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
11
2016
2016年11月18日発行
毎月1回18日発行
通巻379号
（発刊313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

》クラウドの秘密　》バグがないシステムの作り方

新人のときに
知っておきたかった

Special Feature 1

クラウドコンピューティングのしくみ

AWS
Azure
SoftLayer
Heroku
さくらのクラウド

恐れずに
リファクタリングを
するために

Special Feature 2

レガシーコード改善実践録

サイボウズ流バグゼロまでの道のり

特別企画

[次世代言語] Elixirの実力を知る
——Phoenixで高機能Webアプリ開発

新連載

及川卓也の「プロダクト開発の道しるべ」

特別連載

Jamesのセキュリティレッスン

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

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)

11
2016
［目次］
Software Design

contents

Special Feature 1 第1特集

新人のときに知っておきたかった

クラウドコンピューティングのしくみ

AWS
Azure
SoftLayer
Heroku
さくらのクラウド

11
2016
［目次］
Software Design

Special Feature 2 第2特集

恐れずにリファクタリングをするために

レガシーコード改善実践録

Extra Feature 一般記事

àla carte アラカルト

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]
Nature Picture Library/
Nature Production/amanaimages
[イラスト]
フクモトミホ
高野 涼香
どこ ちゃるこ
[本文デザイン]
*安達 恵美子
*石田 昌治(マップス)
*岩井 栄子
*ごぼうデザイン事務所
*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。

ISBN978-4-7741-8388-6
A5判／416ページ
定価（本体3200円+税）

科学技術計算のための
Python入門
開発基礎、必須ライブラリ、高速化

●中久喜健司 著

科学技術計算向けの、Pythonの実践的な入門書。理工系の学部や研究室等でもPython採用実績が増えてきています。本書では、実験やシミュレーション等で役立つPythonによる開発の基本を徹底解説。冒頭でロケットシミュレータの作成場面を想定し、コーディングの基礎からデバッグやテストまで、いつ、何をするか、具体的なフローを平易に紹介します。押さえておきたいライブラリである Numpy/SciPy/Matplotlib/pandas等もコンパクトな例と共に紹介。広く初学者の方々へ、言語の基本から実践テクニックまで一挙にわかる1冊です。

ISBN978-4-7741-8411-1
B5変形判／456ページ
定価（本体2980円+税）

改訂新版
JavaScript本格入門
モダンスタイルによる基礎から現場での応用まで

●山田祥寛 著

30,000部突破のベストセラー、日本で1番売れている JavaScriptの本が、6年ぶりに全面リニューアル！

「ECMAScript 2015」によって、いっそう進化をつづけるJavaScriptの新記法はもちろんのこと、基本からオブジェクト指向構文、Ajax、クライアントサイド開発まで、そしてテスト、ドキュメンテーション、コーディング規約など、現場で避けられない知識もしっかり押さえました。

ISBN978-4-7741-8426-5
B5変形判／272ページ
定価（本体2980円+税）

[改訂新版] プロのための Linuxシステム構築・運用技術

Red Hat Enterprise Linux対応

●中井悦司 著

好評につき重版してきた『プロになるためのLinuxシステム構築・運用』が、最新版のRed Hat Enterprise Linux（ver.7）に対応し全面的な改訂を行いました。これまでと同様に懇切丁寧にLinuxのシステムを根底から解説します。そして運用については、現場で得られた知見をもとに「なぜそうするのか」といったそもそも論から解説をしており、無駄なオペレーションをせずに実運用での可用性の向上をねらった運用をするためのノウハウをあますことなく公開しました。もちろん、systemdもその機能を詳細にまとめあげています。

ISBN978-4-7741-8422-7
B5変形判／256ページ
定価（本体2680円+税）

Xcodeではじめる 簡単iPhoneアプリ開発

[Xcode 8 & Swift 3対応]

●正 健太朗 著

本書は、「iPhoneアプリを開発してみたい！」と思う人が、最初に手に取っていただくことを想定した解説書です。難しいことは気にせず、開発ツールである「Xcode」をとにかく説明文のとおりに操作すればアプリを作ることができます。画面上の操作も、1つひとつのステップを掲載しました。プログラミング言語「Swift」のことをまったく知らなくても、iOSアプリを作れます。
iOSアプリの開発は、Swiftのプログラムと Storyboardでのグラフィカルな設定の組み合わせで成り立っています。本書では、とくにStoryboardでの作業に重点をおき、極力プログラムを書く量を少なくしています。

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

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

Software Design plus

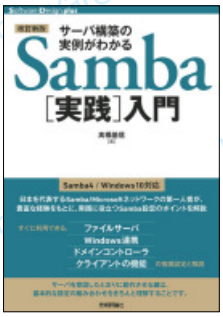
最新刊！

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。

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

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



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

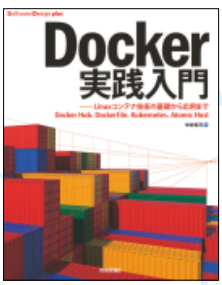
高宮安仁、鈴木一哉、松井暢之、村木暢哉、山崎泰宏 著
A5判・352ページ
定価 3,200円（本体）+税
ISBN 978-4-7741-7983-4

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

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



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

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

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・168ページ
定価 2,280円（本体）+税
ISBN 978-4-7741-7962-9

Unityエキスパート養成読本
養成読本編集部　編
定価 2,480円+税　ISBN 978-4-7741-7858-5

データサイエンティスト養成読本 機械学習入門編
養成読本編集部　編
定価 2,280円+税　ISBN 978-4-7741-7631-4

C#エンジニア養成読本
岩永信之、山田祥寛、井上章、伊藤伸裕、熊家賢治、神原淳史 著
定価 1,980円+税　ISBN 978-4-7741-7607-9

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

AWK実践入門
中島雅弘、富永浩之、國信真吾、花川直己 著
定価 2,980円+税　ISBN 978-4-7741-7369-6

シェルプログラミング実用テクニック
上田隆一 著、USP研究所 監修
定価 2,980円+税　ISBN 978-4-7741-7344-3

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

事例から学ぶ情報セキュリティ
中村行宏、横田翔 著
定価 2,480円+税　ISBN 978-4-7741-7114-2

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

OpenSSH［実践］入門
川本安武 著
定価 2,980円+税　ISBN 978-4-7741-6807-4

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

技術評論社

新連載

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1回 Product Managerとは

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

成功するテック企業の影の主役Product Manager

昨今、Product Manager（略してPM）という職種が注目されています。

米国、とくにシリコンバレーでは、以前よりPMが製品開発における重要な役割として定着しています。たとえば、GoogleのCEOであるSundar Pichai氏はGoogle ToolbarやChromeなどの製品開発においてProduct Managementの要職を務めた後、CEOの座まで上り詰めました。現在、苦境に陥ってはいますが、米国Yahoo!のCEO、Marissa Mayer氏もGoogleのConsumer担当Vice Presidentでしたが、彼女も元はPMです。シリコンバレーの成功したテック企業というと、優秀なソフトウェアエンジニアが有名ですが、製品開発の現場ではPMも欠かすことのできない職種として認識されています。実際、LinkedInやIndeedなどで米国でのProduct Managerの職を探してみると、多くの企業が募集していることがわかるでしょう。

一方、日本では「PM」と言うとProject Managerを指すことが多いようです。システムインテグレーターが行うような大規模のプロジェクトから自社開発案件など、さまざまなレベルのプロジェクトにおいて、おもにその進捗を管理する役割です。「プロマネ」と略されることもあります。Project Managerの知識体系はPMBOKとしてまとめられています。

このように、日本ではProduct Managerの知名度が低い時期が長く続いていましたが、この状況が変わりつつあります。シリコンバレーのテック企業の開発手法や企業文化を積極的に学ぶスタートアップ企業などを中心に、Product Managerを積極的に採用したり育成したりする企業が増えています。PMJP[注1]というコミュニティは、Slack上でのPMに関する日々の情報交換や非定期のオフ会を行っています。また、筆者の勤めるIncrementsでPM Meetupを開催したときも、予想を上回る人数の参加希望者が集まりました。10月には日本初のProduct Manager Conferenceも開催される予定です。

PMはミニCEO

さて、それではいったいProduct Managerは何をする人でしょうか？

よく言われるのが、PMはミニCEOだという例えです。これは、担当する製品やサービスの開発において、企業のトップであるCEOと同じ役割を果たすのがPMであるということを意味しています。CEOというと偉い人とか思う人がいるかもしれません。確かにその企業において最も重要な人物であることは事実ですが、一方で、すべての責任を負い、経営の舵取りを行うのがCEOです。そのためにはすべてのことをします。考えてもみてください。自分の会社でふんぞり返って命令だけ下していても、そのとおりに実施されないのならば、実施されるように変えていくのは、自分以外にはいません。

注1）http://productmanagers.jp/

やる人がいないので、自らが動くことさえあるでしょう。同じように、製品開発について最終的な責任を負う立場になるのがPMです。

製品開発は、複数の部署からのメンバーで構成されるプロダクトチームにより行われます。プロダクトチームにはエンジニアやデザイナー、テスト(品質管理)担当者などの技術メンバー以外にも、営業、マーケティング、広報、ユーザーサポート、法務などのさまざまな部署からのメンバーが所属します。もちろん、中には専属ではないメンバーがいるでしょう。もしかしたら、スタートアップ企業のように会社の規模が大きくない場合には、一部の担当者、たとえばマーケティングや広報などがまだ社員としてはいないこともあるかもしれません。いずれの場合も、足りない役割はPMが担うことになります。

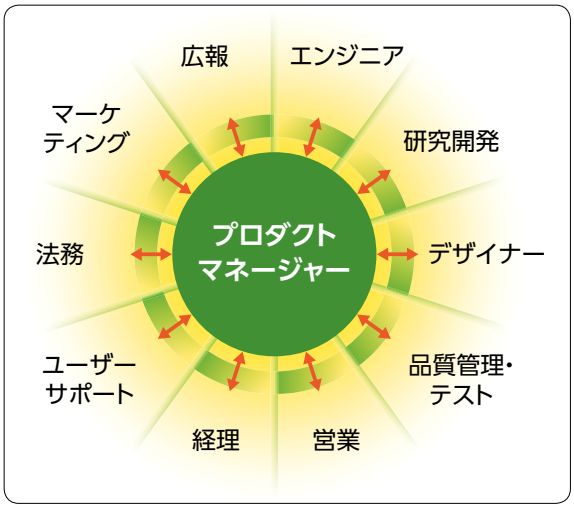
これらからわかるように、PMは製品開発の最終責任を担うとともに、開発において必要となる要素をすべてつなぎあわせる「糊(のり)」のような役割です。すべての役割を担うメンバーがそろっていても、きっと誰が担当とはっきりと決められないタスクも出てくることでしょう。その場合、PMが自ら担当するか、プロダクトチームのメンバーで担当を決めて、タスクを完了させる必要があります。

「マネージャー」という言葉に惑わされない

マネージャーというと、上長というイメージがつきまといます。確かに、指揮命令者と同義であることも多いでしょう。ですが、本来はManage、すなわち管理を行う役割を担った人がマネージャーです。何を管理するかによって、その役割は変わります。

PMはプロダクトを管理する人です。プロダクトは1人では作られず、チームで作るので、プロダクトチームを束ね、プロダクトを成功に導くプロセスを管理することがPMの役割です。

▼図　複数のチームメンバーから構成されるプロダクトチームとPM



マネージャーのイメージを変えるには、高校野球などのスポーツチームのマネージャーを思い浮かべると良いでしょう。この場合のマネージャーは、選手が試合において最高のパフォーマンスを発揮できるための健康管理から備品の整備など、さまざまなことを行います。練習メニューを考えるために試合のスコアを付けたり、分析したりすることもあるでしょう。

芸能タレントのマネージャーも良い例です。担当のタレントがより良い芸能活動を行えるように、仕事を段取りし、移動を助けます。

どちらもマネージャーは影の存在です。主役である選手やタレントが活躍し、ゴールを達成するためにすべてのことを行います。

組織において、このように働くリーダーのことをサーバントリーダーと呼びますが、PMもプロダクトチームにおいてサーバントリーダーとして振る舞う必要があります。我を強く出すだけでなく、チームメンバーが自発的にプロダクトのことを考え行動するような文化を醸成する必要があります。

ただし、このように言うと誤解されることもあるのですが、良い製品は合議制だけでは生まれません。強いリーダーシップの下に決断をし続ける。それもまたPMの大事な役割です。これについては次回さらに解説しましょう。SD

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

第95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。

BuckleScript

OCaml から JS を生成する「BuckleScript」

「BuckleScript」は、プログラミング言語「OCaml」で記述されたコードから、JavaScriptのコードを生成することができるコンパイルツールです。Bloomberg社によって開発され、LGPLに基づいてオープンソースで公開されています。

OCamlは、フランス国立情報学自動制御研究所（INRIA）で開発されているオープンソースのプログラミング言語です。関数型言語とオブジェクト指向言語の両方の要素を兼ねそろえており、型安全な静的型システムをベースとして信頼性の高いプログラムを作成できるという特徴があります。また、強力な型推論や代数データ型、モジュールシステム、多相バリアント、第一級モジュールといった言語機能によって、整理された可読性の高いコードが記述できる点も強みとして挙げられています。

BuckleScript自体は新しいプログラミング言語というわけではなく、OCamlの言語仕様に準拠したコンパイラとして動作します。生成されたJavaScriptコードは一般的なJavaScriptエンジンで実行することができます。すなわち、BuckleScriptを利用すれば、Webブラウザ上で動作するアプリケーションをOCamlを使って開発できるようになるということです。

BuckleScript 誕生の背景

BuckleScriptはJavaScriptをベースとした大規模なシステムをターゲットとして開発されました。いまやJavaScriptは単なるWebブラウザのための言語ではなく、クロスプラットフォームのための共通基盤として広く利用されています。JavaScriptエンジンの高速化も積極的に進められており、大規模なアプリケーションで採用されるケースも増えてきました。

BuckleScriptはそのような背景から生み出されたものです。大規模システムに必要となる信頼性をOCamlの言語機能によってカバーしたうえで、それを汎用性の高いJavaScriptに落とし込もうというわけです。開発元のBloombergによれば、BuckleScriptを使うことで次のようなメリットを得ることができるとのことです。

- **高い型安全性**……OCamlが備える堅牢で先進的な型システムと、強力な型推論機能を利用できる
- **オフライン最適化**……オフラインでのコンパイル時に多くの最適化が行われるため、極めて高速な実行コードを生成可能
- **ネイティブコードの生成も可能**……OCamlからiOSやAndroidなどの特定のプラットフォームのネイティブコードにコンパイルすることもできるため、性能が要求されるシーンでの選択肢が広がる
- **生成されたコードの可読性が高い**……人間にとっても読みやすいコードが生成されるため、デバッグ性／メンテナンス性が高く、既存のJavaScriptライブラリとの統合も容易
- **コンパイルが速い**……OCamlの優れたネイティブコード処理によって高速なコンパイルを実現している

JavaScriptコードを生成ターゲットとするプログラミング言語という発想そのものは決して新しいものではなく、メジャーな言語としてはTypeScriptやBabelJSといったものを挙げることができます。実際、BuckleScriptもそれらの言語に着想を得て開発されたそうです。ただし、TypeScriptなどが新しい構文を持った言語として開発されたのに対して、BuckleScriptではOCamlという既存の言語を採用しており、従来の資産を活用できるという点が大きく異なります。

OCamlは金融系システムの開発などにおいてすでに高い実績を持っている言語です。その資産やノウハウを、JavaScriptベースのシステムに活かせるということは、開発者にとって非常に大きなメリットと言えます。金融関連の情報サービスを手がけるBloombergが、自社のビジネス分野にフォーカスしたこのようなツールを公開したというのも興味深い点です。

BucklescriptはGitHub上のプロジェクトサイトからダウンロードできるほか、npm（Node Package Manager）を使ってインストールすることも可能です。また、Webブラウザ上で動作を確認できるデモページも公開されています。SD

BuckleScript
https://github.com/bloomberg/bucklescript

DIGITAL GADGET

vol.215

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

コンピュータグラフィックスの祭典SIGGRAPH2016[後編]
～ディズニーランドの街アナハイム。VRと映像技術編

技術の進化と映像の進化

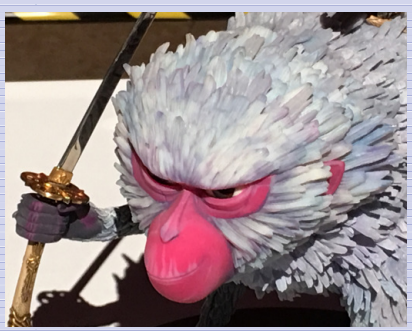
コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である、第43回SIGGRAPH 2016が7月24日から28日の5日間、米国アナハイムで開催されました。先月号に続いて、デジタルガジェット視点でレポートをお届けします。

デジタルダブルと呼ばれる、役者の代わりにCGの代役に演技させる映像技術がかなり浸透してきました。俳優の仕事を阻害しないためにデジタルダブルの活躍の場は限られているようですが、もとの俳優の顔や体の動きをキャプチャして作成されたデジタルダブルは、本人と見まごうほどの再現性で、危険な演技や、実写では不可能な演技を実現しています。

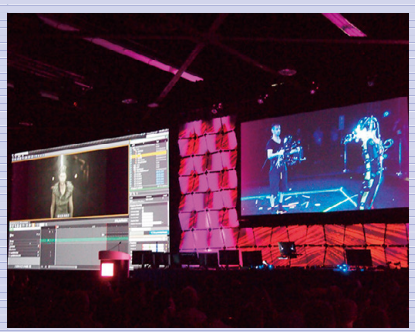
今回のSIGGRAPH中の目玉イベントの1つに、Real-Time Liveという、リアルタイム生成されたCG分野のテクノロジーやコンテンツを紹介する時間がありました。そこで紹介された「From Previs to Final in Five Minutes: A Breakthrough in Live Performance Capture」が会場の絶賛を浴びていました（**pic.1**）。これはタイトルどおり、膨大な時間をかけて演技をキャプチャする必要がある現状の時間のかかるデジタルダブル映像作りを打破するもので、たった5分で最終映像を製作するまでの手順を紹介したものです。

全身の動きを収録するモーションキャプチャと顔全面の動きを収録するフェイシャルキャプチャ、加えて音声収録用の機材を装着した女優さんが1名、CG映像空間内でのカメラ撮影を担当するバーチャルカメラ用の機材をかまえたカメラマンが1名で、女優さんは1人で2役をこなす映像作りでした。デモでは、CG映像で作られた2人の人物の動きや表情を一気に収録してしまい、通常は長い時間をかけて作る映像を、約5分で完成させてしまいました。映像生成には高画質の汎用ゲームエンジンとして知られるUnreal Engine 4が使われました。こういった技術の進歩でリアルタイムで映像作りが可能となり、ゲーム制作や、映画の製作の方法も変わってくることでしょう。

展示会場に鎮座した、バック・トゥ・ザ・フューチャーのデロリアン。顔をトラッキングするツールのブースでの展示。



3Dプリントをひとコマごとにストップモーション撮影した映画「Kubo and the Two Strings」で用いられた3Dプリントキャラクタ。



pic.1　1人2役のモーションキャプチャで、5分間で映像を完成させてしまう様子。左側がCG映像、右側が撮影の様子。

※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。

VRだけでない、触覚と視覚と没入感

VRでキャッチボール

展示会場で印象的だったのは、OptiTrack社の展示。VRヘッドマウントディスプレイを装着して、VR映像だけが見えており外が見られない状態の人と、バスケットボールをやり取りしている場面でした（**pic.2**）。VRの映像の中には、相手とバスケットボールがCGで描かれているわけですが、現実世界と比べても遅延がないからこそ実現できている振る舞いです。今までは無理だったことも、VRでいろいろ実現できるのではないかと、実感をもって期待を抱かせた展示の1つでした。

VRで震動

VRヘッドマウントディスプレイで、他メーカーの一歩先を行くOculus Researchからは、HapticWaveというVR視聴の際に用いる震動デバイスが体験展示されていました（**pic.3**）。HapticWaveは時計の文字盤のように360度方向に16個の電磁石ソレノイド（ZYE1-P40/20）を埋め込んだレコードのターンテーブルのような形状をしています。デモでは、テーブルの上で玉が跳ねている様子や、遠くで火花がバチバチしている様子が描かれていました。HapticWaveに手を載せると、手から伝わる振動によって、それらの映像とともに、どの方向からどの程度の振動があるのかが伝わってくるため、目からの情報を強化する役目を果たしていました。

VRの音

Googleが提供するVRコンテンツのプロジェクト、Google Spotlight Storyの中で、VRミュージックビデオとしても話題になった「Pearl」のメイキングセッションが人気でした。Pearlは、Pixarに所属するPatrick Osborne監督が手がけた映像で、父親とその娘が車で旅を続けながら音楽の夢を追いかけるストーリーです（**pic.4**）。登場するキャラクターの口の場所から、ちゃんとセリフの音声が聞こえるように音を収録したり、車の中での反響の量、車の中で反射する音量を実際の車を用いながら収録したそう。楽器の演奏も車の中にステレオマイクを用いて、車の中で収録したり、映像シーンの違いによって、車のドアを開けた状態で車の外で演奏したものなど、さまざまな状況での演奏を収録したとのこと。

これからのコンピュータグラフィックスの進化

現在のスマートフォンの性能は、フルCGアニメ映画『トイ・ストーリー』の制作時に使われていたコンピュータの2倍ほどの性能を持っていると言われています。10年前、20年前には、利用するために数千万円の機材が必要だったVRも、ローエンドのものはスマートフォンとダンボールの箱で楽しめるようになってきました。まだまだ理想のVR環境に到達するには時間がかかるかもしれませんが、確実にVR視聴の裾野は広がり、機材も映像コンテンツも出そろってきています。報道の分野でも、臨場感のあるパノラマ映像で、現地の様子をVRで体感できるといった、単なるゲームやエンターテインメントだけではない、次のVR時代が到来している気配がしています。

その一方、映画や演劇では、長年の演出の蓄積で、見せ方、的確に見せるための手法が確立しています。視聴者もその見せ方を受け入れ、慣れています。ときどき従来型ではない、見たことのない演出や映像を目にすることがありますが、それさえも、既存のルールを知ったうえで、意図的にルールを破って作られた映像です。それは映像言語、視覚言語と呼ばれる決まりごとや、ショット、カット割り、撮影の仕方、カメラや照明の決まりがあるからです。

VRの世界では、それらの万人に共通する伝え方がまだ確立しておらず、製作のための絵コンテひとつとっても、従来の手法はそのままでは成り立ちません。ハイエンドの機材を使った没入感のあるVRや、手軽なスマートフォンVR、パノラマ映像の視聴、スマートフォンの普及による縦動画の浸透も、従来の映像手法がそのままでは役立たなくなっています。

360度すべてが見渡せる状態で、実際どこを見てほしいのか。奥行きを見渡せる状態で、どこに焦点を当てて見れば良いのか。音がする方向が気になって、そちらを向いてしまったり、動く物を目で追いかけてしまったり、映

pic.2　VRヘッドセットのまま、バスケットボールをパスする様子。

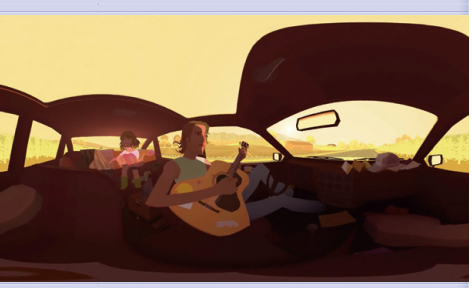
pic.3　ターンテーブルのようなパッドに取り付けられた震動素子で、VR空間の震動を感じる様子。

像制作者が演出として見てほしいものを、正しく的確に見てもらう方法が求められています。

今回のSIGGRAPHでメイキングが上映された映画『ジャングルブック』は、実写とCGで、ジャングルと多数の動物を描いた映画です。このジャングルブックでは、主人公の少年以外、背景のジャングルも動物達もほとんどすべてCGで描かれ、自由な演出、自由な映像作りができたそうです。そうすると、映像作りはすべては人間の想像力次第であり、想像できるものは、何でも作れるし、想像できないものは、どうやっても描けないということになります。

SIGGRAPHのセッションの各所で共通して言われていたことは、リファレンスの大切さでした。それはCGを使って自由に想像したものを描くとしても、その元となる本物を観察したり、現地に取材に行ったり、どのように自然に存在し、物の質感がどういったものなのか。素晴らしい映像作品を作り上げるには、頭の中ではわかっていると思いがちの事柄を詳細に観察することから始まるという考えが、どの製作者も共通してこだわっていることがわかった次第です。

来年夏のSIGGRAPH 2017は7月30日から8月3日の5日間、米国ロサンジェルスで開催されます。また今年の冬、12月5日から8日の4日間開催されるSIGGRAPH ASIA 2016は、マカオでの開催です。アジア各国から集まるCG作品や、最新技術展示に多くの期待が集まっています。SD



pic.4　声や楽器の音が出る位置を正確に再現したVR作品「Pearl」。

Gadget 1

Orah

https://www.orah.co/

ライブVR用カメラ

Orahは360度同時撮影、ライブVR配信用のカメラです。複数のカメラを使った360度撮影リグは数多く使われていますが、Orahは1台に4個のカメラ、4つの魚眼レンズ、4つのマイクを搭載し、機材のセットアップや取り扱いが容易で、4K（4800×2400）解像度のパノラマ映像が撮影できます。本体にあるイーサネット端子にネットワークケーブルを接続して配信することが可能です。暗い場所での撮影も得意とのこと。80×70×65mm、重量500g以下。予定価格は3,959ドルです。SIGGRAPHでブース展示がありましたが、実機はまだ開発中とのことでした。

Gadget 2

Ximmerse

http://www.ximmerse.com/

モーショントラッキングデバイス

XimmerseはパソコンVRにもモバイル用VRにも活用できる、VR用に手の動きを感知するモーショントラッキングデバイスです。光る球体を埋め込んだデバイスを両手に持って、光学カメラとして位置を検知するしくみです。スマートフォンを活用した簡易型のVRデバイス単独では手の動きと連動したコンテンツ作りは難しいため、このような入力デバイスと組み合わせて、操作感を高めるしくみが検討されています。

Gadget 3

Slate

http://www.iskn.co/

普通の紙をタブレットに

isknのSlateは使い慣れた普通の紙とペンで、デジタルタブレットの機能を享受するタブレットデバイスです。ゴムっぽい表面にA5サイズの紙を置いて描くと、USB経由でPCへ、またはBluetooth LE経由でiPadなどに画像が転送されます。また、使い慣れたペンや鉛筆に専用のリングを装着することで、Slateで使えるようになります。一度全部書き終わってから転送するのではなく、描いているタイミングで随時送信されていくので、デジタルデバイスの画面にもリアルタイムで反映します。一式169ドルで販売開始しています。

Gadget 4

Tango

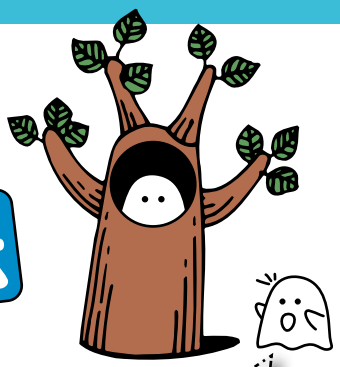
https://get.google.com/tango/

奥行きセンサー付きのタブレット端末

Google TangoはAndroidスマートフォンやタブレット端末に、Kinectのような距離センサーを搭載したデバイスです。SIGGRAPH会場では、会場を背景にCGで描かれた動き回る恐竜をAR（拡張現実感）で重畳表示するというデモを披露していました。Tangoに対応したスマートデバイスが各社から発売される予定となっており、流行のゲームのAR対応が加速するかもしれません。また、ゲームのみならず、三次元空間を把握するために用いることができ、窓のサイズを計測したりもできるそうです。

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

結城浩の再発見の発想法 42



デッドロック

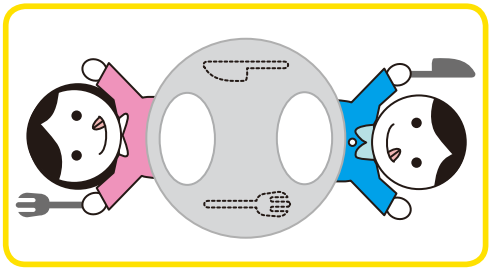
デッドロックとは

デッドロック(deadlock)とは、複数の動作主体が、**複数の資源を取り合った結果、動作を続けられなくなった状態**のことです。「動作主体」と抽象的な言い方をしましたが、コンピュータではスレッドやプロセスである場合が多く、実社会ではチームや人間などになります。

たとえばA、Bの2人がテーブルについて食事をするとします。テーブルにはフォークとナイフがそれぞれ1本だけあり、食事をするためにはその両方を必要とします。Aがフォークを取ると同時にBがナイフを取った瞬間、この2人はデッドロックになります(**図1**)。AはBがナイフを手放すのを待ち、BはAがフォークを手放すのを待ったまま、この2人は動作を続けることができなくなるからです。

このようなコメディタッチの食事は、複数のプロセスやスレッドが動作するときにどのように協調動作を行うかのたとえ話です。A、Bの2人はプロセスやスレッドなどの動作主体を表し、フォークとナイフは動作主体が必要とする資源を表しています。

▼図1　デッドロック



複数の動作主体が複数の資源を取り合うというのは、よくある状況です。たとえばA、Bという2つのプログラムがあり、Aは銀行口座XからYに送金し、Bは逆にYからXに送金するとしましょう。プログラムAはいったんXをロックして自分以外の誰もアクセスできない状態にし、続けてYをロックしようと試みます。プログラムBは逆にYをロックしてからXをロックしようと試みます。このままでは先ほどの食事と同じ状況が起こり得ることがわかるでしょう。

デッドロックの回避

デッドロックを回避するための方法はいろいろあります。**タイムアウト**を使えば、デッドロックは回避できます。たとえば、「10秒待ってもフォークとナイフの両方を確保できなかったら、いったん自分が確保したものを手放し、再度トライする」という方法です。動けなくなってから時間が過ぎると自分が確保したものを手放すことになるので、「動けなくなる」状態から抜け出せるからです。しかし、A、Bの2人が同じ回避策をとってはだめです。なぜなら、2人が同時に再度トライするため、さっきとまったく同じように「相手が手放すのを待つ」状態に陥ってしまうからです。動けなくなるというデッドロックは回避できましたが、今度はスタベーションという「動いているけれど資源が確保できない状態」に陥ってしまったことになります。こ

れを避けるためには、再度トライするまでの時間をランダムにするなどの工夫が必要です。

デッドロックを回避するために**対称性を崩す**という方法があります。たとえば、「フォークとナイフでは、必ずフォークを先に取る」のように、資源に順序を付けてしまうのです。このようにすれば、A、B両方ともフォークをまず取ろうとするので、デッドロックは防げます。

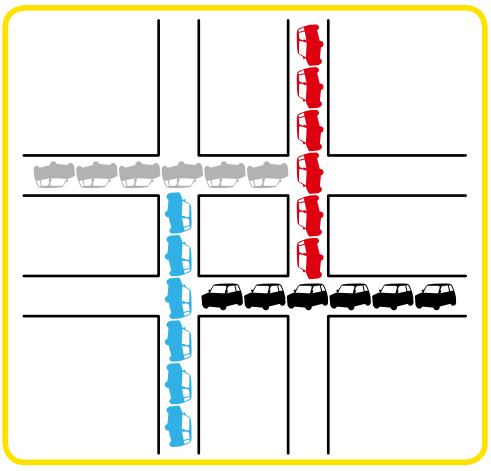
あるいはまた**資源をまとめる**という方法もあります。フォークとナイフを別々に確保するのではなく、「食器セット」という1つの資源にまとめてしまい、それ1つを確保するという方法です。これでデッドロックを防ぐことができますが、動作主体と資源がたくさんある場合に一般化すると、資源の利用効率が悪くなるという問題が起こるでしょう。

いずれにせよ、複数の動作主体が複数の資源を必要とする場合には、デッドロックが起きないかどうか、起きた場合の対策はどうするかを考える必要があります。

日常生活とデッドロック

日常生活で起きるデッドロックとして、信号のない道での交通渋滞が考えられます。1つの道を自動車Aで抜けようとしたら、前方に自動車Bが止まっていて動きが取れない。ところが自動車Bが止まっているのはそのさらに前方に自動車Cが止まっているから。そして自動車Cが動くのを阻んでいるのは自動車Dで、それを自動車Aが止めている……という状態です(**図2**)。通常、交通量が多いところには信号が設置され、信号がどちらの方向を優先するかを決定して、デッドロックを回避します。

▼図2　交通渋滞



複数の作業者がうまく意思疎通できないときも、デッドロックに近いことが起きる場合があります。2人の人がどちらも「相手から情報がやってきたら、自分の作業を進め、完成したら相手に情報を送ろう」と考えてしまったら、いつまでたっても話が進まないことになります。このデッドロックを回避するには、タイムアウトを使う(一定期間が過ぎたら相手からの情報を待たずに作業を進める)、2人の対称性を崩す(リーダー的な人を配置する)などの方法があるでしょう。もちろん、より根本的な解決策としては、仕事をどのように進めるかを事前に話し合っておくことでしょう。

作業者が少ないうちはデッドロックの発見は難しくありません。でも、共同で作業している人数が多くなってくるとデッドロックを見つけるのは難しくなります。また、デッドロックとまでは言えなくても、無駄な待ち時間が発生するのはよくあります。**図2**のように全体を俯瞰(ふかん)できれば、ばかばかしい事態が起きていることはわかりますが、多くの場合、個々の作業者は自分の周りしか見られません。リーダー的な存在が俯瞰的に作業の流れを見て、交通整理をしなくてはいけませんね。

◆　◆　◆

あなたの周りを見回して、「複数の作業者がいるのに思ったほど効率が上がらない」という状況はありませんか。複数の作業者の間でデッドロックに近い状況が起きることはないでしょうか。待ち状態が発生したときの対処法を作業者に伝えたり、作業者同士の対称性を崩したりして、効率を上げ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

増井ラボノート

コロンブス日和

第13回　廃れるページ

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

Author　増井 俊之（ますい としゆき）　慶應義塾大学

Webページの古さの視覚化

Webで面白い情報を見つけたとき、よく見たらとても古いページだということに気づいてがっかりすることがあります。がっかりするだけだとまだ良いのですが、SNSなどで紹介したあとで古さを指摘されたりすると恥ずかしいものです。

世の中のたいていのものは古くなるとゴミが溜まったり変色したりするものなので、古さになんとなく気づくのが普通ですが(**写真1**)、ファイルやWebページのようなデジタル情報は古くなっても見栄えが変わらないため、こういう失敗をしやすいと言えるでしょう。

情報が劣化しないことは、デジタルデータの大きな特長なのですが、古さが直感的にわかったほうが都合が良い場合もあります。古くても価値が変わらない情報なら気にする必要はないのですが、ニュースや技術情報のように新しさが重要な場合、間違って古い情報を参照しないように気を付けなければなりません。今回はWebページの古さを直感的に感じることができるようにするための工夫を紹介します。

廃れるリンク

はこだて未来大学の塚田浩二氏は、古いページへのリンクが汚く見える「廃れるリンク[注2]」というシステムを開発しました。

図1のようなWebページは古い情報へのリンクを含んでいるのですが、これを見るだけではどのリンクがどれだけ古いのかはわかりません。廃れるリンクシステムを使うと、**図2**のように古いページへのリンク文字列が汚く表示されるため、リンク先のページが古いことが明らかにわかります。

廃れるリンクシステムはプロキシサーバとして実装されています。廃れるリンクのプロキシサーバ経由でWebページにアクセスした場合、アクセス先のページの古さを取得したあとで、古さに応じてリンクのCSSを調整することによって古さを表現しようとしています。

▼**写真1**　明らかに古い壁

▼**図1**　古い情報へのリンクを含むWebページ

▼**図2**　廃れるリンクを利用した結果

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

注2）http://mobiquitous.com/dying-link.html

廃れるバックグラウンド

リンク先の古さを視覚化するだけでなく、ページそのものも古く見せると良いでしょう。自分のWebページの場合、作成時刻や編集時刻を利用して、きめ細かく古さを表示できます。

図3は私が昔作成した「廃れるバックグラウンド」というシステムです。古さに応じてランダムドットの数を増やしていますが、あまり美しいとは言えないようです。

図4はページ内の各部分のアクセス状況によってバックグラウンド表示を変えてみたものです。gimpで自動生成したシミのような画像と日付を表示することによって、古さを直感的に理解させようとしています。

廃れるバックグラウンドシステムは、バックグラウンド画像がイケてなかった点や、古いページが読みにくくなってしまった点であまり評判がよくなかったため、真面目に運用せず放置状態になったまま現在に至っています。

廃れるページ

「廃れるリンク」も「廃れるバックグラウンド」も10年以上前のシステムなのですが、手間の割に便利さが不足していたり見栄えが悪かったりしたため、日常的に使ってはいませんでした。一方、Web上の情報がますます増えてきた現在、古い情報を新しいものと勘違いする危険は増えていますし、古さの理解は以前よりも重要になっている気がするので、「廃れるバックグラウンド」と同様のシステムを次の方針で作りなおしてみることにしました。

- ブラウザ拡張機能として実装
- ページの古さを最初の数秒だけ全面に表示

任意のWebページに対して古さを視覚化するためには、どこかになんらかの細工が必要になるわけですが、最近のChromeやFirefoxは**ブラウザ拡張機能**を簡単に作れるようになってきたので、最近はこれを利用するのが良い気がしています(**図5**)。

従来はブラウザ拡張機能を作るのはかなりたいへんでしたが、最近はFirefoxでもChromeでも同じ形式で開発ができますし、開発の手間もかなり小さくなっています。

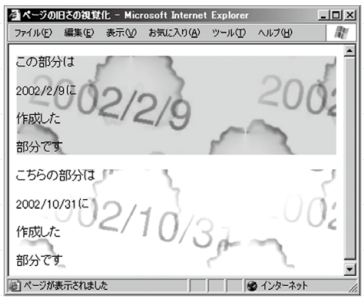
ブラウザの挙動を制御する方法

ブラウザ拡張機能

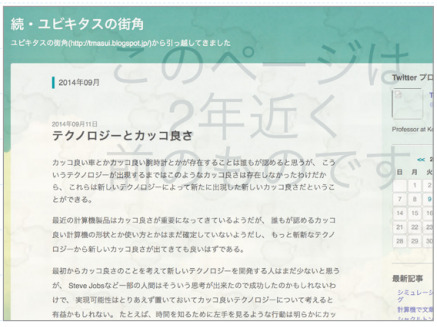
ブラウザの挙動を変えたり機能を追加したりするために、さまざまな「ブラウザ拡張機能」が利用されています。ブラウザ拡張機能とはブラウザアプリケーションの見栄えや動きを変更したり拡張するためのもので、ブラウザの挙動を制御するためのたくさんのAPIを利用して開発されるものです。たとえば ブラウザのタブ

▼ 図3 古くなったWikiページ

▼ 図4 バックグラウンド画像を工夫してみたもの



▼ 図5 廃れるページ



の挙動を変更したい場合は、タブの挙動を変更するAPIを利用することになります。

便利なブラウザ拡張機能を使っている人は多いでしょうが、ブラウザ拡張機能を自分で作って活用している人は少ないと思います。

ブラウザ拡張機能を作る方法はブラウザごとに異なっていましたし、APIが膨大で理解するには努力が必要ですし、作成方法がしょっちゅう変わるので対応がたいへんだったりしたからです。

たとえばFirefoxの場合、

- Chrome XULを利用して開発
- Pythonベースのcfxコマンドでadd-on SDKを利用して開発
- Nodeベースのjpmコマンドでadd-on SDKを利用して開発
- Web-extコマンドで開発

のように開発方法が追加されたり変化したりしており、最新の開発手法について行くのは苦労したものです。

ブックマークレットとGreasemonkey

ブラウザの挙動に変更を加えるのではなく、Webページの内容を利用したり修正したりするためには「ブックマークレット」や「Greasemonkey」などのシステムが利用されてきました。ブックマークレットとはブラウザのブックマークとして登録したJavaScriptプログラムで、普通のブックマークを呼び出すのと同じ方法で呼び出して実行するものです。

ブックマークレットはブックマークメニューなどから手動で起動する必要がありますが、Webページに対して自動的になんらかの処理を行いたい場合はGreasemonkeyというシステムが従来よく利用されていました。

GreasemonkeyはFirefoxの拡張機能で、ユーザがGreasemonkeyに登録したJavaScriptコードを自動的に呼び出して利用できるというものでした。

私の場合、拡張機能のAPIを勉強するのは面倒だけれど、自動的にWebページを操作したいことは多かったので、ブックマークレットやGreasemonkeyをよく利用していました。

たとえば、パスワードを要求するWebサービスに対して、本誌2016年6月号で紹介したEpisoPassのような「なぞなぞ認証」でログインを可能にするようなGreasemonkeyスクリプトを作って使っていましたし、面白いページをGyump(2015年12月号)に登録するためのブックマークレットは頻繁に使っています。

このように、これまでは、

①ブラウザの挙動を変えるためにブラウザ拡張機能を利用
②Webページを手動で操作するためにブックマークレットを利用
③Webページの自動処理化のためにGreasemonkeyを利用

のような使い分けをするのが普通だった気がします。

現在見ているWebページに対する処理を指定できるブックマークレットというのは便利なしくみなのですが、そもそもWebページのURLを記憶するためのブックマーク機能をWebページの操作に利用するというのはわかりにくいですし、登録も起動も簡単ではないので世の中で広く使われるようにはなっていないようです。本来、別のWebページにジャンプする機能と今見ているWebページに手を加える機能はかなり性質が異なりますから、同列に扱うのは適切でない気がします。

一方、ブラウザ拡張機能というのは、ブラウザの見栄えや挙動を変えるというはっきりした意図が明らかですし、インストールや管理も簡単なので、そういう用途にはこちらの方が向いていると思われます。最近は開発事情が若干改善しており、それほど苦労しなくてもJava

Scriptで拡張機能を作成できるようになってきたので②も③もブラウザ拡張機能を利用して問題ない状況になりつつあります。ブラウザ拡張機能の作成が簡単になってきた現在、もっとさまざまなブラウザ拡張機能が出現してきてほしいものだと思います。

現在、どこでもEpisoPassが使えるようにするための拡張機能を開発中なのですが、拡張機能をうまく使うとEpisoPassがかなり便利になりそうです(図6)。

こういう状況なので、廃れるページもブラウザ拡張機能として実装するのが良いと思われます。

廃れるページの実装

廃れるページの原理はとても簡単で、

①Webページの時刻を取得する
②古い場合は全画面を古い感じにして古さを表示する

というだけです。②については「廃れるバックグラウンド」で利用していたような画像を タグで重ねるだけなので簡単ですが、時刻を取得するのは少し注意が必要です。

Webページの作成時刻はdocument.lastModifiedで取得できるはずなのですが、これがうまく取得できないページがあります。また、動的に生成されるページでは現在時刻が設定されてしまう場合があります。私はLivedoor Blog[注3]で時々ブログを書いているのですが、このサービスではブログ記事ページの作成時刻は常に現在時刻になってしまい、記事内容が古くてもページは最新という状況になってしまいます。本当の古さを知るためにはきちんと内容を調べる必要がありますが、これは難しいので、ページ中に出現する日付をページの古さだと解釈することにしています。

利用結果

廃れるリンクも廃れるバックグラウンドも、

▼図6 Amazonのログイン画面でEpisoPassを利用する拡張機能

▼図7 「廃れるページ」の利用結果

いろいろ用意が必要な割に効果がパッとしないせいか、流行するということはありませんでしたし、私自身も利用していませんでした。しかしブラウザ拡張機能を使ったものは、インストールが簡単ですし、ページの古さがすぐにわかるので、今のところ満足して利用しています(図7)。ページの古さの表示はウザいこともあるでしょうが、数秒待てば完全に消えてしまいますし、古いページにアクセスする機会はそれほど多いわけではありませんから、なんとか便利さとウザさのトレードオフが我慢できるレベルになっている気がします。

廃れるページの拡張機能は、ChromeやFirefoxのストアに「sutare_extension」という名前で登録してあるので簡単にインストールできます。またソースをGitHub[注4]で公開しているのでご利用ください。SD

注3) http://masui.blog.jp/
注4) https://github.com/masui/SutareExtension

第9回　OSunCリレー(川越→千葉→金沢)とAKB

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

ダイエットに成功して、お酒解禁です

6月頭からスタートした断酒ダイエットですが、目標である「3ヵ月間で75kgまで落とす」を無事達成できました。3ヵ月目の9月1日(木)に75.6kgになりましたので、減酒フェーズに移行しました。今後「食生活は変えない」「体重をリバウンドさせない」「73kgを目標」「飲酒は最小限に抑える」の予定です。

「頑張らない」で集まるアンカンファレンス

アンカンファレンスの定義はいろいろとありますが、筆者の中での定義は「頑張らない」ということです。通常のカンファレンスを開催する場合、次の作業が必要となります。

- 日程決め/場所取り
- セミナーなどのプログラム調整
- デモ展示の調整
- 配布物作成
- 事前受付
- 当日運営

OSCはほぼ毎月、各地で開催されており、フルタイムスタッフが3名、日々準備作業を行っています。しかし、規模を拡大するには限界があります。また、多くの企業・団体からのご支援も未来永劫続くとは限りません。OSCを持続的に開催するためには、低コストで運営できる形態も模索しておく必要があります。そこで、2016年はとくに「オープンソースアンカンファレンス(OSunC)」の開催に力を入れています。

4回開催で実績のあるOSunC川越

現状、OSunCで実績のあるのが川越(小江戸)です。2013年から毎年、すでに4回開催の実績があります。とくに第3回は筆者が体調不良で不参加でしたが、問題なく開催されました。これは、開催の中心になっている小江戸らぐのみなさんの貢献が大でしょう。OSunC川越では、ライトニングトーク+立食のパーティーという形式で、和やかな雰囲気が最大の特長です(**写真1**)。

発表者も、当日その場で決定するので、事前準備を頑張らなくてもよいアンカンファレンスの趣旨に合っているといえます。OSunC川越は、OSunCのリファレンスモデルの1つと言ってよいでしょう。

千葉工業大学で開催したOSunC千葉

OSC東京は、会場が新宿から西に小一時間とやや遠いのが難点です。とくに千葉方面からは片道2時間はかかります。とある日、千葉で活動している皆さんと飲んでいて、「OSunCを千葉でやろう」「千葉工業大学が駅前で便利だよ」「そうだ、シェル芸の人(上田隆一氏)がいるじゃないか」というわけで、OSunC千葉の開催

▼写真1　いつものメンツでゆっくり話せるのがOSunC川越のいいところ

▼写真2　千葉は首都圏だけあって、たくさんの人が集まりました。千葉工業大学のキャンパスで集合写真

▼写真3 シェル芸の人の挨拶で2次会開始。2次会も30名以上で座るところがない人も

▼写真4 北陸ではOSC系のイベントは初開催ですが、50名も集まってくれました

が決まりました。OSunC川越のスタイルを踏襲しつつ、学生食堂を食事付きで借り、飲み物などを近くで買い出し、というスタイルで開催しました(**写真2**)。会場を借りる手続きなどでやや手間取るところはありましたが、最後は現場合わせで乗りきり、2次会まで楽しく過ごせました(**写真3**)。反省点は、飲み物が多過ぎたことでしょうか。持ち帰りに備えて、飲み物は350ml缶、食べ物は個別包装を選ぶとよいでしょう。また、「足りなかったら買い足せばいい」と割り切り、最初から少なめに見積るべきかもしれません。

北陸地域で初開催のOSunC金沢

これまで北陸地域でOSCは開催されませんでしたが、2015年3月から北陸新幹線が開通したこともあり、8月27日(土)にOSunC金沢を開催しました。北陸地域のコミュニティの皆さんが中心になり、初開催ということで県外からの参加者も多数集まりました(**写真4**)。開催場所は、観光スポットでもある近江市場の目の前の会場を選択しました。午前中の会場準備のあと、近江市場で名物の「ノドグロ」やお寿司などを買ってきて皆でランチ会、その後はLT＋立食形式の川越方式に、展示を追加する形での開催となりました。「次は冬の名物のブリやカニを楽しむ合宿をやろう」「OSunCやOSCもやりたい」という意見が出るなど、地域のコミュニティ活動を活性化するきっかけ作りになったようです。

自由にやることの難しさ

アンカンファレンスで最も難しいのは「自由にやる」ということです。ルールが定まっていれば、それに従えば問題ありません。OSCは「イベントのルール」の提示を目的としてきました。しかし、自由にやることは「1から自分で考える」ということです。多くの人は自由に慣れていないので、アンカンファレンスで何をしていいのかわからず、戸惑ってしまうのかもしれません。今後、開催を重ねて「アンカンファレンスとは」という共通理解を築いていき、時にはそれを壊してみるのが大事ではないか。そう感じる千葉、金沢でのOSunC開催でした。SD

Report

近江市場は食の宝庫

金沢を訪れた観光客が必ず訪れるスポットの1つが近江市場です。日本海の幸だけでなく、独特な「加賀野菜」も並んでいて、楽しませてもらえます。

今回は、OSunC金沢のお昼ご飯だけでなく、朝ご飯も市場で海鮮丼という、かなり贅沢な食事を楽しんできました。季節によっては甘エビやカニ、ブリなども美味しいそうです。次回は冬の時期に合宿をしたいね、という話をしているので、北陸のAKB(Amaebi、Kani、Buri)を楽しみにまた訪れたいですね。

朝から豪華海鮮丼が食べられるのも日本海側の魅力ですね

市場でお寿司の買い出し。左は北海道から参加してくれた松井健太郎氏

ツボイの なんでもネットに つなげちまえ道場

第17回

mbed OS 5での開発

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

前回説明をしてきたように、mbed OS 5はmbed 2.0からの移行コストが比較的低くできています。今回は、mbed 2.0のころから公開されていたmbedアプリケーションボード[注1]向けのライブラリを使って、ちょっとしたアプリケーションを実際に作ってみることにします。この連載のタイトルは「なんでもネットにつなげちまえ」ですので、mbed LPC1768をネットワークに接続してセンサの値を送信したいところです。しかし、現在のところmbed Clientがすんなり動くデバイスにmbed LPC1768が含まれていません。そこで、今回はクラウドにつなげるのではなく、スタンドアローンで動かすところまでやってみます。

ライブラリ

まず、mbedアプリケーションボード向けのライブラリを見つけましょう。mbed.orgにあるページ[注2]を見てみます。今回は、ここにある液晶のライブラリと温度センサのライブラリを使ってみることにします。

まず、プログラムを作ります。mbedオンラインコンパイラを開き、「新規」の「新しいプログラム」をクリックします（**図1**）。ここでプラットフォームとしてmbed LPC1768が選択されていることを確認し、テンプレートとして「mbed OS Blinky LED HelloWorld」を選択します。プログラム名は何でもよいですが、今回は「mbed-os-example-app-board」としました。下部にある、「プログラムとライブラリを最新のバージョンにアップデートする」のチェックボックスは、チェックを入れておいてください。

次に、液晶（C12832）と温度センサ（LM75B）のライブラリのインポートをしましょう。先ほどのページにあるC12832の、「Import library」ボタンをクリックします。するとオンラインコンパイラが開き、ライブラリをインポートするダイアログが表示されます（**図2**）。

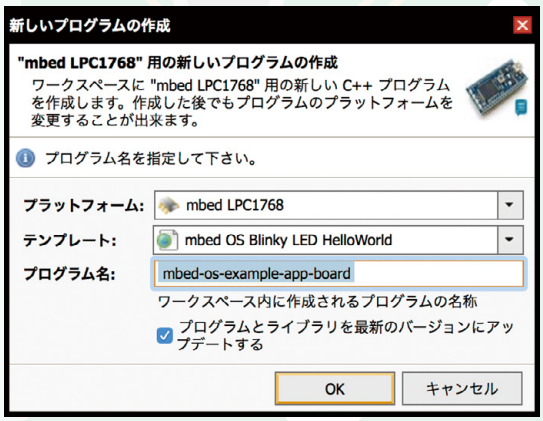
「Target Path」が先ほど作ったプログラムになっていることを確認し、「Import」ボタンをクリックしてください。C12832のライブラリのインポートを終えたら、LM75Bのインポートも同様に行います。

プログラム

今回参照したサンプルプログラムは、mbed 2.0用の液晶を使うサンプルコード（**リスト1**）

注1） http://ssci.to/1276

▼図1 新しいプログラムの作成



注2） https://developer.mbed.org/cookbook/mbed-application-board

と、mbed 2.0用の温度センサの値を読むサンプルコード(**リスト2**)です。

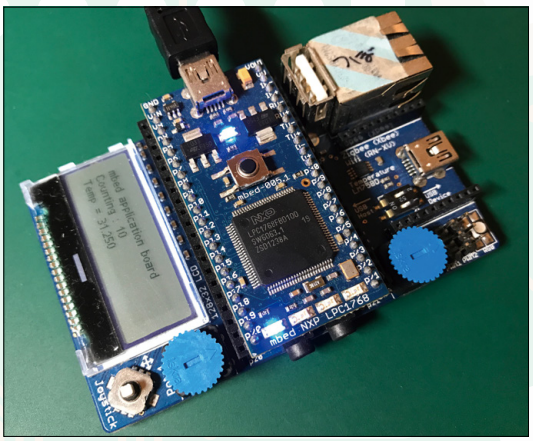
インポートを終えると、ワークスペースは**図3**のようになっていると思います。ここでmain.cppをダブルクリックして開き、それぞれのライブラリ用のサンプルプログラムを参照してプログラムを書きます。液晶にカウントアップをするプログラムと、温度センサの値を読んでUARTで送信するプログラムを参考にしたので、液晶に温度センサの値を表示するようなプログラムを書いてみました(**リスト3**)。

変更点は、mbed 2.0の`wait()`を、mbed OS 5の`Thread::wait();`に置き換えたことくらいです。また、このサンプルを長時間実行する人はいないでしょうが、intであるjを際限なく加算しているのが気になったので、一定値でjを1に戻すようにしました。それだけではつまらないので、別にスレッドを実行して、LEDの点滅もさせてみました(**写真1**)。

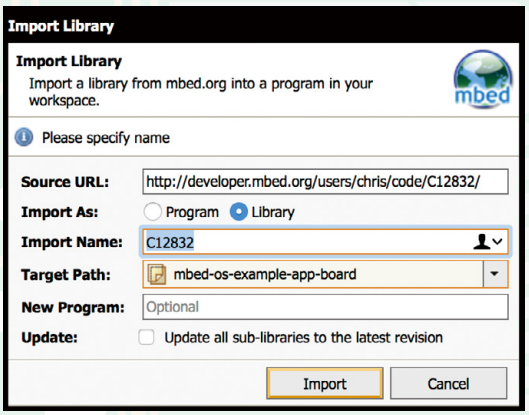
ローカル開発

次に、mbed-cliを使ってローカルでの開発を行ってみましょう。ここでは、OS Xを使ってビルドを行ってみました。もちろん、Windows

▼写真1　動作しているところ



▼図2　ライブラリのインポート



▼リスト1　mbed 2.0用の液晶を使うサンプルコード

```
#include "mbed.h"
#include "C12832.h"

C12832 lcd(p5, p7, p6, p8, p11);

int main() {
    int j=0;
    lcd.cls();
    lcd.locate(0,3);
    lcd.printf("mbed application board!");

    while(true) {
        lcd.locate(0,15);
        lcd.printf("Counting : %d",j);
        j++;
        wait(1.0);
    }
}
```

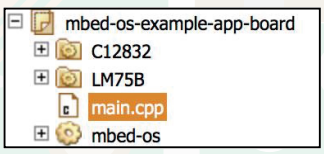
▼リスト2　mbed 2.0用の温度センサの値を読むサンプルコード

```
#include "mbed.h"
#include "LM75B.h"

LM75B tmp(p28,p27);

int main () {
    while (1) {
        printf("%.2f\n",tmp.read());
        wait(1.0);
    }
}
```

▼図3　ワークスペース



やLinuxでも同様の手順で開発を行うことができます。

なによりも最初に、mbed-cliの環境構築をしなければなりませんが、そこはWebに書いた記事を参考にしてください。この手順は、Windows用[注3]と、OS X用[注4]が日本語で書かれています。

環境が構築できているところで、mbed-cliを使って、まずプロジェクトを作成します。

```
$ mbed new mbed-os-example-app-board
```

すると、mbed-os-example-app-boardというディレクトリが作成され、中にmbed OSがダウンロードされます。とりあえず、まず作られたディレクトリの中に入り、オンラインコンパイラの操作でインポートしたときと同様にインポートを行います(**図4**)。

次に、プログラムを書きましょう(筆者は、vi党員です)。内容は、先ほどの**リスト3**と同一です。

```
$ vi main.cpp
```

ビルドするターゲットの設定をしましょう。GCCを使って、mbed LPC1768用のバイナリを作るように設定します。

```
$ mbed toolchain GCC_ARM
$ mbed target LPC1768
```

最後にコンパイルします(**図5**)。

これでバイナリができあがりましたので、mbedに転送をします(**図6**)。

コピーを終えたところで、mbed LPC1768のリセットボタンを押すと、オンラインコンパイラで開発を行ったときと同様の成果が得られます。

注3) https://developer.mbed.org/users/ytsuboi/notebook/ja-setup-mbed-cli-on-windows/

注4) https://developer.mbed.org/users/okano/notebook/setup-mbed-cli-on-mac-os-x-JP/

▼リスト3　今回作成したサンプルコード

```
#include "mbed.h"
#include "C12832.h"
#include "LM75B.h"

C12832 lcd(p5, p7, p6, p8, p11);
LM75B sensor(p28,p27);
DigitalOut led1(LED1);

void led_thread() {
    while (true) {
        led1 = !led1;
        Thread::wait(1000);
    }
}

int main() {
    int j=1;

    Thread th1(osPriorityNormal, (DEFAULT_STACK_SIZE), NULL);
    th1.start(led_thread);

    lcd.cls();
    lcd.locate(0,0);
    lcd.printf("mbed application board");

    while(true) {
        lcd.locate(0,11);
        lcd.printf("Counting : %d",j);
        if (j == 60) { j = 0; };
        j++;

        lcd.locate(0,22);
        lcd.printf("Temp = %.3f", sensor.read());

        Thread::wait(1000);
    }
}
```

コンパイラによる違い

オフラインでビルドしてできあがったバイナリのファイルサイズを見てみると、筆者の手元での実行時は、72,080byteでした。これに対して、オンラインコンパイラでビルドしたバイナリは、38,556byteです。ファイルサイズが大きく異なるのは、

オンラインコンパイラがARMのarmccというコンパイラを使っているのに対し、ローカルではGCCを使っているのが原因です。

OS XではGCCしか実行できませんが、Windowsでは、MDK-ARM[注5]や、IAR Embedded Workbench[注6]といった商用のコンパイラが実行できます。試しにWindowsで、これらのコンパイラを使ってビルドしてみたところ、MDK-ARMでビルドすると38,920byte、Embedded Workbenchでは44,080byteのバイナリが生成されました。今回は、MDK-ARMのほうが小さなバイナリを生成しましたが、プログラムによっては、Embedded Workbenchのほうが小さなバイナリを生成することもあります。

普段使っているLinuxなどのソフトウェア開発では、バイナリのサイズをあまり意識する機会がありませんが、マイコンで動かすときにはFlashメモリのサイズなどの制約があります。また、バイナリファイルのサイズが小さいということは、一般的には実行効率も良いバイナリが生成されているということを意味します。以前、マイコン向けのベンチマークソフトをビルドして実行してみたことがありますが、こうした商用コンパイラでビルドしたバイナリのほうが明らかに高いスコアを出しました。本格的な開発を行うときには、こうしたコンパイラの検討も必要となるでしょう。当面は、オンラインコンパイラを使うことで、効率のよいバイナリをビルドして使うことができます。SD

注5） http://www.arm.com/ja/products/tools/software-tools/mdk-arm/

注6） https://www.iar.com/jp/iar-embedded-workbench/

▼図4　手動でインポートを行う

```
$ cd mbed-os-example-app-board/
$ mbed add http://developer.mbed.org/users/chris/code/C12832/
$ mbed add http://developer.mbed.org/users/chris/code/LM75B/
```

▼図5　コンパイル作業

```
$ mbed compile
～中略～
Link: mbed-os-example-app-board
Elf2Bin: mbed-os-example-app-board
+-------------+-------+-------+------+
| Module      | .text | .data | .bss |
+-------------+-------+-------+------+
| Fill        |   106 |     4 |    5 |
| Misc        | 53056 |  2228 |  800 |
| hal/common  |  4428 |     4 |  401 |
| hal/targets |  5383 |     4 |  244 |
| rtos/rtos   |   829 |     4 |    4 |
| rtos/rtx    |  6343 |    20 | 6810 |
| Subtotals   | 70145 |  2264 | 8264 |
+-------------+-------+-------+------+
Allocated Heap: 2048 bytes
Allocated Stack: 3072 bytes
Total Static RAM memory (data + bss): 10528 bytes
Total RAM memory (data + bss + heap + stack): 15648 bytes
Total Flash memory (text + data + misc): 72409 bytes
Image: ./.build/LPC1768/GCC_ARM/mbed-os-example-app-board.bin
```

▼図6　mbedへ転送する

```
$ cp ./.build/LPC1768/GCC_ARM/mbed-os-example-app-board.bin /Volumes/MBED/
```

PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2016年11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 Parallels Desktop for Mac Pro Edition 2名

Mac上で、WindowsやほかのOSを実行できる仮想化ソフトです。最新版では、Windows 10およびmacOS Sierraに対応しています。また、20以上のツール・ユーティリティをまとめた「Parallels Toolbox for Mac」を搭載。「Pro Edition」では、DockerやChef、Jenkinsといった開発ツールにも対応。パッケージ版（1年期間更新ライセンス）をプレゼントします。

提供元 パラレルス https://www.parallels.com/jp

02 8BITDO DPAD USB HUB 2名

レトロゲーム機を彷彿とさせる十字ボタン風デザインのUSBハブです。3つの出力ポート（USB2.0/1.1）と1つの入力ポート（microUSB-B）を搭載。PS4、各種パソコンに対応しています。USB-A・microUSB-BのUSBケーブルも付属。

提供元 サイバーガジェット http://www.cybergadget.co.jp

03 Qiita ノベルティTシャツ（Mサイズ） 1名

Incrementsが展開する、プログラマのための技術情報共有サービス「Qiita」（https://qiita.com）のノベルティTシャツです。

提供元 Increments http://increments.co.jp

04 プログラミング Elixir

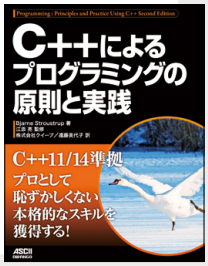
Dave Thomas 著

国内初の、Elixirに関する言語本。言語の基本からElixirの得意分野である並列処理、さらにはマクロといった応用機能まで広く解説しています。著者は『達人プログラマー』執筆の有名Rubyプログラマ。

提供元 オーム社 http://www.ohmsha.co.jp 2名

05 C++によるプログラミングの原則と実践

Bjarne Stroustrup 著



大学におけるプログラミング授業の教科書として構成されており、C++の例題・練習問題を解きながら、プログラミングのスキルを磨けます。総ページ数は1,248、価格は税抜き7,000円。

提供元 アスキードワンゴ http://asciidwango.jp 2名

06 みんなのGo

松木 雅幸、mattn、藤原 俊一郎、中島 大一、牧 大輔、鈴木 健太 著

日本における著名Goプログラマが勢ぞろいして書かれた、Go言語のムック本。実際に開発・運用するうえでのTipsが盛りだくさんです。表紙のキャラクタはGoのマスコット「Go Gopher（ジリス）」。

提供元 技術評論社 http://gihyo.jp 2名

07 [改訂新版]プロのためのLinuxシステム構築・運用技術 中井 悦司 著

Linuxサーバの構築、運用、問題判別に必要な知識と手法を、具体的な設定方法と合わせて解説しています。改訂にあたり、最新版のRHEL 7に対応し、systemdに関する加筆が行われました。

提供元 技術評論社 http://gihyo.jp 2名

第1特集

新人のときに知っておきたかった

クラウドコンピューティングのしくみ

AWS・Azure・SoftLayer・Heroku・さくらのクラウド

「クラウドコンピューティングのしくみを解説できますか？」そんな問いかけが本特集の始まりでした。新入社員の仕事が最初からクラウドというのも、もはや決して珍しい話ではありません。しかし、これだけ普及したにもかかわらず、そのしくみをわかりやすく解説しようとすると意外とできないものです。本特集では、若手エンジニアに向けて最前線で活躍する執筆者がクラウドのしくみ解説を書き下ろしました。これらの知識を深めることで、技術習得の手がかりを得てください。

第1章 猫先生かく語りき

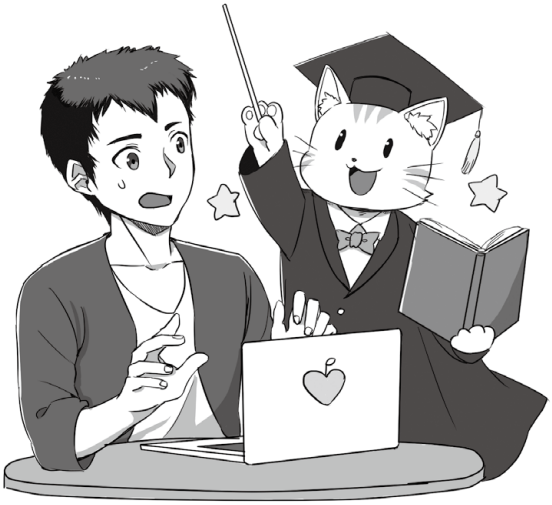
そもそもクラウドって?

Author 五十嵐 綾(いがらし あや)　堀内 晨彦(ほりうち あきひこ)

クラウドコンピューティング導入が進んだ2008年頃に比べ、現在はさまざまなサービスが普及し、より複雑なシステムになってきています。マイクロサービスやサーバーレスという概念まで出てきました。本章では知識の整理をしながら調べていきましょう。登場人物は、とあるクラウドサービス企業に務める新人君。そして猫(?)先生です。

仮想マシンってなんだ?!

――クラウドを使った社内システム構築チームに配属された新人君。ある日曜日の昼下がり。カフェでクラウドについて勉強を始めました。するとどこからか猫が現れて……。



うーん。会社でクラウドサービスを使うことになったんだけど、何から手を付ければいいのかなあ。

どうしたの?

?!　猫が喋った!

さっきから唸っているけど、何か困っているのかな……。

クラウドについて知りたいのだけど、クラウドってなんだかモヤモヤとしたイメージしかなくて。

そうね。君はどのようなものだと思っているんだい?(猫が喋っているというのに、順応が早いな……)

えっと、インターネットの向こうにサーバがあって、それをいつでも作ったり消したりして使える……みたいなもの?

なるほど。クラウドのしくみについて理解が曖昧(あいまい)みたいだから、まずはクラウドサービスの基礎となっている仮想化技術を学ぶのがよさそうね。

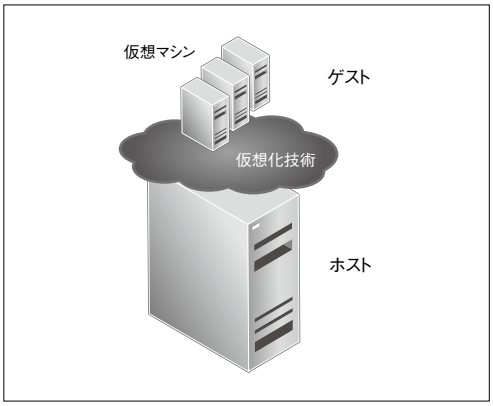
仮想化技術?

そう。一口に仮想化技術と言ってもいろいろあるんだけど、まずは仮想マシンを実現している仮想化技術を知るのが良いかな。

教えてください。よろしくお願いします!

仮想マシンとは、仮想化技術によって1台のマシン上で複数の仮想的なマシンを動かせるようにしたものだよ。

▼図1　仮想マシン（ホストとゲストの関係）




ふむふむ。会社でよく聞くVMってVirtual Machine（仮想マシン）の略なのでしょうか。


そう！


なるほど。しかし……文字だけだと想像しづらいですね。


では、**図1**の仮想マシンのイメージを見てみようか。


ゲスト、ホストという謎の言葉ががが。


これはただの呼び名。一般的に仮想化される側をゲスト、仮想化を提供する側をホストと呼ぶ。よくゲストOSという言葉が使われるけど、ゲスト側の仮想マシン内で動くOSのことを示すの。

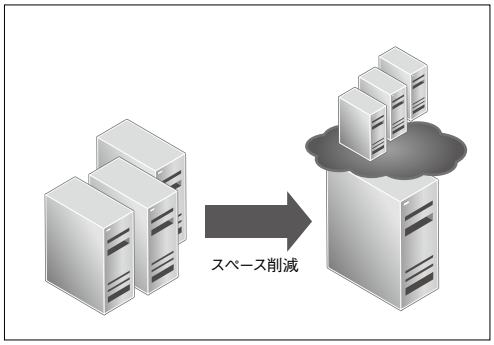

そういうことなんだ。じゃ、仮想化を提供する側の物理マシンは、ホストマシンと呼ばれているの？


正解！

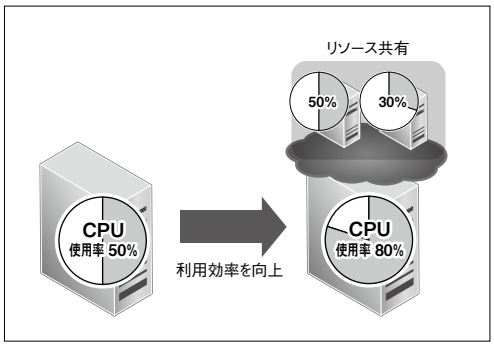

やった！　仮想化技術によって仮想マシンが実現されることはわかってきました。ただ、なぜクラウドサービスには仮想マシンが利用されているの？


よい質問！　仮想マシンを活用するメリットはいくつかあるけど、新人君、何か思いつくものはある？

▼図2　仮想化技術でサーバの占める場所を節約



▼図3　仮想化技術で利用効率を向上




うーん。1台で複数のマシンが動かせるということは、複数台使用するときと比べてサーバを置く場所や電力の消費を抑えることができるかも（**図2**）。


そのとおり。あとは仮想マシン間で共有してリソース（CPUやメモリ、ディスク容量などのこと）の利用効率を高めることもできるね（**図3**）。


確かに。最近のサーバは性能が良いので、単一用途でリソースを使い切ることはあんまりないとか言われていますね。


ほかにもスナップショットを利用して環境の保存や複製をすることができるよ。


スナップショット？　写真でも撮るの？


一瞬を記録するという点では写真と近いね。スナップショットとは、ある時点で

の仮想マシンのディスク、データや設定の状態を記録したもののこと（**図4**）。

そうだったんだ。じゃあ開発環境を1つ作って、これを複製してチーム全員が同じ環境で開発するということもできそう！　あれ、もしかして任意の時点を記録しているということは、仮想マシンに対して修正を加えたあとも元に戻すことができちゃうの？

そう！　クリーンな状態でスナップショットを撮っておけば、試験を実行していて環境が汚れたとしても、そのつどクリーンな状態に復元できるよ。

それは超便利！

新人君は、サービス公開されているものの前任者が異動してしまったりして、再構築すらできなくなった古いサーバを見たことはないかな？

そういえば思い当たるものが1つ……。サーバのパーツも生産終了しちゃって、先輩が壊れないよう祈っていたなぁ。

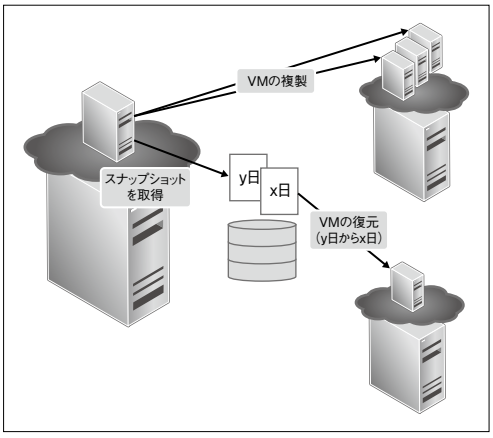
そんなサーバも、仮想マシンを使っていたら救えたかもしれないよ。

なんと！　どんな機能を使うの？

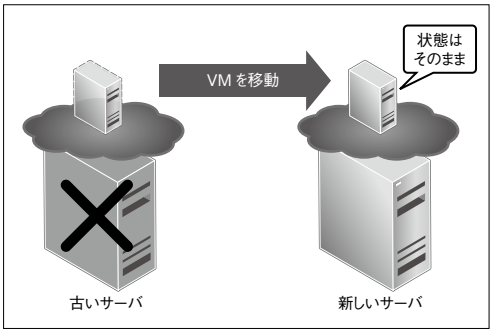
仮想マシンは、別のホストマシンに移動させることができるんだ（**図5**）。サービスが動いている環境……つまり、OSは当時の環境のまま、ホストマシンには最新のサーバを使うということができるよ。ライブマイグレーションという、動作中の仮想マシンを停止させることなく別のホストマシンに移動させる機能もあるね。

仮想マシンってすごい！　もっと詳しく知りたい！

▼図4　便利な仮想マシンのスナップショット機能



▼図5　仮想マシンを引っ越し



仮想マシンのしくみを知ろう

ハイパーバイザの種類にはいろいろある?

では、Hypervisor（以下ハイパーバイザ）の種類について説明しよう。

いきなり知らない単語！

ハイパーバイザとは、仮想マシンを実現するための制御プログラムのことだよ。仮想化モニタ（VMM：Virtual Machine Monitor）とも呼ばれているね。

「ハイパー」ということは「スーパー」とかもある？

一般的にOSのカーネルをSupervisorと呼ぶよ。仮想マシンは、この上位で制御を行うのでハイパーバイザと名付けられたんだ。

なるほど。

では話を戻そう。分類方法はいくつかあるけど、動作方法に着目すると大まかに

2つに分類できる。図6を見てみよう。左がホスト型、右がネイティブ(ベアメタル)型のハイパーバイザの構造を示しているよ。

むむっ。なんだか複雑だなぁ。

単純なので大丈夫だよ。それでは、1つずつ説明していこう。ホスト型は、ホストOS上で動くハイパーバイザだ。Type2 VMMとも呼ばれ、VMwareのWorkstation[注1]などがこれにあたる。

あ、これは使ったことがあります！　デュアルブートのように再起動も必要なく、ホストOSのアプリケーションを起動したまま開発に必要なOSも利用できるので便利。

そう。この型はアプリケーションとして手軽に使えるという利点がある。ただ、ホストOSを経由してハードウェアを操作するために、オーバヘッドが大きくあまり性能がでないという欠点もある。

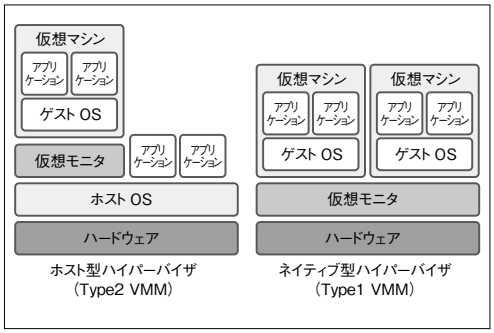
たしかにサーバで仮想化するときは仮想化以外のアプリケーションを起動することは少なそうですし、ハードウェアの処理効率が下がるのが大きく響きそうですね。

そう、そこで登場したのがもう1つのネイティブ型なんだ。この型では、ハイパーバイザ自体がOSを兼ねてハードウェア上で動く。仮想化に最適化されているので処理効率がよくなっているよ。Type1 VMMとも呼ばれていて、CitrixのXen[注2]がこれにあたるね。

ふむふむ。

ハイパーバイザは明確に上記の2つに分けられるわけではないんだ。Linux KVM[注3]は、ネイティブ型に分類されることが多いけど、ゲストOSがホストOS上で1プロセスとして動作するためホスト型とも言われているよ。

▼図6　ハイパーバイザの構造



ちょっと混乱してきました……。

まずはこんな分類があることだけ覚えておいてね。

了解です！

準仮想化と完全仮想化の違い

今度はゲストOSの種類を見ていこう。

ゲストOSはハイパーバイザ上で動くOSのことでしたっけ？

正解だね。このゲストOSは、準仮想化と完全仮想化の2つに分類することができるよ。

準ということはOSに仮想化されてない部分がある？

そうだね。準仮想化とは、ゲストOSのカーネルに手を入れ、仮想化によるハードウェア操作のオーバヘッドを減らしたものなんだ。ネイティブ型の例で挙げたXenはこの準仮想化で実装されているよ。

うーん。OSのカーネルに手を入れるのはたいへんそうですね。

そこが欠点なんだ。完全仮想化では、準仮想化に比べて仮想化処理の負荷は大きくなるものの、ライセンスなどの関係でカーネルに手を入れることが難しいOSも動作させることができる。

注1）URL http://www.vmware.com/jp/products/workstation.html
注2）URL https://www.xenproject.org/
注3）URL http://www.linuxkvm.org/

▼表1　仮想化の種類

	ホスト型	ネイティブ型
完全仮想化	VMware Workstation、Oracle VirtualBox[注4]、QEMU[注5]	Citrix Xen、Microsoft Hyper-V[注6]、VMware ESXi[注7]、Linux KVM
準仮想化		Citrix Xen


なんでも動かせるのはいいですね。


そう、便利なのもあって、Xen以外はほぼ完全仮想化によって実装されているんだ。XenについてもCPUの仮想化支援機能により性能が向上してカーネル側でチューニングを行うメリットが減ったため、完全仮想化をサポートするようになっているよ。


なるほど！　わかってきた気がする！


では、まとめとして各方式に対応するソフトウェアの表1を見てみよう。


それにしても、準仮想化を実装しているものは少ないね。


仮想マシンに限定しているからね。ほかの準仮想化については後ほど説明しよう。


楽しみ！

クラウドサービスって何だろう?

