Special Feature 1 | ChatOpsしてますか?

Special Feature 2 | Ansibleで効率化!

Software Design

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

2016 January 1

2016年1月18日発行
毎月1回18日発行
通巻369号
(発刊303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

Slack

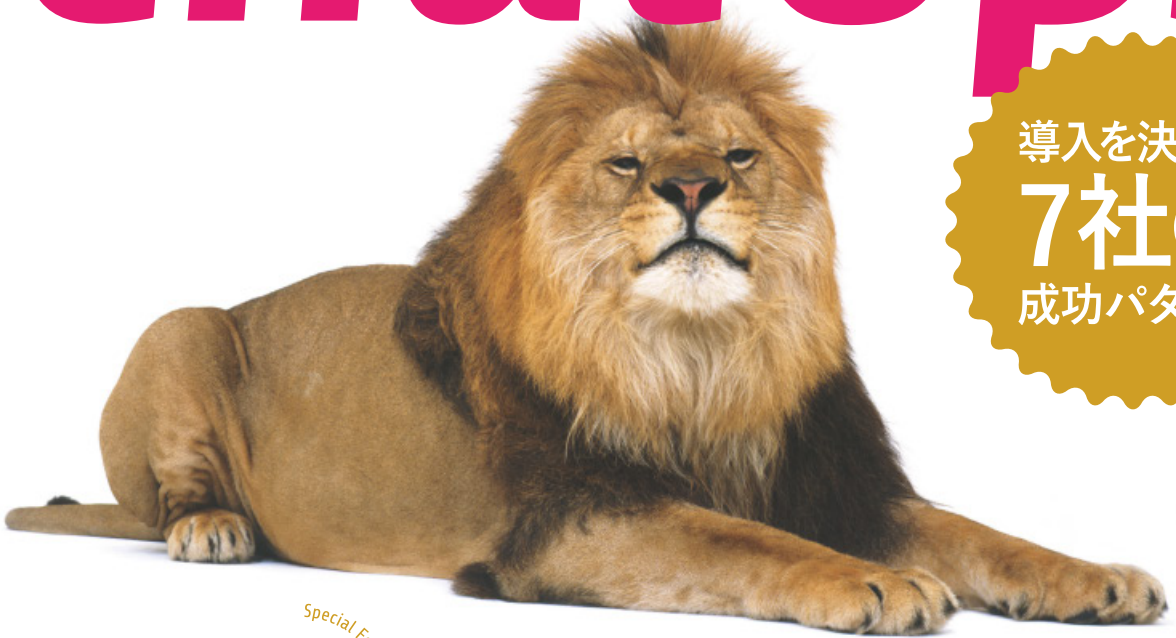
HipChat

Hubot

Special Feature 1

はじまっています。
ChatOps

導入を決めた
7社の
成功パターン



Special Feature 2

手軽さとコード化しやすさが人気!

Ansibleで
サーバ構成管理を省力化
新定番! プロビジョニングツール最新解説

New Serialization

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

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）

2016 January
1
Software Design

第1特集

導入を決めた
7社の
成功パターン

Slack
HipChat
Hubot

はじまっています。
ChatOps

contents

2016
January
1
Software Design

contents

第2特集

手軽さとコード化しやすさが人気!

Ansibleでサーバ構成管理を省力化 063

短期連載

Catch up trend

アラカルト

Column

Development

OS/Network

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

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

BOOKS INFOMATION

お金をドブに捨てないシステム開発の教科書

なぜ、要件定義がうまくいっても
使えないシステムができてしまうのか？

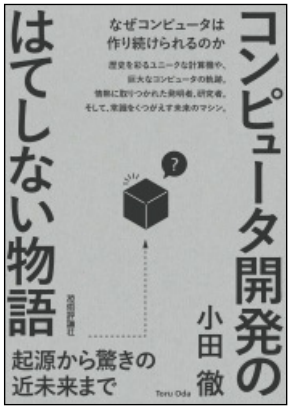
●中川充　著
A5判／192ページ
定価（本体1880円+税）

ISBN978-4-7741-7817-2

家は「一生ものだから」とよく考えて買うのに、なぜ中堅企業でさえ数千万から数億円になるシステム開発では思考停止してしまうのか？
なぜ、要件定義がうまくいってもまったく使えないシステムが出来上がってしまい、お金をドブに捨てるハメになってしまうのか？
システムコンサルタント兼公認会計士という異色の肩書を持つ著者が、“稼げるシステム”の作り方を教えます。
ベンチャーから中堅企業まで50社以上、業務設計・改善から会計監査さらにIPO支援まで20年近いコンサルティング実績があるからこそ書けたノウハウが満載！

コンピュータ開発のはてしない物語

起源から驚きの近未来まで



●小田徹　著
A5判／320ページ
定価（本体1980円+税）

ISBN978-4-7741-7831-8

いまや日常的に使われ、身近にあふれるコンピュータ。
その起源は実に3万5000年前まで遡ることができる。古代からの計算道具が、現在私たちの知るコンピュータに姿を変えるまでには、パスカルやライプニッツといった誰もが知る偉人、チューリングやノイマンなどコンピュータ科学の基礎を築いた先駆者たちの壮大なドラマがあった。現代に続くパソコンの開発競争、最新鋭のコンピュータまで報告しつつ、さらには未来のコンピュータの姿を探ります。

現場で使える［最新］Java SE 7/8 速攻入門

●櫻庭祐一　著
B5変形判／336ページ
定価（本体2880円+税）

ISBN978-4-7741-7738-0

Java SE 7に続きリリースされたJava SE 8において、予定されながらも遅れていたJavaへの最新機能が搭載されました。これらは現場のプログラマーにとって非常に重要な変更点であり、Java7/8はプログラマーにとって重要な意味を持ちます。本書では、コーディング自体も大きく様変わりしたJava7/8を、他のどの本よりもわかりやすく解説します。

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

技術評論社の 確定申告本

平成28年3月締切分

初めてでも大丈夫！ マネして書くだけ 確定申告 平成28年3月締切分

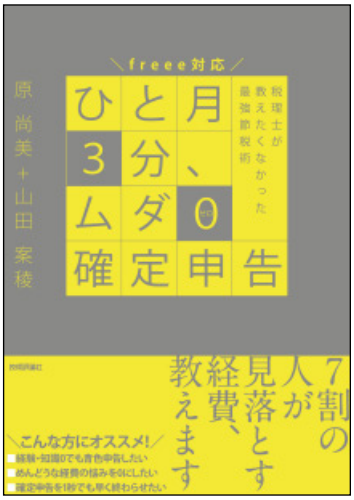
山本宏　監修／A4判／176ページ
定価（本体1,380円＋税）
ISBN978-4-7741-7684-0

家族が働いていたり副業があるために確定申告が必要なサラリーマンのために、確定申告に必要な書類の作成方法を、簡単にわかりやすくまとめました。ほかにもアルバイトやパート、フリーランス、不動産オーナー、年金生活者などが確定申告を行う場合についても、個別のケースごとに詳しく解説。本書を参考にすれば、該当するケースを探して、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などにオススメです！

フリーランス&個人事業主 確定申告でお金を残す！元国税調査官のウラ技 第2版

大村大次郎　著
A5判／224ページ
定価（本体1,580円＋税）
ISBN978-4-7741-7664-2

税金には、シロなのかクロなのかはっきり線引きされていないグレーゾーンがいくつもあります。その境界線を賢く活用することで、確定申告は絶好の節税の機会になります。フリーランス・個人事業主の皆さんが確定申告するときにぜひ知っておいてもらいたいことを元国税調査官の著者が厳選しました。はっきりシロと認められていることでも、納税者に有利になる（＝税金が安くなる）確定申告のやり方を税務署がすすんで教えてくれることはありません。自ら知識を仕入れて、確定申告でトクする人になりましょう！

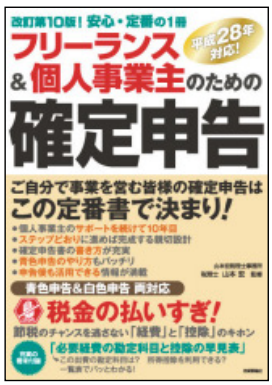


ひと月3分、ムダ0確定申告

税理士が教えたくなかった最強節税術

原尚美・山田案稜　著
A5判／272ページ
定価（本体1,580円＋税）
ISBN978-4-7741-7792-2

「勘定科目なんて知らなくていい」「仕訳なんてしなくてもいい」最高に簡単な確定申告の解説書です。経費で落とせる領収書と落とせない領収書といった悩みどころもしっかり解説。自動で帳簿付けしてくれる話題の全自動クラウド会計ソフト「freee」にも対応したかつてない確定申告本です！



フリーランス&個人事業主のための確定申告 改訂第10版

山本宏　監修
A5判／240ページ
定価（本体1,480円＋税）
ISBN978-4-7741-7665-9

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

第85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。

Open API Initiative

RESTful API 記述の標準化を目指して

現在のWebアプリケーションでは、RESTfulなAPIが事実上の標準として使用されています。しかしこれまで、RESTful APIを記述するための業界標準のフォーマットというものはとくに定められていませんでした。そのため、それぞれのWebアプリケーションやWebサービスごとに独自のフォーマットが使用されており、相互運用性が低いなどといった問題が生じていました。

このような現状を解消するために、The Linux Foundationが主導する形で設立されたのが「Open API Initiative」です。Open API Initiativeの目的は、RESTful APIを記述するためのベンダーに依存しない標準フォーマットを策定することです。設立メンバーにはMicrosoftやGoogle、IBM、Smart Bearといったベンダーが名前を連ねています。

複数のWebサービスを組み合わせて新しいWebサービスを生み出すといった手法は、現在では当たり前に使われています。RESTはその連携のための標準的なしくみとして用いられていますが、もしもこれらのWebサービスのAPIが統一されたフォーマットを持っていれば、それぞれの相互運用性が高まり、Webの可能性をより拡大させることにつながります。

似たような試みとしては「WSDL（Web Services Description Language）」がありました。WSDLはXMLベースのWebサービスを記述するための標準フォーマットであり、API呼び出しのプロトコルやデータフォーマットを規定することができます。さらに、APIを呼び出すためのコードを自動生成することもできるようになっています。

現在はXMLよりもRESTが主流となっていますが、Open API Initiativeが目指すのはまさにRESTful APIのためのWSDLのような存在を作り上げることにあります。とはいえ完全にゼロから構築するわけではなく、オープンソースのAPIフレームワークである「Swagger」の仕様をベースにして、新しいRESTful APIの記述フォーマットを策定する方針とのことです。

「Swagger」とは

SwaggerはREST APIのリファレンスを生成するためのオープンソースのフレームワークです。Swaggerには、RESTful APIを記述するためのプログラミング言語に依存しないJSON形式のフォーマットが規定されています。記述方法は極めてシンプルであり、開発者や設計者がフルスクラッチで書くことも可能ですが、YAML形式で記述したものをJSON形式に変換してダウンロードできるオンラインエディタ「Swagger Editor」なども用意されています。

Swagger Editor以外にもSwaggerにはAPIの実装やドキュメント化をサポートする各種ツールが備えられています。たとえばSwagger UIを使えば、SwaggerのAPI定義ファイルから、自動でHTML形式のAPIリファレンスを生成してくれます。Swagger Coreは、ソースコード中のアノテーションからSwagger準拠のAPI定義ファイルを生成するJava用ライブラリです。またSwagger Codegenでは、SwaggerによるAPI定義からサーバおよびクライアントのスケルトン実装を自動生成することが可能です。

Swaggerの強みは、汎用的なAPI記述用のフォーマットに加えて、これらのツール群による強力なエコシステムを形成している点にあると言えます。

Swagger APIプロジェクトの権利はSmartBear社が取得していましたが、同社がこれをThe Linux Foundationに寄贈したことが、Open API Initiativeの立ち上げにつながったそうです。Open API Initiativeでは、Swaggerの仕様をそのまま採用するのではなく、これを拡張して中立で移植性の高い仕様の構築を目指します。今後は、The Linux Foundationのガバナンスモデルに基づいて、Technical Developer Committee（TDC）が中心となって仕様の策定とメンテナンスを行っていくとのことです。SD

Open API Initiative
https://openapis.org/
Swagger
http://swagger.io/

DIGITAL GADGET vol.205

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

浸透してきた人工知能

古い人工知能と新しい人工知能

古くは、人工知能研究というと、コンピュータチェスや、人工知能を学ぶのに最適だと言われるLISP言語で何か作るなど、実用とは少し離れた分野で探求が進んでいました。しかし現在では、ごくごく身近に存在するものになってきています。

一口に人工知能といっても、さまざまな分野とアプローチが存在します。アルゴリズムによって人工知能的なものとして作り上げられたものや、膨大な機械学習によって振る舞いを導き出されたものもあります。

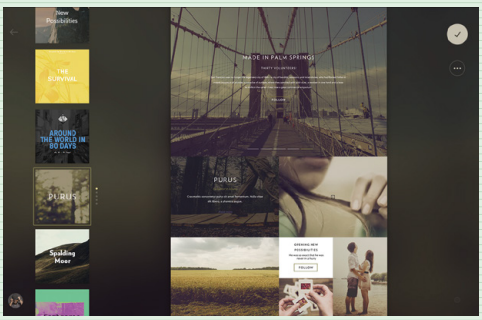
現代では、料理のレシピを考えたり、ファッションのコーディネートを推奨するといった、従来は人、それもある特定のスキルを持った人でなければ対応できなかったところまで人工知能が進出しています。まだまだレベルとしてはトップクラスの人間にはかなわないかもしれませんが、これもそのうち、膨大なデータが蓄えられれば、一般の人が考えるレシピやコーディネートを凌駕するようになるかもしれません。

最近では、不倫相手を探すWebサービスが不正アクセスによって内容を暴露された際、不倫相手の女性のほとんどが、アルゴリズムによって実装されたボットによる会話だったというような、驚きの話題もありました。

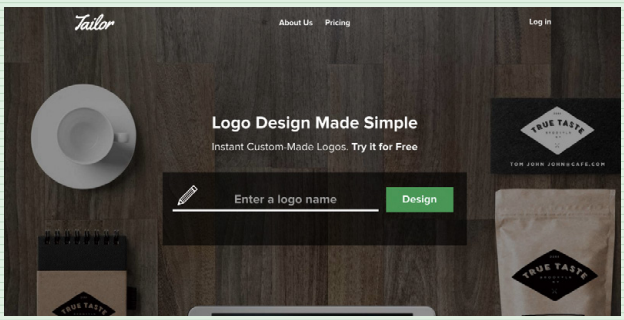
人工知能がテーマになった映画「her／世界でひとつの彼女」では、相手は自分だけかと思っていた人工知能に、実は641人とつきあっていると言われてショックを受けるというシーンもありました。ほかにも、人工知能を搭載した、人間と見まがうロボットを描いた「エクス・マキナ（日本公開未定）」や、人間の知識、知能、意識を全部コンピュータに移してしまう「トランセンデンス」など、映画の中だけでもずいぶん人工知能的な表現が現実化しています。

ロボットや人工知能にお株を奪われ、将来はなくなっていることが予想される職業リストが話題にもなりました。ビジネスマンの間でも人工知能の存在が取りざたされる中、事務の業務などを代行するエージェントツールやスマートフォンに搭載された音声エージェントなど、わざわざ「人工知能」と銘打っていないものも多く、それほど意識しなくても人工知能的なものは生活の中に少しずつ浸透してきているのが実情です。

こういった動向は、Googleが人工知能関係の企業DeepMind／自己学習するdeep Q-network／スケジュール調整のTimeful／自然言語解析のDark Blue Labs／視覚認識のVision Factoryを次々と買収したり、Appleも人工知能を活用した音声認識のVocalIQ／画像認識のPerceptioを買



The GridでWebページ制作中の様子



Tailorでロゴを制作中の画面

※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。

収したことからわかるとおり、続々と体制強化が進んでいます。さらに、人工知能プラットフォームのVicariousが多数の投資を集めたり、IBM WatsonもAPIを公開し、応用を推し進めていることから、おおいに盛り上がっていることがみてとれます。Facebookがこれからサービスを提供しはじめるパーソナルアシスタント「M」は、人工知能技術半分、人手による対応が半分と言われていますが、今後登場する新しいテクノロジーは何らかの形で人工知能の影響を受けているものばかりかもしれません。

人工知能を活用した事例

人工知能には、人間を限りなく模倣するというアプローチの事例と、人間ができないこと、たとえば大量のデータを扱ったり、ものすごく素早くデータ処理をしたりといったアプローチのように、人が扱いきれないものを扱う事例、そして、純粋に楽しむためのエンターテインメントの事例が広がりつつあります。

完全に人の代替とはならずとも、従来、人間が経験のもと、苦労して判断したり、面倒ながらも作業していた事柄で、コンピュータが代替できるものは数多くあるでしょう。仕事を奪われると危惧する人もいますが、面倒なことをコンピュータに任せてしまえば、人間はもっと創造的なことや、娯楽に時間を費やすことができるかもしれません。

人工知能でWebデザイン

The Grid
http://thegrid.io
用意されたテンプレートに無理矢理コンテンツを当てはめるのではなく、さまざまな要望に応じたWebデザインをコーディングなしで作成できる。目的をチョイスをしていけば、最適化されたWebページができるサービス。制作ユーザが用意するのは、画像と文章のみ

人工知能でロゴデザイン

Tailor
https://www.tailorbrands.com/
ユーザが雰囲気などを示したキーワードを入力したり、提示された条件をいくつか選択し、デザインテイストの好き嫌いを入力していくと、ロゴデザインを提示してくれるサービス。人工知能と銘打っているが、それほど知能の気配は感じない

そのほか、どこまで実際に人工知能的な実装が成されているのか不明ですが、人工知能的サービスは各種、各分野に広がってきています。さらに現在は人工知能と銘打っていなくとも、着々と準備中のサービスもあることでしょう。

Random
http://www.random.co/
ネットの世界に新しい話題をもたらそうとする技術

Lemon
https://lemonlemon.co/
人工知能が入会審査をするSNS

artomatix
http://artomatix.com/
ゲームのキャラクタ、テクスチャ素材や情景、環境などを自動生成

VALUE
https://value.heyazine.com/
不動産価格を推定

SENSY
http://sensy.jp/
ファッションアイテムを紹介

AIアナリスト
https://wacul-ai.com/
Webサイトを分析して改善案を提示

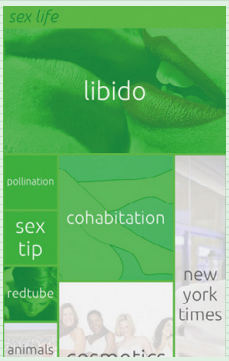
Moov Now
http://welcome.moov.cc/
運動をアドバイスしてくれるパーソナルトレーナー

mitsucari
https://mitsucari.com/
転職マッチングサービス

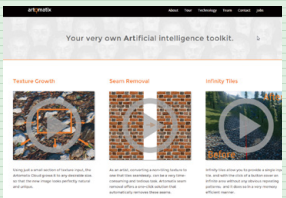
これからの人工知能の活躍の場

人間は、その意志さえあれば、常に学び続けることができます。覚えたり、忘れたり、思い出したり、判断に迷ったり、判断を後悔したり。たまに集中力が途切れたり、焦って間違ったり、罪悪感や高揚感を感じたりと、さまざまな感情とともに「知識」が活用されています。このような人間性を模倣する人工知能もあれば、逆に模倣せず、正確性や信頼性を重視する道を進む人工知能もあるでしょう。

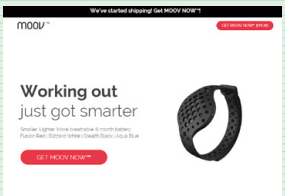
たとえば、音声で対話式コミュニケーションを人工的に構築した場合、



Random：話題提供サービス



artomatix：ゲーム素材の自動生成



Moov Now：運動のアドバイス

mitsucari：転職マッチングサービス

あまりにも素早く回答を返してしまうと、人同士の会話の間とは異なるため、違和感を感じてしまうと言われています。すなわち人間性を模倣する場合は、単に高速化、最適化すれば良いというものでもないのです。

今後は「人工知能に対するUX（ユーザ体験）」、そしてまた逆に人工知能が機械として理解し、対応するための、「人工知能のためのUX」も考慮していかなければいけなくなるのかもしれません。

iPhoneには「Hey, Siri」、Androidには「OK, Google」と呼びかけます。どちらもデバイスやクラウドサービスを擬人化したUIととれます。人にお願いしているかのような自然さがある一方、擬人化してしまうと、人間と同等の受け答えをしてくれるものだと過度の期待をしてしまう面もあります。

人工知能の概念、エージェントの概念をひも解いた古典、マーヴィン・ミンスキーの『心の社会』では、人の意識や思考は1種類しかなく、切り替えたり、割り込んだりすることはできるが、常に1つだということ、また、心を多数のモジュールの集まりだと考え、何かの作業のためにはこれらの階層化した心のモジュールが連携して動くと考えられていることが紹介されています。今はまだコンピュータに人間の心とまったく同じものが実装されるとは思えませんが、研究が進み、少しずつ人の脳や心のしくみがひも解かれていくことに期待したいものです。

そういった人を模倣した実装や考え方のもと、たとえば画像認識技術も単に何が映っているのかを判別するだけでなく、周辺の状況やどういった場面なのか、何が起こっているのかを理解できるようになってきています。みなさんが今まで開発してきたり、使ってきたツールやサービスも「人工知能エージェント」的な要素を追加すると、何かもっと新しいものとして新鮮に、便利に使えるようになるかもしれませんね。SD

Gadget 1

Anki Overdrive

https://anki.com/

人工知能搭載レーシングカーキット

コースタイルで自由に走行コースを作り、そのコースに合わせて人工知能レーサーが走行するオモチャ。ビデオゲームの世界をリアルの世界に持ってきたかのようなオモチャで、コースタイルを増やせば巨大で複雑なコースも実現可能です。車2台、タイル10枚のスターターキットが149.99ドルから。自立式で走らせることも、スマートフォンでコントロールもできます。リモコンカーというよりも、ロボットカーというほうが近いかもしれません。同等の製品でReal FXというのもあります。

Gadget 2

CogniToys

http://www.elementalpath.com

会話をこなすスマートオモチャ

Elemental Pathというスタートアップ企業のCogniToysは、現在は試作段階ですが、もうじき一般販売が予定されているオモチャです。IBM Watsonと接続し、複雑な会話のやり取りができる子供向けの製品で、子供達の「どうして?」「なぜ?」といった疑問に回答してくれます。恐竜型のユーモラスな形状の中には、スピーカー、マイク、バッテリー、ネット接続装置が収められているのみで、事実上の本体はクラウドに存在する、シンプルなデバイスになっています。対象年齢は4歳から7歳、色は3色、現在119.99ドルで予約受付中。

Gadget 3

Mycroft

https://www.indiegogo.com/projects/mycroft-open-source-artificial-intelligence#/

執事のようなエージェントデバイス

Mycroftは人工知能による音声応答システムを利用し、ユーザの声でさまざまなネット上のサービスや、家の中の家電などをコントロールするためのデバイスです。単に音声コマンドで操作するのではなく、インテリジェンスを持って対応してくれるのです。ちなみにマイクロフトとは、シャーロック・ホームズに登場する、イギリス政府に務めるホームズの兄の名前。LEDライトが目と口のように点滅して、ロボットっぽく対応してくれます。一家に一台というよりも、一部屋に一台の勢いで導入してほしいそうです。

Gadget 4

Musio

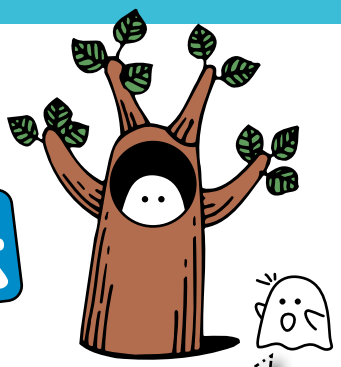
https://www.indiegogo.com/projects/musio-your-curious-new-friend

子供とともに成長するロボット

Musioは、語りかけることによって言語処理能力を伸ばしていくことができる、とのうたい文句の子供用ロボット。単純な会話しかできないバージョンから、大容量のバッテリーを搭載した高機能なバージョンまで3段階のロボットが用意されています。ベースはAndroidで、フルスペックのバージョンが599ドル。開発キットも付属しています。何か役に立つ人工知能ばかりでなく、たよりなく、育ててあげなければ機能的にも成長しないといった、ユーザの扱いに依存するデバイスもこれから増えていくかもしれません。最初の出荷が2016年6月に予定されています。

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

結城浩の再発見の発想法 32



公開鍵暗号

公開鍵暗号

公開鍵暗号とは

公開鍵暗号（Public Key Cryptography）とは、**暗号化と復号化**[注1]で「鍵」を分けた暗号方式のことです。公開鍵暗号では、2つの鍵を分けることで鍵配送問題を解決します。

公開鍵暗号が解決する鍵配送問題を簡単に説明します。アリスが秘密の情報（平文）をボブに送りたいとき、アリスは平文を暗号化して送信します。そしてボブは、暗号文を復号化してもとの平文を得ます。通信経路を流れるのが暗号文だけなら、盗聴者イブは平文を得ることができません。

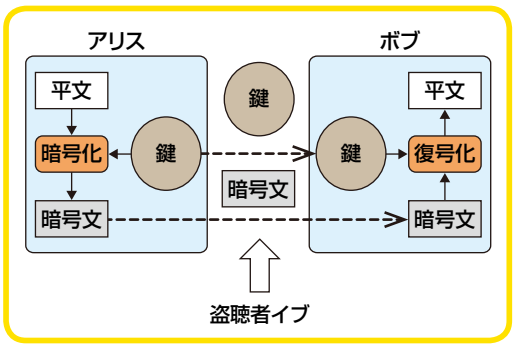
しかし、ここで「鍵配送問題」が起きます。アリスが鍵をボブに配送しなければ、ボブは復号化できません。だからといって、**図1**のように鍵を配送してはいけません。これでは、盗聴者イブにも鍵が知られてしまうからです。**鍵を配送しなければ受信者ボブは平文を得られない。鍵を配送してしまうと盗聴者イブも平文を得られてしまう。**これが鍵配送問題です。

公開鍵暗号では「暗号鍵（公開鍵）」と「復号鍵（プライベート鍵）」を分けることで鍵配送問題を解決します。ボブは暗号鍵をアリスに送ります。アリスは暗号鍵で暗号化を行い、暗号文をボブに送ります。ボブは復号鍵で復号化を行い、平文を得ます。このとき、通信経路を流れる鍵は暗号鍵だけですから、盗聴者イブは暗号文を復号化できません（**図2**）。

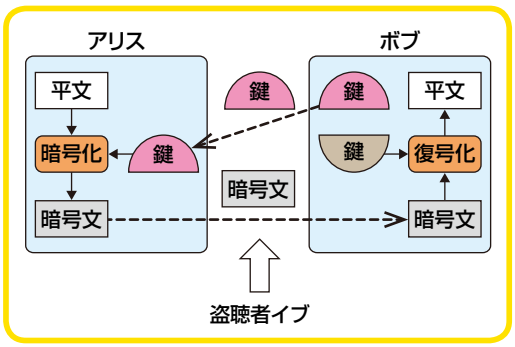
暗号を、鍵の付いたトランクにたとえるなら「トランクを閉める鍵」が暗号鍵で、「トランクを開ける鍵」が復号鍵と言えます。私たちは普段「閉める鍵」と「開ける鍵」を分けて考えません。

注1）通常は「復号」と書きますが、ここでは表記の対称性を優先して「復号化」と書きます。

▼図1　鍵配送問題



▼図2　公開鍵暗号では復号鍵は配送されない



ある鍵で閉めたら同じ鍵で開けるのが普通だからです。公開鍵暗号のすばらしい発想は、**鍵が持つ2つの役割(閉める・開ける)を別々の鍵に割り当てた**ところにあります。開ける鍵(復号鍵)さえ守っていれば、閉める鍵(暗号鍵)のほうは世界中にばらまいてもまったく問題ないのです。

不可分に見えるものはないか

私たちの日常生活で、コインロッカーのシステムは公開鍵暗号に似ています。コインロッカーはお金を入れれば誰でも荷物を入れて「閉める」ことができます。でも、いったん閉めたあとは、第三者がその場所にやってきていくらお金を入れても開けることはできません。開けるときには「開ける鍵」が必要になります。コインロッカーでは、コインが「閉める鍵」となり、閉めたときに得られたキーが「開ける鍵」の役割を果たしていることになりますね。閉めることは誰でもできるが、開けることは「開ける鍵」を持った人に限られる。これは、公開鍵暗号のシステムとたいへん似ています。

公開鍵暗号の発想をもう少し抽象化して考えてみましょう。公開鍵暗号は鍵配送問題を「鍵を2つに分ける」ことで解決しました。抽象化すればこれは**「不可分に見えるものを、役割を明確にして分けた」**という発想と言えます。もしも鍵を「暗号で必要なもの」のようにふわっと考えていただけでは公開鍵暗号は生まれません。「暗号化」と「復号化」という2つの役割を明確にすること、そしてしっかり守らなければならないものは復号化の役割だけであると気づくこと、それが大事だったのです。

会社で仕事を行うとき「忙しいのに誰も手伝ってくれない」という状況を経験したことはないでしょうか。これはいわば「自分」という1つの存在を複数に分けることができないという状況です。これは「不可分な存在」があることを匂わせていますね。もしも、「自分は仕事で忙しい」のようにぼんやりと考えるのではなく、自分がこの仕事で行っているのは「A」という役割と「B」という役割の2つがある(そして「B」についてはほかの人に任せられる)と気づいたなら、作業分担が容易になるのではないでしょうか。

分けて生じる新たな問題

「不可分に見えるものを分ける」という簡単な例として、自分の仕事を人に分担するという話をしました。でもこれはずいぶんナイーブな発想とも言えます。というのは、人に仕事を任せるというのはそれほど単純な話ではないからです。

ほかの人に仕事の一部を任せるためには、その仕事を説明し、また必要に応じて会議や打ち合わせをする必要が生じるでしょう。いわゆるコミュニケーションコストが発生するのです。

公開鍵暗号の場合はどうなっているのでしょう。つまり、暗号鍵で暗号化したものをどうして、別の鍵である復号鍵で復号化できるのでしょうか。

それが可能なのは暗号鍵と復号鍵のあいだに数学的な関係があるからです。公開鍵暗号を使うためには前もって「鍵ペアを作る」という作業が必要になります。そのときに数学的な計算を行い、暗号鍵と復号鍵には一対一の関係が作られます。なので、人間の仕事分担のようなコミュニケーションコストは発生しません。

しかし、公開鍵暗号がすべてを解決してくれるわけではありません。ボブから送られてきた公開鍵が、ほんとうにボブ本人から送られてきた暗号鍵かどうかを確かめなければならない問題は残っているからです。これを解決しようというのが、公開鍵証明書です。公開鍵をめぐる一連の興味深い物語は、拙著『暗号技術入門』をぜひお読みください。

「不可分に見えるものを分ける」というのはすばらしい発想ですが、それですべてが解決するわけではありません。「分ける」という1つの問題の解決法そのものが、「分けたことによって生じる新たな問題」を発生させてしまうのです。

あなたの周りで「不可分に見えるもの」を「2つに分ける」ことで生まれる便利なものはないでしょうか。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3回　Gyamm

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らずに楽できるなら、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

URLで紐付けられた世界で

先月号の記事でも書いたように、製品にも日記にもレシピにも場所にもURLがついており、あらゆる情報がURLで表現できる時代になってきたと言えるでしょう。

ところが、なぜかメールメッセージにアクセスするのにURLを使うのは一般的ではなく、メール専用のアプリケーションやサービスを利用するのが普通で、何かと不便が多い状況になっています。たとえば、会議やイベントの連絡をメールで受け取ったとき、その内容を表現するURLは存在しませんから、ブックマークできませんし、SNSなどにURLを貼ることができませんし、別のページやWikiなどからリンクを張ることもできません。

メールで個人的に受け取った情報を他人に伝えようとすると、メールを転送したり、SNSにコピー&ペーストしたりしなければなりません。あらゆる情報がWeb上で表現されている現在、情報に直接ブラウザからアクセスできないのはたいへん不便であり、**メールで受け取った情報だけが別の世界に存在するように感じられてしまいます。**

なぜメールが使いにくいのか？

メールとWebの相性が悪い原因は次のようなものだと思われます。

- メールはWebより前から存在しており、独自の環境で扱われていた
- メールは個人的な情報のやりとりに使わることが多かったため 公開が前提のWebの世界とは別物と考えられていた

このような事情のため、Webと融合する方法があまり工夫されていなかったのでしょう。

メールの普及はWebの普及よりもずっと前のことですので、しかたがなかったかもしれませんが、Webが発明されてから20年以上経過している現在、いつまでもメールを別世界のまま放置しておくわけにはいかないでしょう。何らかの方法での融合が必要だと思われます。

Gyamm——メールメッセージをWebページ化する

メールメッセージを普通のWebページと同じように扱うことができれば、メールの扱いはずっと便利になるはずです。私はメールを簡単にWebページに変換できるGyamm（ギャム）というシステムを作って運用しています（**図1**）。

Gyammを使うと、メールメッセージをどこからでもアクセスできるHTMLページに変換することができます。

Gyammの使い方

Gyammの使い方はとても簡単です。メールメッセージを[example@gyamm.com]のようなメールアドレスに送ると、送られたメッセージは[http://gyamm.com/example/]から閲覧できるWebペー

