connpass.com/event/52967/
Youtube
https://www.youtube.com/watch?v=1lpKl8xpO-A

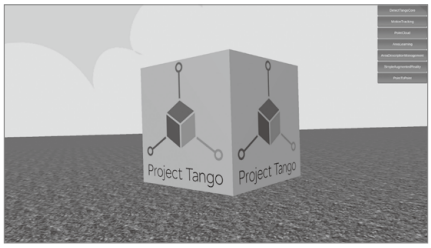
注8）http://news.kddi.com/kddi/corporate/newsrelease/2017/02/07/2278.html

▼図5　DetectTangoCore画面



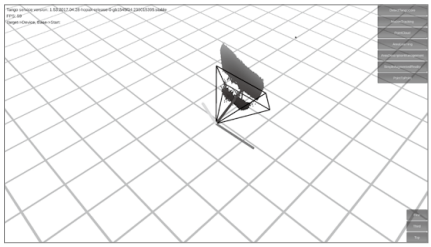
▼図6 DetectTangoCore画面のメニューから行える操作

MotionTracking



仮想空間を表示して、デバイスを動かすと動かした方向にカメラが移動する。箱の見え方が変化

PointCloud

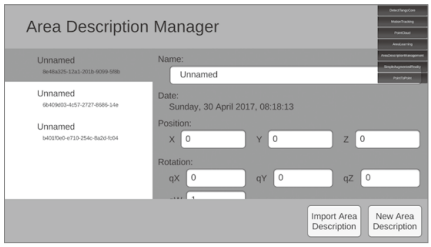


Depth APIを利用してPintCloud情報をワールド座標に表示

AreaLearning

ARモードで画面をタップするとマーカーを立てることができる。ADF(AreaDescriptionFile)を保存すると同じ空間にマーカーを復元する

AreaDescriptionManagement



ADFを作成・編集・削除・エクスポートできる。アプリを作成する場合このシーンは入れておいたほうが良い

SimpleArgumentReality

カメラを使って、現実世界に地球と月の映像を重ねた表示を行う

PointToPoint

タップすることで線が引かれ、線の距離を表示する

COLUMN コミュニティ活動

日本Androidの会(Japan Android Group)ではコミュニティメンバーを募集しています。Androidに興味がある、技術的な質問をしたいといったことがあればぜひ公式サイトから参加してみてください。

また、定例会を月に1回ほど開催しています。詳しい情報はイベントサイト、https://japan-android-group.connpass.com/を参照してみてください。遠方にお住まいの方はYouTubeによる動画も配信しています。

●日本Androidの会ライブチャンネル

https://www.youtube.com/channel/UCEgcvRMuUx49r96YRSrab0Q

一歩進んだ使い方のためのイロハ

Vimの細道

第19回

Vimからデータベースを操作する

mattn
twitter:@mattn_jp

エディタから、テーブルやカラムの確認といったSQLのクエリを手軽に実行できると便利ですよね。今回はそんな要望を叶えるプラグインdbext.vimを紹介。さらにクエリ実行だけでなく、SQLを補完、整形する方法も紹介し、編集面も面もサポートします。

SQLの編集に最適な環境とは？

Webアプリケーションやバッチ処理のシステムを構築する場合、データベースは欠かせないコンポーネントの1つになりました。最近ではORM（Object-relational mapping）を使い、SQLを書かずにプログラミングすることも多くなってきましたが、いまだSQLはエンジニアの必須スキルであることに変わりありません。

みなさんはデータベースを扱うようなプログラミングを行う際、どのようにデータベースを操作していますか。GUIで作られたSQL開発環境を使っていますか？　それともCLIで作られた対話形式のツールを端末から使っていますか？

GUIで操作する場合、グリッド表示されたクエリの実行結果は確かに見栄えも良く、クリップボードにコピーして再利用することもできてとても便利なのですが、単純にクエリを実行したい場合には少し大げさです。また、SSHなどを使ってリモート上で作業する場合には向きません。逆に、CLIで操作する場合には、SQLがシンタックスハイライトされないので可読性が低くなったり、複数行に渡るクエリの編集が難しくなったりする問題があります。

そんな問題を解決するのが、dbext.vimプラグインです。Vimの拡張機能を使ってSQLをシンタックスハイライト表示したり、テーブル名やカラム名を補完しつつ編集、そして実行結果を確認するといったこともできます。

dbext.vimを準備

まずはインストール

残念ながら、dbext.vimプラグインはGitHub上で開発されていません。しかし、http://www.vim.orgに置いてあるいろいろなVimプラグインを、GitHub上でミラーリングしてくれているリポジトリ「vim-scripts」があります。プラグインマネージャvim-plugをお使いであれば次をvimrcに追記し、`:PlugInstall dbext.vim`を実行することでインストールできます。

```
Plug 'vim-scripts/dbext.vim', { 'for': 'sql' }
```

このforは、ファイルタイプがsqlの場合、動的に読み込まれるようにするためのオプションです。dbext.vimは若干古いプラグインで、読み込まれると多くの処理を実行してしまいます。これはSQL以外のファイルを編集する場合であっても有効になってしまいます。そういった場合にこのforを使います。

現在開いているファイルのファイルタイプを調べるには次を実行します。

```
:set filetype?
```

そのほかの有名なプラグインマネージャにも、同様の機能が付いています。

プロファイルの作成

dbext.vimを使用するにはプロファイルが必要です。プロファイルとはデータベースの接続文字列やパラメータといった構成情報です。

dbext.vimがサポートしているRDBMS(Relational DataBase Management System)を示した**表1**の内、db_typeをプロファイルに指定することで、該当のRDBMSを扱うことができるようになります。プロファイルは**リスト1**の方法で記述します。この例では2つのプロファイルを定義しています。g:dbext_default_profile_XXXの形式で複数のプロファイルを記述し(1行目：PostgreSQL、2行目：SQLite)、g:dbext_default_profile(3行目)でデフォルトのプロファイルを指定できます。SQLの編集中などにプロファイルを切り替える場合は、次のコマンドを実行します。

```
:DBSetOption profile=test
```

SQLファイルを単に開いた場合は、デフォルトのプロファイルが使用されます。デフォルトがtestの状態であれば、編集中のSQLは/home/mattn/myDB.dbというSQLiteデータベースファイルに対してSQLが実行されます。現在のプロファイルでどのようなパラメータが設定されているか確認するには次を実行します。

```
:DBGetOption
```

実行モジュールの指定方法

dbext.vimは各RDBMSのCLIコマンドを使ってSQLを実行します。しかし、CLIにパスが通っていない(または通したくない)場合もあります。そこで、各RDBMSのCLIへのパスを別途指定できるようになっています。たとえば、OracleのCLIであるsqlplusへのパスは**リスト2**のように指定します。同様にsqlite3コマンドを指定したい場合は、g:dbext_default_SQLITE_binに設定します。

基本的な操作

まずは接続の確認も兼ねて、適当なクエリを

▼表1　Vimが対応しているRDBMS

db_type	RDBMS
ASA	SAP Sybase SQL Anywhere、IQ
ASE	SAP Sybase Adaptive Server Enterprise
ULTRALITE	SAP Sybase UltraLite
HANA	SAP HANA
ORA	Oracle
RDB	Oracle RDB
SQLSRV	SQL Server
MYSQL	MySQL
PGSQL	PostgreSQL
DB2	DB2
FIREBIRD	Firebird
INGRES	Ingres
INETRBASE	Interbase
SQLITE	SQLite
CRATE	CrateIO

▼リスト1　プロファイルの作成例(PostgreSQLとSQLiteを設定、前者を指定)

```
let g:dbext_default_profile_develop = 'type=PGSQL:user=postgres'
let g:dbext_default_profile_test = 'type=SQLITE:dbname=/home/mattn/myDB.db'
let g:dbext_default_profile = 'develop'
```

▼リスト2　OracleのCLIへのパスを指定

```
let g:dbext_default_ORA_bin = 'c: tools instantclient_12_1 sqlplus.exe'
```

▼リスト3　example.sql

```
CREATE TABLE FOO(ID INTEGER PRIMARY KEY, VALUE TEXT);
INSERT INTO FOO(ID, VALUE) VALUES(1, 'foo');
INSERT INTO FOO(ID, VALUE) VALUES(2, 'bar');
INSERT INTO FOO(ID, VALUE) VALUES(3, 'baz');
```

▼図1　クエリの実行結果

発行してみましょう。example.sqlのように拡張子.sqlのファイルを作成して、**リスト3**のSQL文を記述します。カーソル上のクエリを実行するには、<leader>seをタイプします。<leader>は何も設定していない状態であれば\になっているので、\seをタイプすれば良いです。さきほどの**リスト3**で、1行ずつ実行するか、ビジュアルモードで4行すべて選択してから実行します。そのあと、

```
SELECT * FROM FOO WHERE ID < 3;
```

を実行して結果を確認してみましょう(**図1**)。

パラメータの変更

開いているバッファに対するデータベースパラメータの変更は、<leader>sbpをタイプして実行します。GVimから実行すると、確認ダイアログでのパラメータ入力となります。ログインアカウントの変更なども行えます。

▼表2　Vimが対応しているRDBMS

キー	コマンド	意味
<Leader>sbp	DBPromptForBufferParameters	接続先設定ウィザート
<Leader>se	DBExecSQLUnderCursor	カーソル上のSQLを実行
<Leader>sq	DBExecSQL	引数のSQLを実行
<Leader>slt	DBListTable	テーブル一覧を表示
<Leader>sdt	DBDescribeTable	カーソル上(または引数)のテーブル定義を表示
<Leader>st	DBSelectFromTable	カーソル上(または引数)のテーブルデータ表示
<Leader>stw	DBSelectFromTableWithWhere	カーソル上(または引数)のテーブルデータ表示(WHERE文で条件を付けられる)

キーマッピング

dbext.vimには多くのキーマッピングが用意されています。**表2**はその一部です。そのほかのキーマッピングについては:help dbext-mappingsを参照してください。

SQLの入力補完

Vimにはもともと SQLの入力補完を行う機能が備わっています。ただし、用意されているのは枠組みだけで、追加の拡張を入れることで動作するしくみになっています。dbext.vimもこの拡張インターフェースを提供しており、SQLの入力補完を提供しています。

◆テーブル名の入力補完

たとえば、

```
SELECT * FROM カーソル
```

というSQLでテーブル名を入力補完したい場合は、Ctrl-c t(Ctrlを押しながらcをタイプしたあとt)をタイプします。

◆カラム名の入力補完

dbext.vimはカラム名を入力補完する際、テーブル名を決定させる必要があります。一度、カラム名を補完するのにFOO.までをタイプし、Ctrl-c tで補完を行います。idとvalueが補完候補に現れるので、以降はFOO.なしに補完が可能になります。また、Vimのオムニ

▼リスト4　SQLの整形に関する設定をグループ化

```
augroup MyFileTypeSQL
  " MyFileTypeSQL で設定される内容を破棄
  au!

  " ファイルタイプが SQL の場合は整形のマッピングを追加
  au FileType sql nmap <buffer> <leader>sf <Plug>(sqlfmt)

  " 拡張子が .sql のファイルを書きだすときは SQLFmt コマンドを実行する
  au BufWritePre *.sql SQLFmt
augroup END
```

補完を行うキー Ctrl -x Ctrl -oはデフォルトでテーブル名の入力補完にアサインされています。次のように設定することで、デフォルトでカラム名の入力補完が実行されます。

```
let g:omni_sql_default_compl_type = 'column'
```

SQLの整形

dbext.vimはSQLの実行をサポートするプラグインです。インデントなどはサポートしていません。入力補完と同様、Vimにはインデントの枠組みだけが用意されています。各種RDBMSに依存したプラグインを導入することで、自動インデントが可能になります。pgsql.vim[注1]というプラグインを入れると、PostgreSQLの一部の自動インデントを行えます。しかしこのプラグインが提供するのは、CREATE TABLEやサブクエリの()の入力時に段下げを行う程度です。きれいな整形を行うには、専用の機能を使わなければなりません。

SQLの整形を行いたい場合にはvim-sqlfmt[注2]を導入します。このプラグインはPythonのパッケージsqlparseに付属するsqlformatコマンドを使って、SQLの整形を行います。vim-sqlfmtはカレントバッファの整形を行う:SQLFmtというコマンドと、マッピングを行う際の<Plug>インターフェースのみを提供しています。何かのキーに連動してSQLの整形を行いたいのであれば次のように設定します。

```
nmap <buffer> <leader>sf <Plug>(sqlfmt)
```

また、SQLファイルを保存する際に整形を行うのであれば、次のように設定します。

```
autocmd BufWritePre *.sql SQLFmt
```

これらのように任意の拡張子やファイルタイプに関する設定をvimrcに書く場合は、**リスト4**のようにaugroupを使ってグループ化するのが一般的です。これは、vimrcを再読み込みしても二重にautocmdが登録されないようにするための書き方です。vimrcにすべての設定をまとめることにこだわっていないのであれば、~/.vim/ftplugin/sql/sqlfmt.vimというファイルを作成し、次のように書いておきます。

```
nmap <buffer> <leader>sf <Plug>(sqlfmt)
au BufWritePre *.sql SQLFmt
```

こうすることで、ファイルタイプがsqlに設定された場合のみスクリプトが実行されるので、SQLを編集しない場合に無駄な設定が行われることを回避できます。vimrcをリロードしても、autocmdやmapが二重に登録されることはありません。

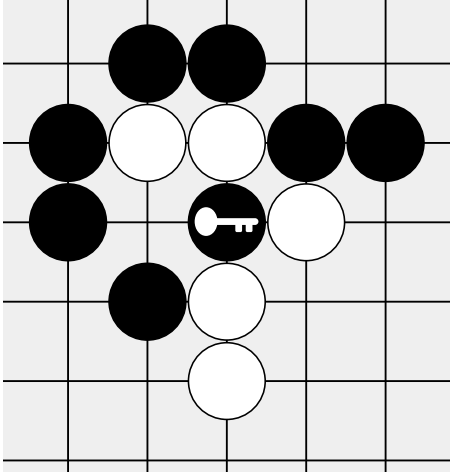
まとめ

今回は、VimからSQLを操作できるdbext.vimを紹介しました。ちょっとしたSQLの開発や修正であれば、これだけでも十分開発できます。ぜひ活用してみてください。SD

注1）URL https://github.com/lifepillar/pgsql.vim
注2）URL https://github.com/mattn/vim-sqlfmt

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

【第四五回】WannaCryの問題の本質

2017年5月12日に感染が拡大したランサムウェア「WannaCry」は、世界的にも大きな注目を集めました。今回は、その本質的な脅威は何であったかを踏み込んで考えてみます。

ランサムウェア WannaCry

2017年5月12日、CNNや英ガーディアン紙は、イギリスの病院のコンピュータがランサムウェアに感染し、病院で患者を受け入れられなくなった、と一斉に報じました[注1]。その報道が引き金となり、世界中にWannaCryの報道が広がっていきます。

WannaCryはPCに感染すると、PCの接続している内部ネットワークを探索し、MS17-010（Microsoft Windows SMBサーバ用のセキュリティ更新プログラム）[注2]のパッチが適用されていないPCがあれば、そこに感染していきます。このパッチが適用されていないと、攻撃者は攻撃コードをMicrosoft Server Message Block 1.0（SMBv1）に送信することで、リモートで任意のコードを実行できます。

つまり、何らかの形でLAN内のPC1台に感染させてしまえば、あとはLAN内にあるMS17-010でアップデートされていないPCに網羅的にWannaCryを感染させられます。正しくは、WannaCryのランサムウェア機能をインストールしていくことができると言うほうが正しいかもしれません。

侵入モジュール（エクスプロイトコード）

近年のマルウェアは複数の機能モジュールから構成されており、機能モジュールを入れ替えるだけで機能を拡張できます。侵入するモジュールと、侵入したあとにその侵入先に本体を導入するためのモジュールは独立しているのが、今日的なマルウェアです。システムにマルウェアを投入するのでドロッパー（Dropper）と呼ばれたり、あるいは内部に侵入したあとに外部からマルウェア本体をインストールするのでインストーラ（Installer）と呼ばれたりしますが、要は防御を突破し攻撃目標に到達する機能部分が必要になります。

脆弱性をついて侵入するタイプにフォーカスして考えてみます。脆弱性は次々に発見され、次々にセキュリティアップデートが用意され、脆弱性が消されていきます。脆弱性を使って侵入するには、その速いローテーションで消費される脆弱性をキャッチアップして、侵入するためのエクスプロイトコード（exploit code）を量産しなければなりません。しかも、かなり専門的な知識が必要です。そこで何が起こるかというと、エクスプロイトコードがレンタルされたり、売買されたりする状況が生まれます。

Multilockerというランサムウェア本体とボットネットを貸し出す形で営業しているタイプのランサムウェアがあります。貸し出す側が20～25%、これを使ってしかける側が75～80%の割合で利益を分配します。ちなみに、Multilockerを運用している側はこの形態をアフィリエイトと呼んでいるそうで

注1）"NHS seeks to recover from global cyber-attack as security concerns resurface"
https://www.theguardian.com/society/2017/may/12/hospitals-across-england-hit-by-large-scale-cyber-attack
注2）マイクロソフトセキュリティ情報 MS17-010 - 緊急 https://technet.microsoft.com/ja-jp/library/security/ms17-010.aspx

す。しかし、このMultilockerもエクスプロイトコードは別売りでほかからレンタルしたり、購入したりして調達しなければなりません。正しく動作するエクスプロイトコードを作成するには、相応の技術力が必要です。また、それ以上に、エクスプロイトコードはセキュリティパッチが現れてそれが広く適用されるまでしか使えません。ほかのモジュールと比較し極めてコストが高くつきます。

兵器級マルウェア Stuxnet

Stuxnetというマルウェアがあります。これは、「イラン核施設の遠心分離機で使われる制御コンピュータに感染し、機器の性能を低下させ、核兵器レベルの濃度のウランを生成させないようにする」という目的のために作られたマルウェアです。

世界で最初の兵器級マルウェアとも呼ばれるStuxnetは、次の3つの脆弱性に対して感染する能力を持っていました。

- MS10-046：Windowsシェルの脆弱性
- MS08-067：Serverサービスの脆弱性
- MS10-061：印刷スプーラーサービスの脆弱性

この3つの脆弱性はいずれもリモートから任意のコードを実行できるものです。そのうちMS10-046とMS10-061は、Stuxnet発見時にはパッチが存在していなかったいわゆるゼロデイ攻撃でした。

誰も知らない脆弱性を複数見つけ、しかも、完全に動作するエクスプロイトコードを作成するのは、並の技術力ではできません。今では、Stuxnetはジョージ・W・ブッシュ大統領政権時に、コンピューター・ウイルス計画OlympicGames（作戦コード名）によりNSA（米国家安全保障局）が作成したというのが定説になっています。