クラウドサービスが登場するまで


サーバの仮想化はわかったけど、本題のクラウドサービスって結局何なんだろう。


理解をスムーズにするために、まずはクラウドサービスが登場するまでの歴史を振り返ってみよう。新人くんは、ホスティングサービスやレンタルサーバって言葉は聞いたことないかな？

注4）URL https://www.virtualbox.org/
注5）URL http://wiki.qemu.org/
注6）URL https://msdn.microsoft.com/jajp/virtualization/hyperv_on_windows/windows_welcome
注7）URL http://www.vmware.com/jp/products/esxi-and-esx.html


ホスティングサービスは会社で契約してるって聞いたことがあります！　レンタルサーバは、先輩がブログを公開するために使ってるって言ってました。


それなら話が早いね。ホスティングサービスは、データセンターなんかにサーバを設置してもらって、メンテナンスやサポート込みで月額いくら、というふうに契約するサービスなんだ。でも、サーバが必要になったときにすぐに使うことができないし、比較的料金も高めなんだよね。


なんか法人向けって感じのサービスですね。


そうだね。レンタルサーバの方は、1台の物理サーバを複数人で共有したり、権限をWebページの公開などに絞って提供されているサービスのことなんだ。新人くんの先輩のように、ブログを公開するのに使われてたりするね。


なるほど！　ここまでは物理サーバ上のサービスでしたけど、仮想化はいつ登場するんですか？

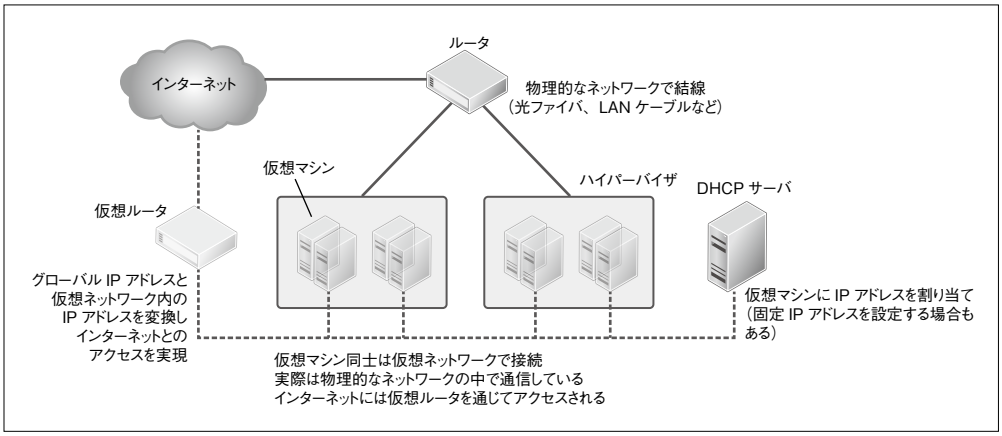

仮想マシンを提供するサービスとしては、VPS（Virtual Private Server）があるよ。レンタルサーバと違って仮想マシン1台を丸ごと提供しているから、いろんなことに使えるんだ。


じゃあ、VPSがクラウドなんですか？


良い質問！　でも厳密に言うと少し違う。クラウドでも仮想マシンは提供されてるけど、もっと柔軟に使えるんだよ。たとえば1時間とか使った分だけ料金を払えば良いし、新しいサーバが欲しかったらすぐに作ることもできるんだ。この「**従量課金**」と「**拡張性**」がクラウドの大きな特徴だね。


なるほど。クラウドでは仮想マシン以外のサービスも提供されているんですか？

▼図7　クラウドとハイパーバイザの関係



そうだよ。作ったアプリケーションを簡単に動かせる環境だったり、インターネットを通じて保存や読み込みができるストレージだったり、数え切れないようなサービスのラインナップがあるんだ。それらを組み合わせることで、今ではクラウドの上で何でもできちゃうよ。

クラウドってスゴいね！　実際には、どんなしくみでできているの？

至れり尽くせりなクラウドのオモテナシ

クラウドが何で、どんなことができるのかはわかったんですが、最初に勉強したハイパーバイザはどのように使われるのでしょうか？

わかりやすいように、クラウドサービスを提供するハイパーバイザがどのように構築されているかを図7にしてみたよ。ハイパーバイザはデータセンタに設置された物理マシン（ホストマシン）にインストールされていて、その上ではユーザが使う仮想マシンが動いているんだ。基本的な使われ方は、手元で動かすときと同じだね。

なるほど。その仮想マシンをユーザはインターネット経由で使うと思うんですが、ネットワークはどのようにつながっているんですか？

良い質問だね。まず、ハイパーバイザはルータと物理的なネットワークでつながっていて、ルータを通じてインターネットにアクセスしているんだ。仮想マシンは、その物理ネットワークの中を通る仮想的なネットワークに接続していて、ほかの仮想マシンや仮想ルータとつながっているんだ。

ネットワークが仮想化されているのはどうしてなんですか？　仮想マシンと同じように、何か便利になることがあるのかな。

仮想ネットワークにすることで、仮想マシンと紐付けて作成したり削除したりできるし、他の仮想マシンとのつながりなんかを柔軟に設定できるようになるんだ。そして、仮想ネットワーク上にはDHCPサーバもあって、仮想マシンにIPアドレスを割り当ててくれるんだ。

インターネットから仮想マシンにアクセスするには、グローバルなIPアドレスが必要って聞いたのですが、そこはどうやって設定しているんですか？

仮想ルータが、インターネットからアクセスできるグローバルIPアドレスと、仮想ネットワーク内部のIPアドレスを変換してくれるおかげで、インターネットから仮想マシンにアクセスできるんだ。サービスを提供している事業者によって実現しているしくみは異なるので、あくまで一例だよ。

なるほど、ハイパーバイザと仮想マシン、インターネットのつながりがよくわかりました！

クラウドサービスで便利になること

そういえば、クラウドサービスにはAPIが付いてるって聞いたんですけど、APIって何ですか？

APIは「Application Programming Interface」で、プログラムの機能を呼び出すための入口のことだよ。少し難しいかもしれないから、具体的な例を出してみようかな（**図8**）。新人くん、クラウドでサーバ100台作ってって言われたらどうする？

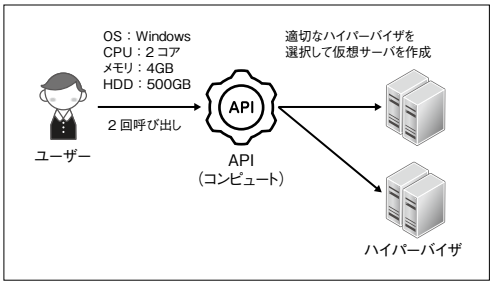
ダッシュボード（クラウドの操作画面）で1台ずつ作る……のは面倒ですね。それとAPIってどう関係してるんですか？

APIに対して、サーバの名前やスペック、IPアドレスやOSなんかの情報を渡してあげると、その情報どおりにサーバを作ってくれたりするんだ。APIは別のプログラムから簡単に呼び出すことができるから、for文のようなループを使ってAPIを100回呼び出してあげれば良いよね。

なるほど！　APIを使えば、自分で作ったプログラムからクラウドを自由に操作できるようになるんですね！　サーバを作る以外にはどんなことができるんでしょうか？

たとえば、新人くんのWebページが人気になってアクセスがたくさんあったとしよう。APIを使えばサーバの状態を知ることもできるから、サーバが重くなってきたら新しいサーバを追加して快適にWebページを見てもらう、なんてことも自動でできるようになるんだよ。

▼図8　クラウドサービスとAPI（Application Programming Interface）



なるほど、APIを使っていろんなことを自動でできるのも、クラウドの特徴なんですね。

クラウドは「組み合わせ」のテクノロジー

ここまでで、ハイパーバイザとネットワークのつながりや、APIって便利な機能があることはわかったんですが、ユーザがサービスを利用するときに、裏側ではどんなふうに仮想マシンが作られてるんでしょうか？

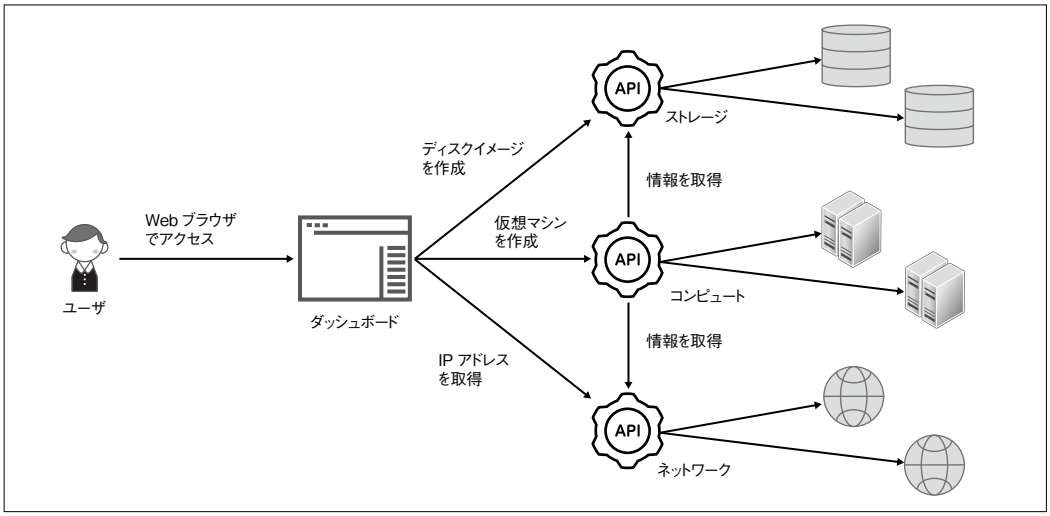
良い質問！　では、仮想マシンを提供するクラウドがどんなふうに動いているのか、詳しく説明しよう。仮想マシンを提供する場合、最低でも次の4つの機能が必要になるんだ（**図9**）。

- ダッシュボード：Webブラウザなどでアクセスする、ユーザが実際に操作するためのクラウドの画面。ダッシュボードから各機能に指示を出して、仮想マシンやストレージ、ネットワークの作成などを行う
- コンピュート：ハイパーバイザ上の仮想マシンを操作するための機能。ハイパーバイザ上にサーバを作ったり消したり、何台もあるハイパーバイザのどこにサーバを作るかを決めたりする
- ストレージ：仮想マシンにインストールするOSのテンプレート（雛形）や、仮想マシンのディスクを保存するためのサービス
- ネットワーク：仮想マシンをインターネットや他の仮想マシンと繋ぐための機能。サーバと同じように仮想化されていることが多い

仮想マシンを作るだけかと思っていたら、こんなに機能が必要なんですね。クラウドの裏側って複雑だ……。

クラウドを提供するためのいろいろな機能は、それぞれ独立して動くようになっていることが多いんだ。APIを使ってお互いの機能を使ったり、情報を取得したりしているんだよ。そうすることで、他のサービスからも簡単に使えたりするからね。こういうのを「マイクロサービ

▼図9　仮想マシンを提供するための4つの機能



スアーキテクチャ」って言うんだよ。

マイクロサービス、聞いたことがあります！でも、やっぱりクラウド作るのってたいへんそうだなぁ……。

簡単に作る方法もあるよ。最近の事業者は、これらの機能をゼロから作るんじゃなくて、「クラウド基盤ソフトウェア」や「クラウドOS」と呼ばれるものを利用することが多いんだ。

クラウド基盤ソフトウェア？

クラウド基盤ソフトウェアでは、サービスに必要な機能一式を提供しているから、簡単にクラウドサービスを作ることができるんだ。代表的なクラウド基盤ソフトウェアに「OpenStack[注8]」があるよね。

OpenStackって聞いたことがあります！クラウドを作るためのソフトウェアだったんですね！

OpenStackは、オープンソースで開発されていて、たくさんの企業が利用しているんだ。

OpenStackを使っているクラウドって、どんなのがあるんです？

Hewlett Packerd Enterprise の Helion Open Stack[注9]や、NTTコミュニケーションズのEnterprise Cloud[注10]があるね。ただ、そのままじゃなくて、一部の機能を各社独自のものに置き換えたりして、オリジナリティを出してるみたいよ。

なるほど、マイクロサービスだとAPIでつながっているから、互換性があれば機能を置き換えることもできるんですね。クラウドのしくみ、なんとなくわかってきました！

いろいろなクラウドサービス

クラウドには大きく分けて「IaaS」「PaaS」「SaaS」の3つがあるよ。**表2**を見ながら、順番に説明するね。

はい、お願いします！

IaaSはInfrastructure as a Serviceの略で、仮想マシンや仮想ネットワークなどのインフラを提供するサービスなんだ。スペックやOSを自由に選べたり、ネットワークの構成を好きなように変えられるなど、自由度が高いのが特徴だよ。

注8）URL https://www.openstack.org

注9）URL http://www8.hp.com/jp/ja/cloud/hphelionopenstack.html

注10）URL https://ecl.ntt.com

▼表2　クラウドの分類

	提供するもの	具体的なサービス
IaaS	サーバ、ネットワーク	Amazon EC2[注11]、GCP Compute Engine[注12]
PaaS	アプリケーション実行環境	Heroku[注13]、Google App Engine[注14]
SaaS	ソフトウェア	Office Online[注15],Google Docs[注16]

▼表3　新しいクラウドの種類

	提供するもの	具体的なサービス
ベアメタル	物理サーバ	Softlayer Bare Metal Servers[注17]、Enterprise Cloud ベアメタル
コンテナ型仮想化	コンテナ実行環境	Arukas[注18]、Google Container Engine[注19]
サーバーレス	さまざまな機能	Amazon Lambda[注20]、Amazon S3[注21]


IaaSはこれまでのホスティングやVPSの進化版って感じですね!


次に、PaaSはPlatform as a Serviceの略で、PHPやRuby、JavaScriptなどで作ったアプリケーションを簡単に動かせるサービスなんだ。普通だと、Webサーバを立てたり、OSの設定をしたり、アプリを動かすための準備が必要だけど、PaaSを使えば作ったプログラムをアップロードするだけで動くんだよ。


とっても便利ですね。PaaSだと、Herokuってサービスが有名だって聞きました。


最後のSaaSはSoftware as a Serviceの略で、ソフトウェアをインターネットを通じて使えるサービスだよ。今までだとお店で買っていたオフィスソフトなんかを、Webブラウザで使えたりするんだ。データはクラウド上に保存されてるし、他の人と一緒に編集することもできる。


もしかして、Microsoft OfficeがWebブラウザで使えるようになったOffice OnlineはSaaSになるんでしょうか。


まさしくSaaSだね。

進化し続けるクラウドサービス


仮想マシン以外のクラウドのしくみについても知りたいです！


じゃ、比較的新しいクラウドの概念をいくつか紹介しよう。**表3**を見ながら解説するね。

ベアメタルクラウド


最初は「ベアメタルクラウド」だ。


ベアメタル？　なんかカッコ良さそうだけど、どういう意味？


直訳すると「剥(む)き出しの金属」っていう意味なんだ。物理サーバを仮想マシンと同じように、簡単に使えるようにしたクラウドサービスだよ。仮想マシンより性能の高いサーバを使えたり、好きなハイパーバイザをインストールしてオレオレクラウドを作ったりすることもできるんだ。


なるほど、物理サーバならば仮想化していない分、効果的に性能を引き出せそうですね。

注11) URL https://aws.amazon.com/jp/ec2/
注12) URL https://cloud.google.com/compute/
注13) URL https://www.heroku.com
注14) URL https://cloud.google.com/appengine/
注15) URL https://www.office.com
注16) URL https://www.google.co.jp/intl/ja/docs/about/
注17) URL https://www.ibm.com/marketplace/cloud/bare-metal-server/jp/ja-jp
注18) URL https://arukas.io
注19) URL https://cloud.google.com/container-engine/
注20) URL https://aws.amazon.com/jp/lambda/
注21) URL https://aws.amazon.com/jp/s3/

コンテナ型仮想化

次は「コンテナ型仮想化」だよ。

コンテナってDockerとかですよね、なんか最近流行ってますよね。

コンテナはハードウェアの動きまで再現する仮想マシンよりも、軽量で高速に動かすことができるんだ。仮想化というより、ホストマシンのリソースの一部を隔離して、コンテナを使っているイメージが近い。たとえば、仮想マシンの占めるディスク容量は数十、数百GBだけど、コンテナのイメージは小さくまとめると数十MBくらいから作ることができるんだよ。

とっても小さいんですね。それならクラウド上のどこにでも持っていけますね。

携帯性の良さだけじゃなくて、「どこでも動く」というのもコンテナの特徴なんだ。たとえばノートパソコンで作ったコンテナでも、イメージをそのままクラウドにアップロードするだけで、同じように動かすことができるんだ。

コンテナはスゴいですね、なんだか使ってみたくなりました！

サーバーレス

最後は「サーバーレス」だ。これは具体的なサービスというより考え方に近いかな。ただ、まだ明確な定義がなくて、フワッとしたものなんだ。

サーバーレスってことは、サーバを使わないってことですか？

「サーバを意識せずに使えるサービス」と言うのが正しいかな。クラウドサービスの種類が増えたことで、わざわざ仮想マシンを作らなくても、いろんなことができるようになったんだよ。たとえ、Webサーバを構築しなくても、ストレージに保存したWebページをそのままWebサイトとして公開できちゃったりする。

何だか難しそうですけど、今まで仮想マシンを作って実現していたことを、いろ

んなクラウドサービスを連携させて実現できるようになったってことなんですね。

そうだよ。ほかにもコンテナを使ったり、PaaSを使ったりするのでもOKだよ。

なるほど、僕にもできそうな気がしてきました。

おわりに

ここまでで仮想化やクラウドのことはわかったかな？

はい！　最初はモヤモヤしていましたが、クラウドのしくみが理解できました！　たくさん勉強しないといけないことがあるのもわかりました……。

では、次章からは実際のクラウドサービスや使い方を紹介していくよ。ここで勉強したことをしっかり思い出しながら、次章以降を読んでみてね。

頑張ります！

――結局、なぜ猫が喋ったのかは気にしないままの大雑把な新人君でした。SD

第2章

これからクラウドする人に教えたい

Amazon Web Servicesのシン・ノウハウ

Author 多田 貞剛（ただ さだよし） ㈱サーバーワークス
Mail tada@serverworks.co.jp

本章では、Amazon Web ServicesについてCler（Colud Integrater）として得られた経験から、その利用方法からシステム移行まで、実際に使用してわかった使い方のコツとそのエッセンスを解説します。まずAWSのしくみを明らかにし、体験から得られたメリット／デメリットをまとめました。そしてクラウドに移行したシステムをいかに運用するか紹介し、サーバーレスアーキテクチャにも触れます。

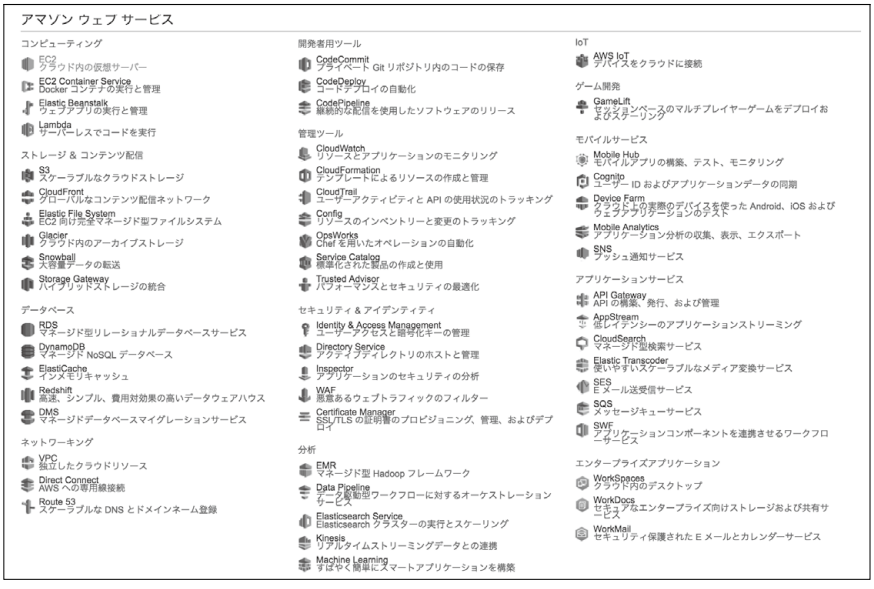
Amazon Web Services のしくみと必須ポイント

AZをまず押さえよう!

Amazon Web Services（以下、AWS）とは、IaaS型のパブリッククラウドです。2016年9月の執筆時点では利用できるクラウドサービスが約70を超えており、利用者の幅広い利用用途に対応できるクラウドサービスと言えるでしょう（**図1**）。

AWSでは、世界各地にデータセンター（以下、DC）を保有し、それらを地域ごとに束ねたものをリージョンと呼びます（**図2**）。リージョン内には地理的に離れた、複数のDCが存在します。AWSでは、これをアベイラビリティゾーン（以下、AZ）と呼びます（**図3**）。AWSでは、利用するサービスに応じてリージョンを選択します。また、サービスによっては、AZを選択しなければ、利用できないサービスもあります。その場合は、サービスを展開するAZを選択していきます。

▼図1　Amazon Web Services（70を越えるクラウドサービス：一部表示割愛）



AWSは何でできているのか?

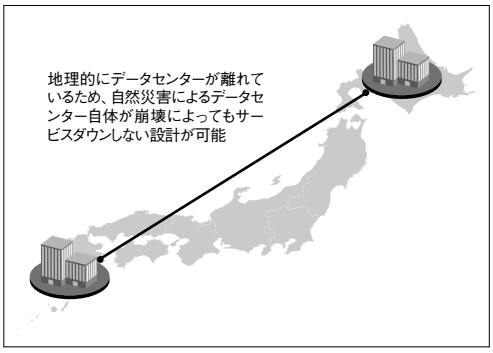
そして、AWSのもう1つの特徴としては、ネットワークからプラットフォームまですべてのレイヤを API（Application Programming Interface）でプログラマブルに制御できることです。SDK（System Developper Kit）、CLI（Command Line Interface）といったツールが提供されており、IaaS型のクラウドですが、インフラエンジニアだけでなく、アプリケーションエンジニアも手軽に利用できます。

そのAWSのしくみについて触れます。AWSの内部では、一部のサービスオプションを除き、AWS社が管理するDC内のインフラリソースをほかの利用者とともに共有する形でサービスが提供されています。たとえば、仮想サーバサービスのEC2（Amazon Elastic Compute Cloud）の仮想化基盤にはXenが使われています。

Xenとは、ハイパーバイザ型の仮想化技術の1つです。特徴としては、物理サーバのCPUやメモリ、ストレージなどのリソースを、各仮想マシンのスペックに割り当てて利用します。そのため、基本的には1つの物理サーバ上にスペックに応じて仮想マシンを複数台展開できます（**図4**）。EC2もほかの利用者とリソースを共有しつつ、利用する形態になります。AWSでは、ほかの利用者とリソースを共有するXenをベースに、独自カスタマイズしたサービスを提供しています。

▼図2　世界各地にデータセンターを配備

▼図3　アベイラビリティゾーンの概念



AWSのメリット／デメリットから見極めるポイント

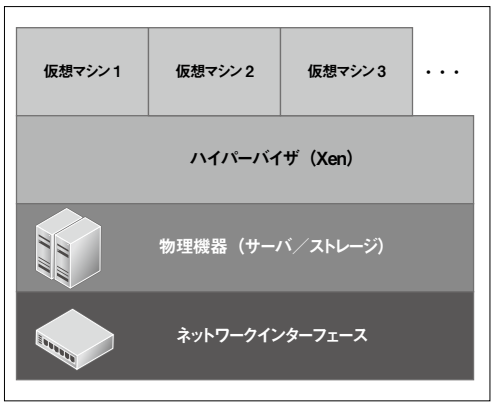
パブリッククラウドのAWSの特徴を概観したので、本項ではより深くAWSのメリットとデメリットについて触れていきます。メリット／デメリットを**表1**にまとめました。

ポイント1「利用するインフラは早めに確保する」

リソースについて**表1**のとおり、AWSでは利用者同士でリソースを共有しながらサービスを利用しています。AWSは膨大なインフラをDCに保有していますが、利用するリソースによっては、まれに枯渇する場合が存在します。

たとえば、EC2の購入オプションには、サーバのスペックに応じた1時間あたりの利用料を支払うオンデマンドインスタンス、余剰分のリソースを利用者が価格をAWSに入札して利用

▼図4　Amazon Elastic Compute Cloudにおける Xenによる仮想化



▼表1　AWSのメリット／デメリット

	メリット	デメリット
リソース	利用者はAWSが提供するインフラをはじめとするリソースを使う	利用者間で、各リソースを共有しているため、枯渇する場合もある
コスト	使用した分だけを課金する従量課金制	毎月の利用状況によって利用額が変動するため、将来の予測がしづらい
拡張性	柔軟なインフラ制御が可能	インフラが拡張／縮小するのにかかる時間を確認する必要がある
機能	利用者のニーズに応える豊富なサービス	新サービスやアップデート内容を追い続けないとAWSを活用した構成を実施できない
セキュリティ	セキュリティが高いインフラ環境が利用可能	セキュリティホールを攻撃される設定をした場合、最悪の場合利用停止が発生する

するスポットインスタンスがあります。これらを利用する際、自分が購入するスペックのサーバをほかの利用者が多数利用している場合、サーバを起動しようとするプロセスで必要なリソースが枯渇してしまい、エラーが発生する場合があります。

そのため、AWSのインフラを確実に利用する方法としては、ほかの利用者よりも先行でリソースを押さえておくと良いです。たとえば、EC2では、リザーブドインスタンスという購入オプションがあります。リザーブドインスタンスとは、利用者が1年もしくは3年先まで利用するサーバのリソースを確保しておくことが可能なオプションとなっています。リザーブドインスタンスを利用することで、利用者は、サーバのリソースが枯渇することなく必要なスペックのサーバを手に入れることができます。

ポイント2「予測は予測にすぎない」

コストについて、AWSはサービスを使った分だけ課金する従量課金制が基本的な課金モデルとなっています。つまり、当月の利用料は、どれくらい月間にクラウドサービスを利用したかで変動します。したがって、AWS利用料を見積もることはあくまでも予測にすぎません。そのため、AWS Billing and Cost Managementサービスで日々の利用状況を確認できるので、利用状況に応じて月間の利用料の予算に近づける対応を検討する必要があります。

たとえば、利用料の抑制が目的の場合、EC2は1時間当たりの課金が行われるため、利用する時間帯が日中帯ならば、夜間帯はEC2を停止することで利用料を抑えられます。また、利用しなくなったサービスリソースは削除することで利用料削減に寄与できます。筆者の場合、EC2に固定のグローバルIPアドレスを付与するElastic IP（EIP）をサービスリソースとして確保したまま、EC2に紐付けずに放置していたことがあります。その結果、利用料が嵩んでいました。利用しないリソースは削除しておきましょう。また、利用料の閾値（境目となる値）を設定して利用者に対して通知を行う、Billing Alertを設定することも対策になります。閾値の利用料に到達した際に、不要なサービスを利用していないかどうかを見直すことによって、月間の利用料を抑えることにつながります。

ポイント3「自動拡張／縮小のタイミングの確認」

拡張性について、AWSではインフラサービスの柔軟な制御ができます。たとえば、キャンペーンサイトへの膨大なアクセスや、メディアに取り上げられたことで、通常では想定できない急激なWebサイトへのアクセス増加に対応するために、AWSでは、インフラを拡張するサービスや機能が存在します。

代表的なサービスとしては、Auto Scalingがあります。このサービスは、特定の条件のときにEC2の台数を増減させるもので、サーバの可用性を高めます。

たとえば、CPU使用率が60%を上回ったと

きに、1台増やすことや、CPU使用率が40%を下回ったときに、1台減らすといった制御が可能なサービスとなります。したがって、急激なEC2へのアクセス増加に対して、サーバの台数を増やし、ほかのサーバへもアクセスを分散させていくような対処を行うことで、サービスダウンを防ぐインフラの拡張ができます。

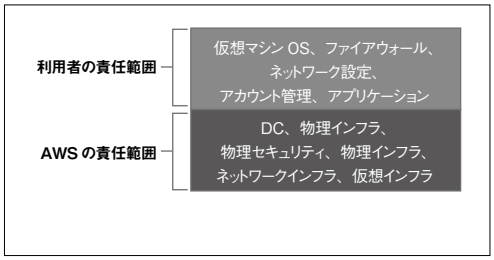
他方、Auto ScalingはEC2の増減に時間を要するサービスです。上記の例のように、CPU使用率が閾値に達したからといって瞬時に増減するわけではありません。EC2の増減するタイミングは、利用するスペックやサーバの環境、閾値の条件に依存します。そのため、本稿の冒頭で触れた具体例のようなシステムで利用するときに、必要とする時間内で増減するかの確認をしましょう。本番稼動する前に必須です。

ポイント4「サービスのアップデートのキャッチアップ」

機能について、AWSが提供するサービスは約70を超えています。新規サービスやアップデートは頻繁です。たとえば、2013年は280個、2014年は516個、2015年は722個の新規リリースがあり、年々増加傾向にあります。さまざまなアップデートがなされていくなかで、利用し始めたころには存在しなかったサービスや機能が追加されることもあります。従来は、外部サービスやAWS内でも利用者の管理負担があったサービスからAWS内で完結するものやAWSが管理／運用する、マネージドサービスに置き換わったことで、利用者の運用管理負担が減ることがあります。

幅広い顧客ニーズに応えるクラウドサービスだけに、新規サービスやアップデートのキャッチアップが顧客ニーズに応える機会につながります。AWS上でシステムが安定稼働しているからといって慢心してはいけません。どうすればよりクラウドの機能を活かした使い方ができるのか、日頃から意識して情報収集をしましょう。

▼図5　責任共有モデル



ポイント5「自分の環境は自分で守る」

セキュリティについて、AWSでは、SOC、PCI DSSなど高いセキュリティ性が認められたインフラ環境を利用できます。AWSがセキュリティの責任を負う部分と、利用者がセキュリティの責任を負う部分が分かれています。このような考え方を責任共有モデルと呼びます（**図5**）。責任共有モデル内では、AWSはサービスを支える、DCの施設自体や物理インフラ、物理セキュリティ、Xenの仮想化基盤のセキュリティに関して責任を負っています。Xenのセキュリティ上の脆弱性が発生した際、ハイパーバイザ層のアップデートのため、EC2を再起動する対応を行ったことからもAWSがサービスの根幹部分に責任を負っていることがうかがえると思います[注1]。

他方、利用者は、利用するサービス内のセキュリティに責任を負います。そのため、サービス内のセキュリティが万全かのチェックを利用者が怠ることはできません。セキュリティ設定が甘い設定の場合、その分セキュリティホールを攻撃されやすくなります。攻撃された結果、ほかの利用者へセキュリティの危険性が及んだ場合、AWSの利用停止が行われる場合があるので、注意が必要です[注2]。

注1）URL http://aws.typepad.com/aws_japan/2014/09/ec2mente.html
注2）URL https://aws.amazon.com/jp/agreement/

AWSへのシステム移行

筆者が勤めるサーバーワークスは、AWSに特化したクラウドインテグレーター(CIer)です。幸いなことにAWSの移行／構築案件を多くいただいています。本稿では、筆者が経験したAWSの移行／構築のポイントを紹介します。

ポイント1「最初は小さく始めて、次第に大きくする」

既存システムや新規開発するシステムをAWS上に構築して運用していきたいという要望が最も多いのですが、弊社では、とくにお客様のご要望のスペックが厳格でなければ、最小のスペックかつ最小のサーバ台数で環境を構築します。その理由は、AWSでは運用していく中でサーバのリソース拡張(**図6**)や、台数を増やす(**図7**)などの柔軟なインフラ制御が可能なためです。

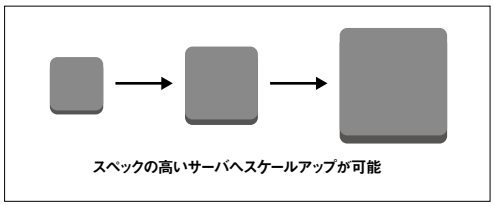
また、利用当初からスペックの高いサーバを選択したものの、想定よりもスペックが低くても運用に耐えられる場合もあります。スペックが高い分課金も高くなるため、初めは小さなものを用意し、アプリケーションを載せたあとに様子を見て上位のサーバへ変更することを検討するのが費用対効果的にも良いです。お客様にシステムを引渡したあと、サーバスペックが問題となれば、スペックをより上位のものに変更を行います。

筆者も実際に最小のスペックで構築後、アプリケーションの試験中にリソースが不足してしまったことがありました。その際は、お客様より必要なマシンスペックをヒアリングし、その要望に適うインスタンスタイプへとスケールアップを行いました。なお、インスタンスタイプの変更を行う際、EC2はサーバの停止が発生するため注意が必要です。

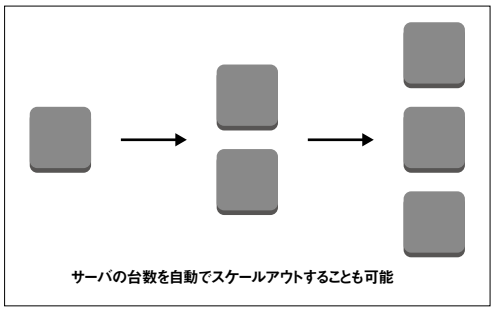
また、サーバのスペックではなく、台数を増やすことでシステムの可用性を高める要望にも対応できます。この場合サーバ自体のスペックは変更しません。また、サーバのスペックを変更するときと異なり、EC2の停止は不要であらかじめ準備したマシンイメージからAuto Scalingを使って台数を増やします。急激なアクセスやあらかじめ予想できる大量アクセスに対して、サーバ台数を増やしてシステムダウンを防ぐときに有効になります。

どちらを実施するかは、お客様の要件との兼ね合いになりますので、弊社ではお客様と協議していく場合が多いです。

▼図6　スケールアップ



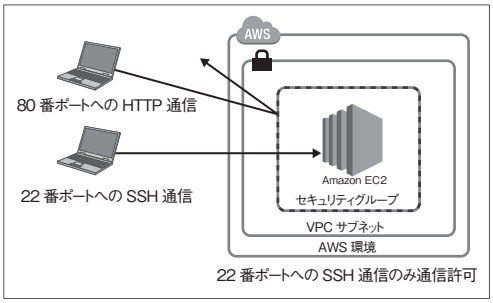
▼図7　スケールアウト



ポイント2「セキュリティを高める」

AWSでは、利用者と事業者が双方でセキュリティに責任を負う責任共有モデルを採用しています。そのため、利用者は自分が使う環境のセキュリティを高め、システムを脅威から守る必要があります。弊社では、ネットワークのセキュリティについては、Virtual Private Cloud(以下、VPC)サービスで、アカウントのセキュリティは、Identity and Access Management(以下、IAM)サービスで対策を行います。また、AWS環境全体のセキュリティ対策については、AWS CloudTrail(以下、CloudTrail)サービ

▼図8　Virtual Private Cloudによるアクセス制御



スやAWS Config（以下、Config）サービスを利用しています。

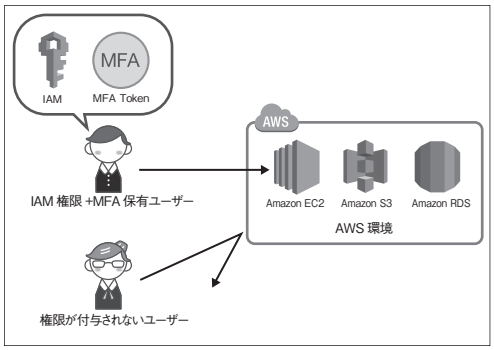
VPCは、通信元のIPアドレス、ポート、サブネットを絞って通信可能なトラフィックを制御します（**図8**）。これにより、VPC内に配置されたサービスのネットワークレベルでのセキュリティを高めることができます。弊社で行う設定では、接続できるIPアドレスとポート番号の範囲を広く設けず、できる限り絞った設定を行います。

たとえば、お客様の会社で全国にさまざまな支社が存在し、その支社から通信を許可してほしいというご要望が多いです。このときは、支社ごとにインターネットと通信するエンドポイントのIPアドレスと使用するポート番号の設定を行うことで、接続できる通信元を絞ることができます。

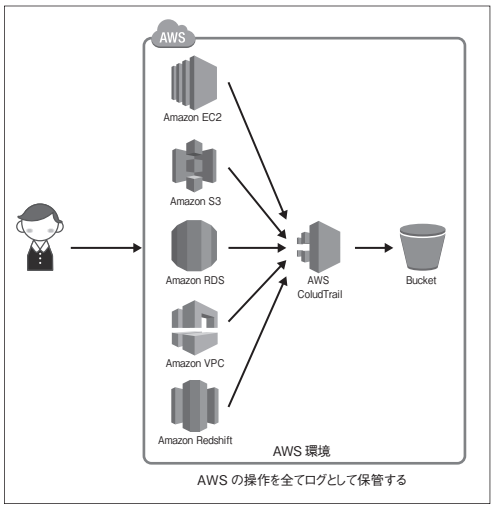
IAMは、利用者がアクセスできるサービスリソースに必要な最小限の権限しか付与しない権限管理を行うことができます。また、AWSマネジメントコンソール（管理画面）へログインし、操作できる利用者のIPアドレスを絞ることや、AWSマネジメントコンソール多要素認証を導入するなどでAWSへアクセス可能な利用者を絞ってセキュリティを高めます（**図9**）。

弊社のお客様でIAMの権限を管理する場合、お客様が実行したいAWSにおける操作を確認し、その操作に対応するIAMの権限を付与するようにしています。そうすることによって不要なAWS環境への操作を防ぐことができます。

▼図9　AWSマネジメントコンソールでのユーザアクセス制御



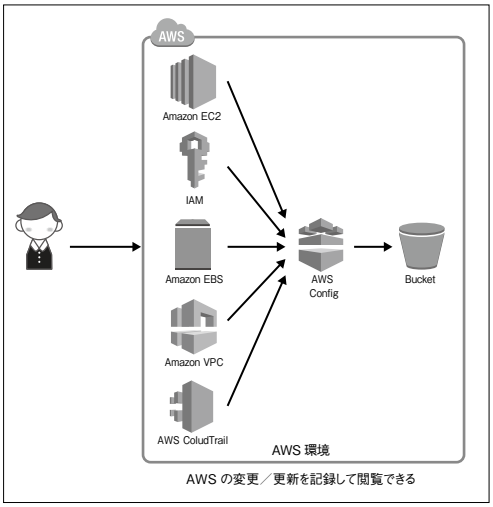
▼図10　CloudTrailによるログ管理



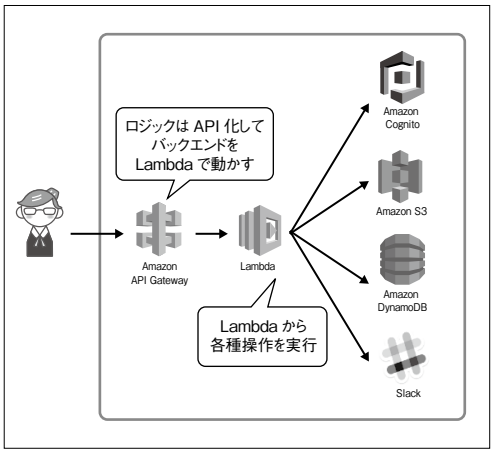
CloudTrailは、AWSにおけるすべての操作をログとして保管できます（**図10**）。また、ConfigはCloudTrailほど対応しているサービスの範囲は広くないですが、サービスの変更や更新があった場合の構成管理に役立つサービスです（**図11**）。

弊社では、CloudTrailもConfigもすべてのリージョンに対して設定の有効化を行っています。その理由は、AWSアカウントを乗っ取られた場合やトラブルシューティングを行う場合、どの時点の操作に問題があるのか、AWS環境に対してどんな操作を行ったから問題なのかを追跡するために役立つためです。

▼図11　Configによる構成管理



▼図12　サーバーレスアーキテクチャ（AWS Lambdaによる例）



筆者の体験としてもお客様からの問題事象に対する問い合わせでCloudTrailのログを活用して原因を特定できた経験があるので、CloudTrailとConfigの有効化を行うことでAWS環境全体のセキュリティ向上に資すると考えます。

サーバーレスアーキテクチャ

近年、AWS Lambda（以下、Lambda）の登場によりサーバーレスアーキテクチャという考え方が広まっています（**図12**）。

サーバーレスアーキテクチャとは、利用者はサーバを管理せず、マネージドサービスを活用してシステムを構築することです。Lambdaは、利用者が実行する処理をコードで制御し、コードを実行する条件となるトリガーイベントが発生したときだけ処理が走ります。たとえば、Lambdaにはcron機能があるため、スケジュールのジョブ実行ができます。設定した時間になったとき（トリガーイベント）、Slackに対してお知らせを投稿するといった処理を行うことができます[注3]。Lambdaでは、サーバの管理はAWSが行うため、利用者のサーバ管理が不要となります。

また、利用者はサーバの運用負担軽減やシステムでの考慮点が減り、ビジネスロジックの開発に集中できるのもメリットです。処理を実行するトリガーが決まっているものはLambdaにオフロードして、そのほかのシステムの開発／運用に注力できるのがサーバーレスアーキテクチャの魅力です。

まとめ

ここまでAWSはどんなクラウドサービスなのか、どんなメリット／デメリットがあるのか、AWSを使ううえでポイントになる事項について触れてきましたが、いかがでしたでしょうか。

AWSがサービス開始したのが2006年で、今年で10年目の節目になります。近年、サーバーレスアーキテクチャを始め、IoTや機械学習といった話題のトピックも使った分だけ課金され、不要な場合はリソースを破棄できるAWSだからこそ始めやすいと思っています。また、AWSには、これらに対応するサービスがそろっています。IT業界の最先端にAWSを使いながら、チャレンジしてみるきっかけに本記事が少しでも役立てば幸いです。SD

注3）URL http://blog.serverworks.co.jp/tech/2016/08/31/lambda-point/

第3章

Linuxが動く！ RedHatが動く！ オープンソースとの親密度を深めるMicrosoft Azureのいま

Author 戸倉 彩（とくら あや） 日本マイクロソフト(株) デベロッパーエバンジェリズム統括本部 テクニカルエバンジェリスト
Blog https://blogs.msdn.microsoft.com/ayatokura/ Twitter @ayatokura

本章ではMicrosoft Azureについて解説します。Windowsエコシステムとして使う利点はもちろんのこととして、各種Linuxディストリビューションをはじめ、オープンソースソフトウェアの利用環境も整っています。選んですぐ使える豊富なサービス群や、オンプレミス環境を視野に入れた開発中のAzure Stackも注目です。

はじめに

こんにちは。皆さんは「Microsoft Azure（読み方：マイクロソフトアジュール）」のことをどこで知りましたか？　気になっていた人はもちろん、名前しか知らない人もいるかと思います。本章でMicrosoft Azureの特徴や構成を把握しながら、サービスの使いどころをイメージしてみてください。

そもそも「Azure」とは何でしょうか？　英語では「空の青い色」という意味です。無限に広がる大空のように、広大なスケーラビリティと高い柔軟性を持つクラウドコンピューティングを、必要なときに必要なだけ使えるマイクロソフトのパブリッククラウドサービスがMicrosoft Azure（以降、Azure）です。

大規模パブリッククラウドサービス

Azureは、マイクロソフトが運用管理するデータセンターのネットワークやクラウドコンピューティングによって支えられています。執筆時現在（2016年9月）、Azureデータセンターは発表されているものも含めると世界各国34ヵ所のリージョンに設置されており、そのうち26のリージョンが一般向けに運用されています。日本国内では、東日本と西日本の2つのリージョンを利用することができます。Azureリージョンと実際のデータセンター所在地の関係については**図1**をご覧ください[注1][注2][注3]。

大規模なクラウドだからこそ実現できることがあります。たとえば、日本国内では東日本と西日本の2つのリージョンを組み合わせることで、システムの冗長構成はもちろんのこと、大規模災害などに備えたディザスタリカバリを国内に閉じて構成することが可能になります（**図2**）。Azureストレージのデータは、ハードウェア障害が発生した場合でもサービスレベルアグリーメント（SLA）[注4]を満たすように、同じデータセンター内でコピーを自動的に3つ保持するようになっています。これをローカル冗長（LRS）といいます。

しかしながら、大規模災害などでデータセンター全体に障害が発生した場合は、SLAの未達はおろかデータ消失のリスクにつながります。これを回避するために、Azureでは設定を有効

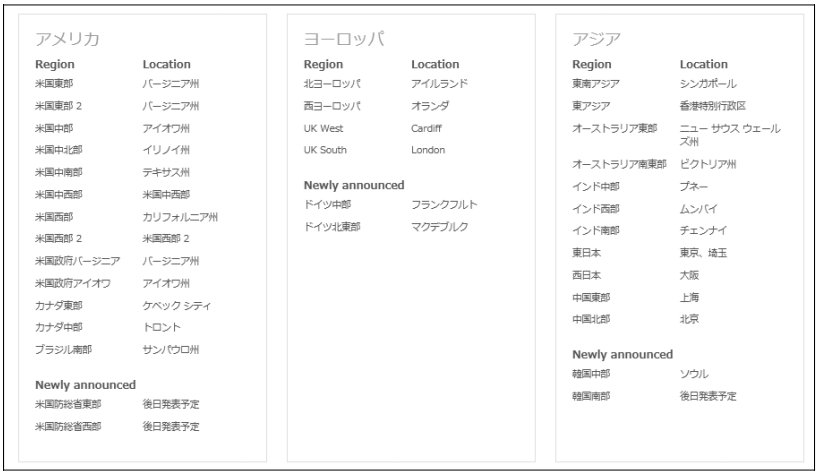
注1） 各リージョンの詳しい情報については「Azure regions」ページに掲載されています。URL https://azure.microsoft.com/ja-jp/regions/

注2） 各リージョンによって提供されているサービスが一部異なります。「Products available by region」ページで最新情報を確認できます。URL https://azure.microsoft.com/ja-jp/regions/services/

注3） 各リージョンの稼働状況は「Azureの状態」ページで情報を一般公開しています。URL https://azure.microsoft.com/ja-jp/status/#current

注4） AzureのSLA適用には各サービスごとに条件が設定されています。詳細は「サービスレベルアグリーメント」ページで確認できます。URL https://azure.microsoft.com/ja-jp/support/legal/sla/

▼図1　Azureリージョンとデータセンター所在地



アメリカ

Region	Location
米国東部	バージニア州
米国東部 2	バージニア州
米国中部	アイオワ州
米国中北部	イリノイ州
米国中南部	テキサス州
米国中西部	米国中西部
米国西部	カリフォルニア州
米国西部 2	米国西部 2
米国政府バージニア	バージニア州
米国政府アイオワ	アイオワ州
カナダ東部	ケベック シティ
カナダ中部	トロント
ブラジル南部	サンパウロ州

Newly announced

米国防総省東部	後日発表予定
米国防総省西部	後日発表予定

ヨーロッパ

Region	Location
北ヨーロッパ	アイルランド
西ヨーロッパ	オランダ
UK West	Cardiff
UK South	London

Newly announced

ドイツ中部	フランクフルト
ドイツ北東部	マクデブルク

アジア

Region	Location
東南アジア	シンガポール
東アジア	香港特別行政区
オーストラリア東部	ニュー サウス ウェールズ州
オーストラリア南東部	ビクトリア州
インド中部	プネー
インド西部	ムンバイ
インド南部	チェンナイ
東日本	東京、埼玉
西日本	大阪
中国東部	上海
中国北部	北京

Newly announced

韓国中部	ソウル
韓国南部	後日発表予定

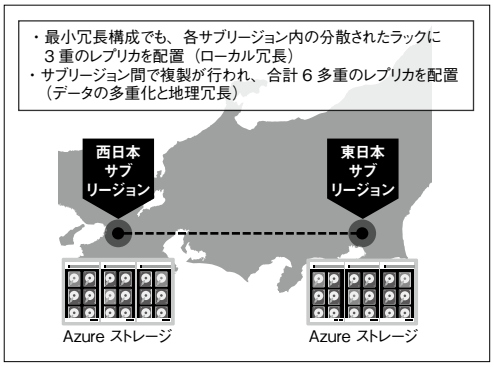
にすることで、地理冗長ストレージ（GRS）を使用できます。GRSは、LRSに加え、固定ペアとなっているデータセンターにも非同期にレプリケーション（複製）されたデータを3つ保持します。プライマリ拠点で障害が発生した場合、ストレージはセカンダリ拠点にフェールオーバーされ、プライマリ拠点のデータセンターのデータが保証されるしくみになっています。

日本国内の場合、東日本データセンターにシステムを配置しつつストレージをGRSに設定すると、西日本データセンターがセカンダリ拠点としてデータ複製を持つため、日本からデータが海外に出ることを防ぎながらディザスタリカバリ対策ができるということになります。

OSSへの取り組み

より多くの人が、より多くの環境を利用するのに伴い、データの運用性と統合性が今まで以上に重要になってきました。マイクロソフトは、多様なニーズにも対応できるよう標準化を推進する組織や団体との協業に取り組んでいます。開発者はAzure上に、広範なオープンソースのプログラミング言語やフレームワークを使ったアプリケーションやサービスを構築することができます。ほかにも仮想マシン上では、

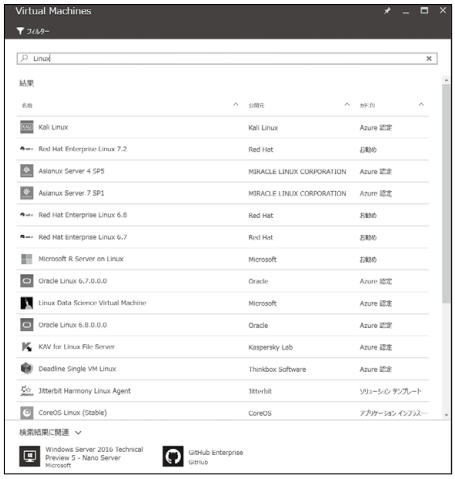
▼図2　可用性を標準装備したストレージ



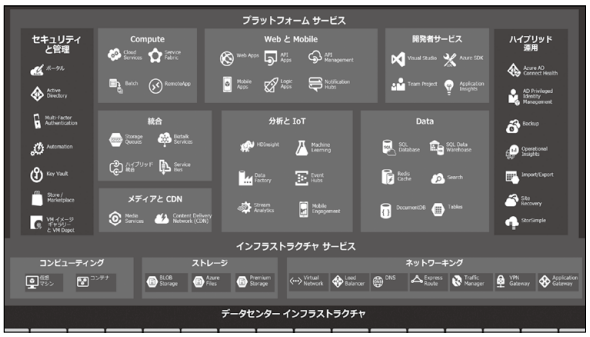
Windows Serverに加えてRed Hat Enterprise LinuxなどのLinuxも選択できます。現在、Azure上の仮想マシンの約1/3ではLinuxが利用され、Azure Marketplaceでは1,000以上のLinuxイメージが公開されています（**図3**）。

たとえば、Azureポータルから「Marketplace」の中で「Linux」を選択すると、Azure仮想マシンとして展開できるLinuxディストリビューションが表示されるので、自由に選択して利用することができます。Red Hat社のRed Hat Enterprise Linux、Oracle社のOracle Linux、ミラクル・リナックス社のAsianux、SUSEのSUSE Linux Enterprise Server、Canonicalの Ubuntu Server、OpenLogic の CentOS、credativのDebianなど、Linuxのイメージは

▼図3　MarketplaceでLinux系仮想マシンを表示



▼図4　Azureクラウドプラットフォームの全体像



多くのパートナーから提供されており、マイクロソフトはさまざまなLinuxコミュニティと協力して、動作保証済ディストリビューションの一覧をより一層充実させようと努めています[注5]。

これらを選択できると聞いて、驚いた人もいるかもしれませんね。ギャラリーから選択できないディストリビューションの場合も、Linuxオペレーティングシステムを格納した仮想ハードディスク（VHD）を作成し、それをAzureへアップロードすることで、いつでも独自のLinuxを展開することができます。ディストリビューションごとに、サポートしているバージョンや利用が無償のものと有償のものがありますので、実際に利用する際にはホームページなどで最新情報を確認するようにしてください。

Microsoft Azureの全体像

ここからはAzureの概念に関するお話です。Azureは、①データセンターインフラストラクチャ②インフラストラクチャサービス③プラットフォームサービスの3つの層に大別できます（**図4**）。Azureでは、さまざまなサービスが用意されているので、何か新しいシステムを作りたい場合には、ゼロベースから考えるのではなく、Azure上で使えるサービスは何か、を考えることで効率的にシステムが構築できるのではないでしょうか。

データセンターインフラストラクチャ

Azureデータセンター内には、土台となる物理的なデータセンター基盤となるデータセンターインフラストラクチャが配置されています。Azureは継続して毎月どんどん新しいサービスが登場してくるので、必要に応じて新しいハードウェアを追加するだけでなく、定期的なハードウェアリフレッシュにより故障したハードウェアの交換や、最新ハードウェアプラットフォームへの移行も行われています。

インフラストラクチャサービス

データセンターインフラストラクチャの上には、インフラストラクチャと呼ばれる基盤レベルのサービスが実装されています（**表1**）。この部分を使うことで、自社内のインフラ環境と同じようなかたちの環境をAzure上に再現することができます。

プラットフォームサービス

プラットフォームサービスは、インフラストラクチャサービスをビルディングブロックとし

注5） マイクロソフトは、オープンソースの開発にどのような規模感で携わっているのかを「Open source software on Azure」でリアルタイムに情報公開しています。
URL https://azure.microsoft.com/en-us/overview/open-source/

▼表1　インフラストラクチャサービスで提供されるサービス概要

サービス	概要
コンピューティング	
仮想マシン	Windows と Linux Virtual Machines を数分でクラウド上に配置できる IaaS
コンテナ	Docker ベースのツールを使用してコンテナを配置したり管理するためのサービス
ストレージ	
Blob Storage	Binary Large Object の略。文書や画像、動画などの一般的なファイルの保存が可能
Azure Files	標準的なサーバメッセージブロック(SMB)プロトコルを使用してファイル共有を提供
Premium Storage	高負荷の I/O ワークロードのためのディスクサポート
ネットワーキング	
Virtual Network	Azure 上に仮想ネットワークを作成できるサービス
Load Balancer	ネットワーク負荷を自動的に分散するために利用
DNS	DNS ドメインを Azure にホストするためのもの
Express Route	Azure とオンプレミスデータセンター間に高速かつ安全なプライベート接続を実現
Traffic Manager	トラフィックの負荷分散
VPN Gateway	Azure 仮想ネットワークとオンプレミス間や、Azure 内の仮想ネットワーク間(VNet 間)のネットワークトラフィックを送信
Application Gateway	アプリケーションレベルのルーティングおよび負荷分散サービス

て使い、目的に特化した各種サービスをあらかじめ Azure で構成して提供するというサービス群です(**表2**)。よく使われるサービスとして Web やモバイル向けサービスや、最近ではデータの分析などに使われる機械学習、IoT などもあげられます。また、大切なデータをしっかり格納するための分散ストレージ、あるいはデータベースも多く使われています。

▼図5　Azure ポータル

Azure アカウントと管理

Azure を利用するためには、Azure サブスクリプションが有効になっている Microsoft アカウントが必要です。Microsoft アカウントを持っていない場合には新規に作成します。その後、利用を希望する Azure プランより Microsoft アカウントとの紐づけを行うことでサブスクリプションを有効にします。Azure を初めて試してみたい方は「1か月無償評価プログラム」、開発者の方は開発ツールやトレーニングと合わせて12ヵ月にわたって Azure 無料枠を利用できる「Visual Studio Dev Essentials」を活用いただくことをお勧めします。

Azure の操作は Web ブラウザを使った「Azure ポータル」(**図5**) または、Azure コマンドから行います。Windows 環境の場合は PowerShell、Mac や Linux 環境の場合には Azure コマンドラインツールを公式サイトからダウンロードできます。

Azure ポータルを使用すると、利用する各サービスはリソースとして取り扱われ、さらにそれらをグルーピングすることでリソースグループという単位で管理します[注6]。

マイクロソフトの仮想化技術

マイクロソフトは、さまざまな仮想化技術を

注6) 現時点ではすべてのサービスで Azure ポータルまたはリソースを管理するためのリソースマネージャがサポートされているわけではないため、一部のサポートされていない場合には、旧 Azure ポータルの「クラシックポータル」を使用する必要があります。Azure ポータルの各サービスの対応状況は「Azure ポータルの可用性チャート」サイトで確認できます。URL https://azure.microsoft.com/ja-jp/features/azure-portal/availability/

▼表2　プラットフォームサービスで提供されるサービス概要

サービス	概要
Compute	
Cloud Services	アプリケーションをホスティングできるPaaS
Service Fabric	マイクロサービスが構築できるPaaS
Batch	大規模な並列／バッチの実行のときに利用
Remote App	Windowsクライアントアプリをクラウド上に配置し、あらゆるデバイスで実行
WebとMobile	
Web Apps	Webアプリを短期間に作成して配置
Mobile Apps	モバイルに特化したmBaaS。Windows、Android、iOS、Xamarin、Cordovaに対応
API Apps	簡単操作によるクラウドAPIの作成と利用
Logic Apps	ビジネスプロセスを自動化
API Management	APIとマイクロサービスを保護、発行、および分析するためのスケーラブルなAPIゲートウェイを提供
Notification Hubs	バックエンドからモバイルにプッシュ通知を配信
開発者向けサービス	
Visual Studio	マイクロソフトの開発ツール(IDE)
Team Project	チームプロジェクト
Azure SDK	Azure用のSDK
Application Insights	Webアプリおよびサービスにおける問題の検知や診断
統合	
Storage Queues	非同期なメッセージ配信に用いられるストレージ
ハイブリッド接続	エンタープライズとクラウドをシームレスに統合
Biztalk Services	Azure BizTalk Servicesの機能の一種
Service Bus	プライベートとパブリックのクラウド環境間での接続
メディアとCDN	
Media Services	大規模にビデオおよびオーディオをエンコード、ストリーミング配信
Content Delivery Network (CDN)	高帯域幅のコンテンツを配信
分析とIoT	
HDInsight	ApacheのHadoopソリューションをクラウドに移行するHadoopベースのサービス
Data Factory	データ変換と移動の整理と管理
Stream Analytics	リアルタイムストリーム処理
Machine Learning	クラウドベースの機械学習
Event Hubs	1秒間に何百万ものイベントを取り込み処理
Mobile Engagement	モバイルの使用率、継続率の増加をサポート
Data	
SQL Database	SQLデータベース
Redis Cache	Redis CacheベースのAzureアプリケーション専用のキャッシュ
DocumentDB	管理されたサービスとしてのNoSQLドキュメントデータベース
SQL Data Warehouse	大量並列SQL Server処理アーキテクチャに基づいたエラスティックデータウェアハウス
Search	完全に管理されたサービスとしての検索
Tables	非リレーショナルデータストレージ
セキュリティと管理	
ポータル	Webベースの管理ポータル
Active Directory	クラウドベースのActive Directory
Multi-Factor Authentication	高度な認証により、データとアプリへのアクセスを保護
Automation	プロセス自動化でクラウド管理を簡素化
Key Vault	キーやその他のシークレットを保護し、制御を維持する
Store / Marketplace	Azureソリューションを提供
VMイメージギャラリーとVM Depot	仮想マシン用イメージ
ハイブリッド運用	
Azure AD Connect Health	オンプレミスのIDインフラストラクチャと同期サービスを監視
AD Privileged Identity Management	組織内のアクセス権を管理、制御、監視
Backup	クラウドへのシンプルで信頼性の高いサーババックアップサービス
Operational Insights	オンプレミスのIDインフラストラクチャと同期サービスを監視
Import/Export	大量にデータを安全にAzureへ転送
Site Recovery	プライベートクラウドの保護と回復の調整
StorSimple	ハイブリッドのクラウドストレージサービス

提供していますが、今回はAzureと最も関係が深い仮想化技術「Hyper-V」についてご紹介していきます。Hyper-Vとは、Windows Serverだけでなく現在Windows 8以降のクライアントOSの機能としても実装されている仮想化システムで、1台の物理的なコンピュータ上に複数の仮想的なコンピュータを配置し稼働させることができます。

Windows OS上にHyper-Vの仮想マシンを作成する前には、Hyper-Vを有効にする必要があります。Windows Serverの場合は「役割と機能の追加ウィザード」から、Windowsクライアントの場合は「Windowsの機能の有効化または無効化」から手動で設定を行うか、PowerShellを使用して有効にすることもできます。手元にWindowsマシンがある場合は試しに利用してみると良いかもしれません[注7]。

Hyper-VとAzureの関係

Azureインフラストラクチャサービスは、Windows ServerのHyper-Vの仮想化基盤をもとに構築されています（**図6**）。すなわち、皆さんがオンプレミスで仮想化を実現するために用いることができるHyper-Vと同じ技術がクラウド上でも活用されているというイメージを持っていただけると、少し身近に感じていただけるのではないでしょうか。Hyper-Vという共通の仮想化テクノロジが使われているメリットとして、オンプレミスとクラウドを連携したハイブリッド化や、オンプレミスからAzureへの移行がしやすいことが挙げられます。

Azure仮想マシン（IaaS）

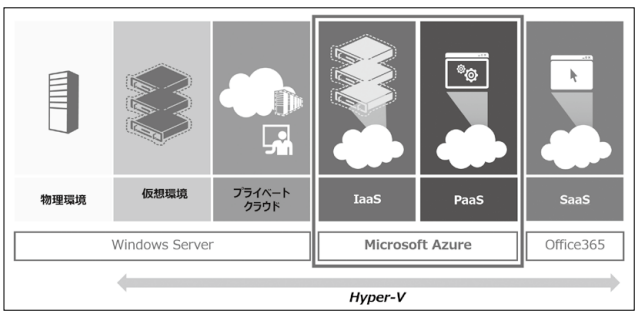
ここからは、Azureにおけるインフラストラクチャサービスのコンピューティングカテゴリで提供される「仮想マシン」の概要についてお話していきます。Azure仮想マシンは、Hyper-Vベースのサーバをホスティングできるサービスです（**図7**）。仮想マシンは、Webサーバやデータベースサーバ、ファイルサーバや開発環境などさまざまな用途で活用することが可能です。

Azure仮想マシンは、Azure Marketplaceやオープンソースの仮想マシンイメージが提供されているVM Depotサイト[注8]のベースイメージを選択して、新規に作成することができ、さらにカスタマイズして独自のベースイメージを作成することもできます。Hyper-Vで作成したVHDをイメージとして利用するためにはSysprep（一般化）されている必要がありますのでご注意ください。Azure仮想マシンで展開された仮想サーバは「インスタンス」と呼んでいます。仮想マシンのVHDファイルはBlobと呼ばれるストレージに保存され、異なる3つのノードに保存されることで3重化されます（前

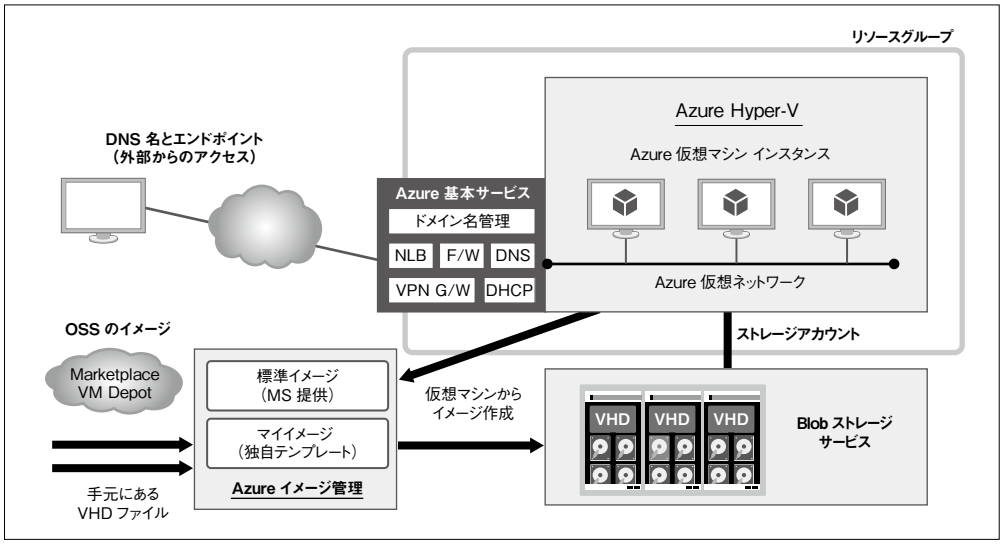
注7） クライアントOSで実装されているクライアントHyper-Vは、Windows 8 Pro 64bit以上で使うことができます。

注8） URL https://vmdepot.msopentech.com/

▼図6　Hyper-VとAzureの関係



▼図7　Azure仮想マシンの利用に必要な基礎知識



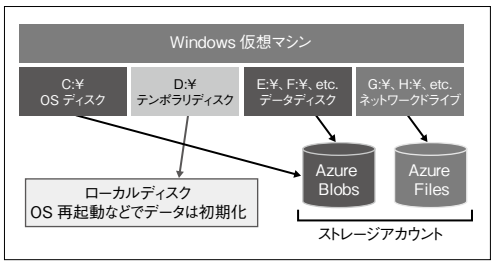
述のとおり）。

Azure仮想マシンのインスタンスは、さまざまな規模のCPUコア数、メモリ、データディスクなどのサイズや数を指定して利用することができます。サイズは、仮想マシンの処理能力、メモリ、記憶容量に影響するため、用途にあったサイズを選択することが大切です。標準サイズは、複数のシリーズ（A、D、G、F）で構成されています[注9]。

・Aシリーズ：一般的な用途で使われる。小〜中規模のサーバに最適
・Dシリーズ：Aシリーズよりも高性能なCPUでローカルSSDディスクをサイズごとに指定できる
・Gシリーズ：大規模なデータベースワークロードに最適
・Fシリーズ：より高速のCPUを必要としつつも、CPUコアあたりのメモリやローカルSSDについてはそれほど多くを要求しないワークロードに最適

注9）選択が可能なAzure仮想マシンのサイズは、配置する場所および価格にも影響します。詳細な情報については「Virtual Machinesの価格」をご確認ください。URL http://azure.microsoft.com/ja-jp/pricing/details/virtual-machines/

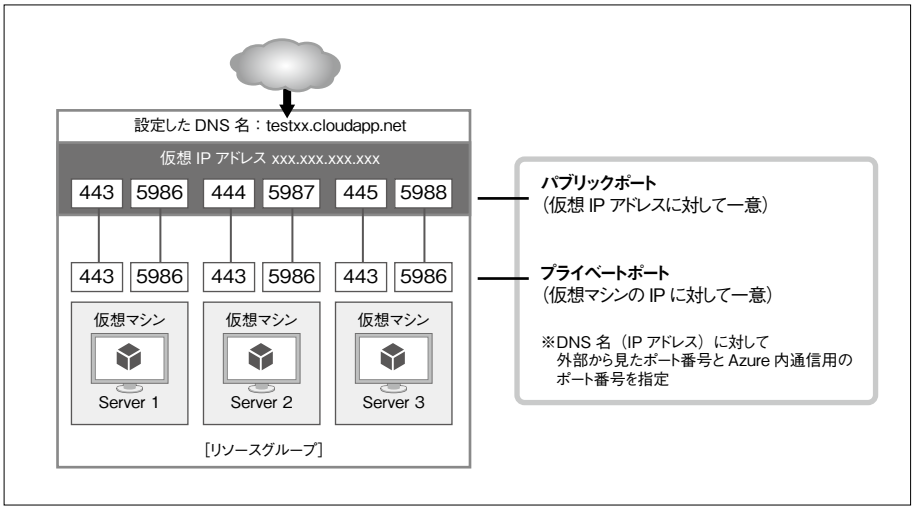
▼図8　Azure仮想マシンのディスク構成



Azure仮想マシンは、OSディスク、テンポラリディスク、データディスクで構成されています（**図8**）。テンポラリディスクは非永続化のため、再起動やサイズを変更するタイミングでテンポラリディスクに格納されているデータはすべて初期化され消えてしまいます。そのため、既存データやアプリケーションはデータディスクに格納して利用します。データディスクが足りないときは、仮想マシンのサイズに応じたディスクの数を追加することができます。

・OSディスク：OS起動用に使用される
・テンポラリディスク：Azure Hyper-Vのローカルストレージを利用したAzureのサービス。非永続化

▼図9　仮想マシンへの外部からのアクセス：エンドポイント



・データディスク：利用者固有のデータやアプリはここに格納する
・ストレージアカウント：Azureストレージアカウントはスタンダードストレージ（Blobストレージ）とプレミアムストレージ（VM Disk）がある

Azure仮想マシンには、管理に必要なDNS名とIPアドレスが付与されます。DNS名は＊.cloudapp.netとなり、＊の部分は仮想マシンの作成時に任意の文字列を指定します。外部からAzure仮想マシンへアクセスする場合には、仮想マシンとの通信を許可するようエンドポイントの設定を行います（**図9**）。ACL（Access Control List）機能により、エンドポイントのポートに対してIPアドレスによるアクセス許可・不許可を細かく設定することもできます。

Azure仮想マシンは、使用した時間に基づいて課金されるしくみになっています。仮想マシンは、Azureポータル上でシャットダウンを実行することで課金を停止することができることを覚えておくと、コスト削減につながります。

Microsoft Azure Stack（Preview）

マイクロソフトでは、自社のデータセンターでAzureサービスを実現できる新たなハイブリッドクラウド基盤として「Microsoft Azure Stack（以降、Azure Stack）」の開発を進めています。現段階では、物理サーバ1台で動くTechnical Previewを公開しており、一部の機能を試すことができます[注10]。

Azure Stackは、Azureで使える機能をオンプレミスで使えるようになるため、ビジネス的あるいは技術的にすべてのシステムをパブリッククラウドに移行することが困難なケースにおいても、開発面や運用面における一貫性を失わずに課題を解決することができますし、世界中のAzureエンジニアが検証した結果やノウハウを社内に持ち込むことが可能になります（**図10**）。

以前のAzureは、Azure Service Management（ASM）と呼ばれる管理モデルが使われていましたが、今では適切なサービス（IaaS/PaaS）を論理的にまとめて展開／管理するための手段としてAzure Resource Manager（ARM）が登場し、Azureの管理基盤が一変しました。そして、Azure Stackの管理モデルにも、その

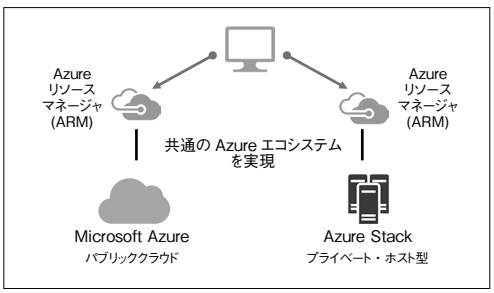
注10) Microsoft Azure Stack Previewサイト
URL https://www.microsoft.com/ja-jp/server-cloud/products-Microsoft-Azure-Stack.aspx

ARMが採用されているため、複数リソースから構成される複雑なシステムを、JSON形式でテンプレートファイルに記述することでハイブリッドなリソース展開が可能となります。よく使われると想定される構成のARMテンプレートは、AzureとAzure Stack両方ともにGitHub上に公開されているので、これを使うと変数の部分のみ手入力で編集するだけで、一からJSONを書き起こす手間を省くこともできます[注11]。たとえば仮想マシンを1台構築するというシンプルなスクリプトから用意されています。また、実行中のリソースグループからテンプレートをエクスポートすることも可能です。

AzureとAzure StackのARMによる一貫性はAPIレベルで共通化されており、Azure用のPowerShellやコマンドもそのまま利用可能ですし、1つのアプリケーションを同じ手法でハイブリッドに展開することも可能になります。ただし、パブリッククラウドで使えるAzureサービスのすべてが利用できるというものではなく、オンプレミスであるAzure Stackで実装することで価値を見出せるサービスから順次提供を開始する予定です。

注11）AzureおよびAzure Stack ARMテンプレートを公開しているGitHubサイト
URL https://github.com/Azure/azure-quickstart-templates
URL https://github.com/Azure/AzureStack-QuickStart-Templates

▼図10　Azure Stackの概要



※引用：【ウェブセミナー】理想のハイブリッドクラウドを実現するMicrosoft Azure & Azure Stack

本章のまとめ

今回はAzure概要について紹介させていただきましたが、いかがでしたでしょうか。Azureには、クラウドを楽しむ要素はもちろん、最新の技術に触れるサービスもたくさん提供されていますので、実際に触れてみるのも面白いかもしれません。最後に今後役立つサイトをご紹介しておきますので、参考にしてみてください。SD

Azureに関するお役立ちサイト

◆Microsoft Azure公式サイト
https://azure.microsoft.com/ja-jp/
Azureに関する情報が集約されている日本語サイトです。各サービスの説明や料金計算ツール、必要なコマンドラインツールやSDKの入手などができます。また、このサイトからAzureを開始するための無料評価アカウントの作成や、Azureポータルへアクセスすることもできます。

◆マイクロソフトマーケティングチーム公式ブログ
https://blogs.technet.microsoft.com/mssvrpmj/
サーバ&クラウド関連の製品やサービスの発表内容を日本語でお届けしています。各製品ごとのブログ記事やテクニカルサポートブログ記事へのリンクが豊富なので、技術的な情報を検索したいときや最新情報の収集をする際にはお勧めです。

◆MSDNオンラインフォーラム（日本語OK）
https://social.msdn.microsoft.com/Forums/ja-JP/home?forum=windowsazureja
マイクロソフトが運営しているユーザ同士で技術的なナレッジやノウハウを共有するためのオンライン知識共有サービスです。いつでも過去の投稿を閲覧でき、質問や回答する場合にはお持ちのマイクロソフトアカウントがあれば無料で利用することができます（投稿時にはオンラインフォーラム用の表示名を指定できます）。MVP（Microsoft Most Valuable Professional）を受賞した専門家やマイクロソフト社員が回答するケースもあるため、専門的な回答が蓄積されています。

◆stackoverflow（英語）
http://stackoverflow.com/questions/tagged/azure
海外の技術系掲示板サイトです。世界中のユーザが活発に質問や回答を投稿していることで知られているため、豊富な情報量が期待できるサイトです。

第4章

ベアメタルクラウドにはどんな利点がある?

SoftLayerとBluemixを擁するIBM Cloudの強み

Author 常田 秀明(ときだ ひであき) 日本SoftLayerユーザ会
Mail tokihide@gmail.com

本章ではIBMのクラウドサービス、IBM Cloudの製品であるSoftLayerとBluemixについて解説します。SoftLayerは、インフラエンジニアがこれまで培ってきた技術を最大限に活かせる「ベアメタルサーバ」が特徴です。Bluemixは、開発したアプリケーションを素早くリリースできるPaaS環境で、アプリケーションエンジニア注目のサービスです。

はじめに

筆者は普段SIerのエンジニアとしてユーザ企業様へ、システムをクラウド化するお手伝いをさせていただいております。ほんの少し前は「クラウド、何それ?」という反応が、ここ1、2年ですっかり一般的な選択肢となり、日々時代の早い流れを感じております。今回この章で紹介するIBM Cloudもこの時代の流れで急激に変化をしているクラウドサービスです。

一昔前はIBMが業界を牽引していました。しかし、現在クラウドサービスを牽引しているアマゾンのAmazon Web Services(AWS)、そしてマイクロソフトのAzureに比べると、一般のエンジニアにとって「SoftLayer」というのは少し遠いところにあり、あまり知られていないのかなと感じています。「IBM」が作るこのクラウドサービスも実は日本でのサービス開始はまだ日が浅く、「IBMフリーク」なユーザの方の中でもまだあまり知らない方も多いのではないかと思います。最近ではIBMというと「ワトソン」が非常にポピュラーなキーワードとなっています。また「Bluemix」という新しいプラットフォームの名前も少し耳に入ってきているかと思います。この章では、IBMのクラウドサービス「SoftLayer」そして「Bluemix」をエンジニアの視点から見ていきたいと思います。

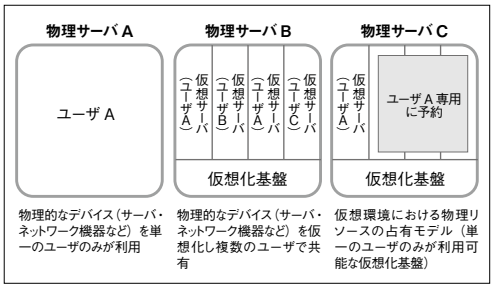
SoftLayerの概要

SoftLayerの提供形態

IBMのクラウドといえば、「ベアメタルクラウド」として有名です。このクラウドサービスは現在「SoftLayer」と呼ばれています。これまでクラウドといえばユーザに対して仮想化されている環境である点や、共有リソースであることを強要してきた背景がありますが、SoftLayerでは「ベアメタルサーバ(仮想化されていない素の物理サーバ)」や「占有型プラン(仮想化環境においても共有利用でないプラン)」などが用意され、ユーザがその提供形態を選択できます(**図1**)。

こういった背景にはいろいろな事情がありますが、SoftLayerでは「仮想サーバを提供していたプロバイダ」が物理サーバを提供したので

▼図1 共有環境と占有環境



はなく、実はもともと「物理サーバを提供している企業」が事業の始まりでした。その後の経緯でベアメタルサーバと仮想サーバを分け隔てなく利用することができるようになり、今に至ります。ベアメタルサーバも仮想サーバもOS(利用者)から見るとほぼ同じように扱うことが可能な設計になっています(**図2**)。

ネットワークに強い世界32ヵ所のデータセンター網

昨今のクラウドベンダーの中ではめずらしくホスティング事業出身という出自のクラウドであり、ネットワークに非常に強いことも特徴の1つです。執筆時の9月上旬現在、世界中の32ヵ所のデータセンターまで拡張されており、それらすべてのネットワーク上のサーバに自由にアクセスすることができます。ネットワークはクラスAのプライベートアドレス帯[注1]が割り当てられたネットワークとして構成されています。

SoftLayer上ではユーザのネットワークとしてVLAN単位でIPアドレスの割り当てが行われます。複数のVLANをまたいで通信を行うために「VLANスパニング」とSoftLayerにて呼ばれる機能を有効にすると、ユーザの管理しているVLAN間でIP通信が可能となります[注2]。このあたりはネットワーク設計としてかなり割り切った設計をしていると思います。昨今では、閉域網として専用線でSoftLayerと接続した際、ユーザが管理する企業側のプライベートネットワークのアドレス体系が同じクラスAで競合してしまうケースなども出てきているため、専用線で接続する際にはNATなどの技術によりアドレスの変換を行う対策が必要です。

とはいえこの構成により、32ヵ所を相互に接続する単一のネットワーク、そしてネットワーク制御装置が仮想化されていないことによるスループットが非常に良いネットワークを作れていることも事実です。速さや安定性という面では素晴らしいと感じます[注3]。

SoftLayerのしくみ

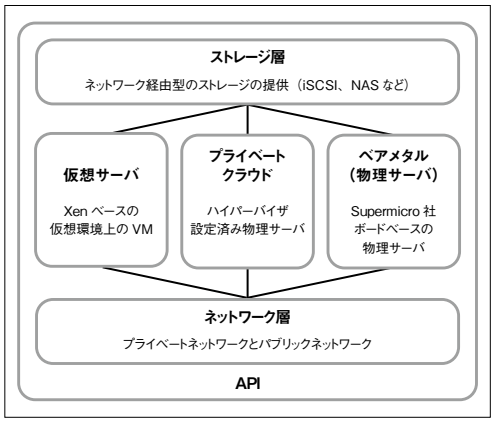
もう少し中をみると**図3**のように、サーバには2つのネットワークへ配線がされています。

注1) プライベートIPアドレスの範囲はRFC 1918で規定されており、クラスAからCに分けられています。クラスAは10.0.0.0〜10.255.255.255(10.0.0.0/8)、クラスBは172.16.0.0〜172.31.255.255(172.16.0.0/12)、クラスCは192.168.0.0〜192.168.255.255(192.168.0.0/16)。

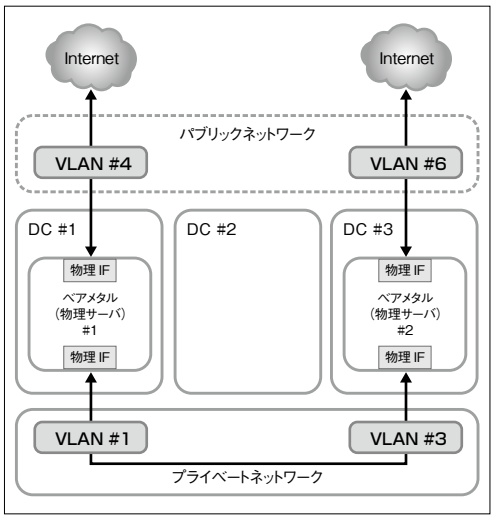
注2) これらの通信は単一のデータセンター内だけでなく、東京←→ダラスなど異なるデータセンター間でも有効です。したがって一度設定してしまえば、SoftLayerの全データセンター内の自分の管理しているサーバに非常に簡単にアクセスが可能です。

注3) 32ヵ所ものデータセンターを結べば地球を一周するネットワークになり、小型のインターネット網のようになっています。

▼図2　SoftLayerの論理構成図



▼図3　ベアメタルサーバのネットワーク構成



1つは「パブリックネットワーク」で、もう1つが「プライベートネットワーク」です。このうちのパブリックネットワークは実際にグローバルIPアドレスがOS上の設定として付与されており、昨今の仮想化されたネットワーク環境を持つクラウドと比べると珍しい構成になっていると言えます[注4]。そしてもう1つのプライベートネットワークの接続は、VLANを経由してすべてのデータセンターとの通信を可能にします。海外に複数拠点展開してるユーザであれば、このシンプルさは目を引くのではないでしょうか。

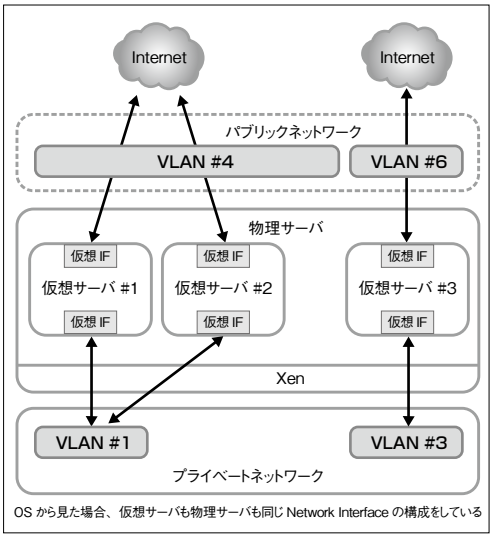
またSoftLayerでは、これらのネットワーク周辺の構成などが公開されている部分もあり、透明性の高さをアピールしています。実際にシステムを構築した際に仮想化されたネットワークでトラブルシューティングが難しいという話も聞かれますが、SoftLayerではかなりのところまで問題原因を追うことができるインフラサービスとなっています。

ちなみに仮想サーバにおいても同様のネットワーク構成がされており、ハイパーバイザ経由で仮想ネットワークインターフェースが提供され、それぞれパブリックとプライベートに接続される構成になっています（**図4**）。このように物理サーバと仮想サーバでの差がハードウェア構成上も少なくなるように設計されています。

このようなことからベアメタルサーバと仮想サーバを相互にバックアップ→復元することができるFlex Imageと呼ばれるサービスも提供しています[注5]（**図5**）。

注4）想像がつくように、グローバルIPアドレスが付与されているということは直接インターネットに接続されているということでもあるので、キチンとネットワークなどの設計をし、Firewallなどでセキュリティを担保していく必要があります。

▼図4　仮想サーバのネットワーク構成



ベアメタルクラウドとは?