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

ジに変換されます。メールが添付ファイルを含んでいたり、HTMLメールだったりした場合でも、普通にWebページとして閲覧できます。

たとえば、受け取ったHTMLメールを[example@gyamm.com]に送ると、**図2**のようなWebページが生成されて[http://gyamm.com/example/20151119003655]のようなURLでアクセスできます。メールの情報を簡単な手間でWebで公開できたことになります。

添付ファイルがある場合は「▶」の右にファイル名が表示されます(**図3**)。

Gyammの利用例

Gyammは、次のようなさまざまな用途に利用できます。

情報の公開／共有

受け取ったメールをそのままの形式で、Webページで公開するのは普通は簡単ではありませんが、Gyammを利用すれば、HTMLメールも添付ファイルつきメールもWebページに変換して、ブックマークしたり他人と共有したりできます。

▼ **図1** SoftwareDesign@Gyamm.comに送られたメールのリスト

▼ **図2** 書店からの案内メール

メーリングリストのアーカイブ

xxxxという名前のメーリングリストのメンバとして[xxxx-archive@gyamm.com]のようなアドレスを登録しておくと、メーリングリストのあらゆるメッセージが[xxxx-archive@gyamm.com]に送られ、その結果全メッセージが[http://gyamm.com/xxx-archive/]に蓄積されるので、メッセージのアーカイブとして利用できます。

フロー情報のストック化

宴会などの案内がメールで送られてくることはよくありますが(**図4**)、こういうフロー情報をWebページのようなストック情報に変換しておくと、後でアクセスしやすくなって便利です。

情報の集約

何かのトピックに関して、いろいろな人から

▼ **図3** 添付ファイルつきのHTMLメール

▼ **図4** 宴会案内メールをWebページに変換したもの

メールを受け取ったとき、それを全部[special-topic@gyamm.com]のようなアドレスに転送するようにしておけば、関連するあらゆるメールを[http://gyamm.com/special-topic/]からアクセスできるようになります。アンケートやレポートをメールで集計するような場合に便利です。

情報の分類管理

仕事に関するメールなど、やらなければならない仕事は全部[masui-todo@gyamm.com]に転送することにしておけば、やらなければならない仕事をすべてGyamm上にリストできます。Gyammではメールが表示されないように指定できるので、終わった仕事に関するメールは非表示にしておけば良いでしょう。同様に、トピックごとに別のGyammアドレスに送るようにすれば、フォルダのように管理できます。

Gyammの実装

Gyammは次のように実装されています。

①SMTPを解釈するGyammサーバをgyamm.comで動かし、[???@gyamm.com]宛のメールをすべて受け取る
②届いたメッセージを解析してHTMLに変換する

SMTP(Simple Mail Transfer Protocol)はインターネットのメール転送で標準的に利用されているプロトコルで、各種のメール送信アプリケーションやPostfixのようなメール転送エージェント(MTA)で広く利用されています。Gyammはメールサーバではありませんが、SMTPを解釈してメールサーバのふりをすることによって、Gyamm.comへのメールを受け取って処理しています。

SMTPサーバの実装

SMTPの仕様はRFC821で定義されています。SMTPサーバには25番ポートを利用して**図5**のように対話的にアクセスできます(太字がユーザ入力です)

正式なSMTPサーバはメール転送などの処理をする必要がありますが、Gyammサーバはメールを受け取ってデータを処理するだけですので、実装はそれほど複雑ではありません。**図5**のようなやりとりができるようにするために、**表1**のような一部のSMTPコマンドを解釈するコードを実装しておけば大丈夫です。

メールをHTMLに変換

DATAコマンドによって メールのメッセージの本体が送られます。`Subject:`や`From:`のようなメールヘッダもこの中に含まれます。メールのヘッダと中身を適切にデコードして解析してHTMLに変換することによって、Webブラウザから読めるようになります。

MIME

日本語を含むタイトルや添付ファイルを利用するとき**MIME形式**へのエンコーディングが用いられています。

メールのSubject(タイトル)に日本語を利用したり、画像や文書などをメールに添付するのは現在あたりまえになっていますが、も

▼**図5** telnetでメールサーバにアクセスしてみる

```
% telnet gyamm.com 25
Trying 133.242.135.184...
Connected to gyamm.com.
Escape character is '^]'.
220 gyamm.com ESMTP GYAMM
HELO example.com
250 gyamm.com
MAIL FROM: masui@pitecan.com
250 ok
RCPT TO: masui@example.com
250 ok
DATA
354 send the mail data, end with .
From: masui@pitecan.com
To: masui@example.com

Hello
.
250 ok
QUIT
221 Bye
Connection closed by foreign host.
%
```

▼**表1** SMTPの仕様の一部

HELO	通信開始
MAIL	送信アドレスを指定
RCPT	受信アドレスを指定
DATA	メッセージ本文を指定
QUIT	通信切断

ともとのメールの仕様ではタイトルも本文もASCII文字しか利用できませんでした。インターネットのメールが世の中に普及しはじめた1980年代は、メールの本文にJISコードを利用することにより、なんとか日本語のメッセージを送ることはできましたが、正式な仕様ではないので微妙なところがありました。また、パソコン通信などの非インターネットの世界では、タイトルや本文にShiftJISの日本語テキストが利用されていることが多く、メールの世界はかなり混沌とした状態になっていました。

1990年代のはじめまではそういう状況が続いていたのですが、1996年にMIME規格がRFC2045として制定されたおかげで、既存のメール配信システムを変更することなく、以下のような手法でさまざまなマルチメディアデータをエンコードしてメールメッセージとして送れるようになっています。

・データ型の指定
Content-Type：フィールドでデータ型や文字コードを指定する
・データのエンコード
画像や文書など任意のデータをテキスト形式に変換する方法を定義
・データの階層的な構造化
マルチパートというデータ型で複数のデータをまとめて扱う。複数のパートをまとめたデータを上位データのパートとすることもできるので階層的なデータ構造を表現できる
・ヘッダの国際化
任意の文字コードをヘッダで利用できるエンコード方式を定義

MIMEのデコード

Gyammでは、MIME形式で送られてきたメールをデコードして、人間が読める形に変換しています。テキストは普通のHTMLテキストに変換し、それ以外のものは添付ファイルとしてリストするようにしています。

MIME形式で送られてきたメールメッセージをHTMLファイルに変換するには次の処理が必要です。

・ヘッダ文字列のデコード
日本語タイトルなどをデコードする
・MIMEパートの分離／デコード
階層的にパートを分離しつつ、データをデコードして添付ファイルなどを取り出す
・埋め込み画像の変換
1つのパートに含まれるHTMLファイルから、別のパートに含まれる画像を参照する場合はContent-IDを利用した特殊なimgタグが利用されるので、これを通常のimgタグに変換する

Gyammの将来

Gyammを使うのは簡単なのですが、SMTPやMIMEの扱いのため実装は若干複雑になってしまい、コロンブス指数が低いものになってしまったのは残念なところです。そもそもMIMEフォーマットはマルチメディアデータを無理矢理テキストに埋め込むための苦肉の策であり、最善のものではありませんし、将来のコミュニケーションはすべてWebベースになるでしょうから、メールは次第に廃れていくと考えられ、Gyammのようなシステムも不要になると思われます。しかしメールが完全になくなるにはまだ10年はかかるでしょうから、それまでは十分利用価値があると思います。SD

コラム「GitHubにコードあります」

本連載でこれまでに紹介したシステムをGitHubで公開しています。ぜひご覧ください。
・GyaTV　[https://github.com/masui/GyaTV]
・Gyump　[https://github.com/masui/Gyump]
・Gyamm　[https://github.com/masui/Gyamm]

軽酔対談

かまぷの部屋

第18献 ゲスト：タナカユカさん

㈲ユニバーサル・シェル・プログラミング研究所
鎌田 広子（かまた ひろこ）
Twitter：@kamapu

タナカユカ（あくやん）さん

㈱Cerevoマーケティング／広報。また、コミュニケーションデザイナとしても活動するなど複数の顔を持つ。幼いころからWebに親しみ、中学生のときからサイト制作の魅力にハマる。デジタルハリウッド卒業後、Webディレクター／デザイナとして複数の会社を経て、昨年独立。その後、2015年6月より㈱Cerevoの広報として、“人とモノの間のコミュニケーションをデザインすること”をテーマに、活動の幅を広げている。旅行とビールが好き。Twitter @akuyan　http://akuyan.to/

自己紹介をお願いします。

「あくやん」こと、タナカユカと申します。現在はおもに㈱Cerevoで広報をしています。製品を多くの方に知っていただき、使っていただくための活動全般を行っています。会社からは基本的に「広報の立場でよいと思うことはどんどんやって」と言われています。

”あくやん”というハンドルネームにはどんな意味があるのですか？

よく聞かれるんですけど意味はないのですよ（笑）。中学生のころに、ずっと使えるハンドルネームが必要だと自分で考えて「あくやん」と付けました。「あ」で始まって「ん」で終わる名前で、読みやすく親しみやすいもので語呂がよいものを選んだのです。母には「名前のアルファベットを逆さにしたんでしょ？」と言われて、妙に納得したのを覚えています。

デジタルネイティブ世代ですか？

両親が新しいもの好きだったので、小さいころから家にパソコンはありました。世代でいうとISDNが普及し始めたくらいのころからWebに触れることが多くなりました。Webに興味を持ち始めたのは、中学生のころです。当時は部活でイラストを描きまくっていたのですが、そのイラストをWebで仲間と共有したことがきっかけで、だんだんとWeb制作の楽しさを知り、将来の仕事にしたいと思うようになりました。そしてデジタルハリウッドに進学しました。学校ではデザインではなく、ディレクションを中心に学んだのですが、それは今も制作の現場でとても役に立っています。

デジハリ卒業後はどんなお仕事に就いたのですか？

卒業してWeb制作会社に入社しようと思っていたのですが、どの会社を選んでいいのか、経験がなくて迷いました。そんな中途半端な気持ちで就職するのは不本意だったので、武者修行のためにいろいろなことを経験するべく、派遣という形を中心に働いていました。おかげでいろいろなお仕事を経験させてもらいました。私はおもにフロント側で、静的コーディングやCSS、HTML、デザインやディレクション担当でした。

Web制作のお仕事から、IoTベンチャーと呼ばれる現在のCerevoへ転職したきっかけは、何ですか？

以前から、デザインは見た目だけではなく、全体の設計などを含めたものだと考えていました。その中で人と、モノやサービスの間のコミュニケーションについてもデザインできるのではないかと思うようになりました。どう人（ユーザ）に届き、どう印象に残すことができるのか、それを仕事とするには広報なのではないかと考えました。ちょうどそのタイミングで、友人から「Cerevoの広報をやってみないか」というお話をいただいたのです。もともと1人で行動することが大好きなので、それを知っている人には広報という仕事に就いたことに驚かれますが、デザインの一環として今の仕事をしているつもりです。

Cerevoのことについて教えてください。

主力商品はLiveShellというシリーズで、パソコンなしでUstreamなどの動画配信サービスに映像配信できる製品です。映像配信にこだわ




る「簡単にきれいな画質の映像を発信したい」という一部の方の熱い想いに応えています。シリーズ累計で約一万台売れています。Cerevoでは一般的なニーズを探るのではなく、「限られた極端な需要にのみしぼって開発している」という形ですね。そういったニッチなニーズを埋めて行くというのがCerevoのコンセプトの1つです。そのほかにも、鍵をひねるだけでWebサービスをハックできる鍵「Hackey」や、Googleカレンダーと連携してアラームを自動セットしてくれる「cloudiss」という製品も出しています。

まさにIoTですね。

個人的にも、今後もっといろんなモノにネットがつながると思っています。IoTという言葉自体は、使う側は意識しなくていいと思っています。将来全部のモノにインターネットが入るようになれば、ビルの建物全体がインターネットに接続されていて、人がビルに入れば持ち物すべてがオンラインになるし、ビルから出れば勝手に施錠されるとか。ゆくゆくは技術的に不可能ではなくなることがいろいろあります。実際に日本にもIoTの波が来ていて、Cerevoの社員数も2014年から2015年の1年間で、10人から約80人に増えました。今年度は十数製品がリリースされています。

開発はたいへんですか？

社内には3Dプリンタも何台かあり、考えたものをすぐに作って試すことができます。また、去年の11月に秋葉原に“DMM.make AKIBA”という、モノ作りのシェアスペースが誕生したのですが、Cerevoもここの立ち上げにかかわっています。

そこでCerevoがイベントを手伝ったり、モノ作りする人のサポートをしています。作るだけではなく実際の販売のことも考えると、あらゆることを想定して動いて行かなければいけませんし、また多くの問題にぶつかります。そういったノウハウを共有していくことで、この場でハードウェアのスタートアップや、日本のモノ作りを応援していければと思っています。

ところで、お1人で海外にも行かれているようですが。

どこに行くのも何をするのも1人でふらふらするタイプです。先日も1人でオランダ、ベルギー、ドイツに周遊旅行に行って、ビール飲んできました(笑)。

聞いていると、お話をするのが得意な方なので、1人で行動するのが好きというのは意外です。

社会人になりたての19歳のころ、ランチは1人で食べることがほとんどというくらい人見知りが酷くて仕事に支障が出ていました。「これじゃいかん」と考え、直すように努力しました。制作の仕事は1人ではできないですし、自分の行動パターンを変える必要に迫られて、やっと……という感じです(笑)。クライアントとコーダー、デザイナなど、いろんな方とのコミュニケーションが必要ですからね。

ビールがお好きなのですか？

両親は、2人ともビール会社に勤めていました。そのため、ビールが冷蔵庫に冷えているのが当たり前といった家庭でした(笑)。社会に出て、冷蔵庫にビールを貯蔵していない方が多くいたことが、私にとってはカルチャーショックでした(笑)。

ほかに趣味はありますか？

小学生までは少女漫画しか読んだことがなかったのですが、中学に入って週刊少年ジャンプや少年サンデーなどの少年漫画も読むようになり、幅が一気に広がっていきました。そのころ、内藤泰弘先生の『トライガン』という漫画に出会って、人生観が大きく変わりました。この作品に出会ってなかったら、もっと性格がゆがんでいたと思います(笑)。

人生を変えたのは漫画だったのですね。今日はどうもありがとうございました。 SD




ツボイのなんでもネットにつなげちまえ道場

第七回

I²Cで通信してみる

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

はじめに

前々回、マイコンで使われるさまざまなシリアルプロトコルの概要を紹介し、前回はSPIについて詳しく紹介しました。今回は、I²C（Inter-Integrated Circuit、アイ・スクエアド・シーと読みます）について詳しく紹介していきたいと思います。

I²Cは、フィリップス（現在のNXPセミコンダクターズ）が開発した規格です。SDA（シリアルデータ）とSCL（シリアルクロック）という2本の信号線（バス）に、複数のデバイスを接続することのできる規格です。同一バスに接続されたデバイス同士を区別するためには7bitのアドレスが用いられ、規格上、1つのバスに最大112個（2^7=128個から予約されている16個を引いた値）のデバイスを接続できます。

今回も、I²Cで接続できるICを題材に説明を進めていきたいと思います。比較的よく使われる、LM75Bというデジタル温度センサを使ってみましょう。

LM75B

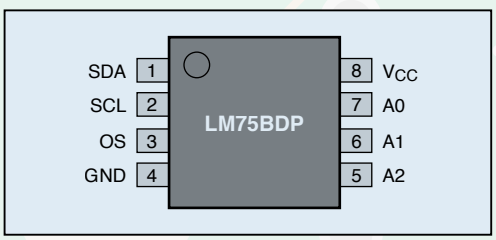
LM75Bはメジャーな部品だけあって、複数の会社からLM75Bという型番のチップが出ています。ここでは、I²Cの規格を生み出した、NXPセミコンダクターズのLM75Bを使ってみることとします。LM75Bには、LM75BD、LM75BDP、LM75BGDといった具合にパッケージ（部品の形状）が異なる複数のバリエーションがあります。しかしどれもブレッドボードにそのまま挿せるような形でピンが付いていないため、スイッチサイエンスで発売している「LM75B温度センサ（I²C接続）[注1]」を使います。

「LM75B温度センサ（I²C接続）」の商品ページには、基板の回路図とデータシートが掲載されています。このデータシートを見ると、LM75Bのピン配置は**図1**のようになっていることがわかります。このA0～A2という3つのピンをHIGHにするか、LOWにするかで、LM75Bのアドレスを設定できます。I²Cの7bitのアドレスのうち、上位4bitは1001に決まっており、続く3bitをA2、A1、A0のそれぞれのピンの状態で決めることができます。スイッチサイエンスの「LM75B温度センサ（I²C接続）」の場合、出荷時の状態では3bitとも0ですので、アドレスは1001000となっています。

I²Cのアドレスは7bit値ですので、Cなどのコンピュータ言語で扱うときには8bit値にして、10進や16進表記をします。マイコンの開発環境によって、この7bit値の上位をゼロ埋めするか、下位をゼロ埋めするかというお作法が異なるので注意が必要です。mbedの場合、最下位ビット（LSB）をゼロ埋めしますので、先ほどの

注1） http:ssci.to/1813

▼図1 LM75Bのピン配置



1001000は、10010000で、0x90と表記します。Arduinoの場合、最上位ビット(MSB)をゼロ埋めするので、01001000で、0x48と表記します。

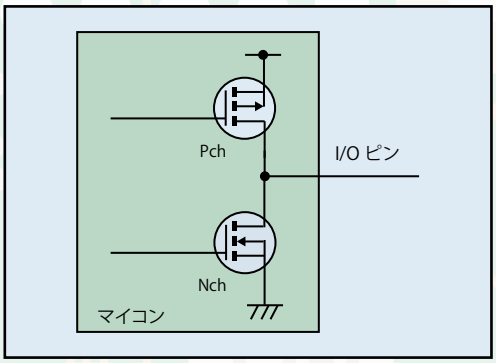
プルアップ抵抗

第四回で記した、マイコンの出力を思い出してみてください。**図2**のように、トランジスタが2つあったと思います。HIGHを出力するときには、電源につながったPchのトランジスタをオンにし、LOWを出力するときには、GNDにつながったNchのトランジスタをオンにして出力を行います。このような構成は、プッシュプルと呼ばれます。

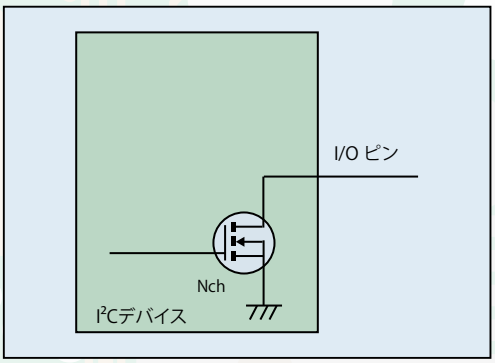
これに対して、I²Cバスに接続するデバイスは、オープンドレインという、出力段のトランジスタが1つだけの構成になっています(**図3**)。オープンドレインでは、GNDにつながったトランジスタをオンにして、LOWを出力することしかできません。HIGHを出力するときには、このトランジスタをオフにして、信号線につながったプルアップ抵抗を使って信号線をHIGHにします。前々回に簡単に触れましたが、I²Cの信号線にはプルアップ抵抗という、信号線をHIGHに引っ張る役割をする抵抗を取り付けます。

I²Cがこのような方法を採るようになっているのは、同じ信号線に複数のデバイスをつなぐことが前提になっているためです。複数のプッシュプルなデバイスが同一の信号線につながっている場合、1つがHIGHを出力し、もう一つがLOWを出力すると、電源とGNDがショートします。すると大きな電流が流れて、デバイスが破損する可能性があります(**図4**)。こういったことを避けるため、I²Cでは、「ワイヤードAND接続」(**図5**)という方法で、複数のデバイスを接続します。接続されたデバイスのいずれかがLOWを出力すると、I²Cバ

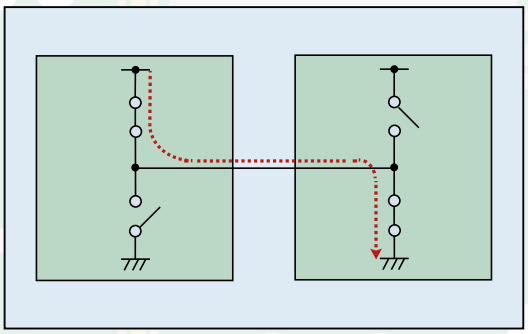
▼図2　プッシュプル



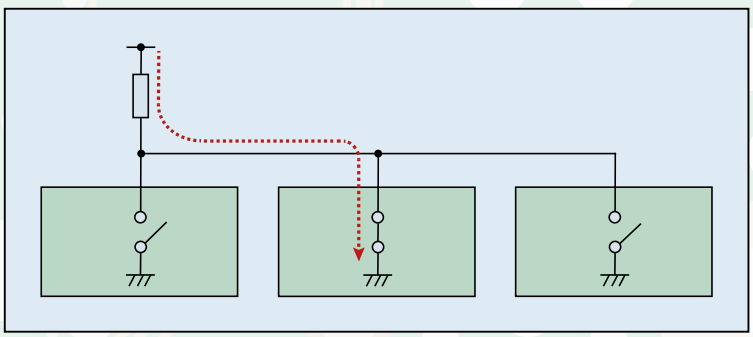
▼図3　オープンドレイン



▼図4　プッシュプルの場合



▼図5　ワイヤードAND接続の場合



スはLOWになります。どのデバイスもLOWを出力していないときには、I²Cバスはプルアップ抵抗を通じて電源に接続されているので、HIGHになります。

つなげてみる

今回必要になる部品は**表1**のとおりです。「LM75B温度センサ(I²C接続)」には、購入時にはピンヘッダが取り付けられていません。はんだづけが必要になります。

部品がそろったら、**図6**のように配線をします。これで、**図7**のような回路ができあがります。「LM75B温度センサ(I²C接続)」には、10kΩのプルアップ抵抗がすでに取り付けられているので、別途プルアップ抵抗を接続する必要がありません。

ソフトウェア

LM75Bのデータシート[注2]に、このデバイスから温度を読み出す方法が記されています。9ページに温度が記録されているレジスタの説明があり、15ページにあるFig 11の手順で読み出せばよいようです。そこで、**リスト1**のようなコードを書いてみました。mbedのAPIについては、ハンドブック[注3]を参照してください。

I²Cでは、データ転送を8bitのデータと、1ビットのアクノリッジ(Acknowledge)の、9bitをひとつの単位として行います。

I²Cの通信は、マスタ(この場合はmbed LPC1768)が開始します。

```
i2c.start();
```

で、**図8**の上段にあるように、マスタはスタートコンディションを作り出します。スタートコンディションというのは、データ転送の開始を表す合図です。続く、

```
i2c.write(addr, cmd, 1, true);
```

で、I²Cのクロック(SCL)に合わせて7bitの通信相手のアドレスを送信、続いて1bitの通信方向を示すbitを送ります。この場合、writeですので、LOW(0)が送信されます。マスタからの転送を受け取ると、スレーブ(この場合LM75B)は、アクノリッジ(以下ACK)を返します。ACKは1bitで、転送されたデータが有効であれば

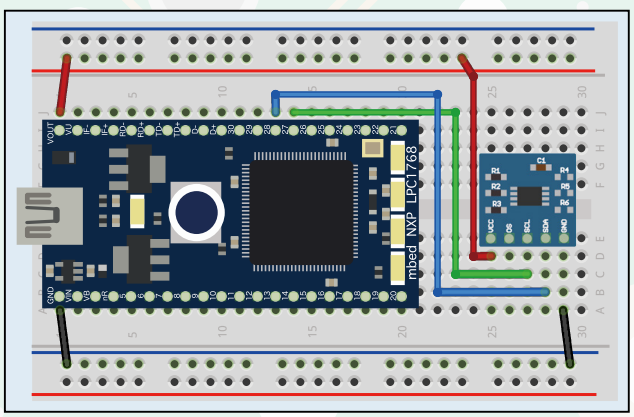
注2) http://www.nxp.com/documents/data_sheet/LM75B.pdf

注3) https://developer.mbed.org/handbook/I2C

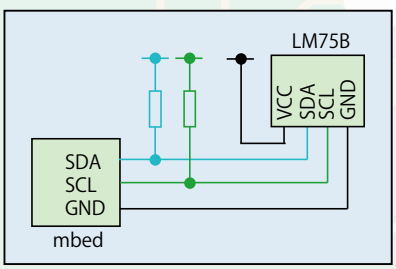
▼表1 部品表

部品名	入手先	参考価格
mbed LPC1768	ssci.to/250	¥5,940
ブレッドボード	ssci.to/313	¥270
固いジャンパワイヤ	ssci.to/314	¥270
LM75B温度センサ(I²C接続)	ssci.to/1813	¥378
普通のピンヘッダ10本セット	ssci.to/92	¥378

▼図6 配線図



▼図7 I²Cの接続例



LOW(0)が送信されます。続いて、温度レジスタのポインタ値である0x00がマスタからスレーブに転送され、これにもACKがスレーブからマスタに返されます。上段の最後にRE-STARTと書いてあるように、リピーテッドスタートを発行します。これは、i2c.write()の最後の引数のtrueで指定しています。リピーテッドスタートは、スタートコンディションと役割は同一なのですが、後に発行するストップの発行をせず、すばやく次のデータ転送に移るために使われます。

次に、温度の読み出しです。ここからは**図8**の下段を見てください。

```
i2c.read(addr, cmd, 2);
```

writeのときと同様に、マスタは通信相手のアドレスを送信したあと、readですのでHIGH(1)で転送方向がデバイスからマスタであることを示します。これを受信したデバイスは、ACKを返します。続いてデバイスは、温度を8bitマスタに送信し、受け取ったマスタはACKを返します。温度は2byteのデータですので、ACKを受け取ったデバイスは下位の8bitを送信します。マスタはi2c.readの引数で指定した2byteを受信したので、ここでACKを返さず、HIGH(1)のNACK(ノットアクノリッジ)を返して転送の終了を知らせます。これで一連の通信は終了ですので、

```
i2c.stop();
```

で、ストップコンディションを発生させます。

ライブラリ

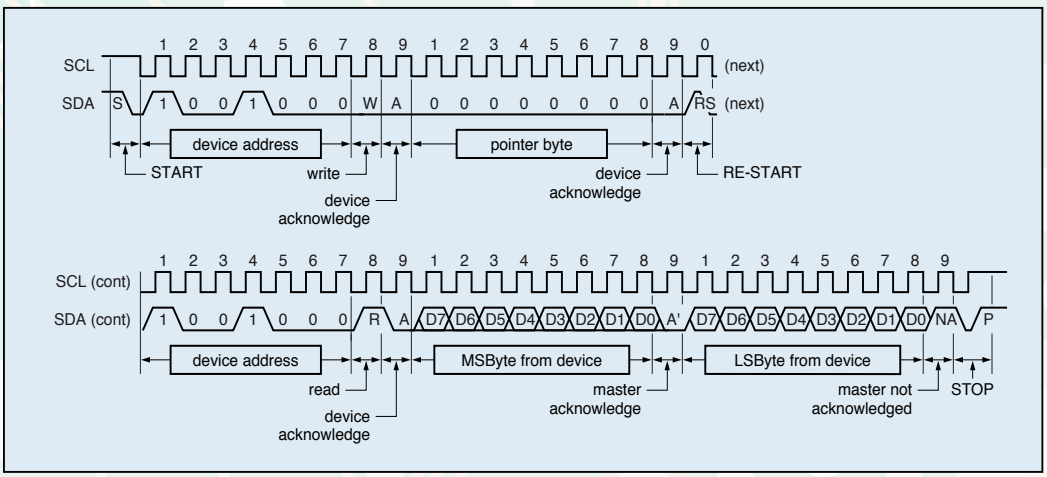
LM75Bのライブラリも、mbedには登録されています。コンポーネントのページ[注4]で配布されていますので、これを使えば手軽にLM75Bを使うことができます。SD

注4) https://developer.mbed.org/components/LM75B-Temperature-Sensor/

▼リスト1　I²Cで温度を読み出すコード

```
#include "mbed.h"

I2C i2c(p28, p27);
const int addr = 0x90;

int main() {
  char cmd[2];
  while (1) {
    wait(1);
    cmd[0] = 0x00;          // 温度レジスタのポインタ値
    i2c.start();
    i2c.write(addr, cmd, 1, true); // I²Cで1byte送信
    i2c.read(addr, cmd, 2);        // I²Cで2byte受信
    i2c.stop();
    float tmp = (float((cmd[0]<<8)|cmd[1]) / 256.0);
    printf("Temp = %.2f\n", tmp);
  }
}
```

▼図8　LM75Bとの通信内容



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1月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートの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 7インチHDMIマルチモニター「LCD-7000VH」

1名

7インチ（WXGA1,280×800ピクセル）のIPSグレアパネルを搭載したHDMIマルチモニターです。入力仕様はHDMI（HDCP対応）／VGAの2つ、電源供給はUSBバスパワー／ACアダプター／単三乾電池の3種類。スピーカー（モノラル）に加え、背面には角度を2段階に調整できるチルトスタンドを装備。デュアルディスプレイは、PCに出力の機能があれば可能です。

提供元 センチュリー　http://www.century.co.jp

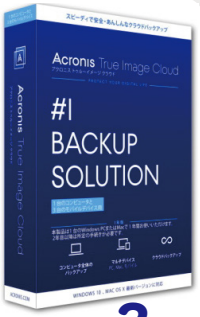
02 世界最小USBハブ「USB2-HUBMC2SS」

3名

Android、Windowsのスマホ、タブレットに取り付けることで、microUSBのポートが「1つ→2つ」になる指先サイズのハブです。PC用USB typeAの機器を使用できる変換ケーブルも付属。

提供元 システムトークス　http://www.system-talks.co.jp

03 Acronis True Image Cloud



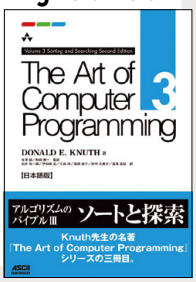
Windows PC、MacをOSまるごとローカル環境とクラウドに、iOSおよびAndroidデバイスの写真・動画、連絡先、スケジュールをクラウドに容量無制限でバックアップできます（本製品で、コンピュータ1台＋モバイルデバイス3台に対して1年間使用できます）。

提供元 アクロニス
http://www.acronis.com

3名

04 The Art of Computer Programing Volume 3

Donald E. Knuth 著



クヌース博士によるアルゴリズムの名著を「アスキードワンゴ」が再刊行しました。第3巻のテーマは「ソートと探索」。要素の並び替えアルゴリズムと、メモリ上から特定要素を集めるアルゴリズムを学びます。

提供元 ドワンゴ
http://info.dwango.co.jp

2名

05 Apache Spark入門

株式会社NTTデータ、猿田 浩輔 ほか 著

並列分散処理基盤「Apache Spark」の入門書。概要からRDD（Resilient Distributed Dataset）による処理のしくみ、導入やアプリケーション開発、周辺ライブラリの活用について解説しています。

提供元 翔泳社
http://www.shoeisha.co.jp

2名

06 Android Wearアプリ開発入門

神原 健一 著

腕時計型デバイス向けのAndroid Wearアプリを開発するために必要な知識をまとめた入門書。標準的な機能を持ったアプリの設計・実装を独力で行えるようにします（開発環境はAndroid Studio）。

提供元 技術評論社
http://gihyo.jp

2名

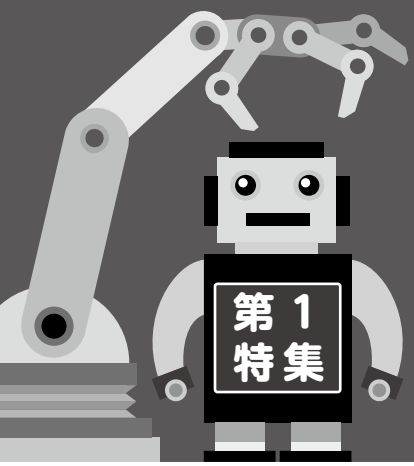
07 ITエンジニアのための機械学習理論入門

中井 悦司 著

機械学習理論を数学的な背景からしっかりと解説していきます。本書中のPythonで書かれたサンプルプログラムを実行し、その結果を見ることで、機械学習を支える理論を実感できるようになります。

提供元 技術評論社
http://gihyo.jp

2名



Slack・HipChat・Hubot

はじまっています。ChatOps

導入を決めた7社の成功パターン

先進的なWeb企業を筆頭に、社内コミュニケーションの中心がメールからチャットへと移りつつあります。チャットには、「テーマを絞った議論が行いやすい」「会話の履歴をたどりやすい」といったメリットがあります。このチャットツールの上に「bot」というプログラムを常駐させ、今日の天気の確認からシステムのデプロイまで、コメント1つで実行してしまうのが「ChatOps」という開発手法。チャットツールにはSlackやHipChatを、botフレームワークにはHubotを使うのがポピュラーです。

本特集ではChatOpsを実際に導入した7つの組織に、その新しい手法をどのように導入したのか、コミュニケーションと開発スタイルはいかに変わったのかを紹介してもらいました。