WannaCryがネットワークに蔓延（まんえん）

今回騒ぎになっているWannaCryがどのようにサイト内に侵入したか、その経過は確定しないでしょう。なぜならば、マルウェアがPCに感染する理由は数えるとキリがないからです。「メールに添付されてきたマルウェアをクリックして動作させてしまう」「Webブラウザに入っているプラグインの脆弱性から感染する」「ほかで感染したPCをネットワーク内に持ち込む」。どんな可能性だってあります。

最近ではマルウェアをしかけているサイトにアクセスすると、Google Chromeのフォントがないと偽の警告を出し、フォントに見せかけたマルウェアをダウンロードさせるといったものもあります[注3]。よく次から次へと考えつくものだなと感心しますが、こういうものでも間違えてマルウェアをPCに入れてしまう人もいるでしょう。

WannaCryの最大の特徴（というか致命的な問題）は、感染先のネットワーク上でMS17-010を対象とした高度なエクスプロイトコードを使って、脆弱性が残っているPCにかたっぱしからランサムウェアをしかけていくことです。企業や組織では同じようなシステム構成のPCが複数台導入されているはずです。そのため、同じような脆弱性を持ったPCが何台もネットワークにつながれているということになります。一般事務などの用途でPCが使用されている環境では、LANは特別なコストをかけてネットワーク防御をしているわけでもないでしょう。

とくに（ゼロデイ攻撃も含み）脆弱性に対応していないのであれば、いったんLANに侵入されて攻撃されては、ひとたまりもありません。これがWannaCryの脅威です。しかし、これはWannaCryが生み出した新たな手法というわけではありません。

この攻撃に関しては、前述のStuxnetがまさにモデルケースです。Stuxnetが最初にしかけられたのは隔離されているネットワーク上にあるコンピュータでした。USBメモリを持ち込んで感染させたと言われています。そこから先に挙げた3つの脆弱性をついて内部感染を広げていきました。最後はイランの核開発施設からなんらかの形で外部に流出し、海外でも感染が起こり、そこで初めてStuxnetの存在

注3）"Hack Alert: Chrome Users Urged to Download Missing Font, Install Malware Instead"
http://news.softpedia.com/news/hack-alert-chrome-users-urged-to-download-missing-font-install-malware-instead-513152.shtml

が明らかになりました。

そもそも今回のイギリスの病院のコンピュータは、2014年4月にサポートが終わっているWindows XPを継続して使っていて、ランサムウェアに感染しました。その対策のためにMicrosoft社は急きょWindows XPのためのMS17-010を出したという経緯があります。サポートが終了したシステムを使っているようではひとたまりもありません。

Windows XPに限らずMS17-010のセキュリティアップデートがかかっていないPCはたくさん存在しているでしょう。「内部ネットワークにつながっているだけで、インターネットへのアクセスはしないPCなので、アップデートは必要ない」という考え違いをしている可能性もあり得ます。内部で閉じているネットワークであっても、どこかに侵入を許してしまえば網羅的に脆弱性を狙われます。

NSAが作る兵器級エクスプロイトコード

WannaCryが使っていたMS17-010の脆弱性は、別名EternalBlueと言います。しかも、EternalBlueのPoC(Proof of Concept、概念実証)コードはすでに出回っており、それは非常に完成度が高いものです。もちろんWannaCryの作者が作ったものではありません。GitHub上にもEternalBlueがどのような動作をするのかがわかるコードが公開されていますし、ネットを検索すれば複数のエクスプロイトコードが簡単に見つけられるはずです。

未知の脆弱性(後にMS17-010として知られる)に対して洗練されたエクスプロイト手法を開発したのはNSAです。そして、(NSAもまた)これらの秘密とされていたエクスプロイト手法を外部に流出させてしまうのです。2016年夏に突如として名乗りをあげたハッカーグループThe Shadow Brokers(以下、TSB)が、これらの手法を入手しています。TSBはいくつかのゼロデイ攻撃のエクスプロイトコードも含め、複数の攻撃用コードをNSAから入手し、それをオークションにかけました。

結局このオークションは成立しなかったようですが、本当にTSBがお金のためにオークションをしたのかどうか、筆者は疑っています。

その後、2017年4月14日にEternalBlueを公開しました。そのちょうど1ヵ月前にMS17-010の緊急セキュリティアップデートが発行されているわけですから、一通りセキュリティアップデートが終わったと見通した時点で公開したとも考えられます。

NSAは米国の数ある諜報組織の1つでSIGINT(通信に対する諜報活動)を専門としており、現在ではコンピュータもNSAの活動範囲になっています。NSAがマルウェアを開発する予算は、米国の諜報組織の膨大な予算の中の一部として用意されており、それを元手に開発されています。ネットワーク犯罪者が密かに作ったり、興味があって実力試しで作ってみたりというレベルとは完全に違うことを理解しておかなければなりません。

The Shadow Brokers

今回の件で、TSBがNSAから流出させたいくつかのエクスプロイトコードを手元に置いていることは証明されました。

WannaCryは、EternalBlueでPCにエクスプロイトをしかけたあと、DoublePulsarと呼ばれるこれまたNSAから流出したバックドアモジュールをメモリ上に常駐させます。これはメモリ上で動作し、外部から遠隔制御するための機能を持ちます。バックドアのツールとして任意のファイルを入れたり、外にファイルを流出させたりすることができます。

今回、DoublePulsarは、ランサムウェアの本体を入れることに使われています。任意のコマンドを外部から実行できますから、外部の第三者がPCをコントロールできるのと同じです。

EternalBlueとDoublePulsarを用意し、ターゲットを狙うPCと実際にターゲットとなるPCがあり、攻撃可能な状態であれば、ターゲットとなったPCは内部から情報が流出していることにはいっさい気づけないはずです。また、DoublePulsarはメモリ上にしか存在しないタイプですから電源を落としてしまえば、各種監査ログでも詳細に見ないかぎり、その存在は見つけられないでしょう。監査ログ

に残っているかどうかも疑問です。

こんなシナリオを考えてみましょう。Raspberry Piのような小型のPCを用意します。EternalBlueとDoublePulsarとあと少しの追加プログラムを入れておきます。MS17-010のパッチはまだ存在していないとします。すると、この小型PCを密かに組織のネットワークに接続してしまえば、その組織内のWindows系[注4]のコンピュータから情報を盗んだり、WannaCryが行ったように情報を破壊したりすることが可能です。

兵器級のマルウェアがあといくつあるのか

WannaCryは、TSBが公開したいくつかのNSA作成のエクスプロイトコードを流用したに過ぎません。WannaCryが登場する以前の2017年4月24日時点で、すでにDoublePulsarが全世界の数万から十数万台のコンピュータに感染しているという報告があります[注5]。WannaCryのようなわかりやすい形で感染を表面化させていなかっただけで、問題は水面下で進んでいたのです。

TSBは“TheShadowBrokers Data Dump of the Month”という、入手したNSAの情報をリークする会員サービスを立ち上げるとしています(**図1**)。

しかし筆者は、TSBのモチベーションが金銭だとはどうしても思えません。むしろ、これらのすべては注目を引くのが目的で、このような厄介な兵器レベルのマルウェアを作って外部に流出させてしまうようなNSAに非難の目が向くようにしかけているように見えます。つまり今後も、NSAに対して冷たい視線がいくような形で公開が続く可能性が大きいということです。

セキュリティアップデートのいち早い適用を

NSAが作った極めて危険性の高いエクスプロイトコードが、まだいくつも存在していて、それがどの段階はわからないが、将来公開されるであろうことが予想できる、私たちはそんな状況にいます。これまでとはまったく違う様相を見せています。

TSBのここまでの論理は「すでにアップデートは公開されている」「PoCを公開しているだけで実際のマルウェアを作っているわけではない[注6]」「そもそもこれらのコードを作ったのはNSAであり、自分たちではない」ということだと思います。NSA、ひいては米国政府が世界から非難されれば目的は達成されるのではないかと筆者は考えています。

今回も、TSBが公開する前に緊急セキュリティアップデートが出ているわけですから、これでどうにかできたわけです。もしゼロデイ攻撃でTSBが情報を公開するなら無責任として非難が集中するので、その形は取らないでしょう。

我々が選択すべき最初の防御は、とにかくリリースされるセキュリティアップデートをいち早く適用することです。そうすれば、かなり対抗することができるでしょう。少なくとも、セキュリティアップデートを怠ったり、セキュリティアップデートすらされないサポート切れのシステムを使ったりできる時代ではなくなったことだけは確実です。SD

◆図1　TheShadowBrokersのツイート

このツイートにあるリンクの先にて“TheShadowBrokers Data Dump of the Month”についてアナウンスされている。

注4）具体的には、Windows Vista、7、8.1、RT 8.1、10と、Windows Server 2008、2008 R2、2012、2012 R2、2016と、Server Coreインストールオプションです。詳しくは注2のURLで確認できます。

注5）“NSAから流出のバックドア「DOUBLEPULSAR」、世界で感染急増”　**https://japan.zdnet.com/article/35100240/**

注6）銃規制を嫌うアメリカ人が使う「銃が人を殺すわけではない。人が人を殺す」というレトリックと同じです。

SOURCES

レッドハット系ソフトウェア最新解説

第10回

コンテナを使ってみよう

パブリッククラウドサービスであるOpenShift Onlineが、2017年5月上旬にリリースされました。この使い方を、同時期に開催されたRed Hat Summitで発表したさまざまなコンテナ戦略と合わせて紹介します。

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

さまざまなコンテナ戦略を発表したRed Hat Summit 2017

2017年5月2日から4日にかけて、Red HatはRed Hat Summit(年次で実施されるRed Hat最大のイベントであり、同社のメッセージから製品のロードマップまでさまざまな事柄を発表)を開催しました。このイベントではコンテナ戦略に深く関係するOpenShiftが一番の注目を浴びており、マッコーリー銀行[注1]などの金融機関を中心とした導入利用事例[注2]や、これから紹介する次のような新規戦略を発表しています。

OpenShift.io[注3]

開発者向けに用意されたコンテナベースのアプリケーション開発環境であり、パブリッククラウドサービスとして提供されます。OpenShift.ioはさまざまなオープンソースの組み合わせで成り立っており、OpenShiftのほかにEclipse Che、Eclipse Vert.x、OpenJDK、PCP、WildFly Swarm、Spring Bootといった、サービス分析やアプリケーションの継続的改善／デリバリーを実現するために必要な機能を一通り備えています[注4]。OpenShift.ioはテクノロジープレビューとして公開されていますが、クローズドベータのようなサービスとなっているため、現在空席待ちの状態となっています。そのため、使用イメージを把握したい場合、YouTubeなどで公開されているOpenShift.ioの紹介デモ[注5]をご覧になるか、アップストリームプロジェクトであるfabric8[注6]を使ってみることを推奨します。

OpenShift Application Runtimes

マイクロサービス化されたアプリケーションのランタイムとして、WildFly Swarm、Spring Boot、Eclipse MicroProfile、Eclipse Vert.x、Node.jsの正式なサポートを発表[注7]しました。OpenShift.ioのほかにも、すでに製品としてリリースしているオンプレミス版のOpenShiftとも組み合わせて利用できるようになる予定です。また、既存のJavaアプリケーションについて、JBoss Enterprise Application Platform[注8](JBoss

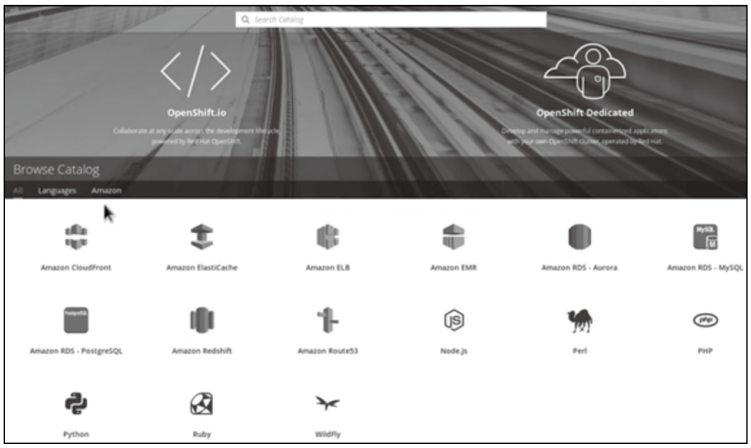
注1) URL https://youtu.be/QnY9TozANr0
注2) URL https://www.openshift.com/container-platform/customers.html
注3) URL https://openshift.io/
注4) URL https://developers.redhat.com/blog/2017/05/02/announcing-red-hat-openshift-io/
注5) URL https://www.youtube.com/watch?v=XNdi4AC_EPQ
注6) URL https://fabric8.io/
注7) URL http://red.ht/2pG5ku2
注8) URL https://www.redhat.com/ja/technologies/jboss-middleware/application-platform

EAP)への移行を促進させる Red Hat Application Migration ToolKit[注9]も利用していくことで、OpenShift環境でのJBoss EAPコンテナを利用したマイクロサービス化を実現できるようになります。

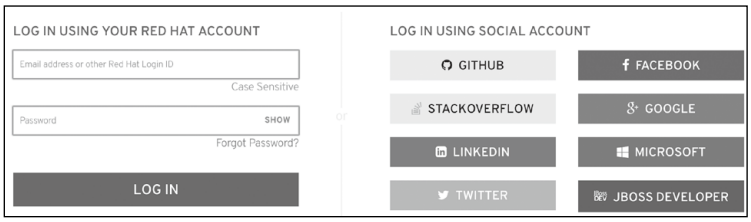
オンプレミスのOpenShiftとAWSとの連携

Amazon AWSとの提携強化が発表され、**図1**のOpenShiftのコンソールから各種AWSサービスをデプロイ、オンプレミスのアプリケーションとの連携が簡単にできるようになります[注10]。これにより、アプリケーションのハイブリッドクラウド対応の促進につながります。この機能は、OpenShiftにOpen Service Broker API[注11]を組み込むことにより実現される予定です。

▼図1　AWSサービスを呼び出すOpenShiftコンソールの画面イメージ



▼図2　OpenShift Onlineで利用できる各種アカウント



OpenShift Online

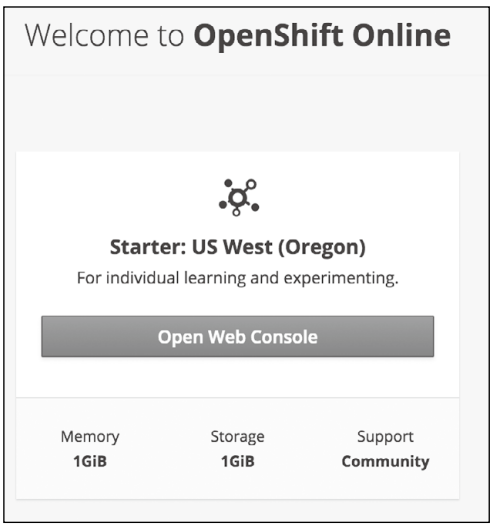
OpenShift OnlineはOpenShift.ioのベースとなっているパブリッククラウドサービスです。Red Hat Summit 2017の開催とほぼ同時期に、正式なサービスとしてリリースされました。執筆時点では無償のFree Planのみ利用できるようになっており、Dockerコンテナをベースとしたアプリケーションの開発を気軽に試せるようになっています。過去記事で紹介しましたオンプレミス版のOpenShift Container Platformと比較すると、OpenShift環境の管理権限が必要となる各種操作ができない、使えるリソース(メモリ／ストレージなど)が限られている、JBoss EAPコンテナを利用するためのテンプレートがないなどの制限がありますが、個人で試してみるには十分な機能を備えています。

それでは、さっそく利用方法を見ていきましょう。最初にhttps://www.openshift.com/にアクセスし、[SIGN UP FOR FREE]をクリックしてアカウントを用意します。OpenShift Onlineのアカウントには、レッドハットのカスタマーポータル[注12]のアカウントや各種ソーシャルアカウント(**図2**)を利用できます。

ログインするアカウントを選択したあとに、プラン(FREE1択)とリージョン(US EASTかUS WESTの2択)を指定して、アカウントにOpenShift Onlineの環境を紐付けます。執筆時点ではほぼ選択肢がないため、何も考えずにクリックしていくだけの作業となります。環境の紐付けが完了すると、Welcome画面(**図3**)が表示されるので、[Open Web Console]ボタンをクリッ

注9) URL https://access.redhat.com/documentation/en/red-hat-application-migration-toolkit/
注10) URL https://www.youtube.com/watch?v=EKo3khfmhi8
注11) URL https://www.openservicebrokerapi.org/
注12) URL https://access.redhat.com/

▼図3　OpenShift OnlineのWelcome画面



クすればOpenShift Onlineのコンソールにアクセスできます。

コンソールにアクセスしたあとは、前述したように機能がいくつか制限されていますが、オンプレミス版のOpenShiftと同じように使うことができます。この使い方については、2016年10月号と2016年11月号の同連載記事やRed Hatの公式ドキュメント[注13]をご参照ください。なお、Webコンソールだけでなくコマンドベースでも操作ができるようになっています。その場合は、OpenShiftのCLIツール（ocコマンド）をインストール[注14]して、OpenShift Onlineの環境にログインします。ログイン後は、ocコマンドを利用してアプリケーションの状況確認やデプロイなどを実施できるようになります。

```
$ oc login https://console.starter-us-west-2.openshift.com
```

このようにOpenShift Onlineはたいへん簡単に利用できるため、興味がありましたらぜひとも試してみてください。SD

注13）URL https://access.redhat.com/documentation/en/openshift-online/
注14）URL https://access.redhat.com/documentation/en-us/openshift_online/3/html/cli_reference/cli-reference-get-started-cli#installing-the-cli

Unityの生涯

Ubuntu Japanese Team　あわしろいくや

今回は主役の座を失うことになってしまったUnityの航路をたどっていくことにします。

Unityとは何だったのか

Unityは、その前身であるnetbook-launcherやその後継となるはずだったUnity 8、あるいは数多のデスクトップ環境やデスクトップシェルと比較しても、変化の激しさと機能の多さは他の追随を許しません。これはUnityに求められることが変わっていったことの証左でしょう。最初のうちはネットブックの狭い解像度でも、通常のデスクトップの解像度でも違和感なく使えること、続いては対応デバイスの増加とWeb技術との統合、最後はUnity 8へスムーズにバトンタッチすべくひたすらバグ修正や使い勝手の向上やパフォーマンス問題への対応など、とにかく問題を起こさないことが重要視されたと考えられます。

というわけで、今回はUnityの多機能ぶりと歩みを振り返っていくことにします。

Unity 7の機能

Dash

画面左上のUbuntuのロゴアイコン（Dashアイコン）をクリックすると表示されるのがDash（ダッシュ）です。当初はアプリケーションの起動のみだったのが、Lensというしくみが導入されてPCの内外を問わずさまざまな項目が表示できるようになりました。とはいえ、最近のUnityでは外部（インターネット）を検索するしくみはデフォルトでオフになっています。LensにはさらにScopeというしくみがあり、検索の対象を広げることができます。