「ベアメタル」は単純に言えば「素のハードウェアのサーバ」です。その特徴は大きいところで次の3つ、

❶自由にリソース構成を選択できる
❷自由に構成をコントロールすることができる
❸ハイパフォーマンス（費用に応じたパフォーマンス）

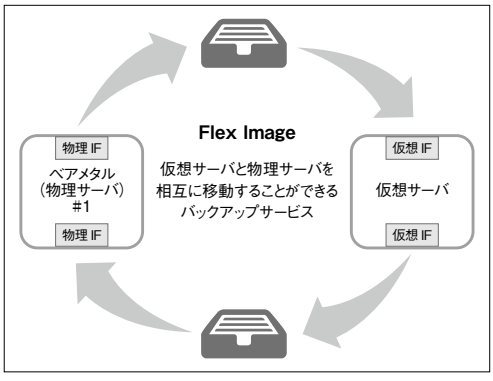
があげられます。順に詳しく解説していきましょう。

❶自由に選択できる

SoftLayerにおけるベアメタルサーバのオー

注5）こちらについては最新のOSに対応していないなどがあり、少し残念なところです。

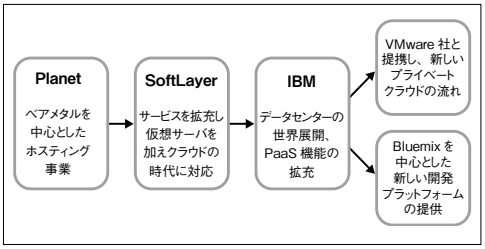
▼図5　Flex Imageの構成



COLUMN ホスティング業者からIBM Cloudに至る道程

まだ世の中がクラウドという言葉で利用していないころ、SoftLayer社がThe Planet Internet Services社と合併して、現在のSoftLayerのように物理サーバを提供するホスティング企業として事業を展開していたのがはじまりだそうです（なんとまだPlanet時代のサーバは稼働中！）。その後SoftLayer社として、AWSやCloudStackといった仮想サーバを主体としたサービスに対抗するために、シトリックス社の技術を利用してSoftLayerも仮想サーバをラインナップに加えてサービスを提供していました。IBMに買収された2013年当時はまだ日本にデータセンターはありませんでしたが、AWSの日本データセンター設立後の躍進に導かれるように、IBMも2014年にデータセンターを日本に設立しています。日本データセンターと今年に入ってからのVMware社との提携などにより、ベアメタルを利用したプライベートクラウドの流れはエンタープライズ企業にとって見過ごせなくなると感じています（**図A**）。

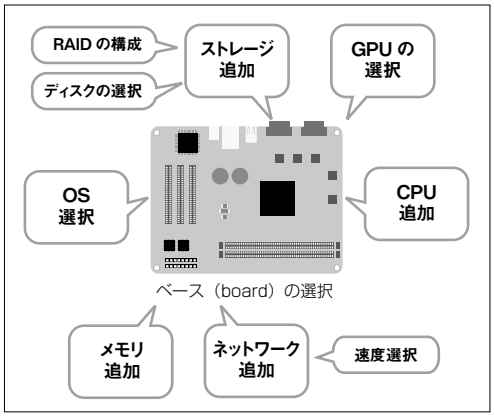
▼図A　SoftLayerの歴史



ダー方法は完全に自作サーバです（笑）。SoftLayerのサイトを見ると、たくさんのマザーボードから利用用途に応じて選択をするところから始まります。フルオーダータイプ（月額）と、構成が決まったセミオーダータイプ（時間額）があります。フルオーダータイプの場合にはCPUの種類や数、メモリの数などサーバを構成する物品を選択していき、オリジナルのサーバをオーダーすることが可能です（**図6**）。最新のチップセットやGPUなど話題のハードウェアが利用できます。

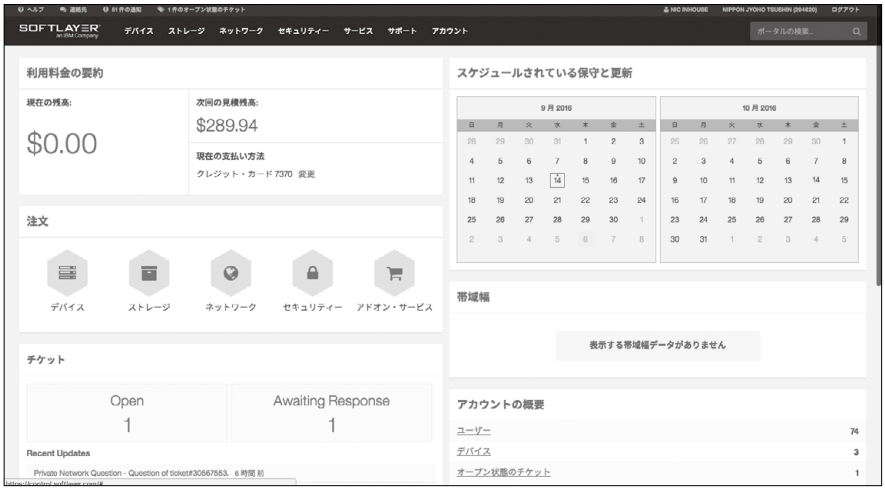
このようにオーダーされたサーバはおおむね4時間程度で利用可能になります（とはいえ大容量のストレージをオーダーすれば初期化するのにかかる時間は必然ですのでその時間はかかります）。しかも初期構成済みの場合には30分で物理サーバが利用可能です。以前からSI案件でインフラを構築していたエンジニアの立場からすると、驚異的な時間の短さと言えます。物理的に何かしらはキッティングする作業は発生しているはずなのですごいですね。自由に選択できるという点ではOSのバリエーションも非常に豊富です（CentOS、Citrix XenServer、CloudLinux、CoreOS、FreeBSD、Windows、

▼図6　ベアメタルサーバのコンポーネント

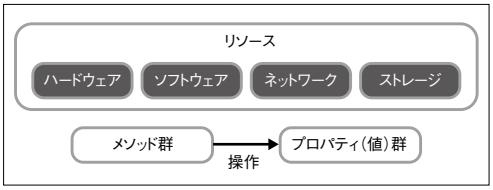


RHEL、Ubuntu、VMware、Vyatta、none OS）。なかには、OSを入れないという玄人な選択肢もあります。使い方としてはかなり自由度が高くなっています。昨今、エンタープライズ企業において企業内の仮想化基盤としてVMwareを利用されているユーザが少なくありません。そういったすでに仮想化基盤を持っているユーザ企業が既存の資産（技術的なものや構築されている環境）を維持しながらもクラウドの恩恵を得るためにベアメタルサーバ上にVMware環境を構築されるケースが多くありま

▼図7　管理コンソール



▼図8　API概念図



す。とくにIBMは今年、VMwareと提携を行ったこともあり、この流れはしばらく続くと考えられます。VMwareでクラウドの基盤を利用してもらい、新規のシステムでは仮想サーバや物理サーバを使っていくというケースが想定されます。

❷自由に構成をコントロールすることができる

SoftLayerでは基本的に、さまざまなことができる権限を極力多くユーザに提供しています。ベアメタルサーバではなんとBIOSの設定までも変更することができます。したがってブートシーケンスの設定だったり、特殊なオプションを有効にするなどといったことが可能です。SoftLayerで提供しているベアメタルサーバのマザーボードはSupermicro提供のもので、多くのツールやしくみを利用できます。

たとえばユーザはIPMI[注6]経由でマザーボードにアクセスし、設定を確認することができます[注7]。

また観点は違いますが、OS上の管理ユーザとしてrootアカウントが提供されます[注8]。こういった点もSoftLayerの自由さとして現れているかと思います。

SoftLayerを支えるAPIの技術

この章に至るまでに「AWS」「Azure」での開発を読まれてきたことと思います。SoftLayerでも同様にサーバ、ネットワーク、ストレージ、そしてそれ以外のサービス群をリモートより簡単に操作して設定していくことができます。これらの操作は「コントロールパネル」と呼ばれるブラウザベースのWebアプリケーションとして提供されています（**図7**）。

この裏側では多数のAPIが呼び出されてい

注6） Intelligent Platform Management Interface：外部から電源のON/OFFやリモートコンソール、温度やエラー情報などを取得できるインターフェース。

注7） 実際に設定の変更をするためにはADMIN権限が必要であり、サポートチケットに依頼するとユーザの権限をADMIN相当にしてもらえます。こうすると設定の変更などができ、たとえば仮想メディアをマウントしたり、BIOSの設定を変更したりできます。

注8） 他社のクラウドの多くは専用アカウントが払いだされ、sudoする環境になっていることが多いかと思います。

ます。APIは、SoftLayerのほぼすべての部分をコントロールできるように構成されており、たとえばサーバの電源を入れたり、またはストレージを追加したり、ネットワークを構成したりすることを自在に行えます。実際にはAPI基盤がサーバや各ハードウェアのファームウェアに対して指示を出したりすることになるわけですが、とくにこのAPI設計が特徴的であり、あらゆるリソースを同じように取り扱うことができます。たとえば、先ほどのネットワークの電源の入れ方もストレージの追加の仕方も統一された仕様で行われており、一度理解すればよいというのは非常に低コストです。

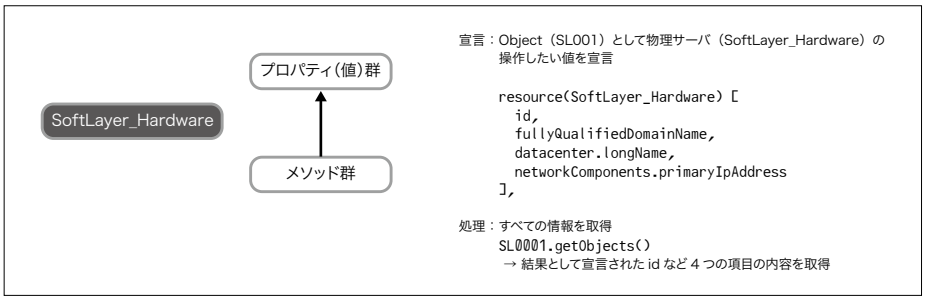
さまざまなリソース（定義上は100種類以上）それぞれに、プロパティと呼ばれる値が存在し、それを操作するためのメソッドが用意されています（**図8**）。

たとえば、ユーザが利用しているすべての物理サーバのID、FQDN（Fully Qualified Domain Name）、データセンター名、IPアドレスの情報を取得する際には、操作したい情報の宣言を行い、その宣言されたオブジェクトに対して情報を取得するというgetObjectsメソッドを実行することで、宣言された値を得ることができます（**図9**）。この操作はSoftLayer全体で統一されており、目的のリソースを調べさえすれば、さまざまなことが同じ操作で行えます。

APIでのアクセスはベアメタルサーバに対しても同様にコントロールすることが可能です。実際に弊社では、これらのAPIを利用して複数のアカウントのユーザ情報・請求情報・構成情報を管理するためのプロダクトを開発し運用しています（**図10**）。

その気になれば、システム専用のコントロールパネルやダッシュボードを作ることができるのは運用面のアドバンテージになります。APIでさまざまなことができるというのは、従来サーバの前でマニュアルを見ながら設定パラメータ

▼図9　例）ハードウェア情報取得の方法



▼図10　日本情報通信㈱の管理ソフト「kumade」コンソール画面

を1つずつ入力していたことから比べて非常に面白く、APIを利用してスクリプトでソフトウェアを導入するようにハードウェアを構成することができます。まさに最近はやりのInfrastracture as a Codeを実施できる環境と言えます。

SoftLayerのAPIは基本的には、コンソールから発行されるユーザID/Tokenキーでアクセスする REST API（またSOAP、XML-RPC）として提供されています。より開発を促進するためにPythonベースのslcliと呼ばれるコンソール同様のコマンド、また開発用のLibraryとしてGo言語（つい先日追加されていました！）、Python（Cliなどはすべて Python製なので一番安定しています）、PHP、Javaなどが提供されています。

❸ハイパフォーマンス（費用に応じたパフォーマンス）

共有環境の悩みの1つに、noisy-neighborと呼ばれる共有環境において、自分以外のユーザがリソースを多く使用してしまい自身が求めるパフォーマンスが出ない問題があります。一般的なクラウドでは、こういった問題に対して帯域保証などをオプションで行うことでSLA（Service Level Agreement）を担保しています。より多くのコストをかけることでパフォーマンスが保証されるトレードオフを行うことになります。ネットワークやストレージなどのI/Oは多くがこのケースで提供されていますが、サーバに関しては共有環境のコストを軽減するために実際のリソース以上のリソースを利用者に割り当てる、オーバサブスクリプションと呼ばれている割当を行うことがあります。この場合にはnoisy-neighborを避けることは難しいです。

SoftLayerにおいてもこの問題は皆無ではありませんが、物理サーバにおいては発生しません。そして費用面でも仮想サーバと同レベルの価格帯から提供されており、物理サーバ＝高価格という図式ではないため、安定したパフォーマンスを得たいという場合においてもベアメタルサーバは有効です。実際にジョブなどを実行して日により実行時間が異なるという状況になれば、問題になるケースも出てくるかと思います。また仮想サーバにおいても、SoftLayerは2GHzという単位で提示されており、オーバサブスクリプションをしていないのではと考えられます（公開されていません）。

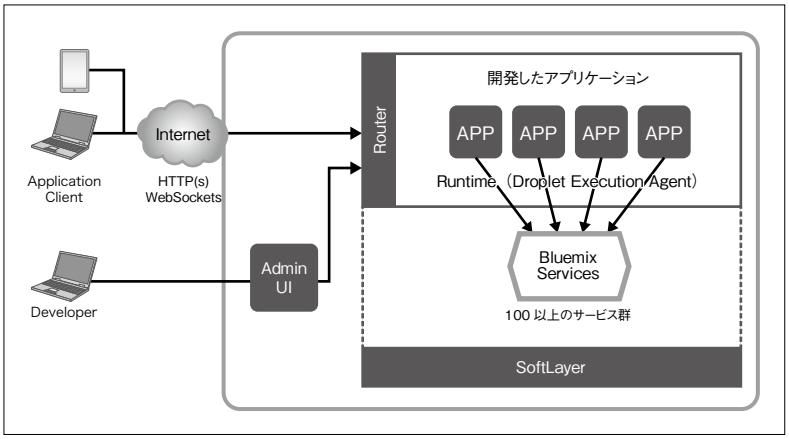
いくつかの観点はありますが、物理サーバとしてオーダーできるスペックは日々進化しており、最新CPUと大容量メモリ（たとえば48ソケット搭載可能なベースであったり、3TBまで搭載できるメモリなど）、サーバ間を接続する高速なネットワーク（10Gbps、InfiniBand）、高速なディスク（SSDやFusion-io）、GPUなどのボードを追加で選択することができます。また、CPUにIntelだけでなくPowerを選択することも可能です。

IaaSとPaaSの両展開の実践

IBMがSoftLayer買収後に打ち出した一番ホットなサービスがBluemixです。これまでのIBMの雰囲気とはかなり違うライトな印象のサイト、開発者を見て作られたであろうダーク調なコントロールパネル、多数のOSSで構成された内容と、IBMにとってBluemixは挑戦的なサービスであると思います。

Bluemixが何かという問いに対しては、構成的には「Cloud FoundryベースのPaaSです」というのが答えになります。Cloud FoundryはHerokuにインスパイアされたOSSベースのソフトウェアです。Cloud Foundryに、非常に多くのサービスと小綺麗なポータルを用意しているものになります。これまでのSoftLayer側の視点はインフラエンジニアにとり面白い視点であり、アプリケーションエンジニアにとっては馴染みがなく、また従来どおり覚えることも多く、始めるのはハードルが高かったこともあるかと思います。それに対してBluemixは、まさにアプリケーションエンジニアにとっての面白い環境になりつつあります（**図11**）。

▼図11　Bluemixのサービス体系の相関図



Bluemixは現在3つのSoftLayerのデータセンターで運用を開始しています。残念なことにはSoftLayerのようにロケーションフリーではないため、各データセンター単位での管理となっています。このBluemixではたくさんの意欲的なことが行われています。IBMはBluemixで多くの初物を扱っています。これまでは自社のRDBMSのみを提供していたスタイルから、Compose社との提携によりMySQL、Elastic Search、Redisなども利用できます。また、SaaS提供をしていたIBMのソフトウェアプロダクトも利用することができます。代表的なIBMのNoSQL DBであるCloudantなども、Bluemix経由で簡単に利用できるようになっています。

現在、SoftLayerの安定性や性能の面からも、Bluemix上のOSSサービスを利用して開発を行い、SoftLayer上で本番環境を構築する、などの展開が行われています。とはいえこのBluemixの波は徐々に広がっており、従来のCloud Foundryが提供するPaaSの実行環境だけでなく、OpenStack VMやDockerそしてOpenWhiskといった、より粒度の細かい単位での制御サービスに発展しています。そのうちにBluemixからSoftLayerのベアメタルが利用できる日も来るかもしれません。SoftLayerからBluemixを利用するためのアイデアはいくつかあります。SoftLayerにはBluemixの「サービス」と呼ばれる部分に該当する機能が提供されていないので、SoftLayer側からBluemixの「サービス」を利用するのは1つのアイデアです。逆に、データなど自身で管理したいデータ部分をSoftLayer上で構築したストレージに、Bluemixからアクセスすることも可能です。

最後に

過去、サーバが目の前にあったころには当たり前だったことができなくなっている現在では、サーバについて学び始めたエンジニアの人にとり、SoftLayerは非常に貴重な場であるように感じます。また、クラウドであってもオンプレミスと同様の技術を利用できるということは、いつでも自社でシステムを別環境で構成できるという安心感にもつながりますね。

また一方で、Bluemixは開発を行い、すぐにでも実行させたいというニーズに応えてくれます。作成したプログラムが一定以上の品質で簡単に公開できるのは非常に魅力的です。この2つはまだ融合されていない点も多いですが、エンジニアのアイデア次第で連携して利用することが可能です。本章の解説で興味を持たれたら、まずは無料トライアルからはじめてもらえればと思います。SD

第5章 インフラの構築・運用はPaaSで省略

スモールスタート&高速開発に最適なHeroku

Author 織田 敬子（おだ けいこ） Heroku

インフラの構築・運用はできるだけ外部サービスに委託してアプリケーションの開発に集中するのが、イマドキの開発手法。それを実現するPaaS、その中でもとく若手エンジニアに人気のあるHerokuを紹介します。併せて、Herokuの機能を最大限使って開発を効率化するHeroku Flowも解説します。

はじめに

AWSは使用したことがあるけれどHerokuやほかのPaaSは使用したことがない、またはPaaSを趣味のアプリ作成では使用したことがあるけれど、業務では使用したことがない、といった方は多いのではないでしょうか。この章では、PaaSとは何かといったところから、PaaSとしてのHerokuのメリット、その裏側とアーキテクチャまで解説したいと思います。

PaaSとは

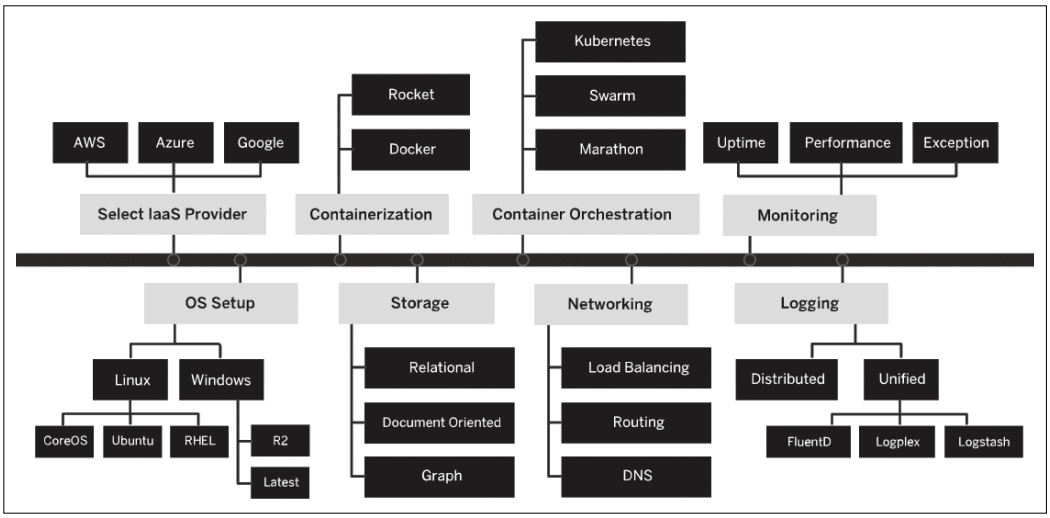
PaaS（Platform as a Service；パースまたはパーズと読む）は、アプリを実行、または運用するためのプラットフォームをサービスとして提供します。このプラットフォームでは、OSだけでなくアプリを実行、または運用するために必要なミドルウェアも提供しますので、開発者はインフラの知識を必要とせずにアプリを開発、公開できます。

PaaSはInfrastructure as a ServiceとSoftware as a Serviceの中間に位置しますが、筆者は「IaaSの上に1枚レイヤを重ねた感じ」といった説明をよくします。IaaSはサーバの調達や管理といったものをクラウド上で行うことができて非常に便利ですが、このサーバのセットアップ部分、運用部分はユーザしだいであり、自由度が高い反面、その使用にはインフラの知識が必要でした。実際にアプリをローカルで開発できる、という方でもいざIaaSを使用してデプロイする、公開する、となるとさまざまな準備や知識が必要となります。PaaSはこの部分をプロバイダが提供することにより、開発者はサーバの管理運用をプロ（PaaSプロバイダ）に任せ、開発に注力できます。SaaSの利用者はおもに一般ユーザであるのに比べると、PaaSやIaaSは開発者に向けたサービスで、その上に何かを作ることを前提としていることからも、PaaSはIaaSに近いものと言えます。

Herokuは代表的なPaaSのうちの1つですが、そのほかにもMicrosoft社が提供するMicrosoft Azure、Red Hat社のOpenShift、Google社のGoogle App Engineなどが代表的なPaaSとして挙げられます。

図1はHerokuのホームページから引用したものですが、IaaSを使用して一般的なWebアプリを作成しようとした場合に必要なステップや知識を示しています。まずIaaSのプロバイダを選定し、OSのセットアップをします。その後コンテナに何を使用するか、データストア（データベース）に何を使用するかの選定およびセットアップ、コンテナの管理運用が必要になってきます。また、ロードバランスやルーティングといったネットワークの設定もWebアプ

▼図1　build apps, not infrastracture（https://www.heroku.com/）



リには必要不可欠です。それらが済んでいざアプリをデプロイしても、その後アプリが動いているかだけでなく、サーバ・コンテナがしっかり動いているかのモニタリングの設定、またログはどうするかなど、アプリ開発という本来の目的とはかけ離れたインフラの設計と開発が必要になってきます。PaaSはこれら一連のものをプラットフォームとして提供しているのです。

HerokuのPaaSプロバイダとしての歩み

Herokuは Salesforce.com が提供するPaaSです。Herokuは2007年の夏に設立されました。当初は、Ruby on Railsアプリをブラウザベースで作成・ホスティングできるといったもので、PaaSの概念からは少し離れたサービスを提供していました。

2009年にこのブラウザベースの開発環境を破棄し、開発はローカルで開発者の慣れ親しんだエディタで行い、Herokuへは`git push heroku master`を用いてデプロイする、といった形にシフトしました。これにより、HerokuはPaaSプロバイダとしてプラットフォームを提供していくという道を進んでいくことになりました。このとき、言語としてはまだRubyのみのサポートでしたが、次々と生まれる新しい言語、またそれらを積極的に利用していく開発者コミュニティの後押しもあり、Ruby以外の言語をサポートする試みが始まってきました。また、同時期にAdd-ons（アドオン）と呼ばれる、サードパーティの提供するさまざまなサービスを簡単にアプリに追加できるサービスの提供を始めました。

この1年後の2010年には、アプリ開発において多く使われるデータベースのアドオン「Heroku Postgres」の提供を始め、より簡単にHerokuのエコシステム内でのアプリの作成が可能となりました。

2010年末にSalesforce.comがHerokuの買収を発表し、PaaSプロバイダとしてさらにサポートする領域を増やしていきました。言語としては、2012年にCedarスタック（実行基盤）がGA[注1]となり、Rubyに加えてClojure、Java、Node.js、Python、Scalaが正式にサポートされるようになりました。

2014年にはPHP、2015年にはGoも正式サポートされるようになりました。このようにサポートされる言語が増え、アプリの性質に合わせた言語を開発者がより自由に選択できるよう

注1）General Availability：正式リリース。

になりました。Heroku社内でも多くのHerokuアプリがデプロイされており、さまざまな言語が使用されています。

DX[注2]、開発者の生産効率の向上やより良いアプリ作りを支援する、という方面からもHerokuは進化していきました。2011年にはモダンなWebアプリを開発・運用するうえで考慮すべきアーキテクチャについて解説されたThe Twelve-Factor App[注3]が公開されました。Heroku上で作成されるアプリについては自然とこれに沿うことができるようになっており、これによってHerokuにデプロイされたアプリは簡単なスケールイン・アウトが可能などPaaSの力を十分に引き出すことができるようになっています。

2014年のHeroku Buttonのリリースにより、開発者は簡単にテンプレート化されたHerokuアプリを自分のアカウントでデプロイできるようになりました。

2015年にはGitHub Integrationが紹介され、これまでGitHubとHerokuの両方にpushしなければならなかった運用から、GitHubにpushするだけでHeroku上にも自動的にデプロイされるようになりました。その後、Heroku Flow[注4]と呼ばれるContinuous Deliverly[注5]をサポートするしくみが発表されました。

これらのプロダクトは、PaaSプロバイダであるHerokuの、長年多くのアプリをホスティングしてきた実績と経験を通じて培ってきたアプリ開発におけるベストプラクティスから来ており、どうすればより良い開発体験が得られるかを追求していったものとなります。これについてはあとで詳しく説明しますが、Herokuではこのようなプロダクトを通じて、ベストプラクティスを開発者が自然に簡単に取り入れることができるような工夫がなされています。

PaaSが認知され始め利用者が増えるに連れ、徐々に、とくにエンタープライズな利用者から、よりセキュアで独立した環境であるプライベートクラウドの需要が増えてきました。Herokuは従来パブリッククラウドのみを提供していましたので、このような需要に応えるためにも2015年、Private Spacesを発表しました。これにより、Heroku本来の開発体験や開発効率は維持したまま、プライベートなPaaS空間が簡単に構築できるようになりました。この従来のパブリッククラウド（Common Runtime）とプライベートクラウド（Private Spaces）についてはのちほどまた詳しく説明します。

このように、HerokuがPaaSプロバイダとして提供しているサービスは、特定の言語に特化したプラットフォームから、より良い開発体験を含めたさまざまな言語・ニーズに応えることのできるプラットフォームへと進化をしていきました。

Herokuの特徴

Herokuプラットフォーム

Herokuでは、アプリは「Dyno」と呼ばれるコンテナ単位で動きます。このDynoは、アプリコードはもちろんのこと、アプリが依存するライブラリ群などアプリを実行するためのすべてのリソースを含みます。Herokuでは、このDynoの作成やスケールイン・アウトが簡単にできるしくみの提供、WebリクエストのDynoへの割り当て、ログの処理などさまざまな機能を実行環境として提供しています。また、この実行環境はHerokuの運用およびセキュリティチームにより24時間365日モニタリングされています。

多言語対応

Herokuでは現在、Node.js、Ruby、Java、PHP、Python、Go、Scala、Clojureの8つのプログラ

注2）デベロッパエクスペリエンス、開発体験。
注3）URL https://12factor.net
注4）URL https://www.heroku.com/continuous-delivery
注5）継続的デリバリ。継続的にアプリの開発・テスト・リリースを行うしくみ。

ミング言語・環境をオフィシャルにサポートしています。Herokuにアプリをデプロイすると、アプリの言語を判別し、それぞれの言語に応じて自動的にプログラミング言語やアプリを走らせるために必要な依存ライブラリなどを格納した「Slug」と呼ばれるオブジェクトが生成されます。このSlugをDynoコンテナの上に載せることにより、アプリを実行できる状態にします。

このプロセスはbuildpack[注6]というしくみを用いて行われますが、オフィシャルに提供されているもののほかにもカスタムのbuildpackを利用することにより異なる言語、もしくは標準のbuildpackでは入らない依存ライブラリなどを利用できます。

これはHeroku/PaaSの長所でも短所でもあります。各言語ごとにしっかりとしたオフィシャルのbuildpackがあるからこそ、コードをデプロイするだけでインフラ部分を意識することなく必要なものがすべて用意されるという長所と、少し変わったプラットフォーム（例：サポートされていない言語やライブラリ）がほしい場合は、オフィシャルにサポートしていないカスタムのbuildpackを自分で作成する必要があるという短所です。ただ、このカスタムbuildpackもエコシステムは充実しており、1,000個以上のものが公開されていますので、誰かがすでに作ったものを再利用できる場合が多々あります。

Herokuエコシステム

Herokuの大きな特徴として、エコシステムが非常に発達していることが挙げられます。Herokuではアドオンを使用することにより、データベースなどを簡単にアプリに追加できます。たとえば、Heroku Postgresアドオンを追加することによってアプリから簡単にPostgreSQLを使用できます。データストア関連のアドオンだけでもPostgreSQL、MySQL、Redis、MariaDB、MongoDB、Neo4j、Kafkaといったようなものが利用できます。

現在アドオンの数は150を超えており[注7]、それらを組み合わせることにより非常に効率的な開発ができます。たとえば、メール関係のアドオンを使用することにより、メールサーバを立てることなくメールの送信ができます。アドオンの追加はたいていワンクリックでできて、設定も非常に容易なものが多いです。

Heroku button[注8]、buildpackもエコシステムのうちの1つで、さまざまな作者によって作られた多くのものが公開されています。これらを利用することによって、開発を加速させることができます。

PaaSとしてのHerokuのメリットと使用例

スモールスタート

スモールスタートはクラウドの利点でもありますが、これはPaaSにも当てはまります。Herokuには無料もしくは少額から使用できるDynoやアドオンが豊富にあり、これらを使用してプロトタイプを作成・公開して必要に応じてスケールイン・アウトしていく、といったことができます。Dynoやアドオンは秒単位の課金ですので、数時間だけ、3日間だけ、といったように短期間だけ使用するような使い方もできます。

スモールスタートで簡単にトライ・アンド・エラーができるということで、PaaSは新規事業などに非常によく利用されています。アジャイル開発ともとても相性が良く、たとえば大きな企業の新規事業開発チームなどで利用されます。Heroku社内でも、新しいアプリを思いついたらすぐにHerokuにデプロイしてみて社内で使用し、ニーズが高ければスケールアウト、もしあまり使われないようなら破棄、といった

注6）URL https://elements.heroku.com/buildpacks

注7）URL https://elements.heroku.com/addons

注8）クリックするだけで、テンプレートを元にアプリがHerokuにデプロイされるボタン。URL https://devcenter.heroku.com/articles/heroku-button

ようなことがよく行われています。

アプリ開発に集中できる

PaaSでは特別なインフラの知識が必要ない、アプリを走らせるために必要な環境を準備する必要がない、というのは非常に大きな利点です。これは、インフラの知識がない人にはもちろん、インフラの知識がある人にとっても構築の手間が省けるということで非常に重宝されています。また、PaaSではIaaS（インフラ）部分は隠蔽され、その部分の管理・運用はPaaSプロバイダが行っています。そのため、アプリを走らせているサーバのモニタリングなどを気にする必要はなく、利用者はアプリ開発に集中できます。

たとえばサーバを管理しなければならないとなると、OSやライブラリにセキュリティ脆弱性が発見されたとき、まずセキュリティに長けた人がその重大性を検証し、パッチを当てる必要があるか判断する必要があります。パッチを当てる必要があると判断された場合、実際にパッチを当てるにあたってどのように実行するかを計画し、管理しているサーバすべてにそれを行う必要があります。このステップだけでもスキルを持ったセキュリティエンジニアとインフラエンジニアが必要になります。そのようなリソースを保持していない企業も多く、重大な脆弱性にパッチが当たらないまま運用を続けてしまっている、といったようなケースもあります。

PaaSの場合はプロバイダ側でプロのセキュリティエンジニアとインフラエンジニアがおり、このようなセキュリティ脆弱性が発見されたときは適時にパッチを当てます。このようなインフラ部分というのは本当に価値を出したいところ、アプリの実装や機能からはかけ離れたところにあり、本来意識する必要がない部分です。意識する必要はないのですが、インフラが安全に安定して動いているというのは非常に重要なことであり、PaaSはそれを提供しているのです。

この特徴を活かして、スタートアップ企業などの小さなチームでPaaSはよく利用されています。PaaSを使用することによってインフラに特化したエンジニアを雇う必要がないですし、その部分の教育をチームメンバーにする必要もありません。その部分のリソースを、アプリをより良くしていく開発に使用できるのです。

開発のベストプラクティスと効率の向上

アプリ開発への集中というのも開発効率を高める要素の1つですが、ここではHeroku Flowを例にとって、Herokuがどのようにしてアプリ開発のベストプラクティスを開発者が自然に簡単に取り入れられるようにし、開発効率の向上を助けているかを説明します。このようにアプリ開発のフローまで提案、提供ができるのは、PaaSならではと言えるでしょう。

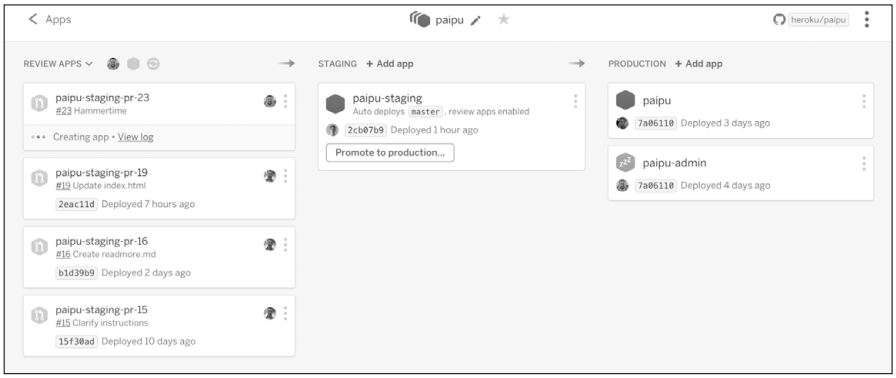
Heroku FlowではHeroku Pipelines、Review Apps、GitHub Integrationの3つを使用してContinuous Deliveryのための構造化されたワークフローを提供しています。これによって、テストやデプロイが非常に簡単になり、イテレーションの加速につながります。実際の開発フローとしては、次のようになります。

- Heroku Pipelinesを使用してpipelineを作成する
- GitHub Integrationを使用して自動デプロイを設定する
- Review Appsを有効にする
- Pull Request（PR）を新規作成し、レビューアプリを使用して確認
- PRをマージしてステージング環境にデプロイする
- 本番環境にプロモートする

このワークフローは多くの企業で採用されています。とくに、Herokuを長く使用してきた企業などは、自前で似たようなことを長い時間をかけてしていましたが、Heroku Flowを用いることによりワークフローが整理され、とても簡単にできるようになりました。

Heroku Flowを実現する機能を1つずつ見ていきましょう。実際にHeroku社内でも、これ

▼図2　Heroku Pipelinesの画面



ら機能を多用してアプリを作成しています。

Heroku Pipelines

Heroku Pipelinesは、同じコードベースを持ったHerokuアプリ（pipline：レビュー・開発・ステージング・本番環境）を管理・可視化するためのツールです（**図2**）。本番環境のみを用意するのではなく、常にステージング環境や開発環境を用意するというのは、アプリ開発において非常に重要です。開発・レビュー環境ではさまざまな新しい取り組みをし、形がまとまった時点でステージング環境にプロモートしさらにテストを行います。ステージング環境でのテストがしっかり通ったもののみ、本番環境にプロモートします。このようなワークフローを使うことにより、重大なデプロイミスを防ぐことができますし、本番環境で見られる問題をステージング環境でデバッグするなどといったことも容易にできます。

Review Apps

Review Appsでは、GitHubのPRベースで、新しい一時的なアプリを作成できます。これは新機能の開発などに非常に便利です。実際に動くアプリがPRをもとに作成されるので、レビュアはそのアプリを自身で操作して実際に目で見てレビューできます。フィードバックをそのPRに反映すると、レビューアプリにも新しいコードが反映されます。そうやって実際に動くものを見ながらレビューを行い、実際に固まったところでこのPRをマージします。マージが完了した時点でHeroku Pipelinesがこれをステージング環境にプロモートし、レビューアプリは自動的に破棄されます。このようにReview Appsは、プロダクト側を巻き込んだイテレーションが容易なレビュー環境を可能にします。

GitHub Integration

GitHub Integrationでは、GitHubのリポジトリをHerokuアプリと同期でき、手動または自動で特定のブランチにpushされたコードをHerokuアプリにデプロイできます。以前はこの機能がなく、GitHubとHerokuの両方にpushをする必要があったので、Herokuのコードと GitHubのコードが一致していないといった現象がよく起きていました。

もしGitHub上でContinuous Integration（CI）[注9]の設定をしている場合は、CIが通るコミットのみHerokuに自動デプロイする、といったような設定もできます。Review Appsもこの機能を使用していますし、また開発環境アプリにも使用できるでしょう。

小さなアプリから大きなアプリまで

Herokuではさまざまなサイズのdynoとアド

注9）継続的インテグレーション。アプリ開発において、継続的にビルドやテストを行うためのしくみ。

オン、またパブリッククラウドとプライベートクラウドの両方を提供しているので、スモールスタートや趣味のアプリといったような非常に小さなアプリから、高セキュリティが必要なエンタープライズアプリ、非常にトラフィックの多い大規模アプリまでサポートしています。これらをすべて、同じHerokuの開発体験を用いて開発していくことができるのです。

Herokuというと日本ではまだまだ趣味のアプリ用のプラットフォームとしての利用が多いですが、近年ではエンタープライズでの利用も非常に増えています。また、プライベートクラウド（Private Spaces）にはTokyoリージョンがあり（パブリッククラウドにはUS/EUリージョンのみ）、日本のアプリも多数デプロイされています。家計簿アプリの「Moneytree」もPrivate Spacesを使用しTokyoリージョンにアプリをデプロイしています[注10]。

Herokuの舞台裏

Herokuのアーキテクチャ

Herokuは実際どのようにしてPaaSを提供しているのでしょうか。最初に「IaaSの上に1枚レイヤを重ねた感じ」といった説明をしたとおり、HerokuではIaaS（Amazon EC2）を使用しています。では、実際にどういったふうにIaaSを使用してPaaSとなっているのか、Herokuのアーキテクチャを見ていきましょう（**図3**）。

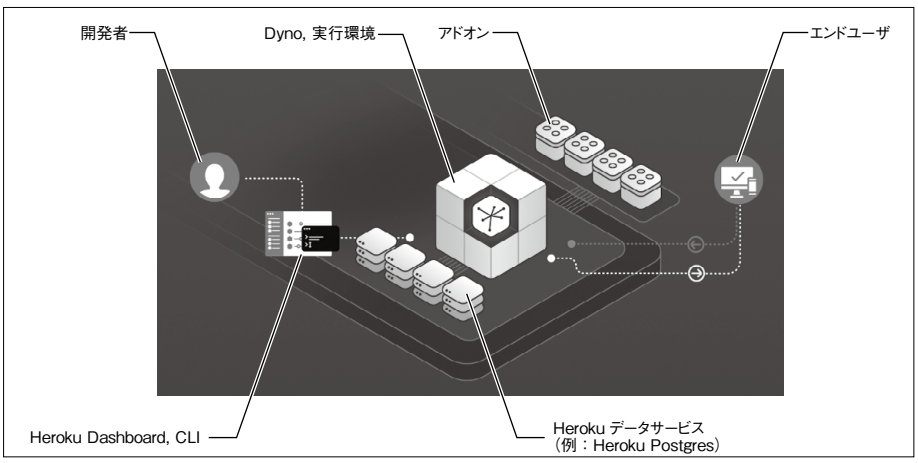
Deploy（デプロイ）

Herokuの使用は、アプリをデプロイすることから始まります。コードはHerokuが管理しているGitへとpushされ、その後buildpackを使用してSlugコンパイルと呼ばれるステップが走ります。buildpackに従い、Slugコンパイラが依存関係、プログラミング言語を読み取って適切に取得・設定をし、それらを包んだSlugと呼ばれるオブジェクトを作成します。このSlugオブジェクトはすでに依存関係などが解決された状態ですので、実行の準備ができた状態になります。

このSlug、つまりアプリのコードとは別に、Config vars（設定変数）の設定ができます。これらの値は実際にアプリが走るときに環境変数として取得できます。HerokuではThe Twelve-Factor Appに則って、こういった設定をコード（Slug）から厳密に分離することを推奨しており、そのしくみを提供しています。Config varsの例としては、データベースの接続情報

注10）URL https://www.heroku.com/form/private-spaces-webinar

▼図3　Herokuのアーキテクチャ



などが挙げられます。

これに加えて、データベースといったようなアドオンを適宜追加します。多くのアドオンは、追加をすればそのほかに特別何の設定もする必要がなく、すぐ使い始められます。

このSlug、Config vars、そしてアドオンをまとめて1つの「リリース」という単位になり、これらがHerokuのデプロイになります。

Runime（実行環境）

デプロイを通じて、アプリが動くためのベースができました。Slugオブジェクトは実行の準備ができた状態ですし、データベースも動いています。では、実際のWebアプリを仮定したとき、これらはどのようにHeroku上で実行されているのでしょうか。

Dynoとプロセス

HerokuではアプリはDynoと呼ばれるコンテナ単位で動くことは前に説明しました。このDynoの実体はLinux containers（LXC）であり、アプリが実際に走るときは、このLXCにSlugオブジェクトをダウンロードし、Config varsを環境変数として設定します。

Herokuでは大量のEC2インスタンスをプールしており、Dynoのサイズによって1つのEC2インスタンスに複数のDynoを載せたり（マルチテナント）、1つのEC2インスタンスに1つのDynoのみを載せたり（シングルテナント）といったことができます。インスタンスのプールがあるからこそ、Dynoを起動するときに実際に必要なことはSlugをダウンロードすることのみとなり、非常に早くユーザのアプリが動くような環境を準備できるのです。また、実際に動くRuntime部分とSlugが分離されているため、Dynoの数を増やしたいときも簡単にスケールできます。これらDynoの立ち上げなどは、Dyno managerと呼ばれるものが管理しています。たとえば、Dynoが走っているEC2インスタンス上で何か異常を検知した場合は、Dyno managerが、このDynoを別のEC2インスタンスにすばやく移動することでダウンタイムを防ぎます。

アプリにはそれぞれプロセスというものがProcfileを用いて定義され、このプロセス単位でDynoを立ち上げられます。たとえば、Rubyのアプリで Pumaというアプリサーバを用いてWebプロセスを立ち上げたいとしましょう。Webプロセスの定義は`bundle exec puma -C config/puma.rb`といった具合になります。Dynoはこの単位で、縦にも（サイズの上下）横にも（個数の上下）スケールできます。

また、こういった定義されたプロセス以外にも、one-off Dynoと呼ばれる一時的なDynoを作成できます。このone-off Dynoではたとえばbashを起動しインタラクティブに会話できますので、実際にSlugコンパイル後のコードがDyno上でどうデプロイされているかの確認などができます。しかし、これは実際に走っているDynoにSSH接続しているわけではなく、新しいまったく別のLXCをbashプロセス用に1つ作ってある、ということに注意してください。

すべてのDynoはephemeral filesystem（揮発性のファイルシステム）を採用していますので、Dyno間でファイルの共有はできず、またファイルもDynoの寿命[注11]とともに消えてしまいます。ここは、馴染みのない人には設計のうえで難しい部分があるかもしれませんが、データはデータストアやS3など外部に保持するようにしましょう。DynoやDyno managerについて詳しくは公式サイトのドキュメント[注12]を参照してください。

Herokuアプリに対するリクエスト処理の流れ

では実際に、複数のDynoで`bundle exec puma -C config/puma.rb`というプロセスを立ち上げた際に、リクエストはどのように処理されるのでしょうか。

注11）最大約1日、デプロイや再起動ごとに入れ替わる。
注12）URL https://devcenter.heroku.com/articles/Dynos

Herokuでアプリを作成すると、最初に「アプリ名.herokuapp.com」といったアプリ独自のドメインが付与されます。Webプロセスが走っているDynoへは、このドメインを叩くことによりアクセスできます。もちろんドメイン名はカスタムのものも使用できます。

このドメインによって送られたリクエストは、一度Herokuプラットフォーム共通のElastic Load Balancing（ELB）[注13]へ行き、そのELBが背後に控える大量のHeroku Routerインスタンスへリクエストを送ります。Heroku Routerでは、リクエストのHOSTヘッダの値を読み、どのアプリ名へのリクエストかを判別します。

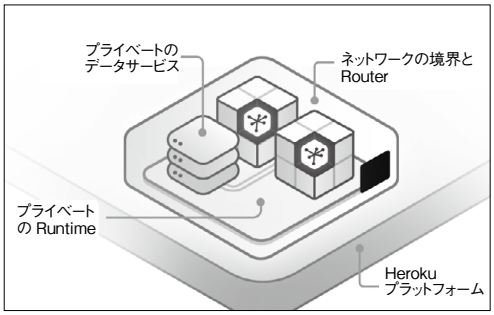
WebプロセスDynoが1つの場合はすべてのリクエストをそのDynoへ送り、もし複数ある場合はランダムなDynoにリクエストを送信します。それぞれのDynoはインスタンス上でポートがアサインされていますので、マルチテナントの場合もポートを指定することにより、目的のDynoへとリクエストを送れます。Heroku Routerについてもっと興味がある人は、ドキュメント[注14]を参照してください。

アプリのログは、すべてstdoutに吐いてもらうと「logplex」と呼ばれるログ収集エンジンによって各Dynoから収集され、それらをheroku logsというコマンドで見ることができます。さらに独自のlog drain[注15]の設定もできますので、任意のログサーバにログを転送して保存することもできます。また、ログ関連はアドオンが充実しており、アドオンを追加するだけで自動的にlog drainが設定され、アドオン側にログが保存されていきます。

Common Runtimeと Private Spaces

これまで説明したアーキテクチャはCommon Runtime、いわゆるパブリッククラウドのアーキテクチャとなります。Herokuはそのほかにも Private Spaces、いわゆるプライベートクラウドを提供しています。

▼図4　HerokuのPrivate Spacesのイメージ



Private SpacesではAmazon VPCという仮想プライベートクラウドを利用して、1つのSpaceごとに1つのVPCが割り当てられ、そのSpaceの中に専用のRouterおよびRuntimeが配置された、非常に独立した空間となっています。また、このVPCの中にはデータベースなどを入れることもできます。1つのSpaceがミニHerokuである、といった感じでしょうか（**図4**）。

Common Runtimeを使用するうえでは、どうしても他者と共有しているリソースが少なからずあります。たとえば、ELBやRouter、マルチテナントの場合はRuntimeインスタンスまで共有しています。こういったことによる恩恵も大きいですが、ほかのアプリの影響を受けてしまう可能性もゼロではありません。Private Spacesはその高い独立性により、セキュリティだけではなく他者からの影響を排除できます。

おわりに

PaaSはインフラを気にせずにアプリ開発に集中するのに非常に適したサービスです。また、簡単に始められるのも大きな特徴です。まだ触ったことがない人は、この機会にアプリを作ってHerokuにデプロイしてみてはいかがでしょうか。SD

注13） AWSのロードバランスサービス。
注14） URL https://devcenter.heroku.com/articles/http-routing
注15） ログを外部に排出できる機能。

第6章 ハードウェアからしっかり解説

“仮想データセンター”を目指したさくらのクラウド

Author 篠田 真一（Shinichi Shinoda） Blog http://kuroeveryday.blogspot.com/
Author 宮堂 達也（Tatsuya Miyado） Blog http://hogesuke.hateblo.jp/
さくらインターネット株式会社

国産のクラウドサービス「さくらのクラウド」について、ハードウェアからネットワーク、さらにはコントロールパネルまで、どのような構成の下で、どのような工夫がなされているのかを解説していきます。データセンター事業者ならではの視点で、“仮想データセンター”としてのクラウドをひも解きます。

さくらのクラウドの特長

さくらのクラウド[注1]とは、さくらインターネットが提供するパブリック・クラウドです。コントロールパネル（**図1**）を備え、トラフィック課金のないさまざまなスペックの仮想サーバを利用できます。ほかにも、スイッチやVPCルータ、DNSや監視のアプライアンス機能、マップ機能なども備えています。APIを公開しているため、プログラマブルな処理やCLIを使った操作、Terraformとの連携もできます。

注1） URL http://cloud.sakura.ad.jp

開発者志向のシンプルなIaaS

開発コンセプトは、「開発者志向のシンプルなクラウドの提供」です。弊社はこれまでレンタルサーバ、専用サーバ、VPS（仮想サーバ）の各サービスを提供していました。しかし、いずれも初期費用が必要であり、かつ課金は月額単位でした。

「必要なスペックのサーバを必要なとき、必要なだけ使えるだけでなく、スイッチやネットワークも含めた仮想データセンターとしてのクラウドを提供しよう」。そうして、さくらのクラウドは2011年11月にサービスを提供開始し、今年で5周年を迎えました。

▼図1 コントロールパネル

使い勝手の良いクラウドを目指して

サービス開始以降、お客様からのご要望を取り入れながら、さまざまな機能を追加・開発し続けています（**表1**）。

シンプルな料金体系

さくらのクラウドの仮想サーバでは、一般的なクラウドとは異なりトラフィックやI/Oに対する課金はありません。課金は時間単位ですが、月額料金に上限を設けているため、使い続ける場合もコスト感が得られやすい特長があります。

自由に組めるネットワークとマップ機能

ネットワークの概念は、クラウドだからといって特殊な要素はありません。物理サーバの場合、ローカルの安全な環境にサーバを置きたければ、スイッチを追加し、その下にサーバを置きます。同様にさくらのクラウドでも、スイッチの追加・接続をコントロールパネル上で操作できます。

そして、さくらのクラウドで一番ユニークなのがマップ機能です。サーバやスイッチ、VPCルータの接続やIPアドレスの情報を画面で一覧できるだけでなく、そのまま詳細の確認や設定変更・接続もできます。まさに、「自分の仮想データセンター内で機器の操作が行える」ような感覚を大切にしています。

専用サーバやVPSとも連携

ブリッジ接続機能やハイブリッド接続を使えば、専用サーバやVPS、データセンター（ハウジング）環境との接続もできます。とくにブリッジ接続であれば、コントロールパネルを通して接続先のスイッチ情報の確認や操作もできます。

Arukas、さくらのIoT Platform

さくらのクラウドは24時間365日の保守体制を整えています。この基盤を通して、Arukasやさくらの IoT Platformを展開しています。

▼表1　さくらのクラウド、機能一覧表

カテゴリ	機能名
サーバ／ディスク	サーバ
	ディスク
	アーカイブ
	ISOイメージ
	スタートアップスクリプト
ネットワーク	スイッチ
	ルータ＋スイッチ
	ブリッジ接続
セキュリティ	VPCルータ
	パケットフィルタ
負荷分散	ロードバランサ
	GSLB（広域負荷分散）
CDN	ウェブアクセラレータ
アクセスコントロール	2段階認証
	アクセスレベル
オプションサービス	DNS
	シンプル監視
	オブジェクトストレージ
	データベースアプライアンス
サービス間接続	ハイブリッド接続
	プライベートリンク

コンテナ・ホスティング・サービス「Arukas（β）」

Dockerコンテナをデプロイするには、一般的に何らかのサーバを準備し、その上にDockerをセットアップする必要があります。Arukasを使えば、サーバを使わずにコンテナを実行できるだけでなく、インターネット上に、自動的にエンドポイントの作成やポートの割り当てを行えます。

Arukasuはコンテナを使ったサービスをすぐにデプロイし、スケールさせたい場合に有用です。

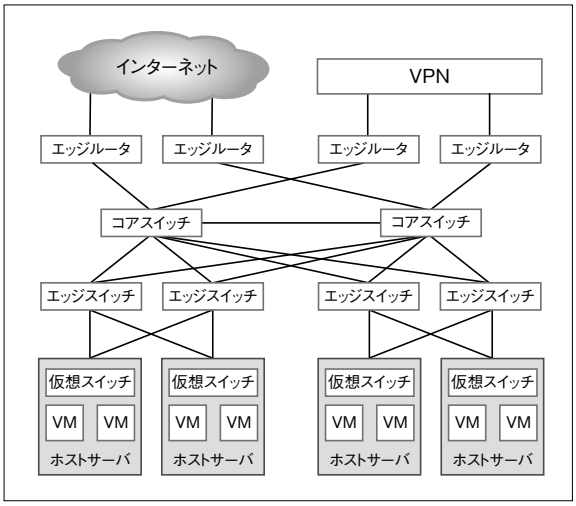
IoTのためのプラットフォーム・サービス「さくらのIoT Platform（α）」

クラウドEXPO（2016年春）で「みまもりポット」というIoTデバイスを展示しました。お湯を出した時間や温度を

センサーで感知し、そのデータをSlackにポストするというものでした。このデバイスは、2016年4月に公開した「さくらのIoT Platform」を使って実現しています。

本サービスでは、弊社が提供している通信モジュールをマイコン／シングルボードコンピュータに接続することで、3G/LTE回線で通信できます。閉域網なのでインターネットを通らず、安全にデータの収集・蓄積を行うことができます。また、WebSocketとWebHookを使って蓄積したデータを通信することで、さまざまなアイデアを実現できます。

▼図2　ネットワーク図



さくらのクラウドの基盤

さくらのクラウドは、国内最大規模のバックボーンネットワークを使って、お客様に安全で安定したサービスを提供しています。はじめに、サービスの根幹を担っているハードウェアやネットワークといったクラウドの基盤について説明します。

ホストサーバ、ストレージなどのハードウェア

弊社の北海道石狩と東京のデータセンター内には、大量のホストサーバやストレージが設置されています。

ホストサーバはCPUとメモリを大量に搭載しており、このリソースをVMに割り当て、「サーバ」としてお客様に提供しています。

ストレージはホストサーバとは切り離され、大量のSSDやHDDを格納し、メイン用／バックアップ用／アーカイブ用と分けて管理しています。このストレージの領域を切り分け、「ディスク」としてお客様に提供しています。

ネットワーク

先に説明したハードウェアが、インターネットやデータセンター内、データセンター間でどのようにつながれているか説明します。

インターネット（外部ネットワーク）は、データセンター内のエッジルータに接続しています。エッジルータからL2ネットワークでコアスイッチ、エッジスイッチに、そしてエッジスイッチからホストサーバの仮想スイッチに接続しています（**図2**）。さらにホストサーバ・ストレージ間は、10GbEスイッチで接続しています。

可用性を高めるための取り組み

ホストサーバは電源やNIC（ネットワークインターフェースカード）、OS起動用ディスクの冗長化を行い、可用性を高める対策をしています。しかし、突然の故障や、ソフトウェアがクラッシュしたりといった状況で仮想サーバがダウンしてしまう場合もあります。そのような場合に備え、自動的に別のホストで仮想サーバを再起動させることで、極力早く復旧できるしくみを実装しています。

ストレージは、電源やコントローラ、HDD、SSDのパーツごとにそれぞれ冗長化されています。ホストサーバとストレージ間の接続も2系統のSAN（ストレージエリアネットワーク）となっており、故障や障害からサービスを保護する設計になっています。

ネットワークは、エッジルータやコアスイッ

チ、エッジスイッチがそれぞれ冗長化され、単一障害点を減らしています。このネットワーク構成により、高い可用性を実現しています。

クラウドの基盤

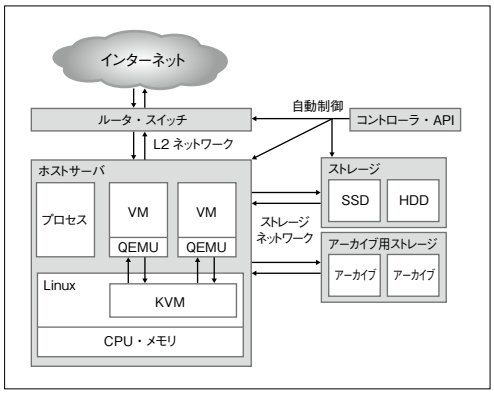
ここまでハードウェアとネットワーク、その可用性について説明してきました。ここからは、そのハードウェア・ネットワーク上でどのようにIaaSを提供しているのか説明します。

さくらのクラウドは、Linux+KVM+QEMUで動作しています。KVMはハイパーバイザ型の仮想化基盤で、ホストサーバのCPUやメモリを仮想化しています。QEMUも仮想化ソフトウェアですが、こちらはVMの入出力を仮想化しています（**図3**）。

KVM・QEMUを管理、操作するためにはコントローラが必要です。OpenStackやCloudStackといったオープンソースソフトウェアもありますが、さくらのクラウドは国産クラウドでは珍しく、そのコントローラをフルスクラッチしています。背景には、サービスコンセプトの「開発者指向のシンプルなIaaS」があり、既存のソフトウェアでは「私たちが求めること」が実現できなかったからです。

コントローラ・APIについては、次節にて詳しく説明します。

▼図3　クラウド基盤



さくらのクラウドのコントローラ

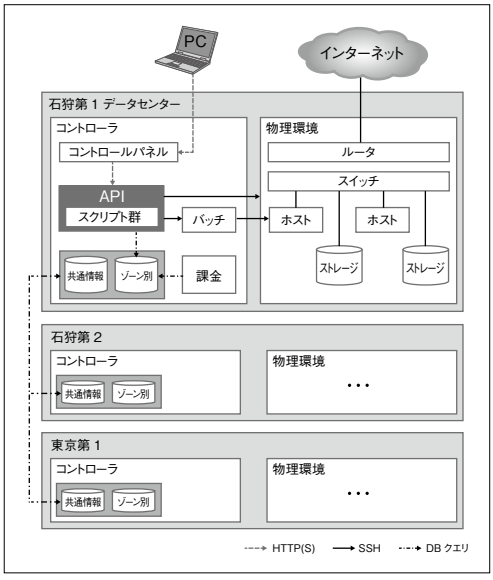
さくらのクラウドのコントローラは、API、バッチ処理、課金処理、運用ツールなどのコントローラ群で構成されています。これらのコントローラが、先ほど説明したクラウドの基盤を管理・操作しています（**図4**）。

APIはさくらのクラウドの司令塔

さくらのクラウドでは、APIをフルスクラッチで開発しており、PHPやPerl、Ruby、Node.jsで書かれた「基盤を操作するスクリプト群」を束ねるような働きをしています。

RESTfulなAPIですので、POSTでリソースを作成、PUTでリソースの設定変更、DELETEでリソースの削除、といった具合にエンドポイントが設定されています。裏側ではエンドポイントに応じたスクリプトを実行して、VMをコントロールしたり、ディスクの設定ファイルを書き換えたりとさまざまな処理をしています。

▼図4　クラウドコントローラ



サーバをつくる裏側

実際にサーバがつくられ起動するまでに裏側で何が行われているか、順を追って説明していきます。

❶サーバの構成情報（CPUとメモリ）を保存する

POST /serverが実行されると、指定された構成がゾーン別DBに保存されます。この時点では、まだVMは作られていません。

❷ディスクを作成する

POST /diskが実行されると、ストレージにディスク領域を作成し、DDコマンドなどを使ってOSイメージやアーカイブをコピーします。

❸ディスクの内容を編集する

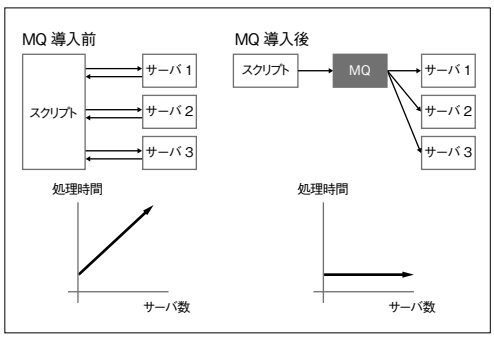
PUT /disk/:diskid:/configが実行されると、作成されたディスクの設定を変更します。スクリプトがファイルシステムの内部を検索し、OSの種類を特定、OSに見合った場所にある設定ファイルを書き換えます。VMのホスト名を変更する場合は、CentOS6なら/etc/sysconfig/network、Ubuntuなら/etc/hostnameを書き換えるといった具合です。

❹ディスクを接続し、サーバを起動する

PUT /server/:serverid:/powerが実行されると、ディスクをiSCSIでホストサーバに接続します。次に、ゾーン別DBに保存されているサーバ構成でVMを作成し、VMからディスクにアクセスできる状態にします。最後にVMの起動コマンドを実行し、利用できるようになります。

このようにしてサーバをつくり、お客様に提供しています。

▼図5　MQの導入



性能向上のためにMQを導入

APIの性能向上のため、MQの導入を進めています。MQとはMessage Queueの略で、リクエストを一時的に貯めこんで随時実行するしくみです。また、リクエストを処理したいサーバがメッセージをサブスクライブすることで、リクエストの一斉配信を実現できます。

たとえば、パケットフィルタの設定を変更するときに、このMQのしくみを使っています（**図5**）。導入前は、パケットフィルタが適用されているサーバを、スクリプトが巡回して適用していました。そのため、サーバ数が多くなると処理時間も比例して長くなっていました。現在はMQを導入し、全サーバがMQをサブスクライブするように設定しています。そのため、サーバ数に関係なく一定、かつ短時間にパケットフィルタの変更が適用できるようになったのです。

さくらのクラウドのコントロールパネル

さくらのクラウドのコントロールパネルはSPA（シングルページアプリケーション）です。JavaScriptを通してレンダリングを行い、動的にUIを構築します。そして、前述のRESTAPIをXMLHttpRequestを用いて呼び出すことで、リソースの取得や作成・削除、編集などの操作を行います。

お客様にとって使いやすいコントロールパネルとするため、アクセスレベル機能やマップ機能を搭載しています。次節で、それらについて詳しく紹介します。

アクセスレベル機能

サーバやディスクといったリソースは「アカウント」ごとに管理されます。そして、そのアカウントにアクセス可能な「ユーザ」を作成します。

ユーザには各アカウントに対するアクセスレベルを4段階で設定できます。

❶リソース閲覧
❷電源操作
❸設定編集
❹作成・削除

下にいくほど強い権限となり、自身より弱い権限を内包します。

これらの権限を各ユーザの役割に合わせて設定することで、操作に制限を設けられます（**図6**）。これにより、想定しないユーザによる想定しない操作を防ぐことができます。

アカウントごとに異なる権限を設定できるため、ユーザの関わり方が異なる場合にも対応できます。たとえば、あるアカウントにおいて、一からインフラの構築に携わる役割である場合には「作成・削除」権限を付与し、すでに構築されたインフラを運用する役割である場合には「設定編集」権限を付与するといった具合です。

また、これらとは別に請求情報の閲覧についても別途権限を設定できます。経理担当者など限られた人のみに請求情報の閲覧を制限したいといった要望を実現します。

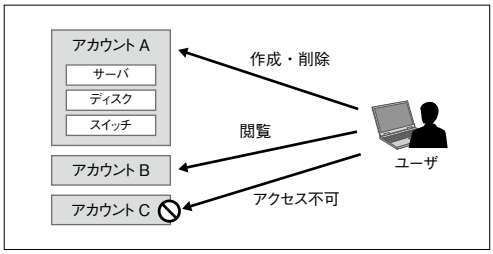
さらに、コントロールパネルから発行できるAPIキーにも、同様の4段階の権限と請求情報閲覧の権限を設定できます。

マップ機能

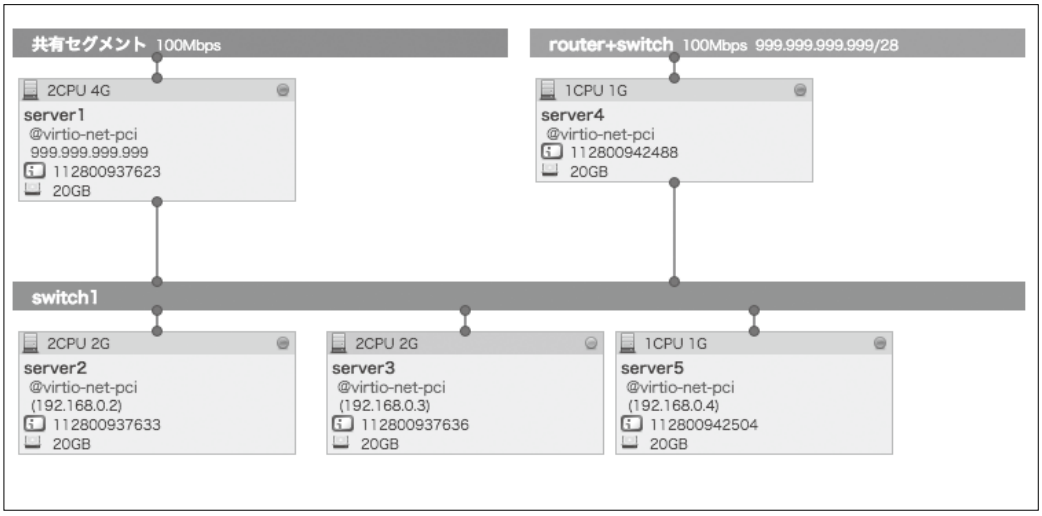
さくらのクラウドの中で、とくに好評をいただいているものが「マップ機能」です。これは、作成したサーバやスイッチなどの関係を示したネットワーク構成図をコントロールパネル上で閲覧・編集できる機能です（**図7**）。

この機能で、作成したリソースを俯瞰でき、全体を把握しやすくなります。サーバに接続し

▼図6　アクセスレベルの設定例



▼図7　マップ機能の表示例



たスイッチのつなぎ変えもドラッグ＆ドロップで簡単に行うことができます。

Sandboxというテスト環境と併用することで、より効果的にご利用いただけます。Sandboxではリソースの作成に料金は発生いたしません。ですので、マップ機能をネットワーク構成の検討にご使用いただけます。マップの印刷・画像での保存機能も実装されているので、ほかの方と共有しながらの検討も行いやすくなっています。

この機能はJavaScriptライブラリのjQueryをベースに作成されています。また、ドラッグ＆ドロップやタッチ操作にはHammer.jsを用いています。

今後もますます便利なコントロールパネルとするため、表示の高速化の取り組みや、複数リソースの一括操作を可能にするといった操作性の向上を図っていきたいと考えています。

さくらのクラウドの最新サービス

データベースアプライアンス

2016年8月10日に、「データベースアプライアンス」をリリースしました。自動・手動バックアップを始め、世代管理、セキュリティ設定などの機能があります。

お客様にとっては作成後すぐ利用できるデータベースに見えますが、内部的にはお客様専用のデータベースサーバを作成し、そこにDBMSをインストール、各種設定を行うなど、次のような処理を行っています（PostgreSQLの場合）。

❶コントロールパネル、またはAPIからの作成リクエストを受ける
❷サーバとディスク2台（マスタとバックアップ）を作成
❸最小構成のCentOSをインストール

COLUMN

コントロールパネルに桜葉愛が登場?!

2016年4月1日、2次元枠採用された桜葉愛（さくらは あい）が声でお知らせする「音声通知機能Ω版」を1日限定でリリースしました。桜葉愛がコントロールパネルに現れて「作成しました」や「起動しました」など、さまざまな音声でお客様をサポートするための機能です（**図A**）。この機能を有効にすることで、処理が完了するまでコントロールパネルを見なくて良くなるので、お客様の貴重なお時間を無駄にしません。

桜葉愛（の中の処理）は、HTML5+JavaScriptで実装されています。話すためには、HTMLAudio Elementを使っています。HTTPレスポンスコードをもとに最適な音声を読み込み、意味が通じるように結合して再生しています。姿を見せるためには、Canvasを使っています。また、日付をもとに季節に応じたイラストをランダムに表示する遊び要素もあります。

この「音声通知機能Ω版」はリリース直後から大きな反響を呼び、「正式に機能として追加してほしい」とたくさんのリクエストをいただきました。

それから1ヵ月後、「サウンド通知機能」と名前を変えて正式リリースしました。大人の事情で音声ではなく効果音に変わりましたが、桜葉愛がサーバの作成や起動の操作を見守ってくれます。コントロールパネルをご利用になられるときは、ぜひ画面右下にある「サウンド」ボタンからサウンド通知機能を有効にしてみてください！

▼図A　桜葉愛が登場

❹phpPgAdmin用にApacheをインストール
❺PostgreSQLとphpPgAdminをインストール
❻コントロールパネルで入力した値をもとにApacheやPostgreSQLなどの設定を行う
❼データベースの初期化を行う
❽バックアップ用にcrontabの設定を行う
❾データベースアプライアンスの完成

このようにして、さくらのクラウドのデータベースアプライアンスは作られています。現在はプレビュー版ということもあり、無料でご利用いただけますので、ぜひお試しください。

シンプル監視

「シンプル監視」は、外形監視に特化した監視機能を提供します。監視のためにサーバを別途用意したり、ソフトウェアをインストールしたりする必要はありません。簡単・お手軽に使うことができるのが特徴です。

監視対象はIPアドレス、またはFQDNで指定します。さくらインターネットが提供するグローバルIPアドレスなら無償でご利用になれます。監視プロトコルはping、tcp、http、https、dns、ssh、smtp、pop3、snmpの9つに対応し、チェック間隔は1分から60分まで1分間隔で指定可能、さまざまな用途のサーバの監視に対応しています。

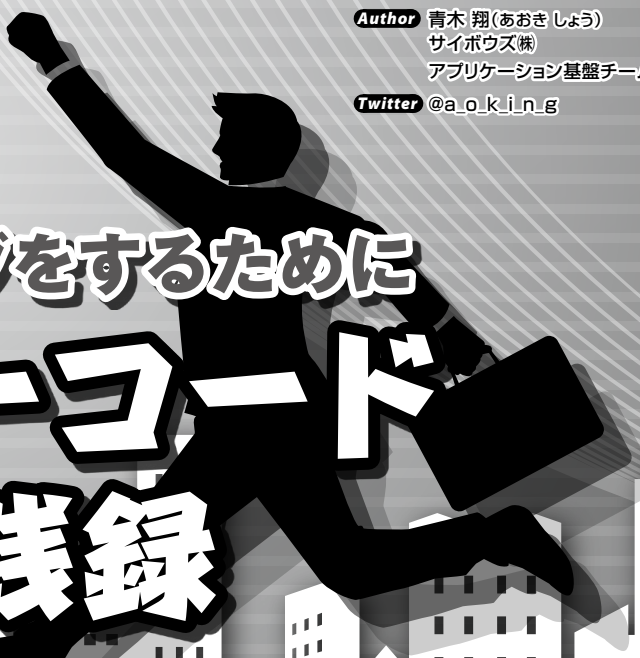
この機能の裏ではNagiosが動作しており、すべての監視を担っています。コントロールパネルから登録した設定値はNagiosのconfigにコンバートされ、監視対象に追加されます。異常を検知すると、MQにユーザへの通知メッセージを送信します。そして、MQからメッセージを受信した通知サーバが、メールやSlackに異常検知メッセージを送信します。

さくらのクラウドでは課金額が設定の値を超えた場合に通知する「料金アラート」という機能を提供しているのですが、実はこれもシンプル監視のしくみの上に実装されています。

簡単、お手軽に使える監視機能、ぜひお試しください。SD

第2特集

Author 青木 翔（あおき しょう）
サイボウズ㈱
アプリケーション基盤チーム
Twitter @a_o_k_i_n_g



恐れずにリファクタリングをするために レガシーコード改善実践録

サイボウズ流バグゼロまでの道のり

本番稼働中の製品なのにバグだらけ。しかもコードのンテナンス性が悪くて、別のバグを作り込まないかビクビクしながら修正を行う。日々のバグ対応に追われて、みんな疲弊……。そんな状況から、1つずつ安全にプログラムを改修し、着実に品質を向上させて、とうとうバグをゼロにした現場があります。高品質を達成するまでにはさまざまな工夫、取り組みがありました。

本特集では、その取り組みにかかわったエンジニアご自身に、過酷な状況の中から試行錯誤のすえに導き出したアイデアを紹介してもらいます。

第1章

ソフトウェアを徐々に高品質にするコードの直し方

Author 青木 翔(あおき しょう)
サイボウズ㈱ アプリケーション基盤チーム
Twitter @a_o_k_i_n_g
Illustration どこ ちゃるこ

稼動中の製品を安全に高品質化できるか?