第1特集

IRCからHubot中心のChatOpsへ

XFLAGスタジオ「モンスターストライク」の裏側

Case 1

モンスターストライクを開発するミクシィのXFLAGスタジオでは、ゲーム上のイベント確認やゲームデータの入れ替えをChatOpsで行えるようにしています。ミクシィのほか数社でChatOpsの導入をサポートした筆者がその事例を紹介しつつ、Hubotをはじめとしたbotソフトウェアの選定ポイントについて解説します。

Author　大塚 弘記(おおつか ひろき)　㈱ミクシィ XFLAGスタジオ
Twitter　@hirocaster　URL　http://hiroki.jp/

ChatOps導入前

本章では、スマートフォンのゲームアプリ「モンスターストライク」を開発する㈱ミクシィのXFLAGスタジオにおける事例を紹介します。筆者は同社でサーバサイドの開発に従事しています。

筆者の所属している部署では、一般にChatOpsと呼ばれるような環境をHipChatとHubot[注1]を中心に構築しています。まずは、そのChatOps環境を構築する以前の環境について振り返っておきたいと思います。

社内専用のIRC

筆者は2013年4月から同社で働いていますが、このころからすでに全社的にIRCが活用されていました。社内専用のIRC(Internet Relay Chat)サーバが稼働しており、エンジニアのみならず、営業職から企画職まで、あらゆる職種がIRCサーバへ接続していました。仕事のちょっとしたやりとりはIRCを利用して行うのが普通でした。もちろん、「誰かお昼一緒に行きませんか？」なんていうプライベートなやりとりもされていました。この時点ですでに、弊社の社員はチャットでコミュニケーションするということに抵抗がなかったと言えます。

IRCとは？

IRCは、サーバとクライアント間でIRCプロトコルをやりとりすることによって稼働するチャットシステムです。古くからインターネットを利用している人には実際に使った経験のある人も多いでしょうが、現在では新規に利用されることは少なくなった印象です。

自身でIRCサーバを用意せずとも、世界中に公開されているIRCnetやfreenodeといったネットワークがあります。クライアントはLimeChatなどが扱いやすいと思います。

チャンネル

IRCには「チャンネル」というチャットルーム(Lineでいうところのグループ)のような概念があります。チャンネルは「#perl」や「#ruby」のようにシャープ記号付きで表記されます。オープンソースプロジェクトのコミュニケーション方法の1つとしてIRCが利用されることが多く、前述したfreenodeには、オープンソース界隈で著名な方たちが現在でも接続しています。何か相談事があればやりとりをしてみるのも良いでしょう。

弊社では部署やチームごとにこのようなチャンネルがあり、「#emacs」や「#vim」など、特定のソフトウェアに関するチャンネルなどもありました。「#2014年新卒」といった同期だけのチャンネルもあったようです。もちろん、チャンネ

注1)　URL https://hubot.github.com/　インストールの方法などはCase2を参照

ルを経由せずとも接続しているユーザ同士、ダイレクトにメッセージをやりとりすることもできます。

このように、弊社は仕事においてチャットを利用するということについて抵抗はなく、すでに文化として根付いていました。もちろん、メールも並行して利用されていました。

IRC Proxyやbot

IRCでは、オフラインのときにチャンネルでやりとりされたメッセージをあとから読み返すようなことができません。これの対策として、TiarraZNCなどのIRC Proxyと呼ばれるソフトウェアを経由してIRCサーバに接続することにより、オフラインのあいだもIRC Proxyがメッセージを受け取って記録できます。これにより、ユーザがオンラインになった際に過去ログを表示できます。そのほかにも、

- 不在時にダイレクトメッセージが来た場合に自動応答メッセージを送信
- 特定の文字列が表示されたらスマートフォンにPUSH通知

など、さまざまな機能がプラグインとして各種IRC Proxyに実装されています。

IRC Proxy以外にも、URLに反応してタイトルを投稿するものなど、チャットメッセージ上で何らかの文字列に反応して処理をする「bot」と呼ばれるようなアプリケーションが、このころからすでに数多くありました。

現代のチャットサービスとの違い

現在多くの企業で導入されているSlackやHipChatといった現代的なチャットサービスとIRCが違う点は、IRC自体はシンプルにメッセージのやりとりしか機能を提供していないということです。もちろんスマートフォン用のアプリケーションなどはありません。IRCでは、ユーザ側がbotなど独自のソフトウェアを利用して、便利な機能を付け加えていったのです。

現代のチャットサービスでは、次のような機能をチャットサービス自身が直接提供してくれます。

- スマートフォン用のアプリケーションおよび通知
- URLからのタイトルや画像の展開
- ほかのサービスとの連携

既存のチャットサービスを利用すれば、すぐにこれらの機能を利用できるようになっています。

IRCでは、botなどの外部のソフトウェアを利用できる知識がある程度必要でした。現代のチャットサービスではWebで設定するだけでさまざまな機能が使えるなど、より多くの人が利用しやすいつくりになっています。仕事や職場でチャットシステムを新しく導入するとなると、“プログラミングの知識を持たない人でも便利な機能を使える”ということは非常に重要になってきます。

外部ツールの情報をチャットに統合

弊社では、GitHub Enterpriseを2013年から導入して徐々に利用部署を広げていったため、Pull Request[注2]スタイルでの開発フローに慣れている人も増えていました。ソフトウェアの自動テストも積極的に利用していたため、CI(継続的インテグレーション)の利用実績もすでに豊富でした。Pull Requestへのコメント通知やJIRAなどのチケット管理システム、Wikiなどのあらゆるツールの更新情報をIRCなどのチャットシステムへ通知してもいました。これにより、普段からコミュニケーションをとっているIRCの画面に、ほぼリアルタイムに更新情報が飛び込んできます。そのため更新情報に早く気付け、対応やフィードバックもすばやくできます。

注2) GitHubにおいて、変更を加えたソースコードの差分を相手のリポジトリに取り込んでもらう要求をするための機能。

このように弊社では、チャットシステムに情報を統合することによって効率的に情報を処理して行動を起こすことができるというメリットをすでに経験していたのです。そのため、現在ChatOpsと呼ばれるような機能を実現しても、スムーズに導入できたのだと考えています。

botの活躍

ここまでで、弊社がChatOpsの機能を本格的に実装する前の環境について話してきました。ここからは、現在実際に使っているbotを、かつてどのように導入して現在に至るまで利用しているのか、その一部を紹介していきます。

導入理由

筆者は2013年10月ごろからモンスターストライクのサーバサイドの開発に携わっています。この時点では、サーバサイドのチームは会社のIRCを利用していました。筆者がかかわり始めてから、GitHubを利用してPull Requestスタイルの開発に切り替えていきました。

Pull Requestスタイルの開発

GitHubのリポジトリにある設定で「webhook」というものがあるのですが、これにIRCを設定することにより、リポジトリでPull Requestが作成されたときなどに、その情報がIRCに通知されるようになります(**図1**)。すなわち、通常業務で利用しているチャットの中にPull Requestの情報がリアルタイムで飛び込んでくることになります。そうすることで、Pull Requestへすばやく反応できるようになります。

Pull Requestを指したコミュニケーション

Pull Requestベースで開発を進めるとなると、「このPull Request」「あのPull Request」というようにPull Requestを指したコミュニケーションが生まれてきます。こういった表現や、「Aという機能のPull Request」というやりとりでは、指しているPull Requestが不明確であり、誤解が生まれることがあります。

Pull Requestとissueには「#1」や「#2」のように、それぞれユニークな番号が割り振られています。そこで、これを利用して「#32のレビューお願いします」という表現をしたほうが、効率的で間違いの少ないやりとりができます。

先ほど例として挙げた「#32のレビューお願いします」というメッセージの投稿があった場合、チャットを閲覧しているユーザはGitHubの「#32」のPull Requestを見ようとします。URLにすると、「http://github.com/ユーザ名/リポジトリ名/pull/32」となります。

チャットコミュニケーションを快適にするために

こういった状況では、ユーザがたどり着きたいURLの画面にいち早くかつ快適にたどり着けるかどうかで、チャットコミュニケーションがスムーズに運ぶかどうかが決まります。

具体的な方法としては、ブラウザから手入力してアクセスしてもいいですし、GitHubのトップページやリポジトリの画面を一度表示して、Pull Requestのリストから探してもいいでしょう。メッセージを投稿する人が、Pull RequestへアクセスできるURLを毎回きちんと貼り付けるようにしておくということも1つの手です。ですが、メッセージを投稿する人もPull Requestを閲覧する人も一番簡単にアクセスし

▼図1　Pull Requestが作成されたとき、IRCに通知される

```
github: [sandbox] hirocaster created master (+1 new commit): http://git.io/vlWzU
github: sandbox/master 66d2ad9 hirocaster: first commit
github: [sandbox] hirocaster created sample-pr (+1 new commit): http://git.io/vlWgX
github: sandbox/sample-pr 416706b hirocaster: Sample PR
github: [sandbox] hirocaster opened pull request #1: Sample PR (master...sample-pr) http://git.io/vlWg1
```

やすい方法としては「#32のレビューお願いします」というメッセージを書き込んだあとに、該当のURLをbotがメッセージとして投稿するようにしておくことです。閲覧者はそのURLをクリックするだけで該当の画面にアクセスできます。

URLだけでなくPull Requestのタイトルも同時に表示してくれれば、Pull Requestの概要まで同時に把握できるはずです。閲覧者によっては「あぁ、この機能は早くマージしないとね！」と思いながら該当のURLをブラウザで開き始めることができます。人によってはまったく無関係のPull Requestですのでスルーする、ということもできるようになります。

このように、コミュニケーションを簡素で的確にするために導入したのがHubotで、それがすべてのはじまりでした。

botの選定

すでにHubotを導入したことを述べましたが、どのようにしてbotを選定したのかについてあらためて振り返ってみたいと思います。ここでいうbotとは、チャットシステムに常駐して特定の処理を実行するソフトウェアのことを指しています。こういったソフトウェアは現在さまざまな言語で書かれており、あらゆるチャットシステムに対応しています。代表的なものを**表1**にまとめました。

選定のポイント

botのソフトウェアを選ぶポイントについて触れておきたいと思います。ポイントを3つほど挙げます。

❶利用したいチャットシステム
（Slack、HipChatなど）に対応
❷利用したい機能のプラグインがある
❸自分が扱える言語で開発されている

最初のポイントは“自分たちが利用したいチャットシステムに対応していること”です。コミュニケーション方法の1つとしてチャットを利用するのが主たる目的でしょうから、botを使うことが目的ではありません。自分たちが快適にコミュニケーションが取れるチャットシステムを選択して、そのうえでサポートしているbotのソフトウェアを選ぶべきです。チャットシステムを改造することは難しいでしょうが、botソフトウェアを改造することは非常に簡単です。

2つめのポイントであるプラグインについて。これは“利用したい機能のプラグインがすでにあること”です。多くのユーザがいるbotソフトウェアであれば、同時に数多くのプラグインが公開されているでしょうから、自分がコードを書かずともすでに存在するプラグインを利用することで、必要な機能の大部分がすぐに利用できるはずです。この点について、あえて気をつけるべきポイントを挙げるとすれば、“プラグインが日本語に対応していること”です（詳しくは事例として後述）。

最後の3つめのポイントは“自分が扱える言語で記述されていること”です。チャットやbotを利用する人の多くは、メインのビジネス

▼表1　代表的なbot

ソフトウェア	説明
Hubot	GitHub社によって開発され、CoffeeScriptで記述されている。多くのチャットシステムへ接続でき、プラグインが豊富で、GitHubを始めとしたさまざまな開発ツールと連携できる。国内でも多くの利用者がいる
LITA[注3]	Jimmy Cuadra氏によって開発され、Rubyで記述されている。複数のチャットシステムへ接続でき、プラグインも豊富
Errbot[注4]	Guillaume Binet氏によって開発され、Pythonで記述されている。表中では一番古くから開発されている。多くのチャットシステムで利用でき、プラグインも豊富
Ruboty[注5]	Ryo Nakamura氏によって開発され、Rubyで記述されている。後発であることからも実装が洗練されたシンプルなコードになっている。作者が日本人であり、日本語の情報が比較的多い。後発のためプラグインは少ないが、必要なものはひと通りある

注3）URL https://www.lita.io/
注4）URL http://errbot.net/
注5）URL https://github.com/r7kamura/ruboty

がほかにあるはずです。bot拡張は多くのメリットをもたらすでしょうが、それにかけるコストは抑えられるものなら抑えるべきです。プラグインを利用するだけで済むのであれば、プラグインを利用する。プラグインに少し手を加えれば済む場合は、慣れている言語であればすぐに手を加えることができるでしょう。慣れない言語ですと、このときに小さな障壁が生まれます。ふと思ったときに機能をサクッと追加できるように、扱い慣れている言語で記述されたbotが望ましいです。

一度導入して本格的に利用を始めると、botを全面的に違うソフトウェアに入れ替えることはなかなか難しいです。よって、今回挙げたようなポイントをふまえて検討していただければ幸いです。

Hubotを採用した背景

ここまでで、botにはさまざまな種類があることと、その選定のポイントを紹介しました。ここからは、筆者自身がHubotを採用した背景について簡単に説明します。これは弊社での導入当初時点(2013年後半)のことです。現在の状況とは変わっていることがあります。

Hubotを採用した大きな理由は、筆者がすでに他社でHubotの導入をした経験があったからです。筆者は今の会社に入社する前、フリーランスのエンジニアとしていくつかの会社にお邪魔させていただき、開発のお手伝いをしていました。このときにすでにHubotを数社に導入して、チャットによるコミュニケーションやChatOpsの推進をしていました。こういった経験から、Hubotの導入が迅速でスムーズに行えたのです。

また、Hubotには「adapter」という概念があり、当時の弊社がメインで利用していたIRC以外にも、現在多くの会社で使われているSlackやHipChatなどにも対応していたことが導入の後押しをしました。将来的にIRC以外のチャットシステムを利用することになった場合には、adapterを切り替えれば、今までのbotをほぼそのままの形で利用できます。実際に筆者の所属するチームでは、ある時期からメインで使うチャットシステムをIRCからHipChatへ移行しており、その際にはHubotのadapterを切り替えるだけでbotの移行作業は完了しました。

さらにHubotはnpmパッケージとして豊富なプラグインがあります。前述したissueの番号(#32など)を展開するスクリプトもすでにあり、導入したい機能を手に入れるまでのコストが少なかったことが採用理由の1つです。

Hubotが、まったく知らない言語ではなく、ある程度扱ったことがあるCoffeeScriptで記述されていることも採用理由の1つです。

最初は、社内にある筆者個人が利用するサーバでHubotを稼働し始めました。その後本格的にチームで利用されるようになり、現在はチームで管轄するサーバに移動しています。

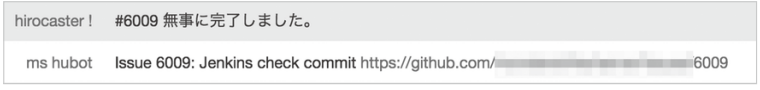
Hubotに何をさせているのか?

ここまでででHubotを採用した理由について紹介しました。ここでは、実際にHubotにどのようなことをさせているのか、その一部を紹介したいと思います。

issueのURL展開

前述したGitHubのissue番号(#32など)をURLに展開するために「github-issue-link」を利用しています。HubotはGitHub社製ということもあって、こういった機能はHubotにあらかじめ用意されており、有効にすればすぐに利用できます。チャットシステムでissueの番号を記述したことにHubotが反応して、該当するissueへのリンクを提供してくれます(**図2**)。

▼図2　issue番号からURLに展開



URLのタイトル表示

issue以外にも、URLの投稿をするとそのページのタイトルをHubotが書き込んでくれるようにしています。これにより、URLが指す情報を自動的に得ることができます。

```
hirocaster !    http://www.monster-strike.com/
ms hubot        モンスターストライク(モンスト)公式サイト
```

当初は「hubot-url-title」[注6]を利用して実現していましたが、Webサイトで利用されている文字コードとしてUTF-8しか想定しておらず、EUC-JPなどを利用したサイトだと文字化けやHubot自身が停止してしまうことがありました。このことから、「hubot-ya-url-title」[注7]というプラグインを新たに開発し、文字コードを自動で判定することで、できるだけ文字化けせずに安全に投稿できるようにしています。

環境の貸し借り

筆者は本番と同じような環境として、ステージング環境[注8]というものを用意しています。たとえば、新しい機能の動作チェックをするとき、チェック中にほかの人が違うバージョンのコードをデプロイしてしまうと動作が変わってしまうので、その環境を一時的に独占して利用したい場合があります。

こういった環境の貸し借りをするのには予約システムのようなものを用意すれば良いのですが、そんな大掛かりなものは欲しくありませんでした。また、チャットシステムにおいて「使います」「空いてますか？」「終わりました」などのコミュニケーションでは明確性に欠け、利用中はチャットを気にしないといけなくなるため合理的ではありません。こういった理由から、ステージング環境を利用するときに、独占するための簡易的な機能を作成しました。

機能はシンプルです。ステージング環境を利用したければ、まずステージング環境の現在の状態を確認します。

```
hirocaster !    staging status
ms hubot        Staging is free.
```

のようにチャット上でコマンドを打つことによって、ステージング環境が現在利用されていないことがわかります。ステージング環境を利用するため、環境をロックしてみます。

```
hirocaster !    staging lock
ms hubot        Change staging status to lock
hirocaster !    #3882 でstaging使います
```

ロック中にほかの人がロックしようとするとすでにロックされているため占有権を獲得できません。ステータス状態も同様に表示されます。

```
[redacted]      staging lock
ms hubot        Staging locked by @hirocaster
```

ステージング環境での検証が終わり、専有を解除するときはアンロックします。

```
hirocaster !    staging unlock
ms hubot        Change staging status to free
```

これで、ほかの人がステージング環境の占有権を入手できるようになります。

このようにステージング環境を利用するうえでは、必ず占有権を取得してから利用するというルールを運用しています。Webの予約画面

注6）URL https://github.com/dentarg/hubot-url-title
注7）URL https://github.com/hirocaster/hubot-ya-url-title
注8）新しい機能や開発中の機能の動作チェックなど、開発中に利用する環境のこと。

などを作らずとも、数行のスクリプトを書くだけで機能が実装できました。この機能では、botが環境のロック状態を保持しているので、デプロイシステムなどと連携して、ロック中はほかの人がデプロイできないようにするなど、さらに安全性を高めることができます。現状ではステータスを通知する機能だけですが、人間側がルールを守ることで事故は一度も起きていません。

みなさんの現場でも同じような事情を抱えていれば、こういった方法で解決できるかもしれません。

イベントのスケジュールを確認

筆者の所属するチームが開発しているのがスマートフォンゲームアプリなので、ゲーム内で行われるイベントのスケジュールが、サーバなどの負荷に影響を及ぼします。ですので、「今日のイベントがなんなのか？」「現在開催されているイベントは？」などイベントの一覧をすぐに確認できるようにしています。あまり詳しい内容をお見せできませんが、botがスケジュールデータを参照して投稿しているだけの簡単なものです。

デプロイ

ChatOpsの代表的な機能としてデプロイがあります。もちろん弊社でも、ChatOpsでデプロイが実現できるようになっています(**図3**)。

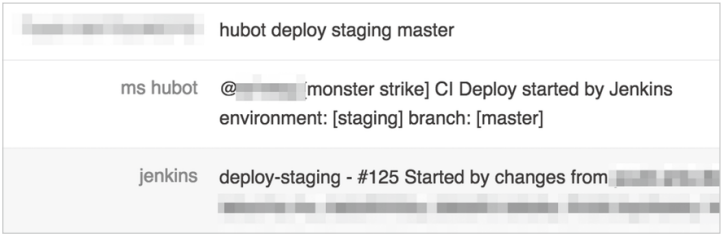
弊社では複数の環境を構築しているので、チャットの中でデプロイするブランチと環境を指定すると、botが反応してJenkinsのJobにパラメータを構築してリクエストを送ります。これにより、デプロイの処理をするJenkinsが処理をスタートするようにしています(**図4**)。

この環境は段階的に構築しました。当初からデプロイではCapistranoを使い、スクリプトによって自動化していました。具体的には、プログラマがサーバからcapコマンドを利用してデプロイしていました。

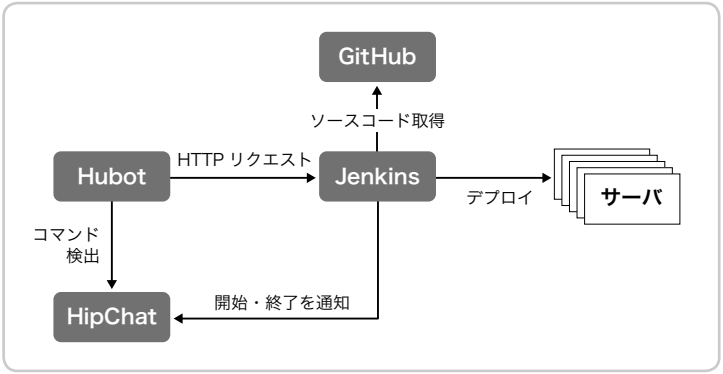
これを、JenkinsのJobとしてコマンドを発行することで、デプロイできるように変更しました。通常はJenkinsのWeb UIなどからJobをスタートさせますが、botからHTTPリクエストによってもスタートできるようにしました。このときbotが行っているのは、HTTPリクエストの構築と送信だけです。デプロイの大部分はJenkinsによって実施されています。

Jenkinsを利用するメリットは、Jobの同時実行などを制御する部分として、Jenkinsのもともとの機能を利用できる点です。デプロイ処理中のコンソールやログもJenkins上にきちんと残ります。また、処理の開始と終了をJenkinsが通知してくれるように設定しています。

▼図3　ステージング環境にデプロイしている様子



▼図4　デプロイシステムの構成概要



ゲームデータの入れ替え

筆者の所属するチームではゲームのパラメータやデータを確認するために、各開発環境のゲームデータをChatOpsで入れ替えられるようにしています。

ゲームパラメータを作っている担当者はデータを作り、データをリポジトリに登録します。そのあと、確認したい環境へ特定のコマンドを実行します。するとbotが、データ投入を行うJenkinsのJobへ、HTTPリクエストを作成して投げつけます。デプロイと同様にJenkinsのJobが先ほど作成されたデータを投入し始め、終了を通知します。

これにより、ゲームパラメータを作成している担当者は、プログラマの手を借りることなく、指定した環境のゲームパラメータを入れ替えられ、動作の確認などができるようになっています。

運用面での工夫

現在botを運用している中で工夫している点についてもいくつか紹介します。

Updateはサーバに入らない

botに対してできるだけ気軽に機能を追加できることはもちろん、運用負荷を下げるためにbotの機能／追加／削除といったUpdateに関しては、サーバにログインしなくても実施できる環境を構築しています。

筆者のチームで稼働しているHubotは「forever」[注9]を経由して立ち上げています。これによりHubotのプロセスがエラーなどで落ちたとしても、foreverによってHubotが再度起動し、チャットシステムにログインしてくるようになっています。

Hubotにはupdate.coffeeというスクリプトがあり、これを有効にすることによって、チャットシステム上で「hubot update」と投稿すると、hubotが「git pull」や「npm update」を実施してくれます。updateがすべて終了したあとに「hubot die」と投稿するとHubotは自分自身のプロセスを終了させます。これをforeverが検知して、再度Hubotを起動してくれます。

HubotのコードはGitHubを利用してチームと共有しています。新たに機能を追加したければPull Requestを送り、Mergeしてもらいます。Merge後は、前述のようにチャットシステムで一連の投稿をするだけで、Pull Requestを送った機能がデプロイされ、実行できるようになります。

特定ルームのみで実行されるようにする

デプロイなどをはじめとした周知されるべき、記録されるべき機能などは、Hubotがデプロイルームでしか実行しないよう実装をしています。仮にHubotのアカウントに直接話しかけて実行できるようにしてしまうと、ほかのチームメンバが知らないところでデプロイが実施されることになってしまいます。このようにbotが特定の影響下でしか反応しないなど運用上の配慮をしています。

おわりに

今回は筆者のチームで運用している、Hubotを中心としたChatOpsの事例を紹介しました。みなさんの現場でも使えそうな内容があったら、積極的に活用してみてください。

今回導入したPull Requestスタイルの開発ワークフロー自体にご興味を持った方は、筆者の著書『GitHub実践入門』(技術評論社、2014発行)をぜひ参考にしてください。ワークフローからGitHubの使い方、CIなどの連携についてまで詳細に解説しています。SD

注9) Node.jsで書かれたスクリプトをデーモン化するツール。URL https://github.com/foreverjs/forever

第1特集

Slack＋Hubotで環境構築解説

ChatOpsで開発から社内交流まで活性化したスピカ

Case 2

本章ではスピカでのSlack＋Hubotによる環境構築の方法紹介、サービス開発基盤とモバイルアプリケーション配布のしくみを解説します。さらに、社内コミュニケーションの改善にChatOpsがどのような効果を上げたのか紹介します。

Author 本寺 広海(もとでら ひろみ)　㈱スピカ　Twitter @__moai

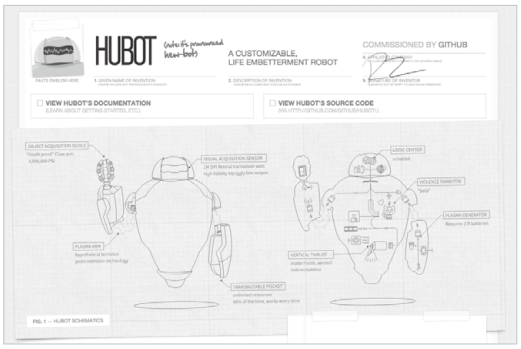
ChatOps環境を作ってみよう!

当社スピカでは、Slack上でHubot（ヒュボット）というChatOpsのフレームワークを利用してChatOpsを実現しています。本稿では、まずHubotとSlackの連携による導入方法を解説します。

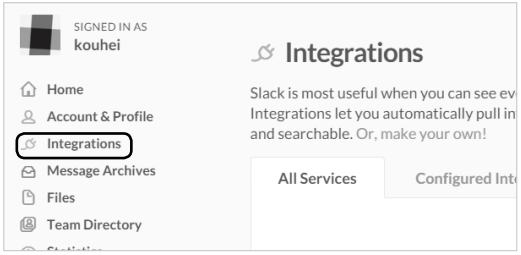
Hubotとは

HubotとはGitHub社が開発したMITライセンスで公開しているオープンソースのChatOpsフレームワークです(**図1**)。Node.js上で動作するように実装されており、CoffeeScriptで独自のHubotスクリプトを記述できます。SlackやHipChat、ChatWorkなどのさまざまなチャットツールに対応しています。多くの方が利用できるという点で非常にお勧めです。

▼図1　GitHubで作られたHubot



▼図2　[Integrations]の選択



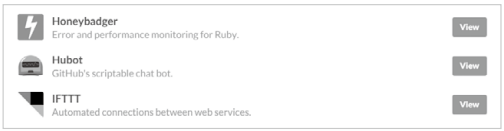
Slackに Hubotを追加する

Slackにユーザ登録し、Sign Inすると左ペインにメニューの一覧が表示されます。そこで[Integrations]を選択します(**図2**)。

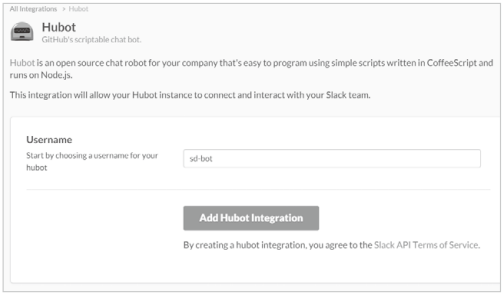
そうすると画面にSlackで連携できるIntegrationの一覧が表示されますので、Hubotの欄の[View]ボタンを押下します(**図3**)。

Usernameの入力欄がありますが、ここで入力したHubotの名前がSlackのチャット上での名前になります(**図4**)。名前を入力して[Add

▼図3　Integrationの一覧からHubotを選択



▼図4　Hubot名の入力



Hubot Integration］を押下します。

次の画面で、下記に示す項目をそれぞれ入力、［Save Integration］で保存します（**図5**）。

・Setup Instructions：HUBOT_SLACK_TOKEN={トークン文字列}
・Integration Settings
　API Token：Hubotとの連携時に必要なトークン文字列
　Customize Name：Hubotの名前
　Customize Icon：Hubotのアイコン画像
　First & Last Name：Hubotの姓と名前
　What this bot does：botの用途を記述する
　Channels：Slackへ追加時に自動的に参加するチャンネル（変更可能）

これでSlackのチャンネル上にHubotが現れます。しかし、まだスケルトンでしかないので、後ほど生成するHubotに紐付けする必要があります。

Hubot環境構築

Node.js 環境準備

Hubotは、Node.js上で動くので環境構築が必要です。その方法はOSやツールによって異なるので、Webでの情報や書籍などを参照ください。必要なものは次のとおりです。なお、今回はNode.jsの環境構築は割愛します。

・OS X環境の場合（wget、Homebrew、node.js package）
・Linux環境の場合（wget、node.js package）

Node.js 動作確認

それぞれのOSの環境にインストールが完了後、動作確認をします。次のコマンドで行います。

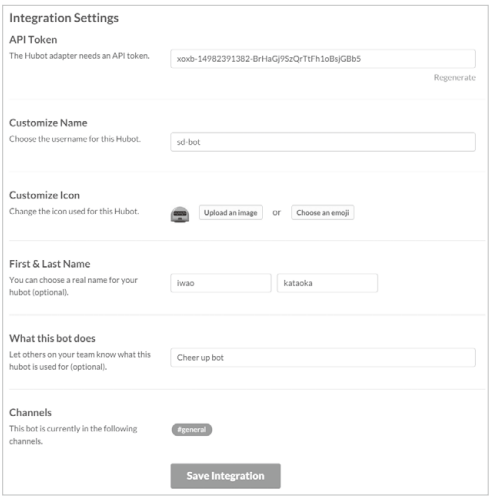
```
$ node -v
```

バージョン情報が表示されます。

npm動作確認

Node.jsの環境の構築では、同時にnpmもインストールされます。npmとはNode.js上で動作するモジュールを管理してくれる管理ツールです。このnpmを使ってHubotをインストールするので、npmの動作確認をします。次のコマンドで行います。

▼図5　Hubotの基本設定の入力と保存




```
$ npm -v
```

バージョン情報が表示されます。

Hubotの導入に必要なnpmモジュール

Hubotの生成にはnpm経由で次のモジュールをインストールします。

・yo：雛形作成コマンド
・hubot：Hubot本体
・generator-hubot：yoを利用したHubotの雛形生成
・coffee-script：CoffeeScript本体（bot動作記述用）

これらは次のコマンドを入力すると導入ができます。

```
$ npm install -g yo hubot generator-hubot coffee-script
```

Hubot生成

Hubot用のディレクトリを用意し、その中で次のコマンドを入力します。

```
$ yo hubot
```

対話形式で必要な情報を入力していきます。今回はSlackで連携することを前提としているのでBot adapterはslackとしていますが、HipChatやChatWorkの場合、それぞれのadapter名を入力します(**図6**)。

すべて入力が完了すると**図7**のような画像が表示されます。

動作確認としてHubotを動かしてみましょう。

```
$ bin/hubot
$ @sdbot ping
$ PONG!
```

HubotとSlackの連携

Hubotに必要な環境変数を追加します。

```
$ export HUBOT_SLACK_TOKEN="{Slackで登録したHubotのAPIトークン}"
$ export PORT={Hubot用に空けたポート番号}
```

Hubotを起動

次のコマンドを入力します。

```
$ bin/hubot --adapter slack
```

起動に成功した場合、追加したHubotがいるチャンネルに@{Hubotのメンション名} pingと入力するとPONGと返してくれます(**図8**)

▼図6　Hubotの生成開始

```
? Owner: {作者のメールアドレス}
? Bot name: {生成したいHubotの名前}
? Description: {生成したいHubotの説明}
? Bot adapter: slack
```

▼図7　Hubotの生成成功!

```
                    ____________________________
_____              /                            \
\    \             |   Self-replication process   |
|    |     _____   |          complete...         |
|__\\|    /_____\   \      Good luck with that.    /
  |//+   |[^_/\_]|   /----------------------------
  |   | _|___@@__|__
 +===+/  ///     \_\
  | |_\ /// HUBOT/\\
  |___/\//       /  \\
        \       /   +---+
         \_____/    |   |
          | //|     +===+
           \//       |xx|
```

Hubotを常駐させる

foreverのインストール

foreverをnpmからインストールします。

```
$ npm install -g forever
```

完了したら、次のコマンドを入力します。

```
$ forever --minUptime 3000 --spinSleep
Time 3000 start -c coffee node_modules/.
bin/hubot --adapter slack --name {Hubot名}
```

再起動する場合は、コマンドforever restartallを使用します。

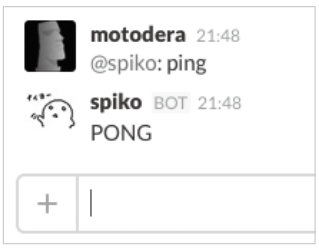
ChatOpsをベースとしたサービス開発基盤

ChatOpsで構築するサービス開発基盤

当社ではモバイルサービスの開発基盤をHubotだけでなく、Slack上でのWebhookによる外部サービス連携も取り入れることで実現しています。これにより開発チーム、運用チーム関係なく互いの業務の状況がSlack上ですべて確認することができます。各チームの領域外のことでも常にSlack上からフィードバックを得られるので、個人のサービスに対する意識を高めつつ、サービスの成長を促進するしくみとして機能しています。その概要を**図9**に示します。