後年はまったく活用されなくなりましたが、ほかにもアプリケーションのインストールや支払い（購入）やさまざまなことがこのDashから行うことができます。

Launcher

画面左端に表示されるのがLauncher（ランチャー）です。よく使うアプリケーションはDashを経由せず起動できたり、現在起動しているアプリケーションやさまざまなウィンドウを管理したり、ショートカットの追加や削除、アイコンを右クリックした際に表示されるクイックリストなど、かなりたくさんの機能があります。Launcherの設定は「設定」-「外観」である程度変更できます。

Indicator

画面上部に表示されているパネルの右側にあるのがIndicator（インジケーター）で、向かって左側に表示されるApplication Indicatorsと右側に表示されるSystem Indicatorに分かれています。前者の仕様は公開されており[注1]、たくさんのアプリケーションで対応しているほか、KDE Plasmaでもほぼ同じ仕様で対応しています。Unityが使われなくなっても、この

注1）https://unity.ubuntu.com/projects/appindicators/

仕様は残ることでしょう。

HUD

Heads-Up Displayの略で、アクティブなウィンドウでAltキーを押すと表示されるショートカットメニューです。たとえばAltキーを押したあとcキーを押すと、ウィンドウを閉じることができます。キーボードから直接文字を入力できる言語圏では便利でしょうが、日本語では便利とはいい難いです。

Global MenuとLocal Integrated Menu

Unityではデフォルトで上部のパネルに表示しているアプリケーション(ウィンドウ)のメニューが表示されます。これをGlobal Menuといいます。また、個々のアプリケーションのタイトルバーにこのメニューを表示することもできます。これをLocal Integrated Menu(LIM)といいます。どちらにするかは「システム設定」-「外観」-「挙動」タブで変更できます。

オンラインアカウント

「システム設定」-「オンラインアカウント」を開き、デフォルトではFacebook、Flickr、Googleの各種Webサービスのアカウントを有効にして、Unityやそのほかのアプリケーションから便利に使えるようにするための機能です。GNOMEにも似たような機能がありますが、実装はまったく異なるようです。

WebApp

Webサービスを、さもアプリケーションであるかのように扱う機能でしたが、最近はまったく使用されていません。たしかに多様なプラットフォームに対応したアプリケーションを用意するのはたいへんですので、このような機能が必要だったのでしょう。しかし、Unity 7はほぼPC専用と言える状況になったので、活用されなくなったものと思われます。

セキュリティとプライバシー

「システム設定」-「セキュリティとプライバシー」で各種の設定が行えます。Unityはセキュリティについて話題になったことがあり、後年にはどんどん厳しくなっていき、最終的にはオンラインでの検索はデフォルトで無効になりました。

Unity Tweak Tool

unity-tweak-toolパッケージをインストールし、起動するとUnityの細かな設定を安全に変更できます。Launcherを横表示にする設定もこれを使うと簡単です。

Low Graphics Mode

具体的な方法は紹介しませんが、Unityにはグラフィックスが遅い環境用のLow Graphics Modeがあります。各種エフェクトを省略して相対的に高速化しています。これは16.04以降に実装されましたが、設定はやや難しいです。しかし17.10からはこれが簡単になるということです。17.04に間に合わなかったのが残念です。

UnityとUbuntuの歴史

netbook-launcher

Unityの前身はnetbook-launcherといい、その名のとおり今となっては懐かしいNetbook用のランチャーでした。デスクトップ用とNetbook用のランチャー(デスクトップシェル)を統一したものですのでUnityという名称であり、またデスクトップ用とスマートデバイス向けのランチャーを統一しようとしたのもUnity(厳密にはUnity 8)でした。そういう意味ではポリシーは一貫しているといえます。

Netbook用のUbuntuが最初にリリースされたのは、インストール用イメージではなくプリインストール用でした。DELLやASUSなどから発売されたようですが、日本でも販売されており、筆者はDELL Inspiron Mini 9を所有していました。残念ながら現在は電源が入らなくなってしまいましたが、**図1**がデスクトップのスクリーンショットです。

Ubuntu Netbook Remixとしてインストールイメージが公開されたのが9.04、Ubuntu Netbook

Editionと名称が変わって公式フレーバーになったのが10.04（**図2**）、最後のリリースとなったのが10.10と、Ubuntuの歴史を駆け抜けていきました。

Unity

最初に注意ですが、ここで登場するバージョンはすべて最終バージョン（サポート終了時点、あるいは執筆段階での最新版）です。Unityはアグレッシブにアップデートされているので、リリース時点とサポート終了時点でバージョンの乖離（かいり）が激しいものも存在しました。

Unityが最初に採用されたUbuntu Netbook Editionは10.10（**図3**）です。すなわち10.04まではnetbook-launcherが採用されていました。そして11.04からはUbuntu Netbook EditionとUbuntuが統一されました（**図4**）。このあたりの流れは現在のUbuntuとUbuntu GNOMEの関係とよく似ています。

Ubuntu 11.04にはUnityのほか、ハードウェアアクセラレーションが有効にならないときの環境用にUnity 2Dも選択可能になっていました。Unity 2DはQtとQMLで書かれ、Unityと似たような使い勝手にするよう開発されていました。

Ubuntu Netbook Edition 10.10のUnityのバージョンは0.2.46です。このころはまだGNOME Shellと同じウィンドウマネージャであるMutterを採用していました。機能もシンプルで、おおむねアプリケーションランチャーとしての機能しかありませんでした。

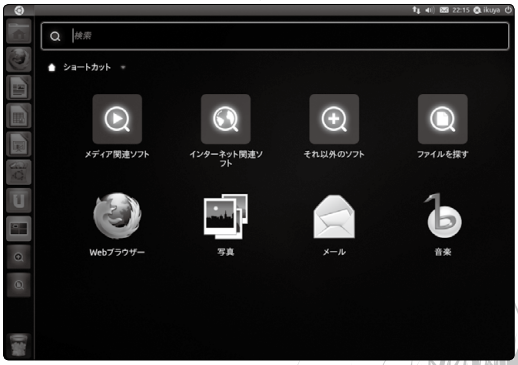
Ubuntu 11.04のUnityのバージョンは3.8.16、Unity 2Dのバージョンは3.8.4.1でした。Unityはウィンドウマネージャを Compizに変更した最初のバージョンです。バージョニングは0.2から3.8に上がっていて驚きますが、全面的に書き換えるにあたって3から始めたものと思われます。changelogに残る最後の0.2系列が0.2.46、最初の3系列が3.1.2ですので、単純に頭の小数点を取ったのでしょう。Unity 2Dのバージョンは3.2.0から始まっており、

図1　Dell Inspiron Mini 9のデスクトップ。Ubuntuのバージョンは8.04だった

図2　Ubuntu 10.04 Netbook Editionのスクリーンショット。LTSではなかった

図3　Ubuntu 10.10 Netbook EditionからUnityに移行した

図4　Ubuntu 11.04のUnity



これはUnityに合わせたとのことです。

Ubuntu 11.10のUnityのバージョンは4.30、Unity 2Dのバージョンは4.12で、Lensとインジケーターが実装されてだいぶ現在のUnityに近づいてきました(**図5**)。LauncherとパネルとインジケーターがGTK+ 3にポーティングされたのと、Dashが日本語入力に対応したのも大きな変更点です。LTS前だからか、とにかくアグレッシブに開発されており、11.10のリリース時点では4.22だったのが、最終的には4.30までアップデートされました。

Ubuntu 12.04 LTSはついこの4月にサポートが切れたばかりですので[注2]、最近までお使いだった方もいらっしゃるのではないでしょうか。Unityのバージョンは5.20、Unity 2Dのバージョンは5.14でした。もちろんUnityがデフォルトになって初めてのLTSでした。HUDが実装され、「システム設定」-「外観」-「挙動」タブでいくつかの設定が変更できるようになりました。

Ubuntu 12.10のUnityのバージョンは6.10で、ここまで来るとUnity 7と見た目に関する大きな違いはなくなってきます。Lensの機能が強化され、さまざまなことが検索できるようになりました。またWebAppsも実装されました。「システム設定」に「プライバシー」が追加され、Unityに関するプライバシーの設定が細かく変更できるようになりました。Unity 2Dのサポートが削除され、Unityに一本化されたのも大きな変更です。

Ubuntu 13.04でついにUnity 7.0に到達します。ここから4年間このバージョンが続くのは、おそらく誰も予想していなかったでしょう。7.0にアップデートしたとはいえ6.xと機能的には大きな違いはなく、パフォーマンスチューニングや使い勝手の向上がおもに行われました。デフォルトでワークスペースが1つになったり、タイプミスをそれっぽいものに修正してくれるようになったなどの挙動の変更もあります。

Ubuntu 13.10のUnityのバージョンは7.1.2です。このころからUnity 8の開発が始まり、Unityの開発速度が低下していきますが、バグフィックスなどは確実に行われています。

Ubuntu 14.04 LTSのUnityのバージョンは7.2.6です。High-DPIディスプレイに対応したり、グローバルメニューをアプリケーションのタイトルバーに表示できるようにしたなどの変更点があります。また、gnome-settings-daemonとgnome-control-centerをそれぞれunity-settings-daemonとunity-control-centerへとフォークし、独自の変更を加えやすくなりました。

Ubuntu 14.10のUnityのバージョンは7.3.1です。さらなるHigh-DPIディスプレイへの対応などが行われています。

Ubuntu 15.04のUnityのバージョンは7.3.2、15.10のUnityのバージョンは7.3.3、16.04のUnityのバージョンは7.4.0、16.10と17.04のUnityのバージョンは7.5.0です。いずれも細かな修正が行われています。16.04以降のUnityではLow Graphics Modeオプションの追加、Launcherを下中央に移動するオプションの追加が行われています。

注2) Canonicalからサポートサービスを購入すれば、あと2年延長できます。

図5 Ubuntu 11.10のUnity。ようやくDashにLensが実装された

Unity 2D

前述のとおりUnityのデフォルト化に伴って11.04から12.04まで採用されていたUnity 2Dですが、消滅した理由は担当社員の退社と記憶しています。奇しくもUnity 8と同じくQtとQMLで書かれていたので(ただしこちらのQtは4で、Unity 8のQtは5です)、そのまま継続していればどうなったのか歴史の

ifを考えざるを得ません。

Unity 8

最初はqml-phone-shellという名称で2012年8月から開発が開始され、2013年6月からUnity 8という名称に変更されました。qml-phone-shellの最終バージョンは1.80、Unity 8の最初のバージョンは7.80.0です。たしかにUbuntu 13.10にはunity8というパッケージが存在します。

計画はWikiにまとめられており[注3]、現在でも読むことができます。最大の目標は今さら言うまでもなく1つの実装ですべてのハードウェアに対応させることです。そして移行のスケジュールは14.10以降となっていました。ディスプレイサーバとしては当初からMirのみの対応だったようです。このWikiには書かれていませんが、initデーモンはUpstartのみの対応で、systemdに移行したものの結局全部の移行はできませんでした。すなわち、Unity 8はUpstartとMirに強く依存していたということです。

プレビューのままで終わってしまった17.04のUnity 8を見ても(**図6**)、ユーザインターフェースとしては実にいいところまで来ていたので、なんとも惜しいです。

Unityの今後

2017年5月下旬現在ではUbuntu 17.10の開発が進んでいます。今のところはまだデスクトップシェルはUnity 7のままです。以後GNOME Shellに移行するものの、アップグレードした場合はUnity 7も引き続き使用できるようになる予定です。とはいえUnity 7のためにあたっていた各種パッチは落とされると思われるので、どの程度同じ使い勝手が維持されるかは不透明です。

Unity 8セッションはすでにデフォルトではインストールされなくなり、Upstartに依存していた部分が削除されたり、GTK+のMirバックエンドが有効にされなくなったりなど、現段階ですでに動作しなくなっています。今後はフォーク先に期待するのみになりました。

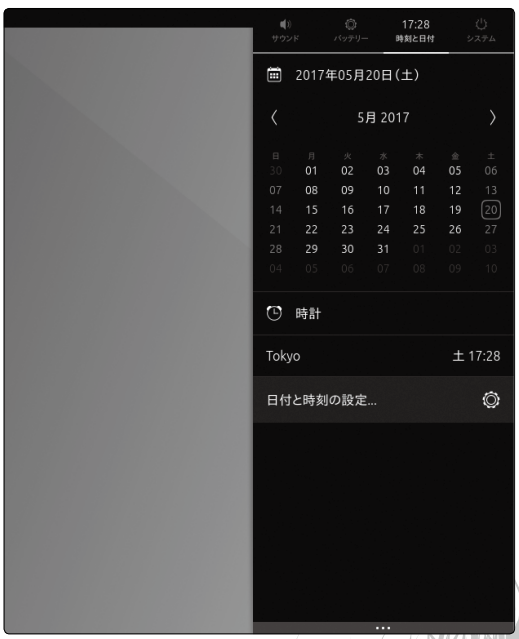
ポイントオブノーリターン

蛇足ですがUnity 8に全面的に移行する未来が訪れるためには何が必要だったのかを考えてみることにしましょう。

まずはディスプレイマネージャをMirとX Window System(X11)の両対応にしておくことです。ひとまずUnity 8に移行しておいて、デスクトップ版のみあとからMir(あるいはWayland)に移行する、という手が使えていれば、移行は叶っていたのではないかと思えてしまいます。事実14.04まではunity8-desktop-session-x11というセッションが存在していました。14.10の段階はもちろん、現在であってもX Window Systemなしは現実的ではありません。

あとはsystemdへの移行も、結局Unity 8では達成できなかったことを考えると手間ばかりかかって効果的ではなかったといえます。とはいえこれがあったからGNOMEへの移行も比較的容易にできるので、未来を予測することはいかに難しいかがよくわかります。SD

図6　Unity 8は右側のメニューが特徴



注3) https://wiki.ubuntu.com/UnityNextSpec

48

Debian Developer　やまねひでき　henrich@debian.org

Debian 9“Stretch”の新しい点

Debian Hot Topics

Debian 9はどこが新しくなったの?

ここしばらくの間、続けて状況をお伝えしているDebian 9“Stretch”ですが、本稿執筆中に、「6月17日にリリース予定」というアナウンスが出ました[注1]。本誌を読んでいらっしゃる方は、すでにインストール可能な状態となっていることでしょう。

今回は、Debian 9“Stretch”のリリースについて、概要をまとめてお伝えします(さらに詳しい内容を知りたい場合はリリースノート[注2]をご参照ください)。

Debian 9のサポートアーキテクチャ

Debian 8からは、64ビットリトルエンディアンMIPS(mips64el)アーキテクチャが追加されました。

逆にサポートが削除された機器として、Pentium(PentiumIIより前です)とその互換CPU[注3]が載っているPCや、PowerPCアーキテクチャの古めのMacなどがあります。これらの機器では、残念ながらDebian 9“Stretch”は利用できません。このような機器を使う場合は、前のリリースであるDebian 8“Jessie”をお使いください(Debian 8では、i386アーキテクチャはLTSサポートも含めて2020年まで利用できます)。

インストーラ／カーネル

まず、インストーラはGUI(Graphical User Interface)がデフォルトになりました。そのため、これからDebianを触る人も、実機へのインストールが多少とっつきやすくなったのではないでしょうか。

ただ、期待されていたインストーラのUEFIセキュアブートへの対応は、9.0リリース時点では間に合わないかもしれません(間に合わなかった場合は、9.xのポイントリリースで対応が追加される予定です)。

Linuxカーネルは4.9をベースとしたものになっています。Linux 4.9はLTS(Long Term Support)バージョンであり、メンテナンスをするうえでDebian単体でがんばるのではなく、ほかの開発者の力も借りられるので順当な選択でしょう。

最近出たAMDのCPU「Ryzen」への最適化などは難しいですが[注4]、周辺機器のサポートについては、新しいデバイスドライバがポイントリリースで随時更新／追加されていくことになります。

注1) URL http://deb.li/3zGE0

注2) URL http://deb.li/33dxA

注3) どのぐらい古いかというと、Pentiumの発売開始が1993年(20年以上前)です。このような環境を今でも使っているのは、かなり珍しいでしょう。ここしばらくで使われているものとしては、Intelのシングルボードコンピュータ「Intel Galileo」ぐらいでしょうか。

注4) このような場合は、stretch-backportsリポジトリに追加されるであろう、新しいバージョンのカーネルパッケージを使うのがお勧めです。

プログラミング言語

プログラミング言語についても順当にアップデートが実施されています(**表1**)。

後方互換性を非常に大事にしているPerlは、とくに変わったところもなく利用できます。しかし、カレントワーキングディレクトリ(.)が@INC(デフォルトでインクルードするディレクトリの一覧)から削除されているので、自作スクリプトについては動作の確認を行いましょう[注5]。

PHPは7.0にバージョンアップすることで、大きく性能が向上しているのは見逃せません。

RubyとPythonについては、最新のメジャーバージョンはDebianのフリーズとスケジュールが合っていないため、1つ前のものになります。Rubyを使った開発などでどうしても新しいバージョンのRubyを使いたいという場合、rbenv(パッケージとしてはrbenvとruby-build)を使うことで対応できます(**図1**)。

一方、残念ながらupstreamと大きくバージョンが乖離(かいり)しているのが、Node.jsです。セキュリティサポートも行われない予定なので、Node環境を使いたい場合はNode.jsのサイト[注6]からダウンロードして利用することになるでしょう。

C/C++コンパイラとしては、GCC 6.3とclang 3.8/3.9が採用されています。C++ 14がデフォルトとなって、C++ 17の実験的サポートが部分的に追加されているのは、うれしい方も多いことでしょう。

▼表1　おもな言語と採用バージョン

言語	バージョン
Perl	5.24.1
PHP	7.0
Python	3.5.3/2.7.13
Ruby	2.3.3
Java	8
Go	1.7
Node.js	4.8.2

▼図1　rbenvを使って複数バージョンのRubyをインストールする

```
必要なパッケージを入れて初期設定する
$ sudo apt install rbenv ruby-build
$ echo 'eval "$(rbenv init -)"' >> ~/.bashrc && . ↵
~/.bashrc

rbenvを利用して、Debian 9ではパッケージされていないRuby2.4.0を入れる
$ rbenv install 2.4.0
Downloading ruby-2.4.0.tar.bz2...
-> https://cache.ruby-lang.org/pub/ruby/2.4/ruby- ↵
2.4.0.tar.bz2
Installing ruby-2.4.0...
Installed ruby-2.4.0 to /home/testuser/.rbenv/ ↵
versions/2.4.0

一方、古いバージョンのRubyをインストールしようとするとエラーになる ※
$ rbenv install 2.2.6
Downloading ruby-2.2.6.tar.bz2...
-> https://cache.ruby-lang.org/pub/ruby/2.2/ruby- ↵
2.2.6.tar.bz2
Installing ruby-2.2.6...

BUILD FAILED (Debian 9.0 using ruby-build 20160913)
(..略..)
        from ./tool/rbinstall.rb:754:in `each'
        from ./tool/rbinstall.rb:754:in `<main>'
uncommon.mk:246: ターゲット 'do-install-all' のレシピ↵
で失敗しました
make: *** [do-install-all] エラー 1

この場合はlibssl1.0-devをインストールしてからビルドする
$ sudo apt install libssl1.0-dev
(..略..)
以下のパッケージは「削除」されます:
  libssl-dev
以下のパッケージが新たにインストールされます:
  libssl1.0-dev
(..略..)

$ rbenv install 2.2.6
Downloading ruby-2.2.6.tar.bz2...
-> https://cache.ruby-lang.org/pub/ruby/2.2/ruby- ↵
2.2.6.tar.bz2
Installing ruby-2.2.6...
Installed ruby-2.2.6 to /home/testuser/.rbenv/ ↵
versions/2.2.6
```