みなさんはじめまして。サイボウズで働くエンジニアの青木(@a_o_k_i_n_g)です。サイボウズではクラウドでグループウェアを提供するサービス、cybozu.comを提供しています。cybozu.comは2016年9月時点で契約社数1万6,000社、ユーザ数は58万を超えており、筆者はそのcybozu.comを支えるミドルウェアを開発するチームに所属しています。

cybozu.comはOSS含め多数のミドルウェアに支えられていますが、自作しているものも多くあります。たとえばファイルサーバやメッセージキューイングを含むジョブの非同期処理システムなどです。

触れるのも怖いほど粗悪なコード

cybozu.comが公開された当初のこれらの自作ミドルウェア群は、正直言って品質が高くありませんでした。筆者のチームのミドルウェアが原因でサービスが停止することも珍しくなく、ユーザには多大な迷惑をかけました。ユーザだけでなく、弊社の運用部隊にもたいへんな迷惑をかけ、慌てて不具合調査して改修、緊急リリースということがしばしばありました。コードの悪さゆえに不具合の原因を特定するのにも時間がかかり、しばらくは「障害が発生したら再起動で対処」という対症療法でしのいでいたこともありました。

cybozu.comが公開された当時、この出来の悪いミドルウェア群によって、プログラマのみならず品質保証部やプロダクトマネジメント部、そして運用部隊も疲弊していました。

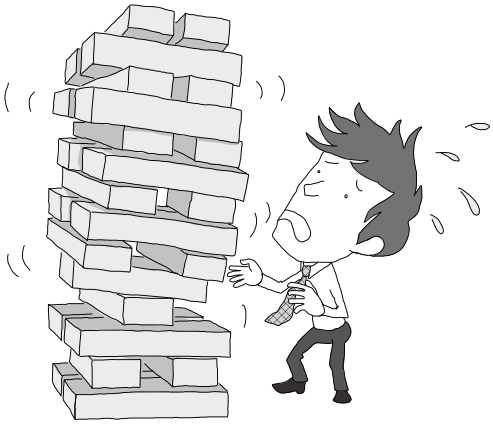
控えめに言っても当時のコードは粗悪でした。巨大なメソッドがひしめき合い、if文やfor文が何重にも重なり、そしてカプセル化されていないオブジェクトのフィールドを直に操作する、そんなコードでした。名は体を表しておらず、継承は直感に反し、同期処理の責任が各所に分散していました。そのようなコードで構成されたマルチスレッドなプログラムのデバッグは悲惨なものでした。

とある機能は誰も全体像を把握できなくなっており、「コードが仕様書」を体現していました。複雑な部分に手を入れるときは、ジェンガのような不安定さを感じながら、そして新たに不具合を仕込んでいるのではないかという不安を感じながら、コードに手を入れていました(**図1**)。

今のコードをメンテナンスし、かつ今後も機能を拡張していくのは困難です。でも少しずつでも改善していかなければ、未来になっても環境が変わっていないだろうと思いました。それは無限に押し寄せてくる不具合を改修し、絆創膏(ばんそうこう)の上に絆創膏を貼り続けるような仕事のように思えました。

実際、筆者は以前そんな環境に一度身を置いたことがあり、その苦痛はわかっているつもり

▼図1　粗悪なコードで作られた製品はまるでジェンガのよう……



です。筆者はそういった環境がどこか賽の河原みたいに思え、これは今立ち上がって改善せねばならない、というある種の強迫観念のようなものに捕らわれたのです。それに、筆者だけが苦労するならまだしも、いずれやってくるであろう後輩にこのコードを渡すというのはあまりに申しわけがなく、筆者のプログラマとしてのプライドがそれを許せませんでした。

粗悪なコードが生まれてしまったものはもう過去のことなので今毒づいてもしかたありません。でも、これからの対応で今後の品質を変える力を筆者たちプログラマは持っています。今あるコードを、今後安定した大樹のようなコードにするか、それとも噛み合わせの悪い十徳ナイフのようなコードにするか、その分水嶺に筆者たちは立っていました。筆者は、高品質化の道に進むことを決心しました。

地道な努力で高品質化を達成

結論から言うと、筆者たちは高品質なコードを手に入れたと言える状況になったと思います。既知の不具合はすべて改修し、コードの品質が良くない部分もほとんどすべて駆逐しました。メンテナンス不可能なレベルのコードをいくつも葬り去り、そのたびにシンプルなコードと十分な量のテストで武装し、バグが入り込まないようにしてきました。日々ログを洗い、通常ならほとんど発見不可能な不具合をいくつも発掘し、1つ残らず改修しました。その結果、我々は高品質なコードを手に入れ、それだけでなくプロダクトとしての安定性や拡張性も手に入れました。

本特集では、いかにして我々が悲惨な状況から高品質なコードを手に入れたのかについて、我々の足跡と知見を記そうと思います[注1]。

使っていないコードを破棄すべし

さまざまな機能を追加したりバグフィックスしたりして、コードは日々増加していきます。日々の修正は小さなものでも、それが積み重なると工数に影響を与え始めます。機能をまるごと削除したのにコードが残っていたり、「いつか使うだろう」と思って実装した関数が使われていなかったりなどの理由がよくあります。

ストラテジパターンで完全に使われなくなったストラテジが残ることもあります。ひどいときでは、あるxxxという関数やクラスを高速化するなどで再実装に近いことを行い、xxx2やxxxFastという名前で実装し、元の実装がまるごと残っているようなケースもあります。過去のバージョンのコードをすべてコメントアウトして残しておくプロジェクトもあるようですが、コメントアウトされているとはいえ、コードを読みにくくするという点では同罪です。VCS[注2]が発達している今の時代にそんなことをするまっとうな理由はないでしょう。

このような使われていないコードは、保守性を悪化させます。新機能を追加するときにどこに機能を追加すればいいのかがわかりにくくなったり、不具合の原因を探るときに時間をいたずらに消費する原因になったりします。とくに、障害が発生して早急に原因を突き止めなくては

注1）我々のプロダクトはJavaで実装されています。よって、紹介するツールやコード例はJavaです。
注2）Version Control System：バージョン管理システム。

▼図2　不要なコードの削除はもっと評価されるべき

ならないときには、そのような無用な時間の消費はサービス停止時間の増加につながります。

というわけで、不要なコードは削除しましょう。不要なコードの削除ならば、それは稼働しているコードを修正するよりも気軽に行えます。もちろんそのコードが真に不要なのかどうかは厳重にチェックしましょう。

この節で筆者が主張したいことはこれにつきます。**コード行数を減らしたプルリクエストをもっと評価しよう**（**図2**）。不要なコードの削除は、コードを追加することと同じくらい重要です。コードの行数の分が仕事量や生産性に比例しているというような考えは時代遅れと言わざるを得ません。不要なコードを削除したり、既存の機能をより簡潔に表現できたりしたならば、それは今後のメンテナンス工数の削減という意味では大きな意味を持っています。

レガシーコードは機能まるごと削除する発想も必要

レガシーコードに手を入れたくないという理由もよくわかります。コードを修正するということは、バグが入り込む余地が出てくるからです。

筆者のチームのコードでもそんなコードが多数ありました。1つはとあるWebアプリケーションのセッションマネージャークラスだったのですが、改行をすることも憚られるほどのコードでした。テストも満足に書けず、その機能一帯は誰も手を入れることができない「聖域」になっていました。しかしあるとき、諸事情によりセッションの管理方法を切り替えたため、そのセッションマネージャーをまるごと破棄できました。

とあるサムネイル作成機能も提供していたのですが、その仕様とコードはあまりにひどく、サムネイルという機能の重要度の低さの割に大量のメンテナンス工数が投入されていました。しばらくはそのコードを四苦八苦しつつメンテナンスしていたのですが、これは小手先では太刀打ちできるものではないと判断し、サムネイル作成周りの機能を再設計して新しいコンポーネントを実装しました。機能の再設計にはそれなりに工数がかかりますし、ほかのチームとのやりとりもあるので手間はかかりますが、再実装されたサムネイル作成ツールは冗談抜きで保守性が数十倍になったのではと思います。

このように、歴史が詰まっているコードだからこそ、何か別のプロダクトや手法でまるごと消し去ることができる場合もあります。イニシャルコストが高くつくこともありますが、メンテナンスコストを削れるなら利益を出すはずです。これができるケースはあまり多くはありませんが、コードの聖域を取り除くうえでは一撃必殺に等しい一手になり得ます。

2種類の不具合の直し方

たいていのプロジェクトではさまざまなしがらみや歴史的背景があり、簡単にはコードを削除したり修正できなかったりします。そこで、筆者は不具合改修をする際に、試験範囲に影響のない範囲で周囲のコードもきれいにする、と

いう方法でコードを少しずつ直していきました。

バグの直し方には2種類あります。1つめは、コードの差分を最小にしてレビュー時の工数をできるだけ少なくし、影響範囲も限りなく小さくするというものです。このやり方で直すときは、修正個所が1行だけということも珍しくありません。レビューも簡単で、diffを見た時点で一目瞭然なことも多いです。

2つめの直し方は、不具合を改修するとともにコードを少しずつきれいに直していく方法です。diffが増え、したがってレビュー時にも少し工数がかかりますが、長期的に見てコードの品質を高めていけます。

筆者の経験上、前者のパターンでは高品質なコードになることはないと判断しました。できるだけ後者の方針で直すように心がけました。試験範囲には影響が出ないよう、修正をやり過ぎないように注意しつつ、少しずつ少しずつコードをきれいにしていきます。すぐに効果が出るものではないですが、着実にコードの品質が上がっていきます。

コードの本質的でない部分をなるべく隠蔽しよう

コードを見る際、本質的でない部分に気を取られてしまうことがしばしばあります。よくあるのが、コードのフォーマットです。コードのフォーマットはあくまでフォーマットであり、本質的な部分ではありません。複数人で開発する場合、レビュー時に不要なdiffが出て混乱することもあります。そのような本質的でない部分に時間を取られることがないよう、フォーマットをそろえるようフォーマッタを用意し、常にそれを用いるようにしましょう。

また、たとえばJavaではSetter/Getterの記述が冗長で、クラス内のコードを大きく占めていることがあります(**リスト1**)。Javaに限った解法ですが、lombokというライブラリで冗長なコードを劇的に短くできます。lombokはいわばJavaのプリプロセッサのようなもので、アノテーションを指定すればSetter/Getterやコンストラクタ、toStringメソッドやequals/hashCodeメソッドを自動生成してくれます。

▼リスト1　通常のSetter/Getterの記述例

```
public class User {
    private String name;
    private int age;

    public void setName(String name) {
        this.name = name
    }
    public String getName() {
        return this.name;
    }

    // age プロパティも同様
}
```

▼リスト2　lombokを使えば@Setter、@Getterだけ書けば良い

```
@Setter
@Getter
public class User {
    private String name;
    private int age;
}
```

リスト1の例では、@Setter、@GetterアノテーションをUserクラスに付与することで、自動でそのクラスのフィールドのSetter/Getterを生成してくれます(**リスト2**)。

もちろん、フィールドごとにSetterのみを生成したり可視性を指定したりと細やかな指定も行えます。ほかにも@DataでコンストラクタやtoString/equals/hashCodeも生成してくれるなど、お決まりのコードがわずかなアノテーションで済むようになります。

lombokを導入するとJavaの冗長なコードを削減でき、コード内のロジックの濃度が上がります。それだけ本質的な部分に集中できます。lombokはコンパイル時に処理を行うという性質上、IDE[注3]にプラグインをインストールする必要がありますが、EclipseでもIntelliJ IDEAでもスムーズに動いています。lombokはたいへん

注3) Integrated Development Environment：統合開発環境。

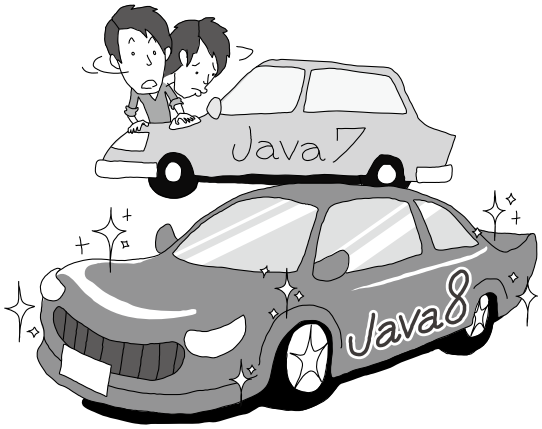
すばらしいツールなので、ぜひ導入してみてください。

コードや環境は時間が経つだけで劣化していく

コードが劣化する原因は人の手によって複雑化していくという理由も大きいですが、それだけではありません。時代が進むに連れてプログラミングの技法や言語仕様は進化していくので、それに取り残されていくとレガシーコードになっていきます。**コードは、時間が経つだけで劣化していくのです**（**図3**）。

極端な例ですが、今新規プロジェクトで用いる言語にCOBOLを選択する人はまずいないでしょう。それと同様に、たとえばJavaも6から7、7から8へと進化していっており、モダンな機能が取り込まれています。同じJavaとはいえ、過去のバージョンでは正当だった記法も、現行のバージョンでは警告が出るというようなこともあります。たいてい、新しい言語ほどよりシンプルな記法やモダンなメソッドが導入されているので、できる限り最新のものを導入するように心がけましょう。

▼図3　コードは時間が経つだけで劣化していく



▼リスト3　Runnableインスタンスの生成（Java 7以前の書き方）

```
Runnable runnable = new Runnable() {
  @Override
  public void run() {
  doSomething();
  }
};
```

▼リスト4　削除されていない、かつaから始まるユーザが存在するかどうかを調べる（Java 7以前の書き方）

```
List<User> users = ...
for (User user : users) {
  if (!user.isDeleted()) {
    if (user.getName().startsWith("a")) {
      return true;
    }
  }
}
return false;
```

例1

Java 7以前ではRunnableインスタンスを生成するには、**リスト3**のような記述が必要でした。

Java 8でラムダが導入されてからは、これだけで済みます。

```
Runnable runnable = () -> doSomething();
```

例2

「ユーザのリストから、削除されておらず、かつaから始まるユーザが存在するかどうか調べたい」。そんなときには**リスト4**のようなコードになります。

Java 8のStream APIを用いれば、**リスト5**のように記述できます。

◆ ◆ ◆

環境が最新のものに追随できないと、システムがレガシーになっていく以外にも困ることが

▼リスト5　削除されていない、かつaから始まるユーザが存在するかどうかを調べる（Java 8での書き方）

```
return users.stream().filter(u -> !u.isDeleted()).map(u -> u.getName()).anyMatch(e -> ➥
e.startsWith("a"));
```

出てきます。開発環境の構築が困難になったり、またそれが原因でビルドシステムがブラックボックスになったりするということが起こり得ます。最新のバージョンでのみ動くツール、たとえばプロファイラのようなものが古い環境だと動かないということもあります。

時代はどんどん良い方向に向かっており、開発環境をできるだけ新しいものに追随できれば、それだけでさまざまな恩恵を受けることができます。最新とは言わずとも、それなりに主流やそれに近い環境で開発できるしくみを構築しましょう。

コードの品質と性能について

性能を追い求めると一般的にコードは複雑化します。性能のためにキャッシュ機構を導入したり、オブジェクトに本来は持たせなくて良い余分なプロパティを持たせて値を使いまわしたり、変数のスコープを本来の用途より広げたり。このように、コードの読みやすさや保守性という観点と相容れないケースがしばしばあります。設計段階で性能の点でも十分に考慮されているコードは、シンプルかつ性能も出るのですが、小手先のチューニングを施したコードはシンプルとは言いがたいコードになります。

元から複雑なコードに対して性能を求めた改善を行うと、より複雑になり、それだけ不具合が混じる可能性が高まります。不具合含みの性能改善にいったいどれだけ価値があるのでしょう？　保守工数の観点ではまったく時間の節約にならず、小手先のチューニングを施した質の悪いコードの手入れに時間を奪われることでしょう。

『プログラミング作法』(第5章の参考図書(2)を参照)という古典的名著にも書かれているとおり、「早すぎる最適化はするな」というルールはほとんどの場合正しいです。

まず第1にコードのシンプルさを心がけて実装し、それで性能テストや運用で問題が出た場合にチューニングをする、というのが正攻法でしょう。ちなみに早すぎる最適化が問題ではないケースは、性能の重要度が高いソフトウェアを実装するときです。たとえば、RDBMS(Relational DataBase Management System)やKVS(Key-Value Store)を自前で実装する際、おそらく性能は重要な観点になるはずです。設計が大きく性能を左右するので、初期段階の設計時での性能問題のあぶり出しが重要になります。

コードの品質を上げると、性能を追求しやすくなります。ドナルド・クヌース先生[注4]も述べているように、「通常、実行時間の半分以上はプログラムの4パーセント未満の部分に費やされる」というのは体感的にもそのとおりのように思います。

役割や責任が明確に分割されていると、コードを拡張しやすくなります。すると、性能のためにキャッシュ機構を入れるとしても、影響範囲やコードの修正量が小さく済みます。結果として、コードの品質を上げることは、粗悪なコードで性能を追求するよりも、性能や保守工数両方の観点で良い結果をもたらしてくれます。

参考になるメトリクス、参考にならないメトリクス

日々コードを改修していくうえで、さまざまなメトリクスが収集できます。筆者のチームではSonarQube[注5]を利用し、コミットのたびに集計していました。

カバレッジだけ見ても、行カバレッジ、ブランチカバレッジ、ステートメントカバレッジ、

注4) ドナルド・クヌースはTeXの開発者として知られている数学者です。『The Art of Computer Programming』の著者でもあります。TeXもまた不具合が少ないことで知られています。
Donald E. Knuth 著、有澤誠、和田英一 監訳、青木孝、筧一彦、鈴木健一、長尾高弘 訳『The Art of Computer Programming Volume 1 Fundamental Algorithms Third Edition 日本語版』ドワンゴ、2015年

注5) SonarQubeとは、スイスのSonarSource社が開発を行っているソースコードの品質管理システムです。さまざまな言語に対して、ソースコードの静的解析のレポートを表示したり、テストを実行してカバレッジレポートを表示したりできるツールです。過去のデータも保持しているので、時系列でメトリクスを参照できたりする優れたツールです。

ブロックカバレッジ、などいくつもあります。その他のメトリクスも、凝集度や複雑度、コメント率、重複行数、などいくつもあります。いったい我々はこれらのメトリクスの中から何を参考にすれば良いのでしょうか？

この節では、コードを高品質化する際に参考にしたものを紹介します。

まずテストを書くときに一番参考にしたのは、行カバレッジでした。まだテストが十分にそろってない時代、カバレッジを見ればテストされていないコードがひと目でわかるので、それを参考にテストを次々と書いていきました。プロダクションコードもテストしやすいように修正しつつ、カバレッジを参考にし、カバレッジを85%以上まで引き上げました。残りの部分はロジックではない小さな部分であったり絶対に通らないようなコードであったりしたので、この数字はほぼ限界近くまでプロダクションコードがテストコードによって通っていることを表していると思います。

一方、ブランチカバレッジやその他のカバレッジは、あまり参考にはしませんでした。というのも、当初はカバレッジ率が非常に低かったのでブランチカバレッジを気にするほど余裕がなかったのと、行カバレッジが上昇するに従ってブランチカバレッジなども上昇してきていたからです。

次に参考になったのはサイクロマティック複雑度でした。サイクロマティック複雑度は循環的複雑度とも呼ばれ、メトリクス測定では一般的なものです。これは、メソッドごとに始まりから終わりまでの経路のパターン数を表したもので、分岐が多いほど増加するものです。この値が大きいものほど複雑度が高いことを表しています。サイクロマティック複雑度は、おおむね人間の直感とそう離れていない程度には複雑度をうまく表しているように思います。これを参考に、数値の大きいもの順にリファクタしたりクラスを分割したりして、効率的に複雑なコードを除去できました。

それから単純ではありますが、大き過ぎるクラスやメソッドというのも効果がありました。これらは単にメソッドの行数が100行を超えているかどうかとか、1クラスに20以上のメソッドがあるかどうかといった至極単純な計算をするものです。単純とはいえ、判断基準が明瞭な分人間にもわかりやすいもので、度々参考にしていました。

メトリクスを参考にする際の注意ですが、あくまでこれらは目安でしかないということを念頭に置いたうえで参考にしましょう。カバレッジが低いからテストをもっと書くべきだとか、複雑度が高いから悪いコードだとは一概に言えるものではありません。

このメトリクス周りには後日談があります。筆者たちのチームではある程度のコードの品質を手に入れて以降、メトリクスをほとんど見なくなりました。というのも、既知のコードは直せる部分は直したのと、新規の部分はレビューによってたいてい弾けるようになったからです。

メトリクスを眺めていると、限界までカバレッジを上げてみたくなるなどの中毒症状に似たものが出ますが、おそらくこれ以上カバレッジを上げてもほとんどリターンは得られないだろうと判断しました。それはコードの品質とカバレッジの関係の飽和点のようなものなのだろうと思います。

IDEの警告を無視しないように

Javaで開発している方なら、まず間違いなく何かしらのIDEを用いて開発を行っていると思います。EclipseやIntelliJ IDEA、NetBeansなどいくつも優れたIDEがありますが、IDEの機能の1つとして、コードをチェックして怪しげな個所に警告を出してくれる機能があります。たとえばDeprecated(非推奨)なメソッドを利用していたり、型パラメータを指定すべき個所で指定していなかったりする場合に、該当する部分に下線を引いてくれるなどで開発者に知らせ

▼図4　コードにおける割れ窓理論「粗悪な部分がゼロだと、粗悪なプログラミングはしづらい」

てくれます。

これらの警告は、正直言って些細なものです。警告を放置したからといって即不具合につながるようなケースは多くありません。しかしそれでも、警告は無視しないようにしましょう。というのも、警告が出ているコードを放置しておくことでコードがレガシー化していくからです。たとえば、Deprecatedなメソッドを利用している個所を長期間放置しておくと、そのライブラリをバージョンアップしたくともインターフェースが大きく変わってバージョンアップができなくなる、あるいは大きな工数がかかるようになってしまうかもしれません。型パラメータを指定すべき個所で指定していなかったというケースも、言語やライブラリが提供している安全性を高める機能を捨てているようなもので、不吉な匂いの一因です。

個々の警告を見るとひとつひとつは些細なもので、「この警告1つくらいなら無視しても大丈夫だろう」と思うことでしょう。しかし、警告が多数あると細かな警告に注意がいき渡らなくなってしまい、結果としてコード全体がレガシー化の方向に向かい始めてしまいます。警告は修正するのが一番ですが、諸事情でどうしても直せない場合、特定の個所の警告を消す方法もあるのでそれを使うのも1つの手です(もちろんあまりお勧めはできません)。警告を0件にしておくと、次に警告が出たときにすぐ気づけるようになり、気づけることで修正しようという意思が働きます。

これはコードの中の割れ窓理論のようなものだと筆者は考えています。割れ窓理論とは犯罪学で用いられる言葉で、小さな犯罪を取り締まることで凶悪な犯罪を抑止できる、という理論です。1枚の窓が割れているとほかの窓を割ることに抵抗が少ないですが、1枚も窓が割れていない建物の窓を割るのは抵抗が大きい、という心理を表したもので、割れた窓のような些細なことも放置せずきれいにしておくことでほかの窓も割られず、結果大きな犯罪も防げるそうです(**図4**)。

この理屈は犯罪学上では眉唾(まゆつば)ものではありますが、筆者はコード上ではこの理論は正しいのではないかと考えています。全体から見れば小さな粗悪なコードがあったとして、その粗悪なコードに引きずられるようにしてほかのコードも粗悪になっていく。そういった体験はプログラマなら誰でもあるのではないでしょうか。

書籍『ビューティフルコード』[注6]でも同様のことが言われています。2章の「スプーン一杯の汚水で」というタイトルが付けられた章で、「樽一杯のワインにスプーン一杯の泥水を入れたらそれは泥水になる。樽一杯の泥水にスプーン一杯のワインを入れても泥水のままである。」ということを述べています。

コードに泥水が入らないよう、些細な警告も取り除きましょう。警告を取り除くことで、新規の警告に気づきやすくなり、警告が出ないように修正する。そうして好循環の車輪が回り始めるので、ぜひ警告は取り除いてください。SD

注6) Andy Oram、Greg Wilson 編、Brian Kernighan、Jon Bentley、まつもとゆきひろ他 著、久野禎子、久野靖 訳『ビューティフルコード』オライリー・ジャパン、2008年

第2章

効果的なテストを無理なく導入する方法

Author 青木 翔(あおき しょう)
サイボウズ㈱ アプリケーション基盤チーム
Twitter @a_o_k_i_n_g
Illustration どこ ちゃるこ

テストが品質を担保してくれる

高品質なソフトウェアを作り上げるには、テストは必須です。スーパープログラマならまだしも、ほとんどの凡人プログラマにとってはテストはなくてはならないものでしょう。

今の時代、テストの重要性はかなり理解されやすくなっていると思います。オープンソースの著名なソフトウェアも膨大なテストによって、その品質が担保されているということが当たり前になってきました。少し知名度があるOSSには、ほとんどすべてテストが付属しています。組み込み型RDBとして有名なSQLiteは、本体のコード量に比べてテストコードの量はおよそ680倍もあるということで話題になったこともありました。テストは、高品質なソフトウェアを作り上げるためには必須と言って良いでしょう。

本章では、高品質を生み出すテストコードについて記します。

テストを書く文化を取り入れよう

テストの重要性はかなり浸透してきたと言えますが、一方で、テストを書く文化がないチームがあるのもまた事実です。それは忙しくてテストを書く時間がなかったり、実装的にテストが書きづらかったりとさまざまな理由があると思います。あるいは「俺が書くコードはバグがないからテストは必要ない」というような理由からかもしれません。1人で開発しているソフトウェアならテストがなくても良いかもしれませんが、多くのケースではソフトウェアは複数人で作り上げるものです。その全員でメソッドの細かな仕様まで認識を合わせ、かつ常にそれを忘れないように実装するということは現実的には不可能です。それを助けてくれるのがテストで、テストを書くことによって仕様の破壊を防ぐことができます。

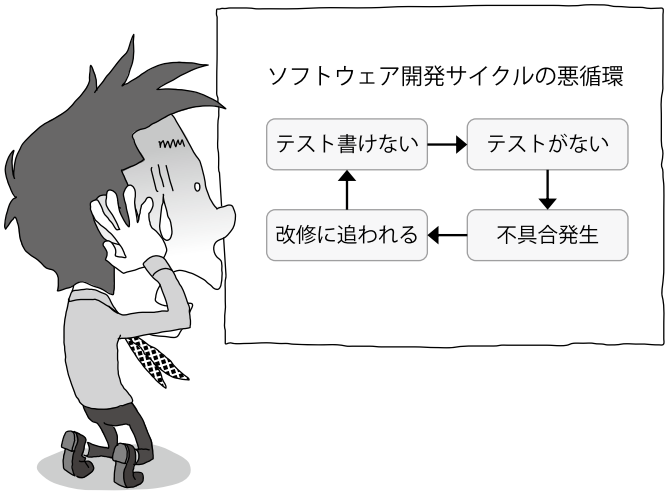
忙しいときこそテストを書こう

忙しくてテストを書くことができていない場合、それは**ソフトウェア開発サイクルが悪循環に陥っている**可能性があります(**図1**)。その忙しさの要因に、不具合改修がある程度の割合を占めているのではないでしょうか。テストがないことによって不具合を生み出してしまい、その不具合の改修が忙しくテストが書けない。悪循環ですね。また、実装的にテストが書きづらいというケースもあるのではないでしょうか。しかしそのソフトウェアのすべてがテストしにくいコードというのはほとんどあり得ず、小さな部分に着目すればテストを書きやすい部分があるはずです。

テストを書く文化がない現場では、「では、今日からテストを書きましょう」と言ってもおそらく簡単に合意は取れないでしょう。そこで筆者

がお勧めしたいのは、「不具合を改修したときに、その部分についてのみテストを書いていく」という方法です。これなら小さなスタートを切ることができるので、あまり障壁は高くありません。少しずつ少しずつ、テストケースが溜まっていくことでしょう。しばらくは手元でテストを実行するだけでも良いと思います。いずれはCI[注1]も必要になりますが、最初はとにかく小さなスタートで良く、テストコード0行から1行を目指しましょう。そうしていくうちに少しずつテストが充実し、不具合をテストで救えるケースが出てきます。その不具合をテストで救えたケースをチーム内で共有すれば、テストの有用性について理解してくれることでしょう。そうしてだんだん、テストを書いていく文化が生まれていきます。

▼図1　ソフトウェア開発サイクルが悪循環に陥っていませんか?



小さな一歩からはじめよう

筆者も失敗した体験があるのですが、いきなりテストを書く文化をチームに導入しようとしてもなかなかうまくいきません。それはテストの必要性が理解してもらえなかったり、テストを書く工数が取れないなどの理由があったり、そう簡単にこの壁を葬ることはできません。最初は小さな一歩から開始し、少しずつ少しずつ導入していくのが良いのではないかと思います。

幸い、テストコードはあくまでテストコードであり、プロダクションコードには影響を及ぼしません。プロダクトマネージャや品質保証部にお伺いを立てることなくテストを書くことはできるのです。これはテストコードのすばらしい性質の1つであり、この性質を活用しない手はありません。

テストがあれば、コードを見たり手動でテストしたりする以外に不具合を検出するしくみを得ることができます。実装時には気づかなかったコーナーケースに気づけたり、意図せず組み込んでしまった些細な仕様変更に気づけたりと、人間では対応することが困難なケースをサポートしてくれます。これは言い換えれば、テストを書くということは、プログラマが自身の無能さを克服するための手段の1つであると言えます。バグを作りこまないプログラマはまず存在しないので、テストは書くべきでしょう。

すべてのプロダクトにテストを書くべきか?

前述のようにテストはソフトウェアに福音をもたらしますが、**テストは常に書くべき、というものではありません。**テストを書くにも工数が割かれるわけですし、CIサーバやテストデータの管理コストなどもかかります。テストはすばらしいものではありますが、効果を発揮しやすいポイントとそうでないポイントがあるので、テストを書くことに割ける有限なリソースの価値を最大化することを意識して書きましょう。

テストが効果を発揮するのは、次の2点にマッ

注1）Continuous Integration：継続的インテグレーション。

チするかどうかです。

①対象となるコードが開発者だけではなくユーザが使うものであること
②対象となるコードの寿命が長いこと

①対象となるコードが開発者だけではなくユーザが使うものであること

これは、「開発者だけが使うツールにはテストコードはあまり必要とされない」ということを指しています。また、「ユーザが使うものである」というのは、ユーザが直接的に操作するもの、たとえばアプリケーションサーバやブラウザ上で動くJavaScriptのコードだけでなく、システム内でユーザデータを扱うものすべてを指してます。

具体的には、たとえばデプロイ処理用のスクリプトやコードに静的解析をかけるスクリプトなどです。これらは開発者向けであり、製品のユーザとは直接的には関係がありません。とくにデプロイ処理スクリプトなどは周辺の環境や時代に応じて日々変化していくものですので、仮にテストを書いていてもそのテストのメンテナンスに工数を取られがちになるでしょう。

②対象となるコードの寿命が長いこと

たとえば、テストが書かれていない製品があるとします。仮にその製品が今後も継続的に売れ続け、5年後や10年後も現役として稼働し続けそうならば、テストを書くことによる恩恵は大きいでしょう。テストを書き、テストコードを保守し続ける工数を鑑(かんが)みても、その工数分をペイするだけの品質向上の恩恵があると考えられるからです。一方、仮にその製品が今後クローズする方向に向かっているならテストを書く恩恵はあまり大きくありません。

プロダクトコードとともにテストコードが存在する期間が長ければ長いほど、テストコードは価値を発揮していきます。製品の寿命を想定することは困難なことではありますが、少なくとも終わりが見えているコードよりは将来性のあるコードにテストを書いたほうが価値を発揮するでしょう。

テストの実行時間も気にするべし

よりテストコードの価値を高めるために、テストの実行時間も気にしましょう。仮に十分な量のテストがあるとして、コミットのたびにCIでテストを実行し、テストが通らなければプロダクトコードにマージできないようなしくみが構築されているとします。そういった状況下で、仮にテストの実行に1時間かかるとなると、ログメッセージの修正のような細かな修正でもマージまでに少なくとも1時間以上の時間がかかってしまうことになります。それが積み重なると、人間側の待ち時間が積み重なってきます。肌感覚ではありますが、単体テストは10分以内程度に終わると快適な開発プロセスを回せるように思います。

たいていの単体テストのツールでは、テストケースごとにかかった時間が表示されます。簡単なスクリプトで、時間がかかったテストの上位一覧を出せることでしょう。もしくは、各種メトリクス測定ツールを導入しているなら、自動でテストに時間がかかったものが表示されるので、上から順に潰(つぶ)していけばかなり改善できます。

I/Oに時間がかかるケース

テストが遅くなる理由として、テストデータの出し入れなどによるI/Oに時間がかかっていることがありがちです。もしI/Oが問題なら、たとえばテスト時に使うRDBMSのデータディレクトリを、メモリ上に配置するだけで劇的に高速化するでしょう。Linuxならば/run/shm下はデフォルトでtmpfs[注2]でマウントされているので、ここを用いると良いです。

注2）tmpfsとは、Linuxで利用可能なメモリベースのファイルシステムです。これを用いるとメモリの空いた部分をストレージのように扱うことができます。メモリベースゆえに、超高速で読み書きできます。マシンを再起動すると内容が失われます。

▼リスト1　必ずスリープしてしまうメソッド

```
public void doSleepIfNeed() {
  if (isXXX()) {
    try {
      Thread.sleep(SLEEP_TIME);
    } catch (InterruptedException e) {
      (..中略..)
    }
  }
}
```

ほかにも、一時ディレクトリにファイルを読み書きするのが遅いケースがあります。たとえばJavaならjava.io.tmpdir環境変数で一時ディレクトリを変更できるので、そこでtmpfsを指定すればI/Oに関する時間を短縮できます。ただしtmpfsはあくまでメモリ上ですので、SSDやHDDに比べると領域が小さく、大き過ぎるデータを書き込まないよう注意する必要があります。

コード中でスリープしているケース

次に単体テストが遅い理由としてよくあるのは、テスト内でスリープ処理を入れてしまっているということが挙げられます。何かの条件が満たされるまでスリープする、あるいはリトライ処理で次回のリトライまでに一定時間待つなどスリープを使う場面は多々あります。このような処理をテストする際、愚直にスリープを待つと不要に待ち時間が発生してしまいます。このようなときは、スリープする処理を別のメソッドに切り出し、テスト時はそのスリープ用メソッドをモック化して実際にスリープ処理が行われないようにしましょう。

たとえば、**リスト1**のようなメソッドは問答無用でスリープしてしまうので、テストしにくいです。

▼リスト2　スリープするメソッドを切り出した例

```
public void doSleepIfNeed() {
  if (isXXX()) {
    doSleep(SLEEP_TIME)
  }
}

protected void doSleep(long sleepTime) {
  try {
    Thread.sleep(sleepTime);
  } catch (InterruptedException ignore) {
    (..中略..)
  }
}
```

SLEEP_TIMEを外部から渡せる形にするのも良いですが、ここではThread.sleep(long)を呼び出さないようにします。**リスト2**のように修正し、スリープするメソッドを切り出しました。

テストコード側では、doSleepをモック化しましょう。**リスト3**ではMockitoを利用しています(Mockitoの紹介は後述)。sutはテスト対象のインスタンスです。

Mockitoの記述方法を知らないと、**リスト3**のテストコードはわかりにくいと思います。やっていることは、doSleep内の処理をテスト時には実行しないようにしていて、ただし呼び出されたかどうかはチェックしている、というものです。これで、テスト時にスリープ処理が呼ばれなくなりました。

それ以外の遅いケース

それでも遅い場合は、テストを並列で動かせるようにするという手段もあります。筆者が所属するミドルウェアを開発するチームでは、画像変換処理を行うツールも提供しています。画像変換はテストすべきパターンが多く、枝切りしてもテストするパターン数は1,000を超えて

▼リスト3　doSleepをモック化する

```
public void testDoSleepIfNeed_sleepするケース() {
  doNothing().when(sut).doSleep(anyLong()); // モック化
  sut.doSleepIfNeed();
  verify(sut).doSleep(SLEEP_TIME); // doSleep が呼ばれたことを確認
}
```

いました。画像変換処理はただでさえ計算量を必要とするので、この1,000パターンを超えるケースを逐次実行していくとかなり時間がかかります。これを改善するため、パターンごとに並列実行できるように修正しました。

テストの並列化は強力な手段ですが、注意点もあります。それはテストコードが複雑になりやすい、ということです。また、今回のケースのように画像変換のテスト内でパターンごとに並列化するならまだしも、別個のテストケースを並列化して実行してくれるようなフレームワークを用いてテスト時間の短縮化を図る場合は、さまざまな依存関係の解決に時間を取られてテストコードのメンテナンスに工数をとられることになり得るという点にも注意が必要です。

単体テストならまだしも、Selenium[注3]テストまわりはとくに時間がかかりがちです。

弊社の例では、十分な量のSeleniumテストがあり品質を十分に保証しているチームがあるのですが、そのSeleniumテストをすべて実行するのには約7時間かかっていたというケースがありました。これだけ時間がかかるとプルリクエストを出してCIでテストを実行、ということが気軽にできません。テストが落ちることがわかるのも翌日になり、スピーディな開発からは遠のいてしまっています。

このケースは、弊社のテストエンジニアリングチームが改善を行い、テストの並列化と、Google Computing Platformの計算力を活用して時間の短縮を図りました。

良いテストとは

テストを書く文化が根づき、日々CIでテストを実行するようになったら、テストコードの品質を気にするフェーズです。というのも、テストコードが粗悪だと、プロダクションコードを修正した際にテストが落ち、そのテストの修正に時間を取られてしまうからです。複雑なロジックが書かれているテストや、事前に挿入するテストデータが大きいときなどがこれに該当します。

そのような状況を防ぐため、弊社ではテストコードもレビューを行っています。良いテストを書くうえで一番注目するところは「十分なテストケースが記述されているかどうか」です。

不具合は境界で起こりやすい

テストケースの中でもとくに注目するのは、境界値やその前後のケースです。不具合は境界で起こりやすいので、そこを重点的にチェックしています。たとえば文字列をある長さnに切り取るテストでは、

- 長さnを超える文字列
- 長さnと同じ長さの文字列
- 長さn未満の文字列

というテストケースを必ず用意してもらいます。そのうえで、

- 文字列が空文字列のケース
- 文字列がnullのケース
- Javaの場合、切り取る境界上にサロゲートペア[注4]文字が来たときのケース

も必須です。レビューでは、このようにさまざまな条件を十分に試したテストコードであるかを確認しています。

テストコードはシンプルに

テストコードをプロダクションコードと同等のつもりで書くと、テストコードに不要なロジッ

注3) Seleniumとは、Webブラウザ経由でWebアプリケーションのテストを行うツールの名称です。手動でWebアプリケーションを操作するような試験を自動化できます。便利な一方で、Seleniumには単体テストのような軽量さがなく、一般的にテストの実行には時間がかかります。

注4) Surrogate Pair：もともとUnicodeは16bitで1つの文字を表す仕様ですが、表現できる文字数を増やすためにUnicode 2.0から16bitの文字コード2つで1つの文字を表す仕様が追加されました。この1対の16bit文字コードをサロゲートペアと呼びます。Javaでは、このような文字は、文字を表す型であるcharでは表現できません。そのためchar 2つで1文字を表します。Javaにおいては、この2つのcharをサロゲートペアと呼びます。

▼リスト4　イマイチな例

```
Map<String, String> targetFiles = new HashMap<>();
targetFiles.put("foo.txt", "Shift-JIS");
targetFiles.put("bar.txt", "UTF-8");
targetFiles.put("baz.txt", "ASCII");

for (String file : targetFiles.keySet()) {
  testTextFileCharset(file, targetFiles.get(file));
}
```

▼リスト5　良い例

```
testTextFileCharset("foo.txt", "Shift-JIS");
testTextFileCharset("bar.txt", "UTF-8");
testTextFileCharset("baz.txt", "ASCII");
```

クが入ってしまうことがあります。たとえばテキストファイルの文字コードを判定するテストがあるとします。このとき、**リスト4**のように書きたくなるかもしれません。

しかしテストコードでは**リスト4**のようなロジックはあまり必要ではないケースが多く、**リスト5**のようにシンプルにベタ書きしてしまうのが一番わかりやすい、ということもしばしばあります。

同様に、たとえばJavaでは文字列結合が遅いときにStringBuilderを使いますが、テストコードでは文字列結合がボトルネックになることはまずないうえに、コードが読みにくくなるので害悪ですらあります。コンパイラによる最適化もかなり効くので、ほとんどのケースではStringBuilderを使って文字列結合をする必要はありません。

テストメソッドの名前

テストメソッドの名前も、できるだけ明瞭にしましょう。Javaではメソッド名に日本語を使えるので、明瞭さが十分確保できるなら、日本語のメソッド名も許容しています(**図2**)。

たとえば弊社では次のようなテストメソッド名を使用しています。これはHTTP経由で受け取ったデータをパースして別のデータに変換する処理です。

- testConvert_ステータスコードが200かつJSONが正しい()
- testConvert_ステータスコードが200かつJSONが不正()
- testConvert_ステータスコードが200かつJSONがパース不可()

このあとも、ステータスコードが404や500などいくつものケースのテストがあり、かつそれぞれに応じてJSONが正しいのか、期待してないフォーマットなのか、パース不可なのかのケースを書いています。一目瞭然ですね。**日本語は情報の密度が高い**ので、日本人で構成されるチームならテストメソッド名に日本語を使う

▼図2　日本語のメソッド名のほうが簡潔に表現できることも

▼リスト6　ユーザ管理をするロジックをモック化する例

```
UserLogic logic = mock(UserLogic.class); // モック作成
when(logic.getUserCount()).thenReturn(27); // getUserCount() が27を返すよう設定
sut.setLogic(logic); // テスト対象のインスタンスにロジックを設定
sut.doSomething(); // テスト対象のメソッドを実行（内部で logic.getUserCount() が呼び出される想定）
verify(logic).getUserCount(); // getUserCount() が呼び出されたかチェック
```

のも良い選択肢かと思います。

ただ、テストコードのレビューはプロダクションコードほど厳密に行っているわけではありません。たとえばリファクタリングなどでファイルの変更数が膨大になると、「テストコードは別途修正するのでプロダクションコードだけレビューお願いします」と言ってプルリクエストをある程度の大きさに絞るようにしています。

優秀なテストライブラリを使おう

Javaの単体テストを行うためのフレームワークといえば、言わずと知れたJUnitですね。しかし近年のテスト事情ではJUnit単体では機能が足りないことがあり、JUnitを補強するテストライブラリがいくつか出ています。

ここでは、弊社が利用しているすばらしいテストライブラリ、MockitoとAssertJを簡単に紹介します。

Mockito

Mockito[注5]はモック化を簡単に行うためのライブラリです。モック化とは、テスト時に特定のインスタンス変数やメソッドの処理を差し替えられるようにすることです。モック化することで得られる利点は、単体テスト時の依存関係を限定的にできるということです。あるクラスAがBに依存し、BはCに依存するコードで、Aの単体テストをしたいとします。そのときに、BやCのインスタンスを生成したり変数を書き換えたりして、Aをテスト可能な状態に持っていく必要があります。現実的にはこれらはけっこう手間がかかり、テストコードも複雑化します。そこで、Bをモック化し、任意の振る舞いを行えるようなインスタンスを設定することでAはテストしやすくなります。

リスト6は、ユーザ管理をするロジックをモック化する例です。ここではこのモックに、`getUserCount()`メソッドを呼び出したら27を返すという振る舞いを設定しています。

これで`logic`は、`getUserCount()`が呼び出されると27を返すようになりました。これでユーザ数は一見27人に見えるようになりましたが、あくまでモックです。たとえば`getUsers()`のようなメソッドがあるとして、それを呼ぶとnullが返ります。このモックだけではほとんど利点を感じることはできないかもしれませんが、このロジックに大きく左右されるようなクラスがあるとき、このMockitoをうまく使えば事前条件をスムーズに指定できたり、事後条件をアサーション（条件が正しいかどうかの判定処理）できたりします。

任意のクラスをモック化できますし、完全なるモックでなくとも通常のインスタンスの一部だけ挙動を差し替える、ということもできます。引数によって戻り値を変更したり、例外を投げたり、メソッドが呼び出されたときの引数を後から取得したりすることだってできます。少々クセのある記法を用いるので最初は慣れないかもしれませんが、使いこなせれば手放せなくなるたいへん優れたライブラリです。

AssertJ

AssertJ[注6]は、流れるようにアサーションを

注5）URL http://mockito.org/

注6）URL http://joel-costigliola.github.io/assertj/

▼リスト7　文字列のアサーションの例

```
assertThat(user.getName()).startsWith("Ao")
  .endsWith("ng")
  .isEqualToIgnoringCase("aoking");
```

▼リスト8　ユーザリストのチェックのテスト

```
List<User> users = ...
assertEquals(3, users.size())
assertEquals("tanaka", users.get(0).getName());
assertEquals("aoki", users.get(1).getName());
assertEquals("mori", users.get(2).getName());
```

▼リスト9　AssertJを用いてリスト8を改良

```
assertThat(users).extracting("name").containsExactly("tanaka", "aoki", "mori");
```

▼リスト10　各ユーザ年齢の比較

```
assertThat(users).extracting("age").containsExactly(29, 25, 32);
```

▼リスト11　extractingとtupleを用いてリスト9とリスト10を改良

```
assertThat(users).extracting("name", "age")
  .containsExactly(
    tuple("tanaka", 29),
    tuple("aoki", 25),
    tuple("mori", 32));
```

行えるライブラリです。心地良くアサーションを書け、かつ読みやすさにも優れています。

たとえば文字列のアサーションでは、**リスト7**のように流れるように条件を連続して書くことができます。この例はほとんどおもちゃのようなコードですが、流れるように(Fluentに)記述できていることがわかるかと思います。

AssertJは、とくにコレクション関係ですばらしい力を発揮します。ユーザのリストを取得するメソッドで、結果をチェックするとき、**リスト8**のようになると思います。

このようなテストはAssertJを用いると**リスト9**のように書けます。extractingというメソッドでリスト内のオブジェクトのプロパティを抜き出して、その後に続くメソッドで一気に比較できます。

また、さらに各ユーザの年齢も比較したくなったとします。同様に**リスト10**の1行を追加しても良いですが、extractingとtupleを用いて**リスト11**のように記述することもできます。

1行が長くなり過ぎると可読性が落ちますが、AssertJにはいくつもの便利なメソッドが用意されており、ケースバイケースでテストに見合った内容のものを利用できます。また、入力補完が効くのも大きなメリットです。「assertThat(list).」と打った時点でリスト系の比較メソッドが候補に出てくるのでサクサクとテストを書いていくことができます。このAssertJを用いれば簡潔明瞭なテストコードを書く手助けになるでしょう。SD

第3章

漏れがなく負担も少ない コードレビューとは

Author 青木 翔(あおき しょう)
サイボウズ㈱ アプリケーション基盤チーム
Twitter @a_o_k_i_n_g

人間はミスをするもの

高品質なソフトウェアを作るという点において、コードレビューは非常に重要です。これを怠るのは、高品質なソフトウェアを作るということを諦めるに等しい行為とすら言えます。

人間の能力には限界があります。慣れ親しんだコードにわずかな修正を入れるとき、簡単な機能追加を行うときでさえ、うっかり凡ミスを入れてしまったり不自然な設計にしてしまったりすることは日常茶飯事です。体調やそのときの心境、直前に読んだコードや書籍などの影響を受けることもあります。深夜残業やリリース直前で時間がないときも低品質なコードを書きがちです。プラスの方向に影響を受けるなら良いですが、そうでないこともしばしばあります。そういった既存のコードを汚染するようなコードを弾(はじ)くことがレビューの大きな役割です。

レビューはコードに着目すべきです。レビュイーが上司だからというような理由で粗悪なコードであることを指摘しにくい、というような環境では高品質なコードは生まれにくいでしょう。役職や権力にかかわらず、レビューの際はコード本位で考え、議論しあえるチームの土壌が必要です。そういう土壌作りはおそらく一朝一夕では不可能です。もし、上司の権力やコードが絶対であり、指摘するのが困難なチームなら、真剣に転職活動を検討すべきと思います。

通常、実装とレビューは1対1のペアで行いますが、重要な部分についてはレビュワーを複数人に指定することもあります。たとえば、筆者たちが実装する機能の1つに、ドメインプロテクションと呼ばれる顧客同士のデータを絶対に混じらないようにするための論理的な壁を構築するしくみがあります。あるとき、諸般の事情でこのドメインプロテクションのしくみを再実装することにしました。ここは極めて重要な個所で、些細な不具合でも混じれば会社への信頼が揺らぎ得る部分です。コードの行数的にはそれほど多くありませんでしたが、複数人がレビューを行いました。結果、その部分は不具合を起こさず、現在もうまく稼働しています。

サイボウズのレビュー文化

サイボウズではCI(継続的インテグレーション)はJenkins、VCS(バージョン管理システム)はGit、レビューツールにはGitHub Enterpriseを利用、というチームが多いです。

通常、プロダクションコードにはレビューなしでマージやコミットされることはなく、常にレビューを必要とします。

ただし一部例外があり、たとえばプロダクションコードの範囲内でも、コメントの追加のような些細なものならレビューなしでマージすることもあります。具体的には、障害対応や不具合調査の際に既存のコードを眺めることになりますが、そのとき

▼リスト1　サイボウズのレビューポイントのWiki（一部抜粋）

```
## 忘れ物はないですか?
* File、接続の閉じ忘れはないか？
* 永続的に使うスレッドに紐づく物をためっぱなしにしてないか？
** 必要だからといってコネクションを接続しっぱなしにする場合、ポートが枯渇するような状況には陥らないか？
* 例外発生時もちゃんと close されるか？
** try-with-resources 使いましょう

## 国際化対応
* ロケール（言語）とリージョン（国）を混同しないようにしましょう。
* 日時を扱う際はタイムゾーンは基本 UTC で

## 不要なファイル
誰からも参照されない不要なファイルが残っていないか？
diff に出ないので注意です。

## ツールで実環境で動作させましょう!
* 結合テスト的なイメージ
* 実環境で動かすのが手間な場合、少なくともローカルでは動かして動作確認すること。
```

に一見して何をしているのかわからないコードに遭遇することがあります。コードを修正できるならそれが一番ですが、そうでないことも多いので、そのときはコメントを追加して意図が簡潔に伝わるようにしています。とはいえレビューなしでのマージは基本的には稀（まれ）で、ほとんどありません。ログの1行を追加するだけでもレビューは必須です。

レビューにはそれなりの技術を要します。コードの正しさや拡張性といった観点のほかに、ほかの製品と組み合わせたときに問題が起きないかといった**大局的観点**などいくつもの必要な技術があります。

そのため、筆者のチームでは多数のレビューを経て得た知見を社内Wikiに蓄積しています。“Middleware Review Points”というタイトルのそのWikiページには、「ファイルのclose漏れがないかチェック」「ゴミファイルを残さないように」といった基本的なレビューポイントはもちろん、その機能独自のレビューポイント、たとえば「○○の処理が実行される順序に注意」といったポイントも記されています。一時期、筆者は実装者としてレビューに出すときも、レビュワーとしてコードを見るときも、このWikiページを常にブラウザで開いていて各項目を確認するようにしていました。このレビューポイントのWikiの一部を紹介します（**リスト1**）。

レビューポイントのひとつひとつは単純なもので、ほとんど知っていて当然というものです。しかし一方で、レビューポイントは多岐に渡り、そのすべてを常に把握しておくことは困難です。人間は物事を忘れやすいので、要点だけを記したレビューポイントを見ながらの実装とレビューは効果がありました。

レビュー時には小さな一言を

もう1つ、レビュー時にやっていることがあります。それは、実装者がレビューを投げる際に、レビューポイントをコメントで書き記しておき、レビュワーの負担を下げるようにすることです。レビューに投げる際、たとえばif文の構造を少し変えただけでもインデントが変わって一見diffが大きく見えてしまったり、本質的ではない変更が複数のファイルにまたがってしまっていたりすることはしばしばあります。

そのようなとき、事前に「ここはif文を変更したのでインデントが変わっています」「ここが肝な部分です」というような一言があるだけで、ぐっとレビューをしやすくなります。レビュワーの負担を減らし、かつ滑らかなコミュニケーションのために、そういった小さな一言を添えておくと良いでしょう。SD

第4章 ログ監視で人が気づかないバグも発見・撲滅する

Author 青木 翔(あおき しょう)
サイボウズ㈱ アプリケーション基盤チーム
Twitter @a_o_k_i_n_g
Illustration どこ ちゃるこ

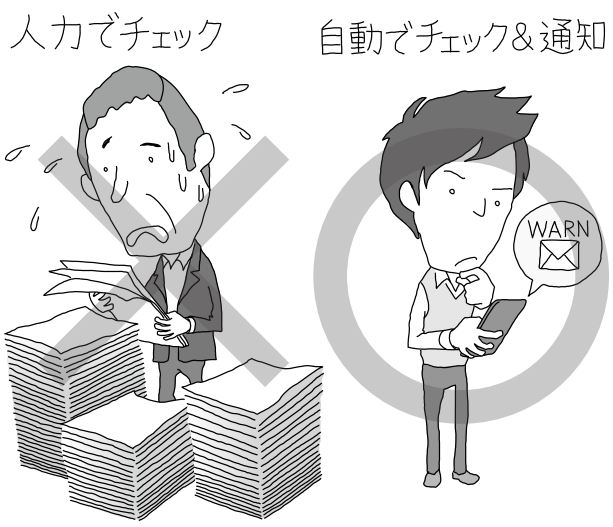
ログも品質向上に役立つ

ソフトウェアを作るうえで、良いログを出し、それを監視するということは想像を絶するほど重要です。少なくとも良いコードを書くことと同程度には重要で、これを怠ってはなりません。当たり前ではありますが、**ログは宝**です。より良いソフトウェアを作り上げるためには、この宝の価値を最大限まで有効活用しましょう。

平常時でもログを監視しよう

ソフトウェアは日々多くのログを出力します。

▼図1 平常時からログを監視できるよう自動化しよう



▼図2 ログファイルの絞り込み

```
$ cat $LOG_FILES
    | grep ERROR
    | grep -v "known error message 1"
    | grep -v "known error message 2"
    (..略..)
```

そのほとんど、おそらく99%以上は誰の目に触れることもなくひっそりと消えていくものと思います。ログを見る必要があるときは、たいていトラブルの現象解明のために読まれることが多いのではないでしょうか。しかしそれはとてももったいないことであり、トラブルが起こる前にこそログを確認することでその価値を発揮できるものだと思います。

膨大なログをすべて日々人間の目で追うことは非現実的ですので、これをある程度自動化しましょう(**図1**)。ログを半自動で監視し、未知の例外やログにすぐ気づけるようなしくみを構築しましょう。このような処理は簡単に書くことができます。

図2の例では、ログファイルをcatしてログレベルERRORでgrepし、既知のログをgrep -vで除外しています。すると、最後には未知のERRORログが残ります。このフィルタを通した結果は元のログの量に比べればごくわずかですので、人間の目でも十分確認できる程度のサイズになります。未知の例

外が多過ぎたら、改修しましょう。同様に、ログレベルWARNや、Exceptionでgrepすると良いです。

これは非常に単純な方法ではありますが、絶大な効果を持っています。たとえば、めったに利用されないAPIに対して特定のデータを投げると、NullPointerExceptionが発生するような不具合があったとします。これを人間の目で発見するのは困難です。弊社の例では、数十台のマシンで日々数万回以上繰り返し行われる処理の中に、4年間以上潜んでいたマルチスレッドの不具合も検出できました。針の穴を通すようなタイミングでのみ発生する不具合で、発生頻度は非常にまれなものでした。こういった小さな不具合も潰すことで、ソフトウェアはより安全堅牢になります。

このログフィルタは実装が簡単な割に、膨大なリターンが得られます。「ログをスクリプトで処理して人間の目で見る」というのは、最近のモダンな監視に比べればアナログな部類に入りますが、何より手軽に導入できるのでたいへんお勧めです。ぜひみなさんのプロジェクトでも導入してみてください。

除外して良いログとそうでないログ

さて、どのようなログを除外すれば良いのでしょうか。ある程度の指針ですが、通常操作で起こりえるもの、ユーザ操作によって引き起こされたものは除外して大丈夫です。たとえば認証に失敗したり、ユーザがAPIに対してJSONを投げてパースエラーが出たり、CSRF[注1]対策チケットの期限が切れていたり、などです。これらはソフトウェアの不具合ではなく、いわば正常な例外です。もちろんケースバイケースであり、ソフトウェアの性質によって変わってきます。

一方、既知であっても除外すべきでないログがあります。そのうちの1つは、頻度が重要な意味を持つログです。ログは「ログのメッセージ」が重要なことは言うまでもありませんが、「似たようなログが書かれた頻度」も同じくらい重要なケースがあります。たとえばファイルサーバに問い合わせしてダウンロードする処理で、1秒以上時間がかかった場合に、遅延したことを表すログが出力されるとします。このとき、ログフィルタでこの遅延ログを除外してしまうと、遅延が増加したときに気づくことができません。遅延ログは頻度が重要な意味を持つので、既知であっても除外するのはお勧めできません。

ただ、それでも量が多くて無視したいときがあります。そのときはログフィルタ部分では遅延ログを除外し、別の個所で、遅延ログが何件出ていたかの統計情報を出すと良いでしょう。もちろん、モニタリングツールなどの環境が整ってきたら、それらに監視させるのがよりベターです。

上記ログフィルタを実装しても、使われなかったら意味がありません。筆者のチームでは上記スクリプトをCIで毎朝実行し、メールを送信するようになっています。そうして毎朝の日課としてログを眺めています。

監視しやすいログ

前述したようにログの活躍できる場はトラブル発生時だけでなく、正常に稼働しているように見える際にも何か問題が発生していないか監視するときにも活躍します。そのことを念頭に置いてログを書くとさらに良いです。

たとえばエラーが発生した際に書くログは、ログのサマリを書いておいて監視スクリプトで一目でわかるようにしておくと便利です。というのも、ログのパースは実はけっこう困難ですので、1行単位で扱うことが多く、複数行を持って意味のあるログにすると扱いにくいケースがあるからです。

たとえば例外が発生した際、**リスト1**のよう

注1）Cross Site Request Forgeries：Webアプリケーションに存在する脆弱性の一種。リクエスト強要とも呼ばれる。

にログ出力をさせることはよくあると思います。

すると、フォーマットにもよりますが一般的には**図3**ようなログが出力されます。

このログを監視したいと考えた時、1行目のERRORの部分は例外の内容にかかわらず常に同一で、ERRORで単純にgrepしただけではほとんど意味がわからないでしょう。

これを改善するために、ほんの少しだけログの部分を書き換えてみます(**リスト2**)。

こうすることで、ERRORログにはちゃんとログのメッセージも表示され、ログをERRORでgrepした際に、どれが既知でどれが未知なのか判断しやすくなります。同様に、既知のERRORメッセージならgrep -vで除外することも簡単になります。ほんの小さな改善でぐっとログを監視しやすくできました。

▼リスト1　例外発生時にログを出力する

```
try {
    (..中略..)
} catch (Exception e) {
    log.error("An exception occured", e)
}
```

▼図3　リスト1によって出力されたログの例

```
2016-03-21 19:03:43,613 ERROR ClassName An exception occured
java.lang.Exception: org.apache.XxxException: <例外のメッセージ>
    … スタックトレース …
```

▼リスト2　リスト1を改良したもの

```
log.error("An exception occured. message: " + e.getMessage(), e)
```

▼リスト3　例外が握りつぶされている例

```
・Case1
try {
    (..中略..)
} catch (Exception e) {
    return; // 例外eを握りつぶしている
}

・Case2
try {
    (..中略..)
} catch (Exception e) {
    throw new RuntimeException(); // 例外eをCauseとして渡していない
}
```

意味のないログ出力指示を書かないようにする

意味のないログ出力指示を、ついうっかり書いてしまうこともしばしばあります。ログを出力しただけで満足してしまって、「いざ運用して障害が発生したとき、ログを見返すと必要な情報が書かれていなかった」ということもあります。ログ出力の指示をする際には、あとから見返したときに意味がある情報を出力するように心がけましょう。

意味のないログの例としては、次のようなものが挙げられます。

- ファイルがないときに投げる例外にファイルのパスを書いていない
- 失敗した情報にお客様情報が載っておらず、顧客を特定できない
- パースエラーした際にエラーが出た場所を書いていない
- HTTPレスポンスが200 OKではなかったが、ステータスコードが判別できない

などなど、無意味なログ書き出しは意外と作りこんでしまうものです。

最悪なのは、本来はログに出すべき例外を握りつぶしてしまうことです。握りつぶす場合は本当にそれで良いのか十分に考慮してください。たとえ握りつぶさなくとも、例外チェインに追加せず新規に例外を生成してスローするようなコードもほぼ同罪です。

具体的には、**リスト3**のような例です。これはやってはいけません！SD

第5章 高品質を目指すときに、心がけたいこと

Author 青木 翔(あおき しょう)
サイボウズ㈱ アプリケーション基盤チーム
Twitter @a_o_k_i_n_g
Illustration どこ ちゃるこ

常に高品質を目指すべきというわけではない

ここまでソフトウェアの高品質化について記してきましたが、筆者は常に高品質を目指すべきではないと思います。ソフトウェアの性質や時期によって変わってくるものです。

ソフトウェアの性質によるもの

まず、筆者のチームが既知の不具合を0件にできたのは、ミドルウェア開発チームという性質も大きいです。ミドルウェアはその性質上、新機能よりは安定性を求められやすいポジションにあります。ファイルサーバを開発するとして、ファイルのバージョン管理機能やウイルスチェック機能があったらうれしいかもしれませんが、それよりも第一にたとえマシンクラッシュ時でもファイルが消えることなく信頼して読み書きできるほうが何倍もうれしいですよね。

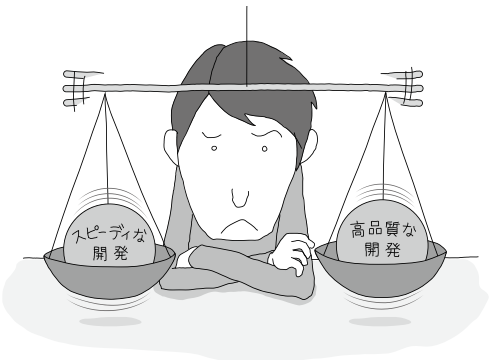
一方、フロントエンド界隈はユーザが直接触る場所ゆえ、日々絶えることなく新機能の要望がやってきます。あまり使われていない画面で特殊な操作をしたときに発生する小さな不具合を直すよりは、新機能に注力するという判断が採られることはしばしばあるのではないでしょうか。日々顧客からの新機能や改善の要望が山のようにくる中では、その要望を取り入れる方向に向かうのは、それもまた1つの正しい選択肢でしょう。

筆者たちはミドルウェアを開発するチームです。新機能の要望ももちろんありますが、フロントエンドに比べればそれほど多くありません。求められるのは安定性や性能でした。そういった性質があったからこそ品質の向上に集中できたという面は確実にあります。

時期的なもの

それから時期的な問題もあります(**図1**)。ソフトウェアのライフサイクル内の時期によっても品質は変わりますし、意図して変えることもあります。たとえば、筆者のチームが作り出したミドルウェア群は確かにできた当初は不具合の塊でした。でも筆者は、それはそれで1つの正しい選択肢かなと思います。というのも、サービスインする際は方針転換することもしばしば、

▼図1 サービスイン前は、スピードと品質の見極めが大切



仕様も深く検討する時間なしに急造することもあります。そのような場面でサービスインを目標として開発をする際には、どうしてもスピーディな開発が求められ、テストを書いたりじっくりレビューしたりというのは、それに反する面が少なからずあるからです。

もちろんそのあとの不具合改修には、それなりに時間や工数がかかります。相応の代償を払わなくてはなりません。コードの悪さならある程度改善しやすいですが、**設計の悪さの改善にはかなり長い時間が必要**です。そのあたりはサービスインする際の設計や実装にどれだけコストをかけるかというバランス感覚が必要です。

品質をコントロールするということ

最初からバグを生まないコードを書けたら最高ですが、現実的にそれは理想論であり、不可能と言って良いでしょう。DJB[注1]レベルのプログラマになればできるかもしれませんが、きっとそんなコードを書けるのは世界に数人で、大半の凡人プログラマは注意してコードを書いたところでバグを作り出してしまうものです。

となると次の一手は「品質をコントロールする」ことに落ち着きます。サービス開始時に品質が低いことはしかたがないと割りきって、その後の対応で品質を高めていく方向に持っていくべきではないでしょうか。新機能を追加することだけに時間を費やすのではなく、既存のコードを改善していくこと、不具合を検出して直していくこと、そういったことに時間を割けるかどうかが品質をコントロールできるかどうかの分岐点でしょう。

後ろ向きな実装をしない

ある実装を行う際、それが後ろ向きな実装であるケースがあります。ここで言う後ろ向きな実装とは、過去の実装や工数の問題、プロダクト間の事情などにより、一時しのぎ的なとりあえずの問題を回避する実装のことを指します。本来はより良い改修方法があるにもかかわらず、さまざまな事情でその良い改修を行えずにしぶしぶ醜い実装をしてしまうことは多いのではないでしょうか。

後ろ向きな実装の1つの例を紹介します。弊社ではファイルサーバを自前で実装しており、各アプリケーションはこのファイルサーバにユーザがアップロードした画像やそのほかのファイルを置いています。このファイルサーバと関連がある1つの機能に、ファイル内のテキストを抽出する機能があります。たとえばPowerPointのファイルをファイルサーバにアップロードし、このテキスト抽出機能を用いるとPowerPointファイル内の文字列が返されるというものです。

あるとき、このファイルサーバを利用していない弊社のアプリケーションが、テキスト抽出機能を利用したいということになりました。現状の構成上、真っ当に進めるならまずファイル管理をこのファイルサーバに移管することが先決であり、大局的に見ればそのほうがいくつかのメリットがあったのですが、機能の移管はコストが高過ぎるということで却下されてしまいました。結果、我々はこのアプリケーションのために、ファイルサーバを利用していなくともテキスト抽出機能を利用できるよう醜いif文やその他いくつかの実装を入れ込まねばなりませんでした。

このような後ろ向きな実装は、予想に反して長く生き残ってしまうことがあります。また、一時しのぎ的な実装というだけあって、本来実装すべき直感的な処理とは異なっており、保守性もあまり良いとは言えないものが多いです。

一方で、後ろ向きな実装は低コストで一時しのぎができるというメリットもあり、合理的な選択と見ることもできます。とはいえ、長いスパンで見るとその合理性は落ちていき、いずれ

注1) DJBは、ダニエル・ジュリアス・バーンスタイン(Daniel Julius Bernstein)というイリノイ大学教授の通称で、qmailやdjbdnsを公開しています。DJB製のプログラムはバグやセキュリティホールが非常に少ないことで知られています。

は害悪なコードになっていきます。

モチベーション的にも、過去の実装に縛られた一時しのぎを繰り返すより、幸福な未来を目指してコードを書くほうが何倍もモチベーションが高まります。**未来に向けて書いたコードは生きたコード**であり、若々しく、モダンなライブラリや技術を取り入れやすいという面もあります。そういうコードを書けるプログラマは技術を吸収し、発揮できます。後ろ向きな実装ばかりを繰り返すプログラマは古い技術に取り込まれ、モチベーションも高まらず、いずれ転職してしまうことでしょう。

後ろ向きな実装を行うのはしかたがない面も大きいです。でも、それを回避する選択肢があるケースもあります。後ろ向きな実装を行わなくて済むように、みんなが幸福になれるように、

参考図書(1)

ここで、筆者がソフトウェアの高品質化の観点で役に立った書籍を紹介します。いずれもすばらしい名著なので、未読ならぜひ入手して読んでみてください。チーム内のメンバーや関係者を集めて輪講するとさらなる効果を発揮できると思います。

『新装版　リファクタリング』[注2]

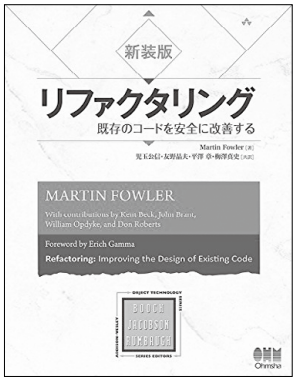
アジャイル開発やエクストリームプログラミングでも有名なマーティン・ファウラー先生の著書で、いかにして安全にコードを修正していくか、ということに焦点を当てた書籍です(**図2**)。役割ごとにクラスを作り、責任をそれぞれのクラスに分担させていく様は、リファクタリングだけでなくオブジェクト指向を学ぶのにもうってつけでしょう。

『リーダブルコード』[注3]

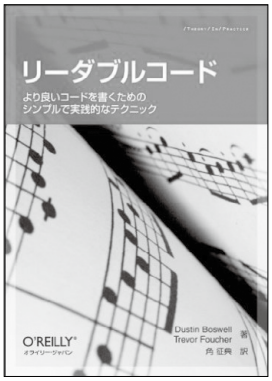
良いコードとは何か、読みやすいコードとは何か、を簡潔に明瞭に示した良書です(**図3**)。良いコードとは何か、という点については『Code Complete』[注4]や『Clean Code』[注5]などいくつか書籍がありますが、その中でも本書は内容や読みやすさの点で群を抜いており、エッセンスも凝縮されているように思います。

注2) Martin Fowler 著、児玉公信、友野晶夫、平澤章、梅澤真史 訳『新装版　リファクタリング　既存のコードを安全に改善する』オーム社、2014年
注3) Dustin Boswell、Trevor Foucher 著、角征典 訳『リーダブルコード――より良いコードを書くためのシンプルで実践的なテクニック』オライリー・ジャパン、2012年
注4) Steve McConnell 著、㈱クイープ 訳『Code Complete 第2版 上下 ― 完全なプログラミングを目指して』日経BP社、2005年
注5) Robert C. Martin 著、花井志生 訳『Clean Code アジャイルソフトウェア達人の技』アスキー・メディアワークス、2009年

▼図2 『新装版　リファクタリング　既存のコードを安全に改善する』



▼図3 『リーダブルコード――より良いコードを書くためのシンプルで実践的なテクニック』



そんなコードを書ける環境や方針を作ることもプログラマの1つの責任だと筆者は思います。

高品質化を手に入れた結果

前述のことをすべて取り入れ、筆者たちのチームのプロダクトは高品質を手に入れたと言える状況になったと思います。複雑過ぎて手出しできない聖域もなくなり、自信を持って後輩にコードの手入れを任せられるようになりました。これは、チームがスケールするようになったとも言えます。聖域があるとその歴史を知っている人だけが手を入れ続け、ほかの人がメンテナンスできないという状況がしばしば発生します。そのようなチームはスケールすることなく、属人的なノウハウの塊になってしまいます。

今では、当初頻繁に発生していた、我々のチームのプロダクトによるサービス停止はほとんどなくなりました。年単位の時間がかかりましたが、それでも得るべきものは得られたと思います。

ユーザに幸福をもたらすこともできました。テストばかり書いて、リファクタばかりしていたように見られていたかもしれない筆者のチー

参考図書(2)

『レガシーコード改善ガイド』[注6]

手のつけようがない酷いコード、レガシーコードには多くの人が悩まされていると思います。本書はそのようなレガシーコードに対してどう対処していくかを記した1冊です(**図4**)。閉塞感漂うプロジェクトに光明をもたらしてくれる1冊になると思います。

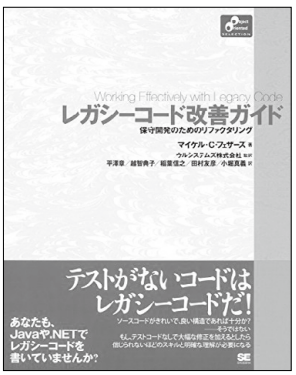
『プログラミング作法』[注7]

ブライアン・カーニハン(UnixやC言語の開発をした方)とロブ・パイク(UnixやUTF-8、Go言語の開発をした方)によって書かれたプログラミングのお作法に関する書籍です(**図5**)。先に挙げた3冊の書籍と比べて本書は古典的であり、サンプルコードや時代背景などに若干の古臭さがあることは否めません。とはいえ、「長すぎる変数名は良くない」や「コメントと実装が乖離しないように」といったお作法そのものについては時代や言語が移り変わってもほとんど変わっていません。個人的にではありますが、本書は筆者が今まで読んできたすべての技術書の中で最も愛する1冊であり、バイブルです。

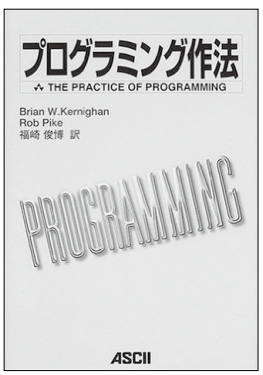
注6) マイケル・C・フェザーズ 著、ウルシステムズ株式会社 監訳、平澤章、越智典子、稲葉信之、田村友彦、小堀真義 訳『レガシーコード改善ガイド』翔泳社、2009年

注7) Brian W.Kernighan、Rob Pike 著、福崎俊博 訳『プログラミング作法』アスキー、2000年

▼図4 『レガシーコード改善ガイド』



▼図5 『プログラミング作法』



ムですが、結果として不具合が少なく安定性が高いサービスを提供できており、ユーザの幸福に一役買っているのは間違いないと断言できます。筆者の感覚では、世間一般は、「ユーザに幸福をもたらすのは新機能である」という風潮が強過ぎるのではと思います。ユーザに価値を提供するのは新機能だけでなく、安定性や性能によっても価値をもたらすことができるはずです。

不具合が少なくなってきたので、時間にある程度余裕ができます。すると、その時間を新しいツールの調査やフレームワークの選定などの先行調査などに配分できるようになりました。この調査に工数を取れるようになると、日々現れる新たなツールやフレームワークを取り入れ、時には捨て、というサイクルにつながり、好循環を生んでいます。

そして最後にもうひとつ、一番大きなメリットを紹介します。それは、高品質なソフトウェアをリリースしているという満足感からくる心地良さです。ソフトウェア開発では、たくさんの不具合を含んでいるのを知りつつも、工数やその他さまざまな関係で不具合を直せないことがしばしばあります。そういったときは「バグがあることをわかっているのに直せない」、「ユーザに対して迷惑をかけてしまっている」という後ろめたさが心のどこかにあると思います。高品質なコードは、そのような後ろめたさから解放してくれます。プロダクトがほかの製品やユーザに迷惑をかけず、日々安定して稼働するという事実は何にも代えがたい誇り高さがあります。

我々がやってきたコードの高品質化への取り組みは、ただ単にプログラマにとって読み書きしやすいということだけに終わるものではなく、高品質を良しとする文化をチームに根付かせることができたように思います。この手塩にかけたプロダクトもいずれは筆者の手を離れていきますが、でもきっと、高品質への想いは今後も受け継がれていくことでしょう。SD

特別企画

［次世代言語］

Elixirの実力を知る 前編

Phoenixで高機能Webアプリ開発

Author 大原 常徳（おおはら つねのり）
株式会社ドリコム
Twitter @ohrdev
Mail ohr486@gmail.com

［関数型言語］Elixirのはじめ方

Rubyのような書き味で、簡単に並列処理が実装できる関数型プログラミング言語「Elixir」。とくに若手エンジニアの間で注目され、企業においても少しずつ採用がはじまっています。本記事では前後編で、Elixirの実力を確かめます。前編で扱うのはElixirの概要、環境準備、簡単なコーディングです。

はじめに

みなさんはElixir[注1]をご存じでしょうか？

ElixirはJosé Valim氏によって開発されている、並列処理を取り扱うのが得意な関数型プログラミング言語です。後述するErlangVMの上で動作するので、分散システム、耐障害性、ソフトリアルタイム[注2]といった特徴を持ちます。またRubyに似たシンタックスと、マクロ、プロトコル、メタプログラミングといったモダンな機能も持ち合わせています。

2016年8月には『プログラミングElixir』[注3]が翻訳出版され、日本での認知度も上がってきました。著者は、Elixirに関するイベントやMeetUpを運営しているのですが、とても強い盛り上がりを感じています。

本記事では、前編と後編の2回に分けて、Elixirをまったく知らない読者を対象に、Elixirの概要を解説します。前半ではElixirの紹介とインストール方法の説明を行い、簡単なサンプルプログラムを使ってElixirの特徴について解説していきます。

Elixirの基礎知識

まずは、Elixirと関係が深いErlangについてみていきましょう。

Erlang/OTPとは

Erlang[注4]はエリクソン社によって開発された、並列処理を取り扱うのが得意なプログラミング言語です。OTP（Open Telecom Platform）という、アプリケーション作成のための汎用的な処理やパターンを抽象化したフレームワーク／ライブラリ群と一緒に配布されています。そのため、ErlangとOTPを合わせて「Erlang/OTP」と呼ばれることが多いです。

OTPはサーバの振る舞いやパターンを「ビヘイビア」という規約で抽象化し、面倒な例外処理や監視などのしくみを隠蔽します。開発者はこのビヘイビアを使い規約に従ったコードを書くことで、ユーザに提供したい処理に専念して開発を行うことができるようになります。Elixirも、ErlangのOTPをラップしたビヘイビアモジュールを標準ライブラリとしてバンドルしています。**表1**は主要なビヘイビアの一覧です。後編では、

注1） URL http://elixir-lang.org

注2） あらかじめ決められた時間内に処理が終了しなくても、深刻な影響を及ぼさないが、提供するサービスの価値は低下するというシステムの特性。

注3） Dave Thomas 著、笹田 耕一、鳥井 雪 訳、オーム社、2016年、ISBN = 978-4-274-21915-3 URL http://shop.ohmsha.co.jp/shopdetail/000000004675

注4） URL https://www.erlang.org

▼表1 主要ビヘイビアー覧

名称	振る舞い
GenServer	リクエストを受けてレスポンスを返すクライアント／サーバ
GenEvent	動的に追加・削除できるイベントハンドリング
Supervisor	プロセスの死活監視と、指定可能な設定(戦略)でのプロセスの再起動
Application	ErlangVM上で動作する再利用可能なアプリケーションの起動と停止

ビヘイビアを使ったサンプルアプリを作成します。

Erlang/OTPの学習は必要?

筆者はイベントやMeetUpで「Elixirを使うためにErlang/OTPの学習は必要ですか?」とよく質問されます。この質問に対する回答は「作成するプログラムが小規模なら必須ではない、中規模以上であればOTPの概念は知っておく必要があるので、学習したほうが良い」です。

使い捨てのスクリプトや規模の小さなプログラムを書く分には良いのですが、ある程度の規模のアプリケーションになってくると、Supervisorビヘイビアによる死活監視と再起動、Applicationビヘイビアによるアプリケーションの起動と停止、標準的なクライアント／サーバの処理のためのGenServerビヘイビアなど、OTPを使ったアプリケーションを開発することになるでしょう。ですので、OTPの概念を理解する必要があり、ある程度のErlang/OTPについての学習は必要だと考えています。

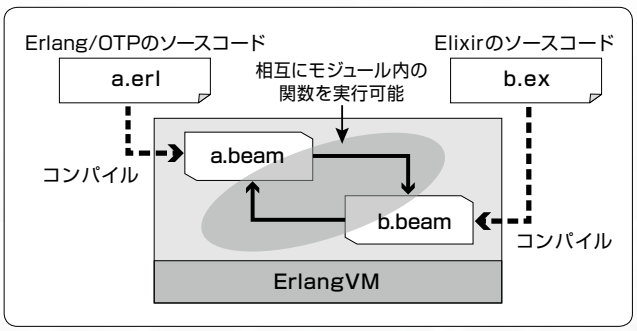
ElixirとErlang/OTPの関係

ElixirはErlangVMの上で動作します。これはどういうことでしょうか?