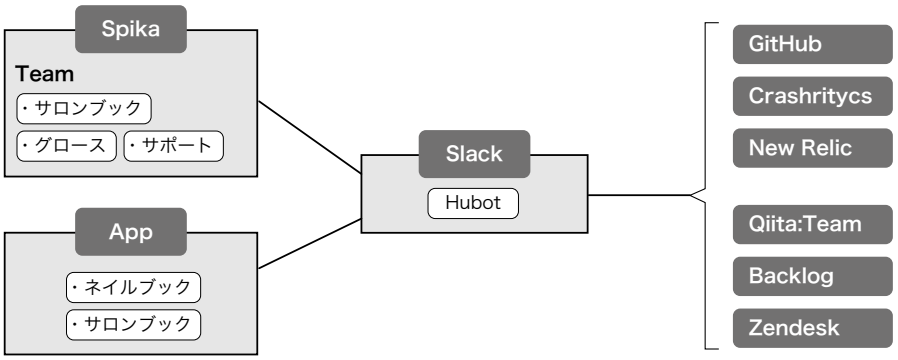
モバイル向けサービスを支えてくれるspiko BOT

spiko BOTは、以前からアプリの配布や定期的なイベントを教えてくれるようなしくみが欲しいという社内でのニーズから生まれました。



◀図8
spiko BOT

▼図9　Slack+Hubotを中心した各種ツールとのサービス開発基盤



当社は女性向けのサービスとして2つのモバイルアプリをリリースしています。モバイルアプリ開発に特化したbotとして皆を支えてくれています。

Slackからサービスを見るしくみ

業務に関連するサービスの監視

Slack上で連携しているbotを次に挙げます。

・Zendesk（https://www.zendesk.co.jp/）
・Qiita:Team（https://teams.qiita.com/）
・Backlog（http://www.backlog.jp/）
・GitHub（https://github.com/）

Zendeskでユーザからの問い合わせをSlack上から確認できるようにしました。連携以前はサポート担当からエスカレーションが上がってきたときぐらいしか、その内容を知る機会がありませんでしたが、今ではリアルタイムにすべての問い合わせが共有されています。これによってエンジニア側のユーザ理解が深まりました。結果として、ユーザ目線を意識したプロダクト開発がエンジニアに浸透しています

Qiita:Teamは、社内のドキュメント管理で利用しており、誰がいつドキュメントに対して操作をしたのかがSlack上から確認できるようになっています。更新状況が見えるようになったことから、連携以前に比較してメンバのQiita:Teamのドキュメント作成量が増えました。見える化は非常に大事です。良いコミュニケーションのきっかけとなっています。

GitHubをSlackと連携させて、アプリ・サーバのリポジトリに対して操作ログをSlack上に流すようにしています。コミットログが共有されるため、チーム・プロジェクトをまたいだメンションが入るようになり、コード品質が向上しました。

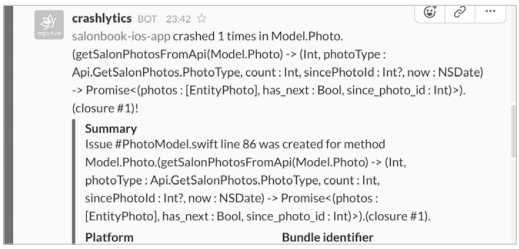
Backlogは、まだSlackと連携できていません。Backlogのチケット登録時・期限切れチケットがあった場合などにアラートをあげるように対応予定しています。

Crashritycsで開発に関するサービスの監視

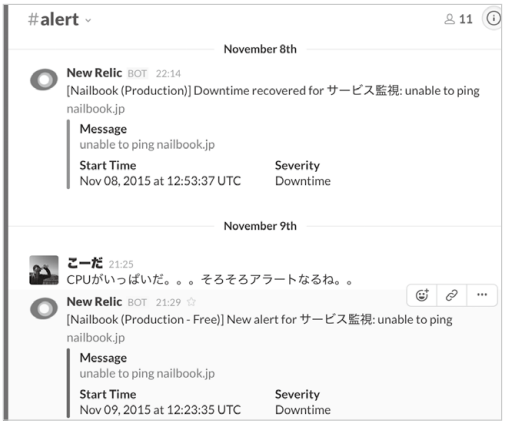
モバイルアプリのクラッシュを検知するためにCrashritycs[注1]を利用しています。アプリをリリースした後の障害検知が早くなりました（**図10**）。導入以前はユーザからの問い合わせ、アプリストアでのレビューなどをきっかけに障害検知することが多かったですが、クラッシュログの発生がSlackに通知されることによりほぼリアルタイムに検知が可能となりました。以前はアプリの開発者だけしかアプリに異常が起きたことを認識できませんでしたが、アプリのどこでクラッシュしているかをSlack上で確認できることによってアプリエンジニア・サーバエンジニア問わず確認ができるようになったため、

注1）URL https://try.crashlytics.com/

▼図10　Crashritycsのbotメッセージ



▼図11　New Relicのbotメッセージ



障害の切り分けがしやすくなりました。

New Relicでサーバ監視

当社ではAWS上で、モバイルサービス基盤となるサーバを構築しています。そのサーバをNew Relic[注2]で監視しています。その監視結果の情報を流す[alert]というチャンネルをSlackに用意して、サーバ開発者だけでなく皆が状況をわかるようにしました(図11)。

以前はサーバ管理者だけしか問題が検知できませんでした。そのため、属人性が高いのが欠点でした。今では公平に確認できるようになったので「見える化」が進みました。インフラ管理者へのアラート通知だけだったときは、対応が遅れることがありましたが、Slackとの連携後は情報共有が進み、対応が早まりました。さらにインフラに関心が薄かった開発者も興味を持ったり、負荷・障害を意識した設計を心がけたりする良いきっかけにもなりました。

ChatOpsによる開発支援

モバイルアプリの配布

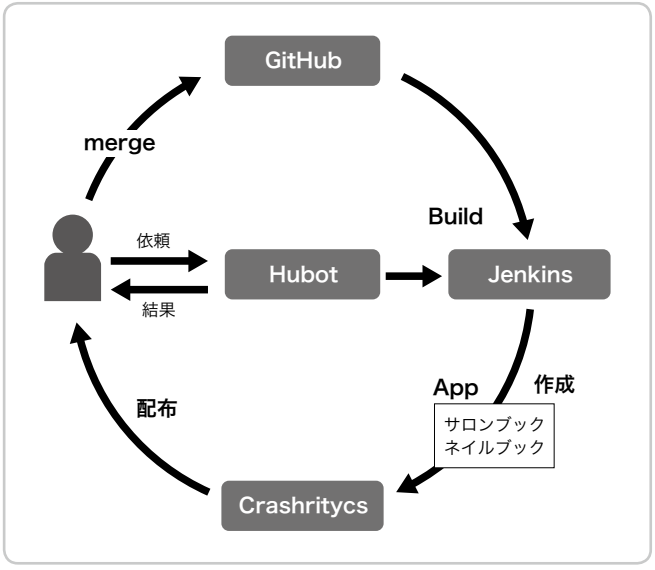
モバイルアプリを配布するしくみをSlack上から行えるようにしています。概要を図12に示します。

配布までの流れ

大きく2つの方法で配布を行っています。1つめは、GitHubで作業分をマージしたタイミングで配布するものです。2つめはSlack上でspiko BOTに依頼する方法です(図13)。次の流れでアプリが配布されます。

①Slack上でHubotに対してアプリのビルド依頼
②HubotがJenkinsに対してアプリ作成を実施するAPIを叩く
③Jenkinsのアプリ作成が完了したタイミングでCrashritycsでアプリを配布

▼図12　モバイルアプリの配布での利用方法



注2）URL http://newrelic.com/

▼図13　ChatOpsによる開発支援

このしくみを導入する前は、アプリエンジニアが毎回ビルドして配布作業をしていました。ディレクターが内容を確認したいタイミングで行えなかったり、アプリエンジニアの工数がとられたりするなどの問題がありました。導入後、アプリエンジニアの工数が削減されました。ディレクターがSlack上から自分でアプリをビルドできるようになったので確認のフィードバックが早くなりました。

ChatOpsでのコミュニケーション改善

朝会＆夕会

毎日の定時にある朝回や夕会の10分前・時間になるとアラートしてくれます(**図14**)。

天気

spiko BOTに天気を聞くと、東京の今日、明日、明後日の天気を教えてくれます(**図15**)。

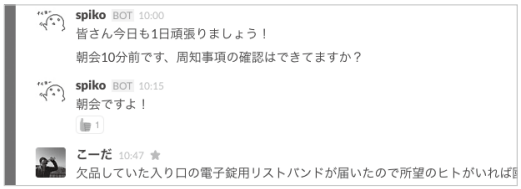
ゴミ出し当番

ゴミ出し当番を毎週持ち回り制でやっていますので、週の頭に当番の人をメンション付きで教えてくれます、これにより社内での活動に一役買ってくれています。

輪読会

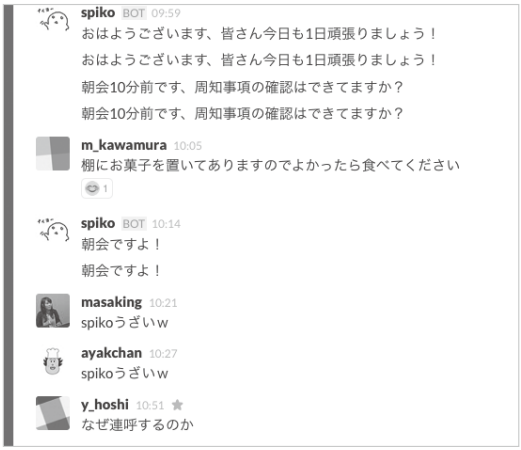
当社では、輪読会を週1回のペースでやっており、読む人の順番を毎回変えているのですが、spiko BOTからランダムでメンションが呼ばれた人が担当になります。これによって順番がまだ来ないメンバにも緊張感が走るので、ドキドキしながら輪読会を待つことができます。

▼図14　アラートするspiko BOT



▼図15　天気を教えてくれるspiko BOT

▼図16　二度言うクセで苦情を言われるspiko BOT



導入初期の失敗

Hubotのcronの定期実行処理が2回走る問題がありました(**図16**)。

◆　◆　◆

botは愛着が湧きやすいので会話のネタになりやすく、やはり導入をお勧めします。SD

第1特集

Slackで、世界を、もっと、はたらきやすく

コミュニケーションの向上をめざして――サーバーワークスの場合

Case 3

かた苦しい挨拶文、検索効率の悪さなどから、最近ではメールによるコミュニケーションが敬遠されることがあるようです。サーバーワークスはそんな課題にいち早く気づき、社内メールを禁止しました。今はチャットツール「Slack」でやりとりをしています。それにより、社内コミュニケーションはどう変わったのでしょうか。

Author　千葉 哲也(ちば てつや)　㈱サーバーワークス

はじめに

著者が所属する、㈱サーバーワークスでは「クラウドで、世界を、もっと、はたらきやすく」というビジョン掲げ、これから起こる「働きかた」の変化に向けて、いくつかの取り組みを進めています。

作らないSI／社内サーバゼロ／会社支給PCゼロ／社内LANゼロ／社内メール禁止／フリーアドレス／リモートワーク

これらの中で、もっとも歴史が古いものが2006年から進めている「社内メール禁止」となります。本章では、著者らがコミュニケーションの多くをメールに頼っていたころ(以下、メールの時代)に抱えていた課題を、Slackを導入したことで、どのように解決してきたかを紹介します。

メールの時代における課題

はじめに、メールの時代に抱えていた課題の代表的なものを説明させていただきます。

メールでは、送信先を選定することが難しい

メールの時代には、同僚が困らないようにとの配慮から、同僚のメールアドレスをCCに追加していました。その結果、1日の始まりに仕事をするための仕事として、メールボックスに大量に届いたメールから「自分が確認するべき重要なもの」と「そうでないもの」を分類するという、生産的とは言えない作業を必要としました。

また、その逆に送信者の手違いでメールが配信されず、必要な情報が届かないといった事故も起きていました。送信者が「気をつける」ことには限界があり、この問題はメールを使う限りは解決できないものだと、とらえていました。

メールでは、誤りを訂正することは難しい

送信したメールの誤りを訂正するためには「先ほどお送りしたメールに、一部誤りがあったため、申しわけありませんが次のとおり訂正させてください」といったメールを再送する必要がありました。これだけでも十分に面倒なのですが、その訂正メールを準備している途中に、訂正前のメールに対するReply(返信)が届いてしまったら目も当てられません。

メールでは、過去の情報を確認することは難しい

1つのテーマについてのコミュニケーションが、1通のメールに対するReplyで完結していれば良いのですが、そうならないことも多くあります。複数のメールをまたいだコミュニケーションを後から確認するには多くの労力を必要とします。

❶文字列や、送信先のドメインを駆使して全文検索する
❷日付順でソートして、時期でのあたりをつける
❸上から順番に1通ずつ読みながら探す

これには多くの労力を必要とするうえに、検

索から漏れているメールがないことを証明することは非常に難しいです。

ほかにもメールにおける課題は多々ありますが、本章でお伝えしたい内容を説明するにはこれで十分ですので控えたいと思います。ここからは、Slackを導入したことでメールの時代に抱えていた課題を、どのように解決したかについて紹介します。

Slackとは

Slack（スラック）とは、Flickrの創設者であるスチュワート・バターフィールド氏によって開発された、チームコミュニケーションツールです。

チャンネルと呼ばれる公開／非公開の部屋や、ダイレクトメッセージを使い、メンバー間でのコミュニケーションを取ることができます。

メールを含めた多くのコミュニケーションツールでも同等のことができますが、Slackの優れているポイントの1つに導入が容易であることが挙げられます。そのほか次の特徴があります。

- メンバー数の制限なく無料で利用し続けられる（ログの保存期間などの制限がある）
- インターフェースが洗練されていてエンジニア以外のメンバーでもマニュアルなどを必要とせずに利用できる
- Mac/Windows/Linux/iOS/Android/Windows Phone向けのアプリが提供されている

Slack導入時に設けたルール

Slackの導入にあたって、始めに“1つのシンプルなルール”を設けました。

“すべてのチャンネルはオープンであり、興味があれば誰でも自由に参加・閲覧を可能とする”

もちろん、機密性の高い情報を扱うプロジェクトなどの例外はありますが、基本スタンスとして、すべての情報（チャンネル）を開き、面倒な手続きなどいっさいなく、望めば誰でもいつでも参加することを可能としました。この狙いは、メールの時代に課題となっていた「受信者の選定」の難しさを解決することです。このルールを設けたことで「念のためCCに追加する」といった、行為が不要になりました。情報を必要とするメンバーは、必要になったタイミングに、その情報が存在するチャンネルへ参加することで、過去からの経緯を含めたすべての情報を利用できます。

たとえば、セールスチームのメンバーが提案資料を作成するために、自社サービスのアカウント増加数を確認したいのであれば、サービス開発チームのチャンネルに参加することで、過去から現在までの増加推移を確認できます。メールの時代のように、サービス開発チームの担当者に「お疲れさまです」で始まり「よろしくお願いします」で終わる、長文の依頼メールを送る必要はありません。

外部サービスとの連携

著者の所属するサービス開発チームでは、コミュニケーションのためのチャンネルとは別に通知専用のチャンネルを作成して、プロジェクトで利用している各種サービスからの通知を集約しています。

バージョン管理サービス／CIサービスとの連携

GitHubのIssue/Pull Requestの更新や、Circle CIのBuild結果を通知することでSlackのチャンネルから、進行中タスクの状態を正確に把握できます。

異常の検知

StatusPage.io[注1]が検知するサービスレベル

注1）Webサービスのステータス（APIの状況、メトリクス）を表示するページを簡単に作成できるサービス。

の異常、Papertrail[注2]が検知するアラート、Errbit[注3]が検知する例外といったアプリケーションレベルの異常、AWS/Herokuといったプラットフォームレベルの異常などをSlackに通知することで、トラブルが起きたことの早期検知を可能としています。また、異なるサービスからの通知が、1つのチャンネルに時系列で集約されるため、原因切り分けにかかる労力を最小化します。

▼図1　UserVoiceの機能リクエストに投票があったことを通知

▼図2　管理機能へのサインインを通知

ユーザとのコミュニケーション

UserVoice[注4]の更新も通知しているので、ユーザからの問い合わせや、機能のリクエスト／投票があったことを、リアルタイムに関係者へ共有することが可能です(**図1**)。

また、のちほど紹介するZapierを利用することで、ユーザのサインアップ／プラン変更といった行動や、管理コンソールへのサインインを通知し、利用の証跡をリアルタイムで把握できるようにしています(**図2**)。

このように、プロジェクトを取り巻くすべての情報を集約することで、プロジェクト関係者が「今起きていること」を正確に把握できるようになりました。

たとえば、長期休暇から戻った日の朝を想像してください。Slackを使っていれば、メールの時代のように雑多に並べられたメールボックスの整理から始める必要はありません。目的別に分類されたSlackのチャンネルへアクセスするだけで、休暇期間中の出来事をすべて把握できます。

ここで紹介したサービスは、すべてSlackへの連携機能を提供しているので、特別なしくみを必要とせずに通知を設定することが可能です。

Zapierを使って、もっと便利に

次に、Zapier[注5]というサービスを使って、Slack連携機能を持たないサービスや、自前のデータソースからSlackへ通知する方法を紹介します。

Zapierとは、各種Webサービスを連携させるハブサービスです。たとえば、「foo@example.com宛てに届くメールをSlackのチャンネルに通知する」といったことや、「DBのテーブルにレコードが追加されたらSlackのチャンネルに通知する」などの連携をノンコーディングで実現します。これにより、非エンジニアのメンバーも積極的にSlackのチャンネルへ情報を公開するようになりました。

社内メールを廃止して、多くの情報がSlackで共有されるようになったことで、コミュニケーションの質が向上したと感じています。

たのしいSlack開発

Slackのすばらしいところは、自作botや、スラッシュコマンドと呼ばれる機能の拡張に簡単にチャレンジできる点にあります。

注2) さまざまなサーバのログを収集し、閲覧・検索できるサービス。
注3) Webサービスを監視し、エラーの検知や通知をしてくれるオープンソースのツール。
注4) 顧客からのフィードバックの受付やヘルプデスクなどのシステムを簡単に構築できるサービス。
注5) URL https://zapier.com/

bot

たとえば、弊社のSlackには、昨年入社の新人@GALACTIC1969[注6]がHubotを使って作成した@buri(ブリ)と呼ばれるbotが住んでいて、「xxx is 何？」と投稿すると、Wikipediaから検索して知ったかブリをしてくれます(**図3**)。

もともとは知ったかブリ機能だった@buriは順調に機能を拡張しており、今では褒めてくれるまでに育ちました(**図4**)。

Hubotの勉強のため、という理由で始まった@buri開発ですが、上記で紹介した"知ったかブリ"／"褒める"のほかにも新機能が続々と追加されています。ちなみに、次に実装を予定している機能は社内Wikiと連携する機能だそうです。彼によると「社内Wikiを調べるまでもないようなことを手際よく調べたいとき、人に聞いたほうが早いという理由で、つい人に聞いてしまうが、そこをHubot(@buri)でカバーしたい。人が覚えなくて良いことをWikiなどから取り出しやすくするインターフェースとしてHubotが適しているのではないかと考えている」とのことです。

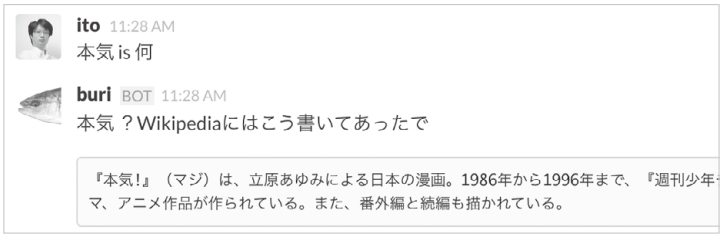
このように、入社して間もない新人が「誰かの役に立ちたい」と思った際に、上司の説得や、関係各所の承認を取るといった煩わしい手続きを必要とせずに、機能拡張にチャレンジできるサービスポリシーもSlackを評価するポイントです。

スラッシュコマンド

次に、スラッシュコマンドについて紹介します。弊社では、事業所移転を機にフリーアドレス制度を導入しました。これまでは各メンバーの自席に設置していたビジネスフォンを撤廃し、各自が所有するBYOD端末(iOS/Android)にSmartPBX[注7]をインストールし、内線電話として利用するようにしました。事業所の移転＋フリーアドレス制度の導入＋内線番号の変更が、同時に起こったことで、一時的に電話の取り次ぎが混乱してしまいました。

この問題を解決したものが、スラッシュコマンドです。Slackに「/info @メンバー」と投稿することで、対象メンバーの内線番号や現在い

▼図3　「本気」とは何か教えてくれる@buri



▼図5　スラッシュコマンドinfoでメンバーの状況を確認

図4▶ほめてくれる@buri

注6）URL https://twitter.com/galactic1969
注7）URL hthttps://www.ntt.com/a_smartpbx/

るエリアといった情報を返してくれます。こちらも@buriと同じく機能拡張を経て、今ではGoogleカレンダーと連携して、今日のスケジュールも返してくれるようになりました(**図5**)。

ノイズとメンバー間の配慮

Slackはチャットですので、「お疲れ様です」から始めて「よろしくお願いします」で締める必要がありません。そのため、自然とラフなコミュニケーションをする機会が増えました。もちろん、コミュニケーションが増えることは良いことだと思うのですが、その一方でメールの時代とは異なる「ノイズのストレス」が生まれました。

まず、はじめにチャンネルから離脱するメンバーが増えました。メールの時代に毎朝行っていた「自分が確認するべき重要なもの」を選別する作業と似ていますが、Slackの場合はチャンネルから離脱することで、以後の情報を完全にシャットアウトすることができます。また、あらためて必要と感じた際は、再度チャンネルへ参加することで離脱期間の情報を取り戻せることから離脱への抵抗が少ないのだと思います。

次に、誰が提唱したわけでもなくメンバーの行動に2つの変化が起きました。興味深い変化でしたので、紹介したいと思います。

①チャンネルメンション(@channel)は使わなくなった

@channelとは、オンライン／オフラインを問わず、そのチャンネルに所属するメンバーにメッセージを通知する機能です。とても便利な機能ですので、Slack導入当初は多くのメンバーが積極的に使っていました。しかし、通知された側のメンバーがメッセージを確認すると、自分とは関係のない内容であることが多く「ノイズ」と感じることがありました。現在は、オンライン中のメンバーだけに通知する@hereや、メンバーを指定して通知するような配慮がされるようになりました(**表1**)。

②目的別にチャンネルを細分化するようになった

業務連絡、チーム、雑談、部活といった目的別にチャンネルが細分化され「xxxの話題であれば#xxxチャンネルに投稿してください」といったように、メンバーが情報の流れを意識して注意するようになりました。その結果、チャンネル数は350まで増えてしまいましたが、メンバーごとに必要なチャンネルを取捨選択できているので、全員にとってノイズの少ない環境になった結果だと考えています(**表2**)。

◆ ◆ ◆

著者らが「社内メール禁止」の取り組みをはじめてから9年間、いろいろなサービスを試してきましたが、今はSlackがベストだと考えています。

メールの是非については本稿での言及を避けますが、メール以外のアプローチのほうが効果的なコミュニケーションとなるケースがあることをご理解いただければ何よりです。SD

▼表1　例：通知への配慮

投稿例	通知先
@channel: xx日にビルの停電があります	全員
@here: 下のコンビニに芸能人いた!!	オンライン中の全員
@foo: コードレビューをお願いします	@foo 1名

▼表2　例：チャンネルの紹介(一部)

チャンネル名	目的	参加を必須とするメンバー
#announce	全社アナウンス	全員
#random	雑談	なし(全員自由に参加できる)
#dept_service_dev	サービス開発チーム	サービス開発チームメンバー
#ext_kokeshi(参加することで部員と認定)	こけし部	なし(全員自由に参加できる)
#prj_xxx	xxxプロジェクト	プロジェクトメンバー
#rss_xxx	各種RSSの通知	なし(全員自由に参加できる)

第1特集

Case 4

ゆるきゃら「ぺこbot」が生まれた理由

ぺこbot爆誕!@Socket

Socketの開発チーム内では、botのキャラクタとしてペコッターというグルメQ&Aサービスの「はらぺこ君(図1)」を採用し、ぺこbotと名付け親しんでいます。本章では、なぜぺこbotが誕生したのか、ぺこbotはどのように作られているのかを解説します。

Author　前當 祐希(まえとう ゆうき)　㈱Socket　Twitter　@maetoo11
Author　大城 敦哉(おおしろ のぶや)　フリーランス　Twitter　@atsuya046

ぺこbot誕生の理由

メンバー全員がアラートに気がつくようにしたい!

「ぺこbot」は、システムの異常に開発チームのメンバーが誰でも気づくことができるようにしたい、という動機から誕生しました。当社が運営する「Flipdesk(フリップデスク)」は、ECサイトなどWeb上で訪問者の状況にあわせて実店舗のような接客体験を提供するサービスを行っています。Webサイトを運営するお客様は、自社サイトにHTMLタグを1行埋め込むだけで、クーポン発行やお知らせ配信、チャット対応などの機能を組み込むことができます。Flipdeskがきちんと動作すること——これがお客様サイトの印象にも密接にかかわってくるため、安定したサービスを提供することが重要です。さらにシステムやDBの監視も必要不可欠です。

しかし、ぺこbot導入以前は、特定のメンバーだけが監視情報のアラートメールを受信していました。もしくは問題を気にかけていたメンバーがコンソールに入って状態を確認するしかありませんでした。

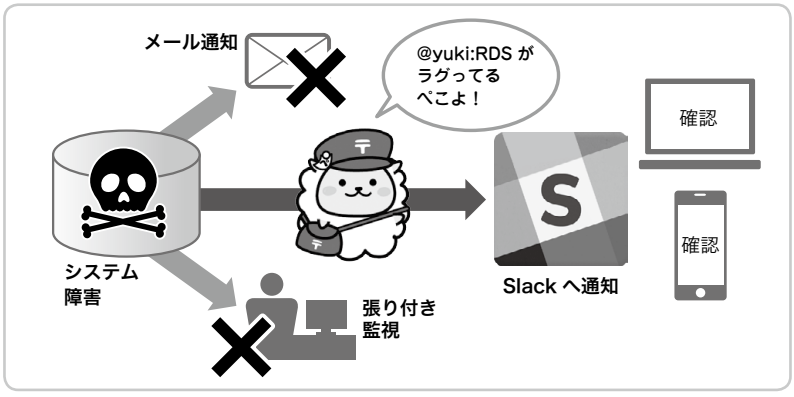
このような「気にかけていないと気づけない。気にかけている人しか気づけない」という状況を解決するため、ぺこbotの導入が始まりました。

Slack＋Hubot＋はらぺこ君＝ぺこbot

ぺこbotは、システムを監視し異常の兆候があればチャットツールのSlackに通知することができます(**図2**)。当社ではSlackを日常的なコミュニケーションの場として使用しています。

▼図1　メッセージを言うはらぺこ君

▼図2　ぺこbotの役割



通知先としてSlackを選択したのは、アラートを通知することで、開発にかかわるメンバー全員がシステムの状況を把握できるようにしたかったからです。

また、botのSlack連携方法についても、インターネット上に情報が多いHubotを使用してシステムを構築しています。これは開発難易度を下げることにもつながりました。

しかし、普通のbotでは単なる通知機能のようで面白みがありません。そこでキャラクタを考え、性格付けをすることにしました。botのマスコットとしては、㈱ブライトテーブルが提供する「ペコッター」というグルメQ&Aサービスのキャラクタで、開発チーム内で人気のあった“はらぺこ君”を採用しました(もちろんデザイナーさんから許可を得ています)。具体的には、botのアイコンをはらぺこ君の画像にし、アラートメッセージでも語尾に必ず「ぺこ」を付けるなど、はらぺこ君らしさをいろいろな部分に盛り込みました。

今では、キャラクタ化されたbotは「ぺこbot」と呼ばれ、メンバーから親しまれています。キャラ付けをしたことで、メンバーがbotにより親しみを感じ、botを触ってみることへのハードルが下がりました。導入の初めのころは、ぺこbotに可愛いことをさせたいという思いから「つらい」という言葉に反応して「そのつらさはいつか人生の糧になるぺこな」というような簡単なメッセージ応答機能を追加して楽しんでいたメンバーも、今ではバッチ監視のような複雑な機能を実装するなど、活発に機能拡張をしています。

「可愛い」という印象を持ったことを通して、導入したメンバーだけではなく、複数のメンバーがさまざまな機能を追加するようになり、ぺこbotはみんなで育てるものになりました。

ぺこbotの導入と効果

当初、ぺこbotは簡単な定型文を投稿する機能しか持っていませんでした。しかし、複数のメンバーが機能追加をすることで、現在はバッチ処理の監視ができるまでに成長しています。

ここではそれぞれの機能がなぜ生まれたのか、どのような効果をもたらしたのかを解説します。

おやつの時間ぺこ!