※　古いバージョンのRubyを使いたい場合、Ruby 2.4.0よりも古いバージョンは新しいOpenSSLに対応しておらず、そのままではビルドできません(Debian 9で提供しているデフォルトのOpenSSLのバージョンは新しめの1.1.0であるため)。

注5)　URL https://www.debian.org/releases/stretch/amd64/release-notes/ch-information.ja.html#perl
注6)　URL https://nodejs.org/ja/download/

サーバ関連

サーバ周りに目を移します(**表2**)。

DBサーバとしてMySQLは非常にメジャーなものですが、Debian 9"Stretch"ではMySQLは含まれず、代わりに派生DBの「MariaDB 10.1」が採用されています。これはMySQLがOracle社の傘下になってからクローズドな姿勢を強めており、「セキュリティ問題に対して個別の詳細な情報を公開しなくなった」、「リグレッションテストを提供しなくなった」、「バグデータベースが非公開になった」などの要因によりディストリビューション側でメンテナンスがしづらくなってきたからです(すでにFedora、openSUSE、Red Hat Enterprise Linux とRHELクローンのCentOSなどの主要なLinuxディストリビューションは、MariaDBを標準としています)。

これまでMySQLが利用されてきた多くの場面では、MariaDBでも問題なく動作するはずですが、どうしてもMariaDBではなくMySQLを使いたいという場合は、次の2択になります。

- Oracleが配布するMySQLパッケージを使う[注7]
- Debian unstableからパッケージを持ってきて利用する

詳細な情報については、「https://wiki.debian.org/Teams/MySQL」を参照してください。

そして、最近注目のコンテナ環境であるDockerですが、残念ながらdocker.ioパッケージが複数のRCバグのためにStretchから削除されてしまっているという問題があり、そのままでは利用できません。後日、stretch-backportsリポジトリに登録されることを期待しましょう。すぐに使いたいという場合には、Dockerのサイト[注8]を参考にリポジトリを追加のうえで、「docker-ce」パッケージを導入してください。

▼表2　おもなサーバ環境と採用バージョン

サーバ環境	バージョン
Apache	2.4
Nginx ※1	1.10
PostgreSQL ※2	9.6
MariaDB	10.1
MySQL	外部リポジトリ
Docker	外部リポジトリ
Samba	4.5

※1　若干古めなので、新しいものを使いたい場合は「http://nginx.org/en/linux_packages.html」を参照。
※2　PostgreSQLの開発中バージョンや過去のバージョンをStretchで使いたい場合は、外部リポジトリがあります。「http://wiki.postgresql.org/wiki/Apt」を参照。

注7）URL https://dev.mysql.com/downloads/repo/apt/

デスクトップ／ブラウザ／エディタ

デスクトップ環境も順当なバージョンアップをしています(**表3**)。大きく使い勝手が変わったり見栄えが違ったりすることはありませんが、たいていのものが以前よりもスムーズで使いやすくなっているはずです。

ブラウザでは、FirefoxはESR(Extended Support Release、延長サポート版)である45が採用されています[注9]。一方、Chromiumは ESRのようなものはないので、可能な限り最新のバージョンが入ることになりそうです。

本誌でもよく特集が組まれるエディタについては、Vimは8.0が、Emacsは24と25の2つのバージョンが利用できます。

その他

全体的な話としては、バイナリの多くがASLR(Address Space Layout Randomization)が

▼表3　おもなデスクトップ環境と採用バージョン

デスクトップ環境	バージョン
GNOME	3.22
Cinnamon	3.2
MATE	1.16
Xfce	4.12
KDE	5.8

注8）URL https://store.docker.com/editions/community/docker-ce-server-debian
注9）ESRのバージョンに合わせますので、しばらくするとFirefox 52に変わります。
URL https://www.mozilla.jp/business/

有効にされてコンパイルされており、バッファオーバーフローなどの攻撃に対して被害を抑えられるようになっています。

これらのパッケージを管理するツールAPTは、かなり速度が向上しており容易に体感できるほどです。そして、「apt-getとapt-cacheの2つの使い分けが面倒」という声からか、この2つの機能を合わせた「apt」コマンドが実装されています(使い方はこれまでのapt-get/apt-cacheとほぼ一緒です)。一点、APTまわりで注意が必要なのは「pin」機能の振る舞いが変わっていることです。pinを活用している人はmanページをよく確認してください。

また、ファイルがどのパッケージに含まれているのかを検索するのに使う「apt-file」というツールがあるのですが、このapt-fileのインデックスがAPTと統合され、apt-file updateを実行しなくても済むようになって地味に使い勝手が良くなったのが、筆者としては気に入っています(**図2**)。

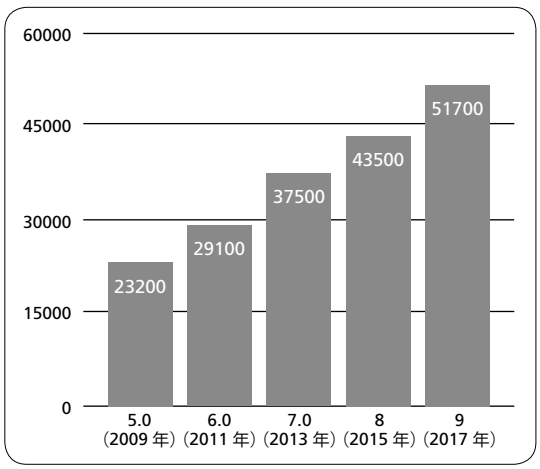
その他、数々のソフトウェアがパッケージとなりDebian 9に含まれた結果、バイナリパッケージの個数は約51,700個と膨大なものになっています(**図3**)[注10]。

筆者個人としては、**表4**、**図4**のようなフォント関連パッケージを新たに追加していますので、読者のみなさんもぜひインストールして使ってみてください[注11]。SD

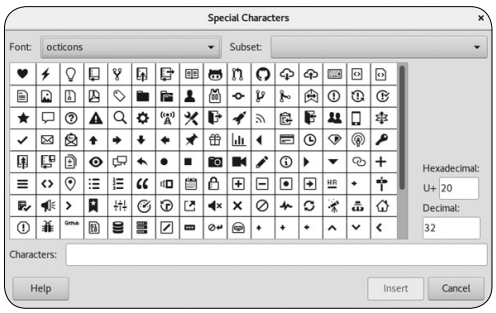
▼図2　apt-fileを使って必要なヘッダファイルを検索する

```
$ sudo apt install apt-file
（..略..）

ソフトウェアをビルドする際に
ヘッダファイルgconv.hが足りないため、
どのパッケージに入っているのかを探す場合
$ apt-file search /usr/include/gconv.h
libc6-dev: /usr/include/gconv.h
```

▼図3　Debianの各バージョンに含まれるバイナリパッケージ数比較



▼図4　fonts-octiconsで利用可能な文字の例（LibreOffice Impressで表示）



▼表4　筆者がDebian 9で追加したフォント関連パッケージ

パッケージ	説明
fonts-seto	手書き風「瀬戸フォント」
fonts-ricty-diminished	プログラミングなどに向く「Ricty Diminished」
fonts-oradano-mincho-gsrr	レトロな「Oradano明朝GSRR」
fonts-octicons	GitHubの絵文字フォント「Octicons」。(図4を参照)
birdfont	フォント作成ツール「birdfont」

注10) 逆に、Debian 9から削除されたパッケージ数は6,700を超えます。
注11) 逆に、「もう役目を終えただろう」ということで削除したものとしては東風フォント(ttf-kochi)があります。

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

sshの基本的な使い方を紹介します。

第15回 ssh(その1)

安全なネットワーク接続に欠かせないコマンド

大型コンピュータに端末を接続していたホスト集約型の環境から、Unixワークステーションの普及によって、リソースの分散が可能になりました。ネットワーク通信は、サービスを提供するプログラムとサービスを受けるプログラムと役割を分けたクライアント・サーバモデルを実現します。ファイルシステムの分散にNFS、ローカルでできていることをリモートシステムから実行できるプログラム、GUIでも表示とサービスプログラムを分離したウィンドウシステムが開発されます。異なるソフトウェアやハードウェアの開発者が、お互いに接続できるように協力しあい、標準となるプロトコルが定められました。

ssh——Secure SHell

sshは、フィンランドのTatu Ylonen氏が1995年に開発しました。それまでの、平文で送るネットワーク接続コマンドに対して、暗号化した通信により安全を確保できる画期的な発明です。sshは、Secure SHellという名称ですが、本連載でも馴染みのあるbashやcshなどのOSと利用者の間に立つ一般のシェルとは異なります。ネットワークを介して、別のシステムと接続する際に使うプロトコルであり、接続のためのサービス・デーモン(sshd)とクライアントプログラム(ssh)です。またSSHプロトコルを用いたクライアントプログラム(scp)や暗号鍵生成や、鍵を管理するためのツール群があります。

探検隊では、Unixでの伝統的な通信ソフトウェアである、ftp、telnet、BSD rコマンド[注1]よりも先に、現代では不可欠な技術であるsshを取り上げます。それまで通信プログラムの主流だったrコマンドは、ホストベース(リモートホスト名もしくは、IPアドレス)で認証をしますので、IPアドレスを詐称(IPスプーフィング)されても、利用者は気づかず被害を受けることがあります。sshには、こうした問題を解決するしくみがあります。

sshは、その成り立ちと発展の過程からいくつかのバージョンがあります。元祖sshは、後に商用となりました。OpenBSDを開発するチームが、ライセンスがフリーだったころのsshを改良してOpenSSHを作り、近年のLinuxやmacOSのほとんどで使われています。

また、SSHプロトコルにもSSH1とSSH2とがあります。使える暗号方式の違いや改ざんチェックのアルゴリズムの違いによりますが、サーバ側、クライアント側がそれぞれどちら

注1) BSD Unixで発展したさまざまなリモート接続プログラムとプロトコル。これらの名称のほとんどは、rで始まる。

を使うかなどを設定できます[注2]。SSH2では、鍵交換(Diffie-Hellman方式)を使って、外部に情報を漏らすことなくクライアント、サーバそれぞれが共通する鍵を入手しますので、SSH1プロトコルの公開鍵暗号系と秘密鍵暗号系を組み合わせた共通鍵の伝達方式(一方が共通鍵を用意してから公開鍵暗号を使って鍵を送る)よりも安全だと考えられています。

今回解説するsshのバージョンはOpenSSH 7系です。使っているsshのバージョンは、次のとおりssh -Vオプションで確認できます。

```
sshのバージョンを確認する
$ ssh -V
OpenSSH_7.2p2 Ubuntu-4ubuntu2.1, OpenSSL
1.0.2g  1 Mar 2016   Ubuntu 16.04
OpenSSH_7.4p1, LibreSSL 2.5.0   macOS
```

sshで対話シェルに接続する——基本形

sshコマンドは、sshdが動いているシステムへ接続するためのクライアントコマンドです。

接続元マシンと接続先(ホスト)の環境で同じ利用者名で接続するなら、オプションにホスト名(もしくはIPアドレス)を指定するだけで接続できます。

```
ssh ホスト名
```

利用者が異なる場合は、

```
ssh ホスト名 -l 利用者名
```

もしくは、

```
ssh 利用者名 @ ホスト名
```

とします。

```
my-mac:~ $ ssh her-ubuntu
her-ubuntu:~ $

my-mac:~ $ ssh her-ubuntu -l root   ここでは、rootとして接続していますが、sshdの設定でrootログインを禁止している場合は接続できません
her-ubuntu:~ #
$
```

接続すると、ターゲットマシン上のログインシェルが起動して、対話的処理ができるようになります。このときsshは、ターゲットマシンに対する端末として働いているイメージです。もちろん、接続、ログインするには、接続先にアカウントが必要です。

ターゲットマシン上では、sshdが動作していて、接続元のsshコマンドのリクエストに応答しているのです。ですので、接続先にsshdが導入されていなかったり、ファイアウォールなどによって接続制限が設けられている場合には、接続できません。

接続先のマシンが置き換えられている場合、名前をかたって、悪意のあるマシンに接続させようとするなどの異常を発見できるのがsshの特徴です。最初にサーバに接続するとき、**図1**のように、クライアントがサーバを認証するためにメッセージが表示されます。サーバの公開鍵のフィンガープリント(fingerprint)が表示されますので、問題なければ、yesと入力して接続します。

注2) 使用するsshのバージョンや設定により、どちらかのプロトコルでしか通信できない環境もあります。

▼図1 以前の接続情報と照合され、リストになければ接続するか確認を促される

```
$ ssh my-ubuntu
The authenticity of host 'my-ubuntu (192.168.33.117)' can't be established.
ECDSA key fingerprint is SHA256:F0jxan/LmlZJ/qN2allr9llplrxljMlarzGXDUoANg9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'my-ubuntu,192.168.33.117' (ECDSA) to the list of known hosts.
masa@my-ubuntu's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-62-generic x86_64)
```

STEP UP! フィンガープリントの確認方法

はじめて接続する際にクライアントに表示されるfingerprintは、サーバ側の公開鍵の情報です。この内容を確認する方法は、サーバ側で（通常システム管理者によって）ssh-keygenコマンドで確認でき、電話や印刷したメモなど、別の安全な伝達手段を使って対象ホストのフィンガープリントをユーザに伝達し、目視で確認します。

フィンガープリントは、クライアントのssh環境によってsha256（base64）形式や、md5（hex）形式で表示されます。サーバ側のシステム管理者は、ssh-keygenコマンドの-lオプションに加えて、-Eオプションで表示形式を指定して、両方の形式のフィンガープリントを利用者に伝えておけばよいでしょう。

```
クライアントの表示形式がsha256の場合
ECDSA key fingerprint is SHA256:F0jxan/⏎
LmlZJ/qN2allr9llplrxljMlarzGXDUoANg9.
サーバ側での確認
$ ssh-keygen -l -E sha256 -f /etc/ssh/ssh_⏎
host_ecdsa_key.pub

クライアントの表示形式がmd5の場合（:で区切られた16進数が並んでいる）
ECDSA key fingerprint is ff:de:ca:10:22:89:⏎
93:dc:22:3c:e9:4d:85:66:e8:a1.
サーバ側での確認
$ssh-keygen -l -E md5 -f /etc/ssh/ssh_host_⏎
ecdsa_key.pub
```

公開鍵も、暗号化方式によりssh_host_ecdsa_key.pub、ssh_host_dsa_key.pub、ssh_host_rsa_key.pub、ssh_host_ed25519_key.pubファイルが異なります（上の例ではECDSA方式）。どの暗号方式の鍵で通信しようとしているかの確認もしてください。

▼表1　~コマンド（一部分）

入力コマンド	機能
~.	接続を切る
~Ctrl＋Z	現在のsshをバックグラウンドジョブにする
~~	~そのものの入力
~?	エスケープ文字のリストを表示
~V	ログレベルを下げる（接続がおかしいときなど、デバッグ情報を制御する）
~v	ログレベルを上げる（上と同じ）

一度接続できたら、接続先の情報が~/.ssh/known_hostsに登録されるので、以降環境に変化がなければ確認作業はありません。

以前接続したリモートホストのフィンガープリントと異なったフィンガープリントが送られてくると、警告が表示されます（**図2**）。対象のリモートホストが、自分で置き換えたなどと問題がないことがわかっていれば、ssh-keygenコマンドを使って、known_hostsから以前の情報を削除し、接続しなおします。

```
$ ssh-keygen -R ホスト名
```

セッションを操作する

対話シェルで接続中に、接続元のシェルに一時的に戻りたいときや、別のシステムに接続したいときがあります。tmuxやscreenといった仮想端末[注3]を活用していれば、こうした切り替えが簡単という少し進んだ読者もいらっしゃることでしょう。でも、sshの機能だけでも、簡単なセッション操作ならできます。対話シェルの行頭で、~[注4]ではじまるコマンドを使えばsshに対してコマンドを送ることができます（**表1**）。

~Ctrl＋Zで処理を中断すれば、接続元のセッションはバックグラウンドになって、シェルに戻ってきます。fgでフォアグラウンドにすれば、元のセッションに戻ります。**図3**に複数のセッションを切り替える様子を示します。

sshが、リモートシステムが停止するなど何かの理由で応答しないときには、試しに~.を

注3）探検隊ではまだ紹介していません。
注4）チルダ（tilde）と読みます。

▼図2　リモートホストが入れ替わっているときの警告

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
...略...
```

▼図3 ~で複数セッションを行ったり来たり

```
my-mac:SD2016 masa$ ssh my-ubuntu↵        my-ubuntuに接続
masa@my-ubuntu's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-77-generic x86_64)
 ...略...
Last login: Sun May  7 07:28:38 2017 from 192.168.22.104
masa@my-ubuntu:~$ ls↵
a          Boostnote  Documents  Dropbox    Git        Music
Public     Templates  Work       Archived   Desktop    Downloads
examples.desktop      HubotStartup          Pictures   SVN        Videos
masa@my-ubuntu:~$ ~^Z [suspend ssh]    セッションをサスペンド

[1]+  停止                 ssh my-ubuntu
my-mac:SD2016 masa$ ssh her-ubuntu↵       her-ubuntuに接続
 ...略...
[masa@her-ubuntu ~]$ ~^Z [suspend ssh]    セッションをサスペンド

[2]+  停止                 ssh her-ubuntu
my-mac:SD2016 masa$ jobs↵      実行中のジョブを確認
[1]-  停止                 ssh my-ubuntu
[2]+  停止                 ssh her-ubuntu
my-mac:SD2016 masa$ fg %1↵     1番のジョブをフォアグラウンドに
ssh my-ubuntu

masa@my-ubuntu:~$ ls↵
Documents  Dropbox    Git        Music      Public     Templates  Work
Archived   Desktop    Downloads  examples.desktop     HubotStartup
Pictures   SVN        Videos
masa@my-ubuntu:~$
```

（my-ubuntuでの操作／her-ubuntuでの操作／my-ubuntuでの操作）

入力してみましょう。呼び出し元のシェルに復帰できるかもしれません。

リモートコマンドを実行する

sshは、対話端末以外にもリモートシステム上のコマンドを直接実行できます。