Elixirのプログラムが実行される際、Elixirのソースコードは BEAM[注5] ファイルという実行ファイルにコンパイルされ、ErlangVMにロードされて実行されます。Erlang/OTPのプログラムも同様に、ソースコードがBEAMファイルにコンパイルされ、ErlangVMにロードされて実行されます。

これらElixirとErlang/OTPのBEAMファイルはそれぞれ同じ形式で、同じErlangVM上で動作し、相互にモジュール中の関数を実行できます(**図1**)。

▼図1 ElixirとErlang/OTPの関係図



注5) BEAMはBogdan's Erlang Abstract Machineの略です。BEAMファイルの拡張子は「.beam」になります。

Elixirの得意なこと／苦手なこと

ElixirはErlangVM上で動作するので、分散システム、耐障害性、ソフトリアルタイムといったErlang/OTPの特徴を継承しています。このことからElixirは、ネットワークサーバ、高負荷な大規模システム、高可用性システム、並列・分散処理システムといったシステムを実装するのに向いています。

一方、Erlang/OTPは数値計算や高速な処理を行うのには向いていません。したがってErlangVM上で動作するElixirも、高速な処理速度を求められるシステムや、大量のデータを扱う数値計算アルゴリズム処理といったシステムを実装するのには向いていません。

Elixirの環境構築

ここからはElixirのインストール手順を解説します。なおErlang/OTPとElixirのバージョンは、次に挙げる執筆時点(2016年9月)の最新バージョンを前提としています。

・Elixir 1.3.1 / Erlang/OTP 19.0

▼図2　Erlang/OTPのREPLの起動

```
$ erl
Erlang/OTP 19 [erts-8.0] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]

Eshell V8.0  (abort with ^G)
1>
```

▼図3　ショートカットによるErlang/OTPの終了方法

```
2> <CTRL+C>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
a
$
```

Erlang/OTPのインストール

ElixirはErlangVM上で実行されるので、まずErlang/OTPをインストールする必要があります。それではErlang/OTPのインストールと起動方法を見ていきましょう。

OS Xの場合

OS XではHomebrewでErlang/OTPをインストールできます。

```
$ brew update
$ brew install erlang
```

すでにErlang/OTPをインストール済みの方は次のコマンドでErlang/OTPを最新のバージョンに更新してください。

```
$ brew update
$ brew upgrade erlang
```

Windowsの場合

Windowsでは後述するElixirの専用のインストーラを使うことで、Erlang/OTPのインストールを、Elixirと同時に行うことができます。Erlang/OTPを個別にインストールする必要はありません。

Linuxの場合

Linuxではディストリビューションごとにパッケージが提供されています。公式サイトのダウンロードページ[注6]から対応する手順でインストールしてください。

起動確認

Erlang/OTPの対話環境（REPL）の起動は、erlコマンドで行います（図2）。

それでは、伝統に則ってハローワールドプログラムを実行してみましょう。Erlang/OTPのプログラムでは、命令の最後にドット（.）を付ける必要があります。

```
1> io:fwrite("Hello, Erlang!¥n").
Hello, Erlang!
ok
```

終了するには次のようにq()と入力するか、図3のようにCtrl-Cのあとにaを入力してください。

```
2> q().
ok
3>
$
```

Elixirのインストール

次に、Elixirのインストールを行います。

OS Xの場合

OS XではErlang/OTP同様、HomebrewでElixirをインストールできます。

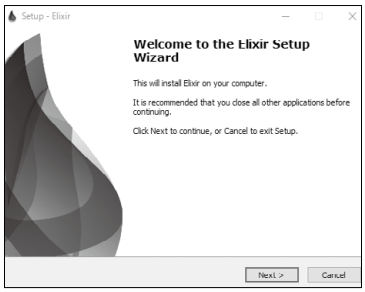
```
$ brew update
$ brew install elixir
```

すでにElixirをインストール済みの方は、次のコマンドでElixirを最新のバージョンに更新してください。

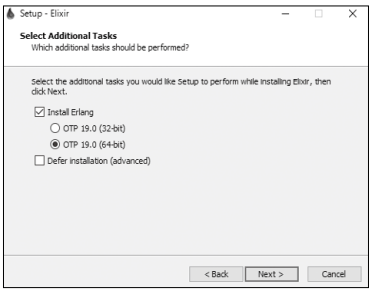
```
$ brew update
$ brew upgrade elixir
```

注6）URL https://www.erlang.org/downloadsの［Pre-built Binary Packages］

▼図4 Elixirインストーラ画面



▼図5 64bit、32bitの選択画面



▼図7 Windowsメニュー一覧(Windows 10)

▼図6 ElixirのREPLの起動

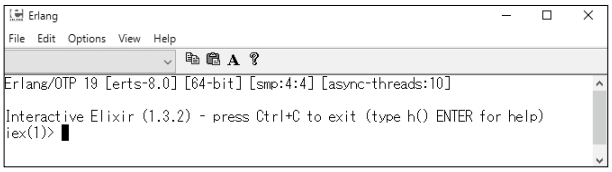
```
$ iex
Erlang/OTP 19 [erts-8.0] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]

Interactive Elixir (1.3.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)>
```

Windowsの場合

Windowsでは、Elixirの専用インストーラ[注7]を使って、ElixirとErlang/OTPをインストールできます(**図4**)。WindowsのErlang/OTPには32bit版と64bit版の2種類があり、どちらをインストールするか指定できます(**図5**)。お使いのPCに合わせて選択してください。

▼図8 iexコンソールウィンドウ



▼図9 ショートカットによるElixirの終了方法

```
iex(2)> <CTRL+C>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
a
$
```

Linuxの場合

Linuxではディストリビューションごとにパッケージが提供されています。公式サイトのインストールページ[注8]から対応するインストール手順でインストールしてください。

起動確認

Elixirの対話環境(REPL)の起動はiexコマンドで行います。OS XまたはLinuxの場合はコンソールからiexと入力してください(**図6**)。Windowsの場合は、メニュー一覧(**図7**)から「Elixir」を選んで起動すると、iexのコンソールウィンドウが立ち上がります(**図8**)。

ハローワールドプログラムの実行は次のようになります。

```
iex(1)> IO.puts "Hello, Elixir"
Hello, Elixir
:ok
iex(2)>
```

終了するには、次のように、

```
iex(2)> System.halt
$
```

と入力するか、**図9**のようにCtrl-Cのあとにaを入力してください[注9]。

注7) URL http://elixir-lang.org/install.html の[Download the installer]からダウンロードできる。

注8) URL http://elixir-lang.org/install.html の[unix-and-unix-like]

注9) Windowsの場合は、iexコンソールウィンドウのメニューから「File」->「Exit」を選択するか、ALT-F4を入力して終了してください。

WindowsでErlangを使いたい場合は、Elixirと同様にメニュー一覧から「Erlang OTP 19 (x64)」を選んで対話環境を起動します。

関数型言語とは

Elixirは関数型言語です。関数型言語や関数型プログラミングと聞くと難しいと思う方もいるかもしれませんが、ご心配なく。関数型とは要するに、命令やオブジェクトを中心に考えてプログラミングを行うのではなく、「関数」を中心に考えてプログラミングを行うというだけのことです。

複数の関数を組み合わせて処理を記述するプログラミングのスタイルを、関数型プログラミングと言います。関数型プログラミングを行ううえで欠かせない、「関数の生成、代入、演算、引数や戻り値としての受け渡しなどの基本操作を制限なしに使用できる」[注10]といった特徴を持ち、関数型プログラミングの考えに基づいて設計された言語を関数型言語と言います[注11]。

実践：関数型プログラミング

曜日計算プログラム

それではサンプルとして、年月日から曜日を計算するプログラムを、関数型／手続き型それぞれのスタイルで見ていきましょう。なお曜日計算には、ツェラーの公式[注12]を使って行います。

作成するプログラムは次のような動作を行うものとします[注13]。

(1)引数が年、月、日の整数でない場合は例外を起こす
(2)引数に日付を入力すると曜日を計算する
(3)変換内容を出力する
(4)返り値として曜日を返却する

Rubyを使って手続き型で実装

日付から曜日を計算する処理をRubyで実装したものが**リスト1**となります。このプログラムを実行すると**図10**のような結果となり、日付から曜日を計算できていることがわかります。2016年9月5日は月曜日、2016年9月6日は火曜日です。

Date2Weekクラスのcalcメソッドに注目してください(**リスト1の①**)。プログラムの仕様の、(1)入力値チェック、(2)曜日計算、(3)計算内容の表示、(4)曜日の返却、の処理をそれぞれ順番に実行しています。よく見かけるコードのスタイルではないでしょうか。

Elixirを使って関数型で実装

Elixirによる実装は**リスト2**のようなコードになります[注14]。このプログラムを実行すると、**図11**のような結果となります。

パイプ演算子

Date2Weekモジュールのcalc/1関数[注15]を見てください(❶)。|>という記号を使って関数がつながっている様子がわかります。|>はパイプ演算子と言い、左側の式の結果を受け取り、右側の関数の第一引数として渡す演算子です。

calc/1関数はこのパイプ演算子を使って、年月日を表すタプル[注16]を引数に、(1)入力値チェック、(2)曜日計算、(3)計算内容の表示、(4)曜日の返却、の処理を行う関数に対して、ベルトコンベアーのように次々と、関数の出力結果を次

注10) このような操作の対象を第一級オブジェクト、またはファーストクラスと言います。
注11) 関数型言語の正確な定義は存在しません。「関数を中心に処理を記述するスタイルを推奨する言語なのであれば、それは関数型言語である」というのが筆者の考えです。
注12) URL https://ja.wikipedia.org/wiki/ツェラーの公式
注13) 今回サンプルとして作成するプログラムでは閏年などの扱いを省略しているため、厳密には正確ではありません。
注14) Elixirのソースコードの拡張子は「.ex」または「.exs」です。「.ex」ではバイトコードにコンパイルして実行するのに対して、「.exs」ではソースレベルで解釈・実行します。
注15) Elixirでは関数の名前が同じであっても引数の数が違うと異なる関数として扱われます。関数名と引数の数の組み合わせを「アリティ」と言います。たとえば、関数名がhoge、引数の数が2の関数のアリティは、hoge/2となります。
注16) コンマ区切りの順序のある要素を波括弧で囲むデータ構造を「タプル」と言います。タプルの要素は、タプルとインデックスを引数とするelem/2関数で取り出すことができます。たとえば、elem({1,2,3}, 0)は1を返します。

▼リスト1　sample.rb

```
class Date2Week
  require 'date'
  attr_accessor :year, :month, :day

  # 年月日から曜日を計算
  def calc
    validate_args                                          # (1)入力値チェック
    week_no = calc_week_no                                 # (2)曜日数計算
    week_str = week_no_to_str(week_no)                     # (2)曜日数を曜日に変換
    puts "#{@year}/#{@month}/#{@day}は#{week_str}です。" # (3)計算内容を表示
    week_str                                               # (4)曜日を返却
  end

  def initialize(year, month, day)
    @year, @month, @day = year, month, day
  end

  # year, month, dayが整数でなければ例外を投げる
  def validate_args
    if !@year.is_a?(Integer) || !@month.is_a?(Integer) || !@day.is_a?(Integer)
      raise "引数がマッチしません"
    end
  end

  # ツェラーの公式で曜日番号を計算
  # 曜日番号は、0:土曜、1:日曜、...として表す
  def calc_week_no
    y_1 = @year + @year / 4
    y_2 = y_1 - @year / 100
    y_3 = y_2 + @year / 400
    m_4 = (@month * 13 + 8) / 5
    (y_3 + m_4 + @day) % 7
  end

  # 曜日番号を文字列に変換
  def week_no_to_str(week_no)
    weeks = ["日", "月", "火", "水", "木", "金", "土"]
    weeks[week_no] + "曜日"
  end
end
```

…①

▼図10　リスト1「sample.rb」の実行結果（Rubyの REPL「irb」で実行）

```
irb(main):001:0> load 'sample.rb'
=> true
irb(main):002:0> Date2Week.new(2016,9,5).calc
2016/9/5は月曜日です。
=> "月曜日"
irb(main):003:0> Date2Week.new(2016,9,6).calc
2016/9/6は火曜日です。
=> "火曜日"
irb(main):004:0>
```

の関数の入力値として適用していきます(**図12**)。関数を通るたびに、一番最初の引数のタプルがどんどん変換されていく様子がわかるでしょう。

なお、パイプ演算子を使わずに書いたのがcalc2/1関数です(❷)。関数の結果を、次の関数の第一引数として引き渡していく様子が見て取れると思います。

パターンマッチ

calc2/1関数やcalc_week_no/1関数には=演算子が使われています。Elixirの=演算子は代入演算子ではなく、パターンマッチ演算子です。パターンマッチ演算子=は、まず右辺を評価し、次に左辺を評価し、両辺が等しくなる方法があれば成功し、その値を返します。

iexでパターンマッチの簡単な例(**図13**)を実行してみましょう。

1行目のx = 1は代入ではなくパターンマッチです。まず、右辺の1が評価されて整数値「1」になります。左辺は変数ですので右辺の「1」を左辺の変数xに束縛することで両辺が等しくなります。2行目の1 = xでは、右辺のxは「1」に束縛さ

▼リスト2　sample.exs

```
defmodule Date2Week do
  # 年月日から曜日を計算
  def calc({year, month, day}) do
    {year, month, day}
    |> validate_args  # (1)入力値チェック
    |> calc_week_no   # (2)曜日数計算
    |> week_no_to_str # (2)曜日数を曜日に変換
    |> display        # (3)計算内容を表示
    |> select_week    # (4)曜日を返却
  end

  # calcをパイプ演算子を使わずに書いたバージョン
  def calc2({year, month, day}) do
    arg0 = {year, month, day}
    arg1 = validate_args(arg0)  # (1)入力値チェック
    arg2 = calc_week_no(arg1)   # (2)曜日数計算
    arg3 = week_no_to_str(arg2) # (2)曜日数を曜日に変換
    arg4 = display(arg3)        # (3)計算内容を表示
    arg5 = select_week(arg4)    # (4)曜日を返却
    arg5
  end

  # year, month, day が整数でなければ例外を投げる
  def validate_args({year, month, day})
  when is_integer(year) and is_integer(month) and is_integer(day) do
    {year, month, day}
  end
  def validate_args(_), do: raise "引数がマッチしません"

  # ツェラーの公式で曜日番号を計算
  # 曜日番号は、0:土曜、1:日曜、...として表す
  # 引数の最後に、曜日を追加して返却
  def calc_week_no({year, month, day}) do・・・
    y_1 = year + (div year, 4)
    y_2 = y_1 - (div year, 100)
    y_3 = y_2 + (div year, 400)
    m_4 = div (month * 13 + 8), 5
    week_no = rem (y_3 + m_4 + day), 7
    {year, month, day, week_no}
  end

  # 引数の最後の曜日番号を文字列に変換
  def week_no_to_str({y, m, d, 0}), do: {y, m, d, "日曜日"}
  def week_no_to_str({y, m, d, 1}), do: {y, m, d, "月曜日"}
  def week_no_to_str({y, m, d, 2}), do: {y, m, d, "火曜日"}
  def week_no_to_str({y, m, d, 3}), do: {y, m, d, "水曜日"}
  def week_no_to_str({y, m, d, 4}), do: {y, m, d, "木曜日"}
  def week_no_to_str({y, m, d, 5}), do: {y, m, d, "金曜日"}
  def week_no_to_str({y, m, d, 6}), do: {y, m, d, "土曜日"}

  # 変換内容を出力
  def display({year, month, day, week}) do
    IO.puts "#{year}/#{month}/#{day}は#{week}です。"
    {year, month, day, week}
  end

  # 最後の曜日を抽出して返却する
  def select_week({_year, _month, _day, week}), do: week
end
```

❶ (1)入力値チェック (2)曜日数計算 (2)曜日数を曜日に変換 (3)計算内容を表示 (4)曜日を返却

❷ (1)入力値チェック (2)曜日数計算 (2)曜日数を曜日に変換 (3)計算内容を表示 (4)曜日を返却

❸ 引数の最後の曜日番号を文字列に変換

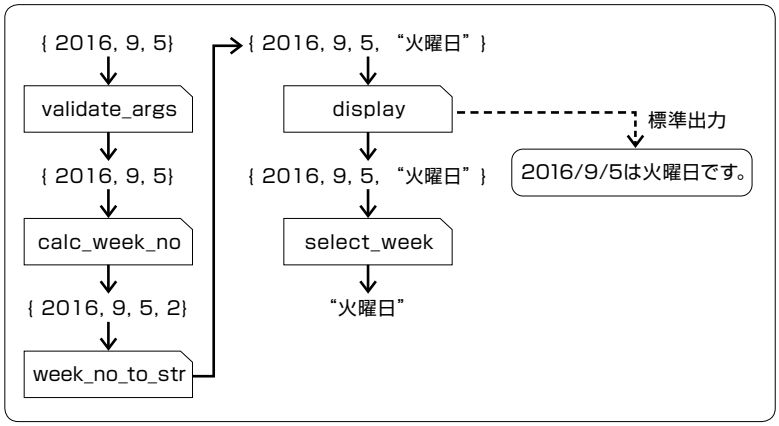
▼図11　リスト2「sample.exs」の実行結果

```
$ iex sample.exs
Erlang/OTP 19 [erts-8.0] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]
Interactive Elixir (1.3.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> Date2Week.calc({2016, 9, 5})
2016/9/5は月曜日です。
"月曜日"
iex(2)> Date2Week.calc({2016, 9, 6})
2016/9/6は火曜日です。
"火曜日"
iex(3)>
```

▼図13　パターンマッチ

```
iex(1)> x = 1
1
iex(2)> 1 = x
1
iex(3)> 2 = x
** (MatchError) no match of right hand side value: 1
iex(3)>
```

▼図12　パイプ演算子



れているので評価されて「1」になります。左辺は「1」ですので両辺値が等しくなり、成功します。3行目の2 = xは、右辺のxは「1」に束縛されているので評価されて「1」になります。左辺は整数値「2」で、どうやっても両辺が等しくならず、マッチエラーとなります[注17]。

関数の引数のパターンマッチ

次に、week_no_to_str/1関数を見てみましょう(❸)。一種類の関数に複数のボディがあることに驚いたのではないでしょうか？

Elixirは渡された引数を、定義されている順にパラメータリストにパターンマッチさせようとします。week_no_to_str/1の場合は、{y, m, d, 0}、{y, m, d, 1}、……と定義されている順にパターンマッチを繰り返していきます。

たとえば、引数が{2016, 9, 6, 2}のときは、{y, m, d, 0}、{y, m, d, 1}とのパターンマッチが失敗し、{y, m, d, 2}とのパターンマッチが成功して、yが2016、mが9、dが6に束縛され、{y, m, d, “火曜日”}つまり{2016, 9, 6, “火曜日”}が関数の結果として返されます。

◆ ◆ ◆

本記事で、Elixirの紹介とインストール手順の説明、ElixirとErlangの関係の説明、曜日計算プログラムを例に、手続き型によるプログラミングとの比較、パイプ演算子やパターンマッチといったElixirの言語機能について説明しました。

今回取り上げなかったマクロによるメタプログラミングや、さまざまな標準モジュールについて知りたい方、さらに深くElixirを理解したい方はぜひ『プログラミングElixir』や公式のドキュメント[注18]を読んでみてください。

次回はいよいよ、Elixirの目玉である「プロセス」を使った並列プログラミング、そしてPhoenixによるWebアプリ開発について解説します。お楽しみに！ SD

注17) Elixirのパターンマッチは、左辺の変数の値しか変更しません。

注18) URL http://elixir-lang.org/getting-started/introduction.html

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7回

SSL/TLSの暗号化通信を復号してみよう!

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、1年ぶりです！　脳内でPrinceが「農！協！牛！乳！」と叫んでいるEiji James Yoshidaです。今年一発目の「Jamesのセキュリティレッスン」ですので音楽ネタは何を書こうか悩みましたが、やはりDavid BowieやPrince、元SOFT BALLETの森岡賢といった大好きなアーティストが今年になって亡くなったことを書かずにいられないですね。まだまだ長生きして、刺激的で実験的な曲を作り続けてほしかったです。

さて、前回(2015年11月号)まではディスプレイフィルタを習得するような内容が続いたので、今回はディスプレイフィルタではなく、SSL/TLSの暗号化通信を復号する方法について説明します。

環境説明

本稿を書く際に使用した環境はWindows 10でWireshark 2.2.0です。使い慣れたWireshark 1.x系を使いたいところですが、今年の7月末でサポート終了(End of Life)となってしまったため、思い切ってWireshark 2.x系に乗り換えました。

・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の設定でインストールしてください。また、筆者のブログからキャプチャファイルをダウンロードしてください。

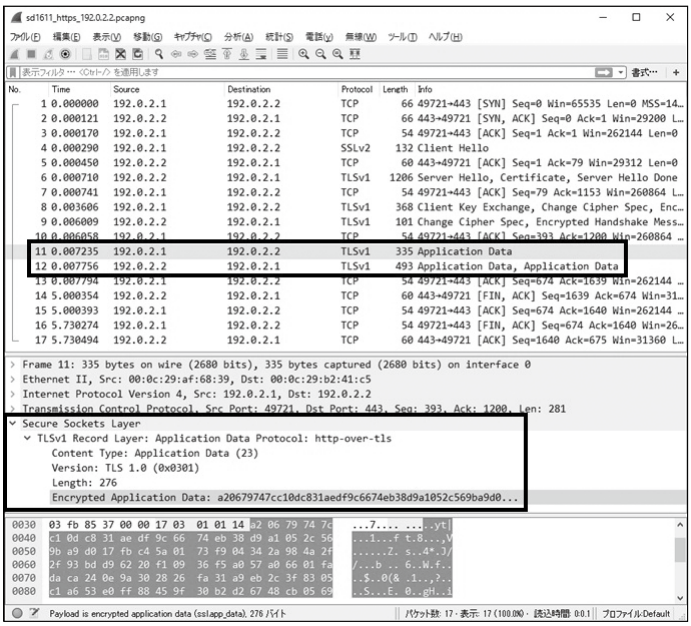
・Software Design 短期集中連載「Jamesのセキュリティレッスン」用キャプチャファイル
http://d.hatena.ne.jp/EijiYoshida/20140907/1410071296
  ・sd1611_https_192.0.2.2.pcapng
  ・sd1611_https_192.0.2.3.pcapng
  ・sd1611_192.0.2.2_server.key
  ・sd1611_192.0.2.3_server.pfx
  ・sd1611_sslkeylogfile.txt

あとは何かBGMでも流しましょう。先ほど書いたアーティストの曲を聴いたことがなければ、David Bowieの「Let's Dance」やPrinceの「Batdance」、あとはSOFT BALLETの「EGO DANCE」といった曲は、懐かしい音ですが聴きやすいのでお勧めです。

多くの通信が暗号化され始めている

近年、セキュリティが重要視されていることから多くの通信が暗号化され始めています。実際、Wiresharkを使って通信内容を確認しようとパケットをキャプチャしたところ、暗号化されていて内容がわからないという経験をした人

▼図1　SSL/TLSで暗号化されているHTTPSの通信



は多いと思います。筆者もダウンロードしたツールの動作確認をしようとパケットキャプチャしてみたら、内容がSSL/TLSで暗号化されていて動作確認できずに困ったことがあります。

そこで今回は、Wiresharkの機能を使ってSSL/TLSの暗号化通信を復号する方法を解説してみたいと思います。

HTTPSで使われているSSL/TLSの暗号化通信を復号してみよう

多くの人がお世話になっているSSL/TLSを利用した通信といえばHTTPSだと思うので、今回はこのHTTPSで使われているSSL/TLSの暗号化通信を復号してみましょう。

WiresharkでHTTPSの通信をキャプチャしただけでは、内容はSSL/TLSで暗号化されていてわからないです。試しに筆者のブログからダウンロードしたキャプチャファイル「sd1611_https_192.0.2.2.pcapng」をWiresharkで開いてください。このキャプチャファイルはHTTPSの通信で送受信されたパケットを保存したものですが、上側の[Packet List]ペインのNo.11やNo.12のパケットを選択して、中央の[Packet Details]ペインにある[Secure Sockets Layer]配下を展開すると、**図1**のようにデータ部分が**Encrypted Application Data:**と表示されていることから、SSL/TLSで暗号化されていることがわかります。

WiresharkでSSL/TLSの暗号化通信を復号する方法は、大きく分けて2つあります。1つはサーバ秘密鍵を使う方法、もう1つは(Pre)-Master-Secretを使う方法です。

SSL/TLSの暗号化通信をサーバ秘密鍵で復号する

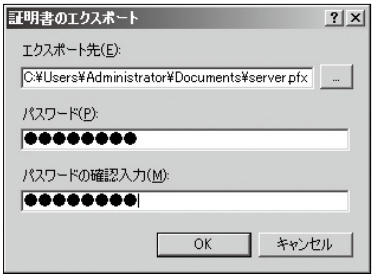
まずはサーバ秘密鍵を使ってSSL/TLSの暗号化通信を復号してみましょう。

サーバ秘密鍵を入手する

Wiresharkでサーバ秘密鍵を使ってSSL/TLSの暗号化通信を復号するには、当然ですが通信相手であるWebサーバのサーバ秘密鍵を入手する必要があります。

▼図2　サーバー証明書の[エクスポート]リンク

▼図3　[証明書のエクスポート]ウインドウ



● Apache (mod_ssl) のサーバ秘密鍵

Apacheでmod_ssl[注1]を使っている場合は、インストール方法や設定によって異なることもありますが一般的に「/etc/httpd/conf.d/ssl.conf」ファイルの「SSLCertificateKeyFile」ディレクティブに、サーバ秘密鍵の保存場所が書かれています。

筆者はテスト環境として、CentOS 7.2でApacheとmod_sslを使っているので、上記ファイルの「SSLCertificateKeyFile」ディレクティブを確認したところ「SSLCertificateKeyFile /etc/pki/tls/private/server.key」と記載されていることから、サーバ秘密鍵は「/etc/pki/tls/private/server.key」ファイルということがわかります。

このファイルが、環境説明のときに筆者のブログからダウンロードした「sd1611_192.0.2.2_server.key」ファイルになります。

注1）ApacheをSSLに対応させるためのモジュール。

● IIS/7.x系のサーバ秘密鍵

Microsoft-IIS/7.x系を使っている場合は、「インターネットインフォメーションサービス(IIS)マネージャ」を使います。

筆者はテスト環境としてWindows Server 2008でIIS/7.0を使っているので、[管理ツール]にある[インターネットインフォメーションサービス(IIS)マネージャ]を起動します。中央ペインにある[サーバー証明書]をダブルクリックするとサーバ証明書の一覧が表示されるので、証明書を選んでから右側ペインにある[エクスポート]リンク(**図2**)をクリックします。[証明書のエクスポート]ウインドウ(**図3**)が表示されたら、[エクスポート先:]にパスとファイル名、「パスワード:」と「パスワードの確認入力:」にはサーバ証明書自体の暗号化と復号に使うパスワードを入力して[OK]ボタンをクリックすると、サーバ秘密鍵を含むサーバ証明書がPKCS#12(PFX)ファイル形式でエクスポートされます。

このファイルが、環境説明のときに筆者のブログからダウンロードした「sd1611_192.0.2.3_server.pfx」ファイルになります。

Wiresharkにサーバ秘密鍵を設定する

サーバ秘密鍵を入手したら、さっそくWiresharkにサーバ秘密鍵を設定しましょう。

● 手順と注意点

キャプチャファイル「sd1611_https_192.0.2.2.pcapng」をWiresharkで開いて、Protocol項目にSSLv～やTLSv～と表示されているパケットを右クリックします。今回はNo.4のパケットを右クリックしてコンテキストメニュー(**図4**)から[プロトコル設定]を選び、[Open Secure Sockets Layer preferences]をクリックします。

[Wireshark・設定]ウインドウ(**図5**)が表示されたら、RSA keys listの右にある[Edit]ボタン(**図5の①**)をクリックします。[SSL Decrypt]ウインドウ(**図6**)が表示されたら、左下の[+]ボタンをクリックしてSSL/TLSの復号に必要な情報を設定します。各項目をダブルクリックすると入力に切り替わります。

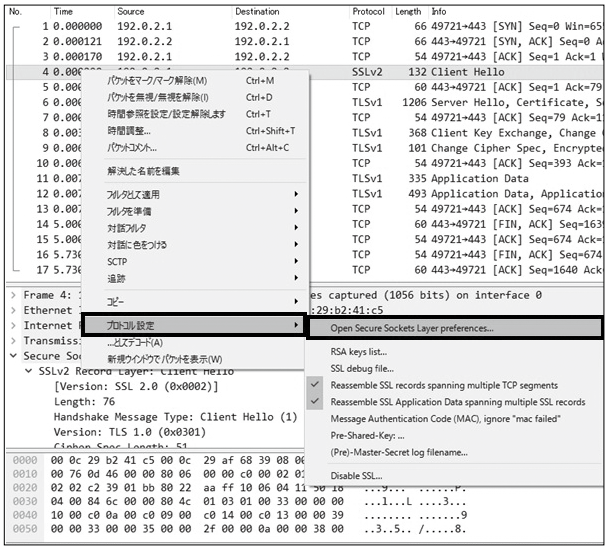
ここで注意すべき点として、Enterを押すと[OK]ボタンが押されたことになり、ほかの項目の入力が終わっていない場合はエラーになってしまうので、Enterは押さないことをお勧めします。もしEnterを押してエラーになったら、再度RSA keys listの右にある[Edit]ボタンをクリックして[SSL Decrypt]ウインドウを表示し、Enterを押してしまった項目を含む行を選択してから、[－]ボタンをクリックして行ごと削除してください。その後、[＋]ボタンをクリックして再度入力します。少々面倒ですが、そうしないと筆者の環境ではEnterで余計なデータが入力されてしまったのか、復号処理がエラーになってしまいました。

● Apache(mod_ssl)の復号設定

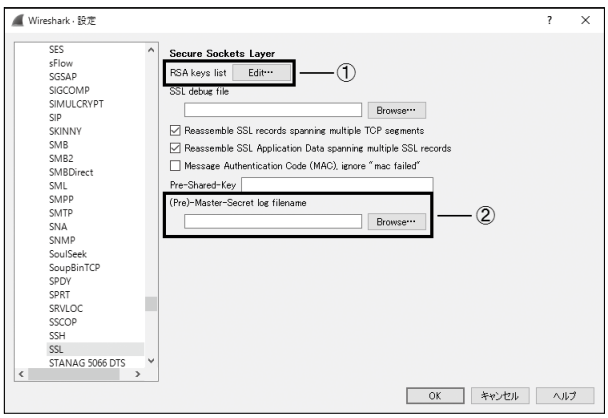
それでは筆者のテスト環境にある192.0.2.2のApache(mod_ssl)のHTTPSを復号する設定をしましょう。

IP address項目には、サーバ秘密鍵を設定し

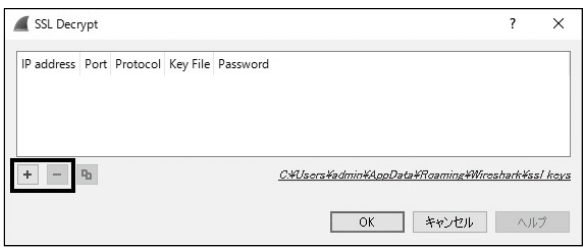
▼図4　No.4パケットのコンテキストメニュー



▼図5　[Wireshark・設定]ウインドウ



▼図6　[SSL Decrypt]ウインドウ



ているWebサーバのIPアドレスを入力します。今回は**192.0.2.2**と入力してください。

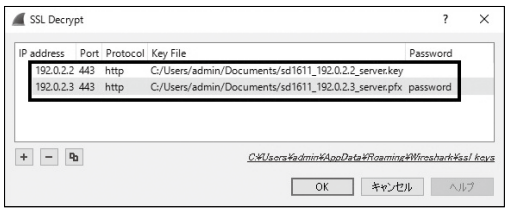
Port項目には、WebサーバがSSL/TLSで使用しているポート番号を入力します。今回は**443**と入力してください。

Protocol項目には、SSL/TLSで暗号化している通信のプロトコルを入力します。HTTPSであればSSL/TLSでHTTPの通信を暗号化しているので、**http**と入力します。今回も**http**と入力してください。

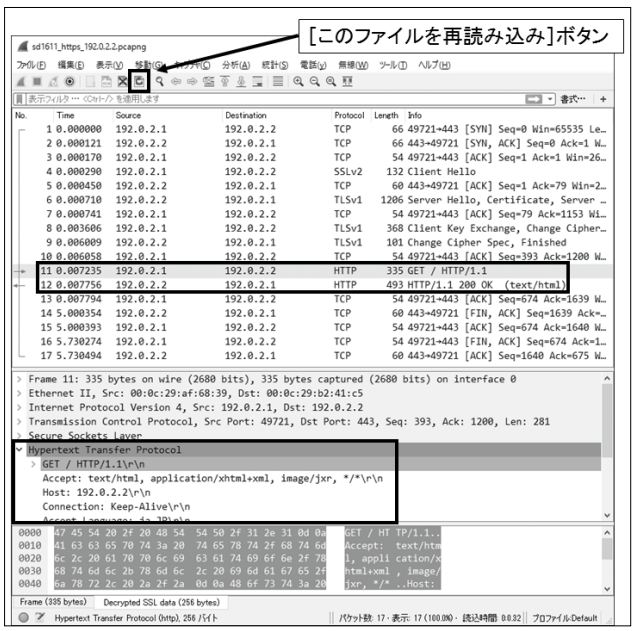
Key File項目には、サーバ秘密鍵のパスとファイル名を設定します。今回は筆者のブログからダウンロードした「sd1611_192.0.2.2_server.key」を設定してください。

Password項目には、サーバ秘密鍵のファイルが暗号化されている場合に復号で使用するパスワードを設定してください。今回使用する「sd1611_192.0.2.2_server.key」はパスワードがないので何も入力しないでください。

▼図7　設定が終わった[SSL Decrypt]ウインドウ



▼図8　192.0.2.2の復号されたHTTPSの通信



● IIS/7.0の復号設定

続いて、筆者のテスト環境にある192.0.2.3のIIS/7.0のHTTPSを復号する設定をしましょう。[+]ボタンをクリックしてIP address項目には**192.0.2.3**、Port項目には**443**、Protocol項目には**http**、Key File項目には筆者のブログからダウンロードした「sd1611_192.0.2.3_server.pfx」、Password項目にはすべて小文字の**password**と入力してください。

◆　◆　◆

これら入力がすべて終わると**図7**のような表示になります。設定が正しいことを確認したら[OK]ボタンをクリックして[SSL Decrypt]ウインドウを閉じます。さらに[Wireshark・設定]ウインドウ(**図5**)の[OK]ボタンをクリックしてウインドウを閉じます。Wiresharkのメインウインドウが設定を反映した表示になっていないこともあるので、念のためメイン・ツールバーにある[このファイルを再読み込み]ボタン(**図8**)をクリックしてキャプチャファイルを再読み込みしてください。

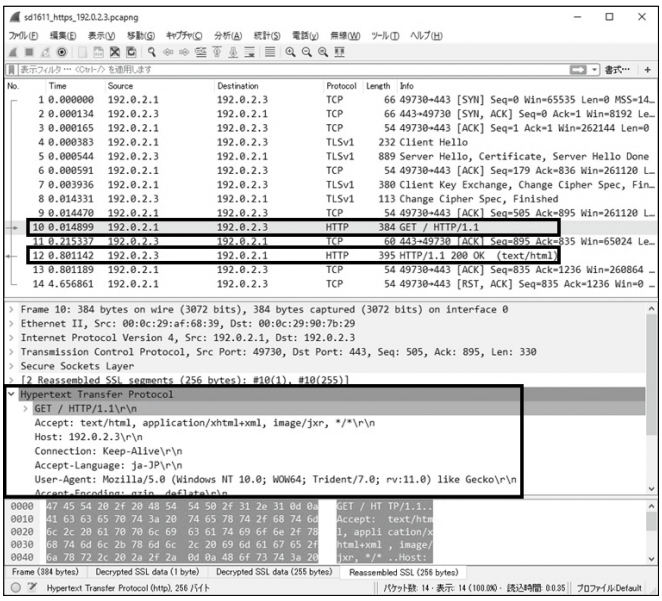
図8のようにNo.11とNo.12のパケットの表示が変わって、[Packet List]ペインのProtocol項目が**TLSv1**から**HTTP**になり、Info項目に通信内容の一部分が表示されています。さらに[Packet List]ペインでNo.11やNo.12のパケットをクリックすると、Packet Detailsペインにある[Secure Sockets Layer]の下に、新しく[Hypertext Transfer Protocol]が表示されて、そこを展開するとSSL/TLSで暗号化されている

HTTPの通信、つまりHTTPSの通信が復号されて平文で丸見えになっていることがわかると思います。

次にIIS/7.0とのHTTPSの通信が保存されている「sd1611_https_192.0.2.3.pcapng」をWiresharkで開いてください。すでに復号に必要な設定をしているので、**図9**のようにNo.10とNo.12のパケットが復号されていることがわかります。

このように、Wiresharkはサーバ秘密鍵を使ってSSL/TLSの暗号化通信を復号できます。[SSL Decrypt]ウインドウで設定した内容は削除しない限り残るので、安全性を考えて設定を削除しましょう。先ほどのサーバ秘密鍵を設定する手順に従って[SSL Decrypt]ウインドウ(**図7**)を表示したら、追加した行を選択して[−]ボタンをクリックすれば削除されます。あとは[SSL Decrypt]ウインドウの[OK]ボタンをクリックしてウインドウを閉じてください。

▼図9　192.0.2.3の復号されたHTTPSの通信



SSL/TLSの暗号化通信を(Pre)-Master-Secretで復号する

Wiresharkはサーバ秘密鍵を使って復号する方法のほかにも、(Pre)-Master-Secretを使って復号する方法があります。(Pre)-Master-SecretはPre-Master-SecretやMaster-Secretのことを指していて、この2つのどちらかを使ってSSL/TLSで暗号化された通信を復号します。この(Pre)-Master-SecretはSSL/TLSで通信しているサーバとクライアントの両方に生成されるため、サーバ秘密鍵がなくてもクライアントの(Pre)-Master-Secretがあれば復号できます。もちろん、復号したい暗号化通信で使われている(Pre)-Master-Secretを入手しないと復号はできないです。

Windowsで(Pre)-Master-Secretを入手する

Windowsで(Pre)-Master-Secretを入手する簡単な方法は、(Pre)-Master-Secretの情報を含むSSLKEYLOGFILEを出力できるブラウザを使うことです。SSLKEYLOGFILEを出力できるブラウザとしてはGoogle ChromeやMozilla Firefoxがあります。どのバージョンから対応しているのかはわかりませんでしたが、筆者のテスト環境ではGoogle Chromeはバージョン53系、Mozilla Firefoxはバージョン48系で出力できることを確認しました。また、筆者のテスト環境にあるMicrosoft Internet Explorer 11系とMicrosoft Edge 25系ではSSLKEYLOGFILEの出力はできませんでした。

Google ChromeやMozilla FirefoxでSSLKEYLOGFILEを出力するには、環境変数を設定する必要があります。[コントロールパネル]の[システムとセキュリティ]にある[システム]をクリックして、左側にある[システムの詳細設定]リンク(**図10**)をクリックします。[システムのプロパティ]ウインドウ(**図11**)が表示された

ら、[詳細設定]タブの下側にある[環境変数]ボタンをクリックします。[環境変数]ウインドウ(**図12**)が表示されたら、中央にある[ユーザー環境変数]の[新規]ボタン(**図12の①**)をクリックします。[ユーザー変数の編集]ウインドウ(**図13**)が表示されたら、[変数名:]には**SSLKEYLOGFILE**、[変数値:]にはSSLKEYLOGFILEを出力するパスとファイル名を入力します。今回は**%USERPROFILE%¥Documents¥sslkeylogfile.txt**と入力してください。あとは[ユーザー変数の編集]ウインドウと[環境変数]ウインドウ、そして[システムのプロパティ]ウインドウの[OK]ボタンをクリックして各ウインドウを閉じます。

設定後はGoogle ChromeやMozilla FirefoxでHTTPSの通信を行うと、[ドキュメント]フォルダに(Pre)-Master-Secretの情報を含むsslkeylogfile.txtが作成されます。試しに、「https://www.example.com/」に接続すると、みなさんの[ドキュメント]フォルダにsslkeylogfile.txtが作成されたかと思います。筆者のテスト環境で出力したファイルは、環境説明のときに筆者のブログからダウンロードした「sd1611_sslkeylogfile.txt」になります。

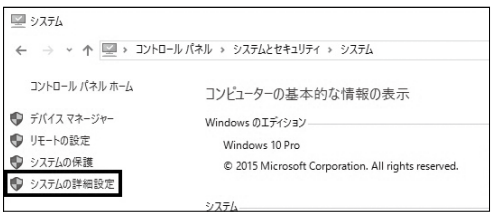
ちなみに、Linuxでも同様の環境変数を設定した端末からブラウザを起動することで、(Pre)-Master-Secretの情報を含むファイルを出力することができます。

Wiresharkに(Pre)-Master-Secretを設定する

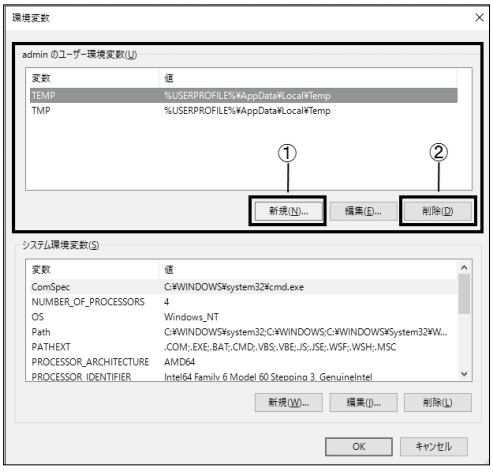
キャプチャファイル「sd1611_https_192.0.2.2.pcapng」をWiresharkで開いてください。すでに開いている場合は、念のためメイン・ツールバーにある[このファイルを再読み込み]ボタン(**図8**)をクリックして表示を更新してください。

No.11やNo.12のパケットを選択して[Packet Details]ペインにある[Secure Sockets Layer]配下をすべて展開すると、**図1**のようにデータ

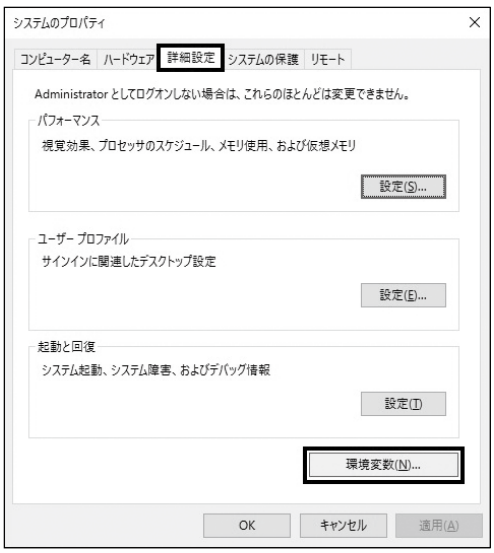
▼図10　[システムの詳細設定]リンク



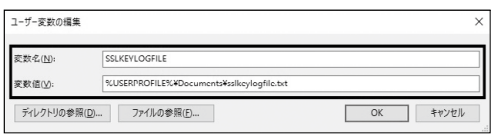
▼図12　[環境変数]ウインドウ



▼図11　[システムのプロパティ]ウインドウ



▼図13　[ユーザー変数の編集]ウインドウ



部分が「Encrypted Application Data:」と表示されていると思います。それでは、今度は(Pre)-Master-Secretの情報を含む「sd1611_sslkeylogfile.txt」を使ってWiresharkで復号してみましょう。

まずはNo.4のパケットを右クリックしてコンテキストメニューから[プロトコル設定]を選び、[Open Secure Sockets Layer preferences](**図4**)をクリックします。

[Wireshark・設定]ウインドウ(**図5**)が表示されたら、(Pre)-Master-Secret log filenameにある[Browse]ボタン(**図5の②**)をクリックして、筆者のブログからダウンロードした「sd1611_sslkeylogfile.txt」を選択して[保存]ボタンをクリックします。あとは[Wireshark・設定]ウインドウ(**図5**)の[OK]ボタンをクリックしてウインドウを閉じましょう。

Wiresharkのメインウインドウが設定を反映した表示になっていないこともあるので、今回もメイン・ツールバーにある[このファイルを再読み込み]ボタン(**図8**)をクリックして再読み込みしてください。サーバ秘密鍵を使って復号したときと同じく、**図8**のNo.11・No.12のパケットのようにHTTPSの通信が復号されて、内容が平文で丸見えになっていることがわかると思います。

このように、Wiresharkは(Pre)-Master-Secretを使うことでも、SSL/TLSの暗号化通信を復号できます。[ユーザー環境変数]に追加したSSLKEYLOGFILEは、削除しない限り残って(Pre)-Master-Secretを記録し続けてしまうので、こちらも安全性を考えて環境変数SSLKEYLOGFILEと[ドキュメント]フォルダにあるsslkeylogfile.txtは削除しましょう。環境変数SSLKEYLOGFILEを削除する方法は、先ほどの環境変数を設定する手順に従って[環境変数]ウインドウ(**図12**)を表示します。変数項目にあるSSLKEYLOGFILEを選択したら中央にある[ユーザー環境変数]の[削除]ボタン(**図12の②**)をクリックすれば削除されます。あとは[環境変数]ウインドウの[OK]ボタンをクリックしてウインドウを閉じてください。sslkeylogfile.txtが削除できない場合はGoogle ChromeやMozilla Firefoxが使用中ですので、環境変数SSLKEYLOGFILEを削除したあとにWindowsを再起動すれば削除できるようになります。

サーバ秘密鍵では復号できない場合も

最近はWebサーバに侵入されたり脆弱性を悪用されたりしてサーバ秘密鍵が漏えいしても、SSL/TLSの暗号化通信を復号できないような特性[注2]を持ったCipherSuiteを使おうという考えが広まっており、鍵交換方式にRSAではなくDHEやECDHEを使用するCipherSuiteを優先するWebブラウザが増えています。実際、DHEやECDHEを使用するCipherSuiteがSSL/TLSで使われるとサーバ秘密鍵では暗号化通信を復号できないのですが、このような場合でも(Pre)-Master-Secretを使えば暗号化通信を復号できます。

おわりに

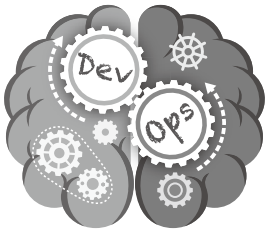
Wiresharkを使ってSSL/TLSの暗号化通信を復号する方法について説明しましたが、今回紹介した機能のほかにもSSLストリームの追跡(Follow SSL Stream)といった便利な機能があるので、興味のある方は試してください。

それと、今年の11月末にWiresharkなどを使ったパケット解析の手法をハンズオンで教える「ネットワークパケット解析コース」の開催を予定しています。開催が決まりましたら筆者のブログでお知らせしますので、ぜひこの機会にご参加ください。

本稿の内容が少しでもみなさんのお役に立てば幸いです。SD

注2) Perfect Forward Secrecyや、略してPFSと呼んだりします。

アプリエンジニアのための［インフラ］入門



Author 出川 幾夫（でがわ いくお） レバレジーズ株式会社 teratail開発チーム Twitter @ikuwow

第5回 バージョン管理入門

「現場でDevOpsを実現させるには、まずアプリエンジニアがインフラを知る必要がある」という前提に立ち、アプリの視点からインフラを広く学んでいく本連載。第5回のテーマはバージョン管理。履歴管理に留まらない、コミュニケーションツールとしてのバージョン管理ツールに注目しましょう。

“ソースコードによるコミュニケーション”の仕方によって、開発の効率は大きく変わります。これは、仕様の認識の違いに気づけず最後の段階で実装しなおしになったり、リリース後にトラブルを起こしたりすることのないよう、安全に開発を進めるためにも大切な観点です。

今回はバージョン管理のしくみをベースとして、効率よく安全な開発のフローのために必要な考え方を紹介します。

バージョン管理の必要性

バージョン管理とは、ソースコードの履歴を管理するしくみのことです。履歴を管理することによって、ソースコードを過去の一時点に戻したり、変更の過程を追従したり、複数人が同じソースコードを編集しても最終的に矛盾なく統合（マージ）させることができます。昔は、ソースコードをファイルサーバなどで共有したり、ファイル名にバージョン番号を付けて複数のファイルをやりとりしていることもありました。今では、バージョン管理をしないと怖くて開発ができないという人も多いでしょう。

ソースコードは要求に合わせて頻繁に変更が行われ、その1行1行がアプリケーションの価値を決めます。バージョン管理の履歴には、誰がいつどのファイルのどの行に変更を行ったのかという詳細な情報が含まれています。このためチーム開発でソースコードをやりとりする方法として、バージョン管理は必須のものとなっています。さらにバージョン管理のしくみの上にテスト自動化などのしくみが作られるため、DevOpsのすべてはバージョン管理から始まるとも言われています。

バージョン管理のためのツールには、Git、Subversion、Mercurialなどがあります。GitHubでオープンソースコミュニティの多くがソースコードを管理している影響で、Web開発やスマートフォンアプリ開発の世界では、Gitはすでにバージョン管理ツールのデファクトスタンダードになっているといえます。以降では、Gitを利用している前提で話を進めていきます。

すべてを再現可能にする

チーム開発でバージョン管理を行う場合、「できる限りアプリケーションのすべてを再現可能にする」という方針が重要です。

「すべて」というのは、開発するアプリケーションのソースコードだけでなく、データベースの構造に変更を加えたSQL文、定期実行するバッチ処理のためのスクリプト、APIのドキュメントなども含まれます。これらもアプリケーションを構成する要素ですので、基本的に履歴をす

べて管理すべきものと考えます。

「再現可能」というのは、ソースコードを開発マシンにダウンロード(`git clone`)したあと、ほんの少しの作業でアプリケーションの動作が確認できることを言います。

この方針は、誰かの頭の中にしかない情報をなるべく少なくしていくことにつながります。バージョン管理することによって情報が属人化せず、誰もがその情報にアクセスできるようになります。また、細かな変更やその時期や判断内容も漏らさず記録できます。そうしないと担当者がチームを離れた場合に、「この手順は誰がいつどういう理由で決めたのか」を知る方法がなくなってしまいます。

とくにデプロイの手順や、アプリケーションのセットアップに必要な作業、細かな処理を書いたスクリプトを意識的にバージョン管理するようにしましょう。話を聞いたり文字を読んだりせずに、「動かせば所望の状態になる」というのは、チーム開発におけるコミュニケーションの最高の状態です。たとえば動作環境の構築を1つのスクリプトにまとめれば、「開発環境では動いたけど本番に上げたら動かなくなった」ということが起きづらくなりますし、開発者の環境構築の手間を省けます。

Webアプリケーションの場合、ChefやAnsibleなどでサーバなどの構成管理をしておくことも重要です。これらのツールはインフラをコードとして扱うことができるため、ソースコードと同様に人にレビューしてもらったり、履歴を記録したりといった動作ができるようになります。また簡単に同じ環境を構築でき、開発の効率も上がります。

バージョン管理で気を付けること

外部のライブラリを利用する場合は注意が必要です。ライブラリは、ソースコードをそのままコピーしてリポジトリに入れ込むのではなく、必ずGitのsubmodule機能やパッケージ管理ツールを用いて、「どのライブラリに依存するかの情報」だけをバージョン管理するようにしましょう。パッケージ管理ツールの例としてRubyではbundler、PHPではComposer、Node.jsではnpmなどが挙げられます。**リスト1**のように「ライブラリのバージョンを変えただけの情報」を保持すれば十分です。

また例外として、バージョン管理すべきでない情報も一部あります。動作環境に依存する情報(ホスト名やIPアドレス)やセキュリティ上ネットワークに載せたくない秘密鍵やパスワードは、環境変数などから取得させるような実装にしましょう。

アプリケーションに必要な情報をなるべくバージョン管理しておくことで、すべての情報共有や変更の追従ができるようになります。基本的にすべての要求や仕様がコードになっていることが大切です。設定方法や作業手順、スクリプトなどをドキュメントのような形で共有するのも1つの手ですが、構成管理ツールなどを使ってテキストにすると動く手順書として、より信頼度の高い情報になります。

コミュニケーションツールとしてのGitHub

GitHubは、単にGitリポジトリのホスティングというだけでなく、開発者間のコミュニケーションツールとしての機能も多く備えています。

▼リスト1　composer.json(PHPの依存ライブラリを記述したもの)

```
{
    "name": "username/repositoryname",
    "require": {
        "league/oauth2-client": "^1.1"
    }
}
```

Pull RequestとIssue

最も特徴的なのは「Pull Request」です。これは、自分が変更を行ったブランチのレビューとマージを他者に依頼する機能です。「このような変更はどうか」と思いついたアイデアを、そのまま変更としてレビュー可能な状態にしておくこともあります。Pull Requestそのものや、変更行に個別にコメントを付けることもでき、まさに“ソースコードによるコミュニケーション”を行うための機能と言えます。

「Issue」も、コードをもとにしたコミュニケーションをするしくみの1つです。基本的にアプリケーションの問題や課題を報告する機能ですが、筆者のまわりでは、チーム開発において実装の要件を書いたり、その内容について議論するといった使い方が多いです(**図1**)。IssueやPull Request同士は**#1234**などの形で相互にリンクを張れます。オープンソースコミュニティでは、このIssueでユーザの意見やバグの報告が頻繁に行われます。

運用方針

実際にGitHubをどう使うかは、開発チームの性質やメンバーの好みによって決めていきます。teratail開発チームでは、Issueには実装することが決まった機能や修正のTODOを書き、その仕様に関して随時issue内のコメントで議論をしています。Pull Requestは、アサインされた人以外も気づいたことをコメントするようにしています。またIssueは、エンジニア以外のメンバーからもバグ報告などを書いてもらうことがあります。

GitHubでのコミュニケーションはSlackなどのチャットツールとは違って、ソースコードの変更や特定の機能の実装に関連させて議論ができます。これによってコードの過去の変更履歴を追う際に「どういう過程を経てこの変更が行われたのか」を知ることができます。GitHubのさまざまな機能を使うことでコードレビューの精度や効率の向上が期待できます。

レビューしやすいPull Requestを作ることも大切な心がけです。なるべく小さな変更や機能単位でPull Requestをすることが推奨されます。こまめに変更をマージすることで不具合が発生した際のリスクが少なく、部分的な変更に絞ってレビューを行えるからです。

9月15日のGitHub Universe[注1]では、IssueやPull Requestをかんばんのように可視化できる「Project」機能や、Pull Requestに対して「In Review」のような状態を付けられるなど、タスク管理に非常に便利な機能の発表がありました。GitHubは今後もしばらく、ソフトウェア開発のデファクトスタンダードとあり続けるでしょう。

▼図1　筆者が所属するteratail開発チームのとあるIssue

早く安全にデプロイをするフロー

バージョン管理のしくみをコミュニケーションのベースとすると、これをもとに開発フローを組むことができます。Gitの場合は、どのようにブランチを切ってどういう役割を持たせ、どういう流れでコードを本番にリリースするかを決めたものが開発フローとなります。Gitを用いた開発フローには有名なものが何種類かあります。

注1)「A whole new GitHub Universe: announcing new tools, forums, and features」
URL https://github.com/blog/2256-a-whole-new-github-universe-announcing-new-tools-forums-and-features

Git flow

Git flow[注2]は、masterブランチのほかにdevelop、hotfix、releaseなどのブランチを用意して開発を行うフローです。

開発者はdevelopブランチからfeatureブランチを切り出し、開発が終了したらまたdevelopブランチにマージします。リリースを行うときは、リリースの準備を行うためのreleaseブランチにdevelopブランチをマージして検証を行います。その後masterブランチをreleaseブランチにマージし、タグを付けます。masterブランチは常にリリース可能な状態にしておくというルールを守りつつ、複数人でも大規模な開発を可能にしたのがこのフローといえます。このフローで開発を行うための「git-flow」というツールも存在します。

GitHub flow

GitHub flow[注3]はGit-flowのアンチテーゼとして提唱されたフローで、ブランチはmasterとfeatureの2種類のみです。リポジトリはforkせず、masterから切り出したブランチに開発を行ってmasterにマージします。masterブランチにマージされたら直ちにリリースを行います。シンプルでわかりやすく、頻繁にリリースをするプロジェクトに向いています。

GitLab flow

またGitLab flow[注4]というフローもあります。これはGitHub flowにproductionブランチを1つ追加したような形になっていて、シンプルさを保ちつつリリースタイミングをコントロールしやすいフローです。teratail開発チームではこのGitLab flowを少しカスタマイズしたフローを利用しています(**図2**)。

注2)「A successful Git branching model」
URL http://nvie.com/posts/a-successful-git-branching-model/

注3)「Understanding the GitHub Flow · GitHub Guides」
URL https://guides.github.com/introduction/flow/

注4)「GitLab Flow | GitLab」
URL https://about.gitlab.com/2014/09/29/gitlab-flow/

これらフローの中には、テストやデプロイの自動化を含めることができます。Travis CIやCircleCIなどのCIツールと連携させることで、すべてのPull Requestがpushされるたびに指定のテストコマンドが走り、テストが通ったときだけマージができる、というように変更の信頼性をある程度担保できます。

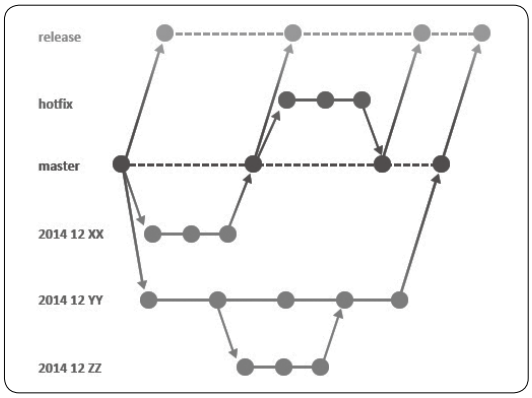
Webアプリケーションでは、Pull Requestをマージした瞬間から本番環境へのデプロイが全自動で行われるようなしくみが理想です。これにより、作ったものを素早くユーザに届けられます。

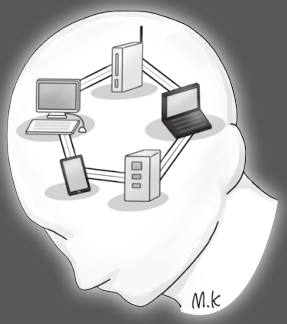
テストやデプロイが自動化されると、信頼性の高いリリースが頻繁に行えるようになり、インフラの知識がないデザイナーやディレクターもリリースを行えるようになるなど、非常に柔軟な開発が可能になります。

◆ ◆ ◆

今回はバージョン管理のしくみをベースにした開発の流れにおける考え方を紹介しました。バージョン管理はDevOpsに関わるすべての土台です。単なる履歴の管理以上に、コミュニケーション方法の1つとして非常に重要です。また開発のフローはこれがベストというものはなく、アプリケーションの性質やチームメンバーの好みで大きく変わってきます。たまには普段の開発から一歩引いた視点でそのフローを見直してみて、最も開発のパフォーマンスが上がる方法を模索してみましょう。SD

▼図2 teratail開発チームのGitLab flow(https://teratail.com/blog/article/ba2)





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)
有限会社 ネットワーク・メンター
URL http://www.network-mentor.com/indexj.html

第6回 仮想マシンの管理

これまでの連載では、デフォルトの「NAT接続/単独仮想マシン/仮想マシン着信」、仮想マシンマネージャー処理により、仮想マシンの仮想環境導入から仮想マシン/ゲストシステムの構築設定・利用・運用管理まで、管理者、そして利用者として作業を行いました。

今回は、本連載前半「まずは単一システムで仮想マシンを使ってみよう」というテーマの締めくくりとして、今までに構築し利用してみたデフォルト最低限の仮想環境の運用管理全般について、仮想マシンを中心に問題点を見てみます。着信接続、固定IPアドレス、かなり遅いWindows仮想マシンのディスクI/O、そのほか個々の仮想マシンの詳細設定などです。

仮想マシン宛の接続

これまでの連載では仮想マシンからの発信接続で、仮想マシン宛の接続はしていません。しかし、仮想マシンからの発信接続だけでは仮想マシンがクライアントとして利用することはできても、仮想マシンがネットワークサーバとなるような使い方はできません。

仮想マシンをネットワーク・サーバとして使うためにはどうしても、仮想マシンへの着信接続を可能にするような設定が必要になります。

物理ネットワーク側から仮想ネットワーク側へパケット転送するために、第1に物理ホストで、ファイアウォール通過(転送)許可をすること、第2に物理ホストをそのゲートウェイとして物理ネットワーク側で認識させること、の2つです。

ファイアウォールの転送許可設定

KVM物理ホストのファイアウォールiptablesに、次のような「仮想マシンのIPアドレス宛パケットを通過させる設定」をすると仮想マシン宛の接続が可能になります。

```
iptables -I FORWARD -d 192.168.122.0/24 ➥
-j ACCEPT
```

ゲートウェイ認識設定

これには2つの方法があります。

1つは、物理LAN側のどんなシステムからも、仮想マシン宛パケットはKVM物理ホストへ渡す設定です。これは、物理ホストLAN側のメインのルータで「LAN側の静的ルーティング」設定で仮想ネットワーク「192.168.122.0/24」のゲートウェイを192.168.0.111(KVM物理ホストのIPアドレス=KVMルータ)にします。

もう1つは、仮想マシンにアクセスする物理LAN側のシステム個々のネットワーク設定のデフォルト、または追加のゲートウェイに192.168.0.111を設定します(「メトリック」を2)。設定は[ローカルエリア接続]→[全般]→[インターネットプロトコル(TCP/IPv4)]→「全般」の

デフォルトゲートウェイ、または[詳細設定]→[デフォルトゲートウェイ]に追加します。

Windows 7ファイル共有

Windows 7ファイル共有可能設定は一般の物理PCと同様です(ファイルウォール：[受信の規則]→[該当プロファイル(ドメイン／プライベート／パブリック)]→[ファイルとプリンターの共有(NBセッション受信／SMB受信／エコー要求-ICMPv4受信)]→[スコープ]→[リモートIPアドレス]に「192.168.0.0/24」を追加)。

また、[ネットワークと共有センター]→[共有]の詳細設定をします。

仮想マシンでの固定IPアドレスの利用設定

自動IPアドレス取得から固定IPアドレスへ変更する場合は、次の設定変更を行い再起動します。

- Windows 7：[ローカルエリア接続]→[インターネットプロトコルバージョン4(TCP/IPv4)]→[全般]で、IPアドレスとDNSアドレスを設定
- FreeBSD 10.3：Ifconfig_em0="DHCP"(em0 は NIC ID)→ ifconfig_em0="inet IPアドレス netmask 255.255.255.0"

この連載ではさまざまな環境やトラブルシューティングなどを考え、次回からの後半連載では固定IPアドレスで進めます。

Windows準仮想化デバイスドライバ

WindowsではKVM仮想マシンデバイスドライバのうち、とくにディスクのI/O速度が物理システムに比べてかなり遅いので、実環境では速くしたくなります。準仮想化(virtio)ドライバを使うと、スループットを実システムのように上げることができます。

Windowsのデフォルトでは、IDEデバイスドライバ(Windowsのデバイスマネージャでは「QEMU HARDDISK ATA Device」と表示)が使用されますが、virtioドライバとして、Virtio BlockドライバとVirtio SCSIドライバの2つが使用可能です。

Virtio SCSIドライバは、SCSIチャネルに直接接続し、Virtio Blockドライバ(28デバイスしか処理できず、PCIスロットを使い果たす)よりスケーラビリティ(数百デバイスを処理可能)が格段に上がっています。

virtioドライバのインストール手順は記事末の囲み記事を参照してください。なお、現在Windows 7のvirtioディスクはWindows 10へ更新インストール後使用できません。新規インストール後のvirtioドライバ追加は可能です。

さまざまな仮想マシンサポート

CD/DVD-ROMの利用

仮想マシンでのCD/DVDは、仮想マシンの詳細プロパティでIDE CD-ROMの「メディアを選択」でCD/DVDを指定しておいて使用します。もし、仮想マシンで認識しないときはプロパティ(仮想マシン／詳細)のCD-ROMタブの仮想ディスクの欄の右端の「接続」をクリックします。

仮想環境の運用管理制御

仮想マシン環境に関する仮想マシン接続の詳細設定、仮想マシンとの間の全般的な仮想マシンの全般環境設定、そして仮想マシン個々の詳細設定の3種類です。

仮想マシン接続環境設定

仮想マシンマネージャーの[編集]→[接続の詳細]から設定する「localhost Connection Details」です。仮想ネットワークやストレージ、NICの作成・変更を行います(**図1**)。

NICの設定では、仮想ネットワークでの

DHCP／固定IPアドレス利用やブリッジ／VLAN／チャネルボンディングなどの選択設定が可能です(詳細は連載後半)。

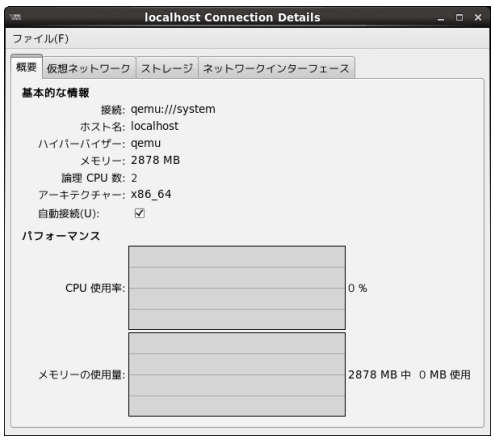
仮想マシン全般環境設定

仮想マシンマネージャーの[編集]→[設定]から設定します。仮想マシンとの間の一般的な設定や統計などです(**図2**。詳細は連載後半)。

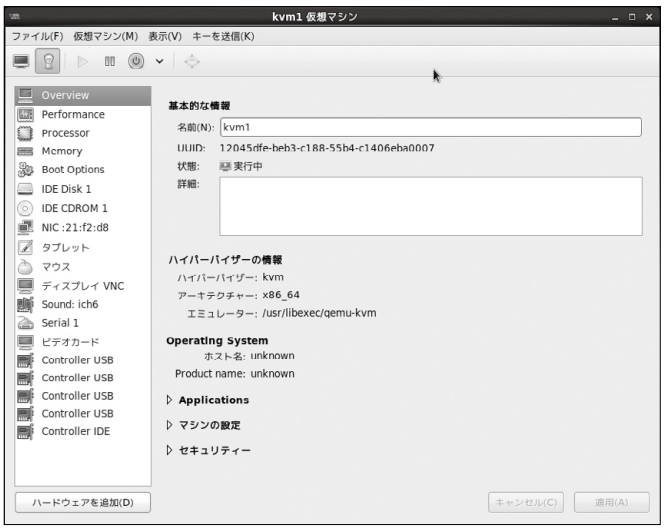
仮想マシン個々の詳細設定の概要

個々の「物理的」設定は、仮想マシンメニュー[表示]→[詳細](**図3**)の左ペインで行います。

▼図1　仮想マシン接続の詳細設定



▼図3　仮想マシン個々の物理的な設定開始画面



ACPI／APIC／時刻設定

「Overview→マシンの設定」でAdvanced Configuration and Power Interface(電源制御)やAdvanced Programmable Interrupt Controller(プログラム割り込み制御)の有効／無効設定、そして、時刻(ローカルとUTC：Coordinated Universal Time協定世界時)選択が可能です(**図4**)。

プロセッサ

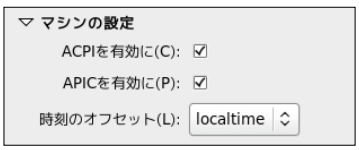
「Processor」で仮想マシンに割り当てる論理ホストCPUの設定などを行います(**図5**)。

メモリ割り当て

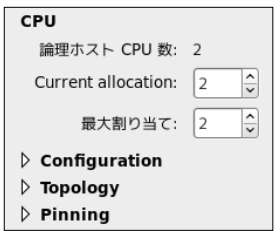
「Memory」で仮想マシンに割り当てるメモリ

▼図2　仮想マシンの全般環境設定

▼図4　ACPI／APIC／時刻設定



▼図5　プロセッサ設定



サイズを設定します(**図6**)。

起動設定

「Boot Options」で(物理ホスト起動時の)仮想マシンの自動起動や起動デバイス順序の設定を行います(**図7**)。

IDEディスク

「IDE Disk 1」でハードディスクの読み込み専用や共有、およびディスクバス(IDEまたはVirtio)などの設定を行います(**図8**)。

IDE CD-ROM

「IDE CDROM 1」で「仮想ディスク」の[接続]から「メディアを選択」でデバイスやISOイメージのパスを設定します(**図9**)。

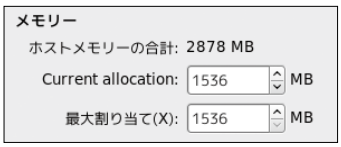
仮想NIC設定

「NIC」で仮想環境のNICデバイスを選択します。あらかじめ仮想マシン接続環境設定(「localhost Connection Details」)で設定しておく必要があります(**図10**)。

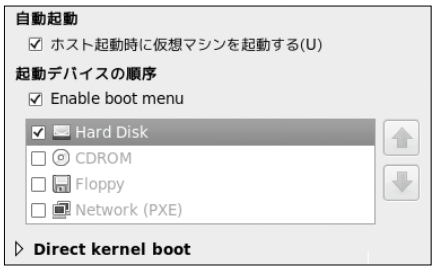
ディスプレイVNC設定

仮想マシンマネージャーからハイパーバイザへのビュアーとして仮想マシンのコンソールを開きますが、VNC(tigervnc)を使用するとVNCビュアー仮想マシンを開くことができます(**図11**)。つまりこれを使うと、リモートから直接仮想マシンコンソールを開くことが可能になり、運用管理の幅を広げることが可能になります(ハイパーバイザビュアーとVNCビュアーは排他使用)。この詳細は次回以降の連載後半で解説します。

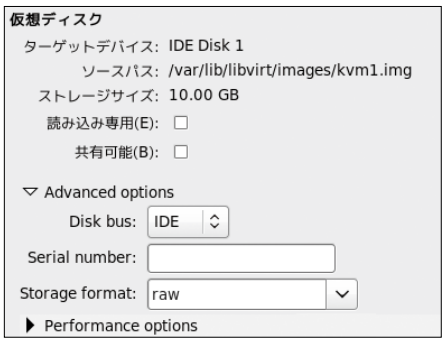
▼図6　メモリ割り当て設定



▼図7　起動設定



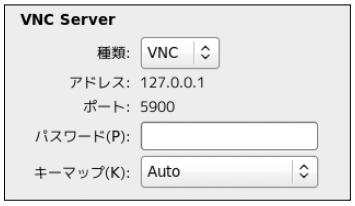
▼図8　IDEディスク設定



▼図9　CD-ROM設定

▼図10　仮想NIC設定

▼図11　ディスプレイVNC設定



仮想マシンの管理者と利用者

集中管理やコスト削減を目的とした仮想化の考え方にしたがえば、また、仮想マシンのOSやアプリケーションを含めたシステムについて利用者組織／部署が「もっとも詳しい」ことを考えたならば、個々の仮想マシンのすべての運用管理については利用者組織／部署(の技術者)が担当することが技術的には合理的に思えます。

小組織／部署、小仮想ネットワークであればなおさらですが、大規模仮想ネットワークでも、その運用管理をサポートできるだけの管理者を置くことも現実的には難しくなります。とくに、仮想マシンのプラットホームのOSやインフラが多種多彩になると、それだけの技能を1人で持つ管理者を集めることはさらに難しくなります。つまり、それらに詳しい利用者組織／部門(や技術者)に運用管理まで任せることが管理コスト的にも合理的です。

こうした「利用者組織／部門」自身による仮想マシンの運用管理については、次回以降の本連載後半の中で詳しく議論していくことにします。

次回予告

次回からは仮想ネットワークを現実的な使い方で考える、連載後半「仮想ネットワーク環境で使ってみよう」を始めます。次回テーマは「ホストシステムと仮想環境の構築」です。SD

連載各回では、読者皆さんからの簡単な「ひとくち質問(QA)コーナー」や「何かこんなこともしてほしい要望(トライリクエスト)コーナー」を設けて「双方向連載」にできればと思っていますので、質問や要望をお寄せください。

宛先：sd@gihyo.co.jp
件名に、[仮想化連載]とつけてください

●Windows準仮想化(virtio)ドライバのインストール手順

※virtioドライバisoファイル名：virtio-win-0.1.102.iso
(2016年9月12日現在)

【手順1】次のどちらかの方法でvirtioドライバをダウンロードし、KVMホストの適当な場所に格納する。

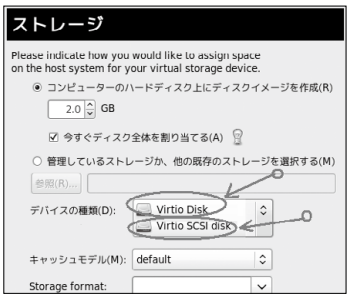
(1-1)ホストの適当な場所に直接ダウンロード。
https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/stable-virtio/virtio-win.iso

(1-2)repoをwgetした後に、yumインストールを行う。

```
①# wget https://fedorapeople.org/groups/virt/virtio-win/virtio-win.repo -O /etc/yum.repos.d/virtio-win.repo
②# yum install virtio-win
```

【手順2】仮想マシン環境設定[注1]でストレージ(Storage)にvirtioデバイス(Virtio Disk)、またはvirtio-SCSIデバイス(Virtio SCSI disk)を追加する(**図A**)。

▼図A



【手順3】仮想マシン(Windows 7)起動後、仮想マシン設定でvirtioドライバisoファイルを接続する(IDE CDROM1→仮想ディスク→[接続]→メディアを選択。**図B**)。

▼図B

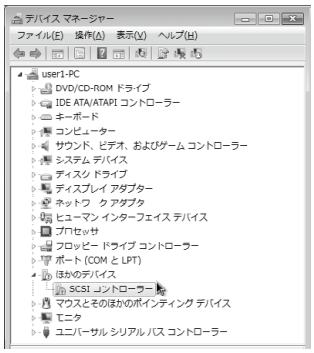
【手順4】仮想マシンWindows 7のデスクトップ上にCD-ROMマウントメッセージが表示される(**図C**)ので、virtioドライバをインストールする。

▼図C

注1) 仮想マシン個々の詳細設定：仮想マシン表示時、[表示]→[詳細]から「ハードウェアを追加」。

(4-1)[コントロールパネル]→[システム]→[デバイスマネージャー]を開くと、「ほかのデバイス」に黄色！三角の「SCSIコントローラー」が表示されている(**図D**)。

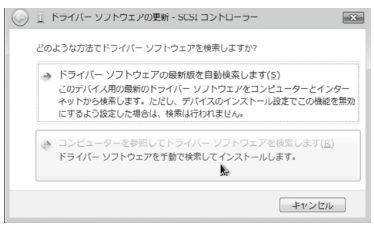
▼図D



(4-2)このSCSIコントローラを右クリックして「ドライバーソフトウェアの更新」をクリック。

(4-3)「ドライバーソフトウェアの更新－SCSIコントローラー」画面の「どのような方法でドライバーソフトウェアを検索しますか？」で下段の「コンピューターを参照してドライバーソフトウェアを検索します(R)ドライバーソフトウェアを手動で検索してインストールします。」を選択クリック(**図E**)。

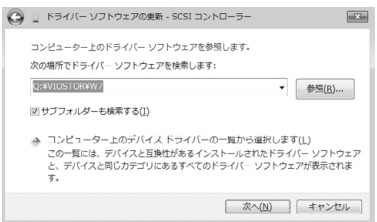
▼図E



(4-4)「コンピューター上のドライバーソフトウェアを参照します。」画面で「次の場所でドライバーソフトウェアを検索します」の枠内に、virtioドライバの種類に応じて次のパスを設定(**図F**)。

・Virtio Block driver＝Q:¥VIOSTOR¥W7
・Virtio SCSI driver＝Q:¥VIOSCSI¥W7

▼図F



(4-5)「次へ」をクリックするとインストールが開始され、正常に完了すると「ドライバーソフトウェアが正常に更新されました。」というメッセージが表示される(**図G**)。

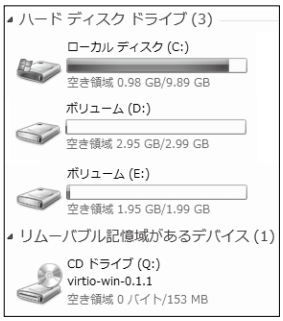
▼図G

(4-6)「閉じる」でデバイスマネージャーの「ディスクドライブ」と「記憶域コントローラー」に**図H**のような表示[注2]が現れる。ここでは、Virtioブロックドライバディスクと Virtio SCSIドライバディスクの2つをインストール完了している。

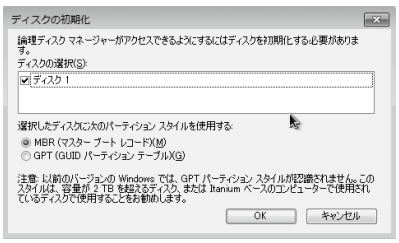
▼図H

(4-7)[管理ツール]→[コンピュータの管理]→[ディスクの管理]画面で「ディスクの初期化」メッセージ(**図I**)が表示されるので、そのまま「OK」で初期化した後、通常のフォーマットを行って使用可能になる(**図J**)。

▼図J



▼図I



注2) ●ディスクドライブ：QEMU HARDDISK ATA Device＝KVMデフォルトIDEディスク／QEMU QEMU HARDDISK SCSI Disk Device＝KVM Virtioブロックドライバディスク／Red Hat VirtIO SCSI Disk Device＝Virtio SCSIドライバディスク
●記憶域コントローラー：Red Hat VirtIO SCSI controller＝Virtio SCSIコントローラ

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9回 APIファースト・メソッドが可能にする「DB分離」の組織体制

アプリからDBを扱うためのAPIをストアドプロシージャで定義する方法「APIファースト・メソッド」(本連載第6回で紹介)。これを使えば、アプリとDBの担当者をうまく分担できます。同様のことを実現する方法として「O/Rマッパ」もあります。今回はこの両者の違いを見つつ、Webアプリ開発の組織体形について考えます。

登場人物紹介

生島氏
DBコンサルタント。性能トラブルの応援のため大道君の会社に来た。

大道君
浪速システムズの新米エンジニア。素直さとヤル気が取り柄。

五代氏
大道君の上司。プロジェクトリーダーでもある。

APIファースト・メソッドのしくみとメリット

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。このところ、取引先の浪速システムズが開発しているお料理レシピ投稿サイトに技術アドバイザーとしてかかわっています。

「おかげさまでこのところ順調に進んでますわ」と言ってくれたのは浪速システムズプロジェクトリーダーの五代さん。

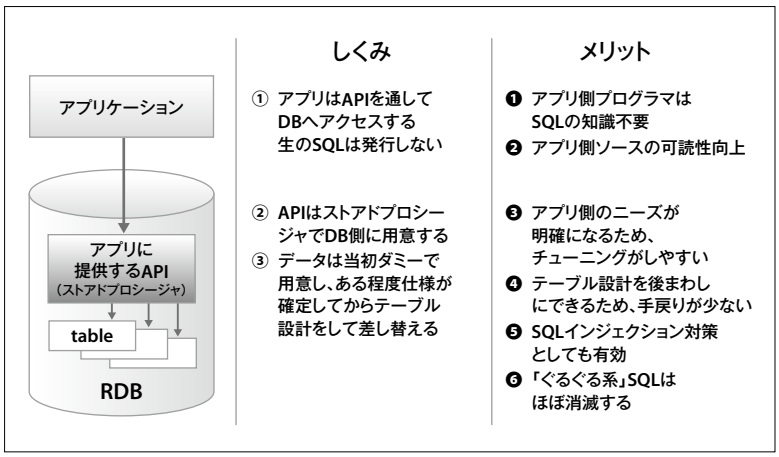
「生島さんが提案してくれたAPIファースト・メソッドのおかげで、開発効率も非常にいいですわ」

「それも大道君の飲み込みの早さあってこそですし、五代さんがこの方針で行こう！ とOK出してくれたからですよ」

APIファースト・メソッドというのはこのプロジェクトで採用した方法で、この連載では第6回[注1]で触れています。要点は**図1**のような方法で、データベース(以下、DB)からアプリケーション(以下、アプリまたはAP)側に提供するAPIをストアドプロシージャで定義してやり、アプリからのDBアクセスは必ずそのAPIを通して行うようにします。それによって、メリット❶〜❻のような効果が得られます。

「しかもこれ、別に勉

▼図1 APIファースト・メソッドによる開発のしくみ



注1) 本誌2016年8月号。

強に時間がかかる複雑なツールを使うわけでもなく、Excelでストアドプロシージャのスタブを自動生成するだけじゃないですか。こんなやり方があったんだ、ってまるで目から鱗な感じですけど、どうしてこういう方法を考えついたんですか？」

「原点として持っていた問題意識は、DBをきちんとわかって使っているシステムエンジニア（SE）、プログラマがあまりにも少ない、ということですね」

「そこはこの半年で本当に痛感しています……」と大道君。彼はこのところ既存システムのトラブルシューティングにもよく駆り出されていて、そのたびにあまりにも下手なSQLを見つけては唖然としているそうです。ベテランのSEでも意外にRDBとSQLのことは理解していないというのが現実で、それは私自身がこの20年もどかしく感じていることです。

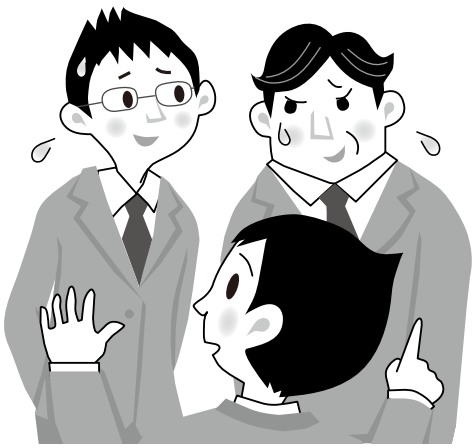
「でも、どうしてそんなことになっちゃってるんでしょう？」と再び大道君。まだIT業界3年目と、若いからこそ抱く素朴な疑問だと思います。五代さんと私は思わず目を見合わせて苦笑いしました。その答えは、業界経験の長い私たちには共通の認識がありました。

プログラマは交換可能な部品扱いだった

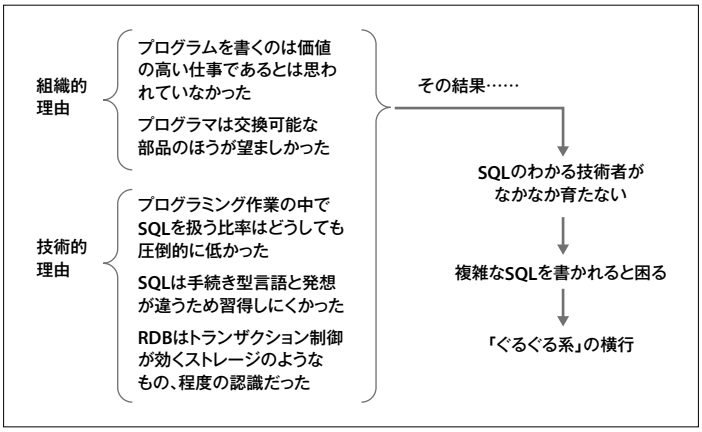
簡単にまとめると**図2**のようになります。

日本のIT業界では、プログラムを書くことを価値の高い仕事と見なさず、プログラマは交換可能な部品であるほうが望ましい、と考えるマネジメントが蔓延していました。「交換可能な部品」と思っているからこそ、「1人1月いくら」の人月商売、技術者派遣ビジネスが成立していたわけです。最近はあまり聞かなくなりましたが、昔は「プログラマ35歳定年説」などとも言われていました。「いつまでもプログラマなんぞやってんじゃねえよ」と蔑むようなマネジャーの下で、素直な若者が技術習得に情熱を燃やすはずもありません。

一方、ただでさえ技術習得が重視されないのにプログラミング作業の中でSQLを扱う比率はどうしても圧倒的に低く、その分勉強にあまり時間をかけられず、しかも手続き型言語と同じ発想では習得しにくいため理解が進まず、結果として「RDBはトランザクション制御が効くストレージのようなもの」程度の認識にとどまる技術者が多かったのです。もちろん、トランザクション制御は一般にACID特性とも呼ばれる「一貫性を保ったデータの更新を保証する」非常に重要なしくみです。しかしこの連載でこれまで書



▼図2　ベテランエンジニアでも意外にSQLをよく知らないのはなぜ？



いてきたように、RDBへの問い合わせ言語としてのSQLの価値の真骨頂は、データを集合的に扱うことを可能にし、複雑なデータ処理を劇的に簡単にできることにあります。そしてこれこそが手続き型言語と違う発想を必要とする部分のため、なかなか理解されていません。

以上のような組織的理由と技術的理由が重なってSQLのわかる技術者がなかなか育たず、たまにいても「複雑なSQLを書かれるとほかの人が読めないから困る」と規制されてしまうためますます技術向上の機会を失い、その結果「ぐるぐる系」と呼ばれるような下手なSQLが横行してしまうわけです。

「というわけや。わかった？」

と五代さんに聞かされた大道君ですが、衝撃のあまり感想の言葉も出てこない様子。

「う、嘘でしょ……と思いたいです」

「まあ日本のIT業界の黒歴史やな、これは。もちろんウチらがこれの真似をする理由はないんで、大道君は本物のプロフェッショナルになってや」

「もちろんそのつもりですよ！　でも……その組織的理由と技術的理由を解決することはできるんでしょうか？」

「ああ、そのための策の1つが、今やってるAPIファースト・メソッドなんよ」

DB担当とAP担当は分けたほうがいい

組織的理由については経営者層の考え方の問題であり、経営者に考えを変えてもらうしかありません。それができない経営者のいる会社は潰れる、というかなり荒っぽい形で変わっていくことでしょう。

「ウチは変わりまっせ〜、潰さへんで〜」と五代さん。はい、そのために私も協力してますから。

一方の技術的理由については、エンジニア側の動き方しだいで解決できます。そのために重要なのが、「DB担当とUI担当を分ける」ということです。

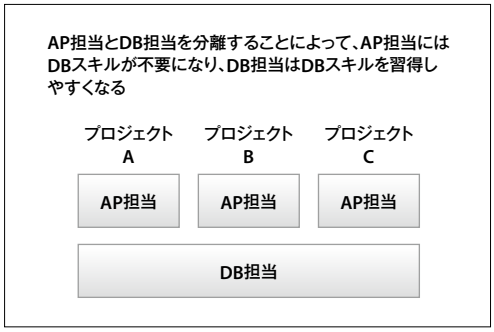
「ああ、そうか！　APIファーストだとそれができるんですね？」と大道君。そのとおり！

もう一度**図1**を見てみましょう。アプリからはAPIを通してDBへアクセスします。生のSQLは発行しません。そのため、「AP側プログラマにはSQLの知識は不要」です。その分、ストアドプロシージャを作るDB担当のほうにその仕事を分担させます。

「でも、DB担当ができる人なんてそう多くないですよね？」

「そこで、DB担当は複数のプロジェクトを受け持つ体制を組むわけ。どうしてもコードの量はAP側のほうが多いし、ユーザとの仕様調整に伴う細かい修正も多く発生するから、AP担当は1プロジェクト専任にする。DB側はストアドプロシージャを書く必要があるとは言ってもAPに比べるとプログラミング負担は少ないから、DB担当のほうは複数のプロジェクトのDB側をまとめて受け持つ。こうすることで、DB担当はDBに集中でき、短時間でSQLスキルを向上させられるわけだ」(**図3**)

▼図3　「APIファースト・メソッド」を可能にする組織体制



「あ、なんだ、それってまさに今やってることじゃないですか！」

と大道君。そうなんです。実はお料理レシピ投稿サイトはまさにこんな体制で開発しています。誌面の都合上登場していませんが、画面の細かいところを作るAP担当プログラマが別にいて、大道君がDB担当、ただしまだ心許ないので私が技術アドバイザーについている、という体制

です。このしくみならDB関係の性能問題は大幅に予防できます。しかもAPのコードが単純化して開発工数が減りますし、DBスキルがいらなくなるので、新人エンジニアはまずAP担当にアサインして現場経験を積んでもらうことが可能。

「いいことづくめに思えますけど、でもそれじゃなぜそういう体制の現場が少ないんでしょうか？」

「技術リーダークラスがSQLというものの価値をわかっていないと、こういう発想にはならないんだよね。DBなんてちょっと賢いファイルシステムでしょ、程度に思っていたら、データをファイルに書き出す部分だけ独立して担当させよう、なんて考えないでしょ？」

「もうひとつ、下請け型のSIビジネスの構造だとこれはやりにくかったんですわ。求人が『DBのプロ求む』じゃなくて、『APもDBもそこそこ書ける奴をとにかく頭数そろえて出してくれ』というような注文がくるんで、AP担当とDB担当を分けづらい……」と五代さん。

一言で言えば、IT業界に求められていたのは人足、いわゆるIT土方であって、プロフェッショナルではなかった……という悲しい実態です。とはいえ、それは過去の話。現代もそのままでは通用しません。

O/Rマッパでは問題は解消しない?