最初に導入された機能は、毎日15時に「おやつの時間ぺこ」という定型文をSlackに投稿する機能です。開発チーム内では「土日にもおやつの時間を伝えてくる」「ぺこが言ってるし、おやつを食べるか」「糖質制限とかストレッチをすすめたりできないの？」などの声があがりました。

このおやつの時間を通知する機能で、メンバーがぺこbotに興味を持ち始めるようになりました。

システム監視機能の追加

定型文投稿でぺこbotに興味を持ったメンバーにより、DBやバッチの監視機能が追加されていきました。これにより「システムの状況を気にかけていないと異常に気づけない」という問題が解決されました。

FlipdeskはECサイト訪問者の状況に合わせた接客をするために、ボタンクリックやページ閲覧など訪問者の行動ログを収集し、Amazon RDSへ蓄積します。また、蓄積したデータを活用するためにGoogle Cloud PlatformのBigQueryへ転送したり、一定期間ごとのデータを集計したりもしています。

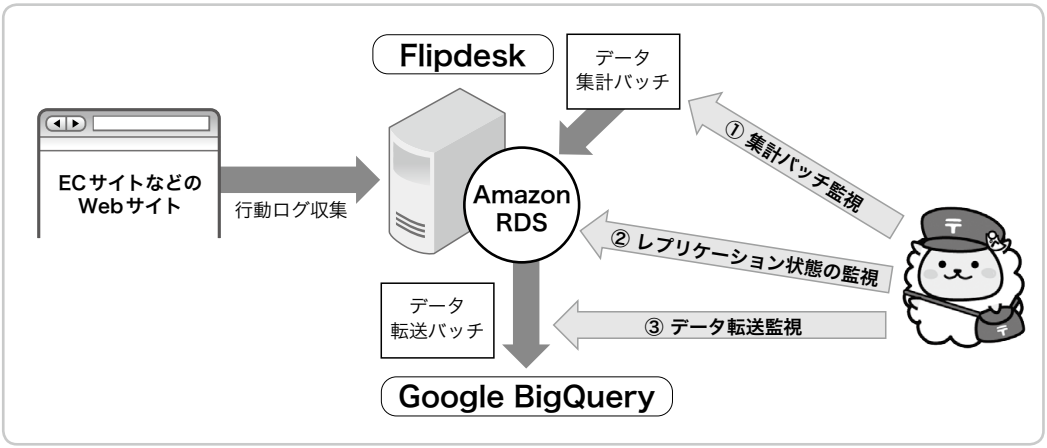
ぺこbotはデータ集計バッチ、DBレプリケーション状態、BigQueryへのデータ転送の3つを監視しています(**図3**)。

異常があった際の通知は、ほかのメッセージに埋もれないよう、Slackに専用チャンネルを作り、そこに出力させています。

データ集計バッチの監視

当社では、Flipdeskの導入がお客様のサイトにどのような影響を与えたか分析するため、Flipdeskの使用が購入に結び付いたかなどの

▼図3　システム監視対象の概念図



情報をログとして取得しています。さらに一定期間に収集されたデータをバッチが集計しています。これらの情報は、Flipdeskを使用した施策の効果を測定するため、データ集計としてとても重要です。

しかし、このデータ集計バッチが、DBの再起動に巻き込まれて停止してしまうことがありました。その件が発生したときは休日だったこともあり、バッチが停止したことにすぐ気づくことができませんでした。これをきっかけに、ぺこbotにバッチを監視させてアラートを出力することにしました。

データ集計バッチの監視には次の3つの機能を実装しています。

①バッチを監視し停止の可能性があった場合にアラートを出す
②バッチが正常に戻ったときもメッセージを出す
③バッチが出力するログをtailする

これらの機能を追加したことにより、バッチ担当者だけではなく、開発チーム全員がバッチの異常に気づけるようになりました。さらに、集計データ量の増加でバッチのパフォーマンスが低下した際、アラートが頻繁に出たことにより事象に気づき、バッチのパフォーマンス改善をするきっかけとなりました。

DBのレプリケーション状態監視

Flipdeskは広告配信プラットフォーム並みの膨大なアクセスを処理するために、Amazon RDSのレプリケーション機能を利用して負荷分散を行っています。もし、短時間でアクセスが増加することでレプリケーションの遅延が発生し、それが継続してしまう場合にはサービスに影響が出る可能性もあります。このような高負荷状態を検知し、いち早く対処できるよう、ここでもぺこbotで監視を行っています。

ぺこbotが定期的にAWS CloudWatchのAPIにアクセスし、レプリケーションの状態を確認します。もし急激なアクセスの増加があり、レプリケーションに遅延が発生した場合には「アクセススパイク発生中」のアラートメッセージをSlackに送信します。また、軽度なレプリケーションの遅延でも、遅延中のサーバ情報と併せてアラートメッセージをSlackに送信します。

このような高負荷状態はすぐに解消される場合も多々あるので、高負荷状態が解消された場合にもその旨のメッセージを送信して、即時の対応が必要なのかどうかを切り分けられるようにしています。

BiqQueryへのデータ転送監視

FlipdeskではAmazon RDSに蓄積した大量の

訪問データなどを、専用のバッチでBigQueryに転送して活用できるようにしています。この転送バッチがまれに停止してしまい、一部の機能に不具合が出てしまうことがありました。しかし、この転送バッチの構成はバックエンドで使用しているRuby on Railsの監視システムと同じものが利用できないものであったため、監視のしくみを別途構築する必要がありました。

そこで、ぺこbotで監視を行うことにしました。ぺこbotに定期的にBigQeuryにアクセスさせ、転送された最新のデータ作成時刻と現在時刻を比較することで、エラーを検知します。そして、もし一定以上の開きがある場合はSlackにアラートメッセージを送るという機能を作り、エンジニアが異常を検知できるようにしました。

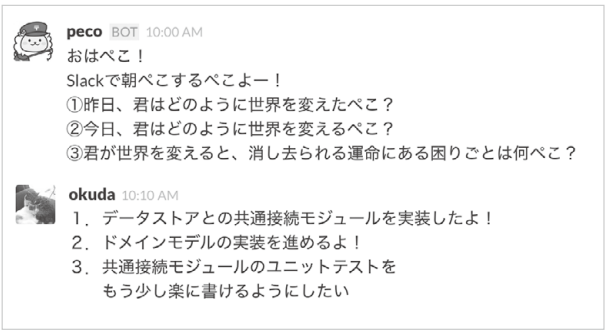
監視のしくみを構築したことで、アラートメッセージが普段より多く送信されてくる場合には警戒姿勢を取ったり、高負荷状態が長く続くような場合は相応のアクションに移ったりできます。このようにしてぺこbotは安定的なサービス稼働にも一役買うようになりました。

開発チーム支援

監視機能が入ったことで、ぺこbotは開発チームに有益な仕事をしてくれる存在として認められました。現在では、開発の支援にもぺこbotが役立っています(**図4**)。

▼図4　ほめられる“ぺこbot”

▼図5　朝ぺこ(朝会)を促す“ぺこbot”



朝ぺこを促す

開発チームにはリモートワークを行っているメンバーもいるため、朝会はSlack上のメッセージのやりとりで行われます。朝会では「昨日やったこと・今日やること・今困っていること」を各メンバーが報告します。毎朝10時になるとぺこbotが**図5**のようなメッセージを出力します(社内では朝会を「朝ぺこ」と呼んでいます)。

これによりお互いの状況がわかるようになり、困っている人により気づきやすくなりました。また、メンバーがどんな作業をしているかが事前にわかるため、メンバー間で話をするときに内容を理解しやすくなるというメリットもありました。

カバレッジ計測

現在、Flipdeskのバックエンドは大部分がRubyで作られていますが、大量のアクセスを効率よく処理できるよう一部をScalaに移行するプロジェクトが進行中です。併せて長期的に運用できるよう再設計することで、メンテナンス性の向上も狙っています。その一環として、ユニットテストを積極的に行っていく方針を立てており、CI(継続的インテグレーション)でテストカバレッジの計測とレポートの生成を行っています。

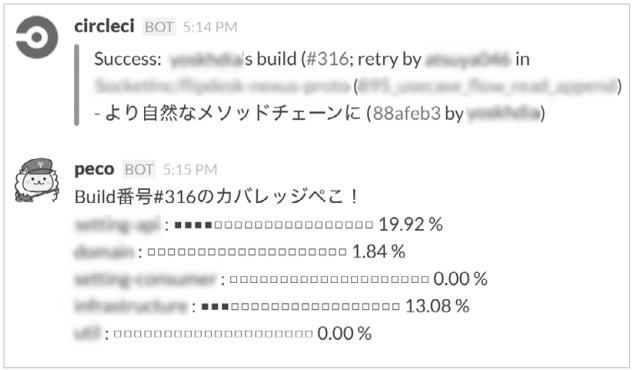
ただ、CIの機能だけでは、能動的にカバレッジレポートを見に行かなければならず、気軽にカバレッジを把握できません。この煩わしさの

ため、そのうち誰も見なくなってしまうのではないかという懸念がありました。

この煩わしさを解消し積極的にテストを書く習慣のサポートができるよう、ぺこbotに活躍してもらうことにしました。

カバレッジレポートはCircleCI[注1]のBuild artifactsという機能を使い、CircleCI API経由で取得できるようにしています。そこにぺこbotが定期的にアクセスし、最新のBuildで生成されたレポートを取得します。このレポートを解析してレポートのサマリーを作り、それをアスキーアートで表現したメッセージに変換してSlackに送信します(**図6**)。

▼図6　カバレッジレポートを行う“ぺこbot”



この機能を導入したことで、メンバーがバラバラにカバレッジレポートを確認しに行かずとも、Slack上で確認できるようになりました。これをきっかけに、テストの薄いところを改善するだけでなく、カバレッジが日に日に厚くなっていく様子が楽しめるなど、積極的にテストを行う動機付けになるのではと期待をしています。

まとめ

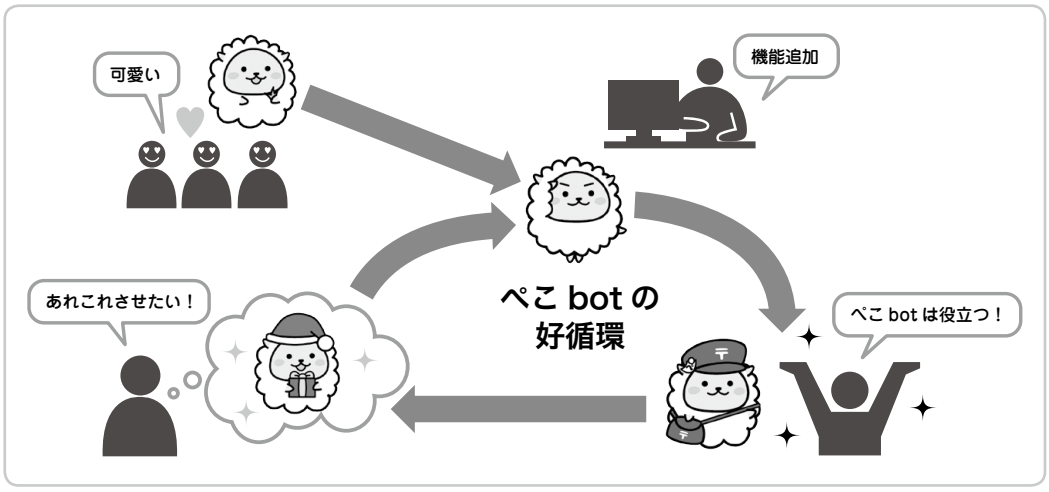
メッセージを定時に出力するという簡単なことから導入し、監視やプロジェクト支援までできるほどにぺこbotは成長しました。

当社ではbotに愛着のあるキャラクタを採用することで、誰かが機能を実装して終わりにするのではなく、

作る→便利さが受け入れられる→また誰かが機能を追加する(可愛い！)

という好循環が生まれています。単にbotを導入するのではなく、このような好循環(**図7**)を創りだしていくこともChatOpsがチーム内に浸透していく秘訣なのではないでしょうか。SD

▼図7　これからもメンバー全員でぺこbotを育てていくぺこ!



注1）URL https://circleci.com/

第1特集

組織にChatOpsを根付かせるために

GaiaxのChatOps実現までの軌跡

Case 5

ChatOpsの導入は、組織のコミュニケーションを大きく変える改革です。メンバーの意識を変え、ChatOps文化を根付かせるには何が必要なのか。Gaiaxの事例を参考に考えてみましょう。

Author 石川 雄基(いしかわ ゆうき)　㈱ガイアックス　Twitter @hoto17296
Author 佐藤 有花(さとう ゆか)　㈱ガイアックス　Twitter @cradle_of_nox
Author 坪井 優朋(つぼい ゆうほ)　アディッシュ㈱　Twitter @ufo_ocha
Author 福本 貴之(ふくもと たかゆき)　㈱ガイアックス　Twitter @__papix__
Author 肥後 彰秀(ひご あきひで)　㈱ガイアックス　Twitter @hidehigo
Author 菊池 正宏(きくち まさひろ)　㈱ガイアックス　Twitter @sohismyson

ChatOps文化の良さと実現方法

Gaiaxではここ1年でChatOps文化が一気に浸透しました。現在では毎日のようにチャットでさまざまなことが行われています。

これだけのChatOpsを実現するまでに、何を考え、いかにして実現させてきたか。また、現在どのようなChatOpsが実現されているかなどを紹介します。

ChatOpsの良さ

そもそもChatOpsとは何で、どんなメリットがあるのでしょうか？　実現方法や事例を紹介する前に、ChatOpsそのものについて述べたいと思います。

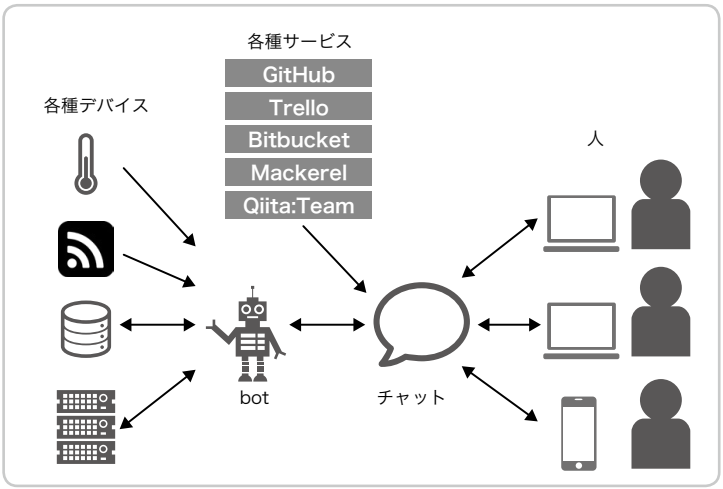
あらゆる情報のハブとしてのチャットツール

ChatOpsは、チャットというツールの今までの在り方を変える概念です。チャットは本来、「人対人」のコミュニケーションツールですが、ChatOpsはこれを「人対何か」に拡張するものです。ここでいう「何か」とは、たとえばbotであったり、チームで利用しているサービス(GitHub、Trello、Bitbucket、Mackerel、Qiita:Teamなど)であったり、さまざまなデバイス(GPS、温度計、家電など)であったりします(**図1**)。

「人」に限らずあらゆるリソースにチャットを通してアクセスできるようになると、あらゆる情報がチャットツールに集約されていきます。情報が集約されると、「チャットを見ればそのチームの状況がすべてわかる」ようになります。「Trelloを更新する」「日報を書く」などのタスクはチーム全体で習慣化して毎日行うことが重要ですが、利用するツールが多くなればなるほど手間が増え、次第に習慣が薄れ、文化として定着せずに使われなくなってしまいます。

ChatOpsによって「チャットさえ見ておけばいい」という状況を作ることで、このような「意識

▼図1　情報のハブとしてのチャット



すべき習慣」を限りなく減らすことができ、新しい文化を取り入れる障壁を下げることができるのです。

ChatOpsによるコミュニケーションの活性化

Gaiaxの社内bot「Gaiachan」にはさまざまな機能がありますが、中でも最も多く使われているのが「ポイント」機能です(**図2**)。これはHip Chat Bot LabのKarmaと同じ機能をGaiaxのbotに実装したもので、チャットで@name ++と書くことでその人にポイントを与えることができるというシンプルな機能です。なんてことはない機能に思えるかもしれませんが、これは気軽に感謝や称賛を表現できるようになるとても強力な機能です。

常日頃の些細な感謝や称賛は、「どう伝えるか」よりも「まず伝えること」が重要です。

▼図2　ポイント機能

「ありがとうございます」なのか「サンキュー！」なのかを考えるよりも、まずどんな手段でもいいから表現することが大切なのです。ポイント機能は非常にハイコンテクストなコミュニケーションで、「その人が何か良いことをしたらしい」ということしか表現できません。しかし、それゆえにどんな言葉よりも気軽に表現することができます。若者が「ヤバい」を汎用するように、感謝・称賛も「ポイント機能」という汎用的な手段でできる限り多く流通させたほうが、チーム全体のコミュニケーションは活性化するのです。

ChatOpsによるコミュニケーション活性化の事例をもう1つ紹介します。Gaiaxには「書庫たん」という、会社で買った書籍の管理をする社内サービスがあり、「誰がどの本を借りた・返却した」「新しい本の購入申請をした」という情報がチャットに流れるようになっています。それを見たほかのメンバーから「おもしろそうだから今度読もう」「それ調べたいならこっちの本もオススメ」というような会話が生まれることも多々あり、新しい本に出会うきっかけになっています(**図3**)。

活発なコミュニケーションを生むために必要なのは、会話の種をたくさん蒔くことです。「本を借りる」という取るに足らない行動も、その情報が共有されることによって新たなコミュニ

▼図3　書庫たんの事例

ケーションが生まれる種になり得るのです。

ChatOps文化を根付かせるために

前項で「ChatOps文化の何が良いのか」について述べました。次に「いかにしてChatOps文化を定着させるか」を考えます。

せっかくbotを作ったのに、なかなか活用してもらえない。気づけば、忘れられている。そういった経験はないでしょうか。前述のような効果を期待するのであれば、組織でbotが活用されるしくみを作り、環境を整備し、ChatOpsを文化として根付かせることが必要です。

この項では、GaiaxでChatOps文化を根付かせるために、これまでどのようなしくみや環境を作ってきたのかを紹介します。

botの機能——愛されるbotを作る

使える頻度が高く、気軽に使える機能を作る

botには、使う頻度が多い機能、気軽に使える機能を実装すると、botの存在をアピールできます。

Gaiachanにあるポイント機能は、人から人へ気軽に「いいね！」を伝えられる手段になっています。良い情報をシェアしてくれたら、機能追加に対応してくれたら、渾身のギャグが決まったらなど、汎用性が高いので、使えるチャンスが自然と多くなります。何度も使われれば、botの存在が目にとどまる機会も増えます。

逆に、botの機能が、使いどきのわかりにくいもの、使うのに抵抗感があるものばかりだと、botの存在は忘れられてしまいます。まずは汎用的に使える機能を用意し、日常会話の中で自然とbotが使われるようになると良いでしょう。

▼図4　順番決め機能

▼図5　発言の内容に合わせたアイコン

コミュニケーションを後押しする機能を作る

botからメンションを送らせて特定の人を巻き込んでいくことで、新しいコミュニケーションを後押しすることができます。

Gaiachanには順番決め機能があります。勉強会の発表順など、複数人の順番をランダムに決めてほしいときによく使われています(**図4**)。普段はbotを使わない人も、ほかの人がbotを通じてメンションを送ることで、botに興味を持ってもらえます。

愛着の湧くbotにする

botはプログラムですので、実装したとおりにしか動きません。しかし、人間味のある口調や反応を用意すれば、挨拶を返してくれるだけの機能でも、不思議と愛着が生まれるものです。

Gaiachanには、発言とともにアイコンが表示されるようにしています。発言の内容に合わせて異なる表情のアイコンを表示することで、よりイメージが湧きやすくなりました(**図5**)。

困ったときのヘルプを作る

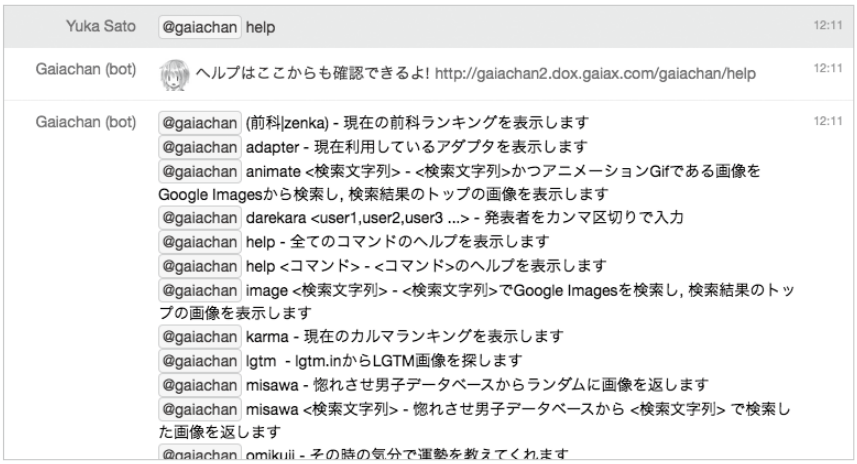
botのインターフェースはコマンドに近く、呼び出されない限りは機能の全貌が見えません。ヘルプ機能を実装しておくと、使い方をすぐ確認できますし、知らない機能が増えていても気づくことができます(**図6**)。

環境の整備——botのいる日常を作る

チャットツールを組織的に導入する

まずは前提として、botを利用できるチャットツールを日常的に利用しており、チャットがコミュ

▼図6　ヘルプ機能



ニケーションの中心となっていることが必要です。

Gaiaxでは以前、Skypeを全社的に利用していましたが、現在ではエンジニア全員がHipChatに移行しています。ChatOpsが文化として根付くには「皆がチャットを見ている」ということが重要です。一部の人はチャットツールを使い、一部の人はほかのツールを使っているという状況では、botをきっかけとしたコミュニケーションが発生しにくくなります。

チャットツールの導入については、後述の「ChatOps実現における障壁と、どう折り合いをつけてきたか」にて詳しく紹介しています。

気軽にbotを試せる場所を作る

botを導入しても、始めは業務連絡以外のコミュニケーションは取りにくいかもしれません。botの発言で会話が流れてしまうため、botの機能を呼び出すこと、新しい機能を試すことがためらわれる場合があります。botは呼び出されないと何が起こるか予測しづらい、というのもその一因となります。

まずは、気軽にbotを使える場所を用意しましょう。Gaiaxでは、HipChat上で若手部屋・Bot部屋など部署横断のグループを自由に作成しています。これらのグループが、botを試せる場、さらには新たな機能が提案される場になっています。

ChatOpsを活用している人たちを増やす

Gaiaxでは、botを広めたいメンバーが積極的に活用していくことで、ChatOpsを使う文化が定着していきました。

多くの人が見る場でbotを使えば、それを見た人に使い方や使う状況が伝わります。興味を持ったほかの人がさらに別の場でbotを使い、またほかの人が……と続いていくことで、ChatOps文化は広まっていくのです。

また、botを使う人が増えていけば、botに対する改善案や新しいアイデアも生まれていきます。Gaiachanでは、作成当初からPull Requestを使った開発が活発に行われており、社内の多くの人が機能追加をしています。自分で実装した機能が追加されれば、さらにbotへの愛着が増します。そうして、botを活用するメンバーが増えていったことも、ChatOps文化定着の大きな要因でしょう。

botのいる日常を

ここまでChatOps文化を根付かせるための、さまざまなGaiaxの取り組みを紹介しました。

しかし何よりも大事なのは、botを皆で楽しむことだと思います。業務に必要な機能だけと言わず、皆が楽しめる機能を自由に追加して、皆で楽しむことによって、日常にbotが馴染ん

でいきます。

botは手を入れれば入れるほど、自分たちに最適な機能を備えるようになり、使えば使うほど、botへの愛着が湧くようになります。いつしか、botはコミュニケーションの一部を担う立派なチームメンバーに育っているはずです。みなさんも自分たちのチームメンバーとなるようなbotを、ぜひ育ててみてください。

そして「Chatで Operation」へ

前節で、私たちはChatOpsを「チャットツールにあらゆる情報を集めるための道具」としてとらえて推進し、Gaiaxのエンジニア文化の1つとして根付かせるための工夫について解説してきました。このように、botが私たちの日常の一部として定着してくると、次は"本来の"ChatOps、すなわちチャットツール上でさまざまなオペレーションをしたくなってくるのは当然のことではないでしょうか。

このようなChatOpsについては、すでにさまざまな会社が実現していて、その知見もたくさん共有されています。しかし、ChatOpsはアイデアやコードをまるごとコピーしてすぐさま便利に活用できるものではありません。なぜなら、ChatOpsはチームの状況や、プロダクトに適したオペレーションに対応できなければ真価を感じることができないからです。そのため、ChatOpsの実現と活用は、単にbotを会社やチームの文化として定着させるより、一段難しいのではないでしょうか。

この節では、Gaiaxが2015年にリリースした「Reactio」[注1]という障害対応支援サービスの開発現場にChatOpsを採用した事例を紹介します。そしてその中で体感した、初めてChatOpsに取り組む際に考慮すべきポイントについて述べたいと思います。

ReactioにおけるChatOps

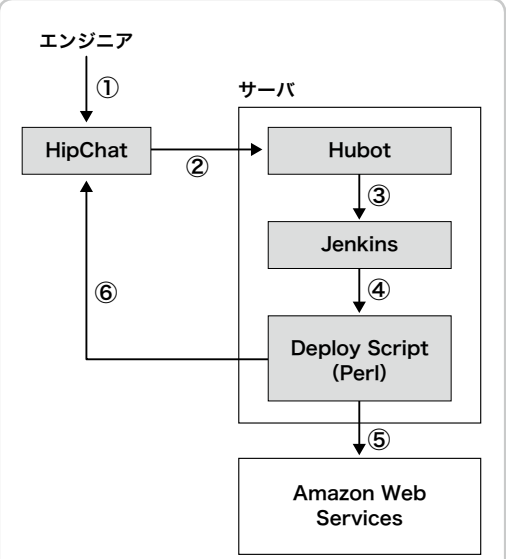
ChatOpsの良さとして、前節では「情報の集約」や「コミュニケーションの活性化」を挙げました。Reactioチームでは、これに加えて「日々繰り返し行われるさまざまなオペレーションのbotによる自動化」を利点と考え、ChatOpsの採用に踏み切りました。とはいえ、これらはあくまで結果にすぎません。大切なのは、ChatOpsによってエンジニアが働きやすい環境をエンジニア自身で作っていけるという点だと思います。

そのため、ChatOpsに挑戦するにあたって、Reactioチームは次の2つのポイントを重視して取り組みました。

・オペレーションを小さい部品に分けて実装していく
・使い慣れた道具を使って実装する

ここからは、ReactioチームにChatOpsが定着するまでの流れを紹介しながら、この2つのポイントについて詳しく見ていきます。

▼図7　Reactioにおけるデプロイ処理の流れ



注1）URL https://reactio.jp

ReactioにおけるChatOps

Reactioチームでは、いくつかのオペレーションをbotに行わせていますが、今回はその中でもデプロイの処理を見ていくこととします。

図7は、Reactioというサービスにおいて、botがデプロイのオペレーションをする際の処理の流れです。まず、エンジニアがHipChat上でbotアカウントに対して、メンションを使ってデプロイオペレーションを呼び出します(①)。すると、そのメンションにHubotが反応し(②)、Jenkinsのデプロイジョブを呼び出します(③)。Jenkinsは、Perlで書かれたデプロイスクリプトを実行し(④)、デプロイスクリプトが実際のデプロイ処理を行います(⑤)。デプロイの途中経過や最終結果は、HipChatの通知APIを利用して、HipChat上に通知されます(⑥)。このしくみを構築することで、Reactioチームではデプロイ作業をHipChat上だけで完結できるようになり、デプロイ作業の負担を大幅に軽減することができました。

この流れは、あくまで最終的にこのような形に落ち着いただけであり、最初からこのような形を目指して進めていったわけではありません。先ほど述べた2つのポイントを実践しているうちに、このような形に落ち着いただけです。

オペレーションを小さい部品に分けて実装していく

オペレーションは、チームやプロダクトの状況に応じて日々変化するものです。そのため、いきなり最終的なゴールを想像し、1つの大きな部品として実装してしまうと、変化に追随することが難しくなってしまいます。まずはチームやプロダクトに必要なオペレーションを小さな部品に切り分け、1つずつ実装し、それを組み合わせることでChatOpsを実現していくのが良いでしょう。

Reactioチームの場合、デプロイのChatOpsを実現するにあたって、まず初めに実装したのは「デプロイスクリプト」だけでした。次にHubotを導入し、エンジニアがHipChatとHubotを通して、デプロイスクリプトを実行できるように環境を加えました。最終的に、Reactioのテストのために Jenkinsを使うようになったので、デプロイスクリプトをJenkinsが呼び出し、HubotはJenkinsのジョブを実行するように置き換えました。

このように、オペレーションを小さな部品に分けながら実装を進めると、ChatOpsを推進している中で、チームやプロダクトの状況が変化したことに起因する路線変更が非常にやりやすくなります。先ほどの例では、最初にデプロイの実行を「デプロイスクリプト」という部品として切り出したおかげで、デプロイ処理の呼び出し方の変更が、非常に容易になりました。もし最初からHubotのスクリプトにデプロイ処理をべた書きしていたら、Jenkinsを導入する際に、「デプロイ処理を呼び出す」部分と、「デプロイを実行する」部分を分離しなければならなかったでしょう。

使い慣れた道具を使って実装する

ChatOpsを実装する手段として「チームメンバーが使い慣れた道具を使うこと」は非常に重要です。なぜなら前述のとおり、ChatOpsは「エンジニアが働きやすい環境を、エンジニア自身で作っていける」ことが重要だからです。これを実現するためには、なるべく多くのチームメンバーがChatOpsに関するコードを読み書きできる状態にしなければなりません。

Hubotを利用する場合、オペレーションをJavaScriptで実装することが一般的ですが、Reactioチームの場合はPerlに慣れているエンジニアが多かったので、オペレーションを実行するためのスクリプトはすべてPerlで実装しました。そしてHubotの役割を、オペレーションのためのスクリプトやJenkinsのジョブを呼び出すことだけに留めました。

使い慣れた道具を使うことで、多くのメンバーをChatOpsに巻き込めるだけでなく、これまでに得ている情報を活かして、高速にChatOps

の実装を進めることが可能になります。

ChatOpsの実現を目指して

この2つのポイントを意識してChatOpsの実現を進めると、ChatOpsはいつでも方向性を変えられるようになります。これはChatOpsを実現するうえで、非常に重要な要素です。なぜなら、ChatOpsはチームやプロダクトの変化に“必ず”、“正確”に追随しなければならないからです。もしChatOpsが追随できなくなってしまったら、ChatOpsは使われなくなる……というよりは、使えなくなってしまうからです。

まだChatOpsを試したことのない方は、ぜひ前述の2つのポイントを意識しながらチャレンジしてみてはいかがでしょうか。そしてChatOpsを活用して、日々の仕事をより良いものにしていきましょう！

ChatOps実現における障壁と、どう折り合いをつけてきたか

この節では、ツール選定の意思決定者の立場でChatOpsの初期の話や経緯について触れてみようと思います。読者のみなさんが自身の環境でChatOpsを実現する際の戦略として活かしていただければうれしいです。

各社の組織構成や歴史によっては、必ずしもChatOpsが可能なコミュニケーションツールや基盤が整っていないこともあるのではないでしょうか。Gaiaxにおいても、ChatOps実現の最大の障壁は、ChatOpsが可能なチャットツールを導入することでした。

新ツール導入までの経緯

Gaiaxでは長らくSkypeを業務におけるコミュニケーションツールとして利用しています。エンジニア以外の事業メンバーも多いGaiaxでは、新ツールであるHipChatを導入するまでには、何度も提案がありましたし、議論が盛り上がったこともあります。実際、IRCを導入し、普及を試みたことも2度はあったと記憶しています。どちらも半年～1年ほどかけて取り組みましたが、キャズム[注2]を超えることはなく次第に利用されなくなっていきました。この失敗の原因は、botと連携しやすいツールという点でのPros[注3]はありつつも、コミュニケーションツールとしてのUXの乏しさがあり、既存のコミュニケーションツールを代替しきれなかったこと。それに加えて、botや通知を中心とした利用として並立(両用)を狙って進めたこと。この2つだと考えています。

これをふまえて、今回の新ツール導入にあたっては、次の2点を重視して進めました。

❶自身だけでなく、組織が動く理由を考え、しっかりとした議論をする

今回、新ツール導入を推進する理由として、「Gaiaxの事業環境、働く環境を考えたときに、招待型の非公開チャット(要はSkype)ではなく、公開部屋(いわゆるルームやチャンネル)がリストされており選んで参加する形へ、コミュニケーションのスタイルの変更が必要」という点を掲げ、検討・導入を進めました。ここで強調したいのはこの理由の是非ではなく、導入の提案者として直接の動機やPros(たとえば「bot作りやすい、bot作るとこんなに良い」)を提案するだけではなく、一歩俯瞰(ふかん)して組織を動かすに値する理由を見つけることも重要だということです。もし、読者のみなさんに、いくら提案しても通らない、とお感じの方がいらっしゃればぜひ参考にしてみてください。

❷エンジニアでのスタンダード

前述したように、エンジニア以外のメンバーも多いGaiaxにおいては、全社でのスタンダー

注2) 新しい商品やサービスが市場でブレイクする際に超えなければいけない一線(深い溝)のこと。
注3) 良い点のこと。逆に悪い点はCons。

ドはまだSkype。インターフェースの面、未読管理や通知の面、正直有用過ぎるとさえ感じます。こういうツールをひっくり返すのはとてもたいへん。エンジニアとしてのProsを足さないと、Pros/Consが拮抗しないので、全社で利用するコミュニケーションツールを一気に置き換えることは難しいと考えました。

一方で、歴史をふまえて、並立(両用)ではなくコミュニケーション基盤を移すことが重要、と考えていたので、初期の導入範囲を絞り、エンジニアでの標準ツールとして導入することにしました。「エンジニア同士のコミュニケーションの場は基本的に新ツールに移行し、各事業でのコミュニケーションは、事業ごとに任せる。そして多くの事業は既存のツールを利用する。その境界にまたがるメンバーは、新旧ツールの両使い」、この方針について、全チームで持ち帰り議論してもらい決定しました。結果としては、全エンジニアチームが既存ツールとの両使いの不便も承知のうえで、新ツールへの移行に合意してもらえました。

まとめ

思惑どおり、まずはエンジニアのコミュニケーションの標準ツールとして定着しました。実際、ツールの並存のオーバーヘッドと新ツールの定着とのせめぎ合いがあったと思っています。ともすれば過去のIRCのように新ツールが定着しない可能性もありました。が、今回は、本章のほかのパートで述べたような、連携性、業務の自動化を含めて、遊び心を刺激するbot(プログラマブル)という要素、メンバーの主体的な活動があってこそ定着したと考えています。

2つめの変化として、エンジニアが人数的にも事業的にも中心の事業は、事業全体のコミュニケーションをHipChatに移行するケースも増えています。自動化が外せない要素になっている、その理解が広がっている、と考えています。

全社に広める、どこかのタイミングで全体を置き換えきる(さらにはSlackへ)、というのは今後の課題ですが、Gaiaxのケースとして、範囲は限定しながらも、しっかりとした議論を行い、ChatOpsを実現していく、というところがみなさんの参考になれば幸いです。エンジニアの感覚で組織を塗り替えていく。まだ道のりは長いですが、引き続き模索とトライを進めていきます。SD

column

HipChatとは?——Gaiaxにおけるチャットツールの選定

HipChat(ヒップチャット)は、BitbucketやJIRAなどでも知られるAtlassian社製のチャットツールです。ビデオ通話、スクリーン共有、APIやWebhookでのインテグレーションなどが可能です。Mac、Windows、iOS、Android、Linuxに対応しており、さらにはブラウザからでも利用することができるので、あらゆる場所からチームとの連携がとれるチャットツールになっています。

GaiaxではSkypeを全社で利用していましたが、さらに開発を加速させるため、ChatOpsを前提としたチャットツールの選定を行いました。選定の基準として、「APIやWebhookが充実していてエンジニアフレンドリーであること」や、情報共有の重要性から、「メンション機能」や「ルーム(公開部屋)がオープンになっていて興味のあるルームに能動的に参加できること」など挙げられました。

Skypeからの移行時にSlackやChatworkなども同時に検討しましたが、選定基準を満たすことのほか、選定当時の各事業や開発チームの状況、費用対効果も踏まえエンジニアのチャットツールはHipChatを利用することに決定しました。利用開始して6ヵ月が過ぎましたが、HipChatの利用は定着したと考えていますし、導入により多くの課題が改善されました。今後もGaiaxでは、その時々で必要なツールやしくみを柔軟に取り入れつつ、開発を加速させていきたいと考えています。

第1特集

エンジニアのための より良い環境づくり

はてなにおけるChatOpsのこれまでとこれから

はてなでは、Slackをコミュニケーションのハブとして、日々の開発や運用を進めています。現在のこのスタイルになるまでには、いくつかの変遷がありました。本章では、IRCを中心にしたスタイルから、今現在のSlackを中心とした使い方までの流れについて、さまざまな試行錯誤とともに紹介します。