```
masa@my-ubuntu:~$ ssh her-ubuntu ls↵
2015
SVN
 ...略...
```

このしくみを使うと、バッチ処理などシェルスクリプト中で、リモートシステムのリソースを使って処理を分散させたり、リモートシステム上の結果も取り込んだ操作が継ぎ目なくできます。

これまでの使い方でおわかりのように、リモートシステムに接続すれば、外部のシステムリソース(CPU、メモリ、ファイルシステム)を活用できます。sshを活用することで、接続元のシステムが貧弱でリソースが欠けていても、快適に処理を実施できます。しかし、それはシステムリソースを気にしないでよいということではなく、接続元のシステムではなく、接続先のシステムリソースを代わりに意識する必要があるということです。

リスト1のスクリプトでは、配列serversのうち、**uptime**コマンドで現在のロードアベレージ(プロセスがキューで順番を待っている程度)を取得し、最も低いシステムにリクエスト(スクリプトコマンドの引数)を投げています。

認証方式

sshの認証方式は、公開鍵による認証、(Unixログイン時に使っている)パスワードによる認証、ホストベース[注5]による認証などがあります。認証の優先順位のデフォルト(いずれの認証も設定で使用を認めている場合)は通常、1)ホストベース、2)公開鍵認証、3)パスワードの順

注5) ホストベース認証は、rコマンド時代からマシンの名前と利用者の名前をリストに記述して、それが一致しているかで接続を許す認証方式。SSHでの安全な接続の設定方法については次回解説予定です。

▼リスト1　ロードアベレージで処理先を切り替えるスクリプト

```
#!/bin/bash

servers=(localhost my-mac my-ubuntu her-ubuntu)
cmds=$@
target=${servers[0]}
lb=9999
for s in ${servers[@]} ; do
  #echo $s
  l=$(ssh $s uptime | awk '{gsub(/^.*averages?:/,""); print $1*100}')
  # 後に解説するawkを使って加工しています。bashは実数をうまく扱えないので、uptimeの結果を100倍して整数に
  if [ $l -lt $lb ]; then
    lb=$l
    target=$s
  fi
done

printf "%s : %d¥n" $target $lb
ssh $target $cmds
```

番です。

サーバ側(/etc/ssh/sshd_config)で、どの方式の認証をするかを設定できます。sshdを導入した直後の環境では、公開鍵認証を実施し、これに失敗するとパスワード認証に切り替わります。ホストベース認証は認めない設定になっているのが一般的です。

パスワードによる認証は、パスワードが暗号化されていてもパスワードがネットワークを流れてしまうのであまり安全ではありません。公開鍵認証では、秘密鍵はローカルホストに置いておき、対となる公開鍵を(リムーバブルメディアで手渡しするなど)安全な方法でリモートホストに配置して、チャレンジ・レスポンス認証という、「ユーザの認証のための情報をまったくネットワークに流さずに、ユーザの正当性を証明する」ゼロ知識認証を用いた方式でログイン認証を行います。ホストベース認証は、利用者はパスフレーズを入力しなくてもよいので便利ですが、反面セキュリティレベルは少し下がります。

そのためsshは、公開鍵認証を中心に運用するのが望ましいです。公開鍵(接続先マシンに配置する)は、ssh-keygenコマンドを実行して、秘密鍵(クライアントマシンにしまっておく)と一緒に作成します。鍵を生成するとき、秘密鍵を復号するためのパスフレーズ(パスワード認証とは異なる)を付けることができます。

リモートシステムでコマンドを実行する際、バッチ処理などでは毎回パスフレーズを要求されるのは困ることがあります。鍵の生成時にパスフレーズの入力を省略しておけば、パスフレーズは要求されません。また、ここでは解説しませんが、パスフレーズを省略しなくとも、認証エージェント機能を使うことでも解決できます。

公開鍵は、リモートホストの~/.ssh/authorized_keysに登録します。これは鍵束ですので、接続元の数だけ、このファイルに登録しておきます。次の鍵作成手順例でもcatコマンドを追記でリダイレクトしているところに注目してください。

```
$ ssh-keygen -t rsa -b 2048↵
 ...略...
 パスフレーズを省略するには、パスフレーズ入力要求時に何も入れないで改行

$ scp ~/.ssh/id_rsa.pub remotesystem:.ssh/▽
id_rsa.ME↵
 安全な方法で公開キーを接続先~/.sshに配置
 ここでは便宜上scpを用いて、コピー先にもid_rsa.pubがあるかもしれないので、名前を変えてコピー

$ ssh remotesystem↵
 リモートシステムに接続して
$ cd .ssh↵
$ cat id_rsa.ME >> authorized_keys↵
$ exit↵
 もとのシステムに戻って...確認

$ ssh remotesystem ls↵
```

```
Enter passphrase for key '/home/bot/.ssh/id_rsa':
パスフレーズを設定していれば、ここでパスフレーズを求められる
...略...
lsの結果が表示されれば、成功!!
```

鍵の堅牢性は、コンピュータの進歩とともに相対的に弱まってきます。鍵は十分に長くないと、解かれてしまうかもしれません。近年では、RSA暗号で2,048bit相当の堅牢性以上でないと安全ではないと言われています。

一方で長い鍵は、悪意を持った解読者だけに解くための時間をかけさせるだけでなく、利用者である私達も影響を受けます。ここ数年の高速なマシン間で通信しているなら、RSA 4,096bit長のキーを使っても違和感はないでしょうが、遅いマシンでは接続時に「ずいぶんと待たされるな」と感じるでしょう。鍵の形式の選択は、機密性と実用性のバランスを考慮するようにしましょう。

重要な情報は~/.ssh/に

sshを使うとき、ホームディレクトリに、.sshディレクトリが作られます。ここに入るファイルは、

- 鍵ファイル：id_XXX.pub、id_XXX（XXXは暗号化方式）
- 接続済みホストのリスト(クライアント側の.ssh/)：known_hosts
- 承認済みのホストと鍵(サーバ側の.ssh/)：authorized_keys
- 個別の設定ファイル：config

などがあります。

.ではじまるファイルやディレクトリは、lsに-aオプションを付けないと表示されません。~/.ssh/は、他人からのアクセスができないようにこのディレクトリのパーミッションは、700(rwx------)に設定されていないといけません。また、ここに配置するファイルもパーミッションは600にしておきましょう。パーミッションの設定が不適切だと、非常に危険です。不適切な設定だと、sshは機密性を優先して、期待どおりに動かないこともあります。

秘密鍵が奪われると、ほかのホストに不正に侵入される可能性があります。秘密鍵を失うと、その鍵を利用したログインはできなくなります。鍵の管理は、物理的な鍵と同じように、細心の注意をはらってください。

今回の技術が活躍するところ

今回は、sshコマンドの使い方と、認証方式について解説しました。また、SSHプロトコルでは、通信内容の暗号化とチャレンジ・レスポンス認証の2つに暗号化技術を使っていることを解説しました。

現在のサーバオペレーションで、sshを使わないことはほとんどないでしょう。明示的に対話シェルとして使うことや、SSHプロトコルを使ったコマンドの通信経路暗号化など、ありとあらゆるところでsshは活躍します。安全なネットワーク通信を支えているsshのセキュリティが甘いと、重大な事故につながります。ですので、認証方式や暗号キーの管理設定、サーバ側の設定なども理解し、マスターするようにしましょう。

次回について

次回は、引き続きsshとsshを使ったコマンドについて解説します。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)】
ssh(1), screen(1), tmux(1), uptime(1), telnet(1), ftp(1), awk(1), ssh-keygen(1), ssh_config(5), sshd_config(5), scp(1)

Linux カーネル観光ガイド

第63回

RAIDのwrite hole問題を解決する

ジャーナル機能とLinux 4.12で登場するPPL

Text：青田 直大　AOTA Naohiro

Linux 4.12の開発はrc2まで進んでいます。ここから4～5週間で、この記事が出るころにはLinux 4.12がリリースされているかもしれませんね。今回は、RAIDにおけるwrite hole問題と、MDにおいてその問題を解決する機能について紹介します。

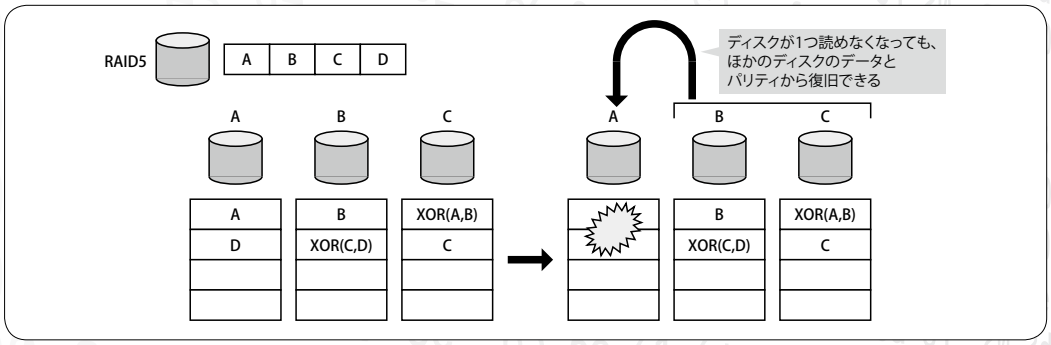
RAIDとは

RAIDとはデータ(やパリティ)を複数のディスクに分散して書き込むことで、I/O速度の高速化や、データの冗長性を実現する技術です。Linuxカーネルにおいては、ブロックレベルであればMD(Multiple Devices)という機能や、LVM(Logical Volume Manager)がRAID機能を提供します。また、ファイルシステムレベルではBtrfsがRAID機能を備えています。

RAIDには、書き込みを分散するだけで、冗長性は高めないRAID0や、データのミラーリングにより冗長性を高めるRAID1など、いくつかのレベルがあります。今回はおもにRAID5への拡張機能について見ていきます。

ここで簡単にRAID5の動きをおさらいしましょう。RAID5に書き込まれたデータは、複数のディスクに分散(ストライプ)されて書き込まれます。ストライプはchunkという単位で行われます。冗長性を高めるため、RAID5ではパリティというものが記録されます。たとえば、**図1**のようにRAID5のディスクに先頭から"AB"(1文字で1chunk長のデータを表現)とデータを書

▼図1　RAID5の構成参照



いていくと、ディスクAのchunk #0に"A"、ディスクBのchunk #0に"B"が書かれます。このとき、ディスクCのchunk #0にはパリティとして、"A"と"B"のXORが書かれます。さらに続けて"CD"と書いていくと、ディスクCのchunk #1に"C"が、ディスクAのchunk #1に"D"が、そしてディスクBのchunk #1にパリティである"C"と"D"のXORが書かれます。

パリティを使うことで、1つのディスクに障害が起きてもデータを復旧できます。たとえば、先ほどの状態でディスクAにアクセスできなくなったとしましょう。このとき、残りのディスクBとCからディスクAの内容を復旧できます。まず、ディスクB,Cのchunk #0を見ます。ここには、"B"とパリティの"XOR(A,B)"が書かれています。"B"とパリティをふたたびXORすることで、ディスクAのchunk #0にあったデータである"A"を得ることができます。同様に、ディスクBのchunk #1のパリティ"XOR(C,D)"とディスクCのデータ"C"から、ディスクAのデータ"D"を復旧できます。

write hole問題

このようにRAIDでは複数のディスクを用いて、ミラーリングやパリティ書き込みを行うことで耐障害性を高めています。その一方で、複数のディスクが使われることで別の問題も発生します。たとえばRAID5では、データを書き込むと、データ本体とパリティの2つを別のディスクに書くことになります。したがって、システムの電源断やクラッシュのタイミングによっては、データかパリティのどちらか一方だけが書き込まれてしまうこともあります。たとえば、先ほどのディスクAのデータ"A"を"X"に書き換えたところで電源が落ちたとします。ここで運悪くディスクCのパリティは書き換えられず"XOR(A,B)"のままで残ってしまうと、データとパリティとの間に不整合が生じます。このあとにほかの書き換えによって、パリティが更新されてしまえば問題は回避されますが、このままディスクBが壊れてしまうと深刻な事態におちいります。ディスクBのデータを復旧するため、ディスクAからデータ"X"が、ディスクCからはパリティの"XOR(A,B)"が読み込まれます。その結果、本来とはまったく違ったデータがディスクBのデータとして復旧されてしまいます。これをwrite hole問題といいます。

システムのクラッシュが起きてwrite holeがあるかもしれない場合には、MDはディスクの再同期を行います。これはすべてのディスクを読んで、データとパリティが合致するかを調べるプロセスで、ディスクのサイズと数が大きくなればなるほどたいへんな時間がかかるようになります。これを高速化する方法としてWrite-Intent-Bitmapがあります。これはディスク領域に対してBitmapを割り当て、これから書く領域にBitを立てておき、書き終わったらBitを落とすという方法です。クラッシュが起きたときには、Bitmapを参照してBitの立っているところを見れば、壊れているかもしれない領域だけを検査しにいくことができます。

Write-Intent-Bitmapで、再同期の時間が短縮できても根本的な解決にはなっていません。どんなに短くなっても、再同期の途中にディスクが壊れてしまう可能性は残っています。

今回紹介するのは、そんなwrite hole問題を解決する2つの機能です。1つはLinux 4.4から導入されているjournal deviceを追加する機能です。もう1つは、Linux 4.12(現在開発中)から登場する予定のPPL(Partial Parity Log)という機能です。

RAID5 journal

write holeの問題はディスクへの書き込みがアトミックではないことに起因しています。これはファイルシステムにおいて、メタデータの一貫性が壊れることと根を同じくする問題です。ファイルシステムにおいては、Ext4やXFSなどで一貫性を保った更新を行うためにジャーナ

リングが使われています。これをMDにも実装することでデータとパリティとの一貫性を保って更新を行えます。

MDのジャーナル機能は専用のジャーナルデバイスを設定することで使えるようになります。RAIDへの書き込みがあると、データとパリティを、ジャーナルデバイスに書き込み永続化してから、データとパリティを、今度はRAID本体へと書き込みます。

ジャーナルデバイスへの書き込み途中にシステムクラッシュが起きて、不完全な書き込みになったとします。この場合、ジャーナル中のチェックサムとジャーナル上のデータを比較することで、この書き込みは破棄できます。また、ジャーナルへの書き込みが終わってから、RAID本体への書き込み途中のクラッシュであれば、ジャーナル上の情報からRAID本体への書き込みをやりなおすことができます。

実際にMDのRAID5にジャーナルデバイスを作り、ジャーナルログの書き込みを見てみましょう(**図2**)。"`mdadm`"の"`--write-journal`"を使ってジャーナルデバイスを指定します。ジャーナルデバイスは"`mdstat`"には"`(J)`"の付いたデバイスで表現されます。

ここから、先ほどと同様に/dev/md0に"`ABCD`"と書き込み、ログデバイスの中身を見てみましょう。ダンプの最初に見えるのは、MDのsuperblockです(**図3**)。バージョン1.2のsuperblockはディスクの先頭から4KBの位置から始まり、先頭にマジックナンバーの"`0xa92b4efc`"を持ちます[注1]。ここから4KBがsuperblockです。superblockの中にはさまざまな情報が書かれていますが、本稿ではそのいくつかだけを見ていきます。

まず、0x1080から8Byteには、このデバイス上でのデータの開始位置が書かれています。MDではsuperblockの後ろに、ある程度の領域をとっていて、その後ろからデータ書き込み部分が始まるようになっています。ここでは0x4000セクタ(= 8MB)からデータが始まっています。こうしたsuperblockとデータ本体との空き領域は、前述したWrite-Intent-Bitmapなどを保存する領域としても使われます。続けて次の行の8Byte目(0x1098)からは、ジャーナルログの末尾(クラッシュ時にリカバリする範囲の先頭)が記録されています。

次に0x10a0から4Byteに、このデバイスのRAID構成するデバイス群(RAIDアレイ)の中でのデバイス番号が記録されています。このデバイスの場合は"`3`"であり、mdstatの出力中の"`sde[3]`"という部分に対応しています。最後に0x1100からのRAIDアレイの各デバイスの情報を保持する部分を見てみましょう。ここは各要素2Byteの配列となっており、デバイス番号または0xfffdなどの特別なIDを保持します。たとえば、このデバイスに対応する要素の部分(0x1106から)を見ると0xfffdとジャーナルデバイスのIDとなっており、mdstatの"`sda[3](J)`"と対応します。

では、0x800000からのデータ本体に目を移しましょう。すなわち、ここからはジャーナルの構造を見ていくことになります。ジャーナルは管理情報を持つメタブロックとデータブロックが並んだ形になっています(**図4**)。

先ほどのログの末尾位置にしたがって、0x800000 + 8セクタ = 0x8001000からのメタブロックを見ていきましょう。メタブロックのサイズは4KBで、先頭には、いつもどおりマジックナンバーである0x6433c509があります。その次の4Byteはメタブロックのcheck sum(CRC)であり、続けてバージョンと、メタブロックの中で実際にデータが入っているサイズが書かれています。次の16Byteにはシーケンス番号と、ログデバイス上での書き込み位置が記録されています。これらのシーケンス番号・書き込み位置・チェックサムを見ることで、メタブロックが正しい順序で、正しい場所に正しい内容で書き込まれていることを検査します。ここまでがいわばメタブロックのヘッダで、以降にpayloadが続きますが、このブロックはサイズが0x20であり、payloadを持ちません。

注1) リトルエンディアンのため逆順になっています。

0x802000から始まる次のメタブロック(seq：0x7ada1766)を見てみましょう。このメタブロックのサイズは0x160で、0x802020から0x802160まではpayloadが入っていることになります。payloadはこのメタブロックに続くデータブロックの内容を記述します。payloadの先頭2Byteはtypeを示します。このpayloadの場合“0”で、後続のデータブロック#1には、RAIDアレイ上のデー

▼図2　ジャーナルログの書き込みを確認する