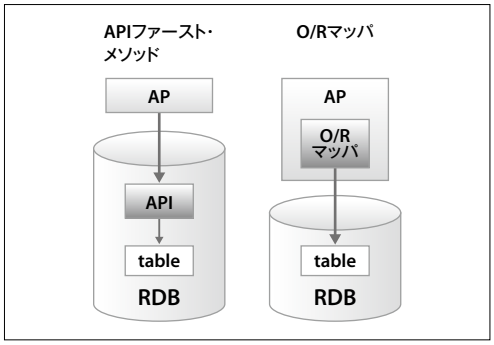
しかし、SQLをきちんと使おうとすると、「SQLのわかる技術者がなかなか育たない」という問題に加えてもうひとつ、「アプリ側のプログラム言語ではSQLを操作しづらい」という、一般にはインピーダンス・ミスマッチと言われる問題もあります。この両者を解決できるのではないか、と期待され非常に広く普及したのが、思いっきり単純にいうと「AP用言語によるDBアクセス操作ライブラリ」といった性格を持つO/Rマッパです。簡単なイメージを描くと**図4**で、APとRDBのインピーダンス・ミスマッチを、APIファースト・メソッドではDB側にAPIを用意して吸収し、O/RマッパではAP側のO/Rマッパライブラリ（SQL文生成機能を持つ）で吸収するわけです。

しかし、O/Rマッパでは本質的な問題は解決しません。というのは、O/Rマッパはあくまでも「SQL文生成機能」であり、プログラミング中にSQLを意識しなければいけない場面が残るからです。

例として、PHP用のO/Rマッパ、Doctrineを使ったDBアクセスコード例（**リスト1**）をご覧ください。これはECサイトの管理画面で当日、前日、今月の売上と受注件数を集計表示するためのコードの概要部分です。

第1のポイントは、getOrdersByDay()関数

▼図4　APIファースト・メソッドとO/Rマッパ



の中でSUM(…) as totalやandWhere(…)のように部分部分でSQL文の断片が残ることです。つまり、結局のところ生成されるSQL文をAP側プログラマがイメージしながら使わなければなりません。さらに第2のポイントは、当日、前日、今月とそれぞれ期間を変えて3回getOrdersByDay()関数を呼んでいることです。プログラムの部品化を重んじる手続き型言語であれば当然の発想ではありますが、これでは同じ受注テーブルを3回スキャンすることになるため性能的には悪影響を及ぼします。

SQLの考え方であればこれは一度で処理すべ

▼リスト1　O/Rマッパ（Doctrine）を使ったときのPHPコード例

```
// 当日、前日、今月の売上/件数をそれぞれ取得
$salesToday = $this->getOrdersByDay(当日, 当日);
$salesYesterday = $this->getOrdersByDay(前日, 前日);
$salesThisMonth = $this->getOrdersByDay(月初, 月末);

// DB操作関数
protected function getOrdersByDay($date_start, $date_end)
{
    // ------ 前略 ------
        $repository = $this->getDoctrine()
            ->getRepository('Entity¥Orders'); // リポジトリは省略

        $qb = $repository->createQueryBuilder('o')
        ->select('SUM(o.payment_total) as total, 'COUNT(o.id) as order_count')
        ->andWhere('o.order_date >= :order_date_start')
        ->andWhere('o.order_date <= :order_date_end')
        ->setParameter(':order_date_start', $date_start)
        ->setParameter(':order_date_end', $date_end);

        // ------ 中略 ------

        $result = array();
        try {
            $result = $qd->getSingleResult();
        } catch (NoResultException $e) {
            // 結果がない場合は空の配列を返す.
        }
        return $result;
    }
```

▼リスト2　SQLのコード例

```
// 当日、前日、今月の売上/件数を一度で取得する
SELECT
      SUM(CASE WHEN o.order_date = :today THEN payment_total ELSE 0 END)
    AS today_amount
    , COUNT(CASE WHEN o.order_date = :today THEN o.id ELSE null END)
    AS today_count
    , SUM(CASE WHEN o.order_date = :yesterday THEN payment_total ELSE 0 END)
    AS yesterday_amount
    , COUNT(CASE WHEN o.order_date = :yesterday THEN o.id ELSE null END)
    AS yesterday_count
    , SUM(CASE WHEN o.order_date BETWEEN :month_start AND :month_end THEN payment_total ELSE 0 END)
    AS month_amount
    , COUNT(CASE WHEN o.order_date BETWEEN :month_start AND :month_end THEN o.id ELSE null END)
    AS month_count
  FROM
    Entity¥Orders o
  WHERE
    o.order_date BETWEEN :month_start AND :month_end
    OR o.order_date = :yesterday';
```

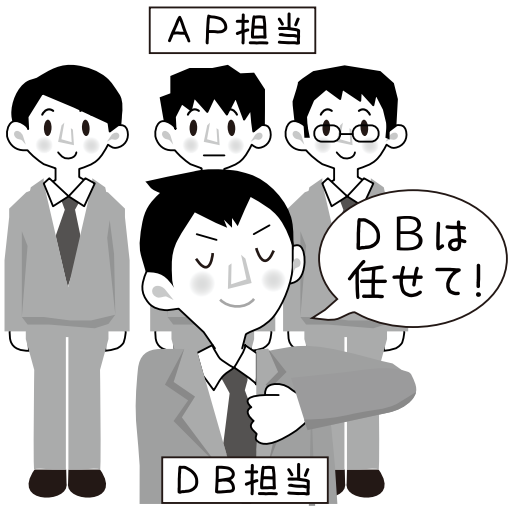
きであり、コードとしては**リスト2**のようになります。SQLの発想が身についていればとくに難しいこともない処理ですが、このSQL文をPHPのコードの中に埋め込むのは可読性を落としますし、AP側プログラマにSQLの理解を強いるため好ましくありません。

そこで、APIファースト・メソッドではストアドプロシージャの中にSQL文を隠蔽(いんぺい)します。PHP側のコードは**リスト3**のようになります。指定するパラメータは日付だけであり、SQL文の断片はまったくありません。

結局のところ、RDBとSQLは「集合的データ操作に向く」ように作られたシステムであり、それを徹底的に活用するならAPIファースト・メソッドのほうが適しています。

このような部分を適切なSQLに変える修正をすると、平均500ミリ秒のレスポンスタイムシステムを平均100ミリ秒にできたりします。これは1ユーザの体感速度としては問題にならない差ですが、AWSなど従量課金のサービスを利用して一般公開するようなシステムの場合、レスポンスタイムが延びるとその分同時アクセス数も増えるため、ランニングコストが数倍違ってくることもあります。

簡単なDB操作しか使わず、性能要求もゆるいプロジェクトならO/Rマッパで間に合うと思いますが、そうでない場合はAPIファースト・メソッドでDB担当も分ける組織体制での開発をお勧めします。SD



▼リスト3　APIファースト・メソッドを使ったときのPHPコード例

```
// 当日、前日、今月の売上/件数を一度で取得する
 protected function getOrdersSummary($today, $yesterday, $month_start, $month_end)
    {
        $em = $this->getDoctrine()->getEntityManager();

        $dql =
        'CALL pr_getOrdersSummary(:today, :yesterday, :month_start, :month_end)';

        $q = $em
            ->createQuery($dql)
            ->setParameter(':today', $today)
            ->setParameter(':yesterday', $yesterday)
            ->setParameter(':month_start', $month_start)
            ->setParameter(':month_end', $month_end);

        $result = array();
        try {
            $result = $q->getSingleResult();
        } catch (NoResultException $e) {
            // 結果がない場合は空の配列を返す.
        }
        return $result;
    }
```

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11回 Android 7.0のセキュリティ

谷口 岳（たにぐち がく）
リスクファインダー（株）

Androidは世界で出荷される8割のスマートフォンに搭載※される、事実上スマートフォンの標準OSです。このOS上で動くアプリケーションを開発することは、自分のアイデアを世界に発信し、最も多くのスマートフォン上で動かしてもらえる可能性がある行為です。このAndroidの最新情報と、それを取り巻くコミュニティを知って、魅力的な開発をはじめませんか？

※IDC Worldwide Mobile Phone Tracker, August 7, 2013

はじめに

Androidの新しいOS、Nougat（Android 7.0 API Level 24）がリリースされました。ヌガーと読みますが、聞きなれないですよね。マシュマロに続きスペルミスが非常に心配です。もう少し一般的な名前にしてもらいたいものです。

さてAndroid N（Nougat）はすでに公式アップデートが開始され、筆者の手元のNexus 5Xにも公式アップデートが届きました。また、7.0が初期インストールされたモデルもLGから発売されました。

しかし10月4日にGoogleから、Android 7.1を載せた新端末がリリースされる可能性が非常に高いです。しかもバージョン番号は0.1しか上がらないのに、結構変更があるとか……新しい情報をキャッチアップしていくのもたいへんです（残念ながら本稿を書いている段階ではわかりませんが、本誌が発売されるころには明らかになっていると思います）。

さて1年前のOS、Android 6.0では、Runtime Permission機能が導入されました。これによりAndroidのセキュリティレベルは上がりましたが、一方ほぼすべてのアプリに影響を及ぼす機能追加でした。変更が必要になり忙しくなった方、対応するのが大変なのでアプリケーションの公開自体をやめてしまった方、あるいは聞かなかったことにして放置している方、さまざまでしょうが、非常にインパクトがあった仕様変更でした。

安心してください。今回のAndroid 7.0のバージョンアップでは、セキュリティ的には、すべてのアプリケーションに影響を与えるような大きな変更は加わりませんでした。筆者の感覚としては、いぶし銀的な変更が多かったという印象です。

アプリ開発者に直接関係のないOS自体の機能変更、よりセキュアなアプリケーションを作るための変更など、さまざまな機能追加がありますが、紙幅の都合上すべてを取り上げることはできません。今回は、通信まわりの拡張であるNetwork Security Configについて解説します。

Network Security Configとは

Network Security Configとは、その名のとおり、ネットワークのセキュリティ設定のことです。ネットワークのセキュリティに関する事項を、ソースコードとは別に設定ファイルを使ってコントロールすることができるようになりました。

AndroidManifest.xmlに**リスト1**のように、ネットワークセキュリティ設定ファイルの場所を指定します。この記載では、network_

▼リスト1　AndroidManifest.xmlでのネットワークセキュリティ設定ファイルの指定

```
<?xml version="1.0" encoding="utf-8"?>
(...略...)
<app (...略...) >
    <meta-data android:name="android.security.net.config"
               android:resource="@xml/network_security_config" />
(...略...)
</app>
```

▼リスト2　ネットワークセキュリティファイルの記載例

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <trust-anchors>
            <certificates src="@raw/my_ca"/>
        </trust-anchors>
    </domain-config>
</network-security-config>
```

security_configがネットワークセキュリティ設定ファイルです(ファイル名は自由に付けることができます)。

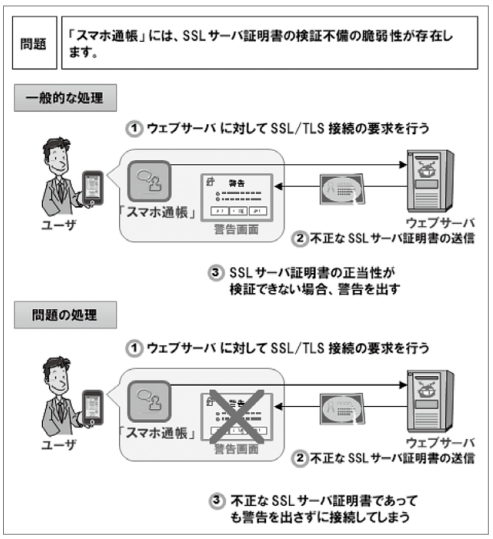
そして、ネットワークセキュリティ設定ファイルに、ネットワークに関するさまざまな設定を書いていきます。**リスト2**のネットワークセキュリティ設定ファイルでは、<certificates>タグにアプリケーションで利用するCA(Certification Authority：認証局)による電子証明書(以降、CA証明書と書きます)のファイル名を指定しています。CA証明書をアプリケーションに同梱し、簡単に利用できるようになったのも新しい機能です。

ネットワークセキュリティ設定ファイルで、指定できることについては後述します。まず、なぜこのような機能が追加されたのかを解説したいと思います。

アプリケーションのセキュリティ問題

現在では、アプリケーションのほとんどはインターネット通信を行います。Android OSが安全になってきたこともあり、SNSをはじめ、銀行、証券、さまざまなアプリでログインを行ったり、重要なデータを送信するようになってきました。アプリケーションに脆弱性(セキュリティホール)があり、パスワード、個人情報などが盗まれたら大変です。しかしながら、多くのアプリには脆弱性が存在します。

▼図1　JVNDB-2015-000015「スマホ通帳におけるSSLサーバ証明書の検証不備の脆弱性」より
http://jvndb.jvn.jp/ja/contents/2015/JVNDB-2015-000015.html



1つの例をあげますと、過去に大垣共立銀行が提供していたスマホ通帳で、SSLサーバ証明書の検証不備の脆弱性が発見されました(**図1**)。この脆弱性は、中間者攻撃(man-in-the-

middle attack)による暗号通信の盗聴などが行われる可能性があります(現在は修正されておりますので安心してください)。

銀行など、セキュリティに注意している会社のアプリでもこのような脆弱性が見つかったことで当時は話題になりましたが、残念ながらこの教訓は生かされず、同じ問題を抱えたアプリケーションが現在でも山ほど存在します。

日本では情報処理推進機構[注1]が、アプリケーションの脆弱性報告を受け付け、公表しています。過去どのような脆弱性が発見されたかは、JVN iPedia[注2]などで見ることができます(**図2**)。検索をすることで、どのようなアプリケーションで、どのような脆弱性があったのかを確認できます。

JVN(Japan Vulnerability Notes)には、第三者によって報告が行われ、開発者側が修正を行った後、脆弱性の公表を許可した場合にのみ情報が掲載されます(大垣共立銀行は公表を許可したわけで、脆弱性があったことは確かに落ち度かもしれませんが、そのおかげでこのようにサンプルとして誌面に載せることができ、セキュリティの啓蒙活動に役立っています。大変すばらしい会社だと思います)。

発見されていないもの、発見されても報告されないもの、報告を受けてからこっそり修正して、公表されないものも多くあり、JVNで確認できるものはほんの一握りです。とくに先にあげた、SSLサーバ証明書の検証不備の脆弱性は比較的発見しやすいこともあり、数多くのアプリケーションが脆弱性を持っていることが確認されています。

SSLサーバ証明書の検証不備はなぜ起こるのか

なぜ多くのアプリケーションがこのSSLサーバ証明書の検証不備の問題を抱えているのでしょうか？ たとえば次のようなことが想定できます。

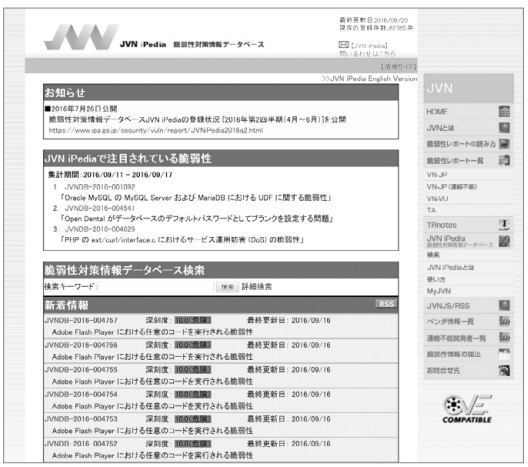
Androidアプリの多くはサーバと通信をします。ログインなどの処理や重要な情報をサーバとやり取りするときは暗号化通信を行わなければなりません。そこで多くのアプリはhttpsを使用して通信を行います。https通信を正しく行うにはサーバ証明書が必要ですが、開発段階では、サーバの正式な証明書は用意できないことがあります。正式なサーバ名が決まり外部に公開されるのは、開発の最後の段階だったり、開発が終わってからだったりするためです。多くの場合、開発段階ではテストサーバを立てて開発を行います。テストサーバとhttps通信を行うときはもちろんサーバ証明書が必要になりますが、パブリックCAにお願いしてお金を払ってテストサーバ用サーバ証明書を作成するといったことはコストの面からたいてい行わず、自己署名証明書(通称オレオレ証明書)を自分で作成してサーバに配置して済ませます。

この状態でクライアントから自己署名証明書を持つテストサーバにアクセスすると、正式な証明書ではないのでエラーになります。このままでは開発が進まないので、エラーを回避するコードを記述して「とりあえず」動かします(この状態はSSLサーバ証明書の検証不備の脆弱性がある状態です)。

開発は進み、正式なサーバが用意されました

注1) IPA。情報処理試験も運営している組織です。
注2) http://jvndb.jvn.jp/

▼図2　脆弱性対策情報データベース JVN iPedia
http://jvndb.jvn.jp/



ので、アクセス先のURLを開発サーバから正式サーバに書き換えます。このとき上記の署名のエラーを回避するコードがそのままですと、「SSLサーバ証明書検証不備」の脆弱性を持ったアプリが生み出されます。

開発者は、重要なデータを送信するのでhttp通信ではなく、https通信を使うという正しいセキュリティの知識がありました[注3]。しかしながら、最後の最後で元に戻すのを忘れるという凡ミスを行ってしまいました。

また、https通信をするネット上のサンプルコードをそのままコピペして使用していることも、SSLサーバ証明書検証不備が多い理由と言われています。

このSSLサーバ証明書の検証不備問題は、Androidでは古くから指摘されていますが、なかなかなくならない問題の1つです。そこで、このような凡ミスによる脆弱性を防ぐための機能が、今回追加されたNetwork Security Configに入れられました。

Network Security Config機能

さて、ネットワークセキュリティ設定ファイルに設定できる内容を見ていきたいと思います。ネットワークセキュリティ設定ファイルに記載できる内容は、大きく分けて次の4種類です。そのうち3種類はCAに関する設定項目となります。

1 信頼すべきCAのコントロール機能

通信時に、どのCAを信頼するか柔軟なコントロールができます。端末にインストールされてしまった不適切なCAを信頼してしまう問題を防ぐこともできます。

2 ビルドモードによるCAの切り替え

リリース時に信頼するCA、デバック時に信頼するCAのように切り替えることができます。これにより先に解説した、SSLサーバ証明書検証不備によくあるケースを防ぐことが可能です。

3 ピンニング

本来信頼すべきCAが偽造証明書を発行する問題を防ぎます。

4 クリアテキスト

暗号化したい通信が、誤って非暗号化されてしまうミスを防ぎます。

信頼すべきCAのコントロール機能

端末のデフォルトの設定ではなく、独自に信頼するCAを設定する機能です。普通のアプリではまず使用しない機能です。

Androidでは、[設定]→[セキュリティ]画面から「ストレージからのインストール」を選択することで、端末にカスタムの証明機関(CA)をインストールすることができます(**図3**)。

また、「信頼できる認証情報」画面でシステムによりインストールされている認証情報、ユーザがインストールした認証情報が確認できます(**図4**)。

▶**図3**
[設定]→[セキュリティ]画面

注3) 昔のWeb創世記にはパスワードをhttp通信上に平文で送るひどいシステムがたくさんありました。

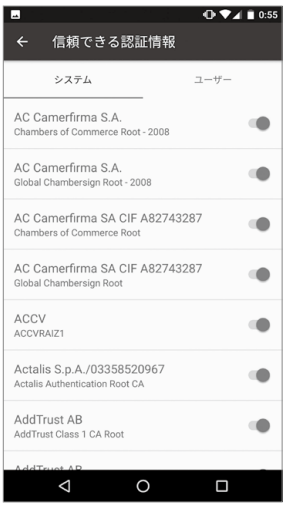
この機能は、アプリ開発時にテストサーバを信頼するためにインストールしたり、セキュリティエンジニアが、通信のパケットをキャプチャしてアプリケーションが不適切に端末の情報を抜き取っていないかを確認するために使用されたりもします。ですがおもに、企業内部CA(大企業でよくあります)を利用するときなどに使われます。

Android 7.0では仕様が変更になり、ユーザがインストールした証明書は使用できなくなりました。認証情報のインストールは下位互換のためできるのですが、Android 7.0以上で動作するように設定されたアプリケーション(AndroidManifest.xmlでtargetSdkVersion=24以上に設定)ではまったく参照されなくなりました。

信頼できる認証情報に攻撃者の認証情報がインストールされると大変危険です。パスワードなどの重要なデータを抜き取られる可能性もあります。スマートフォンは子供からお年寄りまでさまざまな人が使用します。ある悪意のあるアプリケーションをインストールしたとき、促されるままに、攻撃者の認証情報をインストールしてしまうこともありえます。今回の変更により、より多くの人が守られることになると思われます。

企業内CAを使うような場合、Android 7.0からは、Customizing Trusted CAs機能を利用します。CA証明書をアプリケーションの中に含めることができ、通信時にそれを参照できます(**リスト2**参照)。既存アプリは変更が必要になりますが、従来はアプリケーションのインストールに加え、CA証明書を端末に配布(ダウンロード)して手動でインストールといった運用上の手間がありましたが、アプリケーションのみの配布だけで良くなり運用が楽になります。また、証明書の配布問題も解決されます。

また、この信頼するCAのコントロールは強力で、端末にプレインストールされているCAを信頼しないことも可能です。かなり柔軟な指定が可能ですので、もっと詳しく知りたい方はAndroid Developer Site[注4]の方を参照してください。



◀**図4 信頼できる認証情報**

ビルドモードによるCAの切り替え

SSLサーバ証明書の検証不備の原因として、開発時のオレオレ証明書の使用問題を取り上げました。開発時のサーバにアクセスするために、証明書の検査エラーをスキップするコードはこの機能を利用すれば必要はありません。**リスト3**のように"デバック時だけ信頼するCAの証明書"を指定することができます。

この証明書はリリースビルドしたときは使用されません。開発サーバを信頼するCAの証明書を、アプリケーションのプロジェクト内に含める必要がありますが、リリースモードに応じたソースコードの変更は必要なくなります。

ピンニング

通常、アプリケーションはプレインストールされたすべてのCAを信頼します。CAを信用するという前提にたってセキュリティモデルが構築されていますが、世の中にはさまざまな国があります。過去にCA局が偽造証明書を発行するという事件がありました。また、戦争をし

注4) https://developer.android.com/training/articles/security-config.html

▼リスト3　ビルドモードによるCAの切り替え

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <debug-overrides>
        <trust-anchors>
            <certificates src="@raw/debug_cas"/>
        </trust-anchors>
    </debug-overrides>
</network-security-config>
```

▼リスト4　クリアテキストの指定

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config usesCleartextTraffic="false">
        <domain includeSubdomains="true">secure.
example.com</domain>
    </domain-config>
</network-security-config>
```

ているような状況を想定すると、相手国のCA局は信頼できないなど理解できると思います。このため“国内のCA局のみ信頼する”や“特定の国のCA局のみ信頼する”など議論がされてきましたが、インターネット文化として、このような制限は良くないという意見もありました。

そこで生み出されたのがピンニングという技術です。ピンニングを用いることで偽造証明書を検知できますが、ピンニングについて記載しだすと多くの説明が必要になりますので、ここでは詳細は述べません。ピンニングはサーバ側も含めた技術です。サーバの証明書を変更すると(通常定期的に変更します)、クライアントも変更する必要があるなど運用コストが上がります。セキュリティ強度とコストのバランスを考えて採用してください。また、特定のサイトのみにアクセスするサイトでピンニングを採用すると決定した場合は、端末のプレインストールCAを参照せずに、アプリケーション内に含めたCAだけを信用する方法で解決できることもあります。

クリアテキスト

クリアテキストとは、そのまま直訳するとわかりやすいです。「平文」のことです。暗号化通信をしなければいけないところで、誤って平文通信をしたときに通信しないようにしてくれます(**リスト4**)。

https通信をすべきところでhttpで通信をしてしまう可能性などないのでは？と思われるかもしれませんが、サーバサイドから、クライアントがアクセスするURLを指定するような仕様は現在多くあります[注5]。このとき、サーバのプログラムミスにより平文通信を行ってしまうかもしれません。

この機能だけCAとは関係ありませんが、なかなかの機能です。念のため設定しておくのが良いと思います。

最後に

いかがでしたでしょうか？　認証局とか普段あまり意識しない部分の機能追加であるため、あまりピンと来ない変更かもしれません。必須の機能ではありませんし、必ず対応しなければいけないものでもありません。しかしながら、ちょっとした変更で、凡ミスなどによる脆弱性の発生を防ぐことができます。これらの機能を活用することにより、安全安心のプログラムを作ってもらえたらと思っております。SD

注5）TwitterやSNSはほとんどそのような仕様です。

COLUMN

Androidのコミュニティで行われるイベント紹介

Android Bazaar and Conference 2016 Autumn（ABC2016A）開催

国内最大級のAndroidの祭典ABCの2016年秋(ABC2016A)を11月19日(土)に、千葉県柏市の「柏の葉キャンパス」で開催します。テーマは「次世代への孵化装置 Θ Android」として、来年10年目を迎えるAndroidが産み出す、モノ、コト、データを共有します。ぜひご参加ください。
http://abc.android-group.jp/2016a/

一歩進んだ使い方のためのイロハ

Vimの細道

第12回

mattn
twitter:@mattn_jp

Vim使いの必需品 grep

grepをVimから使う場合、設定の仕方やWindows環境でのハマりどころなど押さえておくべき知識があります。今回はそういったVimでgrepを使うえでの基本事項を紹介しつつ、設定することでVimの:grepから使えるようになる5つのgrep代替コマンドを、性能を比較しながら紹介していきます。

エディタとgrep

みなさん、grepはお使いでしょうか。エンジニアならば当然使っていることでしょう。

エンジニアが使っているようなテキストエディタであれば、当然のようにgrep検索という機能が備わっています。検索を開始するディレクトリとファイルの内容を検査するパターンを入力して実行すると、そのパターンでマッチしたファイルを一覧表示する機能です。この機能が使いやすいかどうかが、テキストエディタの価値を大きく左右することもあります。

grepの名前の由来は、**G**lobal **R**egular **E**xpression **P**rintをg/re/pと略したものです。読んで字のごとく、正規表現を使って広域に検索して表示するプログラムを意味します。UNIX系のOSであれば当然インストールされています。Windowsであってもmsys2[注1]などをインストールすることで手軽に利用できます。

Vimにももちろん、このgrepと連携する機能が備わっています。今回はこのgrep機能をVimから活用する方法を紹介したいと思います。

まずはgrepの使い方

シェルからgrepを実行する際の引数は次のとおりです。

```
grep [オプション] [検索パターン] [ファイル]
```

Vimはこのgrepコマンドを呼び出して出力結果を解析し、マッチした行番号や検索語から一覧を作ります。その一覧から、対象のファイルへ簡単にジャンプできるようになっています。Vimからgrep機能を使うには、シェルから実行するのと同じく次のコマンドを実行します。

```
:grep hello *.c
```

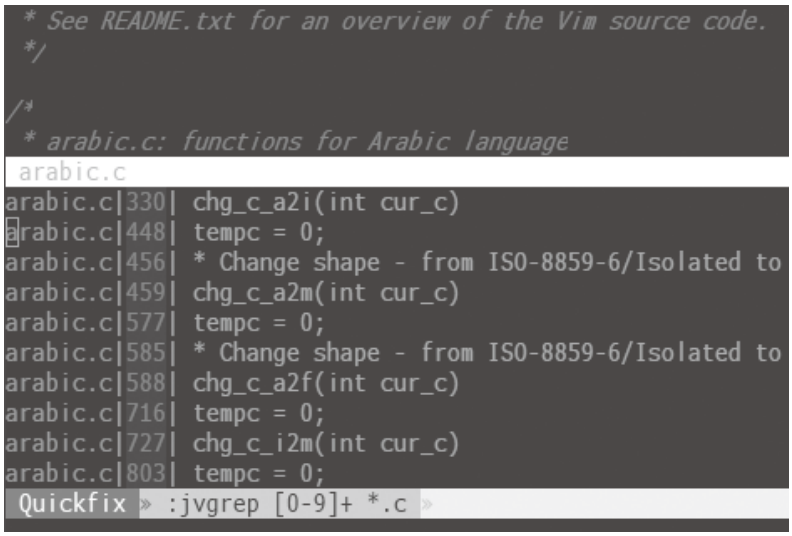
この例では、カレントディレクトリにある拡張子「.c」のファイルから「hello」というパターンを検索しています。この「hello」の部分は正規表現として扱われるので、たとえば、

```
:grep [0-9]+ *.c
```

を実行すれば、数字を検索してquickfixウィンドウに表示してくれます(**図1**)。デフォルトではquickfixウィンドウは自動で開きません。grepコマンドを実行して自動的にquickfixウィンドウを表示したい場合は次の設定をvimrcに設定します。

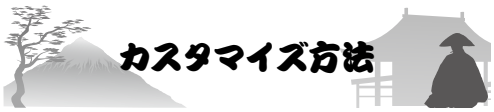
注1) Minimal SYStem 2。Windows用Unixシェル環境。

▼図1　quickfixウィンドウ




```
augroup QuickFixCmd
  autocmd!
  autocmd QuickFixCmdPost *grep* cwindow
augroup END
```

あとはquickfixウィンドウでジャンプしたいファイルの上で Enter を押下すれば簡単に目的のファイルを開けます。cwindowと同じような命令にcopenがありますが、これは結果のありなしにかかわらずquickfixウィンドウを開きます。grepした結果が何もない場合で、quickfixウィンドウを開きたくないときにはcwindowを使います。



Vimはgrepを外部コマンドとして呼び出していますが、その実行するコマンドを変更することもできます。

grepprg

Windowsではデフォルトで、Windowsに標準インストールされているfindstrコマンドが設定されています。しかしfindstrの動作はgrepとは大きく異なり、ANSIエンコーディングしか対応しておらず、簡単な正規表現しか扱えません。そこで多くのWindowsユーザはgrepコマンドを入手し、このgrepprgオプションにUNIXと同様にgrepコマンドを指定しています。

```
:set grepprg=grep¥ -n
```

最近ではmsys2が登場したことでWindowsでもほぼ最新のgrepコマンドを使えますが、以前はWindows向けに正式なgrepコマンドが存在せず、見つけたとしても古いGNU grepで、しかもバグがあったりもしました。

grepformat

前述のとおり、Vimはgrepprgオプションで、使用するgrepコマンドを選ぶことができます。しかし、コマンドによってはVimが期待する出力をせず、行番号を調べることができません。そこでVimは、grepprgで指定したコマンドの出力をどう解析するかを指定するgrepformatオプションを用意しています。デフォルトでは、一般的に想定されるような出力が3つ登録されています。

```
%f:%l:%m,%f:%l%m,%f  %l%m
```

%fがファイル名、%lが行番号、%mが文字列部分となります。お使いのgrepコマンドがVimから正しく使えない場合は、このオプションを見直すと良いでしょう。

各grepコマンドの違い

単純に:grep pattern *.cのように実行するのであれば問題は発生しませんが、オプション引数の指定方法はgrepprgで指定したgrepコマンドに依存します。たとえば、Windowsでのデフォルト設定であるfindstrをそのままお使いの方は、再帰的に*.cを検索するには次のように実行する必要があります。

```
:grep /S pattern *.c
```

また、GNU grepをお使いの方が再帰的に検索する場合は、

```
:grep -r pattern .
```

と実行する必要があります。

再帰的に、かつファイルパターンも指定する場合は標準で指定されているGNU grepではできないため、ほかのgrep互換コマンドを選ぶのが良いでしょう。

vimgrep

Vimからファイルを検索する方法として:grepコマンドを紹介しましたが、Vimにはもう1つ:vimgrepというコマンドが用意されています。このコマンドは外部プログラムを実行しない、Vimに内蔵されたgrepコマンドです。ファイルを検索し、実際にVimがファイルを開いてパターンを検索します。メリットとしてはVimの正規表現がそのまま使えることです。たとえば次のコマンドを実行すると、「foobar」でない「foo」を検索できます。

```
:vimgrep foo\(bar\)\@! *.c
```

また:vimgrepでは、bashのextglob(拡張グロブ)と同じ機能が使えます。extglobとは、ワイルドカードをさらに強力にしたマッチング方法で、Vimではstarstar-wildcardと呼ばれています。

```
:vimgrep pattern /usr/inc**/types.h
```

と実行すると、次のファイルが検索の対象となります。

- /usr/include/types.h
- /usr/include/sys/types.h
- /usr/inc/old/types.h

ただし:vimgrepには1つ癖があり、実行してパターンにマッチすると、そのままそのファイルをVimで開くという動作になっています。プロジェクト内のファイルから検索したい場合であればマッチするファイルの数も知れているのですが、未知のフォルダで多くヒットし得るパターンを指定してしまった場合、意図せず多くのファイルがVimで開かれてしまいます。

場合によっては、一覧しか作成されないgrepコマンドのほうが良いということになります。

grepコマンドを選ぶ

grepコマンドは実行速度が非常に重要です。実行していつまで経っても終わらないgrepは苦痛で仕方ありません。またマルチバイト文字に対応しておらず、Vimから日本語を検索できないものもあります。しかしながら、ありがたいことにgrepと同様の機能を実装したコマンドがいくつかあります。

Ag : The silver searcher

最初はAg[注2]というコマンドです。The silver searcherという名前で、silver(銀)を元素記号で表した「Ag」から名付けられています。C言語

注2) URL https://github.com/ggreer/the_silver_searcher

で書かれており、正規表現のJITやマルチスレッドを駆使することでgrepよりも高速であると言われ、世界中の多くのユーザが使っています。Vimでは次のように設定することで:grepコマンドからagを使用できます。

```
:set grepprg=ag
```

また、専用のVimプラグインag.vim[注3]もあります。これをインストールすると、

```
:Ag keyword
```

でカレントディレクトリ配下のファイルをkeywordで検索でき、自動でquickfixウィンドウが開くようになります。そこそこ速いのですが、WindowsではANSIキャラクタセットにしか対応していません。Linuxでは文字コードがUTF-8ですのでそのまま使えるのですが、Windowsでは引数で渡されたマルチバイト文字がそのまま検索キーワードとして扱われてしまうため、UTF-8のファイルにマッチしません。

Pt : The platinum searcher

Agに対抗して作られているのがPt[注4]です。The platinum searcherと名付けられており、Agよりも速いと言われています。Go言語(golang)で書かれており、Agと同様にgrepprgオプションで設定できます。こちらは、README.mdにほかのエディタやプラグインと連携する方法が多く書かれています。エンコーディングはUTF-8とShift_JIS、EUC-JPの3つに対応しています。

Hw : Highway

Hw[注5]はC言語で書かれており、スレッドを駆使してAgよりも速く動作します。grepprgにhwコマンドを指定することでVimと連携できます。Shift_JISとEUC-JPに対応しています。こちらもAgと同じく、Windowsで引数に与えられたマルチバイト文字が正しく動作しません。

Sift

Sift[注6]は紹介した3つのいずれよりも速いと自負しているgrep系ツールです。Ptと同じくgolangで書かれています。こちらはUTF-8にしか対応していません。ただしAgとは異なり、Windowsでも問題なく動作します。

Jvgrep : Japanese Vimmer's Grep

Jvgrep[注7]は筆者が開発している日本人Vimmer向けのgrepです。GNU grep互換を目指しており、これまでに紹介したツールとは若干毛色が異なるかもしれません。Japaneseという名前が付けられているとおり、次のように多くのエンコーディングをサポートします。

- ISO-2022-JP
- EUC-JP
- UTF-8
- Shift_JIS
- UTF-16LE
- UTF-16BE

BOM付きも自動で検出されます。:set grepprg=jvgrepで:grepコマンドとして使えます。UNIXのシェルによっては、再帰的にワイルドカードを展開する**/*.cという引数指定(拡張グロブ)が使用できますが、jvgrepを使うとWindowsでもこの指定ができるため、

```
:grep pattern **/foo/*/bar*.c
```

といった複雑なファイルパターンの指定ができます。またファイルパターンでディレクトリが指定された場合には、再帰的にディレクトリをたどって検索します。ですので、たとえば「pattern」をカレントディレクトリ(.)から検索するのであれば、

注3) URL https://github.com/rking/ag.vim
注4) URL https://github.com/monochromegane/the_platinum_searcher
注5) URL https://github.com/tkengo/highway
注6) URL https://sift-tool.org
注7) URL https://github.com/mattn/jvgrep

▼表1　grepツールのカタログ

名前	拡張グロブ	引数の扱い	マルチバイト文字列対応	grep速度
Ag	なし	×	UTF-8のみ	19.987s
Pt	なし	○	UTF-8、SHIFT_JIS、EUC-JP	2.326s
Hw	なし	×	UTF-8のみ	1.951s
Sift	なし	△	UTF-8のみ	2.115s
Jvgrep	あり	◎	※	7.455s

※デフォルトはascii、iso-2022-jp、utf-8、euc-jp、sjis、utf-16le、utf-16be。設定可能なエンコーディングは218個40種類。

```
:grep pattern .
```

とだけ指定すれば良いです。

結局どのgrepがいいの？

紹介してきたgrepツールのディレクトリ拡張ありなし、マルチバイト文字対応状況、速度を**表1**にまとめます。Linuxのソースコードをダウンロードして解凍し、そのディレクトリツリーで「linus」を検索した際にかかった秒数を「grep速度」として扱いました。

jvgrepはGNU grep互換でもあり、利便性を取ったことで、ほかのツールよりも若干ながら遅くなってしまいました。それでも世界中で使われているAgよりも2〜3倍速くて日本語もきちんと扱えます。また、サポートするエンコーディングの指定順は前回(2016年9月号)も説明したとおり、誤検知が起きにくい順になっています。

しかしながら、Highwayの速度は捨てがたいです。使いたいケースもユーザによりさまざまだと思いますので、1つ選んで常用してみるか、vim-localrc[注8]などを使ってプロジェクトごとに設定するのも良いかもしれません。

おわりに

Vimからgrepを利用する方法、カスタマイズする方法を説明しました。grepprgやgrepformatさえあれば、今後登場するかもしれない新たなgrepにも簡単に対応できるはずです。自分にあった最強のgrepを探し出してください。SD

注8）URL https://github.com/thinca/vim-localrc

Vim日報

Vim 8リリース

去る9月12日、Vimのメジャーバージョンアップである Vim 8がリリースされました。channelやjob、timerといった非同期をサポートするAPI、partialやlambdaといったVim scriptを便利にするAPIなど、プラグイン作者にうれしい機能が追加されました。それ以外にもWindowsのフォントレンダリングを強化するDirectWriteやGTK3のサポートも追加されています。一部にはVimがメジャーバージョンアップしたことに驚きを隠せない人もいるようですが、開発に関わる側から見ると衰えがあるようにはとても思えません。ぜひ、新しいAPIを使ったプラグインの作成にもチャレンジしてみてください。

Vim 8.0の新機能については vim-jpが解説する次のページをご覧ください。

・http://vim-jp.org/blog/2016/09/13/vim8.0-features.html

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第33回 犯人は誰!?

「昨日……。そんな昔のことは忘れてしまった。明日……。そんな未来のことはわからない」とヤバイ方向にボケつつある編集が担当しているマンガがあるのは本誌だけ?

to be Continued

過去のコードを見直すたびに、自ら黒歴史を紐解くようなイヤな気持ちになります。3日も前になると、なんでこんなの書いたのかと自分を呪うことオッフンです。自分が楽するために書いたコードが人に見られるとなった際には、「卒業アルバム隠さなきゃ」「エロ本隠さなきゃ」みたいな汗が出ます。すぐポイするつもりだったスクリプトを見られるのに抵抗を感じるのは、ゴミ箱を漁られるのに似てるからでしょうか。OSS業界にいるんだし、いつ見られても良いように恥ずかしくないコードを書いていきたいものですね。コードだけではなく、常日頃も恥ずかしいと思うような行動は慎みましょう。いい大人なんですから(なんで鏡を見ているの?)。

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://rubikitch.com/

第31回 Emacsの正規表現（基本編）

敷居は高いけれど、知っていると作業が何倍も楽になる「正規表現」。Emacsのうえでその正規表現を自由に使いこなせれば、作業効率はさらに何倍にも高まることでしょう。今回はまず基本編として、正規表現の基本から、「正規表現の方言」におけるEmacsの立ち位置、メタ文字の紹介まで行います。

正規表現を身に付けよう

なぜ身に付けるべきか？

ども、るびきちです。ついにメジャーリリースとなるEmacs 25.1がリリースされましたね。今回は正規表現について取り上げます。

正規表現と聞くと、あなたはあの暗号じみた文字列にアレルギー反応を起こしたかもしれません。正規表現……確かに初見だと意味不明かもしれません。けれども、より効率的に作業がしたいと望むあなたは正規表現を身に付ける必要があります。あなたは、あの暗号じみた文字列を直視しなければなりません。

たとえあなたがプログラミングをしない人であっても、正規表現を知って使いこなせるだけで、そうでない人と比べこれからの生産性に雲泥の差がつきます。なぜなら、正規表現はテキストエディタやプログラミング言語をはじめとするあらゆる場面で使われているからです。しかも一度覚えてしまえば、将来ほかのテキストエディタやプログラミング言語を使うときにも活用できます。つまり、正規表現は普遍的で一生モノの知識なのです。多くの知識やスキルが数年で陳腐化するITの世界において、正規表現がいかに貴重な知識なのかがよくわかると思います。

正規表現を習得できれば、検索や置換でできることが広がります。正規表現を知らなければ、単純な文字列の検索や置換を繰り返すことになります。単純作業はとても面倒で嫌気がさします。正規表現検索を使えれば、大まかなパターンにマッチさせられます。

正規表現を習得できれば、複雑なテキスト処理ができます。正規表現を知らずにテキスト処理のプログラミングをしたら、複雑なループや場合分けを記述する必要があるなど、プログラムが読み書きしづらくなります。正規表現を知ることによって短いコードで複雑な検索・置換ができるようになります。

正規表現はとても奥が深い世界で、本気で学ぼうとすると書籍1冊分の勉強量になるほどです。ですが、初歩的な正規表現を知るだけで十分な威力を実感できます。さあ、あなたも正規表現を使ってテキパキと作業しましょう。

正規表現とは

では、正規表現とは何でしょうか？　Wikipediaにはこうあります。

正規表現（せいきひょうげん、英: regular expression）とは、文字列の集合を一つの文字列で表現する方法の一つである。正則表現（せいそくひょうげん）とも呼ばれ、形式言語理論の分野では比較的こちらの訳語の方が使われる。まれに

正規式と呼ばれることもある[注1]。

この説明を読んでもちんぷんかんぷんですよね。ではコンピュータを使う立場で正規表現を定義してみましょう。

正規表現とは「文字列のパターンを指定するミニ言語」です。

正規表現とgrep

あなたと正規表現の初めての出会いは、grepコマンドだったかもしれません。grepは、テキストファイルからパターンに一致した行を抜き出すプログラムです。grepに指定するパターンこそが正規表現です。そもそもgrepとは、Global Regular Expression[注2] Printの略なのです。

一番簡単な正規表現は、正規表現文字列そのものがパターンになっている場合です。スペース、英数字、日本語文字だけからなる正規表現は、その文字列そのものがパターンになります。たとえば、`grep Emacs memo.txt`というコマンドを実行したとき、memo.txtの中から「Emacs」という文字列が含まれる行のみが出力されます。これだけならば、何ら難しいことはありません。

メタ文字

先ほど、正規表現とは「文字列のパターンを指定するミニ言語」だと定義しました。ミニ言語であるということは、正規表現を構成する一部の文字に、特別な意味が込められていることを意味します。

たとえば正規表現 . は改行以外の任意の文字にマッチします。正規表現`a.c`は「abc」にも「a.c」にも「0abcd」にもマッチします。ほかにも行頭を表す`^`や行末を表す`$`もあります。正規表現`^red`は「red」や「redo」にマッチしますが「hundred」にはマッチしません。正規表現`^red$`は改行を含まない限り「red」のみにマッチします。これらのように特別な意味が込められた文字のことを「メタ文字」といいます。

◆ ◆ ◆

このように正規表現の本質とは、“文字列がパターンにマッチするかどうか”ですので、概念としては難しくありません。正規表現が難しいと敬遠されがちなのは、メタ文字がたくさん存在し、後述の「方言」によってメタ文字が異なるからです。なお、ここで紹介したメタ文字はすべての方言で使えるものです。

Emacs流正規表現

いろいろな正規表現の方言

正規表現の方言には大きく分けて4系統あります。

- 基本正規表現
- 拡張正規表現
- Perl系統の正規表現
- Emacsの正規表現

基本正規表現はgrepやsedで使われている正規表現で、メタ文字が少ないことが特徴です。その代わり、拡張正規表現などと比べて冗長になってしまうという欠点があります。

拡張正規表現はegrep（`grep -E`）で使われている正規表現で、基本正規表現よりもメタ文字が多く、簡潔に記述できます。

Perlの正規表現は拡張正規表現をさらに拡張し、より簡潔に表現できるような新たなメタ文字が登場したり、先読みや戻り読みなど基本・拡張正規表現では使えなかった機能が用意されています。Perlの正規表現は便利で高機能であるため、PCRE（Perl Compatible Regular Expression）という名のライブラリになっています。PCREは多くのプログラミング言語やアプリケーションで使われています。

RubyやPythonやJavaScriptの正規表現も、Perlの正規表現に近いものになっています。お

注1）出典：https://ja.wikipedia.org/wiki/正規表現、2016年10月3日時点
注2）Regular Expression＝正規表現。

そらく“ふつうの正規表現”と言えば、Perl系統の正規表現ではないでしょうか。

Emacsの正規表現

Emacsの正規表現は残念ながらPerlほどの機能はありません。メタ文字も独自のもので、基本正規表現と拡張正規表現がごっちゃになった感じです。

一方でEmacsの正規表現独自の機能として、シンタックステーブルやカーソル位置を表す正規表現も存在します。

Emacsの正規表現は残念ながら嫌われ者です。その最大の理由はメタ文字にバックスラッシュが多用されていることです。そのためPCREと比べて冗長になってしまいます。

おまけにEmacs Lispと正規表現の相性が最悪です。Emacs Lispには正規表現専用のリテラルが用意されていません。そのため文字列で正規表現を表現することになります。そして、バックスラッシュを文字列リテラルで表現するときは二重バックスラッシュになります。その結果、Emacs Lisp内で正規表現を表現すると二重バックスラッシュの嵐になってしまうのです。

▼表1　Emacs/Perl系統共通のメタ文字

メタ文字	意味
.	改行以外のすべての文字に
*	直前の表現が0回以上（欲張りマッチ）
+	直前の表現が1回以上（欲張りマッチ）
?	前の表現が0回か1回（欲張りマッチ）
*?	直前の表現が0回以上（非欲張りマッチ）
+?	直前の表現が1回以上（非欲張りマッチ）
??	直前の表現が0回か1回（非欲張りマッチ）
[…]	文字クラス（どれかの文字に一致）
[^ …]	否定文字クラス（どの文字にも一致しない）
^	行頭
$	行末
\b	単語の境界
\B	単語の境界ではない（途中）
\w	単語の構成要素 （シンタックステーブル依存）
\W	単語の構成要素ではない（同上）
\N（数字）	N番目の括弧にマッチしたテキスト （後方参照）

そんな嫌われ者のEmacsの正規表現ですが、基本の理解は大切ですので、怖がらずに直視してみましょう。別な記法からEmacsの正規表現に変換するアプローチもありますが、それはあくまでも応用です。

メタ文字一覧

それでは、Emacsの正規表現のおもなメタ文字を見てみましょう（**表1**）。

参考のために、Perl系統のメタ文字とも比較します（**表2**）。Perlなどに慣れている人であれば、違いがわかればその分理解しやすいからです。ここからがいわゆる“ふつうの正規表現”との違いになってきます。Emacsの正規表現においては、一般に親しまれているメタ文字 | ( ) の前にはそれぞれバックスラッシュが付いてるので注意してください。

用語説明

表1、表2における見慣れない用語について説明します。

「文字クラス」とは、[]で囲まれた文字のうちのどれかにマッチする表現です。否定文字クラスとは文字クラスの逆で、指定された文字以外

▼表2　EmacsとPerl系統では異なるメタ文字

Emacs	Perl	意味
|	|	| で区切られた表現のうちのどれか
\(... \)	(...)	後方参照ありのグルーピング
\(?: ... \)	(?: ...)	後方参照なしのグルーピング
\{N\}	{N}	直前の表現がN回
\{N,\}	{N,}	直前の表現がN回以上
\{N,M\}	{N,M}	直前の表現がN〜M回
\`	\A	文字列・バッファの先頭
\'	\z	文字列・バッファの末尾
\=	なし	バッファの現在位置
\<	なし	単語の開始位置
\>	なし	単語の終了位置
_<	なし	シンボルの開始位置
_>	なし	シンボルの終了位置

にマッチする表現です。

文字クラスにはハイフンを含めることで、範囲を指定できます。たとえば[0-9]は数字、[a-z]がアルファベットです。ハイフンそのものを含めるには、文字クラスの最初か最後に指定する必要があります。

「欲張りマッチ」とは、* + ?がなるべく長くマッチ（最長マッチ）する習性のことを言います。それらのメタ文字に?を加えれば最短マッチになります。置換の際には欲張りマッチに注意しないと、思わぬ結果になってしまいます。

「グルーピング」は\(と\)で囲まれた正規表現をひとまとめに扱うことです。しばしば、\|* + ?と組み合わされます。

「後方参照」とは、グルーピングにマッチした正規表現に、マッチした部分を記憶して、あとで参照する機能です。同じパターンの繰返しを表現するときに使います。

正規表現の例

最後にEmacs、Perl系統を合わせて正規表現の実例を**表3**に示すことにします。なお、文字列中の\nは改行文字、\tはタブ文字とします。表中の「マッチする文字列」列では、正規表現にマッチした部分文字列を()内に示します。

◆ ◆ ◆

今回はEmacsの正規表現を取り上げました。次回は正規表現を扱うコマンドや応用を取り上げる予定です。

筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指し、毎日更新しています。手元でgrep検索できるよう全文をGitHubに置いています。

つい最近、fishというシェルにハマってしまったため、「fishシェル普及計画」というサイトも立ち上げました。

こちら➡http://fish.rubikitch.com

fishは最初から使いやすいような巧妙なしくみになっていますので、ぜひとも試していただけると幸いです。筆者は15年間使っていたzshからあっさり乗り換えてしまいました。

またEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでもお答えします。「こんなパッケージ知らない？」「挙動がおかしいからなんとかしてよ！」はもちろんのこと、自作elispプログラムや文章の添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

▼表3　正規表現の実例（Emacs、Perl系統）

Emacs	Perl	マッチする	マッチしない
ox	ox	ox(ox)、fox(ox)	oyx
^re	^re	re(re)、regex(re)	ore
^rx$	^rx$	rx(rx)、foo\nrx(rx)	rxt, frx
fo*	fo*	f(f)、fo(fo)、foo(foo)	o
fo+	fo+	fo(fo)、foo(foo)	f
fo+?	fo+?	fo(fo)、foo(fo)	f
fo*?	fo*?	f(f)、fo(f)、foo(f)	o
fo|ba	fo|ba	foo(fo)、bar(ba)	ob
\`rx\'	\Arx\z	rx(rx)	foo\nrx
[0-9]+	\d+（[0-9]+も可）	12345(12345)	w
\bgz\b	\bgz\b	gz(gz)、a\tgz(gz)	tgz
\([a-z]+\)\1	([a-z]+)\1	murmur(murmur)	murxmur
^[^a-c]d	^[^a-c]d	dd(dd)	ad、bd、cd
[0-9]\{3\}-[0-9]\{2\}	\d{3}-\d{2}	0123-456(123-45)	123-4

書いて覚える Swift 入門

第20回 Pokémon GO、iPhone 7、macOS（Sierra）

Author 小飼 弾（こがい だん）　twitter @dankogai

一緒に歩くと いいことあるよ!

本稿執筆現在、(i|mac|tv|watch)OSは無事バージョンが上がり、iPhone 7も筆者の手元に届きました。が、macOS Sierra、Apple Watch Series 2は間に合いませんでしたし、次期MacBook Proは影も形もありません。そんな微妙な時期ですが、**Pokémon GO**は今もなお大ニュースであり続けています[注1]。

まず、待望のバディシステムがバージョン1.7.0に搭載されました。ポケモンを進化させたり強化させたりするためには、「ほしのすな」という共通通貨と「種族のアメ」という固有通貨の双方が必要なのは読者の皆さんもご存じのとおりですが、どんなポケモンをGETしても増える前者と異なり、後者はその種族のポケモンGETからしか入手ができず、レアポケモンの育成は困難を極めていました。それが、選んだポケモンと一緒に歩くことで、一定距離ごとにアメがもらえるようになったのです。これで理論上は、ミニリュウが一匹さえいれば確実にカイリューをGETできることになったわけです[注2]。

ポケモン図鑑完成の難易度はこれで下がる一方、バディシステム以前以後ではある意味別のゲームでもあるわけで、なんとかその前に図鑑完成まで持っていけたらいいけど難しいなあと思っていた矢先の9月10日、幸いにも**国内142種目**をGETすることができました(**図1**)。

▼図1　国内142種目をGET[注3]

▼図2　レベル32を達成

図2のようにゲーム開始から50日目、歩行距離415km、捕まえたポケモン5,990匹、孵(かえ)した卵623個、ポケストップ回し8,120回[注4]。しかし、Pokémon GOはオープンエンド。レベルカンストはあってもゲームオーバーはありません。その後も歩き続けてなんぼなわけですが、幸いなことに歩き癖がすっかりついて、この1ヵ月平均で20,000歩／日歩く体になっていました(**図3**)。台風で土砂降りの日で

注1） Pokémon GO（http://www.pokemongo.jp）
注2） ただし、その場合に必要な歩行距離は620km！
注3） https://twitter.com/dankogai/status/774457126332669952
注4） https://twitter.com/dankogai/status/774458397596921856

▼図3　20,000歩／日も歩く体のログ

▼図4　ラ++と歩く日々

▼図6　Pokémon GO Plus だけではゲームプレーに限界がある

▼図5　Pokémon GO PlusをApple Watch（Series 1）のミラネーゼループに留める

さえ1万歩は歩かないと欲求不満になるなんて。

そしてバージョン1.7.0が到着したのが、iOS 10アップデート直後。さらにiPhone 7と同日にPokémon GO Plus到着というわけで、ラ++と歩く日々がまだ続いています（**図4**）。

ボタンのないiPhone 7、ボタンしかないPokémon GO Plus

で、iPhone 7です。これまた読者の皆さんご存じのとおり、穴とボタンが1つずつ減りました。イヤフォンジャックとホームボタン。後者はなくなったというより機械式から感圧式に生まれ変わったのですが、それで耐水性能を獲得しました。耐水という点では先行していた競合他社のスマフォはジャックやコネクタに蓋をつけ、ホームボタンそのものをなくしてタッチスクリーンに表示するという、わかりやすい代わりに不恰好なものでしたが、そういったことをせず、ある意味「そのままの姿」で実現したというのが実にAppleらしい後出しじゃんけんでした。

iOS 10（のデフォルト設定）で一番戸惑いが多かった「ホームボタン触るだけでアンロックできていたのに押さないといけなくなった」というのも、iPhone 7ではむしろ自然ですし、機械式ではないのに押した感覚を実現するTaptic Engineはホームボタンを「仮想機械式」にするにとどまらず、たとえばメニューのダイアル選択でカリカリと本物のダイアルを選択している感触まで実現するなど実によくできています。機械的可動部は減ったのに機械的感触は増えている。実に見事です。

その意味でPokémon GO Plusというデバイスは、ある意味その真逆にあります。ついているのはLEDで7色、もとい4色に光るボタン（もちろん機械式！）が1つだけ。筆者はApple Watch（Series 1）のミラネーゼループに留めてますが、笑っちゃうぐらいだっさい（**図5**）。

しかしこれのおかげで、iPhoneの画面をにらめっこしなくてもPokémon GOできるようになりました。もちろんボタン1個でできることはたかが知れていて、ポケストップ回しとポケモン捕獲。それもスーパーボールやハイパーボールといった高性能ボールは使えず、赤白ボール1回だけ。失敗すればポケモンハイさようなら（**図6**）。これでピカチュウを取ろうとしてはいけません（笑）。

しかしPokémon GOの一番の狙いはトレーナーを歩かせることにあり、だとしたらむしろ画面をにらめっこしているのは避けるべきことで、オープニング画面も歩きスマフォしているとギャラドスに食われるぞと警告しているぐらいです。実際筆者も恥を忍んで、もとい恥も外聞もなく使ってみたところ実に快適でした。ぼうけんノートに「ミニリュウに逃げられた」という記録を見つけてちょっぴりムッとはしましたが。

しかし、よく考えてみると、Pokémon GO Plusって実にムダなデバイスではあるのですよ。ポケストップに近づくだけでアイテムGETできたり、ポケモンが近くにいるだけで自動捕獲したり、つまりノーボタンにしたほうが実装面では楽なのですから。しかしそれだともはやPokémon GOはゲームではなく単なる通知アプリになってしまいます。Pokémon GO Plusは、まさにその余計なワンクリックがあることでポケモンの存在をリアルにしているわけです。

iPhone 7とPokémon GO Plus。かたや物理を略すことでリアルにし、かたや物理を加えることでリアルにする。リアルとはいったい何なのか。そんな哲学的な疑問がリアルに沸き起こってきます。AR = Augumented Reality = 強化現実といっていますが、もう弾言してしまいましょう。リアルとは「実体」ではなく、リアルとは「実感」なのだと。物理的事実だけではなく、論理的虚構だけでもなく、双方合わせて我々が「これが世界だ」と感じているもの、それがリアルなのだと。複素現実 = Complex Realityという言葉が思い浮かびましたが、ちょっと数学的すぎるかなあ……。

上野の不忍池でプレイが禁止されたり、レインボーブリッジに向かうお台場の自動車専用道路にプレイヤーが押し寄せたりとPokémon GOはリリース後2ヵ月経過した今も社会現象であり続けていますが、驚くべきなのは、楽しんでる人も迷惑を被っている人も誰も「ポケモンなんて存在しない。ただのデータだ」という人がいないこと。もはやポケモンは虚構ではない、今そこにある現実なのです。

これが我々プログラマにとって何を意味するか。我々は、リアルを想像し改変する力を得たということなのです。メタファーではなくリアルに。ふと、金子勇[注5]さんのことを思い出しました。Winnyが無罪判決を勝ち取れたのは、まだまだその影響がコンピューターネットワークに留まっていたからではないのかと。仮に誰かポケソースを偽造するツールを作って、それを利用した誰かがミュウが湧く偽ポケソースをどこかに設置して、結果押し寄せたトレーナーで圧死者が出たら、ツール作成者は著作権ではなく殺人の幇助で追訴されるのではないか……。

妄想にすぎない、というにはARはすでにリアルすぎるというのは心に留めておいたほうがいいかと。

Swift Playgroundsはモバイルプログラミングを実現するか?

このまま妄想を続けたい誘惑を振り切って、Swiftの話題に入りましょう。 ARがモバイルアプリを「デスクトップでしかできなかったことがどこでもできる」から「実際の場所に行ってみないとできない」と一段進めたのであれば、モバイルプログラミングとはいったい何を意味するのでしょうか?

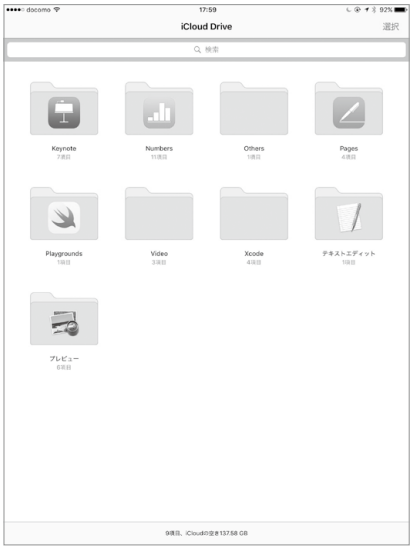
Swift Playgrounds for iPadはその試みの1つかもしれません。

図7を見てのとおり、リアルキーボードなしでもフルセットのSwiftプログラミングができます。それも「単に動く」以上にある程度のモバイル最適化が進んでいます。カーソルの位置によって正しい構文として成立する表現が候補に現れますし、変数やリテラルをクリックすれば

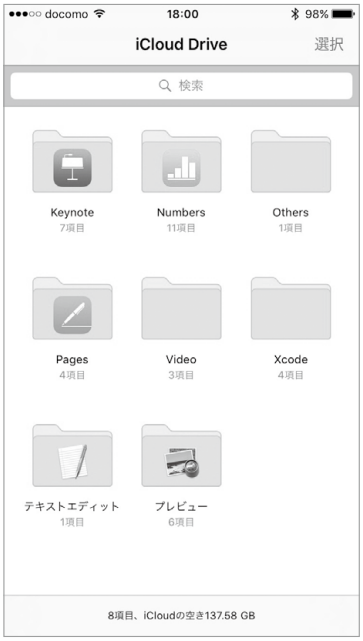
注5) 金子勇(http://blog.livedoor.jp/dankogai/archives/51878044.html)

▼図7　Swift Playgrounds for iPad

▼図8　Swift Playgrounds for iPadはiCloudに対応している



▼図9　Swift Playgrounds for iPadはiPhoneに未対応であることに注意



その型にあった候補が現れます。

とはいうものの、やはり「iPadの方がMacよりも快適にプログラミングできる」というにはほど遠く、実際に使いこむには大まかな作業はMacで行い、それをiPadでレタッチするというiWorkに近い使い方に現状落ち着くと思うのですが、ここで1つ問題が。

現時点で、iPad以外の環境でiCloud DriveがSwift Playgroundsに対応していないようなのです（**図8**、**図9**）。

スクリーンショットを見てもわかるとおり、iPadのiCloud DriveにはPlaygroundsがあるのに、iPhoneの方にはない。Macも同様です。macOS SierraとXcode 8.1待ちということでしょうか……。

というわけで冒頭で述べたとおり「微妙な時期」だけあってなんとも微妙な結果となりましたが、次号までにはmacOSもSierraになるはずですし、もしかしたらMacBook Proも刷新されているかもしれません。次回は本連載の主旨どおり、Swift言語メインの記事をお届けできる……はずです。SD

Sphinxで始めるドキュメント作成術

安宅 洋輔　ATAKA Yosuke　Twitter @kk_Ataka

第20回 Sphinx環境ひとめぐり ——エディタ、ビルド、バージョン管理、公開

今回のテーマ

今回はSphinxを利用してドキュメントを作成するにあたって、知っておくと便利なツール、プラグイン、サービスを紹介します。ドキュメントの作成からSphinxで出力したファイルを外部へ公開するまでの流れを以下の順で追いながら、それぞれの場面で作業の手助けになるようなツール、プラグインについて特徴と使い方を確認していきます。

- **ドキュメントを書く**
  reStructuredText（以下、reST）を便利に書けるエディタのプラグイン
- **ビルドする**
  ビルドの自動化ツール
- **バージョン管理する**
  ドキュメントファイルを管理する方法、ツール
- **公開する**
  出力したファイルを外部へ公開するサービス

今回は**図1**のようなプロジェクトを例にとって進めていきます。sphinx-quickstartコマンドを実行し、以下の質問を除きデフォルトで設定しています。

- Project name：回答必須のため適当な値を入力
- Author name(s)：同上
- Project version：同上
- Project language [en]：ja
- Do you want to use the epub builder (y/n) [n]：y

動作はMac OS X El Capitan 10.11.5、Python 2.7.9以降、Sphinx 1.4以降の環境で検証しています。

ドキュメントを書く

SphinxではreSTというプレーンテキストで表現できる記法を使いドキュメントを書きます。そのため、文字コードをUTF-8に指定できるものであれば、どのエディタでもドキュメントを作成できます。

VimやEmacsといったエディタではreST記法がサポートされています。これらのエディタでreSTファイルを開くとreST記法に従って文章を見やすくハイライトしてくれます。

また、Vimにはriv.vim[注1]、Emacsにはrst.el[注2]

注1）https://github.com/Rykka/riv.vim
注2）http://docutils.sourceforge.net/tools/editors/emacs/

▼図1　プロジェクトファイル一覧

```
sphinx-project/
├── Makefile
├── _build
├── _static
├── _templates
├── conf.py
├── index.rst
└── make.bat
```

というプラグインがあります。導入することで次のような作業がコマンド1つでできるようになり、覚えると便利になるでしょう。

- ドキュメントを見出しごとに折りたたむ
- 見出しのマークアップ
- テーブルの作成、整形

ここでは、Vimのriv.vimを使ってショートカットを実行する例を紹介します。使用するriv.vimのバージョンは0.79です。

riv.vimのREADMEでは、インストールにVundle[注3]を使用しています。.vimrcに`Bundle 'Rykka/riv.vim'`と追加し、riv.vimをインストールしましょう。

ドキュメントを見出しごとに折りたたみ

reSTドキュメントを開くと**図2**のように見出しがすべて折りたたまれている状態になります。

見出しにカーソルを合わせ、Enterを入力することで見出しの展開、折りたたみを切り替えられます。見出しを展開すると**図3**のようになります。

また、すべての見出しを一括で展開、折りたたむこともできます。ノーマルモード時にzRで展開、zMで折りたたみです。

見出しのマークアップ

見出しのマークアップを行うには、**図4**のように見出しにしたい文章にカーソルを合わせ<C-E>s[N]と入力します。<C-E>はCtrlを押しながらEを入力します。[N]は1から6までのいずれかを入力します。

見出しは1から6に対応した記号が出力されます。1なら=(**図5**)、2なら-(**図6**)です。reSTでは「見出しに使う装飾記号の出現順に見出しレベルが決まる」というルールがあります。見出しレベルに合わせて1から6を順番に使うと良いでしょう。

テーブルの作成、整形

テーブル(表)を作成するには、<C-E>tcと入力します。行と列の数を入力したら、その数値どおりにグリッドテーブルが作成されます。テーブルレイアウトの整形や行列の挿入が自動で実行されるため、はじめは戸惑うかもしれません。しかし、慣れると自分でテーブルを調節する手

注3) Vimのプラグインを管理するためのプラグイン。
https://github.com/VundleVim/Vundle.vim

▼図2　見出しが折りたたまれた状態

```
  1 第20回 Sphinx環境ひとめぐり  ----------------------------------| 1       3+
  5 今回のテーマ  --------------------------------------------------| 2      23+
 29 ドキュメントを書く  -------------------------------------------| 3      85+
115 ビルドする  ---------------------------------------------------| 4      81+
197 バージョン管理する  -------------------------------------------| 5      28+
226 公開する  -----------------------------------------------------| 6     110+
337 まとめ  -------------------------------------------------------| 7      10+
~
```

▼図3　選択した見出しを展開

```
  1 第20回 Sphinx環境ひとめぐり  ----------------------------------| 1       3+
  5 今回のテーマ$
  6 ===========$
  7 $
  8 今回は、Sphinxを利用したドキュメント作成をはじめるにあたって、知っておくと便
    利なツール、プラグイン、サービスを紹介します。$
  9 $
 10 (中略)$
 11 $
 12 動作環境はMax OS X El Capitan 10.11.5、Python 2.7.9以降、Sphinx 1.4以降
    ージョンになります。$
 13 $
 14 .. literalinclude:: inc/tree.inc  -------------------------------|        4+
 19 ドキュメントを書く  -------------------------------------------|       85+
105 ビルドする  ---------------------------------------------------|       81+
187 バージョン管理する  -------------------------------------------|       28+
216 公開する  -----------------------------------------------------| ..    110+
327 まとめ  -------------------------------------------------------| 6.3    10+
~
```

▼図4　見出しにしたい行を選択

```
75 $
76 見出しのマークアップ$
77 $
78 $
79 $
```

▼図5　<C-E>s1と入力した結果

```
75 $
76 見出しのマークアップ$
77 ====================$
78 $
79 $
```

▼図6　<C-E>s2と入力した結果

```
75 $
76 見出しのマークアップ$
77 --------------------$
78 $
79 $
```

間が省けます。

◆ ◆ ◆

riv.vimの機能の一例を紹介しました。ほかにも、文字の装飾やリンクの自動生成などできることはたくさんあります。rim.vimのWiki[注4]からやりたいことを探してみるのも良いでしょう。

ビルドする

Sphinxはreストで作成したドキュメントを、make htmlやmake epubなどのコマンドで指定した形式にビルドします。Sphinxでドキュメントを作成する場合、次の①～④の手順を繰り返すことになります。ビルドしたファイルの確認には、記述の内容が確認しやすいHTML形式を使用すると良いでしょう。

①reSTでドキュメントの内容を編集する
②ビルドする(make html)
③ビルドされたHTML(_build/html/index.htmlなど)をブラウザで開く。すでに開いている場合はリロードする
④HTMLでの出力結果を確認する

しかし、ドキュメントを書き進めるたびに何度も手動でビルドやブラウザのリロードをするのはたいへんな手間です。いくつかの手順を自動化して、ドキュメントの作成に集中しましょう。ビルドの自動化は次のような方法で実現できます。

- sphinx-autobuild[注5]を使用する
- Emacsを使用している場合、sphinx-frontend[注6]を使用する
- Windowsでサクラエディタを使用している場合、マクロを作成し、ショートカットに設定する[注7]

注4) https://github.com/Rykka/riv.vim/wiki/1.-Instruction#rivtablecreate
注5) https://pypi.python.org/pypi/sphinx-autobuild
注6) https://github.com/kostafey/sphinx-frontend/
注7) http://advent-calendar2012.usaturn.net/2012/12/10/make_on_sakura.html

今回は、sphinx-autobuildを使用した自動ビルド方法について紹介していきます。

sphinx-autobuildを使用した自動ビルド

sphinx-autobuildを使用すると「ビルドする」「ブラウザで開いているHTMLをリロードする」の手順を自動化できます。インストールは、ターミナルで次のコマンドを実行します。

```
$ pip install sphinx-autobuild
```

インストールに成功したらSphinxプロジェクト内で**図7**のコマンドを実行してみましょう。sphinx-autobuildコマンドの-bオプションは自動ビルドする形式を記述します。ここではHTML形式を指定しています。source/は.rstが格納されているパス、_build/html/はビルド先のパスを指定しています。

sphinx-autobuildを実行するとローカルホストでWebサーバが起動します。デフォルトの設定では8000番ポートで動作しているので、ブラウザから「http://127.0.0.1:8000」にアクセスしましょう。Sphinxでビルドされたhtmlが表示されます。

次に、ブラウザでドキュメントを開いたまま、Sphinxプロジェクト内の.rstファイルを更新してみましょう。このとき、**図7**を実行したターミナルは終了させないように注意します。ファイルを更新し、保存すると、sphinx-autobuildが変更を検知してビルドを実行してくれます(**図8**)。

先ほど開いたブラウザを確認しましょう。自動でページがリロードされ、更新されたHTMLが表示されているはずです。

このように、sphinx-autobuildを利用することで「ビルドする」「ブラウザで開いているHTMLをリロードする」作業を自動化できました。簡単な作業ですが、ドキュメントを作成するうえで何度も行う必要があります。自動化できるとドキュメント作成に集中することができるでしょう。

バージョン管理する

Sphinxのプロジェクトをバージョン管理するにはいくつかの方法が考えられます。その中でreSTと相性が良いのはGit、Mercurial、Subversionといったバージョン管理ツールを使用する手法です。プロジェクトのreSTファイルをはじめとする構成ファイルはプレーンテキストであるため、これらのバージョン管理ツールと相性が良いです。

バージョン管理について、詳しくは本連載の第7回「Webサイトを作ろう（後編）」[注8]で説明されています。

今回はバージョン管理から公開までの手順をすべてGitHubで完結させてみましょう。バージョン管理ツールはGitを使用します。Sphinxドキュメントをバージョン管理する際の注意点を紹介します。

▼図7　sphinx-autobuildコマンドを実行

```
$ sphinx-autobuild -b html source/ _build/html/

+--------- manually triggered build -------------------------------------------
| Running Sphinx v1.4.1
(..中略..)
| build succeeded.
+--------------------------------------------------------------------------------

[I 160611 15:22:43 server:281] Serving on http://127.0.0.1:8000
[I 160611 15:22:43 handlers:59] Start watching changes
[I 160611 15:22:43 handlers:61] Start detecting changes
```

▼図8　変更を検知してビルドを自動で実行

```
+--------- source/index.rst changed -------------------------------------------
| Running Sphinx v1.4.1
(..中略..)
| build succeeded.
+--------------------------------------------------------------------------------


+--------- source/index.rst changed -------------------------------------------
| Running Sphinx v1.4.1
(..中略..)
| build succeeded.
+--------------------------------------------------------------------------------

[I 160611 16:07:31 handlers:132] Browser Connected: http://localhost:8000/index.html
```

成果物は管理対象から外す

_buildディレクトリ以下のファイルはmakeコマンドで生成することができるため、バージョン管理下に置く必要はないでしょう。.gitignoreファイルに_buildディレクトリを追加します。

注8） 本誌2015年10月号。

COLUMN

Makefileに自動ビルドのターゲットを追加する

sphinx-autobuildを利用するにあたって、Makefileに**リストA**を追加すると、make livehtmlと入力するだけで自動ビルドを実行できるようになります。この方法はsphinx-autobuildのpypiページ（**注5**参照）の「Makefile integration」にも書かれています。

▼リストA　Makefileにターゲット追加

```
livehtml:
    sphinx-autobuild -b html $(ALLSPHINXOPTS) $(BUILDDIR)/html
```

公開する

最後に公開です。第7回「Webサイトを作ろう(後編)」ではRead the Docs[注9]を使用した公開方法を解説しました。Read the Docsはもともと Sphinx専用に作られたサービスなので、Sphinx ドキュメントを公開するのに適しています(現在はSphinx以外にMkDocs[注10]にも対応しています)。

Sphinxで生成したHTMLはそのまま配信できるため、Read the Docs以外のサービスでもHTMLファイルを配信可能なサービスを利用することでドキュメントを公開できます。有名なサービスとしてはGitHub Pages、Bitbucket、Amazon S3などがあります。VPSやレンタルサーバを利用しても良いでしょう。

前の「バージョン管理する」の節で触れたとおり、今回はGitHubでバージョン管理から公開までを完結させます。GitHub Pagesでは**表1**のように、リポジトリの特定ブランチのファイルを公開できます[注11]。

表1に加え、2016年8月23日からプロジェクトページではmasterブランチのdocs/直下のファイルを公開できるようになりました[注12]。

以下、公開手順を説明していきます。

GitHub Pagesで公開するときの注意点

GitHub Pagesで公開する際に注意点があります。Sphinxでビルドしたファイルをそのまま GitHub Pagesで公開すると**図9**のような表示になります。

これはGitHub Pagesの仕様[注13]で、アンダースコア(_)から始まるディレクトリを読み込めないために起こります[注14]。ビルド時に画像やCSSファイルが_images/や_static/ディレクトリに格納されるので、読み込みに失敗しているのです。次のどちらかの方法で回避する必要があります。

- docs/直下に.nojekyllファイルを追加する
- sphinxtogithub[注15]を使う

注9) https://readthedocs.org
注10) Markdownで書かれたドキュメントからHTMLを生成するツール。 http://www.mkdocs.org/

▼図9 CSSや画像が表示されない

注11) https://help.github.com/articles/user-organization-and-project-pages
注12) https://github.com/blog/2233-publish-your-project-documentation-with-github-pages
注13) https://jekyllrb.com
注14) https://help.github.com/articles/files-that-start-with-an-underscore-are-missing
注15) https://pypi.python.org/pypi/sphinxtogithub

▼表1 リポジトリと公開用ブランチの対応

リポジトリの種類	URL	公開するファイルのブランチ
ユーザページ	username.github.io	master
オーガニゼーションページ	orgname.github.io	master
ユーザのプロジェクトページ	username.github.io/projectname	master または gh-pages
オーガニゼーションのプロジェクトページ	orgname.github.io/projectname	master または gh-pages

.nojekyllファイルを配備すると、アンダースコアから始まるディレクトリもGitHub Pages上で通常どおり読み込めるようになります。

sphinxtogithubは、ビルド時にアンダースコアから始まるディレクトリをリネームするプラグインです。sphinxtogithubを導入し、make htmlを実行すると**図10**のようにディレクトリをリネームしていることがわかります。

GitHub Pagesの公開設定

ブラウザで、GitHub Pagesに公開するプロジェクトのリポジトリを開き、「Settings」リンクを選択します(**図11**)。SettingsページのGit Hub Pages の Source を「master branch /docs folder」に変更します(**図12**)。そして、「Save」を選択します(**図13**)。

これでmasterブランチのdocs/直下のファイルを公開できるようになりました。

HTMLファイルをコミットする

ビルドしたHTMLファイルをdocs/直下にコピーし、変更をpushすれば公開完了です。設定に誤りがなければ、CSSも正しく適用されるでしょう(**図14**)。

まとめ＆次回予告

「ドキュメントを書く」〜「ビルドする」の節では、エディタのプラグイン、ビルドツールを導入し、ドキュメント作成時に繰り返し行う必要がある煩雑な作業を自動化することができました。「バージョン管理する」〜「公開する」の節では、GitおよびGitHub Pagesを採用し、GitHub内で完結してドキュメント公開する際の注意点を紹介しました。

これらを組み合わせることで、よりドキュメント作成に集中できるようになると幸いです。

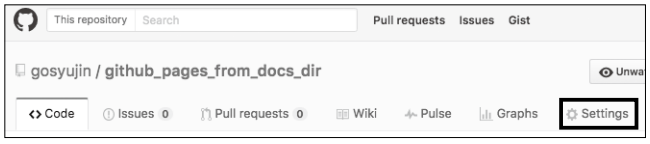
次回はSphinxドキュメントからPDFを作成する方法を紹介していきます。SD

▼図10　sphinxtogithub導入時のmake htmlログ

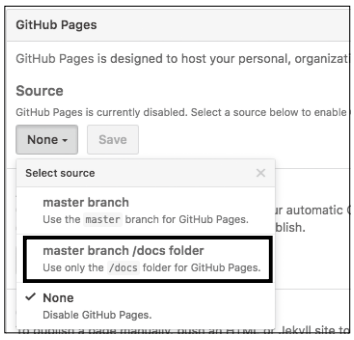
```
$ make html
sphinx-build -b html -d build/doctrees   source build/html
(..中略..)
build succeeded.
Renaming directory '_sources' -> 'sources'
Renaming directory '_static' -> 'static'