Author　田中 慎司(たなか しんじ)　㈱はてな
Twitter　@stanaka

Case 6

はてなのChatOps初期

IRC時代

弊社では2000年半ばより、チャットツールとしてIRCを活用してきました。IRCはインターネットプロトコルの一種で、RFC2810からRFC2813[注1]までで定義されています。IRCのサーバ実装はいくつかありますが、弊社では「ircd」[注2]を利用していました。ircdはシンプルな実装ですので、メッセージ保存のような機能は「Tiarra」[注3]を利用している人が多くいました。

2014年夏にSlackを導入することになるのですが、それまでの10年ほどはIRCとさまざまなツール／サービスを連携させつつ、IRCによるコミュニケーションを軸に業務を進めてきました。最初は人同士の会話だけでしたが、次第にさまざまなbotを立ち上げるようになり、徐々にツール連携を増やしていきました。

IRCでのツール連携

たとえば、Capistranoのデプロイを実行すると**リスト1**のようなメッセージを流していました。これは、serviceAやserviceBといったアプリケーションをデプロイしていた様子です。IRCと各ツールの連携には「ikachan」[注4]を利用していました。ikachanを経由することで、HTTPのREST APIで簡単にIRCにメッセージを投げることができました。

そのほかの例として、Jenkinsを連携させて、CIテスト結果を投稿させたり、ジョブキューに溜まっているジョブ数を投稿させたり、サービス上のユーザアクティビティ(投稿など)を流したりしていました(**リスト2**)。

もちろんインフラ監視も連携していました。たとえば、Nagiosのアラートを通知させたり、monitによるプロセスメモリ監視の結果を通知

▼リスト1　「Capistranoとの連携」IRCに流れるデプロイのメッセージ

```
19:43:23 (#cap@hatena:ikachan) serviceA: stanaka @production update (on 4f3b1e2)
19:44:58 (#cap@hatena:ikachan) serviceB: stanaka @production update (on 7877c28)
```

▼リスト2　「Jenkinsとの連携」IRCに流れるテスト結果など

```
00:00:47 (#main@hatena:hudson) Project Foo build #379: STILL FAILING in 6.8 sec: http://jenkins/job/Foo/379/
00:00:54 (#main@hatena:hudson) Project Bar build #369: STILL FAILING in 14 sec: http://jenkins/job/Bar/369/
00:42:01 (#service@hatena:ikachan) job count: SomeJob has 14700 jobs. Landmine jobs are inserted at
(2013-03-10 04:30:00).
00:05:23 (#service@hatena:ikachan) [Q] : foobar: http://example.com/foobar
```

注1）URL https://tools.ietf.org/html/rfc2810(2811,2812,2813)
注2）URL http://www.irc.org
注3）URL http://www.clovery.jp/tiarra
注4）URL https://github.com/yappo/p5-App-Ikachan

▼リスト3 「Nagios、monitとの連携」IRCに流れるアラートなど

```
20:35:03 (#main@hatena:Nagios) CRITICAL somedb18/mysql_process_my Duration:0d 0h 2m 19s -> http://
server/host/10.0.xx.yy (processcount:  524)
21:04:19 (#main@hatena:Nagios) OK somedb18/mysql_process_my Duration:0d 0h 0m 3s -> http://server/
host/10.0.xx.yy (processcount:  15)
00:49:17 (#monit@hatena:ikachan) someapp:/ apache [trigger] memory out of bounds [3405280kb, 3425872kb,
*3730976kb, *3706148kb, *3866624kb] (GroupMemoryUsage)
00:49:24 (#monit@hatena:ikachan) someapp:/ apache [trigger] process is running (ProcessRunning)
08:00:01 (#statuscode@hatena:ikachan) someapp37 app-edit 07:00:01-08:00:01 2xx:4768(91.66%)
3xx:356(6.84%) 4xx:76(1.46%) 5xx:2(0.04%)
```

させたり、Webサーバのアクセスログからステータスコード種別ごとのアクセス数統計を投稿したりしていました(**リスト3**)。

Slackへの移行

IRCの限界

IRCはシンプルで拡張性も高いツールなのですが、次のような課題がありました。

- 非エンジニアにとって導入のハードルが高い
- テキスト情報のみでメッセージの表現力が低い
- 画像の共有方法がクライアントツール依存
- 過去ログを保存する標準のしくみがない
- GitHubのようなSaaSからメッセージを投稿するのが面倒

Slackの採用

これらの課題を解決するより良いツールがほかにないか探していた中、見つけたのがSlackでした。弊社ではSlackを、2013年末のベータ期間中から試用開始していました。当初は実験的に利用するのみで、引き続きIRCをメインで利用していました。半年ほど試用した結果、本格的に社内のコミュニケーションツールとしてIRCをリプレイスすることを決定し、2014年夏には有料プランへ移行、本格的な利用を開始しました。

現在では、ほぼ完全にIRCからの移行を完了し、さまざまなサービスやツールと連携しています。IRCを利用していたころと比べると、社内でGitHubやTrelloなどのSaaS利用が促進されたこともあり、さまざまな情報がリアルタイムにSlackに流れるようになってきています。また、IRC時代の終わりから利用していたHubotを引き続き利用したり、ikachanからSlackに投稿できるようにすることで、既存資産をそのまま流用できるようにしたりもしました。

Slack上でのChatOps

ChatOpsは、アプリケーションやインフラのフロー情報をチャットツール上に流すことによって、関係者が同じ情報をリアルタイムに共有しながら、日々の運用や障害時の緊急対応をスムーズにするための方法論です。

弊社では、このために次のような情報をSlackに通知させています。

- デプロイ情報
- ステージング環境の起動
- Mackerel[注5]からのアラート情報
- エラーログ情報
- Twitterエゴサーチ結果

デプロイ通知

ChatOpsと言えばデプロイとアラートの通知ですが、まずデプロイ情報連携から紹介します。弊社ではデプロイにCapistranoを利用していますので、Capistranoのスクリプト内でSlack連携のコードを挿入しています。

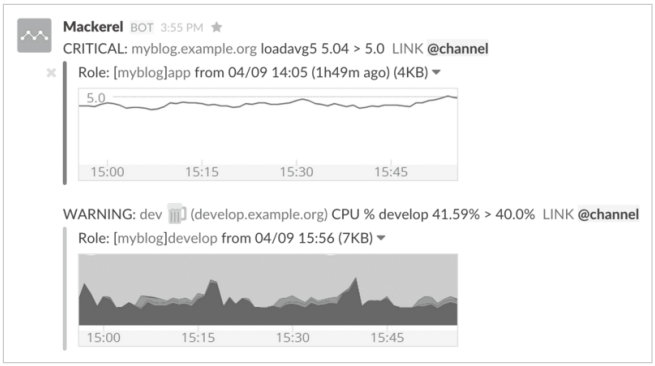
注5) URL https://mackerel.io

実際の様子は**図1**のようになります。ここでは“魚”と“釣り竿”と“寿司”の絵文字が使われていますが、それぞれの魚がデプロイ先のホストを意味し、釣られて寿司になるとデプロイが完了したことを示しています。**図1**では魚が2匹いますので、2台のホストにデプロイされており、すべて寿司になるとデプロイのプロセスがすべて完了したことがわかります。

また、デプロイ先がステージング環境の場合、デプロイしたブランチ用のドメインを用意して、Webサーバを新しく起動することもあります。その際には、新たに起動したWebサーバにアクセスするためのURLを、同時に投稿するようにしています。これにより、ブランチごとにWebサーバを起動したとしても、「アクセス先のURLを間違えてしまう」というミスを減らすことができます。

▼図1　デプロイの様子

▼図2　Mackerelのアラート通知



アラート通知

アラート通知は、「Mackerel」から流すようにしています。Mackerelは弊社で提供しているサーバを監視するためのSaaSで、アラートをSlackを含む各種サービスに投稿できます。また、その際にアラート対象のメトリックグラフを画像として同時に投稿します(**図2**)。これにより、アラート通知がどの程度の緊急性を持っているのか、即時の対応が必要かどうかを直観的に判断できるようになります。

さらに、対応するエンジニアがリアルタイムに流れてくるグラフを同時に見ている前提で、チャット上で対処方法を話し合えますので、より精度が高く、よりスムーズな対応ができるようになります。

Mackerelからのアラート通知以外にもインフラ関連ではエラーログ情報をSlackに流すようにしています。これは「fluentd」[注6]と「fluent-plugin-slack」[注7]を利用して実現しています。fluentdにアプリケーションサーバのエラーログを読み込ませ、fluent-plugin-slack経由でSlackのチャンネルに通知しています。エラーログは出ていることになかなか気づきにくいものですが、Slackに通知することで、いつどのようなエラーが出ているか普段から把握できます。もちろん、障害発生時には重要な参考情報となります。

ただし、エラーログがWarningレベルで普段から大量に出力されている場合、その情報をSlackに流すようにしてしまうと、重要な情報が埋もれてノイズばかりが流れるチャンネルになってしまい、誰も気にかけなくなってしまいますので、S/N比[注8]を適切に保つことが大事です。

また、弊社が提供する「はてなブログ」や「はてなブックマーク」といったサービスの場合、障害発生時にTwitterにユーザ反応が表れることが多くありますので、障害の影響範囲の感覚を掴むためにも、エゴサーチ結果をSlackで見られるようにしています。Twitterのエゴサーチには

注6）URL http://www.fluentd.org
注7）URL https://github.com/sowawa/fluent-plugin-slack
注8）Signal(信号)とNoise(雑音)の比。

fluentdを利用しており、「fluent-plugin-twitter」[注9]と「fluent-plugin-grep」[注10]、「fluent-plugin-suppress」[注11]を組み合わせて実装しています。

Twitterのエゴサーチの結果はノイズが多いこともありますので、適宜fluent-plugin-grepの設定でフィルタリングを行うことでS/N比を一定以上に保つようにしています(「Mackerel」でエゴサーチを行っていると、鯖料理の写真がよく流れてきてお腹が減ってしまいます[注12])。

もちろんTwitterのエゴサーチは障害時だけではなく、新しい機能のリリース時や普段のサービスに関するユーザ反応の体感値を獲得するためにも役に立っています。いろいろな意見をサービス開発にフィードバックするとともに、リアルタイムに反応が可視化されることで、エンジニアのモチベーションを引き上げることもできます(もっとも、ネガティブな意見も可視化されることになりますのでそこは辛いのですが、貴重な意見であることには変わりありません)。

フロー情報とストック情報

このようにSlackをリアルタイムコミュニケーションの中心に置いているわけですが、それだけでは次のような問題があります。

- 発言した内容が流れていってしまい、検索が難しい
- 複数の話が混ざることがあるため、あとから話を追いにくい

これらの問題は、フロー情報とストック情報を意識することで解決できます。Slackでやりとりされる情報はおもにフロー情報で、個々の発言は断片的かつ、あとから参照がされにくい情報です。一方、ストック情報はある程度まとまった量の情報で、あとから参照しやすくしておく情報です。

弊社では、ストック情報を溜める場所として「はてなグループ」を利用しています。はてなグループは2004年にリリースされ、それから10年以上情報を蓄積し続けており、今でも、2005年のはてなブックマークリリース時当初のやりとりを参照できます。

ただ、フロー情報とストック情報の境界はあいまいで、あとで参照する価値のある有意義なディスカッションがSlack上で行われることもあります。Slackのようなフローに強いツールに慣れると、ついつい情報をストックするのが面倒になってしまいます。

そこで、はてなグループにエントリを投稿するとSlackにも流れるようなツールを利用しています(このツールはオープンソースにできておらず、すみません)。

弊社では、チームごとに別のグループを利用しており、グループごとに対応するSlackチャンネルにエントリのURLと概要が流れるようにしています(**図3**)。これにより次のようなストック情報とフロー情報の使い分けを行っています。

❶議論の切っ掛けになるようなストック情報となるエントリを、はてなグループに投稿する
❷それがSlack上に流れてくるのを受けてフロー情報となるリアルタイムのディスカッションを行う
❸議論が収束したら、まとめを再度エントリとして投稿する

▼図3　はてなグループのエントリ投稿

注9) URL https://github.com/y-ken/fluent-plugin-twitter
注10) URL https://github.com/sonots/fluent-plugin-grep
注11) URL https://github.com/fujiwara/fluent-plugin-suppress
注12) Mackerelは日本語で「サバ」。

❹最後にまとめエントリが流れてくるのをみんなで確認し合う

はてなグループに投稿されたエントリはあとから簡単に検索できますので、数ヵ月後に議論を振りかえったり、チームに新しい人が入ってきたときに情報を共有したりできます。

そのほか連携しているサービスとbot

これまでに紹介してきた以外にも、メンバの予定の可視化だったり、サポートまわりのツールであったり、少し柔らかい話題を提供するような連携も設定しています。それらについて紹介します。

Googleカレンダー

Slack ではGoogle カレンダーの予定を流すことができます(**図4**)。予定直前のリマインド通知、予定の追加／変更／削除の通知だけではなく、1日のサマリも、毎日指定した時間に投稿できます。毎朝チームの予定を自動投稿させることで、チームメンバが予定を忘れてしまうことがないようにできます。

desk.com

ユーザからのサポート宛てのメールは、サービスの問題点や改善点を発見するうえで有意義です。弊社では、サポートメールの処理に「desk.com」[注13]を利用しており、サポートメールがSlackに流れてくるようにしています(**図5**)。

engineerkun

「engineerkun」[注14]は、エンジニアのあいだで話題を共有するために、はてなブックマークの指定したタグが付与されたエントリを、定期的にSlackに流してくれるbotです。ときおり話題が提供されることで、会話が始まったりすることがよくあります(**図6**)。

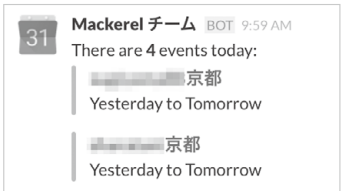
はてなにおけるChatOpsの今後

Slackを利用するようになって1年以上が経過し、本章で紹介したような連携を日々試していますが、まだまだ新しい改善ポイントが見えてきており、進化の余地があります。具体的には、次のような課題が見えてきています。

- 連携が多くなり過ぎると、流れが早くなり過ぎて流れを追うのがたいへんになる
- 1つのチャンネルにディレクター、デザイナなどさまざまな職種の人がいろいろな話をしており、会話と通知が混線する

これらを解決するために、用途ごとにチャンネルを分割したり、連携するサービス／ツールの取捨選択を進めたりしています。みなさんもいろいろ試しながら、自分たちにフィットしたチャットツールの使い方を見い出していってはいかがでしょうか。SD

▼図4　Googleカレンダーとの連携



▼図5　desk.comとの連携

▼図6　engineerkunの様子

注13) URL http://www.desk.com

注14) URL http://hakobe932.hatenablog.com/entry/2014/12/18/193819

第1特集

Case 7

「MYM」でコミュニケーション改革

ヤフーの爆速開発を支える自家製ツール

ヤフーでは、既存のチャットツールを利用するのではなく、新たに「MYM」という独自ツールを開発することで、ChatOpsを実現しました。情報共有を効率化するため、コミュニケーションを円滑にするため、また会社の体制の変化に対応するために行った、機能改善の歩みを紹介します。

Author　市川 貴邦(いちかわ たかくに)　後藤 拓郎(ごとう たくろう)　中根 智大(なかね ともひろ)
光野 達朗(みつの たつろう)　山口 寛(やまぐち ひろし)　ヤフー㈱

ヤフーとChatOps

一口にChatOpsと言っても、その内容はさまざまです。「リリースやジョブの開始を指示する」「アラートの対応を指示する」、はたまた「六曜を教えてもらう」など、日々の業務効率化からちょっとした息抜きまでエンジニアの目的に応じて自由に設計できます。

そんな中、弊社のChatOpsは、とくに情報の可視化・共有、コミュニケーション向上にこだわって発展しました。本章では、それに至る経緯と具体的な事例を紹介します。

MYM

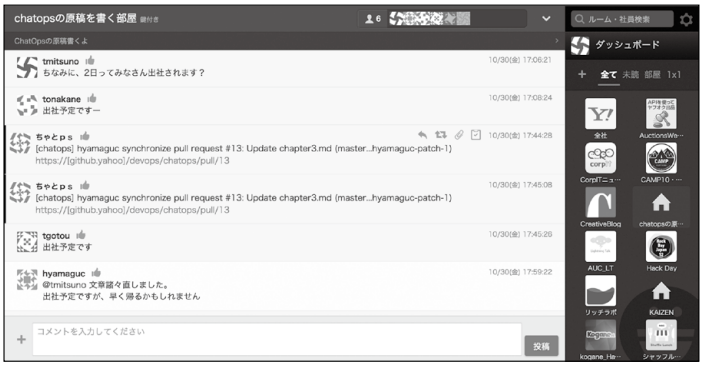
ChatOpsを実践するには、最低限チャットツールが必要となります。最近ではSlackやHipChatが人気ですが、弊社ではMYM(エムワイエム)と呼ばれる内製のチャットツールが広く使われています(**図1**)。チャットツールを内製する理由はいくつか挙げられますが、とくに弊社のChatOpsの立場からみると、カスタマイズ性の面で大きなメリットがあります。チャットツールの本体にまで手を加えられることで、たとえば情報管理やパフォーマンスなど、さまざまな社内ツールと連携するための特殊な要件にも柔軟に対応できます。

当社のケースはさておき、もしまだあなたの会社でチャットツールが導入されていない場合、まずはその選定と普及から始める必要があります。

生い立ち

MYMは2011年2月、Hack Day[注1]と呼ばれる社内ハッカソンで誕生しました。それまで広く利用されていたYahoo!メッセンジャーは1対1でのコミュニケーションを主体とするツールで、巨大化したヤフーという組織には合わなくなってきていました。そこで大規模な組織における情報共有フローの最適化を目指して、グループコミュニケーションを主体としたMYMが開発されました。

▼図1　MYMの画面



注1）24時間でプロトタイピングを行う弊社伝統のイベントです。参加資格を社内に限定しないOpen Hack Dayもありますので、腕に覚えのある方はぜひご参加ください！

ルーム機能

MYMはサーバもクライアントも基本的にJavaScriptで組まれています。クライアントは、PC版もモバイル版も同じコードで動作するWebアプリです。このMYM上で会話のコンテキストを分けるには「部屋」を作成します[注2]。

新しい部屋を作るのに特別な操作は必要ありません。部屋名はURLに含まれていてアクセスした瞬間からその部屋を利用できます。会話の流れから自然に部屋を作成して誘導できる簡便さから、MYM上にはさまざまな部屋が作られ、徐々に文化が醸成されていきました。サービスや組織、特定の言語や技術、部活動や趣味の部屋など、現在では約50,000もの部屋が存在します。ChatOpsもこの部屋単位で行われます。

ユーザ拡張

MYMは全体的に、ユーザスクリプトで変更しやすい設計になっています。こうすることで、ヤフーが誇る2,000人強のエンジニア・デザイナが、細かな不満を勝手に解消してくれます。また、それらのユーザスクリプトをMYM上で列挙してオン／オフできるしくみも用意されており、筋の良い機能は公式の拡張機能としてMYM本体に取り込まれます。あとで紹介する独自bot(**事例3**)も、この拡張機能のしくみを利用して作られています。

いいね機能

MYMでは発言ごとに「いいね」ボタンを押す機能があります。誰かがボタンを押すと発言が派手に動き[注3]、リアルタイムに注目度の高い発言がわかるようになっています(**図2**)。また、押された回数に応じて発言のスタイルが変わるため、あとから読んだときにも発言時の雰囲気を掴めます。この「いいね」数はbotとのコミュニケーションにも応用されています(**事例4**)。

そしてChatOpsへ

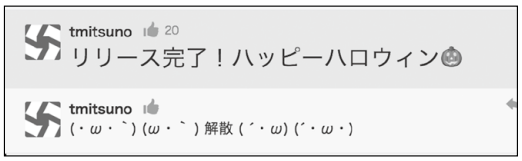
前置きが長くなりましたが、弊社のChatOpsはこのMYM上で展開されています。チャットツール自体は特殊なものですが、個々の事例は他のツールでも再現できるものと思います。これから紹介する事例が、何らか解決の一助になれば幸いです。

現状ではまだChatOpsに関する全社的な方針が決まっているわけではなく、各サービス・各チームが思い思いの形で取り組んでいる状態です。本稿ではそれらの中から抜粋した事例を「片方向」と「双方向」の2パターンに分類して紹介していきます。

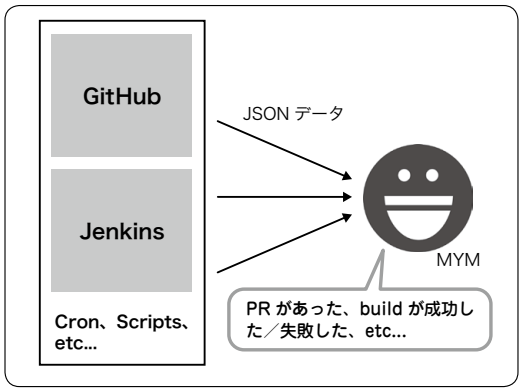
片方向のChatOps：情報の集約と共有

チャットツールへ一方的にメッセージを送るだけのChatOpsです(**図3**)[注4]。この形の場合、必ずしもbotは必要ありません。非常にシンプ

▼図2　「いいね」機能



▼図3　片方向のChatOps



注2) Slackでいうチャンネルです。
注3) 社長がその動きを「ズキュン」と形容したためズキュン機能とも呼ばれます。
注4) 図中右のマークはMYMのロゴです。弊社デザイナーがYahoo!メッセンジャーの"スマイリー"マークをモチーフに制作しました。

ルなのですが、効果はバカにできません。チャットツールさえ開いていれば、アラートであろうとリポジトリの更新であろうと新着情報が1つのツールで手に入るようになります。メールでの通知に比べ、「なんかキタ」「↑のプルリクエスト見てください」と即話題につながるのも大きな違いです。

本節ではリポジトリ変更とサーバ監視での活用事例を紹介します。

▼図4　GHEのissue連携

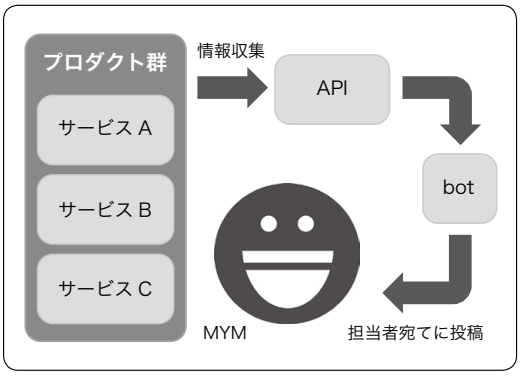
事例1：issue/PRの共有

社内ではGitHub Enterprise(以下GHE)を中心とした開発を行っているのですが、GHEが普及する段階で問題になったのが、issue/pull request(以下PR)が気づかれないという問題です。日常使うサービスのリポジトリはともかく、ちょっとした思いつきを共有し合うようなリポジトリは放置されてしまう傾向にありました。メールでの通知や通知用ブラウザ拡張もありますが、どうしても各人の努力に頼ってしまいます。

そこで、GHEのWebhooks(issuesイベントとpull_requestイベント)をMYMに投稿するという取り組みを始めました(**図4**)。MYMには部屋単位でWebhooks用のエントリポイントが用意されています。たったこれだけなのですが、誰もが目にする場所で共有されることで「issue/PR気づかれない問題」は解決に向かいました。

また、1つだけ工夫がされており、あえて通知専用の部屋を作らず、チームで日々やりとりをする部屋に更新を流しています。通知量など課題はありますが、部屋を分けないことで担当外の案件や修正に対してコメントが付く・議論が始まるという効果が出ており、チャットツールへの情報の集約・共有の恩恵を強く得られる結果となりました。

▼図5　情報収集の流れ



事例2：品質維持のための情報を集める

事例1ではサービス開発時の紹介をしました。事例2では運用フェーズでサービスを安定して届けるためのChatOps例を紹介します。ChatOpsを始める以前は、ライブラリなどのバージョンアップ対応などは人依存となっており、対応する人も限られていました。このため、対応に遅れが出るなど課題を持っていました。

チームとしてプロダクトを支えていくために、チームのコミュニケーションの場に情報をPushする取り組みをしました(**図5**)。しくみは、毎週1回ライブラリなどの更新情報を取得し通知するシンプルなものです[注5]。ポイントは、結果をプロダクトごとの担当者にメンションを付けてMYMのチーム部屋に投稿する点です。コミュニケーションの中心に通知することで漏れがなくなり、プロダクト担当者は早い段階で対応方針の検討を開始できます。また、チームメンバもキャッチアップできるので協力して対応することもできます。以前は対応が遅かったものもすばやく対応できる環境となり、品質の維持につながっています。

注5) 週1回の通知部分について補足しますと、緊急度により通知方法は分かれています。このbotでは1ヵ月程度で対応を求められるもののみ流すようにしています。

継続へのポイント

片方向の事例として2つ紹介しました。どちらもポイントは、通知専用の部屋を作らず日常的なコミュニケーションの部屋に通知を取り入れることです。チームの誰もが目にする場所に通知が来ることで、担当外の人からの気づきが得られたり、情報の共有が促進されたりと高い効果が得られます。始めるときは、ぜひみなさんのチームのチャットルームへの通知を試してみてください。

双方向のChatOps：操作の効率化とコミュニケーション円滑化

botを置いて対話的に進めるChatOpsです（**図6**）。いわゆるChatOpsと聞いて思い浮かべるのはこちらの形ではないでしょうか。片方向のChatOpsでは情報の集約までしかできません。botを利用した双方向のChatOpsが実現すると行動（対応、アクション）までもがチャットツール上に集約されるようになります。

botを準備するため導入のハードルは上がりますが、「XXにある対応手順書を参照して」「どれですか」「あれ、ちょっとまって……」という虚しいやりとりを省略して、

❶チャットツールからbotに指示を送るだけで作業が完了する
❷チームの新メンバもチャット上のやりとりを見れば作業がわかる

という運用の構築ができるようになります。本節では社内での双方向ChatOpsを普及させた事例と、アラート通知に関する事例を紹介します。

事例3：botと自由におしゃべりする

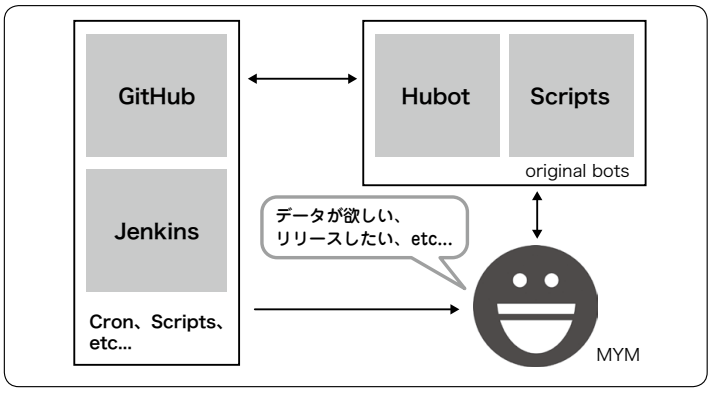
MYMが広まったのは、日本でもHubotなどChatOpsの事例が見られるようになったころでした。このChatOpsによる効率化とコミュニケーションが魅力的で、何より楽しそうだったこともあり、弊社でもすぐに取り組みたいと考えました。その導入を促進するために生まれたのが「Yabot」です。MYMや社内ツールとの相性が良い独自botフレームワークであり、開発者が取り組みやすい環境を実現しています。ここでは、自由にbotとの会話を増やしていけるようにしたしくみと、それを通してどんな会話がなされているかという実例を紹介します。

Hubotを簡素化したしくみ「Yabot」

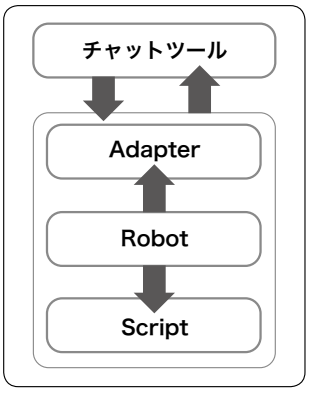
前述のとおり、Hubotに代表されるチャット用botフレームワークがMYMにもほしいと思ったことをきっかけに、そのお手軽版であるYabotを開発しました。

開発の過程では、発言者のみならず閲覧者の分もbotが解釈して実行してしまうなどバグがありましたが、チャットでデバッグしながら完成までこぎつけました。詳細なしくみについては割愛しますが、Hubotになぞらえて用意した

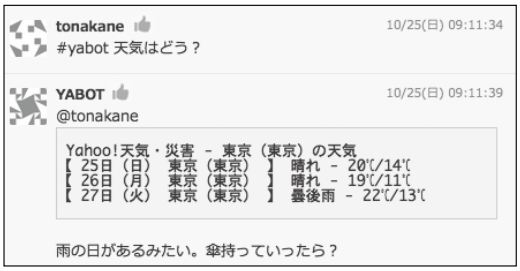
▼図6　双方向のChatOps



▼図7　Yabotの構造



▼図8　天気bot



▼図9　猫画像bot

機構を列挙しておきます(図7)。

・チャットに応じてどのような挙動をするか記述する「Script」
・Scrpitを実行する「Robot」
・Robotとチャットツールを仲介する「Adapter」

「Hello」と呼びかけるのに5分とかからず、また「Script」の追加もチャットを通して行えるなど、導入にあたっての難しさをできる限り排除し、誰でも使えるようにすることを意識しています。

Helloから始まり、生まれた会話

前述のしくみを用意して、社内の誰でも自由におしゃべりできるようにした(=Scriptを追加できるようにした)ことでさまざまな会話が生まれました。

「Hello」と呼びかけ、botが「Hello」と返し疎通確認が完了したあとは、それぞれの会話の始まりです。たとえば「天気はどう？」と聞いたら、botがYahoo!天気サービスの情報から天気を調べて答えてくれます(図8)。

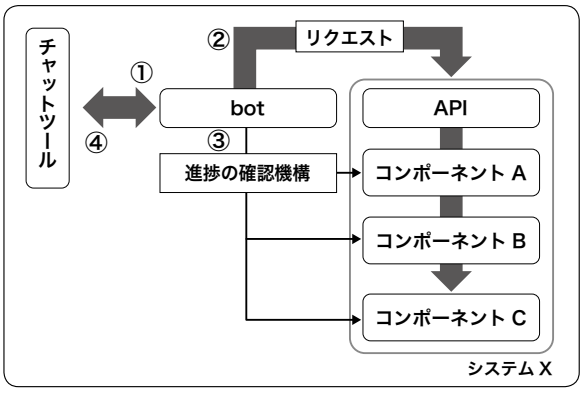
開発者の趣味趣向によって、botの挙動や反応がさまざまなこともおもしろいです。天気を訊いて雨の予報のときは「雨の日があるみたい。傘持って行ったら？」と答えてくれたり、サーバの状態を問い合わせてサービスアウト中のときは、「触れないで」と答えてくれるなど十人十色です。運用作業を簡略化してくれたり、遊び心満載の冗談でチャットルームの雰囲気を良くしてくれたりもします(図9)。

チャットでお手軽な動作確認

次に、実際にシステム運用する際にチームで行っている実例を紹介します。当チームが担当しているシステムはいくつものコンポーネントから構成されており、リクエストを受けてから非同期処理がどこまで進んだのか確認するのも一苦労でした。この動作確認をbotにお願いすることにしたのです(図10)。流れは次のようになります。

①テストする人がシステムの環境とパラメータをbot宛に発言
②発言内容を解析したbotがシステムのAPIをコール
③botが後続のコンポーネントにおいて非同期処理が進んでいるかチェック
④最後のコンポーネントまでチェックが終わるとその結果をbotが返信

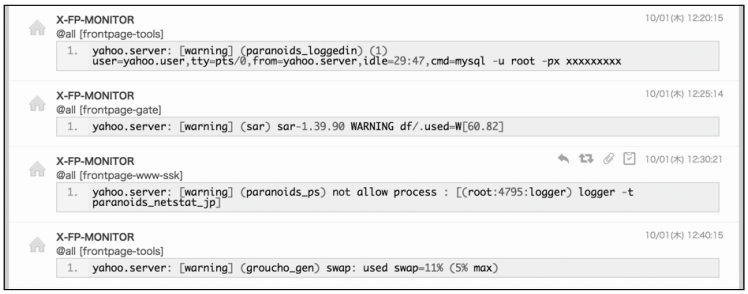
▼図10　動作確認の流れ



このようにすることで、いったい何がうれしいのでしょうか？　いくつかあります。

・動作確認の作業コストが削減できること
・誰でも簡単にテストできること
・テストしたことが記録として残ること
・結果を見ながらのコミュニケーションができること

▼図11　MYMでアラートの見える化



結果として、動作確認がとてもお手軽なものになりました。発言するだけでいいので、担当外の開発者でも企画担当者でもテストできます。チャットのアーカイブはそのままテスト実施記録にもなるので、発言日時における動作保証や遡って調べる際の材料にもなるでしょう。

また、もしテストが失敗しても問題のある個所がわかっているので、あとは開発者が原因を調べるだけです。結果の内容も瞬時に共有されているので、空いているチームメンバがヘルプで入ることもできます。1つの結果を眺めながら、そのままチャットを通して原因や対応についてメンバ同士の議論へと進展する世界ができあがります。

ChatOpsの醍醐味である「誰でもできる」「みんなでできる」ということが、効率化と新しいコミュニケーションをもたらしました。ちょっとした工夫で面倒だった作業を解消し、楽しさをも加えられるのでぜひお試しください。