```
$ sudo mdadm --create --verbose --level=5 --metadata=1.2 \
       --raid-devices=3 /dev/md0 /dev/libvirt_lvm/raid-disk{A,B,C} --write-journal /dev/sde
$ cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 dm-11[4] sde[3](J) dm-10[1] dm-9[0]
      20955136 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3] [UUU]

unused devices: <none>
```

▼図3　ログデバイスのダンプ

```
$ sudo xfs_io   (...省略...)
$ hexdump -C /dev/sde
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  fc 4e 2b a9 01 00 00 00  00 02 00 00 00 00 00 00  |.N+.............| # superblock: マジックナンバー
00001010  89 d6 fa 57 f8 1a 96 6c  4d 4f a9 db 8a 4b a0 f2  |...W...lMO...K..|
00001020  6e 61 6f 74 61 3a 30 00  00 00 00 00 00 00 00 00  |naota:0.........|
00001030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001040  61 9d 26 59 00 00 00 00  05 00 00 00 02 00 00 00  |a.&Y............|
00001050  00 c0 3f 01 00 00 00 00  00 04 00 00 03 00 00 00  |..?.............|
00001060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001080  00 40 00 00 00 00 00 00  b0 0b f9 0d 00 00 00 00  |.@..............| # データ開始セクタ番号: 0x4000
00001090  08 00 00 00 00 00 00 00  08 00 00 00 00 00 00 00  |................| # ログの末尾位置(8-15Byte)
000010a0  03 00 00 00 00 00 00 00  ed 49 dd bf 3f 6f c3 66  |.........I..?o.f| # デバイス番号: 3
000010b0  2f d0 fd 31 57 48 38 a1  00 00 08 00 88 00 00 00  |/..1WH8.........|
000010c0  61 9d 26 59 00 00 00 00  02 00 00 00 00 00 00 00  |a.&Y............|
000010d0  ff ff ff ff ff ff ff ff  3a 96 ca 6b 80 00 00 00  |........:..k....|
000010e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001100  00 00 01 00 fe ff fd ff  02 00 fe ff fe ff fe ff  |................| # RAIDアレイについての情報
00001110  fe ff fe ff fe ff fe ff  fe ff fe ff fe ff fe ff  |................| # 0xfffdがジャーナル
*
00001200  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00800000  09 c5 33 64 26 81 6d ee  01 00 00 00 20 00 00 00  |..3d&.m..... ...| # ここからデータ開始
00800010  54 f0 d9 7a 00 00 00 00  00 00 00 00 00 00 00 00  |T..z............|
00800020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00801000  09 c5 33 64 8d ae d3 bb  01 00 00 00 20 00 00 00  |..3d........ ...| # magic(0-3), check sum(4-7),
                                                            #  version(8), size(12-15)
00801010  65 17 da 7a 00 00 00 00  08 00 00 00 00 00 00 00  |e..z............| # seq(0-7), position(8-15)
00801020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| # payload: ここではなし
*
00802000  09 c5 33 64 03 4f a3 02  01 00 00 00 60 01 00 00  |..3d.O......`...| # seq: 0x7ada1766のメタブロック
00802010  66 17 da 7a 00 00 00 00  10 00 00 00 00 00 00 00  |f..z............|
00802020  00 00 00 00 08 00 00 00  00 00 00 00 00 00 00 00  |................| # payload#0 (0x802020-0x802033)
00802030  91 8e 6c 23 01 00 00 00  08 00 00 00 00 00 00 00  |..l#............| # payload#1 (0x802034-0x802047)
00802040  00 00 00 00 1a 97 28 d4  00 00 00 00 08 00 00 00  |......(.........| # payload#2 (0x802048-0x80205b)
00802050  08 00 00 00 00 00 00 00  91 8e 6c 23 01 00 00 00  |..........l#....|
...
00802160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00803000  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41  |AAAAAAAAAAAAAAAA| # データブロック#0
*
00804000  03 03 03 03 03 03 03 03  03 03 03 03 03 03 03 03  |................| # データブロック#1（パリティ）
*
00805000  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41  |AAAAAAAAAAAAAAAA| # データブロック#2
*
```

タが入っていることを示します。次の2Byteはflagで、そのあとの4Byteにサイズが入ります。payload#0の場合8セクタ(=4KB)になっています。その次の8ByteにRAIDアレイ上のアドレスが記録されます。ここでは"0"ですので、payload#0はRAIDアレイの先頭に書かれるデータについて記述しているということです。そのあと、4KBごとに4Byteのcheck sumが記録されます。今回サイズが4KBですので最初の4Byteでpayload#0は終わりで、0x802034からpayload#1が始まります。

payload#1はtypeが"1"で、後続のデータブロック#1がパリティであることを示します。payload#0と同様にサイズ・RAID上のアドレス・check sumが記録されています。0x802048から始まるpayload#2はふたたびデータブロックを記述し、そのサイズが4KBで、RAID上のアドレスが0x08セクタ(=4KB)の位置であると言っています。xfs_dbコマンドでは、4KBごとに書き込んでいたのでそれに対応して、4KBごとにpayloadが作られていることが確認できます。

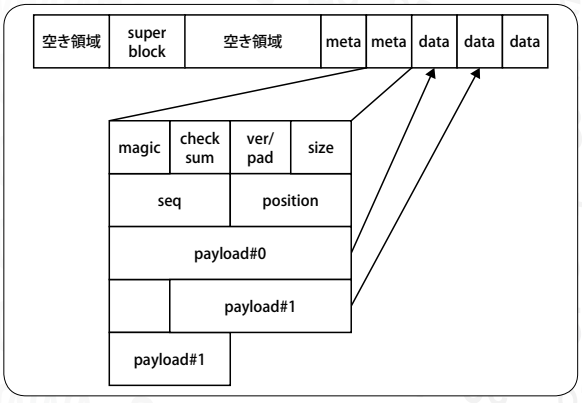
writeback cache

デフォルトでは、MDのジャーナルは"write-through"として機能しています。すなわち、アプリケーションからI/Oリクエストがあると、データとパリティとをジャーナルディスクに書き、RAIDディスクにI/Oリクエストを送ってから、アプリケーションに返ります。

"/sys/block/md0/md/journal_mode"ファイルを書き換えることで、このジャーナルを"write-back"にできます。すなわち、ジャーナルに書いた時点で、アプリケーションに返ります。ジャーナルデバイスが高速であれば、書き込みが高速化されます。

また、ジャーナルからRAIDへの反映がwrite-throughのように即時ではなく、30秒ごと(あるいはジャーナルの残りスペースが少なくなるまで)になるのでI/Oリクエストの回数も削減できることがあります。たとえば、冒頭の例のように"AB"と書くことを考えてみましょう。write-throughであれば、RAIDには"A"とその時点でのパリティ、"B"とその時点でのパリティと4つのI/Oリクエストが発行されますが、write-backであればパリティの更新が一度にまとめられ、I/Oリクエストの数が1つ少なくなります。

▼図4　ジャーナルデバイスの構造参照



PPL

さて、ここまでジャーナルを使ってwrite holeを解決する方法について、ジャーナルの中を見ながら紹介してきました。ここからwrite holeを解決するもう1つの方法であるPPL(Partial Parity Logging)についても紹介します。PPLは現在開発中のLinux 4.12に登場する予定の機能で、まだユーザランドのツールも整ってはいないので、ここでは簡単に紹介します。

PPLは、データ更新時に更新されるchunk以外のchunkでのパリティを記録しておくことで、write hole問題を解決します。**図5**のように4台のディスクで構成されたRAID5アレイを考えます。ここで"A"を"X"に書き換えます。

PPLでは、まず書き換えるchunk以外のchunkでのパリティ(部分パリティ)、すなわち"XOR(B,C)"をパリティが記録されるディスクのメタデータ領域に書き込みます。メタデータ領域は、先ほどのsuperblockとデータ開始までの空白部

分にあたります。部分パリティを書き終わってから、データである“X”と本物のパリティである“XOR(X,B,C)”を書きにいきます。うまく書き込みが終われば、ただ単に無駄に部分パリティを書いただけですが、問題はクラッシュが起き、たとえば“X”だけが反映された状態でディスクが失われたときですね。

まず、BかCのディスクが失われたときを考えてみましょう。このとき、ディスクDには古いパリティである“XOR(A,B,C)”が書かれています。このパリティは現状を反映していないので、これを使うと間違ったデータが復旧されてしまいます。ここで部分パリティと“X”をXORすることで、現状と合致したパリティを得て、失われたディスク上のデータを正しく復旧できます。また、パリティのディスクが失われた場合には単純に“XOR(X,B,C)”が正しいパリティと計算できるので問題はありません。

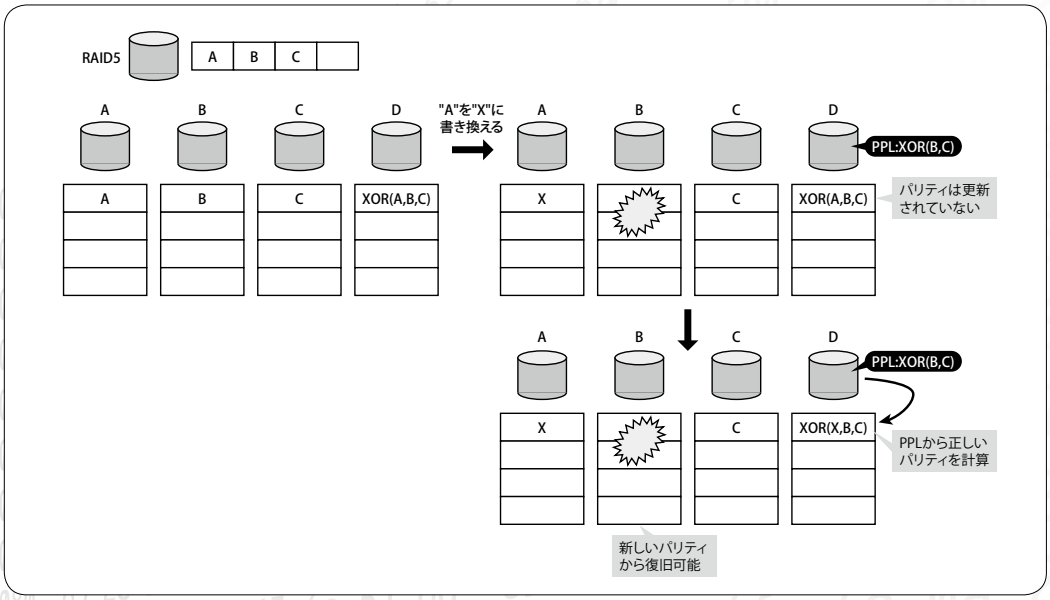
では、ディスクAが壊れた場合はどうでしょうか。このとき、パリティと“B”、“C”からデータを復旧させることになります。パリティがまったく書き換えられていなければ、“A”に戻りますがパリティが一部書き換えられていれば、“A”から“X”に書き換える途中のデータが復旧されることになります。ジャーナルであれば、このような中途半端な状態になることはなく、PPLとジャーナルの違いの1つと言うことができます。

また、部分パリティをパリティと同じディスクに事前に書くことから、書き込みパフォーマンスが最大30～40%低下すると言われています。とはいえ、RAIDを構成するディスクの台数を増やせばその分、部分パリティも分散して書き込まれスケールしやすい構成と言うことができます。このように、（ともすればSPoFともなり得る）ジャーナル専用のディスクが不必要であることは、PPLの利点とみることができます。

まとめ

今月はMDで構成するRAIDについて、write hole問題を解決する手段として、Linux 4.4で導入されたジャーナル機能と、Linux 4.12で今後登場するPPLについて紹介しました。とくにジャーナル機能は、うまく使えば高速化にもつながる便利な機能だと思います。SD

▼図5　PPLの構造



July 2017

NO.69

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
松山 直道　MATSUYAMA Tadamichi　ko@soum.co.jp

IPv6はもう普通に使える!? 各社の対応状況を概観

今回は、4月に行われた「IPv6対応クラウドサービスワークショップ」の模様をお伝えします。

IPv6対応クラウドサービスワークショップ

■IPv6対応クラウドサービスワークショップ
【日時】2017年4月19日（水）13:00～17:30
【会場】ビジョンセンター東京

■はじめに

IPv6の普及は当初の想定より長い時間がかかりましたが、ここ数年は伸びが目立ちます。PCやスマートフォンなどのクライアント機器から、ISP、アクセスラインとして提供される通信ネットワーク、大手のコンテンツサービス事業までかなりの部分でIPv6が普通に使える状況となっています。今後さらに普及を進めるには、大手以外のコンテンツ事業者が広くIPv6サービスを提供することが求められます。そこでコンテンツサービスのプラットフォームとなるクラウドサービスのIPv6対応状況に関する認識を共有することを目的として、IPv6普及・高度化推進協議会（v6pc）・日本ネットワークインフォメーションセンター（JPNIC）・インターネット協会（IAjapan）の主催で、本イベントが開催されました。来場者は80名を数え、熱心な質疑応答が行われるなど関心の高さがうかがえました。

■国内外におけるIPv6普及の現状

江崎浩さん（東京大学教授／IPv6普及・高度化推進協議会専務理事）からの開会の挨拶（あいさつ）とワークショップ開催の背景についての説明のあと、JPNICの佐藤晋さんから、国内外におけるIPv6普及の現状というテーマで最初の講演がありました。

JPNICでは2000年代初頭から国内の各ISPや組織に対するIPv6アドレスの分配を行ってきており、その立場から見たIPv6普及の流れをわかりやすく紹介されました。国内のアクセスラインとしてはNTT東西が提供するNGNサービス（フレッツ光ネクスト／ライト）が最大シェアとなりますが、今年頭には25%超のユーザがIPv6を利用可能な状態であることなどが報告されました。

世界のIPv6普及状況ですが、APNICとAkamaiの調査によると、ここしばらくはベルギーなどのヨーロッパ諸国が普及率上位に位置しており、日本は10位以下へと下がってきています。IPv6の接続環境は整ってきているが、肝心の接続先であるコンテンツサービス事業などのIPv6対応がいまひとつ進んでいないからではないかという話でした。

■モバイルネットワークにおけるIPv6の推進

次に伊藤孝史さん（NTTドコモ）、茂庭智さん（KDDI）、安力川幸司さん（ソフトバンク）から、モバイルネットワークにおけるIPv6の推進というテーマでの報告がありました。総務省IPv6研究会の「スマホユーザが、意識せずにIPv6の利用をはじめている状況に」との方針に従い、今夏以降新規に購入するスマートフォン／タブレット端末ではデフォルトでIPv6が有効となるように主要3社で準備を進めており、一部サービスではすでに提供を始めているそうです。モバイルネットワークでは端末が地域を

超えて移動することがあるため、固定電話網と異なり地域ごとに順次対応とはならずに、全国一斉にサービス開始になるそうです。各社とも2020年ごろまでに、設備の増強を進めるとのことでした。

■グローバル系クラウドサービスのIPv6対応状況

この後、グローバル系2社、国内系2社の計4社のクラウドサービス事業者の方から、それぞれのサービスにおけるIPv6対応状況についての講演が続きました。最初はアマゾンウェブサービスジャパンの荒木靖宏さんの発表です。AWSとして、IPv6のみに特化したプレゼンテーションを行うのはこれが世界で初めてとのことでした。

AWSでは、世界に16ヵ所あるリージョン中で直接ファイバーを引いていない中国リージョン以外では、すべてIPv6が使える状態だと言います。ただし、AWSは約90種類のサービスを提供しており、サービス／プロダクトごとのマネージャが顧客の要望を最優先に対応方針を決定しているため、現時点ですべてのサービスがIPv6対応しているわけではないそうです。最初にIPv6に対応したサービスはClassic Load Balancerで、これにより実質的に主要なアプリケーションはほぼIPv6に対応できると言えるようです。VPCなどその他サービスについても一通りIPv6対応状況を話していただきました。

続いては、田丸健三郎さん（日本マイクロソフト）によるAzureを含むMicrosoft全体の取り組みについての講演です。同社はかなり以前からRFC制定や実装の提供など積極的にIPv6普及に取り組んできました。2010年にはNICTやネットワーク機器ベンダ、通信事業者などと共同でIPv6技術検証協議会を設立し、IPv6利用環境における安全性の検証などを行っています。クラウドサービスのAzureも広い範囲でIPv6に対応していて、人工知能サービスAPIやIoTプラットフォームなどのアプリケーションサービスでもIPv6対応サービスを提供しているそうです。なおAzureではプラットフォームとして、Windowsだけではなく半分くらいはLinuxが利用されているとの話もありました。

■国内系クラウドサービスのIPv6対応状況

次に国内系クラウドサービスとして、堂前清隆さん（インターネットイニシアティブ）からIIJ GIOクラウドの状況について講演いただきました。GIOではPaaSとして普通にIPv6が使え、さらにその上のメールホスティングやWebセキュリティといったSaaSでもIPv6が問題なく使える状況とのことでした。クラウドサービスのほかにIIJモバイル（法人向け）・IIJmio（個人向け）というMVNOサービスも提供していて、そちらでもIPv6接続サービスが使えます。本講演では、LTE内蔵PCやスマートフォンを使用してIIJmio経由のIPv6オンリー環境を用意し、IIJ GIOクラウドの管理画面やコンソールをIPv6のみで操作するデモンストレーションが披露されました。

最後は横田真俊さん（さくらインターネット）の講演です。現在はレンタルサーバ・VPS・クラウド・専用サーバといった主要サービスに加え、高火力コンピューティング（GPU搭載サーバ提供）、sakura.io（IoTプラットフォーム）といったサービスでもIPv6が使えます。ただデフォルトでIPv6を有効とはせず、ユーザが設定で有効状態に切り替えるようにしているそうです。理由としては、利用状況を見る限りまだIPv6を使いたいと思っているユーザの割合が非常に少なく、意図的にIPv6機能を削除しているユーザもいるくらいだからとの話でした。今後1～2割くらいのユーザがIPv6を使うような状況になれば、デフォルトで有効にすることも検討したいとのことでした。

■終わりに

今回の講演は、各クラウドサービスのIPv6の扱いや考え方の違いがよくわかる興味深い内容でした。とくにこれまでIPv6対応に積極的でないとの印象が強かったAWSの話が聞けたのは良かったです。コンテンツサービス事業者が今後積極的にIPv6サービスを提供する方向に向かってくれることを期待しながら、細谷僚一さん（IAjapan）の挨拶で閉会となりました。SD

あなたのスキルは社会に役立つ

2011年3月11日の東日本大震災発生の直後にHack For Japanは発足しました。今後発生しうる災害に対して過去の経験を活かすためにも、エンジニアがつながり続けるためのコミュニティとして継続しています。防災や減災、被災地の活性化や人材育成など、「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ます。

第67回
地震対策Hackathonで感じた、進歩する技術と蓄積された経験の融合

●Hack For Japanスタッフ　鎌田 篤慎（かまた しげのり）　Twitter @4niruddha

我々、Hack For Japanでは、東日本大震災直後に多くのハッカソンを開催しました。本連載の中でもその取り組みを紹介してきましたが、その多くは被災地のニーズにマッチしたものにならなかったり、現地の人たちに使ってもらうにはハードルの高いものであったりなど、試みとしてはうまくいかないものが大半でした。

あの日から6年目を迎え、いくつかの災害や先の熊本地震などの経験を通し、現地のニーズを汲み取り、利用可能なものを提供するためのノウハウやナレッジは溜まりつつありますが、まだまだ課題としては大きなものがあります。

そんな中「TECHNOLOGY → PEACE 世界は誰でも変えられる」をテーマに2017年4月22日から23日の2日間をかけて、Samurai Incubate.inc[注1]による「SAMURAI ISLAND EXPO' 17」が開催されました。その中で、具体的に世界を変えるアクションを取る目的で地震災害対策を題材とした「地震対策Hackathon[注2]」も催されました。このハッカソンに、Hack For Japanスタッフの鎌田（筆者）がハッカソン参加者に対するメンターとして参加しましたので、今号では地震対策Hackathonの様子と、2011年のあの日から進歩したテクノロジなどが、どのように災害に対して機能していくかなど、以前に我々が開催したときのハッカソンと現在のハッカソンの間のアップデートも含めて、お伝えしたいと思います。

注1　http://www.samurai-incubate.asia/
注2　http://samurai-island-expo-hack.strikingly.com

地震対策Hackathonとは