Build finished. The HTML pages are in build/html.
```

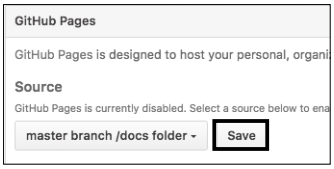
▼図11　プロジェクトのリポジトリを開き、Settingsリンクを選択する



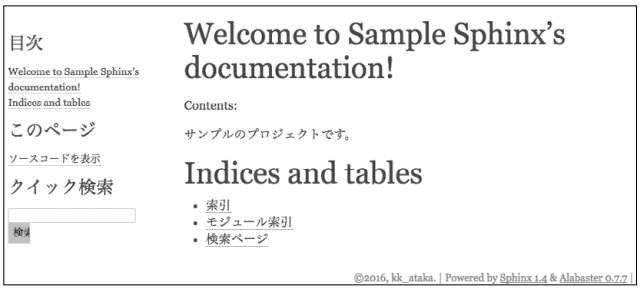
▼図12　GitHub Pagesの Sourceを変更する



▼図13　GitHub Pagesの Sourceを変更し、設定を保存する

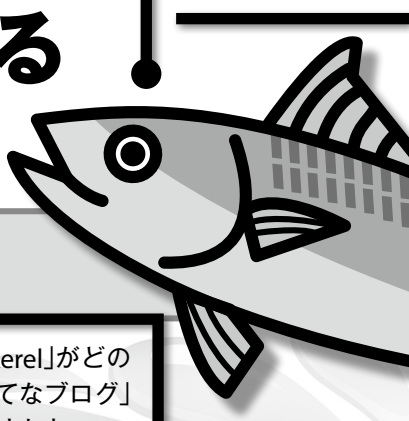


▼図14　GitHub Pages上で正しく表示された



Mackerelではじめるサーバ管理

Writer 杉山 広通(すぎやま ひろみち) ㈱はてな
Mail sugiyama88@hatena.ne.jp



最終回 Mackerelの生い立ちから思想、今後について

20回に渡ったMackerel連載。最終回では、サーバ監視ツール「Mackerel」がどのような目的・思想で開発されているのかを、「はてなブックマーク」「はてなブログ」を支えてきたインフラ管理のノウハウ、昨今の開発事情とともに振り返ります。

前回はAmazon Web Services(以下、AWS)上の各種サービスを簡単に監視するためのしくみである「AWSインテグレーション」を紹介しました。今までの連載では、Mackerelの基本的な使い方からAPIやコマンドラインツール「mkr」を用いた少し進んだ使い方、プラグインの書き方、ユーザの活用事例などを紹介してきました。

今回は本連載の最終回となるため、Mackerelの生い立ちや思想、今後のビジョンについて紹介します。

Mackerelの生い立ち

動的で複雑なインフラ環境

仮想サーバやコンテナ技術、サーバレスアーキテクチャなどをはじめとするインフラストラクチャ・プラットフォームは、日進月歩の進化を遂げ、ますます便利になっています。おかげで従来では考えられないようなスピードでサービスを立ち上げ、ビジネスを展開できるようになりました。一方、この動的なインフラ環境は複雑性もはらむため、適切に維持管理する難しさも生じています。

はてなのインフラ環境(自作サーバから自作コンテナまで)

はてなは、「はてなブックマーク」や「はてなブログ」などの主力サービスに加えて、受託開発サービスやラボサービスも運営しており、たくさんのサービスが存在します。また創業16年の歴史を持つため、レガシーなものから先進的なものまで、たくさんのプラットフォームを同時にサポートする必要もあります。

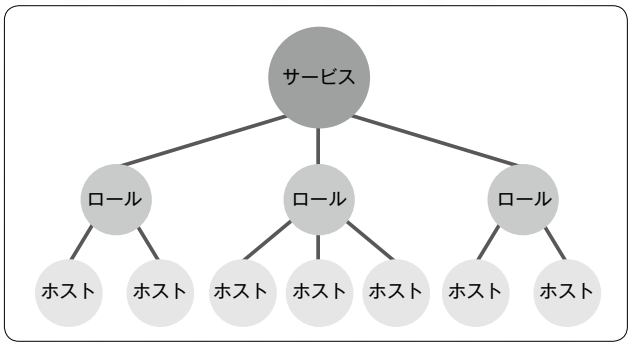
はてなでは、インフラ環境を自社のビジネスへ適合させるため、自作サーバやXenを活用した仮想サーバ、AWSなどのパブリッククラウド、Dockerとchrootの考え方を組み合わせた「droot」[注1]と呼ばれている自作コンテナなどを活用してきました。

効率的な管理を促進するサービス・ロール・ホストの概念

このようにはてなには、数十のサービスを支えるための数百の役割を持った数千のホストが存在します。そのため、非常に複雑かつ膨大な組み合わせへの効率的な対処が必要となります。そこではてなでは、サービス・ロール(役割)・ホストという普遍的で汎用的な管理概念を一貫して採用してきました(**図1**)。非常に大規模で、ともすれば複雑で管理の難しいインフラ環境ですが、この一貫した概念ですべてのサービスを管理しているため、迷いがなくてわかりやすく、さまざまな利便性を得ることができています。結果的に、数名のインフラエンジニアでミドル

注1) https://github.com/yuuki/droot

▼図1　サービス・ロール・ホスト



▼表1　代表的なDevOpsツール

カテゴリ	ツール
モニタリング	Mackerel、NewRelic、Nagios、Zabbix
インフラ構成管理	Puppet、Chef、Ansible、Salt
インフラテスト	Serverspec、InSpec
デプロイ	Capistrano、Fabric
CI	Jenkins、CircleCI
開発フロー管理	GitHub、Bitbucket

ウェアから下のレイヤを効率的に管理することに成功しています。もちろんこの概念は、Mackerelの基本的な管理概念へも採用されています。

Mackerelの思想

DevOpsライフサイクルと支援ツールの関係

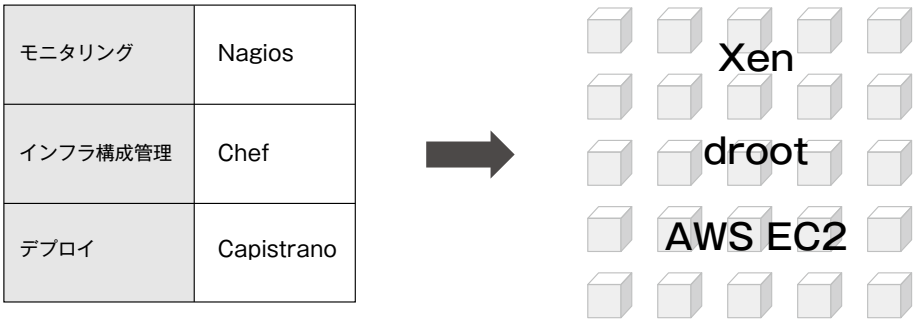
ビジネスのスピードを牽引するため、昨今のDevOpsライフサイクル[注2]はより高速化する一方で、日に何度もこのライフサイクルを繰り返すサービスもあたりまえになってきました。このように開発と運用がより短時間で持続的なライフサイクルを成すためには、優れたアーキテクチャや整備された体制はもちろん、DevOpsを支援するためのツールも必要となります。

動的であるがゆえの難しさ

表1のように、DevOpsを取り巻く環境には多種多様な効率化ツールやサービスが存在しています。汎用的にベストな組み合わせなどなく、取り扱うインフラ環境や目指す思想、担当者の趣向などにより最適な組み合わせは千差万別となります。

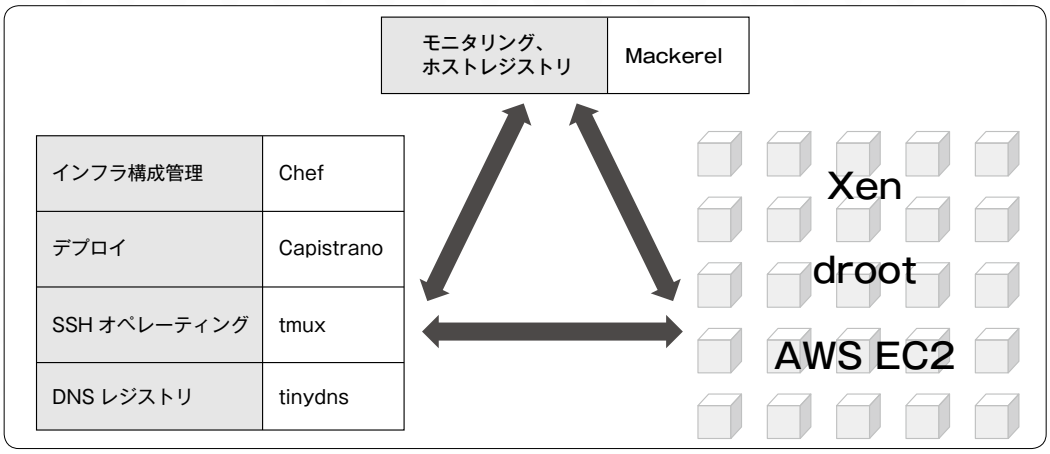
そのため、個別ツールごとの静的な作法に委ねてオンプレ環境やクラウド環境をまたいだ動的なインフラストラクチャを管理することは、非常に困難です(**図2**)。たとえば、動的に更新されるホストなどのインベントリ情報を個々のツールで正確に維持管理しなければならないからです。

▼図2　課題のあるツールの組み合わせ



注2）フィードバック→開発→テスト→デプロイ→監視フェーズなどの永続的なループ。

▼図3　Mackerelで集約したツールの組み合わせ



Mackerelを中核とした疎結合で柔軟な連携

はてなではこの問題への対処として、ホストなどのインベントリ情報をMackerelへ動的に一元集約して管理してきました。これにより、従来では個々のツールで保持しなければならなかったホストなどのインベントリ情報を、便利で柔軟なAPIを介してMackerelから動的に取り出せるようになりました。個々のツールは疎結合な状態を維持できるため、ベストなツールを制約なく組み合わせて活用することができています。

このようにMackerelを活用することで、動的で複雑になるインフラ環境を簡単にわかりやすく管理できます。現在のMackerelは進化を重ねた第四世代で、サーバ監視サービスとして監視やメトリック可視化などの機能を主軸として提供しつつ、動的な環境情報をより高精度に把握して有効活用できるようになっています。

たとえば**図3**のように、自動的にDNSへレコードを登録したり、SSHでサーバへ接続するときもロールを指定するだけでロール全体に一斉にログインできるようにしたりと、アイデアしだいで効率化のレベルを高められます。

Mackerelはユーザからの要望を第一に考え、はてなの大規模な環境でドックフーディングをしながら、常に進化[注3]しています。

Mackerelへのホストレジストリ

Mackerelでの監視やホストレジストリの流れは非常に簡単です。

①ホスト起動時に必ずサービスとロールをセットしてMackerelエージェントを起動(最低限必要なことはこれだけですが、加えて任意のKey-ValueをMackerelへ登録することもできます)
②Mackerelにホスト情報が登録され、あらかじめ定義した監視が始まる
③APIやコマンドラインツールの「mkr」を利用し、Mackerelから動的に情報を取り出して別のツールで再利用(JSON形式で取得できるため可読性が高くjq[注4]などのフィルタで再利用しやすい)

"Infrastructure as Code"の実践

"Infrastructure as Code"はインフラをコー

注3) Mackerelはサービス開始以来、100週以上連続で新しい機能をリリースし続けています。
注4) https://stedolan.github.io/jq/

ドで宣言的に記述して管理することで、今までソフトウェアの開発で何十年と培ってきた有効性の高いさまざまなベストプラクティスを、インフラへも適用するという考え方です。

おもな活用メリットとしては、動的なインフラへ対処しやすくなることはもちろん、迅速かつ確実に同じ変更が何度でも適用可能になること、障害からの復旧時間の短縮、コードレビューによるサービス品質の向上など、駆使することで属人性のある単純作業から離れ、価値あることにより時間が割けるようになります。

Mackerelが実現してきたことは、まさにこの考え方に基づくもので、Infrastructure as Codeをより促進できるようMackerel自体の監視設定もコードで管理できるようにしています。

たとえばMackerelの監視設定をGitHub上でコード管理してPullRequestで変更レビューしながら監視設定を変更したり、CircleCIでGitHub上のコードとMackerelの監視設定との整合性を定期的にテストし、結果をチャットサービスのSlackへ通知することもできます(**図4**)。

このようにして監視設定をコードで管理して可視化することで、「Dev」エンジニアがインフラの構成を把握し、PullRequestで「Ops」エンジニアへ変更リクエストを出したりできるため、開発と運用をより密接な関係にし、サービス品質向上へもつなげられると考えています。

Mackerelの今後

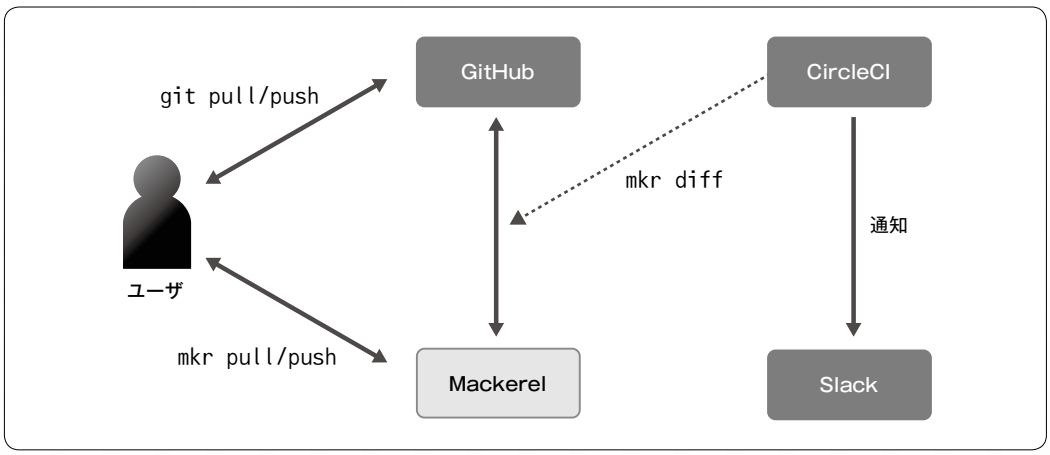
Mackerelは、サーバ監視やホスト管理を主軸としながら、Infrastructure as Codeの実践を可能にするサービスです。そこには、今まではてなが培ってきたインフラ哲学が集大成されており、限られたリソースで効率よく持続的に管理するためのノウハウが詰め込まれています。

直近ではホストレジストリの強化として、任意のKey-ValueをMackerelへ登録できるようになりました。たとえば、導入されているパッケージ名称とバージョン番号を登録しておくことで、脆弱性の発覚したパッケージ名称と特定のバージョン以下の組み合わせで、該当ホストを抽出して一気に対策するような使い方が考えられます。

今後は、異常検知など今まで人手では難しかったことをアルゴリズムを駆使して通知できるようにしていこうとしています。もちろん、従来からの監視機能やメトリック情報の可視化なども引き続き強化していきます。また、サービスの成長に合わせた次世代プラットフォームへの移行も検討しており、ますます目が離せない存在となります。

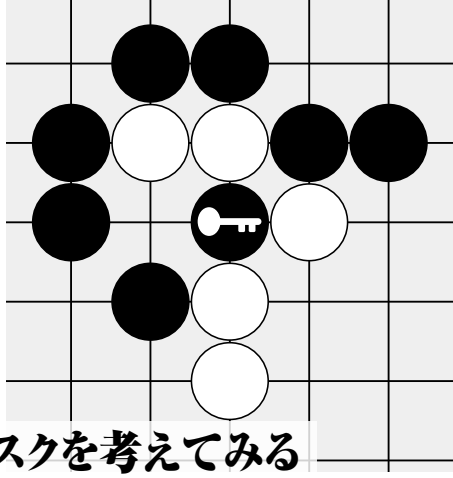
最後になりましたが、全20回にも渡りご愛読いただいたみなさんに、御礼申し上げます。ぜひ、これからのMackerelにご期待ください。SD

▼図4　mkrコマンドによるMackerelのコード管理とGitHub・CircleCI・Slack連携



セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第三七回】ゼロデイ攻撃と脆弱性公開のリスクを考えてみる

今回は、2016年9月に公開されたMySQLの脆弱性（CVE-2016-6662）の話題です。これは、MySQLの設定をバイパスすることで、MySQLのroot権限が入手できてしまうというものです。ただ、この脆弱性はそのしくみや影響はさることながら、情報公開のタイミングや、公開のリスクについても考えさせられるものがありました。

MySQL

Oracle MySQL（以下、MySQL）は非常に多く使われているオープンソースのリレーショナル・データベース[注1]です。

Webサービス開発でのキーワードLAMP（Linux、Apache、MySQL and PHP）からわかるように重要なソフトウェアです。WordPress、phpBB、MyBB、Drupalといった広く利用されているコンテンツマネージメントシステムのプラットフォームに採用されているだけではなく、Google、Facebook、Twitter、YouTubeといったインターネットのサービスでも利用されているデータベースであり、またAmazon EC2といったクラウド環境でも提供されているデータベースでもあります。

最初、スウェーデンのMySQL AB社が開発していましたが、2008年にMySQL ABがSun Microsystems社に買収され、さらにそのSun Microsystemsが2010年にOracle社に買収されました。そのため、現在ではOracle MySQLという形で、Oracle社が著作権および商標権を持っています。ライセンスはGPL v2とプロプライエタリのデュアルライセンスになっています。

MySQLのアップデートやコードのマージ、そして配布に関しては最終的にOracleが行う形で現在は運用されています。

これ以外にもMySQLから派生したデータベースがあります。今回関係しているMariaDBとPercona Serverは、MySQLから派生したデータベースです。MariaDBはMySQLのオリジナル作者による派生で、Percona ServerはPercona社が提供するMySQL互換のデータベースで、いずれもオープンソースです。

MySQLの脆弱性 CVE-2016-6662

CVE-2016-6662の脆弱性について、まずはMITRE社のCVE（Common Vulnerabilities and Exposures：共通脆弱性識別子）のサイト（**図1**）から確認し、手短に概要を書いてみたいと思います。

影響するバージョン範囲

- MySQL： 5.5.52、5.6.x～5.6.33、5.7.x～5.7.15
- MariaDB： 5.5.51以前、10.0.27以前の10.0.x、10.1.17以前の10.1.x
- Percona Server： 5.5.51-38.1以前、5.6.32-78.0以前の5.6.x、5.7.14-7以前の5.7.x

脆弱性の内容

- 一般ユーザがgeneral_log_fileのしくみを利用す

注1） 本誌の読者のみなさんには、いまさら説明は必要ないかもしれませんが。

ると、一般ユーザには書き換えられないはずのmy.cnfを書き換え、任意の設定ができる。この点を悪用して、malloc_libを入れ替え、それによりデータベースのroot権限の入手や、任意のデータベースの命令の実行が可能になる

mallocライブラリの入れ替え機能を悪用

CVE-2016-6662の脆弱性の詳細について、発見者であるDawid Golunski氏の説明[注2]を参考に解説します。

MySQLは、プログラム中で動的にメモリを確保する関数malloc()のライブラリを選択する設定が可能です。このような選択ができるようにしている理由は、実装されたmalloc関数の性能により、それを利用しているソフトウェア（この場合はMySQL）の実行性能が大きく異なってくるからです。

たとえば、GNU/Linux系ではglibcで採用されているptmalloc（dlmalloc）を、FreeBSD系ではjemallocを採用しています。ほかにもGoogleが実装したTCMallocなど、mallocのライブラリにはいろいろな実装があります[注3]。どのようなmallocの実装を導入して使うかは、状況に合わせてケースバイケースと言えます。

mallocはメモリを提供する関数であると同時に、その確保した領域を不正に書き換えるとバッファオーバーフローから任意のコードを実行させることができたり、あるいはアクセスできないはずの情報にアクセスし情報流出などが起こったりすることになります[注4]。このようなことが、たとえばOpenSSL Heartbeat Buffer Over-Read（通称Heartbleed）のような脆弱性として現れています。

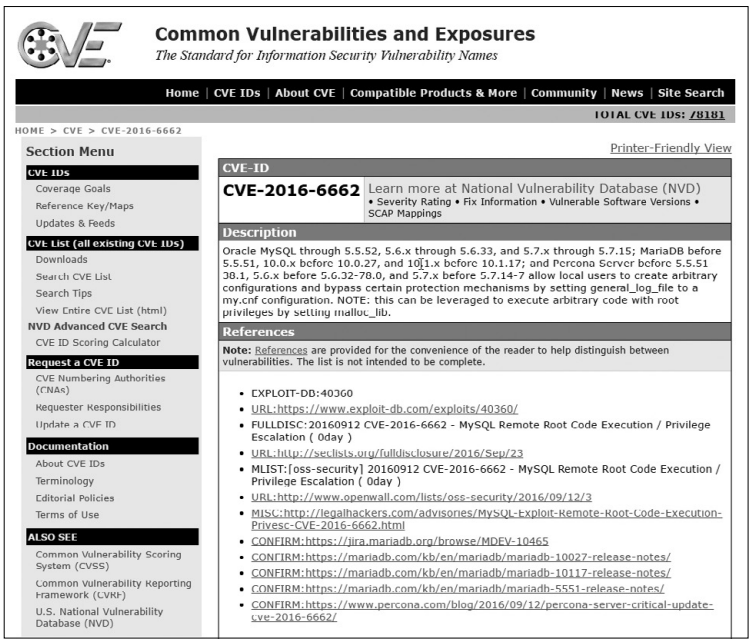
もしこのmallocライブラリを、意図的に情報流出や任意のコードを実行するようなメカニズムを組み込んだ特殊なライブラリに置き換えることができたならば、悪意による利用が可能になります。

この2つのこと、つまり「MySQLはmallocライブラリを入れ替えられる」「正規のmallocではなく、情報流出や任意のコード実行ができるライブラリを用意する」ということができたならば、もう情報は安全に管理できなくなります。

設定ファイルを書き換える

mallocライブラリを入れ替えるには、MySQLを

◆図1　CVEサイトにおけるCVE-2016-6662のページ（https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6662）



注2）http://legalhackers.com/advisories/MySQL-Exploit-Remote-Root-Code-Execution-Privesc-CVE-2016-6662.html

注3）もしプログラマでメモリアロケーションに興味があればIBMの「メモリー管理の内側」（以下のURL）を参照すると良いでしょう。とくに、このページには有用な参考文献へのリンクがたくさん用意されており、さらに詳しく知ることができます。
https://www.ibm.com/developerworks/jp/linux/library/l-memory/

注4）メカニズムに関しては本連載の第13回「動的メモリアロケーションの落とし穴」（本誌2014年9月号）で説明しています。

稼動させる際に使う設定ファイルでライブラリのパスを指定する必要があります。通常はCentOSであれば/etc/my.cnfです。Debianであれば/etc/mysql/mysql.cnfおよび/etc/mysql/mysql.conf.d/以下に用意されている設定ファイルです。

もし、これらの設定ファイルをMySQLの実行権限で書き換えることができると、今回の脆弱性が発現します。ただし、CentOSにしてもDebianにしてもディストリビューションでのMySQLデフォルト設定ファイルの所有者はrootです。ですので、とくに余計なことをしなければ今回の問題は発生しません。

ですが、たとえば自分でMySQLをビルドしてインストールしたりするときにmysqlというUIDを作成し、MySQLの参照するファイル類をmysqlの権限で読み書きできるようにしてしまった場合とか、あるいは自己流の管理をして、ディストリビューションのデフォルトの所有者からMySQLを動かしている権限に変更した場合には、この脆弱性が現れることになります。

SQL文をログに記録する機能を悪用

ここからがなるほど、と思う部分です。

①MySQLにはgeneral_logという入力したSQL文をすべてログに記録する機能がある
②general_log_fileで指定した設定ファイルを変更することができる

◆図2　general_logで設定ファイルを変更する

```
mysql> set global general_log_file = '/etc/my.cnf';
mysql> set global general_log = on;
mysql> select '
    '>
    '> ; injected config entry
    '>
    '> [mysqld]
    '> malloc_lib=/tmp/mysql_exploit_lib.so
    '>
    '> [separator]
    '>
    '> ';
1 row in set (0.00 sec)
mysql> set global general_log = off;
```

③これらはmysqlのコマンドラインから入力することができる

これを組み合わせると、次のようなことが行えます。まずはDawid Golunski氏の説明にある実行結果をそのまま載せます(**図2**)。

図2を実行することにより、ファイル/etc/my.cnfに追加の形で、次の入力が書き込まれます。

```
; injected config entry

[mysqld]
malloc_lib=/tmp/mysql_exploit_lib.so

[separator]
```

これで、次回MySQLを立ち上げたとき、自動的に/tmp/mysql_exploit_lib.soの中にあるmallocが使われることになります。データベースの管理者権限の情報を抜き取るなり、任意のプログラムを動かすなりの情報を盗むことを目的とする実装をしたmallocが現れるのも時間の問題でしょう。

繰り返しますが、通常のディストリビューションでは、設定ファイルは所有者がrootになっており、このような問題は発生しません。ただし、このような問題があるというのを知っていて、そうしているわけではありません。この点においては予期せぬ脆弱性だと思います。

脆弱性の対応と公開／告知

技術的にはなるほど、と思います。ですが、この脆弱性が抜け道となり世の中で使われているMySQLデータベースから次々と情報が盗まれる、というレベルでもありません。深刻な脆弱性であるにせよ、不幸中の幸いというべきで影響は限定的だと言えるでしょう。

さて、今回はその技術的な問題よりも、この脆弱性の公開について考えさせられる点がありました。

この脆弱性の発見者であるDawid Golunski氏によれば、脆弱性に関する情報は2016年7月29日時点でOracleのセキュリティチームに

報告がなされているそうです。またPercona ServerとMariaDBにも同時に連絡しているそうです。

Percona ServerとMariaDBに関しては8月30日までにセキュリティパッチが提供されていました。一方で、OracleはMySQLのパッチを定例パッチとして10月18日に公開するスケジュールに組み入れました。

- **Percona ServerとMariaDBのパッチは公開されており、MySQLの脆弱性は簡単にわかってしまうためゼロデイ攻撃の可能性が高まった**
- **40日ルール[注5]を適用できる期日がきていた**

この2つの理由により、Dawid Golunski氏は今回の脆弱性を公開することに踏み切ったそうです。

筆者が確認した範囲では、設定ファイルがroot所有ではなくmysqlのプロセスから書き込めるような設定にしているディストリビューションは見つかりませんでした。自分でわざわざ変更するか、あるいはとてもたいへんな思いをしてスクラッチからビルドするといった場合においてのみ、今回の問題は発生することになります。

デフォルト設定で問題が発生することは防げており、それを勘案するならば、Oracleが今回の脆弱性対応を定例アップデートに回したというのも、それほどおかしなことではないと思います。

繰り返し述べているように、この脆弱性情報を公開しても、影響を受けるのは利用者自ら変更を加えているようなケースです。そのような人たちは技術的に一定の理解ができる人たちであり、今回の脆弱性情報に接する機会も大きい、と考えることもできます。だから、ゼロデイ攻撃の可能性がある時点で公開するというのも一理あります。

ただし、すでに次の対応リリースの日付も決まっている中で、40日ルールを適用するというのもちょっと違う気がします。同じ機能だとしてもソフトウェアごとの利用の広がりを考慮に入れる必要はある、と筆者は思っています。

とくに利用範囲の大きいソフトウェア、たとえばGNU/Linuxのディストリビューションで必ず採用されているようなソフトウェアなら、複数のディストリビューション経由でセキュリティのアップデートが進むわけですから、その分、調整もたいへんになり40日でも足りないケースはたくさん出てくるはずです。また足並みをそろえるのに定例アップデートのタイミングにしてルーチーン作業としてスムーズに進めるのも悪い考えではありません。

リスクとベネフィットをどう考えるのか

脆弱性を公開することには当然ながらリスク（危険性）とベネフィット（恩恵）が存在し、そのバランスがどうなっているのかを考えなければなりません。そして、そのバランスをどう考えるかは10人いれば10人違うでしょう。リスクを取るにしても、ベネフィットを取るにしても、どちらかの極端なケースでない限り、もしかするとそんなに違いはないのかもしれません。

脆弱性の公開に関しては、これまでも脆弱性が見つかった時点ですべての情報を公開すべきであるという考え方の「フルディスクロージャ」を主張している人たちはいますし、またその反対に「寝た子を起こすな」という人たちもいます。一筋縄ではいかないですし、もちろん「定石」となる答えも見つからないでしょう。とにかく脆弱性の発生件数は多く、次から次へと問題が降ってきます。それを現実的に解決していかなければいけません。

ただひとつ言えることは、毎回、リスクとベネフィットを考えて行動することです。個々の問題を見てみると、同じような問題なのに対応が異なるというケースがあったとしても、そのような経験値を積んでいくことで、ベストではないにしろ、うまく回していけるようになるのではないかと考えています。

みなさんも、このリスクとベネフィットをどう捉えてどう考えるべきなのか一緒に考えてみませんか？ SD

注5） 脆弱性が届けられて40日経っても脆弱性対応がなされない場合、ゼロデイ攻撃を勘案して脆弱性を公開する。

SOURCES

レッドハット系ソフトウェア最新解説

第4回

Red Hat OpenShift Container Local part2

前回はOpenShift Container Localの概要から簡単なアプリ作成まで紹介しました。今回はテンプレートの利用やアプリケーションのリリース管理を紹介していきます。

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

テンプレートの利用

OpenShiftは、アプリケーション作成の雛形として利用できるテンプレートを備えています。このテンプレートには、アプリケーションのビルド／デプロイ時の設定、アプリケーションへのアクセスポイント、利用するソースコードやデータベース、アプリケーション作成時に指定できるパラメータなどの情報が含まれています。デフォルトで提供しているテンプレートについては、「oc get template -n openshift」で確認できます。なお、OpenShiftでは「openshift」という特別なプロジェクトを持っており、openshiftプロジェクトに登録されたテンプレートはOpenShiftの全ユーザが利用できるように設定されています。openshiftプロジェクトにあるテンプレートを確認するために、「-n openshift」オプションを指定しています(**図1**)。

確認したテンプレートを利用して、開発者はアプリケーションを作成できます。前回は簡単なNode.jsアプリケーションを作成しましたが、今回はMongoDBデータベース付きのテンプレート「nodejs-mongodb-example」を利用してNode.jsアプリケーションを作成してみます。

このテンプレートはWelcomeページのアクセス回数を、MongoDBに保存するNode.jsアプリケーションを作成するためのものです。ただし、このMongoDBはコンテナ上に存在するため、脆弱性やバグ修正などでMongoDBのコンテナを修正・再デプロイすると、コンテナ内のデータ領域に保存されたアクセス回数が消去されてしまいます。そこで、そのようなコンテナの再デプロイの場合にもデータが消えないよう、外部ストレージの利用を考えてみます。OpenShiftの基盤技術であるDocker/Kubernetesでは、利用可能な外部ストレージを永続ボリューム(PV：Persistent Volumes)として定義し、ユーザが必要に応じて永続ボリュームを利用する(PVC：PersistentVolumeClaims)ための設定を行いますが、OpenShiftでも同様の作業を実施することで外部ストレージ[注1]を利用できます。

▼図1　OpenShiftのテンプレートの確認とNode.jsの作成

```
$ oc get template -n openshift
NAME
  ……中略……
nodejs-example
nodejs-mongodb-example
$ oc new-app --template=nodejs-mongodb-example
```

永続ボリュームの利用

OpenShiftでは、PVの情報はデフォルト

注1) URL http://red.ht/2d4OSPx

でadminユーザしか見られないようになっていますので、まずはadminユーザでログインします。OpenShift Container Localが提供するCDKでは、デフォルトでPVが3つ(pv01, pv02, pv03)設定されています。この3つのPVには、CDKのNFSによるディレクトリのエクスポートを割り当てています。こうした情報は、「showmount」や「oc get pv」で確認できます。

PVCは用意されていませんので、PVCを作成するためのpvc01.yamlファイルを作成します。pvc01というPVCが、pv01を1Gi分利用するという設定をYAML形式で記載しています。PVやPVCの文法などの詳細な解説は、Kubernetesの公式ドキュメント[注2]に記載されていますので、そちらもあわせて参照ください。最後にpvc01.yamlを利用して、PVCを作成します(**図2**)。

作成したPVCを利用して、PVをMongoDBのコンテナに接続してみます。ちなみに、PV接続などといったアプリケーションのデプロイに関連する設定は、OpenShiftではDeployment Config(以降、DCと記載)というオブジェクトの中で設定することになります。DCの一覧を「oc get dc」で確認できますので、MongoDBに関するDCにPVを追加するためのコマンド「oc volume dc/mongodb」を実行して、PVをMongoDBコンテナに接続します。この例では、マウント先をMongoDBのデータ保存場所に指定しています。コンテナからPVを外すときは、--removeオプションを指定します。このとき、PV接続時に指定した名前(この例ではv1という名前)を指定する必要があります(**図3**)。

▼図2 PVCの作成手順

```
$ oc login -u admin -p admin
$ showmount -e
Export list for rhel-cdk:
 ……中略……
/nfsvolumes/pv01 *
$ cat <<EOF > pvc01.yaml
apiVersion: "v1"
kind: "PersistentVolumeClaim"
metadata:
    name: "pvc01"
spec:
    accessModes:
        - "ReadWriteMany"
    resources:
        requests:
            storage: "1Gi"
    volumeName: "pv01"
EOF
$ oc create -f pvc01.yaml
```

ちなみに、PV接続などの作業をテンプレート化することもできます。その場合は、既存テンプレートの修正をすることでより効率化できます(**図4**)。こうしたカスタムテンプレートの作成機能を使用すると、アプリケーションの環境を使い回せるので、開発／テスト／本番環境で同じ環境を利用できるようになります。

Podのコンテナの起動と再起動

Kubernetes/OpenShiftは、コンテナをPodという単位で制御します。Podには単数または複数のコンテナが入っています。デフォルトのテンプレートには、Podが1個起動する(Podの数は動的に変更可能)ように制御される設定が含まれていますので、「oc get pod」で確認したMongoDBのPodを「oc delete pod」で消去すると、自動的にMongoDBのPodが再起動され

▼図3 MongoDBのコンテナへのPV接続と切断

```
$ oc volume dc/mongodb --add --name=v1 -t pvc --claim-name=pvc01 --mount-path=/var/lib/mongodb/data
$ oc volume dc/mongodb --remove --name=v1
```

▼図4 OpenShiftのカスタムテンプレートの作成

```
$ oc export template nodejs-mongodb-example -n openshift > nodejs-mongodb.yaml
$ oc create -f nodejs-mongodb-persistent.yaml※注3 -n openshift
```

注2) URL http://kubernetes.io/docs/user-guide/persistent-volumes/
注3) URL nodejs-mongodb.yamlを修正したもの(https://git.io/viFtA)

る様子を確認できます。Podが再起動しても、MongoDBのデータがPVに保存されている場合、コンテナ再デプロイによるデータ消去が発生しないことも確認できます。

```
$ oc delete pod/mongodb-1-PODNAME_1
```

アプリケーションのリリース管理

OpenShiftは、アプリケーションを作成する基となるDockerイメージを管理する機能を持っています。たとえば前述したNode.jsアプリケーションを開発していくと、OpenShift内部のレジストリに複数のバージョンのNode.jsやMongoDBのDockerイメージが保存されていくことになります。要件によっては、開発環境はバージョン10、本番環境はバージョン5のDockerイメージを利用するなど使い分けをする場合があることも考えられます。その場合は、OpenShiftのDockerイメージ管理機能であるImage Stream(IS)を利用することになります。

Dockerイメージ管理機能「Image Stream」

Image Streamの実体は、Dockerイメージを参照するための情報(タグ)の集合体となります。OpenShiftが参照するDockerイメージ[注4]は、SHA256のユニークな64桁のハッシュ値によって識別されており、各ハッシュ値に付けられたタグがOpenShiftのアプリケーション作成時に利用されています。タグには任意の名前を付けられますが、latest、2.6、2.4……などバージョン番号を表すものを利用することが一般的です。OpenShiftではこうしたlatestなどのタグをnodejsやmongodbなどの名前でグループ化し、Image Streamとして管理しています。たとえばCDKのデフォルトでは、mongodbに関するImage Streamは「`oc get is mongodb -n openshift`」を実行することで、「`2.4, 2.6, latest`」の3つのタグがあることが確認できます(**図5**)。ちなみに、一度タグを付けられたDockerイメージは、そのタグが別のDockerイメージに付けられてもImage Streamに参照情報が残りますので、「`oc describe`」でDockerイメージの履歴として確認できます。

OpenShiftアプリケーション作成

それでは、リリース管理を意識したOpenShiftのアプリケーション作成を実施してみます。まず、openshiftプロジェクトのImage Stream「mongodb」の最新版をテスト環境用のデータベースとしてデプロイしてみます。作業場所となるtest01プロジェクトを作成し、test01プロジェクトからopenshiftプロジェクトのImage Stream「mongodb」の最新版を、test01プロジェクトのImage Stream「testdb」の最新版として利

▼図5　ImageStreamの確認

```
$ oc get is mongodb -n openshift
NAME DOCKER REPO                                    TAGS            UPDATED
mongodb     172.30.202.54:5000/openshift/mongodb    2.4,2.6,latest  18 hours ago
$ oc describe is mongodb -n openshift
  ……中略(出力結果を読みやすいように書き換えています)……
Tag     Created         PullSpec※注5
latest  3 hours ago     REG_NAME_1/rhscl/mongodb-26-rhel7@sha256:HASH_NUM_3
2.6     3 hours ago     REG_NAME_1/rhscl/mongodb-26-rhel7@sha256:HASH_NUM_3
2.4     1 hours ago     REG_NAME_2/openshift/mongodb@sha256:HASH_NUM_2
<none>  3 hours ago     REG_NAME_1/openshift3/mongodb-24-rhel7@sha256:HASH_NUM_1
```

注4) 外部のDockerリポジトリで配布しているDockerイメージの参照もできます。
注5) PullSpecはDockerイメージを実際にpull(取得)する場合に利用される、`docker pull`の引数です。OpenShiftのアプリケーション作成時に、Dockerイメージを取得するために利用します。

▼図6　Image Stream「mongodb」を利用したアプリ作成

```
$ oc new-project test01; oc project test01
$ oc tag openshift/mongodb:latest testdb:latest
$ oc new-app -f nodejs-mongodb-persistent-tag.yaml※注6
```

▼図7　開発環境用のデータベースイメージの作成

```
$ mkdir works
$ cat <<EOF > ./works/Dockerfile
FROM 172.30.202.54:5000/test01/testdb:latest
MAINTAINER hkojima
RUN touch /var/lib/mongodb/renewal01
EOF
$ docker build -t 172.30.202.54:5000/test01/devdb:latest works
$ docker push 172.30.202.54:5000/test01/devdb:latest
```

▼図8　Dockerイメージの切り替え

```
$ oc tag devdb:latest testdb:latest
$ oc rsh mongodb-2-PODNAME_2
sh-4.2$ ls /var/lib/mongodb/
data  renewal01
```

用できるように「oc tag」でタグ付けします(**図6**)。タグ付けすると、Image Stream「mongodb」のlatestタグが参照するDockerイメージと、Image Stream「testdb」のlatestタグが参照するDockerイメージが同じものになります。そして、「oc new-app」で「testdb」を利用してアプリケーションを作成します。

次にテスト環境用のデータベースとしてデプロイしたMongoDBのDockerイメージに変更を加えるDockerfile(ここではrenewal01という空ファイルを追加)を作成し、開発環境用のデータベースとして利用する新しいDockerイメージを「docker build」で作成します。このとき、「-t」オプションで作成したDockerイメージにタグ付けをし、「docker push」でタグ付けをしたDockerイメージをPushすることで、test01プロジェクトに「devdb」という新しいImage Streamを作成しています(**図7**)。

そして開発環境用のデータベースのテストが完了し、テスト環境用のデータベースとしてデプロイすることを考えてみます。その場合は、Image Stream「devdb」のlatestタグが参照するDockerイメージを、「testdb」のlatestタグでも参照するように「oc tag」でタグ付けします。すると、アプリケーションが利用するMongoDBのDockerイメージが切り替わることになり、自動的にMongoDBのPodが再デプロイされることを、「oc rsh」によるPodへのログインなどで確認できます(**図8**)。renewal01ファイルが作成されていることがわかるため、テスト環境用のデータベースが開発環境用のデータベースに切り替わったことが確認できます。この自動的な再デプロイについては、テンプレート内のDCで無効にすることもできます。こうしたテンプレートの利用やDockerイメージの切り替えにより、効率的に継続的デリバリ(CD：Continuous Delivery)を実現できるようになります。

まとめ

今回はテンプレートやリリース管理について紹介してきました。なお、OpenShiftには、ほかにもアプリケーションの開発を便利にするさまざまな機能(ロールバックによる世代管理やサービス連携など)があります。そうした機能の紹介を含んだ開発者向けのハンズオン資料が公開されています[注7]ので、一度参照ください。SD

注6) URL https://git.io/viFqU
注7) URL https://github.com/nekop/openshift-sandbox/blob/master/docs/developer-handson.md

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第36回 ❖タイムスケジュールでプログラムを実行（その3）

前回はcrontab(1)を使い、タイムスケジュールに従ってプログラムを実行する方法を説明しました。Webから天気予報のデータを取得して雨が降る場合にはメールするというスクリプトを取り上げましたが、これはWebからデータを持ってきて活用する例としてほかにも応用が利くので、今回はそのあたりを説明します。また、FreeBSDにおけるメール機能（mail(1)）についても説明します。

◉著者プロフィール

後藤 大地（ごとう だいち）

BSDコンサルティング（株） 取締役／（有）オングス 代表取締役／FreeBSD committer

エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

ユーザが自分でタイムスケジュールを設定

前回は、fishで作成した天気予報通知シェルスクリプト「wa.fish」を使ってcrontab(1)を説明しました。今回はわかりやすいように、より一般的に使われている/bin/shで作り替えたものを**リスト1**に掲載します。

▼**図1 iPhoneに届いたメール**

このスクリプトを7時から22時の間に3時間おきに実行するようにcrontab -eで**リスト2**のようにスケジュールを書き込みます。必要な環境変数を設定してあるところもポイントです。これをちゃんと設定しておかないと、開発中は動作したのにcron(8)経由で実行すると正しく動かないといったことが起こります。

ここまで設定して運用を始めると、雨が降りそうになると**図1**、**図2**のようなメールが届きます。簡単な例ですが、実用性の高い利用例です。次節から内容を詳しく見ていきます。

スクリプトの内容

スクリプトの中身は大きく分けると次のように

▼**図2 Apple Watchで見た場合のメール（最近はApple Watchからできることが増えて便利）**

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

なっています。

①Webサイトからデータを取得
②データを解析
③必要がなければ処理を終了
④警告メールを送信

リスト1を詳しく見てみましょう。

まず①の、Webからコンテンツを取得する部分です。使うコマンドはcurl(1)ではなくfetch(1)でもwget(1)でもなんでも良いのですが、一番多機能で強力なcurl(1)を使うのが便利です。オプションはいろいろあります。1つポイントとして、curl(1)で取得できなかった場合には--insecure -A Mozilla/5.0 --location -getというオプションを試してみてください。取得できるサイトが増えると思います。curl(1)の説明はしませんが、これはかなり使えるコマンドです。

次に、取得したWebページ（ここではYahoo!ピンポイント天気予報のWebページ）の内容の解析をしています（②）。この部分はアドホックです。ケー

▼リスト1　雨が予想されている場合にはその旨をメールするスクリプト

```
#!/bin/sh

## wa.sh
## 天気警報情報をメールで送信

# 作業用ディレクトリの作成と削除処理の設定
tmpd=/tmp/wa_$(date +%Y%m%d_%H%M%S)_$$; mkdir $tmpd   ･･･ⓐ
trap "rm -rf \"$tmpd\"" EXIT

# 必要データ設定
url=http://weather.yahoo.co.jp/weather/jp/13/4410/13225.html
url2=http://tokyo-ame.jwa.or.jp/

# 天気予報データ取得
curl --get $url 2> /dev/null                          > $tmpd/src   ･･･①

# 天気情報を抽出
key=http://i.yimg.jp/images/weather/general/forecast/pinpoint/size40/
grep $key $tmpd/src                                   |
# 時刻データを付与                                                    ･･･②
awk 'BEGIN{i=0}{print i, $0;i=(i+3)%24}'              |
sed 's/^\([0-9]*\) .*alt="\([^"]*\)".*$/\1:\2/'       > $tmpd/tenki

# 天気予報の中に雨の予報がない場合には処理終了
grep 雨 $tmpd/tenki > /dev/null                  || exit   ･･･③

# 天気予報をメールで報告
tenki=$(cat $tmpd/tenki | tr '\n' ' ')
printf "$tenki\n\n$url\n$url2"                            |   ･･･④
mail -s "ALERT: $tenki" $USER
```

▼リスト2　crontab -eでタイムスケジュールを登録

```
SHELL=/bin/sh
TERM=xterm
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin
LANG=ja_JP.UTF-8

# 雨天注意報
0 7-22/3 * * * /Users/daichi/Documents/weather_alart/wa.sh > /dev/null
```

スパイケースで変わります。ここではgrep(1)、awk(1)、sed(1)を使っています。たいていのテキストは、このあたりのテキスト処理系のコマンドを使えばほしいデータを抜き出せます。こうしたデータの解析は1回限りのコードで、再利用は考えずにサクッと書いてしまうのがポイントです。

アドホックな処理ですが、ここでデータを抽出している処理を説明しておきます。

まず、最初のgrepで天気情報を抜き出しています。このページのHTMLでは「http://i.yimg.jp/images/weather/general/forecast/pinpoint/size40/」のキーワードでgrep(1)すると天気情報のみを抜き出すことができます。「http://i.yimg.jp/images/weather/general/forecast/pinpoint/size40/」がお天気アイコンへのパスになっており、この部分を抜き出すとちょうどお天気情報だけを抜き出すことができます。

次に、awk(1)の処理で時刻データを追加しています。HTMLのほうでは最初が0時でそこから3時間ごとの天気予報が掲載されていますので、awk(1)で同じように0時から21時までの時刻を追加しています。最後にsed(1)の処理で時刻と天気の文字列だけを抽出しています。これで天気予報データでほしい部分だけを抜き出しているというわけです。

HTMLパーサを使って解析するということもできるのですが、HTMLデータは不適切な記述になっていることも多くパーサで解析できないことがままあります。ほしいテキストデータだけを抜き出せれば良いので、パーサを使って処理するよりもここで示したようにgrep(1)で対象部分だけを抜き出して、awk(1)で編集し、さらにsed(1)で切り出しを行うといった処理が有益です。

もうちょっと厳密に処理を行う必要があるといった場合には、いったんHTMLデータから改行データをすべて削除したあと、sed(1)で必要な部分に改行を挿入して整形してから切り出し処理を行うといった方法があります。この方法でだいたいほしい部分のデータが抜き出せると思いますので、やってみてください。

次に、③の雨が降りそうなのかそうでないのかを判断する部分です。ここではgrep(1)で「雨」というキーワードがデータに入っているかを調べて、含ま

▼リスト3　/etc/aliases on FreeBSD 10.3-RELEASE

```
 ...略...
#       >>>>>>>>>>      The program "newaliases" must be run after
#       >> NOTE >>      this file is updated for any changes to
#       >>>>>>>>>>      show through to sendmail.
 ...略...
# Pretty much everything else in this file points to "root", so
# you would do well in either reading root's mailbox or forwarding
# root's email from here.

# root: me@my.domain

# Basic system aliases -- these MUST be present
MAILER-DAEMON: postmaster
postmaster: root

# General redirections for pseudo accounts
_dhcp:  root
_pflogd: root
auditdistd:     root
bin:    root
bind:   root
daemon: root
games:  root
 ...略...
```

れていない場合にはexitでシェルスクリプトを終了させています。grep(1)は、一致しなかった場合には0以外の値で終了しますので、これを利用して処理を切り分けています。||は前の処理が0以外で終わった場合に次の処理を行うためのシンタックスですので、この処理で雨が降らない場合には処理を終了する、という指定になります。

そしてここを抜けたら、④でデータをメールで送信しています。

システムからメールを送る

スクリプトでは④の部分で結果をメールで送っています。mail(1)コマンドのオプション-sで指定している内容がメールのサブジェクト、$USERがメールアドレス、printf(1)で出力してパイプに流し込んでいる内容がメールの本文です。

cron(8)はプログラムから出力があるとメールで送信しますが、ここではメールのサブジェクトを指定したいので、cron(8)からではなく、直接mail(1)コマンドを実行してメールを送信しています。

ここではメールアドレスとして$USERが使われています。自分の環境では「daichi」になるのですが、このメールアドレスの実態は/etc/aliasesに書いてあります。まず、デフォルトの/etc/aliasesを見てみましょう（**リスト3**）。メールアドレスのエイリアス一覧のようになっています。多くの項目がrootへのエイリアスになっていますので、通常はここにrootのメールアドレスを登録します。これは、ほかのアカウントに対しても有効です。たとえばここでは、次の2行を追加したシステムを使っています。

```
root:           daichi@example.co.jp
daichi:         daichi@example.co.jp
```

この例では、rootに対するメールもdaichiに対するメールも、daichi@example.co.jpに送信されるようになります。なお/etc/aliasesは編集後、

```
% newaliases ↵
```

のようにnewaliases(1)コマンドを必ず実行してください。newaliases(1)は、/etc/aliasesからデータベースを作成するコマンドです。このコマンドを実行しないと、書き換えた/etc/aliasesの内容が反映されません。

メールのサブジェクトを指定する必要はないという場合には、mail(1)コマンドを使わないでスクリプトから文字列を出力するだけでメールが送信されますので、それを使えば良いでしょう。

ちょっとしたテクニック

ⓐの処理は筆者がよく使っている実装です。知っておくとコーディング上便利だと思うので説明します。この処理で「/tmp/wa_20160911_180523_29112」といったディレクトリが作成され、「$tmpd/」のような形で使えるようになります。さらに終了時には、自動的にディレクトリごと削除されます。

このようにしておくと、foo bar > $tmpd/outのような形で一時ディレクトリにデータを書き出して作業するといったことが簡単にでき、さらに終了時にディレクトリの削除処理を気にする必要がなくなります。加えて、デバッグ時にはデータを一時ディレクトリに残しておくといったこともやりやすいので、デバッグにも利用できます。

さまざまな応用が可能

今回取り上げたシェルスクリプトとcron(8)によるタイムスケジュールは、さまざまな応用が可能です。更新されたデータをチェックしたいWebサイトからデータを取得して、解析し、結果を報告する。そしてその動作はcron(8)経由で自動的に実施させます。

天気予報のサンプルはプライベートユース向けでもありますが、これはそのまま現場作業者向けに天気情報を送信するシステムとしても使えますし、取得するデータ元を変更すれば、仕入れや販売などにも応用が利きます。さまざまなシステム構築にも利用できる基本的なテクニックです。SD

Network ManagerのVPNプラグイン

Ubuntu Japanese Team
あわしろいくや

今回はUbuntu 16.04 LTSのNetwork ManagerにVPNプラグインをインストールし、VPNサーバにアクセスする方法を紹介します。

Ubuntu 16.04 LTSのNetwork Manager

Network Managerは、今や必須のネットワーク設定ツールです。9月18日時点の最新版は1.4.0ですが、Ubuntu 16.04 LTSでは1.2.0を採用しています。通常UbuntuではLTSではあまり冒険はしないのですが、16.04では1.2.0の開発版（1.1.93）に更新し、リリース後1.2.0にアップデートしています。

Network Managerは、プラグインで対応するVPNサーバを追加できます。このプラグインが1.0.x以前と1.2.0では互換性がなくなりました。主要なプラグインはNetwork Managerとともに開発されていますが、中にはサードパーティのものもあります[注1]。Network Managerの開発はRed Hatの開発者が主体になって行っているのですが、同社のすごいところはサードパーティ製VPNプラグインにも1.2.0対応パッチを提供したところです。おかげで開発がほぼ止まっていたものの、この件がきっかけでメンテナーが交代し、ふたたび活発に開発されるようになったものもあるくらいです。

最近のVPNサーバ事情

手軽なVPNサーバといえばPPTPが長らく使われてきましたが、すでにセキュアでないということが明らかになって数年経っており、一線を退いた感はありますが、それでも対応クライアントの多さと簡便さでまだまだ使われているといったところでしょう。しかしこのたび、最新版のmacOSとiOSでサポートが打ち切られることになり、いよいよ使われなくなっていくことになりそうです。ちなみに今回PPTPサーバだけが手元で動作していなかったため、検証のために設定してみましたが、Ubuntuでも簡単に動作しました[注2]。

PPTPに代わって普及したのはL2TP/IPsecでしょう。主要OSでもサポートされています。しかし、使用環境によってはWindowsのレジストリを変更してNATトラバーサルを有効にしなければいけないとか、古いバージョンのNetwork ManagerのL2TP/IPsecプラグインは問題があってうまく接続できなかったとか、そもそもL2TP/IPsecサーバに対応したルータは高価であるとか、いくつかの問題はあります。それでも、よりこちらに流れていくのは間違いないでしょう。

UbuntuでVPNサーバを動作させたい場合は、OpenVPNがいいのではないかと思います。最近ではASUS製などのルータでも対応するようになっています。デフォルトでクライアントが対応しているOSはないので別途VPNクライアントソフトが必要

注1）https://Wiki.gnome.org/Projects/NetworkManager/VPN にすべてのリストがあります。

注2）あまりちゃんと確認していないのですが、16.04では自動起動はしないようでした。気をつけるのはそこぐらいのものです。

ですが、各種OS用が用意されているので、あまり困ることはないでしょう。

今回はPPTPサーバを除いて、検証にはSoftEther VPNを使用しました。これは各種VPNサーバと互換性があり、OpenVPNサーバやL2TP/IPsecサーバ、今回検証しなかったSSTPサーバなどとしても振る舞うことができます。SoftEther VPNサーバの設定は過去にも解説したことがありますが、あれでも不十分で、詳しくすると本連載数回分になってしまうので解説は省略します[注3]。

以上を鑑みて、今回はPPTPとOpenVPNとL2TP/IPsecサーバに接続する方法を解説します。

PPTP

Network ManagerのPPTPプラグインは、Ubuntuでは[network-manager-pptp]と[network-manager-pptp-gnome]という名前です。必ず両方必要なのですが、これだけはデフォルトでインストールされています。というわけで、これを例にして設定方法を解説します。

Unityの右上にあるネットワークインジケーターをクリックすると[VPN接続]があり、さらに[VPNを設定]というサブメニューがあるので、これをクリックします。[ネットワーク接続]が表示されるので、[Add]をクリックします。[接続の種類を選んでください]が表示されるので、プルダウンメニューから[Point-to-Point Tunneling Protocol(PPTP)]を選択し、[作成]をクリックします。

接続画面が表示されるので、最低限[接続名]と、[VPN]タブの[ゲートウェイ]と[ユーザ名]と[パスワード]だけは設定します。パスワードはそのままだと保存できないため、右側のアイコンをクリックしてメニューを表示し、パスワードの保存方法を選択します。[Store the password only for this user]にしておくのが無難でしょう(**図1**)。

よりセキュアに接続するため、[詳細]をクリックします。[MPPE暗号を使用する]にチェックを入れ、[セキュリティ]を[128ビット(最も安全)]にすれば、少しは安全にPPTPを使用できます(**図2**)。

OpenVPN

前述のとおりOpenVPNプラグインはインストールされていないので、まずはインストールします。コマンドラインで行う場合は、次のコマンドを実行してください。

```
$ sudo apt install network-manager-openvpn*
```

図1　PPTPのVPNタブの設定例

図2　PPTPの詳細設定

注3) Windowsで使う分にはさほど難しくはないのですが。

これで［network-manager-openvpn］と［network-manager-openvpn-gnome］がインストールされます。

OpenVPNのパスワード認証にはCA証明書が必要ですが、OpenVPNサーバの実装によって対応方法が異なります。ca.crtファイルが単体で存在する場合はそれを、SoftEther VPNのように拡張子がovpnのファイルの中に存在する場合はエディタなどで開いてタグの**<ca></ca>**の部分をファイル名“ca.crt”として保存します。その部分のみをコピーペーストしてもいいですし、**図3**のようなワンライナー（これはあまりいい例ではありませんが）を実行してもいいでしょう。

あとは［VPNを設定］をクリックし、［接続の種類を選んでください］で［OpenVPN］を選択し、編集画面で［ゲートウェイ］を入力し、［タイプ］を［パスワード］にし、［ユーザ名］と［パスワード］を入力して［CA証明書］で先ほど生成したca.crtを指定し、保存します（**図4**）。これはあくまでSoftEther VPNがサーバの場合の一番簡単な方法であり、OpenVPNの実装によっては設定方法が異なることもあります。

L2TP/IPsec

L2TP/IPsecプラグインはUbuntuのリポジトリにはないため、自力でビルドする必要があります。ビルド方法は**図5**をご覧ください。

もしかしたら本誌が発売されるころにはPPAなどが用意されているかもしれないため、該当のGitHubリポジトリ[注4]を確認してみてください。

L2TP/IPsecの場合、通常のVPNタブの設定（**図6**）のほか、プレシェアードキー（事前共有鍵）を設定する必要があります。このプラグインの場合は、［IPsec Settings］をクリックして［Enable IPsec tunnel

注4）https://github.com/nm-l2tp/network-manager-l2tp

図4　OpenVPNのVPNタブの設定例

図3　ワンライナーの例

```
$ awk '/<ca>/,/<\/ca>/' /PATH/TO/FILE.ovpn | sed -e '/<[^]*ca>/d' -e '/^.$/d'>ca.crt
```

図5　L2TP/IPsecプラグインのビルド方法

```
$ sudo apt install git
$ git clone https://github.com/nm-l2tp/network-manager-l2tp.git
$ cd network-manager-l2tp
$ sudo apt install intltool libtool network-manager-dev libnm-util-dev libnm-glib-dev libnm-glib-vpn-dev ➥
libnm-gtk-dev libnm-dev libnma-dev ppp-dev libdbus-glib-1-dev libsecret-1-dev libgtk-3-dev libglib2.0-➥
dev xl2tpd strongswan
$ sudo ln -s /etc/apparmor.d/usr.lib.ipsec.charon /etc/apparmor.d/disable/
$ sudo apparmor_parser -R /etc/apparmor.d/usr.lib.ipsec.charon
$ sudo ln -s /etc/apparmor.d/usr.lib.ipsec.stroke /etc/apparmor.d/disable/
$ sudo apparmor_parser -R /etc/apparmor.d/usr.lib.ipsec.stroke
$ sudo systemctl disable xl2tpd.service
$ ./autogen.sh
$ ./configure --disable-static --prefix=/usr --sysconfdir=/etc --libdir=/usr/lib/x86_64-linux-gnu ➥
--libexecdir=/usr/lib/NetworkManager --localstatedir=/var --with-pppd-plugin-dir=/usr/lib/pppd/2.4.7 ➥
--enable-absolute-paths
$ make
$ sudo make install
```

to L2TP host]にチェックを入れ、[Pre-shared key]にプレシェアードキーを入力します(**図7**)。[PPP Settings]は、PPTPの[詳細設定]タブと同じにしておくといいでしょう。

デフォルトルートを変更する

原則としてVPNに接続した場合は、デフォルトルート(default route)はそちらに切り替わります。怪しげな無線LANアクセスポイントなどを使用する場合はそのほうが都合がいいですが、あくまでVPNは補助的に使いたい場合もありますので、その設定方法を紹介します。

何も設定しない場合、`ip route`コマンドを実行すると**図8**のような結果になります。なお、PPTPサーバのIPアドレスは削除しています。

2つあるdefaultのうち、metricの値が小さいものがデフォルトルートになるため、PPTPサーバが選択されていることがわかります。

ルートの設定は[IPv4設定]タブの[ルート]をクリックします。[192.168.3.0]は接続先のローカルIPアドレスです。これを適宜変更し、あとは**図9**を参考にして変更します。この設定を有効にして`ip route`コマンドを実行すると**図10**のようにdefaultが1つだけになります。SD

図6　L2TP/IPsecのVPNタブの設定例

図7　L2TP/IPsecではプレシェアードキーを設定する

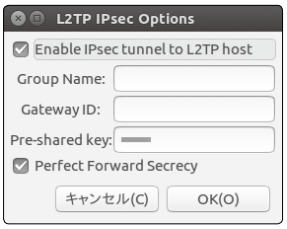


図9　デフォルトルートのサンプル設定

図8　VPNに何も設定しない場合

```
$ ip route
default dev ppp0  proto static  scope link  metric 50
default via 192.168.12.1 dev enp0s3  proto static  metric 100
169.254.0.0/16 dev enp0s3  scope link  metric 1000
192.168.3.2 dev ppp0  proto kernel  scope link  src 192.168.3.234  metric 50
192.168.12.0/24 dev enp0s3  proto kernel  scope link  src 192.168.12.52  metric 100
(PPTP Server) via 192.168.12.1 dev enp0s3  src 192.168.12.52
```

図10　defaultが1つだけになっているのがわかる

```
$ ip route
default via 192.168.12.1 dev enp0s3  proto static  metric 100
169.254.0.0/16 dev enp0s3  scope link  metric 1000
192.168.3.0/24 dev ppp1  proto static  scope link
192.168.3.2 dev ppp1  proto kernel  scope link  src 192.168.3.234  metric 50
192.168.12.0/24 dev enp0s3  proto kernel  scope link  src 192.168.12.52  metric 100
(PPTP Server) via 192.168.12.1 dev enp0s3  src 192.168.12.52
```

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

ユーザとグループ、ファイルの属性と権限がどのように関連しているか解説します。

第7回 ファイルの属性と権限、プロセスの所有

プログラミングが上達してくると、「他人の書いたコードが気に入らない」、「汚く見える」というシンドロームに陥ります。とくに、人のコードをデバッグ・保守しなければならないときは、堪えます。人のソースコードを読むということは、その人のそのときの思考を1つ1つ理解しながらたどっていくことです。理解し難い、受け入れ難いは当然のことかもしれません。著者も、数ヵ月して見た自身のコードが、まったく残念だと思うことは頻繁にあります。一方まれに10年以上前に書いた自分のコードに感心することもあります。いずれにしても3ヵ月もしたら、自分のコードも他人のものと変わりありません。

あれもこれも否定して、コードを書き換えてしまったり、コードを書いた人の人格まで否定するような発言をすることは賢明ではありません。戦線拡大は不利益も拡大します。優れたソフトウェアエンジニアやシステムアドミニストレータは、不愉快は「ごくん」と飲み込んで、システムがどのように機能するかに注目して、どうしたら活用できるのか、最低限何を改善(ハック)すれば目的を果たせるのかを考えます。このような建設的な考え方は近年DevOpsでの組織と文化に対する提唱(お互いの尊重、お互いに対する信頼、失敗に対する健全な態度、相手を非難しない)やAgile Modelingの価値(コミュニケーション、勇気、フィードバック、謙虚さ、簡潔さ)と合致します。

所有者と権限

Unixは、マルチユーザ・マルチプロセスのオペレーティングシステムです。自分が編集しているファイルを、同時にほかの人が見たり編集したりすることもできます[注1]。逆に権限を設定すれば、自分だけしか読めない、実行できないファイルとすることもできます。また、プロセスにも実行しているユーザとグループの情報があります。実行中のプロセスが、ファイルやディレクトリにアクセスする権限は、プロセスのユーザとグループに従います。今回はこれら、ファイルの所有者と権限、プロセスとの関係について少し掘り下げてみることにしましょう。

ファイルの基本属性とプロセスの実行者

権限をはじめセキュリティの管理は、おもにシステム管理者の仕事です。今回説明する内容は、一般ユーザとしてだけでなく、システム管理者の視点も必要です。もちろん、自分のホームディレクトリやファイルは、あなた自身の手でしっかりと戸締まりしましょう。

所有者とグループに対してファイル、ディレクトリのアクセス権限(**図1**)の設定が基本です。実行できるプログラムに対してのセキュリティもファイルへのアクセス権限が軸です。安全に

注1) 編集中の情報がリアルタイムで共有されるわけではありませんが。

サービスが動くようにすること、セキュリティ診断でも、ここから確認していきます。プログラムを実行できないときは、権限があるかを確認してみましょう。

アクセス権限は、ファイルの①所有者、②属するグループ、③その他の利用者、の3つの利用者クラスに対して設定できます。そして、基本の権限は“読み込み可能：r”、“書き込み可能：w”、“実行／検索可能：x”の3つです。これを、rwx bitsと呼ぶのでしたね[注2]。

プロセスは基本、実行した人・グループのもので、そのプロセスがファイルをオープンしたり実行したりできるかは、対象のファイルに設定された権限に従います。

STEP UP! setuid、setgid、sticky属性

rwxに加えて、ファイルやディレクトリには、setuid(set-user-id：セットuid)、setgid(set-group-id：セットgid)[注3]、sticky(スティッキー)の3つの属性を指定することができます(**図2**)。

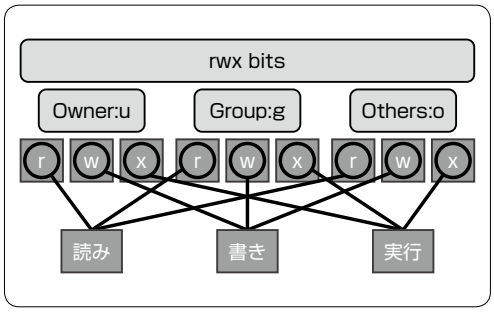
実行可能ファイルのsetuid/setgid属性がオンなら、実行者のuid/gidではなくてそのファイルのuid/gidで実行します(**図3**)。

サービスのユーザアカウント(たとえばユーザ＝www、グループ＝www)を作ります。プログラムには所有者wwwとしてsetuidし、グループwwwに実行権限を与えておけば、誰が起動しても、サービスは常にwwwとしての権限で実行されるようにできます。

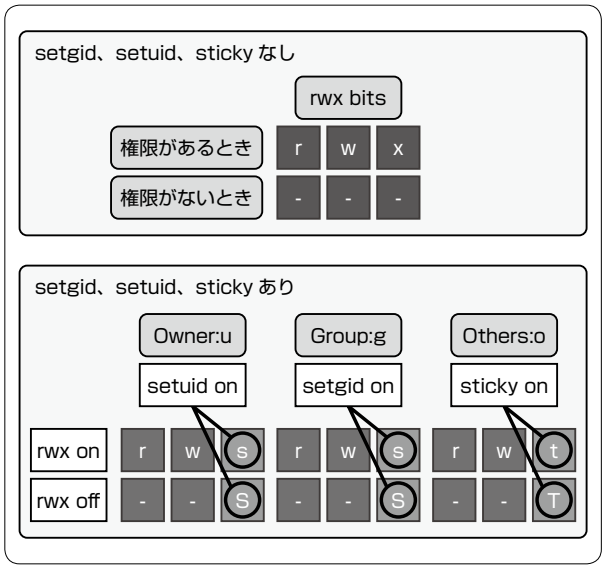
> スクリプトファイルに対して、setuid/setgidを指定しても実行者はファイル所有者にはなりません。ファイルを実行するプログラムは、たとえばシェルスクリプトであれば/bin/shや/bin/bashでしょうし、Rubyスクリプトではrubyが実行されるファイルです。スクリプトは、これらシェルやRubyの動作手順を示したテキスト情報にすぎないからです。

注2) 連載第2回(本誌2016年6月号)参照。
注3) set-uid、set-gidと記述することがあります。

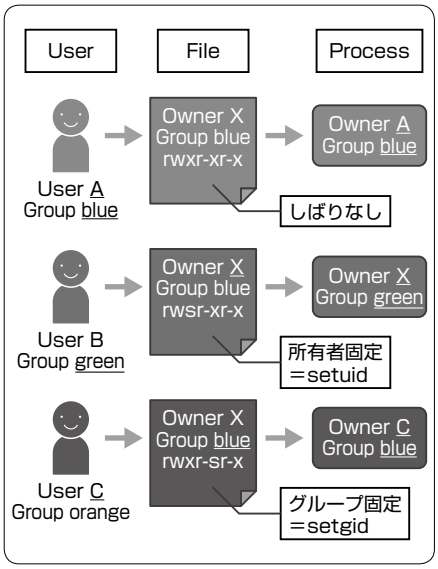
▼図1 rwx bits——3つの属性



▼図2 lsの属性表示



▼図3 setuid/setgidのあるファイルとプロセスの所有



ディレクトリのsetgidがオンなら、そのディレクトリの中のファイルとサブディレクトリを作成する場合のグループが、作成するユーザとは無関係に、そのディレクトリのグループとなります。作業ディレクトリをグループで共有して使うときに役立ちます。

ベトベト、ねばねば、貼り付くという意味のsticky。ディレクトリにsticky bitが立っている場合[注4]は、ディレクトリ中のファイルやディレクトリの削除制限をします。そのディレクトリに入っているファイルは、ディレクトリに対して書き込み権限を持っているユーザでそのファイルの所有者、ディレクトリの所有者、もしくはスーパーユーザがファイルを削除したり名前の変更ができます。つまりstickyを使えば、ディレクトリへのファイルやディレクトリ追加は許可しても、削除や名前の変更ができないようにできます。

スーパーユーザ（root）は特別権限

Unixでは、特別な権限を持ったスーパーユーザが存在します。rootというユーザで、UIDとGIDが両方0です。rootには原則アクセスできないファイルやディレクトリ、制御できないプロセスはありません（**図4**）。rootにもパスワードを設定しておけば、普通にログインすることもできます。セキュリティ上の理由から誰がスーパーユーザになったか、**記録が残るように**しなければなりませんが、rootで直接ログインしてしまうと、ログインした個人を識別できません。そのため、スーパーユーザ権限でアクセスする必要がある場合は、別の一般ユーザとしてログインして、suコマンドでユーザを変更したり[注5]、権限を制御・許可された特定のユーザがsudoコマンドを使ってスーパーユーザ権限を発動します。

su—Substitute User identity—別のユーザidに切り替える

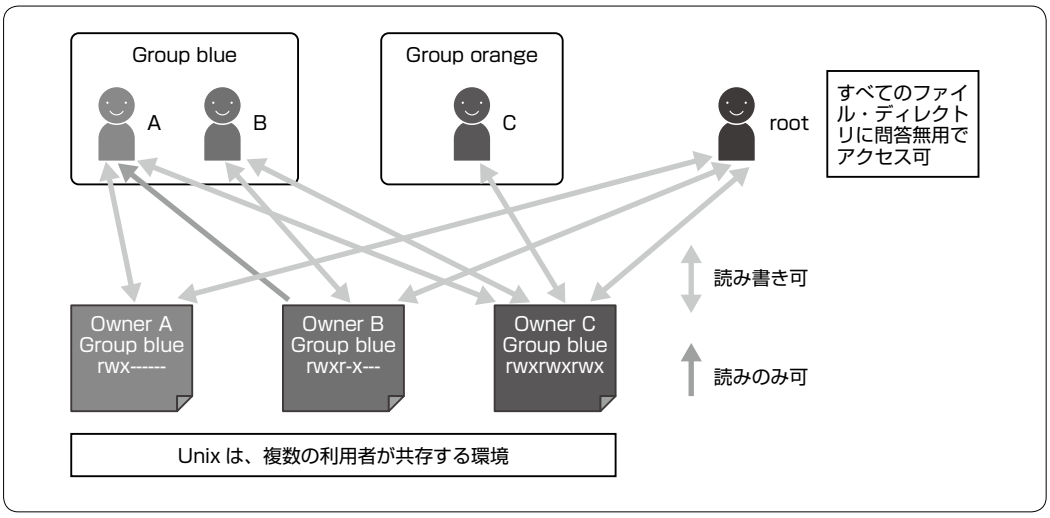
書式
```
su [-] [ユーザ]
```

「ユーザ」を省略するとrootになります。-オプションを指定すると、新規にログインした状態と同じ、指定しなければ現在のシェル情報が

注4）古くは、実行ファイルのstickyがオンになっていると、プロセスができるだけ物理メモリ内にとどまるようにする仕様でした。

注5）この方法もセキュリティの理由で推奨されなくなりました。

▼図4　所有者、グループ、その他によるアクセス制御とスーパユーザ権限



引き継がれます。

sudo—SUしてDO—別のユーザ権限でプログラムを実行する

書式
```
sudo [-s] [-u ユーザ] [コマンド]
```

sudoは、ユーザ対象コマンドの権限を個別に管理でき、sudoするとシステムのログに記録されるなど、rootでの直接ログインやsuに比べて安全面で優れています。-sオプションを使うと、許可されたユーザ[注6]は、rootになることができます。

sudoコマンドを実行したときに要求されるパスワードは、現在の利用者のパスワードを入力します。

ログイン時などで入力を修正するときのTIPS

login、sudoなどでパスワードを求められるとき、入力の途中で間違いに気がついて、バックスペースキーで修正しようとしてもうまくいかないことがありませんか。この入力状態は、バックスペースが文字(Ctrl + H)と入力され、前の文字の取り消しは機能しません。このときわざとログインを失敗して再挑戦するのは、次のプロンプトが出るまでに時間がかかったり、場合によっては複数の失敗でアカウントがロックされてしまうこともあり、賢明な対応ではありません。

パスワードなどの入力状態で使える編集操作キーは、stty -aコマンドを実行したときに出力される情報、"erase = ^?"や"kill = ^U"などなのです。Appleのキーボードでdeleteと刻印されたキーボードはDEL(アスキーコードで7F)が入力され、Windows系のキーボードのほぼ同じ位置にあるバックスペース(BSとかBACKとか←と刻印)はCtrl + H(アスキーコードで08)になります。コマンドライン上は、DELもバックスペースも直前の文字が取り消されるので区別がしにくいかもしれません。で、"erace = ^?"の^?とは、DELのことです。Appleキーボードでは、パスワード入力で間違えた文字をDeleteキーで修正できるのですが、Windowsキーボードではバックスペースではなくて、Delキーが使えます。

またパスワード入力は、入力文字が画面にフィードバックされないので、何文字入れて、どこで間違えたかわからないというのもありますね。このときは、"kill = ^U"というところのCtrl + Uを使えば、入力をすべてキャンセルした状態となり、1文字目から入力をやり直せます。ちなみにAppleキーボードでバックスペースを入力するには、Ctrl + Hを入力します。

注6) sudoは/etc/sudoersで別のユーザになることができるユーザを限定できます。

ユーザ、グループとアクセス権限

id—ID—ユーザID、グループIDなどを確認する

現在ログインしているユーザやグループを確認する必要があるときには、idコマンドで確認できます。ログインしているユーザには、現在のユーザ名と対応する(ユニークな番号)UIDと、現在のグループ名と対応する(ユニークな番号)GIDがありますが、idはこのいずれの情報も確認できます。ユーザ名だけを確認するのであればwhoami、所属グループを確認するにはgroupsなどのコマンドがidと同じように使えます。

id Ubuntuでの例
```
$ id ↵
uid=1000(masa) gid=1000(masa) groups=1000(masa),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ id -un ↵
masa
$ id -gn ↵
masa
```

STEP UP! ユーザやグループの情報を確認・編集する

一般的なUnix系システムでは、ユーザ情報、グループ情報が、/etc/passwd[注7]、/etc/shadow、/etc/group/etc/gshadow[注8]に保存されています。

これらのファイルを直接編集することで、ユーザやグループを追加・変更・削除することができます。

また、useradd、groupadd、userdel、usermod、groupdelなどのコマンドを使って、ホームディレクトリの管理などを併せて操作できます。

OS X(10.6以降)では、/etc/passwd、/etc/groupの情報は使わず、独自のディレクトリサービスを使うようになりました。このディレクトリサービスをコマンドラインで使うdscl[注9]コマンドがあります。

注7) 暗号化されたパスワードファイル。
注8) 暗号化されたグループパスワードファイル。
注9) Directory Service Command Line utility

newgrp—NEW GRouP—新しいグループにログインする

ユーザは複数のグループに所属し、作業の役割ごとにグループを変更できます[注10]。**newgrp**コマンドは、同じユーザIDと新しいグループで、新しいシェルを実行します(**図5**)。ユーザは指定するグループに所属している必要があります。所属していないグループを指定すると、newgrpはそのグループに対するパスワードを要求します。パスワードがマッチしなければ、そのユーザのデフォルトグループでシェルを起動します。

chown—CHange OWNer、chgrp—CHange GRouP—ファイルやディレクトリの所有者、グループを変更する

既存のファイルの所有者、グループ情報を変更するには、**chown**、**chgrp**コマンドを使います。chownは、通常スーパーユーザ権限で実行します。chgrpは、自分が所属しているグループであればスーパーユーザでなくても変更できます。

オーナー情報は、自分が所有しているファイルでも、root権限でなければ変更できません。

```
rootではない一般ユーザによるchown実行の失敗例
$ chown root fig05.pptx ↵
chown: fig05.pptx: Operation not permitted
```

chownで所有者とグループを一度にまとめて変更することもできます。所有者名とグループ名の区切りには“**:**”を使います(**図6**)。Linuxでは名前の区切りに紛れがなければ“**.**”を使うこともできます。**:**の前にユーザ名を指定しなければ、グループのみを変更します。

chmod—CHange MODe—属性／権限を変更する

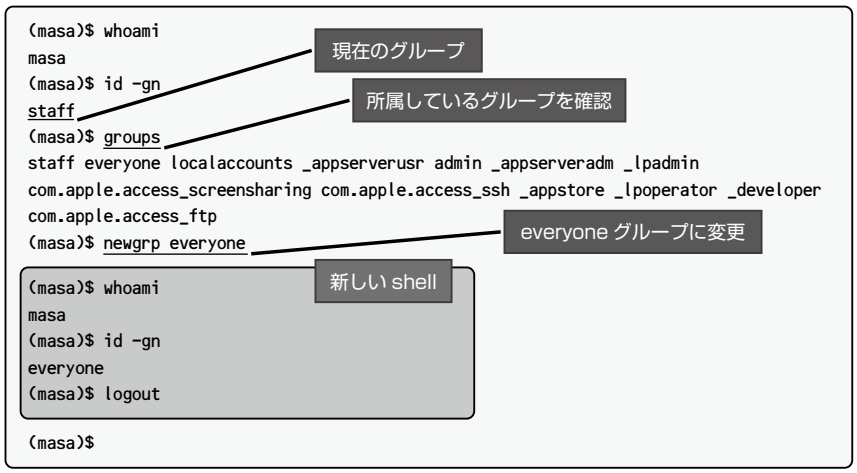
chmodコマンドは、ファイルの属性を2種類の記法で変更します。属性の指定方法はrwxを用いたシンボル表記と、8進数表記です(**図7**)。

8進数表記(**図7**)は、setuid/setgid/stickyで1桁、所有者(Owner)で1桁、グループ(Group)で1桁、その他のユーザ(Other)で1桁の8進数(合計4桁)を、各属性をオンにしたいところの数字を足した値で表現します(**表1**)。setuid/setgid/stickyを指定しない場合は、3桁で表記することができます。

シンボル表記は、次のように“対象 演算子 権限”という式で表現します。

注10) Linux系の環境では、groupに所属していても現在のグループがファイルの実行権限と異なったグループの場合、実行ができません。OS XなどBSD系の環境では、ファイルと同じgroupに所属していれば実行ができます。

▼図5　newgrp



▼図6　ファイル123の所有者、グループを変更してみる

```
bash-3.2# ls -l 123 ↵
-rw-r--r--  1 masa  staff  47  9 16  2015 123
bash-3.2# chown root 123 ↵
bash-3.2# ls -l 123 ↵
-rw-r--r--  1 root  staff  47  9 16  2015 123
bash-3.2# chgrp daemon 123 ↵
bash-3.2# ls -l ↵
total 40
-rw-r--r--  1 root  daemon 47  9 16  2015 123
bash-3.2# chown masa:staff 123 ↵
bash-3.2# ls -l 123 ↵
-rw-r--r--  1 masa  staff  47  9 16  2015 123
```

対象	演算子	権限
[u|g|o]	[+|-|=]	[s|t|r|w|x]

u……所有者　　s ……setuid/setgid
g……グループ　　t ……sticky
o……その他　　r ……読み込み
+……オンにする　　w……書き込み
- ……オフにする　　x ……実行
=……指定した権限にする