事例4：アラート通知のChatOps化で機能横断型組織へ

最後の事例はbotそのものに工夫を凝らして、より高度なChatOpsを実現させている取り組みです。監視アラートをチャットに通知することは組織規模の大小を問わず行っていると思います。この事例では、アラート通知にChatOpsを取り入れることで、チームがどのように変化していったかを紹介します。

開発と運用の意識の壁

弊社では2012年に宮坂 学社長の新体制に切り替わるまで、内部統制のために開発と運用の役割を組織単位で分離していました。新体制になってからは、内部統制を維持しつつ新しいことにすばやく挑戦できるように、開発と運用を組織単位では分けず、機能横断型チームで開発することを選択できるようになりました。

機能横断型チームではお互いの業務を尊重し、コラボレーションを発揮することが期待されますが、開発体制の変化に対してメンバの意識変化が追いつかず、元開発のメンバと元運用のメンバの間で壁ができてしまい、運用業務の属人化が問題になっていました。

アラートの見える化「力の1号」

アラートメールを見る習慣がないメンバに「メールをチェックしてね」と言葉で伝えても、それを習慣として根付かせるには不十分です。自分たちの開発スタイルの中に“見る機会”をどうにか組み込んで、運用意識を高めていく必要が出てきます。

すでにMYMを通してコミュニケーションをしつつ開発するスタイルが確立していたため、監視サーバからメールを送りつつMYMにも通知し、開発スタイルを大きく変えずに意識の改善ができないかと思い、見える化を行いました(**図11**)。

多くのChatOpsでは監視サーバからチャットシステムにデータを送ることと思いますが、弊社のネットワークポリシーでは、監視サーバのあるプロダクションから開発環境にある

MYMに直接通知データを飛ばせません。そのため、定期的に開発環境から監視サーバにデータを取得しに行く必要があります。

定期的に実行というとcronを思い浮かべる方も多いと思いますが、cronは“どこで動かしているか忘れる”、“変更記録がなく、最新コードがどこにあるかわからなくなる”といった問題が起きることがあります。

Jenkinsの定期実行を利用すると、

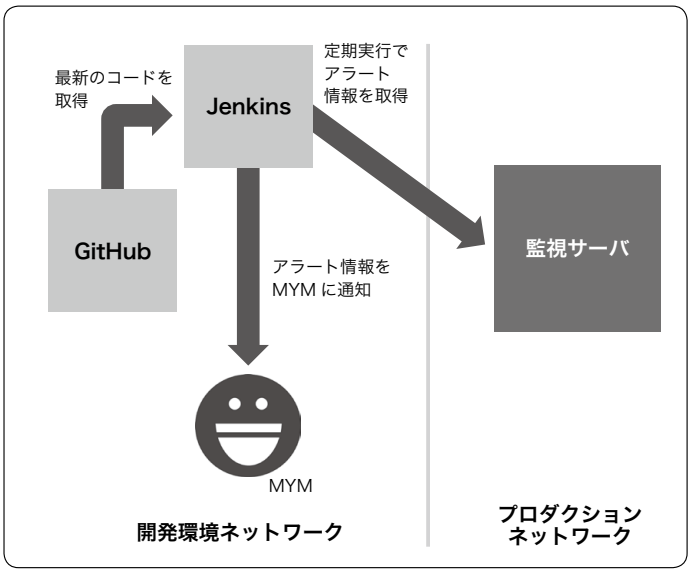
・リポジトリからコードを落とすのでコード管理を行いつつ、常に最新のコードを実行できる
・Jenkinsのjob一覧を見ることでどのような目的で、どのような処理が行われているかがひと目でわかる

といったメリットがあります(**図12**)。

これで改善……しなかった。

これでアラートを定期的に取得しつつMYMに送れるようになりました。目新しさからか、チーム内だけでなくチーム外のメンバも見てくれるようになりました。しかし数週間もすると、通知は見るけど対応まで手が動かず、元運用メンバに対応が集中するという元の状態に戻ってしまいました。botによる見える化は、“見えるようにする”だけでは成功には向かいません。何が問題だったのか、双方向なChatOpsでどう改善できたかを次に説明します。

▼図12　1号システム全体図



アラートの人格化「技の2号」

当時のMYM部屋には、ただちに対応が必要なもの、そうでないものなど、多様な性質のアラートがbotによって並列に投下されていました。それらの判別は人間が経験的に行う必要があり、経験の不足している人間には負担が大きいものでした。それなりの量のアラートが投下される状況の中でこのMYM部屋を見続けるために、botに改良が加えられました。

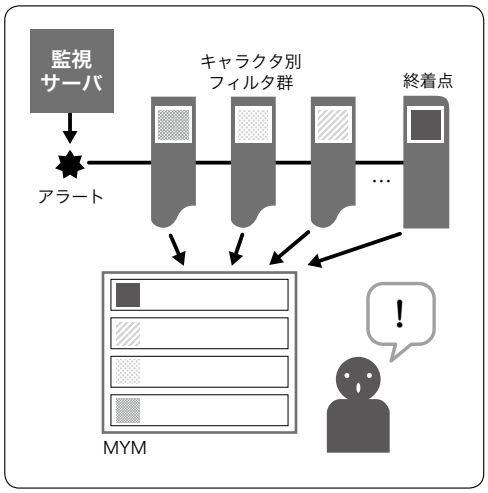
問題のあるアラートをより効率的に見分けるために、「基本的に問題ない」あるいは「ただちに対応が必要」とわかっているものは、フィルタリングして担当のbotを割り振り、さらにMYM上で特徴的なアイコン、名前そしてキャラクタを設定しました。その結果、「この子(bot)が喋ったらすぐ対応」「この子は基本的に大丈夫」といった形で、ひと目でアラートの属性を判別でき、認識の負担が軽減されました。

またアラートのテキストの中で必要とされる情報はごく一部であることが多いため、どのサーバで、何のアラートが挙がったのか、日本語でまとめて報告するようにしました。情報量を最小限に抑え、かつ我々の第一言語である日本語を用いたことで、認識スピードの向上につながりました(**図13**)。

botとのコミュニケーションの導入

前述したキャラクタごとのカテゴリ分けが実装された結果、カテゴリ分けされなかったアラート(**図13**の「終着点」)は、経験的判断を必要とするものと

▼図13　2号システム全体図



▼図14　2号のMYM部屋

して、特別なキャラクタに報告が割り振られました。このキャラクタの発言は、誰かが問題のありなしを判断する必要があるため、MYMの「いいね」機能を使って、判断・対応済みであることをbot側に報告するシステムを導入しました。

このキャラクタの発言に「いいね」をしないと、30分後に未確認のアラートに問題がないか再確認されるため、経験的判断を必要とするアラートの見逃し防止、誰かが確認して対応した／対応が必要ないと判断したことの共有、さらには判断の付かないアラートが発生した際の議論のきっかけを与えることとなりました(**図14**)[注6]。

2号導入の効果

「技の2号」を導入したことによるメリットをまとめました。

- 意味ベースでアラートをまとめることで、botによるポスト数が以前の1/2ほどになった
- アラートの属性とキャラクタを紐付けることで、認識コストが軽減された
- 双方向コミュニケーションの導入で、アラートの確認頻度が向上した
- アラート部屋を起点としたコミュニケーションが活発になり、運用・開発の職能意識の壁がなくなった

まとめ

いかがだったでしょうか。みなさんには馴染みのない弊社独自のチャットシステムですが、ChatOpsで行っていることは一般的なものかと思います。片方向によるチャットシステムへの情報の集約、botを介することによる作業効率化やコミュニケーションの円滑化は、弊社の爆速な開発の支えとなっています。

ぜひみなさんも簡単なものからChatOpsライフに取り組んでみてください。SD

注6）余談ですが、再確認にも「いいね」をしないで放置するとキャラクタが泣き出す仕様にしたところ、これを泣かせまいとしてアラートの確認頻度が向上しました。

第2特集

手軽さとコード化しやすさが人気！

Ansibleでサーバ構成管理を省力化

サーバの構成をコードで管理する“構成管理ツール”が注目されています。コードで管理することによって自動化や手順の明文化ができ、サーバ構築の負担が軽減されます。本特集では、その構成管理ツールの1つである「Ansible」の使い方を解説します。

数ある構成管理ツールのなかからAnsibleが選ばれる理由はどこにあるのか、そしてどんなことができるのか、本特集で確認してみてください。

CONTENTS

第1章 簡単に使い始められます Ansibleの概要とインストール

Author 若山 史郎（わかやま しろう）　ツキノワ㈱

導入の敷居の低さは、Ansibleが評価されている点の1つです。設定対象のサーバ側にはたいてい何の準備も必要とせず、実行用サーバにAnsibleをインストールし、設定に必要な2つのファイルを作るだけです。なにはともあれ、本章を読んで体験してみてください。

なぜ構成管理ツールを使うのか

構成管理とは、インストールされているソフトウェアや設定ファイル、サービスの起動や停止、ネットワーク設定といったサーバの中身、つまり構成を、業務が円滑に運ぶように構築・調整することです。構成管理ツールとはそのためのツールです。構成管理ツールを使うと、少なくとも次の2つの利点が生まれます。

1. 自動化
2. コード化

自動化

人間は必ず間違えます。人の手でサーバの構築などを行うと、間違いが発生する可能性は常につきまといます。さらに、人間であれば同じことを何度も行うと疲れが溜まってきたり、慣れてしまって警告を見逃したりするかもしれません。あるいは人によって手順が異なっているかもしれません。自動化により、間違いが起きる可能性を最小限に抑え込めますし、コンピュータにやらせれば愚直に何度でも同じことを繰り返します。主人たる人間はその間飲み物でも飲んでいればいいのです。

ただ、自動化だけが円滑に事を運べる方法なわけではありません。そもそもその作業が必要なのか、別のもっと良い方法がないか、一度考えてみるのも1つです。その場合に大事なのは標準化です。バリエーションがあると複雑になります。標準化して1つにまとめると、作業自体が必要なくなる場合も多々あります。

コード化

もう1つの利点はコード化です。構成管理ツールを使うには必ず、そのツール用に設定をファイルに記述（コード化）します。つまり、「従来は目に見えなかったり、人間の暗黙知に頼っていた構築手順を、何をするか明確なコードの形に落とし込むことができる」ということです。コードを書くことこそがインフラ構築となるのです。そして、コードは通常テキストファイルで記述されるので、検索が可能になったり、差分がわかりやすくなるという利点も生まれます。また、わざわざ実サーバ上で確認しなくても、コードを見れば設定値などがすべてわかるようになります。

コードはGitやMercurialなどのバージョン管理システムと組み合わせることでより便利になります。さらにはGitHubのPull RequestやIssueとも組み合わせると、作業フローも明確化されます[注1]。Ansibleに限らず構成管理ツールを導入するならば、バージョン管理システムも一緒に導入することをお勧めします。

◆ ◆ ◆

仮想化やクラウドのおかげでサーバ構築が楽になり、インフラ系エンジニアだけでなく、プログラマ（おもにWeb系）も簡単に扱えるよ

注1）GitHubによる管理の一例はAppendix 2を参照ください。

うになった近年、このような利点をもった構成管理ツールを活用するところが増えています。

構成管理ツールは、次のような人にとって助けとなるツールです。

・日常的にサーバの新規構築や構成変更をしている人
・日々変わっていくアプリケーションをサーバにデプロイしている人
・忘れっぽくてすぐに構成や設定値を思い出せなくなる筆者のような人

つまり、何らかの形でサーバにかかわる人は利用の検討をする価値があるでしょう。

Ansibleとは

Ansible[注2]は、構成管理ツールの1つです。類似のソフトウェアには、Puppet[注3]やChef[注4]があります。

Ansibleは、Ansible社が中心となって開発しています。ソースコードはGitHub上で公開され、非常に多くの人がかかわっています。GitHubのスターは13,000を超え、Issueやメーリングリストも非常に活発です。

なお、Ansible社は2015年10月にRed Hat社に買収されました。Red Hat社は「自動化ツールは今後重要になり、その中でも複数のクラウドに対応し、簡単に使い始められるAnsible社を買収した」と発表[注5]しています。Ansible社は買収後も変わらず活発に開発を続けています。

Ansibleの特徴として、次の3つが挙げられています。

・**Simple**：Ansibleは最低限必要なファイルが2つだけと、簡単に始められる。また、YAMLという形式で書くため、プログラマでなくても使える
・**Agentless**：対象となるサーバに特別なツールをインストールする必要はない。SSHで接続できれば動かせる
・**Powerful**：単なる構成管理ツールではない。アプリをデプロイしたり、クラウド上にインスタンスを立ち上げたり、あるいは多数のサーバからログをローカルに一気に持ってきたりと、いろいろな使い方が可能

顧客のサーバを扱っていて、エージェントなどをインストールすることが難しい場合など、とくに**Agentless**はAnsibleの大きな魅力となっています。

また、Amazon Web Services(AWS)やGoogle Cloud Platformなど多くのクラウドサービスと連携するための機能を持っています。たとえばAmazon ELBにインスタンスを付けたり外したりする機能や、Zabbixの監視を一時的に停止する機能などです。これらの機能があることで、単なる構成管理ではなく、デプロイにまで使える強力なツールとなっています。AWSとの連携に関してはAnsible公式ドキュメント[注6]がありますし、各モジュール(モジュールについては後述)に使い方の例が書かれていますので、参考にしてください。

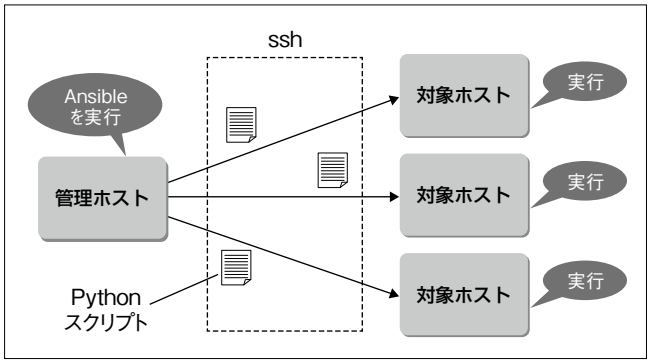
Ansibleの動作概要

前述のとおり、AnsibleはSSHだけで動作します。より正確に言うと、実行対象となるサーバにPython 2.4以上が必要なのですが、現在使われているLinuxディストリビューションのほぼすべてでPythonが最初からインストールされていますので、あまり問題はありません(FreeBSDなどでは入っていないので別途インストールする必要があります)。

Ansibleを実行するホストを**管理ホスト**

注2) http://www.ansible.com/
注3) https://puppetlabs.com/
注4) https://www.chef.io/
注5) https://www.redhat-cloudstrategy.com/why-did-red-hat-acquire-ansible/
注6) http://docs.ansible.com/ansible/guide_aws.html

▼図1　Ansibleの動作



(Control Machine)と呼びます。一方、対象となるホストを**対象ホスト**(Managed Node)と呼びます(**図1**)。

管理ホストで実行されたAnsibleのコマンドは、対象ホストに対してSSHで接続を確立します。その後、管理ホストで生成したPythonスクリプトを対象ホストに送り込み、対象ホスト上で実行します。実行が成功しても失敗しても、Pythonスクリプトは Ansibleのコマンドが終了するときに削除されます。

対象ホストを複数指定した場合、Ansibleは各対象ホストに対して並列にPythonスクリプトを実行します。デフォルトでは5台まで並列実行されますが、コマンドラインオプションで同時実行数を変えられます。SSHですので、並列に実行するための管理ホスト側のコストはさほどでもなく、100台程度に対して並列に実行しても大丈夫です。

Ansibleのインストール

では、Ansibleを管理ホストにインストールしてみましょう。対象ホストではPython 2.4以上であれば動作しますが、管理ホストにはPython 2.6以上が必要です(ただし、Python 2.4では`simplejson`というライブラリを別途インストールする必要があります)。

インストールにはpip[注7]というPython標準のパッケージ管理ツールを使うと、最新版を手に入れられます。インストール時にopensslをビルドするため、あらかじめ次のようにしてpython-devをインストールしておきます。

```
$ sudo apt-get install python-pip python-dev
$ sudo pip install ansible
```

また、各ディストリビューション標準のパッケージ管理ツールを使う方法もあります。その場合、バージョンが古いことがありますので、適宜確認してください。yumでインストールする場合はEPELのインストールが別途必要です。

```
$ sudo apt-get install ansible
もしくは
$ sudo yum install ansible
```

執筆時点ではAnsibleの最新バージョンは1.9.4です。本稿も1.9.4を前提としています。ただし、現在次バージョンの2.0がβテストに入っており、近々リリースされるものと思います。2.0では1.9までに対して互換性がありますし、本稿で紹介した内容は2.0 βでもそのまま動くことを確認しています。

AnsibleはPythonで実装されていますが、Ansibleを使うときには基本的にPythonの知識は必要ありません。これから解説する**YAML**という形式で記述していきます。ただ、本特集では述べませんが、Pluginという Ansible本体を拡張するしくみを自作する場合にはPythonの知識が必要になってきます。

試しに動かしてみる

インストールが終わったら、早速動かして

注7) https://pip.pypa.io/

▼図2　テスト実行環境

▼リスト1　hostsファイル（Inventory）

```
test ansible_ssh_host=192.168.10.10
```

▼リスト2　dir.ymlファイル（Playbook）

```
- hosts: test
  tasks:
    - file: path=/tmp/ansible-test state=directory
```

▼図3　リスト1、2を使ったAnsibleの実行結果

```
PLAY [test] ********************************************************************

GATHERING FACTS ****************************************************************
ok: [test]

TASK: [file path=/tmp/ansible-test state=directory] ***************************
changed: [test]

PLAY RECAP *********************************************************************
test                       : ok=2    changed=1    unreachable=0    failed=0
```

みましょう。Ansibleを使うのに必要なファイルは最低限2種類です。

- Inventory：対象ホストへのアクセスに必要な情報を記載する
- Playbook：対象ホスト上で実行する内容を記載する

細かい説明は後ほど行います。

テスト実行のイメージは**図2**のようになります。まず、**hosts**というファイルに**リスト1**の内容を記載します。これがInventoryファイルとなります。192.168.10.10の個所は認証なしでsshログインできるIPアドレスを記載してください。認証なしでsshするには、パスフレーズを設定しない鍵認証や、ssh-agentなどを利用してください。

ヒント sshでパスワード認証がある場合、コマンドに-kオプションをつけて実行するとパスワードを聞いてくるようになります。

次に、**dir.yml**というファイルに**リスト2**の内容を記載します。インデントが重要なので、間違えないようにしてください。半角スペースで2文字です。これがPlaybookファイルです。書き終わったら早速実行してみましょう。

```
$ ansible-playbook -i hosts dir.yml
```

図3のように表示されるはずです。誌面では白黒ですが、実際にはok: [test]が緑色で、changed: [test]が黄色で表示されます。実行後、/tmp/ansible-testというディレクトリが指定したホスト（対象ホスト）上に作成されていると思います。

ここまでAnsibleの概要を説明し、使い方を体験していただきました。次章から、Ansibleの使い方をもう少し学んでいきます。SD

第2章 非プログラマでも読み書きしやすい InventoryとPlaybook、2つのファイルを理解する

Author 若山 史郎（わかやま しろう）　ツキノワ㈱

第1章でAnsibleのインストールを行い、どんなことができるかを体験してもらいました。本章では、Ansibleの詳しい使い方を説明します。

接続情報――Inventory

第1章で書いたhostsというファイルには、対象ホストへの接続情報を記載しています。このファイルを「Inventory」と呼びます。

Inventoryの作成

Inventoryは、ini形式（正確にはini形式を拡張した形式）で記述したファイルです。第1章での例は簡単すぎましたので、**リスト1**に別の例を示します。このファイルを/etc/ansible/hostsに作成すると、自動的に読み込んでくれます。あるいは、第1章でのコマンド実行例のように-iオプションで指定もできます。

[web]や[db]は**グループ**を示します。また、web01、db01などは**サーバ**を示します。つまり、この例ではdbというグループにdb01.example.com、db02.example.comという2つのサーバが含まれていることを示しています。また、mail.example.comはどのグループにも属していません。ただし、allというすべてのホストが属するグループが暗黙的に作成されており、その中には入っています（**図1**）。

なお、同じサーバを複数のグループに所属させることもできます。その場合、複数グループを実行対象として指定しても、同じサーバに対しては1回の操作しか行われません。

サーバの設定

Inventory内では各サーバに対してさまざまな設定ができます。たとえば**リスト2**のように書くと、web01.example.comに対しては192.168.1.10のポート2222で接続しにいきます。

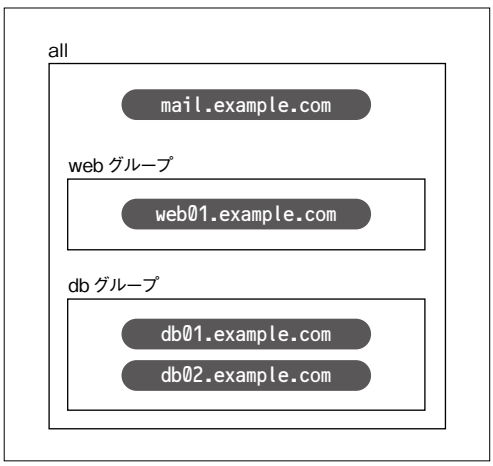
▼リスト1　Inventoryの例その1

```
mail.example.com

[web]
web01.example.com

[db]
db01.example.com
db02.example.com
```

▼リスト2　サーバの設定

```
web01.example.com ansible_ssh_ホ
host=192.168.1.10 ansible_ssh_port=2222
```

▼図1　リスト1のイメージ



▼表1　Inventory内での主な設定例

設定変数名	説明
ansible_ssh_user	sshで接続するユーザ名
ansible_ssh_private_key_file	秘密鍵ファイル
ansible_ssh_pass	接続パスワードを**平文で**指定できる。もちろん安全ではないのでどうしても必要なとき以外は使わないように！
ansible_python_interpreter	コマンドが/usr/bin/pythonでない場合に対象ホストのpythonコマンドを指定する（例：ansible_python_interpreter=/usr/bin/python2）

▼リスト3　グループの設定

```
[db]
db01.example.com
db02.example.com

[db:vars]
ansible_ssh_port=2222
ansible_ssh_user=admin
job=db    # これは変数です
```

▼リスト4　変数の使い方

```
- hosts: all
  tasks:
    - file: path=/tmp/ansible-{{ job }} state=directory
```

ほかにも**表1**のような設定ができます。

グループの設定

サーバごとにansible_ssh_portなどでひとつひとつ設定していく場合、台数が多いと大変になります。せっかくグループを分けているのですから、グループ単位で設定しましょう。

リスト3のように:varsを書くと、グループでの設定をまとめて記述できます。この例ではdbグループに属しているすべてのサーバを一括で設定しています。

変数について

リスト3にjob=dbという行がありました。これは、SSH接続の設定ではなく、**変数**です。変数は条件判断や値を埋め込んだりといった用途で使います。**リスト3**の例ではdbグループに対して、jobをdbというグループ変数として設定しています。

- **グループ変数**：グループ単位で設定する変数
- **ホスト変数**：ホストごとに設定する変数

ホスト変数とグループ変数の両方が定義されていた場合、ホスト変数のほうが優先されます。

変数を使うには、変数名を{{ }}で囲みます。**リスト4**にその例を示します。この例ではjobという変数にdbが設定されているので、/tmp/ansible-dbというディレクトリが作成されることになります。変数の設定方法については後ほどvarsセクションの節で説明します。

手順ファイル——Playbook

Ansibleでは手順ファイルを「Playbook」と呼びます。Playbookには変数やそのPlaybookを実行する際に必要な設定など、Ansibleで実行する一連の手順を記載します。

PlaybookはYAMLと呼ばれる形式で記載します。Playbookを今すぐ書きたい気持ちをぐっとこらえて、まずYAML形式を簡単に説明します。

YAML形式

YAMLは入れ子になった構造や、シーケンス（配列）・マッピング（辞書）のデータ構造を読みやすく記述できます。

まず、YAMLの基本的な書き方を説明します。より詳細は「プログラマーのためのYAML入門（初級編）[注1]」などを参考にしてください。

注1） http://magazine.rubyist.net/?0009-YAML

シーケンス

シーケンスとは、リストや配列などと呼ばれているデータ構造です。行頭に-をつけるとシーケンスとなります。

```
- A
- B
- C
```

また、次のように1行に書くこともできます。

```
[A, B, C]
```

マッピング

連想配列や辞書、マップとも呼ばれているマッピングは次のように:の後に半角スペースを1つ以上入れて書きます。

```
A: aaa
B: bbb
C: ccc
```

:の後ろに半角スペースを入れないとエラーになります。意外と引っかかることが多いので注意してください。

階層構造

YAMLでは半角スペースでインデントすることでデータの階層構造を表せます。

```
A: aaa
B:
  B1: bbb1
  B2: bbb2
C: ccc
```

インデントする文字数は決められていませんが、2文字の場合が多いです。また、タブ文字(ハードタブ)は使えません。

なお、シーケンス同士で階層構造を作り、箇条書きのように書くことはできません。たとえば、次のように記述したとします。

```
- A
- B
  - C
  - D
- E
```

これはB - C - Dとつながってしまい、意図どおりにはなりません。階層構造にするためには、次のように書く必要があります。

```
- A
-
  - C
  - D
- E
```

また、マッピングで次のように書くことはできず、エラーになります。

```
a: aaa
  a1: aaa1
  a2: aaa2
```

a: aaaの次の行にインデントをつけてマッピングの要素は書けません。書く場合には次のようになります。

```
a:
  a1: aaa1
  a2: aaa2
```

column なぜYAML?

プログラム言語ではなくYAML形式を使うことで、書き方が制限され、柔軟な制御はできなくなります。しかし、そのために書き方が統一され、プログラマ以外にも書きやすくなります。また、PlaybookファイルはYAMLなのにInventoryファイルはini形式で書かれていることに違和感を持たれるかもしれません。Ansibleの開発者によると、Inventoryファイルには複雑な階層構造が必要ではないため、より単純なini形式を選択したとのことです。

シーケンスとマッピングの組み合わせ

シーケンスとマッピングは組み合わせて階層構造にできます。**リスト5**のものは実際にAnsibleで使われている例です。

コメント

#をつけることでコメントを記述できます。

```
# ここにコメントを書けます
a:  # ここもコメントです
  a1: aaa1
```

複数行にまたがって書く

Ansibleを使っていると、たくさんの引数を記述したいときがでてきます。その場合、改行すると見やすくなります。改行後にはインデントが必要になります。改行をすると、実際には空白区切りの1行として扱われます。

```
- glance_image:
    login_username=admin
    name=cirros
    state=present
```

シェルスクリプトなど、改行自体に意味がある場合は次のように|を付けます。

```
- shell: |
    ./configure
    make
    make install
```

▼リスト5　シーケンスとマッピングの組み合わせ例

```
- hosts: web
  become: yes
  vars:
    - required: ['docker', 'pyyaml']  # シーケンスを指定
  tasks:
    - name: docker imageを作成
      docker_image: name="my/app" state=present
```

▼リスト6　Inventoryの例その2

```
[web]
web01.example.com ansible_ssh_host=192.168.0.1
web02.example.com ansible_ssh_host=192.168.0.2
```

書いてみる

お待たせしました。いよいよPlaybookを書いてみましょう。まず、**リスト6**のInventoryがあるとします(**図2**)。このInventoryを/etc/ansible/hostsに保存し、自動的に読み込まれるようにしておきます。あるいは、実行時に-iでファイルを指定します。

では、このInventoryにあるwebグループに、基本的なWebアプリの実行に必要な環境を構築するPlaybookを書いてみましょう。

column　YAMLの確認

YAMLがおかしいと、Ansibleはコマンドを実行したときに指摘してくれます。

あるいは、**--syntax-check**を付けて実行すると、形式チェックだけを行ってくれます。CIに組み込むなどすると便利でしょう。

```
ERROR: Syntax Error while loading YAML script, test.yml

The error appears to have been in 'dir.yml': line 3, column 3, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

  job:testjob
  ^
```

UTF-8形式で**リスト7**の内容を記述します。ファイル名はweb.ymlとします。

保存したら、**図3**のようにansible-playbookコマンドで手順ファイル(web.yml)を指定して環境構築を行います。

誌面は白黒なためわかりませんが、changedが黄色く表示されると思います。これで、記した手順がすべて実行されました。appuserというユーザが作成され、/var/log/appディレクトリが作成され、nginxがインストールされたはずです。

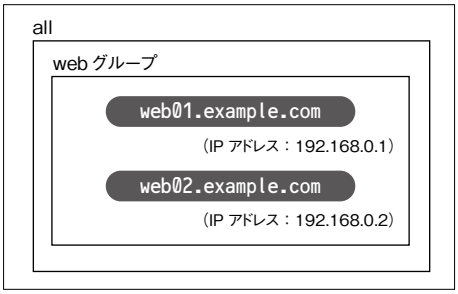
もう一度実行してみましょう(**図4**)。

今度は緑色でokとなりました。okとは、もうすでに環境が構築されているため、2回目は変化が起きることなく終了したということです。

このように、何度同じコマンドを実行しても対象ホストは**同じ状態**になります。これを**冪等性(べきとうせい)があると**呼びます。冪等性がない場合、同じ設定を何度も行ってしまったり、複数回実行したときにエラーになったりします。Ansibleのほとんどのモジュールは冪等性を持つように作成されています。

冪等性があるため、Playbookは何度実行しても構いません。そのため、AnsibleのPlaybookは少しずつ「育てていく」という書き方ができます。つまり、少し書いて実行し、また少し書いて実行、という書き方です。変更していない個所は実行されないので、実行するansible-playbookコマンドを変える必要はありません。

▼図2　リスト6のイメージ



▼リスト7　Playbookの例

```
- hosts: web    # 対象ホストを指定。今回はwebグループ
  become: yes   # sudoを行う
  vars:         # 変数指定
    logdir: /var/log/app
  tasks:        # 実行するtaskの指定を開始
    - name: 実行用ユーザの作成      # taskの名前
      user: name=appuser
    - name: ログディレクトリの作成
      file: path={{ logdir }} state=directory
    - name: nginxのインストール
      apt: name=nginx state=installed
```

▼図3　ansible-playbookコマンドの実行

```
$ ansible-playbook web.yml↵

PLAY [web] ***************************************

GATHERING FACTS ***********************************
ok: [web01.example.com]
ok: [web02.example.com]

TASK: [実行用ユーザの作成] ************************
changed: [web01.example.com]
changed: [web01.example.com]

TASK: [ログディレクトリの作成] ********************
changed: [web01.example.com]
changed: [web01.example.com]

TASK: [nginxのインストール] ***************
changed: [web01.example.com] => (item=nginx)
changed: [web01.example.com] => (item=nginx)

PLAY RECAP ****************************************
web                        : ok=4    changed=3    unreachable=0    failed=0
```

▼図4　ansible-playbookコマンドの実行（2回目）

```
$ ansible-playbook web.yml↵

PLAY [web] ***************************************

GATHERING FACTS ***********************************
ok: [web01.example.com]
ok: [web02.example.com]

TASK: [実行用ユーザの作成] ************************
ok: [web01.example.com]
ok: [web01.example.com]

TASK: [ログディレクトリの作成] ********************
ok: [web01.example.com]
ok: [web01.example.com]

TASK: [nginxのインストール] ***************
ok: [web01.example.com] => (item=nginx)
ok: [web01.example.com] => (item=nginx)

PLAY RECAP ***************************************
web                        : ok=4    changed=0    unreachable=0    failed=0
```

Playbookの解説

では、**リスト7**のPlaybookを解説します。Playbookは多くの場合3つのセクションで構成されています。

- targetセクション：実行対象の設定
- varsセクション：変数の設定
- tasksセクション：実行するtaskの設定

これらのセクションについて説明していきます。

targetセクション

対象ホストを指定する部分をtargetセクションと呼びます。**リスト7**から抜き出すと**リスト8**の部分です。

hostsで対象を指定します。ここの例ではwebグループを指定しています。このhostsの指定は必須です。

このPlaybookでは、ユーザの作成などにroot権限が必要なため、becomeを指定しています。becomeとは、「別のユーザで実行する」ことを指示する設定で、デフォルトではsudoを実行します。以前はsudoと指定していましたが、1.8から変更されました。

そのほか、いろいろな設定を指定できます。**リスト9**にその一部を示します。

▼リスト8　targetセクション

```
- hosts: web    # 対象ホストを指定。今回はwebグループ
  become: yes   # sudoを行う
```

▼リスト9　設定例

```
- hosts: web    # 対象ホストを指定。今回はwebグループ
  serial: 10    # 同時実行数を指定。デフォルトは5
  remote_user: newuser   # 接続先ユーザ名
  gather_facts: no   # 後述のfactを収集するかしないか
```

▼リスト10　varsセクション

```
vars:          # 変数指定
  logdir: /var/log/app
```

varsセクション

varsセクションでは使用する変数を設定します。具体的には**リスト7**の**リスト10**の部分です。変数は先述の「変数について」の節で述べたInventoryで設定するグループ変数およびホスト変数や、このvarsセクションで定義できるほか、後ほど出てくるroleで定義したり、