```
while (Japan.recovering)
    we.hack();
```

今回の地震対策Hackathonは3つのテーマの中から1つを選び、指定されているWeb APIなどを任意で使いながらサービス開発していくものでした。その3つのテーマを紹介します。1つめが「状況収集・共有・避難・安否確認」情報に関すること、2つめが「お金」で、現金以外のお金に関すること、そして最後が「家・モノ」で、IoTに関することとなっていました。2011年当時にはなかった視点としては、やはり最後の「家・モノ」などのIoTに関するものが設定されているのが、2017年でのアップデートではないでしょうか。それらのテーマに従って、新規性や実現性、課題解決性といった観点を中心に審査されるものになっています。

2日間の開発期間の中、1日目の午前中と午後の頭がハッカソン参加者に対してメンタリングが行われる時間として当てられました。筆者のほかに当日のハッカソンにてメンターとして参加された方々は、株式会社トレタのCTO増井雄一郎さんと株式会社日本総合研究所の東博暢さん。技術面のアドバイスやサービス開発で想定しておかなければならない震災時の状況など、多角的な形で参加者のサービスコンセプトやプロトタイプに対して、アドバイスを1チーム15分で7チーム分、短い時間ながらも密度濃く行いました（**写真1**）。

過去の東日本大震災などで災害支援に入っていた、あるいは、災害向けハッカソンに参加していた

参加者は思いのほか少数であったので、被災地のニーズとのギャップであったり、被災者やボランティアセンターの様子、サービスを利用するうえで必須となるリテラシーのギャップなどの過去からの知見も伝えました。開発しているサービスと実際の現場との乖離を埋めていくためのメンタリングによって、その後のサービスコンセプトに大きく影響し、ブラッシュアップされていったので、やはり災害対策系のハッカソンなどでは実際の経験が非常に重要だとあらためて認識することができました。

それでは、実際に開発されていったサービスがどのようなものかを紹介していきます。

マイノリティ向けのサービス

最初に、震災発生時に一般の人たちよりも困難な状況に陥ってしまうマイノリティを対象としたサービスを開発していたチームを紹介します。

食品アレルギーを持つ人たち向けのサービス「Allescue」を開発した山本康史さんが代表を務める「Team-Lightning」です。このサービスは備蓄食料、救援物資の中に含まれるアレルギー物質を避け、安全な食品を提供することをコンセプトとしています（**写真2**）。首都圏直下型地震が発生した場合、400万人が避難所生活を迎えると言われており、その中で食品アレルギー保持者は1.5％の6万人ほど存在すると考えられています。こうした人たちは備蓄食料や救援物資を食べることができない状況に陥る可能性があります。また周囲の人たちには、非常時に食べ物を選り好みする人間として映ってしまうという問題も起こりえます。そうした人々に対するサービスというコンセプトを説明いただきました。

メンタリング当初、備蓄食料としてアレルギー対応食品を保存する話と、避難時のアレルギー対策を適切に救援物資として送り届けるという2つのアプローチがうまくリンクした形にはなっていませんでした。その後、議論を重ね、事前のアレルギー食品情報登録、決済情報の登録などを済ませておき、日常的にアレルギー対応食品の購入をするサービスとし、被災時も既存の決済情報をもとにアレルギー対応食品を購入、避難所に配送する形になりました。こうすることで、地方自治体でのアレルギー対応食品の備蓄が10％以下という課題に対応しつつ、食品提供会社のニーズ、配送の最適化なども考慮したサービスにブラッシュアップされていきました。日常的に使えるサービスであることと被災時のアレルギー対応食品の物資提供を最適にする視点は確かに可能性を感じるものです。

Blockchainを使ったサービス

続いて、東日本大震災当時と比較して、一般的になりつつある技術であるBlockchainを活用するサービスを開発していたチームを紹介します。

1つめは、QRコードを活用した落し物発見を目指す、チーム名と同名のサービスを開発した丸岡豊さんが代表の「Bo3.0」です。東日本大震災では津波被害が大きく、家屋が倒壊するだけでなく、写真など

▼写真1　地震対策Hackathonの様子

▼写真2　Allescueのプレゼンの様子

の思い出の品も失われてしまうことが問題となりました。このサービスではそうした問題を解決することを目的としていました。どういったアプローチのサービスかというと、あらかじめなくしたくない物に対してQRコードを発行、添付しておき、持ち主と物をBlockchainで紐づけて管理します。それを見つけた人がそのQRコードに記されるURLから持ち主とコミュニケーションが取れ、持ち主のもとに品を戻し、持ち主は拾い主に何かしらのお礼を提供するコンセプトです。

実際問題として、拾い主がQRコードからサイトにアクセスしたり、持ち主とコミュニケーションするモチベーションであったり、そもそもなくしたくない物に対して永続的に残せるQRコードをどのように準備するかなど、実際のサービスとしての課題も大きいですし、Blockchainの使いどころについては、もう一歩強みを活かすような発想が必要だと思われます。一方で、多くの人がわかるQRコードを利用するという発想は、利用者のリテラシーに左右されにくいという点で参考になる視点かと思います。

2つめは、個人ボランティア受付デジタル化サービスである「キズナコントラクト」を開発した小岩彩友美さんが代表の「やわらかいIT」です。このサービスには2つの役割が考えられています。1つは、個人ボランティアがすばやく現地に支援入りできるように促す役割です。個人がボランティアの支援作業に入るまでにはプロフィール登録や保険加入などが必要で、この事前準備にかかる手間の問題を解決します。もう1つは、個人ボランティアのボランティア活動の履歴をBlockchainで管理することで、ボランティアに慣れた個人であることや、その人の得意なスキルなどをステークホルダー間で可視化し、ボランティアの効率化などを図れるようにすることです。

初日のメンタリングでは、個人ボランティアの参加の敷居を下げる機能に加えて、ボランティアセンターでの受付業務のデジタル化を軸にしたコンセプトでした。しかし、実際にボランティアセンターでボランティアの受け入れを行う人たちのITリテラシーに差があることや、災害現場で一般的な紙によるやりとりの中にデジタル化を持ち込むことが、非常にハードルの高い行為であることなど、これまでのHack For Japanの活動の中で得た知見をインプットしたりしました。

その結果、サービス全体で機能の絞り込みやブラッシュアップが行われました。ボランティア活動で必要な基本情報の登録のほか、保険加入についてもデジタル化したことによって、申し込みからボランティア参加可能になるまでのタイムラグを大幅に短縮し、申し込みの敷居を下げるフローができました（**写真3**）。ボランティアセンターでの受付業務の効率化については、現在主流である紙でのチェックインも可能にすることで、現場の担当者のITリテラシーにも配慮。これに加えて、QRコードや利用者の多いLINEを活用したチェックインができるようにすることで、ある程度のITリテラシーがあれば受付業務も効率化できる実装にブラッシュアップされました。このサービスのすばらしいところは個人のボランティア活動が履歴としてBlockchain上で共有されることで、新たな災害が発生した際のボランティアの募集から、ボランティアに慣れている人の可視化、効率的な支援活動参加への一連の展開がスムーズに設計されているところです。

3つめは、Blockchainを活用した新しい地震保険を提案した堀口純一さんが代表を務める「TEAM ZBB」です。保険全般に言えることですが、とくに地震保険では被災後の支払い手続きなどが煩雑であり、被災者は被災後の家財の状況などから被災証明を行ったり、保険会社と査定のやりとりをしたり、時間のかかる作業の中で疲弊しがちです。そうした結果、受け取る保険金も納得のいかない額になってしまう場合も想定されます。こうした課題に対して、Blockchainを活用した新しいコンセプトの地震保険を提供するサービスです。メンタリングの時点ですでに非常に完成されたアイデアで、とくにアドバイスも必要ない水準のものでした。Blockchainの特性をうまく活かし、地震保険に関係するステークホルダー間で保険の契約情報を共有することで、保険にかかる一連の作業を非常に簡易的なものに落と

し込んでおり、地震に限らずマイクロ保険としてニーズの見込める点が秀逸でした。

Blockchainを活用するうえでは、関係するステークホルダーのニーズを全体でうまく満たしていく必要があります。このサービスでは、地震保険契約時に家財や家屋倒壊状況がわかるように契約者が持つ家財のデータ、家屋の寸法や水平などの情報をBlockchainで管理します。こうすることで、ステークホルダーである地震保険契約者、損保会社のほか、不動産管理会社や家財を販売する家電量販店などをステークホルダーとして含めたエコシステムができあがります。

それぞれのステークホルダーが望むものとしては、地震保険契約者は被災時のオペレーションの簡易化や納得のいく保険金の受け取りを望みます。このサービスでは被災時に、契約者は地震発生に伴うPush通知で家財や家屋の状況を写真に取り、現状を記録するだけで、その後はBlockchain上のデータとの差分によって保険金の査定などが自動で行われていく運用になっているため、従来の手続きが大幅に簡略化されています。ほかのステークホルダーである不動産管理会社は保険取引の最適化、保険を提供する損保会社は査定の簡素化、オペレーターの自動化、保険金支払いの早期化を望みます。また、それだけでは保険ビジネスとしての旨味は大きくないので、マイクロ保険の開発や保険加入者の家財、家屋データの有効活用などの展開も狙えます。家財を提供するような家電量販店も新しい商品の提案など、エコシステムとして参加するステークホルダーの望みが見事に満たされあう構造で、地震保険に限らず、新しい保険のあり方を示しました(**写真4**)。

新しい災害ハッカソンの方向性

このほかのチームもありましたが、今回の地震対策Hackathonで優勝したのは最後に紹介した、Blockchainを活用した新しい地震保険を発表した「TEAM ZBB」でした。2011年時点のハッカソンから技術も年々いろいろと進歩しています。過去のハッカソンを振り返ると、うまくいかなかった取り組みがおかしがちだった点は、非常事態時に被災者が新しく開発されたものを使う余裕がないという発想を持たなかったことや、善意だけでは継続されないボランティアという活動を意識し、活動資金を捻出することを発想として持たなかったことなどが挙げられます。

この点が新しい技術の登場や既存技術の進歩に伴って、解決することが可能になりつつある印象を今回のメンター活動を通して持つことができました。また、我々のように東日本大震災から継続して活動を続けているようなコミュニティが持つ知見というのは、新しく災害対策に向けて活動を行おうとする人々にとって非常に貴重な視座だということもあらためて実感しました。本連載の読者のみなさんも、ぜひこのような知見、視点を共有していっていただければと思います。SD

▼写真3　プレゼン時のキズナコントラクトのサービススコープの説明

▼写真4　TEAM ZBBが考案した新しい保険のビジネスモデル

温故知新
IT むかしばなし

第67回

パソコン上の日本語ワープロ

速水 祐（はやみ ゆう） http://zob.club/ twitter @yyhayami

はじめに

1980年代、パソコンで動かすアプリケーションの代表は、日本語ワードプロセッサ（以下ワープロ）でした。性能の低い8bitパソコン上で動作するワープロは、当時のワープロ専用機には到底及ばなかったのですが、16bitパソコンPC-9801の登場とその上で動作するワープロの出現によって、一般ユーザへのパソコンの普及に大きくつながりました。

当初のパソコン上のワープロ

1980年代初頭の8bitパソコン上のワープロは、専用ワープロにはほとんど及びませんでした。

そんな中、1983年に登場したのがNECのPC-9801Fと、その上で動作する管理工学研究所のワープロ「松」でした。管理工学研究所はPC-9801の高度な技術的ノウハウを持ち、「松」はアセンブラ言語のみでハードウェア能力を最高のレベルで引き出すコーディングで作成されていました。PC-9801の128KBの標準搭載メモリエリアを最適に使うことで、最大64KBのメモリ空間しか使えない8bitパソコンでは実現できない効率的なデータ管理を行っており、各種プリンタに特化したスプール機能も利用できました。操作の面では、PC-9801の標準機能に沿って、操作選択を10個のファンクションキーで行い、漢字変換はスペースキーの右横にある XFER キーで行うようになっていました。

「松」は、その操作の軽快性と完成度の高さから好評を博し、パソコン上のワープロとしてトップの地位を築いたのです。しかし、その値段は128,000円と高価で、PC-9801F2（398,000円）と24ピンプリンタPC-PR201（298,000円）、そしてディスプレイを合わせると約100万円にもなる高価なワープロマシンでした。フロッピーディスクなしのPC-9801Eをやっとの思いで購入した筆者などには、とても手の届かないシステムでした。

一太郎の登場

1984年秋、NEC PC-100で動作していたワープロソフト「JS-WORD」がPC-9801版としてアスキー社から登場します。JS-WORDの先進性は、そのかな漢字変換機能としてMS-DOSの機能の一部となる日本語入力FEP（フロントエンドプロセッサ）KTIS（Kana- Kanji Transfer Input System）を搭載したことです。KTISは、同社から発売されていた表計算ソフト「Multiplan」からも利用できました。

JS-WORDを開発していたジャストシステム社は、並行してまったく別のテキスト型のワープロを、IBM社のIBM JX用に開発していました。これが高性能・高機能な「jX-WORD」でした。

その3ヵ月後の1985年2月には、その名称を「jX-WORD太郎」としたPC-9801版が発売されました。jX-WORD太郎はMS-DOS上で動作することで、ファイル操作がMS-DOS上で行えました。

jX-WORD太郎の先進性はその操作性にあり、Multiplanのように ESC キーを押すことでメニューが出て操作を選択できたのです。ファンクションキーはかな漢字変換で使用するようになっていました。また漢字への変換操作はスペースキーで行い、変換文字の確定は ↵ キーを押すという、現在のかな漢字変換の操作の原点だったのです。

また、かな漢字変換機能はATOK3が受け持ち、今までの単漢字変換からある程度の長さの文書を一度の変換動作で行える連文節変換機能を実現し、変換の効率を圧倒的に高めて「松」の約半分の58,000円で発売されたのです。

半年後の8月には、jX-WORDが外れて「一太郎」と名称を変え[注1]、かな漢字変換がATOK4となって日本語FEPとして使えるようになり、ユーザの大きな支持を得ることになります。しかし「松」と比べて、その動作スピードと操作の快適性ではかなわない部分もあったようです。

1986年には、一太郎Ver.2が発売され、図形イメージの貼り付けが可能になるなど機能強化が行われたのですが、動作スピードは遅くなったため、コアな一太郎ユーザは変換のスピードアップのために辞書ファイルを、当時普及し始めた拡張バンクRAMのRAM-DISKへ転送することで、操作の快適性アップのための工夫を行っていました。1987年に登場した一太郎Ver.3.0(**写真1**)は、当時のソフトウェアには常識だったコピープロテクトが外れ、高速性がアップするとともに完成度も上がり、ワープロとして確固たる地位を築きました。

一太郎4の葛藤

シェル上で複数のアプリケーションを複数のウィンドウで動作させる環境が注目を集めていました。そこで。ジャストシステム社は、一太郎や花子[注2]などの複数のアプリケーションを切り替えながら動かすことができる、テキストベースのシェルであるジャストウィンドウを開発します。

その上で動作する一太郎Ver.4(以下一太郎4)は、640KBのメモリだけではすべての機能が使えず、花子を同時に動作させるために、当時発売されたばかりのEMSメモリや高価なハードディスクが必須になり、ユーザに大きな出費を促すものでした。また最初のバージョンはバグが多発したため、ジャストシステム社全社一丸となり、全力でサポートを行います。Ver.4.1、Ver.4.2と改良版を無償(郵送)で送り続けVer.4.3で、ようやくまともに動作する製品となったのです。

1988年は、すでにMS-DOS対応になっていた松も大きく進歩します。かな漢字変換部分は日本語FEP「松茸」となり、全面的に書き換えられたプログラムコードにより、圧倒的に高速になり、テキストエディタに匹敵するスピードを実現して「新松」と名を冠して発売されました。新松は操作キーなどをユーザが自由にカスタマイズできる優れたワープロでした。当時、筆者は一太郎4を使うのをやめて、新松を購入して快適に使用していたことを思い出します。

MS-DOS版ワープロの終焉

Windows 3.1が1991年に登場にするとパソコンの環境が大きく変わります。グラフィック操作による複数のアプリケーションの実行が当たり前になり、ジャストウィンドウのようなテキストベースのウィンドウシステムは時代遅れになったのです。そしてWindowsで動作するMicrosoft社製ワープロのMS Wordが使われるようになってきます。

一太郎も1993年にWindows版一太郎Ver.5.0を開発し、同時にMS-DOS版の一太郎Ver.5.0も発売します。しかし、Windowsの技術ノウハウを持つMicrosoft社の技術力はWindows上のソフト開発においては圧倒的であり、Windows版一太郎の動作は、MS Wordにはとてもかなわないものでした。

おわりに

その後、一太郎のバージョンアップは続きWindows版ワープロとしても優れたものに進化していったのです。一太郎の今後の進歩を期待します。SD

▼写真1　一太郎Ver.3.0の起動時のタイトル画面(初代PC-9801で実行)

注1) https://www.justsystems.com/jp/camp/just2010/column/index09.html

注2) ジャストシステムが開発したグラフィックソフトウェア。一太郎Ver.4と同時にジャストウインドウで動作する花子Ver.2が発売された。

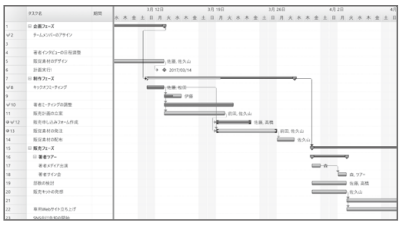
Software

グレープシティ、.NETアプリ向けコンポーネントセット「ComponentOne Studio 2017J v1」発売