```
chmodの使用例
fileのモードをrwxrwxrwxにする
$ chmod 777 file
$ chmod ugo=rwx file
$ chmod ugo+rwx file

fileのモードをrwsr-xr-xにする
$ chmod 4755 file

グループとその他のユーザからrを取り去る
$ chmod go-r
```

サブディレクトリに再帰的に適用する

chmod、chown、chgrpで、ディレクトリの下全部の属性を変更したい場合は、recursive(再帰)指定するオプション-Rで一括処理ができます。

今回のまとめと次回について

ファイルとディレクトリに設定できる属性について確認しました。

互換性のため、同じような機能を持つコマンドもありますが、セキュリティ面、性能面などを考慮して、適切に選択してください。

次回は、シェルスクリプトへの入門です。SD

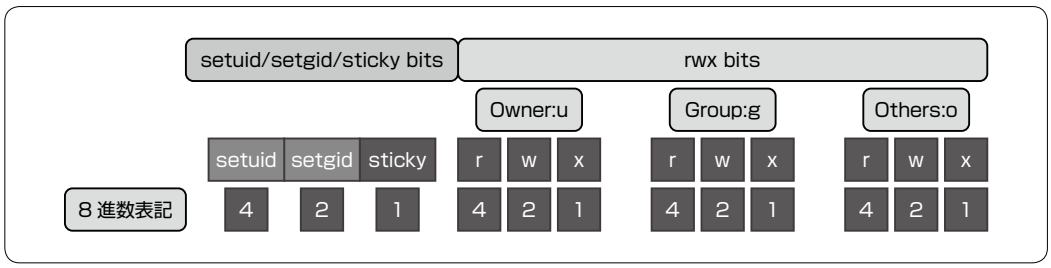
今回の確認コマンド

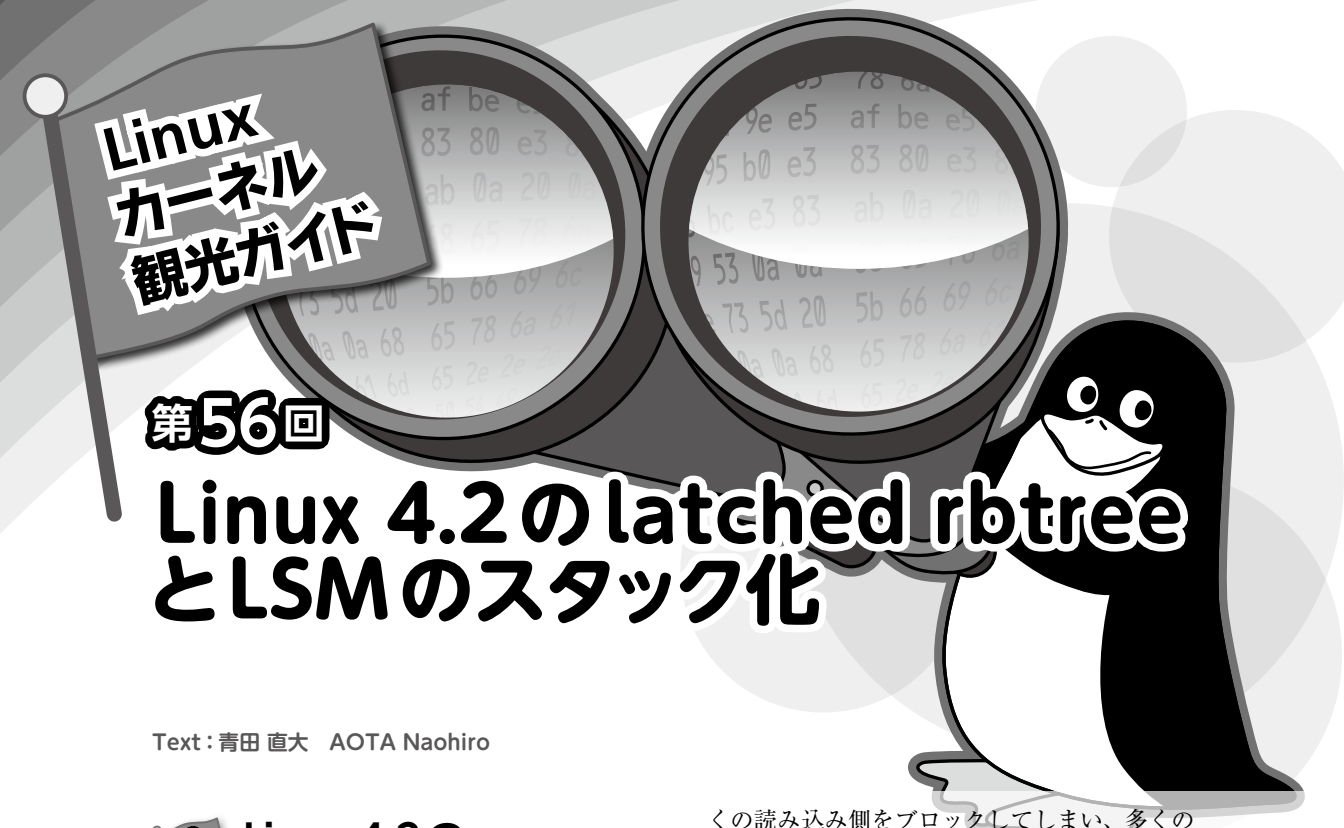
【manで調べるもの
(括弧内はセクション番号)】
su(1), sudo(8), sudoers(5), newgrp(1), id(1), whoami(1), chown(1), chgrp(1), chmod(1), stty(1)
- Linuxのみ -
useradd(8), userdel(8), usermod(8), groupadd(8), groupdel(8), groupmod(8)
- OS Xのみ -
stiky(8), dscl(1)

▼表1　rwx bits——8進数表記例

lsの表示	8進表記
rwx rwx rwx	777
rw- rw- r--	664
r-s r-x ---	4550
rwx rws ---	2770
rwx --S ---	2700
rwx r-x r-t	1755
rws --- --T	5700

▼図7　rwx bits——3つの属性と8進数





第56回

Linux 4.2のlatched rbtreeとLSMのスタック化

Text：青田 直大　AOTA Naohiro

Linux 4.2のその他の変更

Linux 4.8-rc7が9月19日にリリースされています。ここまで来るとだいたい次は正式に4.8のリリース……となるのですが、今回はまだRCのpatchが落ち着いていないようです。ということで、4.8のリリースは10月始めとなるでしょうか……。いずれにせよ、この記事が出るころには4.8がリリースされているでしょう。

これまで数回に渡ってLinux 4.2の新機能を紹介してきましたが、今回はlatched rbtreeとLSMのスタック化の紹介をして、4.2の紹介を締めくくろうと思います。

RCUによる排他制御

Linuxカーネル内のデータ構造は、同時に複数のプロセスからアクセスされます。当然ながら排他制御をしなければ、たとえばリストの内にあったはずのデータがなくなるなど、そのデータ構造は壊れてしまいます。排他制御というと一般にロックを用いますが、多くの読み込み側とごく少ない書き込み側がいるケースでは、それが少しでも、書き込み側が動作している間多くの読み込み側をブロックしてしまい、多くのCPUコアにスケールしなくなるという問題があります。

そこでLinuxカーネルでは、リード・コピー・アップデート(RCU)という排他制御がよく使われています。RCUを使うことで書き込み側が読み込み側をブロックすることなくデータを更新できるようになります。それでは具体的に、RCUで保護されたリストからアイテムを削除する様子を見てみましょう。

リスト1のコードはext4の内のコードで、`ei->i_prealloc_list`というRCUで保護されたリストをたどっていく部分(リストの読み込み側)です。`list_for_each_entry_rcu()`がRCUで保護されたリストの各エントリをたどるためのマクロとなります。読み込み側はリストのアイテムを参照する区間を、`rcu_read_lock()`と`rcu_read_unlock()`で囲まなければいけません。ここで"read lock"とは言っていますが、実際にはほとんど何もしないコードになっていて、読み込み側はほかの読み込みスレッドや書き込み側をまったく気にすることなく処理を行うことができます。この`rcu_read_lock`、`rcu_read_unlock`で囲まれた区間をRCU読み込み側クリティカルセクションと呼びます。

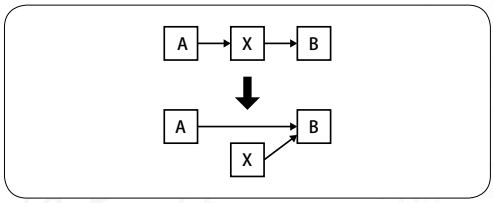
リスト2のコードは、先ほどのi_prealloc_listからアイテムを削除している部分です。spin_lock()は、ほかのRCUリストの更新スレッドとの排他制御に使われています。list_del_rcu()がアイテムはリストから削除している関数です。削除するアイテムをXとすると、list_del_rcu()は図1のようにAからXへのリンクをBへと張り替えます。このとき、XからBへのリンクはまだ残していることに注意してください。リンクの張り替えを行った時点では、まだリストを読んでいるスレッドがいるかもしれません。ここでXからBへのリンクを消してしまうと、ちょうどXを読んでいたスレッドはリストをたどれなくなってしまいます。切り替え後のAやその前を読んでいたスレッドは、Xの消えた新しいリストを見ますし、X以降を読んでいたスレッドは、Xのあった古いリストを見ていたことになります。いずれにしても読み込み側は問題なくリストをたどることができます。

さてXをそのままにしていては、メモリリークとなってしまうので、どこか安全な時点でXを削除する必要があります。これを行うのがcall_rcu()です。この関数は、呼び出された時点でのRCU読み込み側クリティカルセクションが終わったあと、すなわちリストをreadしていたスレッドがすべてリストを読み終わったあとに、指定した関数が呼び出されるようにスケジュールします。このコードで呼ばれているのはext4_mb_pa_callbackです。この関数が呼ばれている時点では読み込み側は誰もXを参照していないことが保証できているので、安全にアイテムをkmem_cache_free()を使って削除できます(リスト2)。

RCUと赤黒木

RCUによる更新は、リスト2の場合で見たようにatomicな更新に依存しています。atomicな更新ができない場合にはRCUがうまく動作しな

▼図1　RCUで保護されたリストからのアイテム削除



▼リスト1　RCUリスト読み込み側

```
    rcu_read_lock();
    list_for_each_entry_rcu(pa, &ei->i_prealloc_list, pa_inode_list) {
        (..中略..)
    }
    rcu_read_unlock();
```

▼リスト2　RCUリスト書き込み側

```
    spin_lock(pa->pa_obj_lock);
    list_del_rcu(&pa->pa_inode_list);
    spin_unlock(pa->pa_obj_lock);

    call_rcu(&(pa)->u.pa_rcu, ext4_mb_pa_callback);

static void ext4_mb_pa_callback(struct rcu_head *head)
{
    struct ext4_prealloc_space *pa;
    pa = container_of(head, struct ext4_prealloc_space, u.pa_rcu);

    BUG_ON(atomic_read(&pa->pa_count));
    BUG_ON(pa->pa_deleted == 0);
    kmem_cache_free(ext4_pspace_cachep, pa);
}
```

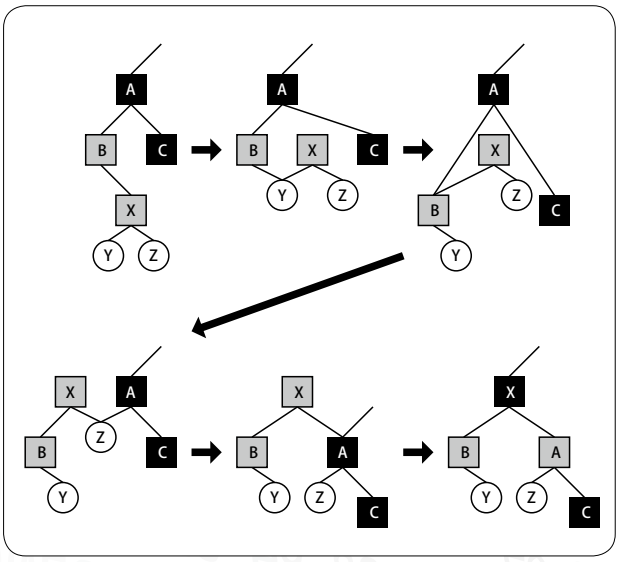
くなります。たとえば赤黒木(rbtree)への挿入の場合を考えてみましょう。rbtreeへのノードの挿入は、❶挿入位置の探索、❷挿入位置へのノードのリンク、❸ツリーのバランスの3つのステップで行われます。❶、❷までは更新側同士でlockをとりさえすれば、RCUを使ってatomicに行うことができます。しかし❸が曲者で、ツリーのバランスの過程で読み込み側からはアクセスできないノードができてしまう瞬間があります。

具体的に**図2**のようなツリーのバランス作業を見てみましょう。**図2**の最初は"X"というノードを新しく挿入したところです。説明の便宜上XにはYとZという下部のツリーをつけていますが、本来は挿入直後ですのでどちらもNULLになっています。また、Aの上の線はツリーの上部からのAへの参照を意味しています。この時点ではBとXがどちらも赤であり、赤黒木のルールに反しています。したがって、ここから**リスト3**のようなコードで木を変形していきます。

変数として「**node=新しく挿入されたノードX**」「**parent=Xの親ノードであるB**」「**gparent=parentの親ノードであるA**」と設定しています。また、**WRITE_ONCE(X, y)**は、"X=y"をコンパイラに最適化・順序入れ替えなどをさせないためのマクロです。最初の**WRITE_ONCE**ではBの右の子が、Xの左の子であるYになります。B、YはXより小さいので、次の**WRITE_ONCE**でXの左の子をBにできます。同様に3つ目のWRITE_ONCEで、ZをAの左の子にするとA、C、ZはXよりも大きいので、AをXの右の子とできます。最後に**__rb_rotate_set_parents()**で、Xの色をAの色(=黒)にして、Aを赤にし、さらにAの親ノードからの参照をXに張り替えます。これでバランスが完成です。

図2を見るとわかるように、バランス中にツリーを読みにいくと、ツリーのルートからの通常の探索ではたどることができないノードが存在してしまいます。これではツリーの探索の結果が信用できません。かといって、読み込み側と書き込み側をロックで排他制御すれば、最初に言ったようにスケールしなくなります。

▼図2　rbtreeのバランス(■部分は黒、■部分は赤を示します)



▼リスト3　バランスを行うコード

```
node = X;
parent = B;
gparent = A;

tmp = node->rb_left;
WRITE_ONCE(parent->rb_right, tmp);
WRITE_ONCE(node->rb_left, parent);

tmp = node->rb_right;
WRITE_ONCE(gparent->rb_left, tmp);
WRITE_ONCE(node->rb_right, gparent);
__rb_rotate_set_parents(gparent, node, root, RB_RED);
```

seqlockによる排他制御

こうした問題への解決策として、seqlockの使用があります。たとえばAFSで使われるRxRPCの実装には、**リスト4**のようにseqlockが使われています。このロックは「シーケンシャル番号」を持っていて、書き込み側がロックを

とったとき、ロックを解放するときにそれぞれ番号を1増やします。読み込み側は`read_seqbegin_or_lock()`でシーケンス番号を確認します。もし番号が奇数であれば、書き込み側がいるということですので書き込み側がいなくなるまでスピンして待ちます。書き込み側がいなくなれば、そのときのシーケンス番号を"seq"に保存します。探索後、`need_seqretry()`でシーケンス番号が変わっていないかどうかを確認します。ここでもしシーケンス番号が変わっていれば、探索中に書き込みがあり、ツリーが編集されたということですので探索をやりなおします。

このようにseqlockを使って、ツリーの編集があればやりなおすようにすれば、ツリーのバランスというatomicではない編集があっても正しい探索結果を得ることができます。しかしseqlockでは、読み込み開始時に書き込み側がいる場合に、書き込み側がいなくなるまで待ってしまいます。そうすると、書き込み側のコードの中で、読み込みができません(読み込むとデッドロックになってしまう)。そこでより広いシーンで使うことができるデータ構造としてlatched rbtreeという新しいツリーがLinux 4.2から導入されました。

latched rbtreeを使った排他制御

Latched rbtreeは、rbtreeをatomicに更新できないという問題点を2つのツリーを保持することで解決しています。latched rbtreeへの挿入を行う`latch_tree_insert()`と探索を行う`latch_tree_find()`をリスト5に示します。`latch_tree_insert()`からみていきましょう。`raw_write_seqcount_latch (&root ->seq);`はシーケンス番号をインクリメントし、探索側がみるツリーを"ツリー1"の方に切り替えます。

▼リスト5　latched rbtreeの挿入関数と探索関数

```
static __always_inline void
latch_tree_insert(struct latch_tree_node *node,
          struct latch_tree_root *root,
          const struct latch_tree_ops *ops)
{
    raw_write_seqcount_latch(&root->seq);
    __lt_insert(node, root, 0, ops->less);
    raw_write_seqcount_latch(&root->seq);
    __lt_insert(node, root, 1, ops->less);
}

static __always_inline struct latch_tree_node *
latch_tree_find(void *key, struct latch_tree_root *root,
        const struct latch_tree_ops *ops)
{
    struct latch_tree_node *node;
    unsigned int seq;

    do {
        seq = raw_read_seqcount_latch(&root->seq);
        node = __lt_find(key, root, seq & 1, ops->comp);
    } while (read_seqcount_retry(&root->seq, seq));

    return node;
}
```

▼リスト4　seqlockの使用

```
// 読み込み側
    do {
        read_seqbegin_or_lock(&peer->service_conn_lock, &seq);

        p = rcu_dereference_raw(peer->service_conns.rb_node);
        while (p) {
            // 探索。見つかれば goto done;
            (..中略..)
        }
    } while (need_seqretry(&peer->service_conn_lock, seq));
done:
    done_seqretry(&peer->service_conn_lock, seq);
```

その次の`__lt_insert(node, root, 0, ops->less);`は、“ツリー0”にアイテムの挿入とバランスを行います。その後、同様に探索対象を“ツリー0”に切り替えて、“ツリー1”にアイテムの挿入とバランスを行います。

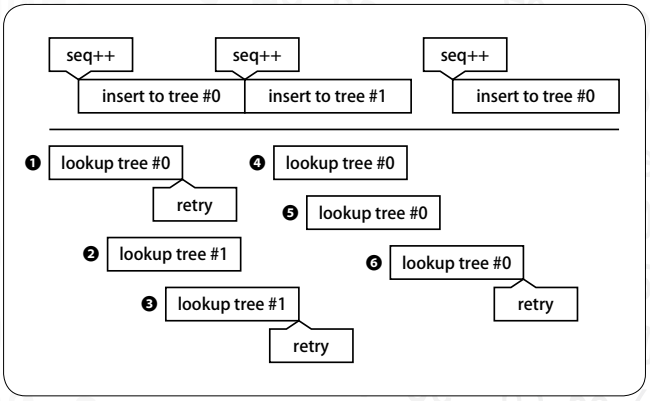
読み込み側のコードは“seqlock”の読み込み側と似ていますね。まず、`raw_read_seqcount_latch();`でシーケンス番号を読み込みます。シーケンス番号の下位1bitが探索すべきツリーを示すので、そのツリーを指定して`__lt_find()`で探索を行います。探索後にシーケンス番号が変わっていた場合、探索対象としていたツリーが探索中に更新されたということですのでもう一度探索をやりなおします。このチェックはseqlockと同様に`read_seqcount_retry()`で行われています。

挿入と探索のタイミングを詳しく考察してみましょう(**図3**)。

❶`latch_tree_insert()`よりも前に始まり、ツリー0への挿入・バランス中に終わる探索は、ツリー0を対象として探索し、シーケンス番号の変更を検知して探索をやりなおします。

❷のようにツリー0への挿入中に探索が開始・完了するケースでは、ツリー1がアイテムの挿入前の状態で安定しているので探索をやりなおす必要はありません。実際シーケンス番号が変わっていないので、リトライすることなく`latch_tree_find()`が完了します。

▼図3　latched rbtreeへの挿入と探索



❸は❶と同様に、探索対象であるツリー1が同時に変更されていたことをシーケンス番号の変化から検知し、リトライします。

❹、❺ではツリー0が挿入後の状態で探索中に変化していないので、リトライなしで探索を完了できます。

一方で❻のように、次の`latch_tree_insert()`と実行区間がかぶってしまうと探索中にツリー0が変更されているので、リトライを行うことになります。このように書き込み中でも必ず1つは変更されていないツリーがあるので、書き込みのコードの中でも読み込み側になることができます。

latched rbtreeの使用例

新しいデータ構造であることと、読み込みを書き込みの中にネストしたいという場面が少ないこともあってか、latched rbtreeは今のところ一個所、`__module_address()`の実装にしか使われていません。`__module_address(unsigned long addr)`は`addr`で指定したアドレスを含むkernel moduleを探索する関数です。

latched rbtreeを使う前はmoduleのリストを線形にたどっていました。これでは読み込まれるmoduleの数が増えると、探索の時間がばかにならなくなります。この関数はperfやトレースによるstackダンプでアドレスとmodule名を表示するときに頻繁に使われます。perfやトレースはどんなコンテキストからでも呼ばれ得るので、まさに読み書きがネストできるlatched rbtreeが必要な場面になっています。

セキュリティモジュールのスタック化

SELinuxやTOMOYOといったLinuxのセキュリティ機構は、Linux Security Modules(LSM)という形で実装されています。LSMは、Linux

カーネルの各所へのhookという形で実装されています。4.1以前では動作するLSMは基本的に1つで複数のセキュリティチェックを同時に動かすことはできませんでした。SELinuxのように「大きい」、カーネルのほとんどの部分をhookするLSMだけがあるうちはこれでも問題はなかったのですが、YAMAのようにごく一部の機能だけをhookするLSMが出てくると問題は複雑になります。YAMAの保護もSELinuxも使いたいという場合には、SELinuxにYAMAの機能をとりいれなければいけませんし、そうするとYAMAを単独で使いたいユーザは困ってしまいます。この問題を解決するためにLSMのスタックが可能となりました。

Linux 4.1とLinux 4.2以後でのptraceのセキュリティチェックの関数を見てみましょう(**リスト6**)。4.1版では**securtiy_ops->ptrace_access_check()**がLSMの関数を呼んでセキュリティチェックを行っています。また、YAMAをスタックする場合、YAMAの関数が「無理矢理」呼び出されているのがわかります。新たにスタックするLSMを追加する場合、その関数を該当個所に追加しなければならず、きれいな実装とは言いがたいですね。

一方Linux 4.2以降では**call_int_hook**というマクロが使われています。これは展開されると、**security_hook_heads.ptrace_access_check**というリストからhook関数をとってきて、順番に呼んでいく(途中で0以外が返れば中断)というコードになります。これでLSMをきれいにスタックできるようになりました。

このようにLSMのスタックが可能になったためか、最近は「小さい」LSMがよく提案されるようになってきました。たとえば、loadpinというカーネルモジュールや、ファームウェアのロードを特定のファイルシステムに限定するLSMや、chrootをよりセキュアにするHardchrootといったLSMが提案されています。SD

▼リスト6　4.1のLSMと4.2からのLSMの比較

```
// Linux 4.1
int security_ptrace_access_check(struct task_struct *child, unsigned int mode)
{
#ifdef CONFIG_SECURITY_YAMA_STACKED
    int rc;
    rc = yama_ptrace_access_check(child, mode);
    if (rc)
        return rc;
#endif
    return security_ops->ptrace_access_check(child, mode);
}

// Linux 4.2から

#define call_int_hook(FUNC, IRC, ...) ({                        \
    int RC = IRC;                                               \
    do {                                                        \
        struct security_hook_list *P;                           \
                                                                \
        list_for_each_entry(P, &security_hook_heads.FUNC, list) { \
            RC = P->hook.FUNC(__VA_ARGS__);                     \
            if (RC != 0)                                        \
                break;                                          \
        }                                                       \
    } while (0);                                                \
    RC;                                                         \
})
int security_ptrace_access_check(struct task_struct *child, unsigned int mode)
{
    return call_int_hook(ptrace_access_check, 0, child, mode);
}
```

November 2016

NO.61

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
榎 真治　ENOKI Shinji　enoki-s@imail.plala.or.jp

ITコミュニティ運営 関西ならではの課題・取り組みとは

今回は、7月に京都で行ったjus研究会の模様をお伝えします。

jus研究会 京都大会

■ITコミュニティの運営を考える
【講師】重兼 史尚（NSC運営事務局）
藏野（くらの） 文子（GDG京都）
榎 真治（日本UNIXユーザ会／LibreOffice日本語チーム）
法林 浩之（日本UNIXユーザ会）
【日時】2016年7月30日（土）13:00～13:45
【会場】京都リサーチパーク（KRP）1号館

毎年夏に、jus研究会京都大会を開催しています。今回は、ネットワーク技術を勉強したい人向けコミュニティ「NSC（Network Skills Community）」の重兼さん、Googleのテクノロジに興味のある有志の集まり「GDG京都（Google Developer Group 京都）」の藏野さんをお迎えして、jusが各地で開催している「ITコミュニティの運営を考える」のパネルディスカッションを行いました。参加者は20名でした。

■コミュニティの方向性を変えるとき

はじめに筆者から「コミュニティの方向を変えたときに、どのような方法で変えていったのか？」というお題を出しました。NSCは、元はNetwork Skills Competitionという名前で、学生向け競技会を開催していました。運営メンバー同士の議論で「参加者は競技会が終わると勉強しなくなる」「イベントを継続するのがたいへん」などの問題提起があり、コミュニティを再定義して方向転換されたそうです。ネットワークについて勉強する際に、最初のハードルを越える体験を提供するという目的で、ネットワーク機器やサーバ、仮想プラットフォームも自前で用意しており、ハンズオンやディスカッションなど参加型イベントを毎月開催されているそうです。

GDG京都は、昔は京都GTUG（京都 Google Technology User Group）という名称でした。使い方ではなく開発を楽しもうと、実体は変わらないものの名前が変わったそうです。また、女性の技術者／参加者を増やしたいということで、各地に女子会も生まれているそうです。GDG京都は女性が60名くらいいるため、GDG京都女子会という形で活動されています。また、GDG京都ではコアメンバーの女性比率が半分程度とほかの地域と比べて高く、イベントを増やし過ぎると生活に支障が出るので、女子会単独のイベントは少なめにしているそうです。また、オーガナイザー（まとめ役）も1人体制から2人体制へ変更したとのことです。

jusは時代に合わせて活動内容を変えてきています。1990年前後にインターネットが流行った際には、ほかに良い受け皿となる団体がなかったためにjusが勉強会などに取り組んだり、オープンソースが出てきたときも、ほかの団体と一緒にオープンソースまつりをやることになったりと、「柔軟な姿勢で活動してきたことが良かったのでは」とのことでした。

■学生とつながるには

続いて、重兼さんからの「学生へのリーチをどう

するか？」というお題です。NSCは競技会から変更した結果、学生がほとんどこなくなったそうです。GDG京都でも学生が少ないという課題があります。卒業や結婚とともによそに行ってしまうケースもあります。「学生だけのコミュニティに参加していて、そこから先には参加してもらえていないこともあるのでは」とのことです。世界中のオーガナイザーの集まりで「大学の先生へのアプローチをすべきではないか」との指摘を受けたこともあるそうです。

LibreOfficeのグローバルコミュニティは、Google Summer of Codeを活用したり、メンターシップを重視していたりするため、若い人がいます。日本はそのあたりができていないことが課題です。

jusは学生へのリーチは壊滅的で「昔は学生もいたが、今は興味を合わせることが課題」だそうです。

NSCではコミュニティイベントでの会場ネットワーク設営を引き受けています。専門学校とタイアップして学生チームに設営を任せつつ、フォローする試みもされているそうです。今後は、「イベントだけで終わってしまっているのであとにつなげる」「ネットワーク以外のコミュニティと一緒にやっていきたい」とのことでした。

■京都や関西を盛り上げるには

次に、藏野さんから「京都で勉強会をやると人数が少ないが、どうやって盛り上げるか？」というお題が出されました。大阪で開催した場合に人数が多かったことで、人数の課題を感じられたそうです。

「機材があるイベントだとやりやすい場所が少ない」「大阪や神戸とは客層が変わる」「イベントによっては京都でも人は集まっている」「平日イベントの場合は遠いので参加がしにくいのではないか」「学生が多いので卒業したらどこかに行ってしまうのではないか」といったコメントがありました。

また、法林さんからは、「関西のイベントを盛り上げるにはどうするか？」というお題が出されました。関西でのGDGは京都と神戸があり、合同イベントの企画を進めているそうです。

NSCのような参加型は、東京では行われていないそうです。「（都会から離れており、勉強会にも参加しにくいために）ネットワークの技術に身近に触れられないような地域でやりたい」「未開催の滋賀と和歌山でもやりたい」とのことでした。

法林さんからは「関西では都市ごとにコミュニティがあることがいいことでもあり、課題にもなり得るかもしれない」というコメントがありました。

■コニュニティ運営者を増やすには

会場から「どこのコミュニティでも人が同じだったり、就職すると東京へ行ったりするのをどうするか？」というお題が出されました。筆者からは「流出しそうな学生を自分のところで採用してはどうでしょう？」という提案をしました。GDG京都では、イベントを京都から離れて行った際に、参加者の半数が初参加だったそうです。「場所を変えるのは効果的ではないか」という議論になりました。

会場からの2つめのお題は「コミュニティの運営者を増やすためにどうするか？」です。NSCではイベント参加者に声をかけて一本釣りしているが、なかなか定着しないそうです。GDG京都では、声がけした人よりも自ら手を挙げてくれた人のほうが、定着率が高いそうです。LibreOffice日本語チームでは、ほかのコミュニティで活動していた人が入ってくれたケースが多いことを話しました。jusでは、制度としてはないのですが、イベントの手伝いで、ボランティアポイントがたまってきた人に幹事として声がけしていることが紹介されました。

◆ ◆ ◆

今回は、今までと少し路線の違うコミュニティに出演いただきました。同じような悩みを抱えている一方で、そのコミュニティならではのユニークな取り組みや、関西独自の話もありました。会場からも積極的にお題が出て、コミュニティ運営について一緒に考える場になったかと思います。動画が公開されています[注1]ので、詳しい内容に興味を持たれた方はご覧ください。SD

注1）URL https://www.youtube.com/watch?v=zv9ek7ia8lM&feature=youtu.be

Hack For Japan
エンジニアだからこそできる復興への一歩

Hack For Japan スタッフ
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka
小泉 勝志郎 KOIZUMI Katsushiro
Twitter @koi_zoom1
及川 卓也　OIKAWA takuya
Twitter @takoratta

第59回 第5回 石巻ハッカソン

今回は2016年7月29～31日に開催された「石巻ハッカソン」についてレポートします。

2016年の7月29日から31日にかけて宮城県石巻市で石巻ハッカソンが開催されました。2012年から毎年この時期に開催されており、今回で5回目となります。Hack For Japanからは2012年の第1回目にIT Bootcampの講師として参加して以来、このハッカソンに参加を続けており、すでに夏の恒例イベントになっていると言えるでしょう。

このハッカソンは、東日本大震災で大きな被害を受けた石巻で、若者が地元で活躍できる場所を作りたいという想いで活動を始めたイトナブ石巻[注1]が主催しています。アプリ開発を通して若者がITに興味を持ったり、自分の夢を持つきっかけになってほしいという趣旨で開催されており、若者と現役のエンジニアやデザイナーが一緒にチームを組むのが特徴です（参加する大人は若者にかっこいい背中を見せるという役目を負っています）。

宮城県内はもちろん、東京や大阪など県外からも多くの方が集まり参加者の数は100名を超え、今年は昨年までと比較して学生が多かったようです。

今年も2つの部門でスタート

1日目は午後から始まり、今年もIT Bootcamp部門とハッカソン部門とに分かれて進められました。

IT Bootcamp部門

「はじめてだけどアプリ開発をやってみたい！」という方のための部門で、講師がサポートすることで3日間でアプリを作ることができるようにします。今回はおもに小学生で構成されるScratchを使う組と、おもに高校生で構成されるCorona SDKを使う組とに分かれて進められました。

Scratch組は、子どもたちが描いたゲームのイラストから素材を作り出してScratchで動くよう組み上げていくということをやりました。講師を務めたイトナブの武山将己さんの「Scratchのやり方を教えるというよりは、Scratchを使って自分の中だけにあった想像物を創造する楽しさを経験してもらえたらなと思ってやりました。『5回敵を倒したら進化させたい！』とか『もっと難しくしたい！』とか、子どもたちが自分でいろいろカスタマイズしていく様を見ているのは個人的にも楽しかった」という言葉が印象的です。大人がやりなさいと言っても子どもの興味を引き出すことは難しく、やはり楽しいと思って自主的に取り組んでもらうことが最初の一歩として大切です。

Corona SDK組には、プログラムの基礎を折り込みつつ、2日間でアプリ開発を学べるようにした教材が講師の中塩成海さんによってイベント前に用意されていました。参加者の皆さんは事前に予習をしてきていたそうです。家に帰ってからもプログラミングを続け、さらには休み時間も惜しんで開発を続けるという、ハッカソン組にも勝るとも劣らない頑張りっぷりのBootcampだったようです。

講師の中塩さんは、第1回目の開催当時は開催場所である石巻工業高校の生徒でした。そこでアプリ開発を体験し、現在はイトナブ石巻のエンジニアとして活躍しています。このイベントで学んだ人が、また次の世代に伝えていくという良い循環が起きています。

ハッカソン部門

ハッカソン部門は最初にアイデアピッチとチームビルドが行われました。今年のテーマは「青春」。会

注1　http://itnav.jp/

場のホワイトボードにはアイデアのヒントとなるように、参加者それぞれの「青春」について書かれた紙が張り出されました。最終的には20のチームができて、最終日の発表会に向けて開発をスタートしました。

ハッカソン2日目

2日目からが本格的なハッカソンのスタートです。初日は夜の懇親会から参加の方も多く、実際に手を動かす作業もここから始まります。

朝から元気よくスタート！

前日の懇親会で夜遅くまで飲み歩いた影響で、皆さん眠い目をこすりながら朝のガイダンスの始まりです。ガイダンスが終わった後、「石巻ハッカソンはこれがなきゃ始まらない」ということでイトナブのフィッシュこと津田恭平さんの掛け声。みんなで「オー！」と拳を上げ元気よくスタートです。

製作風景の紹介

2日目は製作作業の日。筆者（小泉）の視点から製作風景を紹介していきます。

今回のテーマは「青春」。プログラミングをする我々には当然コンピュータに向き合うのが青春。とはいえ、一般に青春という言葉からくるイメージは恋愛色が強いですよね。そこで、この2つを両立するべく「萌えキャラGO」というどこかで聞いたような名前のアプリを作成しました（**図1**）。

筆者はプログラミングはできるけどデザインはできません。初日は懇親会からの参加だったので、危うくソロチームになりそうだったのですが、武山華さんがデザイナーに名乗りを上げてくれました。かわいい女性とチームを組めた時点で、「青春」というテーマ的には十分満足していたのですが、ここはハッカソンなのでチーム作成だけで満足してはいけません。ちゃんと「萌えキャラGO」を二人三脚で作り上げました。

「萌えキャラGO」は聖地（キャラにゆかりの地）で萌えキャラの女の子とのデートがコンセプト。地図上にある聖地に行くと萌えキャラがFacebookメッセンジャーで話しかけてきてくれるというものです。聖地を回ると好感度が高まり、会話も変わっていきます。女の子は集めるものではありませんので、名前の元ネタゲームとはコンセプトはだいぶ違います。青春ですから！

◆図1　萌えキャラGOタイトル

「聖地が生まれるには二次創作の盛り上がりが不可欠」というところから、イラストを投稿すると聖地が作れるようにもなっています。作った聖地に人が来れば、さらに萌えキャラからの好感度が上がるしくみです。

技術的にはSwift、Facebook Messenger Platformを利用しています。ここに武山さんの素敵な絵が加わり、たのしいアプリになりました（残念ながら受賞は逃しました）。

石巻ハッカソン名物カレー

石巻ハッカソン名物牛タンつくねカレー！　カレー部隊は全国から志願してきた精鋭達。みなさん舌鼓を打ちました。筆者（小泉）はおかわりまでしています（**写真1**）。石巻ハッカソンは技術系の人たちだけではなく、こういったバックアップによって成

◆写真1　牛タンつくねカレーを食べる筆者（右）

り立っているのが伝わりました。

発表会

最終日の発表会は昼過ぎから行われました。ランチもそこそこに最後まで開発に勤しむチームもあれば、発表の準備を行うチームもありました。後でわかりましたが、今年は例年以上に発表に趣向を凝らしたチームが多く、そのため事前のリハーサルなどが大事だったようです。

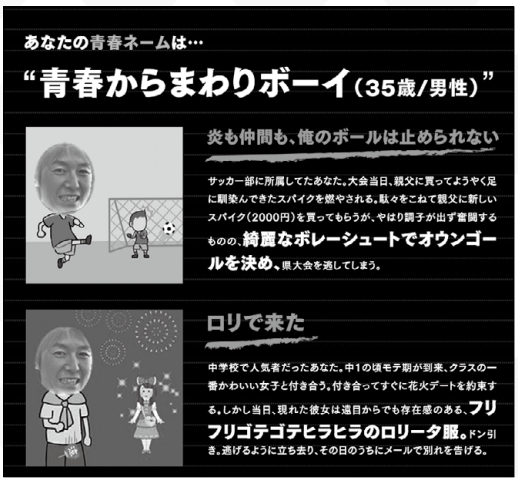
今年の審査員はHack For Japanスタッフで毎年石巻ハッカソンに参加している筆者(及川)と株式会社ゴーガの小山文彦氏の2名でした。

賞としては、最優秀賞、ハッカソン賞、審査員特別賞、青春賞、オール賞が用意されました。このうち、青春賞は参加者の2日目の昼食を用意してくれたカレー部の皆さんに、オール賞はIT Boot Camp参加者に贈られました。

受賞チームをご紹介する前に、今年の総評をお伝えします。

石巻ハッカソンは毎年参加しているので、毎回参加チームのレベルが上がっていくのを目の当たりにしていましたが、今年はさらにパワーアップしていました。機械学習やVRなどの時流に乗った技術を使うチームもありましたが、それも単に流行だからというだけではなく、きちんと実現したいサービスに適した技術を使っていました。技術レベルの高さ以上に驚いたのが、発表時のプレゼンテーション能力です。プレゼンテーション資料の出来や話の上手さもさることながら、寸劇やショートコントを行うなど工夫を凝らしたチームが多かったことに感銘を受けました。寸劇やショートコントは演者が照れながらやっていたりすると、逆効果になることもあります。しかし、今回の参加チームは普段何をやっているのかと思うほど質の高い演技を見せてくれました。アプリケーションやサービスのユースケースなどを説明するには、無味乾燥なスライドでの発表よりも、一種のシミュレーションである寸劇はとても効果的でした。

◆図2　青春ストーリー作成例



審査員特別賞

まずは審査員特別賞です。チームDHが受賞しました。

チームDHの作品は「Happy Days」。これは顔が写っている写真をアップロードすることで、その人物の青春ストーリーを勝手に作ってくれるものです(**図2**)。

しくみは、Microsoft Face APIにより顔を認識し、人物の年齢を推定します。あらかじめ組み込んである世代ごとのストーリーの文章と写真背景に、切り出した顔部分の写真を合成します。画像合成には、Poisson Image Editing[注2]を用いています。

ヤバいで賞

ヤバいで賞は「ヤバい」という形容詞でほめたくなってしまうように、技術などで圧倒する作品に送られる賞です。このヤバいで賞が送られたのは、アダム&フィッシュの「いつか見た夢」です。

チームは技術の力で恋を助けることを目的として、コミュニケーションに注目。テキストチャットの延長としてのビデオチャットが必ずしも盛り上がっていないことを、体験の共有が十分でないことと分析し、VR空間を共有するAndroidアプリケーションを開発しました。

これは、お互いの顔は見せずに、ボイスチャット

注2　SIGGRAPH 2003で発表された画像合成手法。
http://dl.acm.org/citation.cfm?id=882269

とともに自分の体験しているVR空間を相手に共有することができます。URLを共有することで相手はアプリケーションのダウンロードができ、その後、VR動画の視聴を開始できます。顔の向きを変えることでVR空間を移動することができますが、それを完全に相手と同期できます。使われている技術は、Google VR、Deep Link、Web RTC、Google Cloud Platform、TypeScriptなどです。

◆図3　ボットの返信例

ハッカソン賞

実質2位であるハッカソン賞は「石巻生まれストリート育ち」に送られました。

チームが作ったのは、打ち込んだ単語の韻を踏んでくれる（ライミングしてくれる）ボットです（**図3**）。AndroidアプリケーションとFacebookメッセンジャー経由で利用できます。

HerokuとPHPを使って開発されており、ラップのようなやりとりができるのが特徴です。このチームは作品もともかく、映像をメインに使用する発表時のプレゼンテーションが秀逸でした。

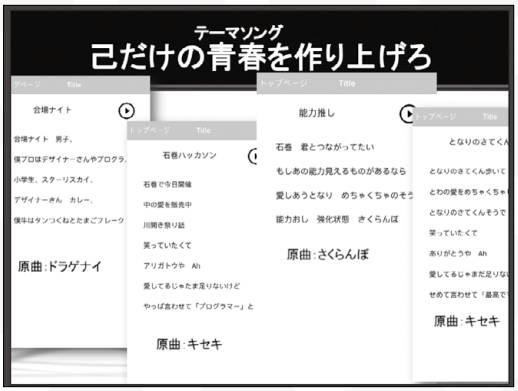
最優秀賞

最優秀賞を受賞したのはIngress GOチームでした。

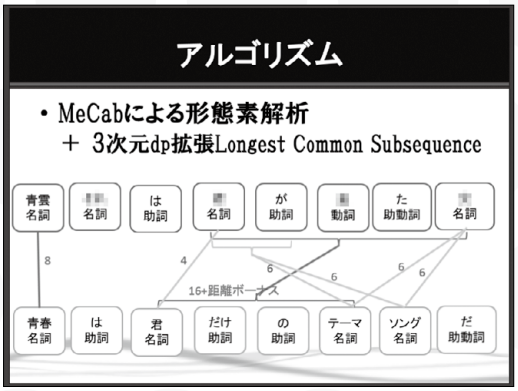
このチームはオンリーソングライター（青春謳歌斉唱）と題して、自分だけのテーマソングを作成してくれるアプリケーションを開発しました（**図4**）。複数のツイートから自動的によく知られているJ-POPの替え歌を作成します。

使用している技術としては、Swiftで書かれたiOSアプリケーションからHeroku上のRuby on Railsにツイートが送られ、そこで替え歌の歌詞が生成されます。そこでは形態素解析エンジンMecabにより品詞分解された後に、3次元dp拡張Longest Common Subsequenceが用いられています（**図5**）。

◆図4　替え歌の例



◆図5　替え歌を作るアルゴリズム



このチームも作品と同じくらいに発表に趣向を凝らしていました。替え歌を作るためにツイートが必要なのですが、修学旅行などに行って友人同士で思い出を作ることを「まず、青春します」と説明するなど、朴訥（ぼくとつ）とした中に地味な面白さがあるプレゼンテーションが印象的でした。また、発表者が実際に作られた替え歌を歌ってみせたのですが、演技なのか本当なのか、とても上手いとは言えない歌声で、会場は笑いの渦に巻き込まれていました。

来年もあります！

これほどバリエーションに富む参加者が集まり、多岐にわたる技術が活用されるハッカソンはなかなかないのではないかと思います。来年も7月末の開催が計画されていますので、ぜひ石巻に足を運んで参加されてみてはいかがでしょうか。SD

温故知新
IT むかしばなし

第60回

リレー式計算機～カシオの実用的な最初の電子式卓上計算機～

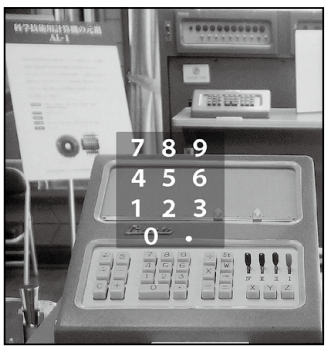
速水 祐（HAYAMI You）　http://zob.club/　Twitter：@yyhayami

はじめに

リオデジャネイロオリンピックにおける日本の400Mリレーの銀メダルは、チームワークによる力の高まりを実感しました。次回2020年は東京オリンピックですが、前回の東京オリンピックが開催された1964年、トランジスタを使った電子式卓上計算機（以下電卓）の歴史は、豊洲の隣の東京晴海のビジネスショウ会場で早川電機工業（現シャープ）、キヤノンカメラ（現キヤノン）、ソニー、大井電気の4社が製品を発表したことから始まります。しかし、ここにカシオ計算機（以下カシオ）の名前がありません。カシオは、この当時スイッチングにリレーを使ったリレー式計算機を販売しており、一定の地位を築いていました。

今回は、このリレー式計算機についてのお話をしましょう。

▼写真1　カシオ14-A型計算機（後にAL-1が写ってます）



最初の電子式卓上計算機とは

1950年代の計算事務の一般的な環境は、個人用のそろばんと、複数の職員で共有して使う日本製のタイガー計算機[注1]、そしてモーターで歯車を回して計算する外国製の電動計算機が並ぶ状況でした。計算業務の多い職場ではその動作音がかなりうるさかったようです。

そんなご時世に、動作音を低減して高速に計算できる計算機の開発に、カシオを創業した樫尾4人兄弟がとりくんだのです。最初にソレノイド[注2]を使った計算機を完成させましたが、機械動作を含むため試作レベルで止まり、商品化はできませんでした。しかし、次男の樫尾俊雄氏は、リレー（後述）をスイッチング部品として使用することを考え、新たな計算機の開発に取り組みました。そして、1957年モーターや歯車類を使わないリレー式計算機「カシオ14-A型（**写真1**）」を製品として完成させ、発売に漕ぎ着けました。14-A型はリレーをわずか342個使用したもので、演算桁数は14桁で、商品名14の数字はこの値のようです。

当時、富士通が1956年に国産初のリレー式商用コンピュータFACOM128Aを完成させました。それを改良したFACOM128Bの演算処理装置で使用されていたリレー素子数は約5,000個（メモリ部を含まない）でした。となるとこの342個という個数はかなり少なく、優れた工夫が多く施されていたことがわかります。

リレー式計算機14-Aは485,000円で発売され、好評を博して計算機としてのシェアを拡大していきました。その後、技術用計算機で開平機能[注3]付きの14-B型、作表計算機TUCと革新的な計算機を続けて開発しました。1962年には、計算の手順をセットしておき、自動的に計算を行うプログラム式計算機AL-1を995,000円で発売し、国内における計算機分野の独壇場を築いたのです。そして1964年を迎えます。

リレーによる計算のしくみ

リレーとは、コイル部と接点部

注1） 手でハンドルを回して計算するもの。
注2） コイルの中心に鉄心を入れ、電気を流してその直線運動を利用する部品。
注3） 平方根（ルート）を求めることを、昔はこう呼んでいた。

リレー式計算機～カシオの実用的な最初の電子式卓上計算機～

が物理的に離れていて、コイルに電流が流れると磁力が発生し(電磁石)、その働きで鉄片を引き寄せ、鉄片につながる可動接点が動くことにより、固定接点と電気的に接触してオン／オフするものです[注4]。デジタル回路は、基本的に0と1の信号で成り立っているので、リレーのオンを1、オフを0とすれば論理演算ができるのです。

リレーによる論理演算の概念を**図1**に示しました。論理積(AND)は、スイッチAとBを直列につなぎ、共にオンになると、コイルに電流が流れてスイッチCがオンになるしくみです。スイッチAかBの片方または両方がオフのときは、コイルに電流は流れないのでスイッチCはオフのままです。

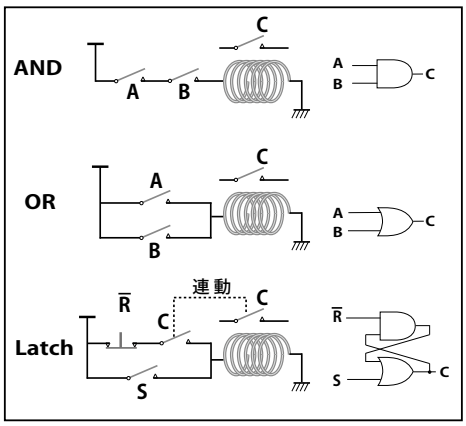
論理和(OR)は、スイッチAとBを並列につなぎ、スイッチAとBのどちらかがオンであればコイルに電流が流れスイッチCがオンになります。

ラッチ(Latch)は、一時的に前のデータを保持します。**図1**の一番下のように、スイッチR(リセット)、スイッチS(セット)を並列にしてリレーで動作するスイッチCをスイッチRと直列につなぎます[注5]。スイッチRがオンの状態でスイッチSをオンにするとコイルに電気が流れスイッチCがオンになり、スイッチSとRの接続側からもコイルに電流が流れるようになります。その後スイッチSをオフにしても、スイッチCはオン(スイッチRの状態)を続けることになります。

このようなラッチを使うことで1bitのデータを一時的に保持できるので、レジスタや一時的なメモリとして利用できます。

リレーに使用されている接点には3種類あり[注6]、前述した回路は基本がメーク接点でしたが、ブレーク接点を使えばNOT演算も実現できます。

▼図1　リレーによる論理演算の概念図



以上のような回路を基本にして、スイッチ部をほかのリレーの接点として回路を組み上げれば、さまざまな演算を実現できる装置を作れます。

カシオの工夫

初めての電子式卓上計算機を実用化するために、さまざまな工夫が成されていました。

◆二重方式のリレー

コイル部が二重になったリレーを使っています。そしてこの独自のリレー素子の接点は、3種類の接点形式をすべて備え、接触の信頼性を上げるためにそれがさらに二重になっているのです。この新たに開発した高機能なリレー素子により、故障率を低減させ、リレー素子数自体も少なくできたのです。

◆テンキー式キーボード

写真1はカシオ計算機14-A(大きさH78cm、W101cm、D42cm、重量120kg)の操作部分のみが写っており、操作テーブルの下に342個のリレーが整列して並んでいます。操作部を見ると現在の電卓と同様な配列で0～9の数字が並んでいます。

テンキーで数値を入力すると表示部に右から数値が表示され、次の数値が入力されると前の数は左にずれ、[+]などの演算ボタンを押すと次の数入力からは新たな数が右から入力され[=]ボタンを押すことで、計算結果が表示されます。昔のキャッシュレジスタ[注7]と比べると、この入力方式は大きく効率と正確性アップにつながるものだったのです。しかし、操作性の大きな違いは、入力の文化ギャップであり、今では当たり前の操作も、当時は丁寧な説明が必要だったようです。

現在の大きく進歩した電子機器も、先人たちが努力して工夫したものの積み重ねによって、ここにあるのです。SD

注4) 本文で説明しているのはメカニカルリレーです。接点部を電子式にしたMOS FETリレーもある。

注5) 電子的なSRラッチ、Dラッチなどとは異なる。

注6) **メーク接点**：電磁石に電流を流したときに接点が閉じる。**ブレーク接点**：電磁石に電流を流したときに接点が開く。**トランスファ接点**：電磁石に電流を流すことで複数の接点を切り替える。

注7) 桁ごとに0～9のボタンが並んでいて、数値を入力するために、それぞれの桁のボタンを押した。

Hardware

ギデオン、サイバー攻撃対策機器「Cyber Cleaner ST」発表

㈱ギデオンは9月30日、サイバー攻撃対策機器「Cyber Cleaner ST」をリリースした。

Cyber Cleaner STはサイバー攻撃対策に特化したセキュリティアプライアンス。外部ネットワークの出入り口に設置して不要パケットを破棄する「Cyber Cleaner」と、ネットワークの内部に置いてギデオンのクラウドサーバから配信されるサイバー攻撃情報（NICT、Kaspersky）をインターネットから受信してCyber Cleanerに反映する「AccessControl」の2台の機器で構成される。

Cyber CleanerはIPアドレスを持たないので攻撃対象となることがなく、またパケットのヘッダのみを見るためSSL/TLS終端をする必要がなく、かつ処理が速いという特徴がある。また、AccessControlはGUIのダッシュボードを持ち、ブラウザから接続することでパケット破棄の設定（国別、ISP別、プロトコル別）を柔軟に行える。

価格は新規＋初年度ライセンスが200万円～、更新ライセンスが20万円～／年となっている。

▲Cyber CleanST「Cyber Cleaner」

CONTACT
㈱ギデオン URL https://www.gideon.co.jp

Software

トレンドマイクロ、「ウイルスバスター」シリーズ最新版を発売

トレンドマイクロ㈱は、セキュリティソフト「ウイルスバスター」シリーズの最新版を9月1日に発売した。

ウイルスバスターは、Windows/Mac/Android/iOS/Fireタブレット向けの「ウイルスバスタークラウド」と、Android/iOS/Fireタブレット向けの「ウイルスバスターモバイル」からなる。

最新版では、指定したフォルダに対するアクセスを監視し、正規のプログラム以外のアクセスを防止することで、ランサムウェアがデータを暗号化することを防ぐ「フォルダシールド」を新たに搭載。さらに、正規のプログラムに偽装する不正プログラムも検知する。

またWindows向けの新機能として、Adobe Acrobat Reader、Javaのバージョンをチェックし、適用可能な修正プログラムがある場合、ポップアップで最新版へのアップデートを促す機能を搭載した。

●おもな製品ラインナップ

製品名	期間	販売価格（税込）
ウイルスバスタークラウド（ダウンロード版）	1年版	5,380円
	2年版	9,680円
	3年版	12,780円
ウイルスバスターモバイル（ダウンロード版）	1年版	3,065円
	2年版	5,637円

CONTACT
トレンドマイクロ㈱ URL http://www.trendmicro.co.jp

Event

サイバーテック、XML・XML DBの国内最大級イベント「NeoCoreサミット2016」開催

11月11日（東京）・11月18日（大阪）、XMLやXML DBの普及啓蒙を目的としたイベント「NeoCoreサミット2016」が、㈱サイバーテック主催で開かれる。

同イベントは今年で9回目。今年のメインテーマは『「XML×Web」～脱WOD・脱DTPによる、Webとの融合～』となっている。本イベントの目的は、さまざまな分野で応用が進む「XML」の最新動向や用途を紹介、啓蒙すること。企業においてのドキュメント管理に課題を感じている来場者に向け、基調講演やパートナー企業による各セッションが行われる。

イベント名に含まれている「NeoCore」とは、サイバーテックが提供しているXML DB製品の名。XML DBはRDBと比べ拡張性が高く、そのため途中でデータ構造が変化することを前提としたシステムに向いている。

●開催概要

	東京	大阪
日付	2016年11月11日（金）	2016年11月18日（金）
時間	14:00～17:20（受付13:40～）	
場所	アマゾンデータサービスジャパン㈱セミナールーム	アマゾンデータサービスジャパン㈱大阪支社セミナールーム
参加費	無料（事前登録制）	

CONTACT
㈱サイバーテック URL http://www.cybertech.co.jp

Service

アカマイ・テクノロジーズ、画像自動処理製品「Image Manager」を発表

アカマイ・テクノロジーズ合同会社は9月28日、Web画像コンテンツの最適化と高速配信を自動化する新製品「Image Manager」の日本での正式展開を発表した。

画像を多用する今日のWebサイトやモバイルアプリには、さまざまな課題がある。Webへのネットワーク接続性は向上しているにもかかわらず、画像の増大によりサイトは重くなり、速度やパフォーマンス、そして最終的にはユーザ体験にも負の影響を及ぼしている。

そんななかImage Managerは、画像の自動最適化を行って、最高の画像品質と高い配信パフォーマンスの両立を可能にすると同時に、派生画像の保存、変換、配信に要するコストや労力を削減、サイトへの反映を迅速化できる。具体的な処理としては、ストレージ上のオリジナルイメージファイルからリンク、ビジュアル関連のポリシーに沿ってファイルを変換、イメージファイルのフォーマットをモバイル表示に適したサイズに圧縮、デバイスごとに適した形式で高速なイメージを配信する、というもの。本製品はアカマイのウェブ・パフォーマンス・ソリューションと連携させて使用することが念頭に置かれており、単体での利用はできないとのこと。

CONTACT
アカマイ・テクノロジーズ合同会社
URL https://www.akamai.com/jp/ja

Report

10月2日、U-22プログラミング・コンテスト2016、最終審査会

10月2日、秋葉原UDX（東京都千代田区）で「U-22プログラミング・コンテスト2016」の最終審査会が行われた。

U-22プログラミング・コンテストは、1980年から経済産業省主催として、アイデアに富んだイノベイティブな人材発掘と育成を目的として開催されてきた。2014年からは民間企業から構成される「U-22プログラミング・コンテスト実行委員会」が主催となっている。

今年は7月1日から作品の募集が開始され252作品が応募、そのうち事前審査・一次審査を通過した16作品が最終審査会にて審査された。経済産業大臣賞それぞれ4部門に輝いたのは次の4作品。

◎経済産業大臣賞　総合部門

『Project Stinger』：チーム藤原重工
（ECCコンピュータ専門学校）

PCで動作する、爽快な操作感の3Dアクションゲーム。UnityやUnrealEngineなど既製のゲームエンジンを使わず、自作のフレームワークを作ってゲーム開発を行った。

▲Project Stingerプレイ画面

◎経済産業大臣賞　プロダクト部門

『わたしのお薬』：青山 柊太朗氏
（ぐんま国際アカデミー中等部）

高齢者の薬飲み忘れを防ぐ「薬服用管理アプリ」。薬服用アラームを家族の声で鳴らすことができる温かみのあるデザイン。重い病気のため、毎日数種類の薬を飲む必要がある祖父のために開発したそうだ。

◎経済産業大臣賞　テクノロジー部門

『WARos』：斎藤 鴻氏
（HAL東京）

IoTデバイスに搭載することを想定した、超軽量カーネルとして開発している「モノのOS」。アセンブリ言語とRustで実装しており、現在x86アーキテクチャ、Raspberry Piでの動作を確認している。

◎経済産業大臣賞　アイデア部門

『Worknote - Organize your Brain』：二ノ方 理仁氏
（東京学芸大学附属竹早小学校）

いろいろな種類の作業をかんたんに記録でき、自分で設定した四象限のマトリクスでわかりやすく表示するTODOアプリ。作業の期限を設定でき、期限が迫るとメールが自動で送信される。

CONTACT
U-22プログラミング・コンテスト
URL http://www.u22procon.com

Report

Black Hat USA 2016＋DEF CON 24 in Las Vegas 視察レポート

Author 中島 雅弘　㈱アーヴァイン・システムズ

Black Hat USA 2016

業界と技術者のセキュリティサミットBlack Hatは今回で19回目。7月30日～8月4日の6日間、米国ラスベガスで開催されました。1日あたり1万1千人以上が参加する大イベントです。見渡すと、参加者のほとんどは男性の技術者と出展している企業スポンサーです。日本のセキュリティ意識は欧米・韓国などと比べて低いという報告を反映するかのように、日本からの参加はそれほど多くはなさそうです。

Black Hatは、キーパーソンによる**(1) ブリーフィング／プレゼンテーション**、デモとハンズオンの**(2) トレーニング**（100以上のコース、1～2日間）、スポンサーによる**(3) ワークショップ**、個人・グループの研究発表の場である**(4) アーセナル**[注1]、そして**(5) エキシビジョン**という構成です。技術者が企業と面談するブースもありました。複数のホールに分かれて一度にいくつものメニューが進行しますので、どれかに時間をかけていると別のイベントに参加できなくなってしまうのが悩ましいところです。

出展企業は、Microsoft、Hewlett Packard、Cisco、Sohos、Symantec、RSA、Tenable、sshなどの大手からベンチャー系の小さい会社多数、Raytheon、Lockheed Martinのような防衛産業で有名な企業も含めて400社ほど。例年、「翌年には50～80%の企業が入れ替わる」という厳しい業界ですが、活気ある成長産業です。

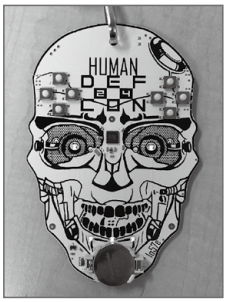
商品やサービス、研究報告は、監視、診断、防衛、暗号化、管理・自動化のツールなど。**監視運用の機械学習を用いた自動化**、**IoTハッキング**、**高度暗号化**[注2]などが、今回の注目技術です。

またアーセナルは、成果の多くをオープンソースとして公開しています。Web脆弱性診断ツール、Raspberry Piを使った攻撃フレームワークなど、すぐにでもセキュリティ診断業務に役立ちそうなものもありました。

▲Black Hat会場

DEF CON 24

ハッカーの祭典DEF CONは8月4日～8月7日の4日間、こちらは24回目の開催です。任意・学術団体、企業30社あまりと、趣味や学術研究色が強いです。公表参加者2万2千人、女性も子供も参加しています。Black HatとDEF CON両方に参加するエンジニアも多数います。



▲DEF CONバッジ。毎年恒例になっている、ハッキング可能なデバイスの入場証。これをハッキングするコンテストもある。

Black Hatと主催・会場は異なるのですが、ネットワークの運用チームは同じスタッフだそうです。開催数日前に会場近辺のホテルなどのフリーWi-Fiは、ハッカーの集会に対し無防備だから使うなという警告が出され、主催者側が用意したハイレベルセキュリティを施したWi-Fiを使えとのことです。

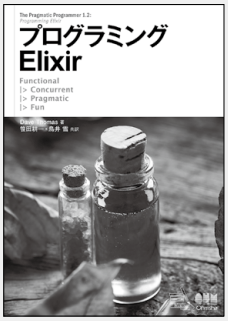
CTF[注3]など30あまりの**(1) コンテスト**が目玉です。子供向けのCTFや、200万ドルを賞金にしたDARPA[注4]が主催するものもあります。ソーシャルハッキング、IoTハッキングなどの**(2) ワークショップ**、研究・開発の成果を発表する**(3) プレゼンテーション**、スポンサー展示の**(4) ヴィレッジ**、Matrixなどコンピュータを題材にした**(5) 映画の上映**、**(6) ライブ**（入場時にもCDをもらう）などと盛りだくさんです。受付、ワークショップやプレゼンテーションなど、DEF CONではすべてが早い者勝ちで、席がなくなれば入場できません。人気イベントは長蛇の列です。

おわりに

セキュリティの分野は多岐にわたり、それぞれが高度で複雑です。システムの問題を突くにも、専門に特化した技術者がそれぞれの持ち分で活躍･連携して挑みます。問題点を共有するサイトも多くあります。サイバーセキュリティの専門家たちは、こうした先端の情報への敏感さや、技術者同士の連携をとても重視しています。日本でも、広くサイバーセキュリティへの意識が高まり、世界で活躍する技術者が増えることを強く望みます。

注1) 弾薬庫とか宝庫といった意味。
注2) TLSも近いうちに使われなくなると言っているところも。
注3) Capture The Flagという、どこかに隠されたキーワードなどを素早く見つけるハッキングゲーム。
注4) アメリカ国防省の研究開発機関。

CONTACT
Black Hat USA 2016 URL https://www.blackhat.com/us-16/
DEF CON 24 URL https://www.defcon.org/html/defcon-24/dc-24-index.html

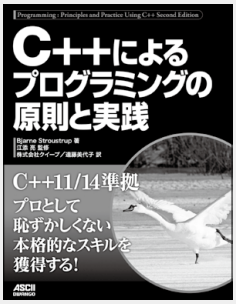


プログラミング Elixir

Dave Thomas 著／笹田 耕一、鳥井 雪 訳
A5判／344ページ
2,800円＋税
オーム社
ISBN＝978-4-274-21915-3

本誌特別企画『[次世代言語] Elixirの実力を知る』でも紹介しているElixirは、高機能な関数型プログラミング言語。Erlangをベースとしており敷居が高い印象だが、使いこなせれば並行処理を効率よく実装できるようになり、耐負荷が求められるアプリでは強力な武器となる。

本書はそんなElixirに関する国内初の言語本。「すべての文法、ライブラリを網羅している1冊」というわけではなく、小さいサンプルコードと要所を突いた説明で、Elixir、引いては関数型プログラミングの良さがすぐにわかるような構成となっている。著者は『達人プログラマー』『プログラミングRuby』執筆の著名Rubyプログラマで、Rubyに親しんだ方には入門しやすい内容になっているのではないだろうか。

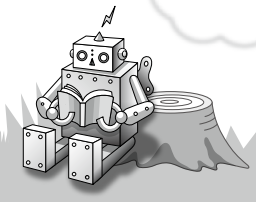


C++によるプログラミングの原則と実践

Bjarne Stroustrup 著／江添 亮 監修／株式会社クイープ、遠藤 美代子 訳
B5変形判／1,248ページ
7,000円＋税
アスキードワンゴ
ISBN＝978-4-04-893051-2

C++の開発者ビャーネ・ストロヴストルップが執筆し、プログラミングとソフトウェア開発技法を、C++のコードを使って解説している。大学におけるプログラミング授業のための教科書として構成されており、著者自身、本書のテキストを使って何千人もの学生を指導してきたそうだ。総ページ数「1,248」という辞書のような厚さと大量の練習問題から成り、すべて読破する場合は年間計画となりそうだ。内容はプログラミング未経験者が読者対象なだけあって、「なぜセミコロンを末尾に付ける必要があるのか？」といった基本事項まで省略せずに解説している。とりあえず動くものが作りたい、という読者の方には不向きだが、1行1行のコードの裏でどのような処理が走っているのかが気になるといった方には、これ以上ない1冊と言える。

SD BOOK REVIEW



みんなのGo

松木 雅幸、mattn、藤原 俊一郎、中島 大一、牧 大輔、鈴木 健太 著
B5判／144ページ
1,980円＋税
技術評論社
ISBN＝978-4-7741-8392-3

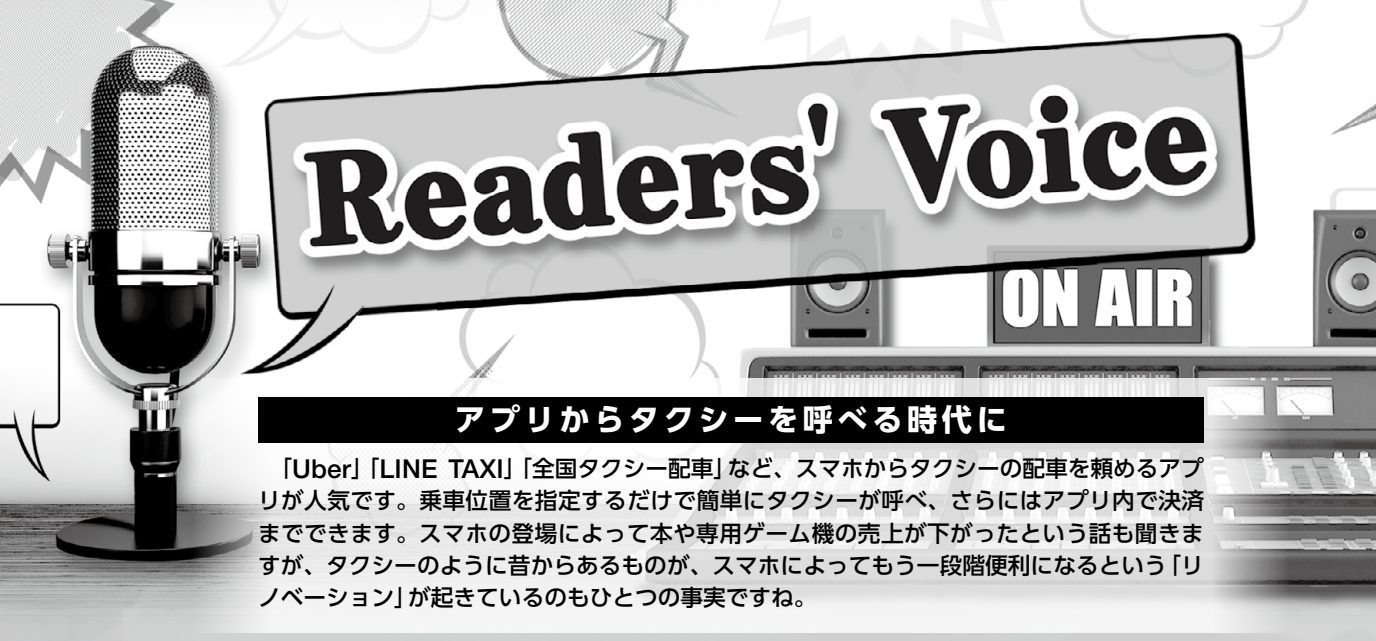
かわいい表紙で話題？ の本書だが、Goユーザ向けの専門書であることは間違いない。なぜ今Goをはじめる必要があるのか。Goを利用するメリットとして「パフォーマンス」「シンプルな言語仕様」「スタイルの統一感」「シングルバイナリの手軽さ」などが挙げられる。また、ほかのプログラミング言語の経験があるエンジニアであればGoはすぐにはじめられる。本書でも触れている「A Tour of Go」(Webで公開されているチュートリアル)を試してみるのが良い。入門を一通り押さえたら本書の出番になるだろう。「チーム開発」「コマンドツールの作成」「マルチプラットフォームアプリ作成」「テスト」など、入門書では扱っていない実用的なテーマで構成されている。Goによる実践的な開発のための書籍としてお勧めできる。

[改訂新版] プロのためのLinuxシステム構築・運用技術

中井 悦司 著
B5変形判／272ページ
2,980円＋税
技術評論社
ISBN＝978-4-7741-8426-5

おかげさまで改訂前の本書は5刷りまで重版し、多くのLinuxユーザがプロの技術を学ぶための手掛かりを本書から得ることができたのではなかろうか。初版からすでに5年が経過し、Red Hat Enterprise Linuxのバージョンも5.5から7までバージョンアップした。

本書でもこれに対応し、systemdなど追加された新機能についても十分な解説を加えた。そうした新しい部分もさることながら、前版のエッセンスも十分に洗練しつつ残している。ストレージ技術もそうだが、問題発生時の切り分けノウハウは、時代が変化しても変わらない絶対的な技術であり、ぜひ継承してほしいエンジニアの知恵の結晶でもある。古くても新しい本書を片手にエンジニア道を邁進してほしい。



アプリからタクシーを呼べる時代に

「Uber」「LINE TAXI」「全国タクシー配車」など、スマホからタクシーの配車を頼めるアプリが人気です。乗車位置を指定するだけで簡単にタクシーが呼べ、さらにはアプリ内で決済までできます。スマホの登場によって本や専用ゲーム機の売上が下がったという話も聞きますが、タクシーのように昔からあるものが、スマホによってもう一段階便利になるという「リノベーション」が起きているのもひとつの事実ですね。

2016年9月号について、たくさんの声が届きました。

第1特集 ログ出力のベストプラクティス

CentOS、Apache/Nginx、MySQL、Samba、Webアプリケーションのログを最大限利用するための出力の設計、設定を考える特集。さらには出力したログの見方、活用方法も解説しました。

知識があやふやなところを補完できたので、たいへん良かったです。
二関さん／東京都

ログを加工していたので、ちょうど良かった。 **jo7oemさん／山形県**

CentOSの5を利用することが多いので、6以降のログ出力を試す際に参考になると感じた。
massakiiiiさん／福岡県

ログの管理については、いろいろ試行錯誤していますが、まだ目視に頼っている部分が多いです（台数も少ないので、何とかなっていますが）。
菊地さん／愛知県

トラブルシューティングにログは必須だと思います。出力レベルやログローテーションの検討は欠かせないと思います。 **永作さん／東京都**

本特集目当てで購入しました。最近、会社でサーバ系のログの話をよく聞くのだが、もともとホストの技術者のせいか、勘所がよくわからない。特集が何かの役に立てば良いのだが。
ねこやまさん／埼玉県

systemdが採用されてから、journalという何かが現れ、よくわからず不安に思っていましたが、今回の特集の第1章だけを読んでよく理解できました。このようなシステムの基盤となっている機能については、定期的に特集してもらえると参考になります。
今井さん／千葉県

ログについて、さまざまな環境でのまとまった知識を得られて良かったという声が多く寄せられました。OS、サーバ、DB、アプリと広く解説したので、仕事ですぐに使えるTipsが1つは見つかったのではないでしょうか。

第2特集 良いPHP、悪いPHP

Web開発では圧倒的なシェアを持つPHPの入門特集記事。文法、ライブラリ、フレームワークについて、良いパターンと悪いパターンを示しながら解説していきました。記事末ではPHPのコミュニティも一挙に紹介しました。

コミュニティについてのまとめが意外と役に立ちました。
サユリナンバさん／神奈川県

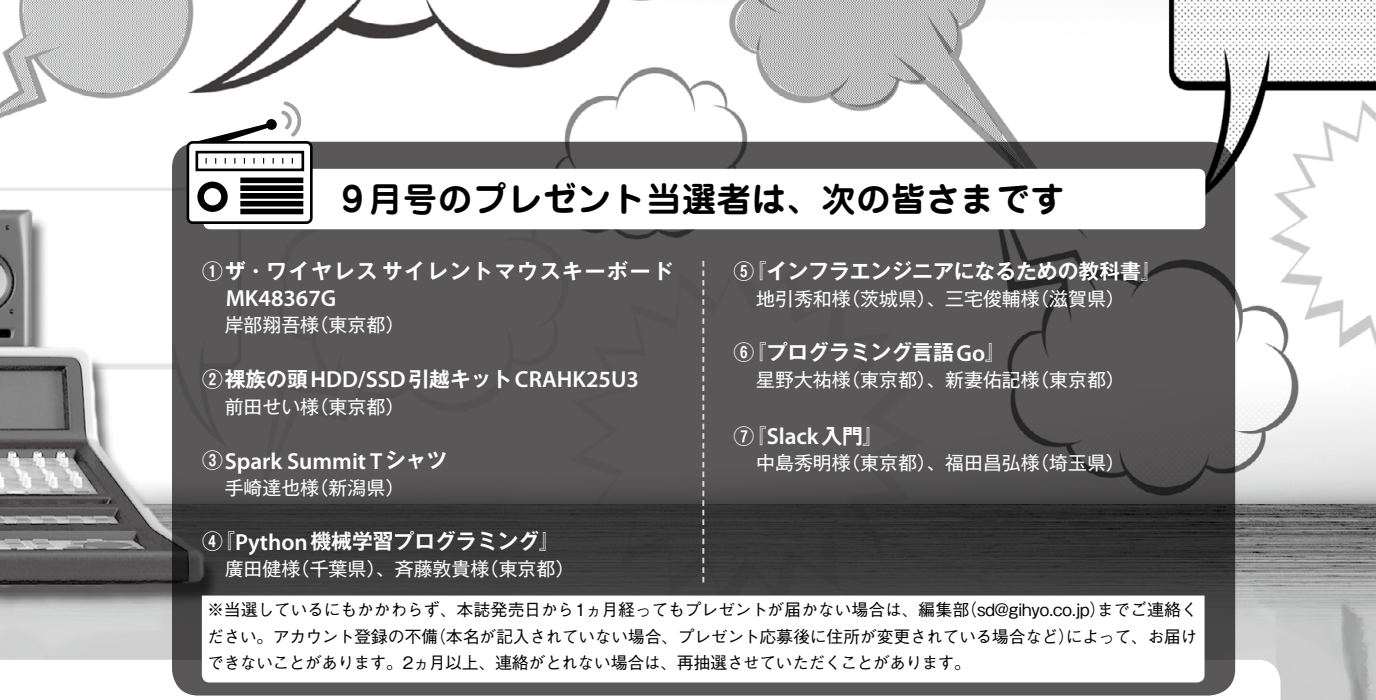
PHPは初心者でも書けるけど、ちゃんと書くのは慣れた人でも難しくて、参考になりました。 **藤田さん／東京都**

PHPの最新情報が見られて良かったです。 **Tayuさん／千葉県**

PHPについて勉強中だったので、非常にためになる記事だった。
TOCさん／東京都

PHPライブラリはPearよりあとはあまり知りませんでしたが、Composerが便利そうなので、導入してみたいと思います。 **NGC2068さん／愛知県**

PHPについては、使ったことがある／使っている／勉強中など、何かしら関わりがあるという読者がほとんどのようです。古い言語ですが、処理系・ライブラリ・フレームワークは今も進化中で、新しい情報を知れて良かったという声もありました。



9月号のプレゼント当選者は、次の皆さまです

①ザ・ワイヤレス サイレントマウスキーボード MK48367G
岸部翔吾様（東京都）

②裸族の頭HDD/SSD引越キットCRAHK25U3
前田せい様（東京都）

③Spark Summit Tシャツ
手崎達也様（新潟県）

④『Python機械学習プログラミング』
廣田健様（千葉県）、斉藤敦貴様（東京都）

⑤『インフラエンジニアになるための教科書』
地引秀和様（茨城県）、三宅俊輔様（滋賀県）

⑥『プログラミング言語Go』
星野大祐様（東京都）、新妻佑記様（東京都）

⑦『Slack入門』
中島秀明様（東京都）、福田昌弘様（埼玉県）

※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。

一般記事　「良いプログラム」のための「良いコメント」

ソースコードにおける「コメント」は、チームでの開発やシステムの引き継ぎに際して非常に重要な役割を持ちます。記事では、どのような個所にどのようにコメントを書けば「良いプログラム」へつなげることができるのかを解説しました。

新人のとき意識していたことを再確認できた。　**ぴょへいさん／大阪府**

若手に読んでほしい。
茅野さん／神奈川県

自分の認識を改める機会になった。
落ち葉さん／千葉県

「作るときは良いが、半年経ったら他人だよ」とアドバイスしていますが、そのときに役立ちそうな記事です。
とーふやさん／神奈川県

以前プログラムを書いていたとき、これで苦労しました。「このコメント必要？」って感じで。今なら、もう少しまともに書けると思います。
あまのじゃくさん／長野県

コードの補完になっているのが、よくわかる。　**桑村さん／兵庫県**

秘伝のタレのように数十年継ぎ足し継ぎ足ししてきたシステムを子守すると、良いコードよりも良いコメントのほうがうれしかったり……。本当はどちらも良いほうがうれしいのですが（笑）。
よっとっとさん／広島県

「コメントをどうするか」はけっこう出てくる話題なのでおもしろかった。
片山さん／東京都

新人のときを思い出した／若手に読ませたい、といった類の声が多かったです。「コメントを適切に書く」ということは、キャリアが長くなるにつれおざなりになっていく部分かもしれませんね。

短期集中連載　乱数を使いこなす【2】

乱数は、シミュレーションやセキュリティ確保に欠かせない技術。その乱数について、作り方／使い方の両面を全3回で追います。第2回「物理乱数ハードウェアを作る」では、デバイスによる乱数生成の方法を紹介しました。

自社製の乱数生成器と比較してみたい。　**lipgtxさん／東京都**

数理的に乱数には興味があったが、今回の物理乱数ハードウェアの話はおもしろかった。　**匿名希望さん／埼玉県**

まだ使いこなせるまでにはいたっていないが、少しずつ理解ができてきている気がします。
yoshitakaさん／神奈川県

少し疑問に感じたのは、それほど大規模でもないようなのに汎用のCPUにハードウェアとして組み込まれていないのはなぜなんだろうか、という点でした。もし理由があるのでしたら、機会があれば解説をお願いします。
出玉のタマさん／大阪府

「難しいけど勉強になる」と好評の本連載。今回取り上げた物理乱数ハードウェアは、聞いたこともなかったという声が多かったです。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
December 2016

2016年12月号
定価（本体1,220円+税）
192ページ

11月18日発売

［第1特集］適材適所で使っていますか?
今学ぶべきNoSQLの教科書
MongoDB、CouchBase、Redis、定番NoSQLの嘘・本当

［第2特集］
プログラマのための文字コード完全攻略マニュアル
HTML・Java・Ruby・MySQLのハマりどころ

［年末年忘れ企画］
温故知新 ITむかしばなしスペシャル
――あっと驚くゲストライター陣によるヤバイ話で振り返るIT業界

［恒例スペシャル年越しマンガ］
特別版ひみつのLinux通信

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「Debian Hot Topics」（第41回）は都合によりお休みさせていただきます。

SD Staff Room

●先月に引き続き新刊の案内です。『ポートとソケットでわかるインターネットの仕組み（仮）』が11月中に発売されます。『小悪魔女子大生のサーバエンジニア日記』のaicoさんのイラストとズバリ核心を突く解説でまとめられたネットワーク技術の入門書です。楽しく学べる本ですのでご期待ください!（本）

●今号の「ひみつのLinux通信」を読みつつ思う。昔の自作プログラムや記事などを見て、「俺ってこんなことしてたんだ」と驚いてしまうのは齢50歳を過ぎてしまったからか!　アセンブラで複数のSPをビシバシ使ったり、int 21hフックしたり、FGALにもTurboCのソース付きでアップしてたなぁ。（退化の戒心、幕）

●週末に腰を痛めた。かがむ必要があるときには腰を伸ばしたままヒザを折り曲げる。常に蹲踞（そんきょ）。力士か!　力士といえば、北の湖関に続き、千代の富士関も亡くなってしまった。子どものころ父と見ていて、相撲が面白いと思ったのはこの2人のおかげ。私にとっての二大横綱のご冥福を祈る。（キ）

●私はたまに、妻がECサイトで買い物をする様子を見ています。妻が品物を熟慮しているときには、サイトのセッションが切れないか密かにドキドキします。また、彼女は注文〆切時刻の1分前に注文確定したりするので「PCとサーバの時刻がズレていたら間に合わないかも」と私一人緊張しています。（よし）

●ピーマンの肉詰め、好きでよく作っています。鶏挽き肉を使ったあっさりレシピもありますが、自分はやっぱり牛豚合挽き派。できるだけ肉厚のピーマンを使うことで、焼いたあとも食感と苦味がしっかり残ってお肉とBest Friend。挽肉にチーズを刻んで混ぜ込むと、さらに本格的なTaste。（な）

●編集部のそばの小路で木蓮の実を初めて見ました。緑の葉を茂らせている中に、鮮やかに赤く色づいた実は、可愛らしい花からは想像できないようなゴツゴツした変わった形で吃驚。その隣には銀白色のふわふわな毛を纏った春待芽（蕾）（はるまちめ）もあってなんとも不思議な光景ですが、早くも冬支度が始まっているのですね。（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
2016年11月号

発行日
2016年11月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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