グレープシティ㈱は5月17日、.NETアプリにデータグリッドやチャートといったUI部品を提供するコンポーネントセット「ComponentOne Studio」の最新版「2017J v1」を発売した。

最新版で注目の新機能は2つ。プロジェクト管理ツールでとくに有用なガンチャートを提供する「GanttView for WPF」。フィールド数の多い業務アプリで関連のある項目をまとめて縦並びにできる「MultiRow for ASP.NET MVC」。加えて、対応環境に新しく「Visual Studio 2017」「.NET Framework 4.7」「ASP.NET Core」が追加された。

初回費用は、すべてのプラットフォームに対応した最上位エディションである「ComponentOne Studio Enterprise」の1ユーザーライセンスで162,000円（税込）、1年単位の更新費用は64,800円（税込）となる。



▲GanttView for WPFの画面例

CONTACT
グレープシティ㈱ URL http://www.grapecity.com

Hardware

NECプラットフォームズ、Wi-Fiホームルータ「Aterm WG1900HP」「Aterm WG1200HP2」を発売

NECプラットフォームズ㈱は4月13日、IEEE802.11ac対応のWi-Fiホームルータシリーズ「Aterm」の新製品「WG1900HP」「WG1200HP2」を発売した。価格はともにオープン価格。

WG1900HPは5GHz帯で最大1,300Mbpsの高速通信が可能な3ストリーム高機能モデル、WG1200HP2は5GHz帯で最大867Mbpsの通信が可能な2ストリーム高機能モデル。両製品とも、動作中に電波状態の良い最適なチャネルに自動的に切り替える「オートチャネルセレクト」、最適な周波数帯に自動的に切り替える「バンドステアリング」、Wi-Fiの見える化および不正アクセスの検知を実現する「見えて安心ネット」を搭載している。

▲Aterm WG1900HP（左）とAterm WG1200HP2（右）

CONTACT
NECプラットフォームズ㈱ URL https://www.necplatforms.co.jp

Event

8月4～5日、「オープンソースカンファレンス2017 Kyoto」開催

8月4～5日、京都リサーチパーク（京都府京都市）にて、オープンソースカンファレンス2017 Kyoto（以下OSC京都）が開催される。

オープンソースカンファレンスは、インターネットの発展を支えているオープンソースの考え方を普及・啓蒙する「オープンソースの文化祭」で、毎年全国で開催されている。京都での開催は今年で11回目。約80の展示ブース、約70のセミナーが入場無料で見学できる。ブース出展としては、以下のものなどが予定されている。

・日本仮想化技術㈱によるOpenStackのデモ
・Oracle Corporation によるMySQLの製品紹介
・サイボウズ㈱による自社でのコミュニティ活動の紹介

●開催概要

日程	8月4日（金）10:00～17:00（展示は11:00～17:00） 8月5日（土）10:00～17:50（展示は10:00～16:00）
会場	京都リサーチパーク（京都府京都市下京区中堂寺南町134）
参加費	無料
主催	オープンソースカンファレンス実行委員会
協力	京都リサーチパーク㈱
企画運営	㈱びぎねっと

CONTACT
オープンソースカンファレンス2017 Kyoto
URL https://www.ospn.jp/osc2017-kyoto

Hardware

ネットギアジャパン、NAS製品「ReadyNAS」シリーズに、オンラインショップ限定のディスクレスモデルを追加

ネットギアジャパン合同会社は5月19日、NAS製品「ReadyNAS」シリーズのディスクレスモデル（ハードウェアサポートオンリーモデル）をオンラインショップ限定で発売することを発表。同日より販売を開始した。

ハードウェアサポートオンリーモデルとは、製品機能はそのままながらHDDを搭載しないディスクレスモデルで、導入先が自由にHDDを選定・搭載できる。HDDやサポートのコストを下げることで、型番の同じ製品をより低価格で提供する。

発売となったのは**表**の4製品（いずれもハードウェアサポートオンリーモデル）。AmazonとNTT-X Storeで取り扱う。

●**製品ラインナップ**

製品名	サイズ（HDD搭載可能数）	ネットワーク	価格（税込）
ReadyNAS 4312S	2Uラックマウント型（12ベイ）	10G SFP＋×2、1000BASE-T×4	993,600円
ReadyNAS 4312X	2Uラックマウント型（12ベイ）	10GBASE-T×2、1000BASE-T×4	993,600円
ReadyNAS 3312	2Uラックマウント型（12ベイ）	1000BASE-T×4	820,800円
ReadyNAS 3138	1Uラックマウント型（4ベイ）	1000BASE-T×4	216,000円

CONTACT
ネットギアジャパン合同会社 URL https://www.netgear.jp

Report

「AWS Summit Tokyo 2017」開催、Amazon CTOによる基調講演をレポート

5月30～6月1日、グランドプリンスホテル新高輪（東京都品川区）にて、クラウドサービスAmazon Web Services（以下AWS）に関するオフィシャルイベント「AWS Summit Tokyo 2017」が開催された。3日目のAmazon.com社のCTO、Werner Vogels氏の基調講演では、日本でのAWSの利用状況について語られたほか、新機能の発表も行われた。その概要をレポートする。

AWSと日本

現在、日本のAWSのアクティブユーザは月に10万以上にのぼる。スタートアップ企業でのAWS導入事例としてはSansan㈱、㈱メルカリ、ウォンテッドリー㈱が紹介され、エンタープライズ企業での事例としては㈱NTTドコモや㈱ファーストリテイリングの例などが挙げられた。また、日本においては強力なクラウドインテグレータの存在も欠かせないとし、アイレット㈱のcloudpack事業や㈱サーバーワークスの名前を挙げた。日本におけるトピックとしてはほかに、2018年に「大阪ローカルリージョン」が開設されることが発表された。

AWSの注力分野

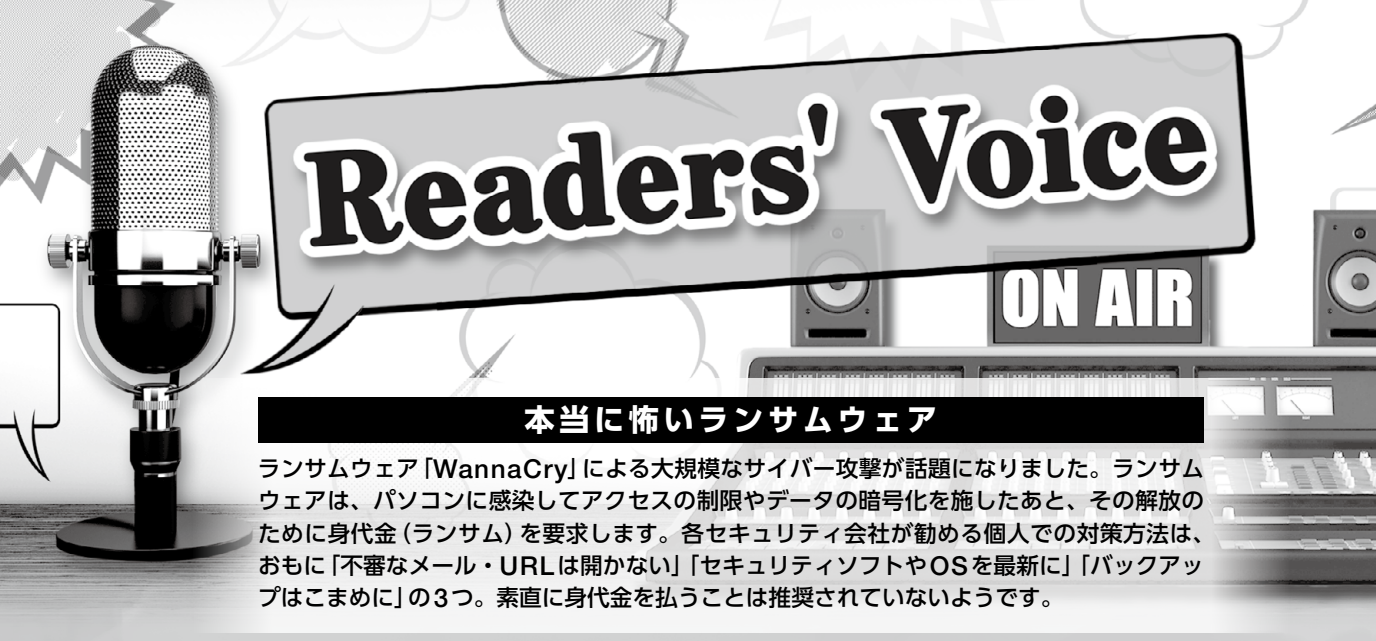
・サーバーレス：AWSでは、サーバーレスの核となる「AWS Lambda」、分散アプリケーションの構成を視覚的なワークフローを使って行える「AWS Step Functions」、分散アプリケーションの分析とデバッグを行う「AWS X-ray」、そしてNoSQLデータベースの「Amazon DynamoDB」を併用する構成を推しているとのこと。DynamoDBについては、キャッシュサービスの新機能「Amazon DynamoDB Accelerator」が発表された。

・ビッグデータ：データ分析分野においてWerner氏が紹介したのは、Amazon S3内のデータに対してSQLのクエリを実行できる「Amazon Athena」、Hadoopのマネージドサービス「Amazon Elastic MapReduce」、データウェアハウス「Amazon Redshift」。とくにRedshiftについては、新機能「Redshift Spectrum」が発表された。これは、Redshiftから直接S3上のデータを参照できるというもので、エクサバイトデータへの複雑なクエリを実行する実験をしたところ、1,000ノードクラスタのApache Hiveでは5年かかったものが、Redshift Spectrumではわずか155秒で済んだとのことで、来場者を驚かせた。

企業は不朽か

Werner氏が最後に語ったのは、企業の永続性について。以前にも増して企業が生き残ることが難しくなってきており、エンタープライズ企業も例外ではないという。クラウドによってスタートアップでもエンタープライズと同等のコンピューティングリソースを手に入れられるようになった一方、エンタープライズにはスタートアップと同等のイノベーションスピードが求められるようになり、企業にとってはいかに新しい領域に挑戦していけるかが重要だと強調した。

CONTACT
AWS Summit Tokyo
URL http://www.awssummit.tokyo



本当に怖いランサムウェア

ランサムウェア「WannaCry」による大規模なサイバー攻撃が話題になりました。ランサムウェアは、パソコンに感染してアクセスの制限やデータの暗号化を施したあと、その解放のために身代金（ランサム）を要求します。各セキュリティ会社が勧める個人での対策方法は、おもに「不審なメール・URLは開かない」「セキュリティソフトやOSを最新に」「バックアップはこまめに」の3つ。素直に身代金を払うことは推奨されていないようです。

2017年5月号について、たくさんの声が届きました。

第1特集　Linux入門【UNIXネットワーク編】

新人歓迎企画第2弾は、ネットワーク技術特集。通信プロトコル、ネットワークコマンド、ルーティング、ファイルサーバ、DNSをキーワードに、コンピュータ同士がどのようにつながり、インターネットを構成しているのかをゼロから解説しました。

先月号と同様、Linuxコマンドが苦手なアプリエンジニアが周りに多く、ネットワーク系となるとからっきし。みんなに読んでもらいたい。

ほまれさん／千葉県

新人さん向け特集かと思ったら、いきなりWiresharkでパケットを見ていてびっくり。Linuxというか、UNIX全般に有効な基礎知識ですね。

KKさん／愛知県

新卒準備号のうち、第2弾の今号だけ買いました。IT業界の中で転職して1年半近くになりますが、第1弾のLinuxとは何ぞや？のレベルからはようやく脱出できたことを今号の発売で気付きました。それでもまだまだ成長の途上。今後もお世話になります。

石丸さん／千葉県

今回の新人準備号もいろいろな世代の方に読んでいただいたようです。新人にとっては基礎を固めるのに良く、中級者にとっては振り返りに良い記事になりました。

第2特集　ドッグフーディング環境の作り方

ドッグフーディングとは、自社製品を社員自ら使って改善点を見つけるという試み。サイボウズのクラウドサービス開発において実際に行われたドッグフーディングを通じ、環境づくりや失敗しないためのノウハウを紹介しました。

サイボウズ流ということで、とても実践的で良かったです。

yoshitakaさん／神奈川県

まずは自分たちで使ってみる。必要だと思いつつなかなか実現できないですが、参考にしたいと思いました。

はにさん／大阪府

お客様からフィードバックを得るだけでなく、社内でも利用してフィードバックを得ることは大事だというのは明確なことです。しかし、それを実現するといった点でしくみ作りが難しく、今回の特集を参考にして実務につなげたいと思います。

牧田牧場さん／東京都

ドッグフーディングという言葉は初めて聞いた、という読者の方が多かったです。社内の人にレビューしてもらうということで、しくみと風土作りの両方が大事というお話でした。

第3特集　いまから学ぶブロックチェーン入門

ビットコインを支える基盤技術「ブロックチェーン」は、新しい取引のしくみとして、金融業界を中心に注目されている技術。本特集ではエンジニアを対象に、利用されている暗号技術やアルゴリズムの面からブロックチェーンを解説しました。

今一番興味があるので、タイムリーだった。

安井さん／東京都

セキュリティに関心のある身としては、金融業務モデルは参考になる。

とっぽさん／愛知県

ブロックチェーンについて、最近よく聞くけどわかってなかったので、良かった。

くまーーーーさん／神奈川県

5月号のプレゼント当選者は、次の皆さまです

①タイプライター風交換用キートップ「DN-914671」
石垣徹徹様（神奈川県）、木田光浩様（奈良県）

②寝るまでスマホ
さりさ様（愛知県）、桑野佳奈様（福岡県）、
今鷹進一様（福岡県）

③Acronis True Image 2017 New Generation
きゃろさん様（埼玉県）、ももんが様（静岡県）、
岡雅善様（東京都）、出玉のタマ様（大阪府）、
野口菅子様（福岡県）

④『エンジニアになりたい君へ』
秋原蘭様（千葉県）、齋藤優太様（広島県）

⑤『ITエンジニアのためのデータベース再入門』
海老原寛大様（神奈川県）、あっきー様（福岡県）

⑥『パーフェクトR』
金岡裕志様（千葉県）、松井一弘様（東京都）

⑦『Ansible構成管理入門』
ゆめかけ様（神奈川県）、野島吉仁様（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。

名前は知っているけれど……、という読者の方が多かったブロックチェーン。ビットコインと聞くとなかなか縁遠いものに感じますが、暗号技術やP2P技術を利用したデータ管理のしくみと聞くと、いろいろと応用が利きそうですね。

一般記事　クラウド事業者が考えるDDoS攻撃への対策と対処法

ニュースでも度々話題になるDDoS攻撃は、もはや他人事ではありません。記事ではDDoS攻撃について、概要、最近の傾向、おもな対策方法を解説し、さらにクラウド事業者が行う大規模な対処方法を紹介しました。

まだDDoS被害を受けたことはないが、見直しの良い機会になった。できる対策は限られていても、可能なところから対策は立てておきたい。
大黒さん／徳島県

CDNでの対策など、参考になりました。
ノリオさん／大阪府

圧倒的な物量で攻撃してくるさまはまるで災害のようです。「個人では対策の立てようもないのでは」とも思ってしまいますが、少なからずできることもあるのだと、プロフェッショナルによる冷静な話を聞くことができました。

一般記事　mBaaSのしくみ紹介【2】

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2回は、開発に関わったエンジニアへのインタビュー、実際の開発の流れの2本立てでした。

ニフティクラウドは弊社業務とも関わりが深いので、おもしろかったです。
チャチャ丸さん／東京都

mBaasについて興味があるので、参考になった。
さくらますさん／東京都

クラウドって、本当に一般化したものだと感心します。
鈴木さん／熊本県

○aaS系の派生が増えに増え、何をどう使えば良いかわからないといった方は多いはず。NCMBは国産のクラウドサービスということで、日本語のサポートを受けやすいという点はメリットですね。

短期集中連載　人工知能時代のLispのススメ【1】

現在使われる多くのプログラミング言語に影響を与えた「Lisp」について、その画期的なしくみや思想に、あらためて注目する短期連載です。第1回は、Lispの「すごいところ」と歴史、これからのLispについて学びました。

Lispを使うのかわからないが、ほかの言語との共通点を確認したりと勉強になる。
エゾモモンガさん／滋賀県

AIは結構好きなので、読ませていただきました。Lispはなんか懐かしいです。
風のピエロさん／長野県

年長者の読者から、「懐かしい」という声が多く寄せられました。初めて触った言語だから愛着がある、という方も多いようでした。若い人にも、今後の連載を通じて好きになってもらいたいですね。

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
August 2017

2017年8月号
定価（本体1,220円+税）
184ページ

7月18日発売

［第1特集］マシンラーニング・ディープラーニング

機械学習エンジニアになりたい!

学習のヒントと実践方法

機械学習（深層学習）はこれまでのビジネスを変革する手段として、各方面で組み込まれていくでしょう。そしていま、機械学習を武器に持つエンジニアが求められています。
本特集で、仕事で求められる機械学習エンジニアになるための第一歩を踏み出してみませんか?

［第2特集］

エンジニアのためのうけるプレゼン・すべるプレゼン――あなたの思いを伝える技術教えます

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2017年6月号
●P.23　特集「あなたのプログラミングを加速させるエディタ」第1章（2ヵ所）
［誤］Ctrl- [　　［正］Ctrl-]
●P.155　連載「Unixコマンドライン探検隊」リスト1中25行目
［誤］char *argv[3];　　［正］char *argv[6];

SD Staff Room

●只管「歩く男」となって幾星霜。「季節はずれの男」と後ろ指さされるのも吝かじゃない。左内坂を発し麹町から赤坂に抜けて、ここは六本木一丁目。隣は洒落乙な麻布十番。夜分「珍奇な男」となって信号の変わり目に「かけ出す男」。魚籃坂まで達してみれば「戦う男」の一里と半の足掻きも其れ迄よ。（本）

●IT昔話にキャリーラボのJET話が出て、大学時代を思い出す。当時はZ80をハンドアセンブルしてた。直接16進を打ち込み、ニーモニックを介さずに意味もわかっていた。雑誌掲載のゲームのダンプが動かないと嘆く友人に、ひと目見て4ヵ所の間違いを指摘したっけ。nパスのBASE80は便利だったなぁ。（Z幕）

●まわりの音が聞こえるイヤホン「ambie（アンビー）」を買いました。これはゲーム好きのお父さんにオススメです!　イヤホンでのゲーム中に家人からの呼びかけに気づけないと、“ゲームで遊んでいて人の話を聞かない、じゃまな亭主”ということになりかねません。でもambieなら家庭円満!（キ）

●ネットの情報を見ていて「Go言語の登場でC言語の必要性は減っている」と感じていましたが、工学分野でご活躍の方に聞いたところ、工学関連ではまだまだCが現役とのこと。現実を知るにはやはり直に現場の人に話を聞くべきだと感じました。人に会うのを面倒がってはいけないと日々、自戒しています。（よし）

●クロスバイク走行記＜第2段＞。今回は横浜みなとみらいまで往復80kmの旅でした。使った道路は環七と第一／第二京浜だけと単純な経路でしたが、アップダウンが意外と激しく、帰りはバテバテに。筋力・体力アップを誓った一日でした。そして世の習いどおり、ロードバイクが欲しくなって参りました。（な）

●いつもはちびぬいの型紙を作るときのできあがりイメージのラフ画くらいしか絵を描かないのですが、懇意にしているギャラリーさんで銀筆画や板絵といった、初めての技法を教われるという、わくわくするようなワークショップに参加する機会があり、ちゃんとした絵を描くのもやっぱり楽しい～と実感!　また頑張ろっかな。（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7月号

発行日
2017年6月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●広告
松井竜馬
大橋　涼
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
法人営業課（広告）
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。