2.0からはタスク単位で定義できたりと、さまざまな個所で定義できます。

Ansibleでは変数の名前空間が1つしかありません。ほかの個所で同じ名前で定義された変数は優先順位に従い上書きされますが、その優先順位がわかりにくいという問題があります。たとえばvarsセクションで定義した変数は、Inventoryで定義したホスト変数やグループ変数よりも優先度が高くなります。変数にはなるべく重複しないような名前を付けたほうが問題が少なくなります。

リスト10ではlogdirという変数に/var/log/appという値を設定しています。この変数はあとでログディレクトリの作成に使っています。なお、YAMLの制限でfoo-barのように-がつく変数名は使用できません。

varsは複数指定したり、入れ子構造や配列を指定することも可能です(**リスト11**)。入れ子構造の子の内容を指定する場合は、ドット(.)で区切ります。

また、varsではなくvars_filesを使うと、別のYAMLファイルから変数を読み込めます。変数が多くなってきた場合には別ファイルに分割し、vars_filesを使用したほうが見通しが良くなります。vars_filesは複数ファイルを読み込めます(**リスト12**)。

同じ名前の変数が設定されていた場合、下にあるほうが後から読みこんで上書きします。そのため、最初にデフォルト値を記述したファイルを指定すると良いでしょう。

▼リスト11　複雑な変数指定

```
vars:     # 変数指定
  username: newuser
  group: admin
  shell:
    bash: /bin/bash
    zsh: /bin/zsh
tasks:
  - user: name={{ username }} group={{ ➥
group }} shell={{ shell.zsh }}
```

▼リスト12　別のYAMLから変数を読み込むvars_files

```
vars_files:
  - default.yml
  - var.yml
```

▼リスト13　tasksセクションの例

```
tasks:        # 実行するtaskの指定を開始
  - name: 実行用ユーザの作成    # taskの名前
    user: name=appuser
```

tasksセクション

tasks以下でPlaybookで実行するtaskを指定します。taskは、1つのモジュールとそれに対する引数で構成されます(**リスト13**)。

この例ではtaskに対する名前をnameで指定し、taskの内容としてuserモジュールを使用して、さらにモジュールへの引数を指定しています。

それぞれは次の意味となります。

- **name**：そのtaskの名前。必須ではないが、付けておくとログに表示されたり、日本語で何をしているかがぱっと見てわかるため、指定することを推奨
- **user**：実行するモジュール。リスト13のnameはuserモジュールの引数。引数はモジュールごとに異なる

1つのPlaybook内で複数のtaskを実行できます。その際インデントはそろえる必要があります。前述の**リスト7**では、次の3つのモジュールを使うtaskを定義しています。

- **user**：ユーザの作成、削除を行う
- **file**：ファイルやディレクトリ、シンボリックリンクの作成や削除を行う
- **apt**：apt-getでパッケージをインストールする

これらのモジュールについては後述の「モジュール紹介」に記載していますので、参考にしてください。

1つのPlaybook内に書けるtaskの数に制限はありません。ただし、長いと見にくくなる

ので、後述するincludeやroleで別のファイルに分割する必要がある場合が多いと思います。

taskの実行順

taskは必ず上から順に**記述された順番で**実行されます。**リスト7**では、user、file、aptの順です。この「上から順に実行される」はAnsibleでの基本ポリシーとなっています。

別の例を示します。**リスト14**の例では、ユーザを追加した後に、ホームディレクトリに.bashrcをコピーしています。逆の順番で実行されてしまっては問題が起きます。このような問題が起こりにくいように、taskは常に記述した順番に実行されます。

モジュール紹介

Ansibleは、200以上(Ansible 2.0からは400以上)のモジュールが最初から使えるようになっており、実に多くの用途に対応できます。しかし、一般的にはそこまで多くの種類のモジュールを使う必要はありません。ここではよく使われるモジュールについて簡単に説明します。そのほかのモジュールについてはAnsibleの公式ドキュメント[注2]を参照してください。

debug

デバッグ用のメッセージを出力します。最初がこれか、と思われるかもしれませんが、開発時に一番多く使うモジュールです(**リスト15**)。

表示する変数がマッピングを含むなど複雑な構造をしていても、そのまま表示してくれます。開発時、後述するwhenによる条件分岐を使うときなどにとくに便利です。

script

管理ホストに置かれているスクリプトを対象ホストに転送し、対象ホスト上でそのスクリプトを実行します。既存のシェルスクリプトをそのまま流用できるので、それまでに使用していた自作構築スクリプトからの移行に便利です。

リスト16で呼び出しているcommand.shというファイルは、Playbookと同じ場所からの相対パス指定、あるいは絶対パスでの指定となります。

また、createsという引数を指定すると、そのファイルがあった場合には実行せずにスキップします。スクリプトの最後でこのファイルを作るようにすれば、何度実行しても1回しか実行されません。つまり、シェルスクリプトであっても冪等性を備えられます。

shell

shellモジュールは、任意のコマンドを実行します。>や|を使うこともできます(**リスト17**)。

引数createsを設定すると、そのファイルがある場合は実行されません。また、引数

▼リスト14　tasksセクションでの実行順について

```
tasks:
  - name: ユーザを追加
    user: name=appuser
  - name: bashrcをコピー
    copy: src=~/.bashrc  dest=/home/appuser/.bashrc
```

▼リスト15　debugモジュール

```
tasks:
  - name: somevarの中身を表示する
    debug: msg="somevar is {{ somevar }}"
```

▼リスト16　scriptモジュール

```
tasks:
  - name: command.shを実行する
    script: command.sh
  - name: files/other.shを実行する。/tmp/done.txtが
あれば実行しない
    script: files/other.sh  creates=/tmp/done.txt
```

▼リスト17　shellモジュール

```
tasks:
  - name: targetという行を抜き出します
    shell: grep "target" /tmp/list >> /tmp/list.out
```

注2）Ansible Documentation　http://docs.ansible.com/

chdirを設定すると、そのディレクトリに移動した後にコマンドを実行します。

shellモジュールは大変便利ですので、なんでもshellモジュールで実行したくなるかもしれません。ただし、creates引数である程度の冪等性があるとはいえ、基本的にはありません。そのため、なるべく用意されているモジュールを使うほうが後々問題が少なくなります。

file

ファイルやディレクトリの作成、所有者やグループの変更、シンボリックリンクの作成などを行います(**リスト18**)。

stateには**表2**の種類があります。fileモジュールだけでファイルやディレクトリに関するだいたいのことが実現できます。

template

Pythonでよく使われているJinja2[注3]というテンプレート言語を使って、ファイルのコンテンツを記述したテンプレートに対して変数を埋め込み、対象ホストでファイルを生成します。Jinja2で変数を埋め込むには{{ }}で変数名を囲みます。

nginxの設定ファイル(の一部)を作成する例を示します。templateモジュールと変数を使うと、たとえば役割ごとに違う内容の設定ファイルを置くことなどが可能になります。

まずtemplate.j2というファイル名でテンプレートファイルを作成します(**リスト19**)。

Playbookのvarsセクションで変数を設定し、templateモジュールを使います(**リスト20**)。

結果として、**リスト21**のファイルが/etc/nginx/nginx.confに作成されます。

この例ではworker_processesが1ですが、たとえば本番環境では異なる値にした設定ファイルを設置することなどが、変数を置き換えるだけで実現できます。

注3) http://jinja.pocoo.org/

▼リスト18　fileモジュール

```
tasks:
  - name: /etc/example.confを設定
    file: path=/etc/example.conf owner=admin group=admin mode=0644
  - name: シンボリックリンクを作成
    file: src=/etc/link/to dest=/etc/symlink state=link
  - name: ディレクトリ作成
    file: path=/etc/deep/dir/ex state=directory
  - name: /home/admin以下すべてのファイルのownerを設定
    file: path=/home/admin owner=admin recurse=yes
```

▼リスト19　templateモジュールで設定ファイルを作成する例

```
user {{ user }};
worker_processes  {{ worker_process_num }};

error_log  {{ error_log_path }};
pid        /var/run/nginx.pid;
```

▼リスト20　varsを利用したtemplateモジュール

```
vars:
  user: www-data
  worker_process_num: 1
  error_log_path: /var/log/nginx/error.log
tasks:
  - name: nginx.confを設定します
    template: src=template.j2 dest=/etc/nginx/ﾏ
nginx.conf
```

▼表2　stateの種類と処理内容

state	処理
file	ファイルの属性を設定する。もしファイルが存在しない場合、エラーになる
link	シンボリックリンクを作成する
directory	ディレクトリを指定した場合、深い階層でも途中のディレクトリをすべて作成する。recurse=yesを指定すると再帰的に指定したディレクトリすべてを変更する
touch	ファイルの属性を設定する。ファイルが存在しない場合作成する
absent	ファイルやディレクトリを削除する。シンボリックリンクやハードリンクの場合そのリンクを削除する

unarchive

管理ホストに置かれている圧縮ファイルを対象ホストに転送し、展開します。.tar.gzや.zipなどさまざまな圧縮ファイルを扱えます(**リスト22**)。また、copy=noを付けると、転送せずに対象ホスト内にあるファイルを展開します。

apt

aptコマンドを使ってパッケージをインストールします(**リスト23**)。また、パッケージの削除も行えます。

Ansibleには、aptだけではなくyumやpkg、pacmanなど、各種のOS、ディストリビューションのパッケージ管理ツールがあり、ほぼ同じ書き方で使えます。また、ツールによってはキャッシュの更新の有無など、そのツール独自の機能も指定できます。

user

ユーザを追加、あるいは削除します(**リスト24**)。

passwordを指定することでパスワードを設定できます。このパスワードはハッシュ化されている必要があります。passlibをPythonでインストールすると、**リスト25**のコマンドでハッシュ化されたパスワードが標準出力に表示されます。

そのほかの機能

AnsibleのPlaybookは完全なプログラミング言語ではありませんが、繰り返しや条件式などいくつかの制御構造を記述でき、複雑な動作が可能になっています。

繰り返し——with_items

1つのtaskを何度も繰り返して実行したい場合、with_itemsを使います(**リスト26**)。

▼リスト21　templateモジュール実行結果

```
user www-data;
worker_processes  1;

error_log  /var/log/nginx/error.log;
pid        /var/run/nginx.pid;
```

▼リスト22　unarchiveモジュール

```
tasks:
  - name: tar.gzファイルを/tmpに展開する
    unarchive: src=foo.tar.gz dest=/tmp
  - name: 対象ホスト内にすでにあるzipファイルを展開する
    unarchive: src=/var/backup.zip dest=/home
copy=no
```

▼リスト23　aptモジュール

```
tasks:
  - name: 最新のnginxをインストールした状態にします
    apt: name=nginx state=latest
  - name: nginxをインストールしていない状態にします
    apt: name=nginx state=removed
  - name: nginxとpython-devをインストールした状態にし
ます
    apt: name={{ item }} state=installed
    with_items:
      - nginx
      - python-dev
```

▼リスト24　userモジュール

```
tasks:
  - name: zshを使うユーザ、appuserを追加します
    user: name=appuser
          shell=/bin/zsh
          password=$1$SomeSalt$Drh7s/vUcl5XnIZ/
Neglz1
```

▼リスト25　passwordの設定方法

```
# passlibをインストール
pip install passlib
# <your password>部分を書き換え、ハッシュを取得
python -c "from passlib.hash import sha512_crypt; ¥
          print sha512_crypt.encrypt('<your
password>')"
```

▼リスト26　繰り返し

```
tasks:
  - name: /opt/foo以下にbin, conf, logディレクトリ作成
    file: path=/opt/foo/{{ item }} state=directory
    with_items:
      - bin
      - conf
      - log
```

with_items以下に配列を指定し、配列内の変数を当てはめたい個所に{{ item }}のようにitemという変数を設定すると、その配列の数だけ指定したtaskを繰り返します。このitemという変数名は固定です。

基本的な繰り返しにはwith_itemsで対応できます。しかし、もっと複雑な繰り返し処理をしたい場合があります。辞書での繰り返しなどいくつかの繰り返し形式はサポートされていますが、YAML形式はプログラミング言語ほどの表現力を持たないため、複雑な繰り返しは表現できません。本当に必要であれば自分でモジュールを作ることも可能ですが、本稿では割愛します。

条件付き実行——when

taskにwhenを付けることで、そのtaskを実行する条件を設定できます。

リスト27の例では、jobという変数がwebの場合とdbの場合とで、異なるアプリケーションをインストールしています。whenを使うと、同じ1つのPlaybookで、対象ホストやグループに応じて異なる動作を行えます。

ansibleが取得する情報

ansible-playbook実行時に、

```
GATHERING FACTS ************************
ok: [test]
```

と表示されます。ここで、対象ホストの情報を収集しています。実際には内部でsetupモジュールを実行します。また、取得した情報をfactと呼びます。

setupモジュールで取得できる情報の一部を紹介すると、

・OS名
・ディストリビューション名
・ネットワークインターフェースやIPアドレス
・搭載メモリ量
・HDD使用量

などなど、かなり多くの情報を収集しています。これらはすべて変数に格納されるため、この情報を使っての条件分岐ができます。

たとえば、Ubuntuだったらaptモジュールを、CentOSだったらyumモジュールを使う場合には**リスト28**のように記述します。

この例で使用しているansible_distributionという変数がsetupモジュールによって自動的に取得されたディストリビューションの情報が入っている変数です。taskがwhenで指定した条件に合致せず実行されなかった場合には、ansible-playbookコマンドの実行結果にはskippedと表示されます。

他のファイルを読み込む——include

taskが増えていくとPlaybookの見通しが悪くなってきます。includeを使うことで、他のファイルからtaskを読み込めます。**リスト29**の例ではtasks/other_tasks.ymlというファイル(**リスト30**)を読み込んでいます。

includeを記述している側にtasksを記述してあるので、other_tasks.ymlの中ではtasks

▼リスト27　条件付き実行

```
tasks:
  - name: jobがwebだったらnginxを入れる
    apt: name=nginx state=installed
    when: job == "web"
  - name: jobがDBだったらmysql-clientを入れる
    apt: name=mysql-client state=installed
    when: job == "db"
```

▼リスト28　setupモジュールで取得した情報を使った条件分岐

```
tasks:
  - name: Ubuntuの場合
    apt: name=apache2 state=installed
    when: ansible_distribution == "ubuntu"
  - name: CentOSの場合
    yum: name=httpd state=installed
    when: ansible_distribution == "centos"
```

▼リスト29　includeの例

```
tasks:
  - include: tasks/other_tasks.yml
```

▼リスト30　tasks/other_tasks.ymlの中身

```
- name: nginxのインストール
  apt: name=nginx state=installed
```

▼リスト31　includeとwhenの組み合わせ

```
tasks:
  - name: Ubuntuの場合、main_ubuntu.yml を読み込む
    include: main_ubuntu.yml
    when: ansible_distribution == "ubuntu"
  - name: CentOSの場合、main_centos.yml を読み込む
    include: main_centos.yml
    when: ansible_distribution == "centos"
```

▼リスト32　Handlerの書き方

```
tasks:
  - name: ログディレクトリを作成
    file: path=/opt/log/nginx state=directory mode=0755 owner=nginx
    notify: reload nginx  # handlers内で定義したtaskのnameと同じ
  - name: nginx 設定ファイルを展開
    template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
    notify: reload nginx
handlers:  # インデントレベルに注意
  - name: reload nginx
    service: name=nginx state=reloaded
```

を宣言する必要はありません。先ほどのwhenと組み合わせると、条件分岐によるファイルの読み込みができます(**リスト31**)。

このように、ファイルを分割しておくと、構成の見通しが良くなります。

変更時だけ実行——Handler

設定ファイルを更新したら、再起動や再読み込みなどをしなければならない場合は多々あります。単純に変更するたびに毎回再起動するのではなく、更新したときにだけ、あるいは、複数ファイルを更新したときでも1回だけ再起動すると、効率が良くなります。そのようなときにHandlerを使います。Handlerはtasksと同じインデントレベルでhandlersと記述し、その下にtaskを記述していきます(**リスト32**)。

こう定義しておき、通常のtasksで、notifyを定義します。このnotifyに書く文字列は、handlersで定義したtaskのnameと同じ必要がある点に注意してください。上記例ではreload nginxです。

この例では、2つのtaskでreload nginxを呼び出しています。両taskで変更があった場合でもreload nginxが呼び出されるのは1回だけです。また、両方とも変更がなかった場合には、reload nginxは呼び出されません。

なお、notifyは複数個の設定もできます。その場合、設定した順番にHandlerが呼び出されます。

第2章のまとめ

駆け足でしたが、Ansibleが持つ機能を一通り紹介しました。ここまで説明したことを覚えれば、Ansibleの基本的な使い方はできると思います。とくにモジュールは非常に多く用意されているので、やりたいことが出てきたら、まず公式ドキュメントのモジュール紹介ページを検索してみてください。やりたいことそのままの機能を持つモジュールがすでに用意されている場合も多いです。SD

第3章 少しずつ積み重ねて理想の自動化環境を作ろう
より便利な使い方で複雑な手順を簡潔に

Author 若山 史郎（わかやま しろう）　ツキノワ㈱

ここまでで述べてきた基本的な使い方でも十分にAnsibleを使いこなせると思います。本章では、もう少し便利な使い方をご紹介します。

Playbookの機能

まとめて再利用――roles

第2章で解説したincludeは単一のファイルを読み込むだけでした。しかし、1つのtaskを実行するためには、変数や第2章で説明したtemplateモジュールで使用するファイルなどが別々に必要になる場合があります。

roleはその役割が必要とする情報をすべてそのディレクトリ内に格納し、再利用しやすくしたものです。

roleの作成

roleはrolesディレクトリ以下に作成します。このrolesという名前は固定です。roles以下には**図1**のようなディレクトリ構造を作成します。この例はcommonというroleを作成する場合です。

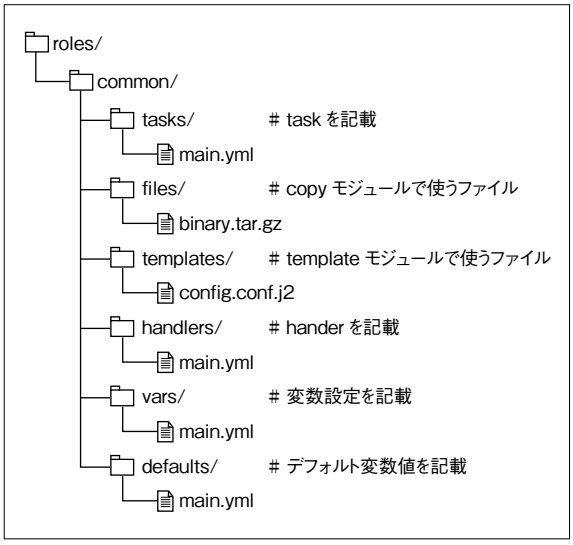
この中で必要なディレクトリは、tasksだけです。それ以外のディレクトリは必要に応じて作成して構いません。また、各ディレクトリ以下にあるmain.ymlは固定のファイル名となります。このファイルの中に各種の設定を書いていきます。

たとえば、roles/common/tasks/main.ymlには、**リスト1**のようにtaskを記載します。今までtasks以下に書いていた内容をそのまま記載するだけとなります。

この例ではcopyとtemplateモジュールを記載しています。copyの場合filesディレクトリ以下を、templateの場合templatesディレクトリ以下を自動的に検索します。

また、defaultsとvarsには両方とも変数を定義しますが、優先順位が違います。defaults以下で定義した変数は優先度が最低となるため、ほかからの上書きが容易です。rolesは基本的にdefaults以下で定義した変数を使うようにし、いつでも上書きできるようにしておくと問題が少なくなります。

▼図1　rolesディレクトリ構造



▼リスト1　roles/common/tasks/main.ymlの例

```
- name: test.confを配布
  copy: src=test.conf dest=/tmp/test.conf
- name: sample.appを配布
  template: src=sample.app.j2 dest=/tmp/sample.app
```

roleの使い方

roleはPlaybookの中で**リスト2**のように使います。

rolesでrole名を指定します。また、varsで設定した変数は、roleの中に引き継がれます。先ほど説明したとおり、同じ名前の変数は上書きされます。

roleにwhenやtagを指定することもできます。その場合は次のようにマップとして指定します。

```
roles:
  - { role: java, when: "mode == prod" }
  - { role: java, tags: ["java"] }
```

▼リスト2　roleの使い方

```
- hosts: web
  become: yes
  vars:
    username: test  # この変数はrole内に引き継がれます
  roles:
    - common
```

▼リスト3　delegate_toの使用例

```
hosts: webservers
 serial: 1
 tasks:
   - name: ロードバランサからインスタンスを取り外す
     ec2_elb: instance_id={{ ansible_ec2_instance_id }} state=absent
     delegate_to: 127.0.0.1
   - name: アプリを再起動
     service: name=fooapp state=restarted
   - name: ロードバランサにインスタンスを付ける
     ec2_elb: instance_id={{ ansible_ec2_instance_id }} state=present
     delegate_to: 127.0.0.1
```

roleの分け方

多くのroleを用意しておくと、適したroleを複数組み合わせることで、簡単に多様な環境に適したPlaybookができあがります。しかし、闇雲にroleを作成していてもうまくいかなくなります。

roleは単体で動作するように分割していくと再利用しやすくなります。task、変数やテンプレートファイルも、roleに必要な情報はすべてroleのディレクトリ以下に押し込められている状態です。また、“web”といった抽象的な分類ではなく、“nginx”や“apache”といった具体的なソフトウェアで分割すると後述するAnsible Galaxyとも組み合わせやすくなると思います。

ひとつ注意しておく必要がある点があります。それは変数名です。Playbookは変数の名前空間を1つしか持ちません。つまり、複数のroleでportなどの汎用的な名前をつけてしまうと重複してしまい、望みの結果が得られなくなってしまいます。この問題を避けるためには、少々冗長にはなりますがredis_portなどのようにrole名(例：redis)を変数名に付けておくのが良いでしょう。ただし、複数のroleにまたがるような変数を定義したい場合はこの限りではありません。

delegate_toとlocal_action

Ansibleには非常に数多くのモジュールがありますが、それらの中にはAWSのインスタンスを立ち上げるなどのクラウドサービスとの連係機能を持つモジュールがあります。このようなモジュールの場合、対象ホストで実行する必要はありません。むしろ、管理ホストでのみ実行する必要があります。

このような場合にはdelegate_toを使います。delegate_toはモジュールの実行を別のホストに移譲します。たとえば**リスト3**のようにdelegate_to: 127.0.0.1と指定することで、管理ホストで実行する指定となります。

この例ではec2_elbというAmazon ELBを扱うモジュールを2回呼んでいます。この2回は管理ホストで実行されます。一方、serviceモジュールは対象ホストで実行されます。

なお、delegate_toではなくlocal_action

という指定もできます。しかし、この場合モジュール名の指定が少し通常と異なりますので、少々わかりにくくなります。詳細は誌面の都合で割愛しますが、Ansibleのドキュメント[注1]などをご参照ください。

run_once

前述のdelegate_toは、対象ホストが複数台あった場合、その台数分taskが管理ホストで実行されます。しかし、多くの場合は1回だけで十分だったりします。そのような「1回だけ実行したい」場合はrun_onceをtaskに指定します(**リスト4**)。これで管理ホストで実行されるのが1回だけになります。

環境変数を設定する

environmentを指定することで、実行時の環境変数を設定できます(**リスト5**)。task単位での設定もできます(**リスト6**)。

ディレクトリ構造のベストプラクティス

Ansibleが公式で紹介しているディレクトリ構造のベストプラクティスを紹介します。**図2**のように、トップにPlaybookファイルとInventoryファイルを置き、roles以下に各roleを置きます。

site.ymlの中身は次のように、webservers.ymlとdbservers.ymlをincludeしているだけです。

```
- include: webservers.yml
- include: dbservers.yml
```

webservers.ymlはこうです。

```
- hosts: webservers
  roles:
    - common
    - webtier
```

対象はwebserversグループだけに絞られています。そして、tasksはこの中には記述せず、roleのみを定義しています。実際に行われるtaskはすべてroleの中に押し込めているため、Playbook自体は非常にシンプルとなります。

このような構成にしておき、webとdbを構築するには、次のように実行します。

注1) http://docs.ansible.com/ansible/playbooks_delegation.html

▼リスト4　run_onceの使用例

```
- hosts: webservers
  serial: 1
  tasks:
    - name: ロードバランサからインスタンスを取り外す
      ec2_elb: instance_id={{ ansible_ec2_
instance_id }} state=absent
      delegate_to: 127.0.0.1
      run_once: true
```

▼リスト5　environmentの使用例

```
- hosts: web
  environment:
    http_proxy: http://proxy.example.com:8080
  tasks:
    - name: proxyを使用してnginxをインストール
      apt: name=nginx state=installed
```

▼リスト6　environmentの使用例(task単位)

```
- hosts: web
  tasks:
    - name: proxyを使用してnginxをインストール
      apt: name=nginx state=installed
      environment:
        http_proxy: http://proxy.example.com:8080
```

▼図2　ディレクトリ構造のベストプラクティス

```
production          # 本番環境用 Inventory ファイル
staging             # テスト環境用 Inventory ファイル
site.yml            # すべてを行う Playbook
webservers.yml      # Web サーバに対する Playbook ファイル
dbservers.yml       # DB サーバに対する Playbook ファイル
group_vars/         # グループ変数を格納
    all.yml
    web.yml
host_vars/          # ホスト変数を格納
    web01.yml
    web02.yml
roles/              # 各種 Role
    common/
    web/
    db/
```

```
$ ansible-playbook -i production site.yml
```

このようにincludeとroleを使うことで、同じ内容を何度も書き加える必要なく、複雑な構成のサーバ群を構築できます。

注意していただきたい点は、必ずしもこのベストプラクティスに従う必要はないということです。この構成はあくまで1つの案であり、それぞれの事情にあった構成を作成するのが一番です。Ansibleはそのような柔軟性も持ち合わせています。

コマンドラインオプション

ansible-playbookコマンドには多くのコマンドラインオプションがあります。どれも有用ですが、その中のよく使う一部のオプションを紹介します。

-v、--verbose

-vをつけると、詳細な情報が表示されます。-vv、-vvvと増やしていくと、より詳細な情報が得られます。Ansibleで問題が生じるのはSSH接続とユーザの場合が多いので、-vvvを使うことが多いです。このオプションを付けて実行すると、**図3**のように接続するユーザ名が表示されます。

-t、--tag

tagはタスクごとに設定するタグです。次のように各taskに設定します。

```
tasks:
  - name: ユーザを追加
    user: name=appuser
    tags: init
```

このように記述しておくと、ansible-playbookコマンド実行時に-tを付け、そのtagを設定したtaskだけ実行できます。

```
$ ansible-playbook -i hosts web.yaml -t init
```

タグは複数設定することもできます。また、Ansible 1.9からalwaysというtagが導入されました。alwaysとtagsに設定しておくと、そのtaskはどんなタグを指定しても必ず実行されます。タグはincludeにも適用できますので、ほかのymlファイルを読み込むinclude文にtagsを設定しておくと、不必要な処理をまとめてスキップできて便利です。

-l、--limit

limitは、実行する対象ホストを絞り込みます。

```
$ ansible-playbook -i hosts web.yaml -l web
```

と指定すると、webという名前のホスト、あるいはwebグループに対してのみ実行します。この指定はかなり柔軟で、たとえばweb*と指定すると、webから始まるホスト名を指定します。

前述のtagとの使い分けをまとめると、

- **limit**：対象とするホストを絞り込む
- **tag**：対象とするタスクを絞り込む

ということになります。この2つを組み合わせて使うことで、特定の処理だけを特定のホストだけで実行できます。

--start-at-task

Playbookを作成しているときにエラーが起きた場合、もう一度最初から実行すると時間

▼図3　-vvvオプションを付けた実行例

```
GATHERING FACTS ***************************************
<web01.example.com> ESTABLISH CONNECTION FOR USER: shirou
<web01.example.com> REMOTE_MODULE setup
```

▼図4　ansibleコマンドの実行結果

```
web01.example.com | SUCCESS | rc=0 >>
16:57  up 2 days,  6:18, 1 users, load averages: 0.31, 0.33, 0.52

web01.example.com | SUCCESS | rc=0 >>
16:57  up 2 days,  6:18, 2 users, load averages: 0.08, 0.03, 0.05
```

がかかってしまう場合があります。先ほどのtagを使って実行するtaskを絞り込む方法もありますが、--start-at-task <タスクの名前>を指定すると、指定した名前を持つタスクからPlaybookが開始されます。試行錯誤しているときにとくに便利です。

ansibleコマンド

Ansibleをインストールすると、ansibleというコマンドも使えるようになります。ansible-playbookがPlaybookに基づいて多くの処理をこなすのに対し、ansibleは単体のモジュールを実行するだけとなります。

ansibleコマンドは、Playbookを用意してある定型的な処理ではなく、一時的なその場だけの処理にとても便利です。たとえば、

・全サーバのプロセスリストを見たい
・ログを手元に持ってきたい
・脆弱性が発見されたので、このパッケージだけ今すぐ更新したい

などなど、複数台のサーバに対して手動で実行するとなかなか大変な作業を1回で行えます。とくに大きな利点として、ansible-playbookで通常使用しているInventoryを使いますので、サーバリストを別途作成する必要はありません。

ansibleコマンドは次のように実行します。-mで使用するモジュールを、-aで引数を指定します。

```
$ ansible -i hosts -m shell -a "uptime" web
```

この例では、shellモジュールを使用して、uptimeをwebグループに対して実行しています。結果は**図4**のように表示されます。

-t outputsをつけると、outputsというディレクトリに各ホストごとに別々のファイルとして結果が保存されます。

ansible-galaxy

roleは、他者との共有も考慮された作りになっています。Ansibleの開発元が提供しているAnsible Galaxy[注2]というサービスでは、全世界の人とroleを共有できるようになっています。

Ansibleをインストールしたときに、ansible-galaxyというコマンドも同時にインストールされます。ここではRedisをインストールするroleを使う例を見てみましょう。

Ansible Galaxyのサイトから使えそうなroleを検索します。今回はDavidWittman.redisというroleを選んでみます(**図5**)。

図5のようにansible-galaxyコマンドを実行すると、roles/DavidWittman.redisというディレクトリが作成され、その中にtaskのファイルや設定ファイルのテンプレートなど、このroleが必要とするすべてのファイルがインストールされます。あとは通常のroleと同じように使えます。

また、ansible-galaxyはAnsible Galaxyのサイトからだけでなく、tar.gzや任意のgitリポジトリから取得することもできます。つまり、プライベートリポジトリに共有roleをたくさ

注2）https://galaxy.ansible.com/

▼図5　ansible-galaxyでredisのroleをインストール

```
$ mkdir -p roles   # rolesというディレクトリを作成する↵

$ ansible-galaxy install DavidWittman.redis -p roles↵
- downloading role 'redis', owned by DavidWittman
- downloading role from https://github.com/DavidWittman/ansible-redis/archive/1.0.2.tar.gz
- extracting DavidWittman.redis to roles/DavidWittman.redis
- DavidWittman.redis was installed successfully
```

ん作成して置いておき、必要なroleをansible-galaxyコマンドでダウンロードすることもできます。このようにすることで、別のプロジェクト間であってもroleをうまく共有し再利用できます。

Windows対応

残念ながら、Windowsを管理ホストとすることはできません。Cygwin[注3]上にAnsibleをインストールすることは可能ですが、多くの困難を伴います。一方、対象ホストとしてはWindowsを扱えます。

Windowsの場合はsshではなく、WinRMを使用して接続し、Pythonではなく PowerShellを利用してモジュールを実行します。ほかのモジュールとは互換性がないため、win_という接頭辞が各モジュールについています。その一部を紹介します。

- **win_user**：ユーザアカウントの作成・削除
- **win_service**：サービスの起動・停止
- **win_msi**：msiファイルからのインストール・削除
- **win_copy**：copyモジュールのWindows版。ただし、大きなファイルを送ると遅い

あわせて、本特集のAppendix 3を参照ください。

おわりに

第1章から第3章までで、Ansibleの使用方法をまとめました。Ansibleは構成管理以外にアプリのデプロイやインスタンスの立ち上げと、幅広く使えます。さらに、ansibleコマンドにより緊急対応などにも使えます。

事前に必要なソフトウェアはsshだけですので、実際上敷居はなく、導入が容易です。普段行っている手作業を少しずつAnsibleに置き換えていくと、手動に比べて間違いが少なくなります。いきなりすべてを自動化するのは大変ですので、少しずつ、少しずつ進めていくのが最終的には手戻りが少なく、成功する確率が高いように思います。さらに言うと、少しずつ育てたPlaybookは、バージョン管理システムに保存し、ほかの人からも見えるようにしていくと、ほか人から理解が得られやすくなります。

Ansibleは慣れるとシェルスクリプトを書くよりも早く、間違いが少なく、何度実行しても誰が実行しても問題がない環境ができあがります。また、YAMLで書かれているためプログラムに不慣れな方にでもわかりやすいうえ、教えやすくもあり、使ってもらう敷居が低くなっています。ぜひみなさまも使いこなして作業を効率化してください。SD

注3） Unix系OSで普及しているGNUプロジェクトによるツール群をWindows用に移植したもの。
https://www.cygwin.com/

A.1 便利なのに使われないAnsibleになるのを防ぐ ディレクトリ構成の熟考のススメ

Author 湖山 翔平（こやま しょうへい） ㈱リブセンス Twitter @sion_cojp

弊社では、インフラチームはAnsibleとChefを使い分けてOSが入ったサーバをメディア開発チームに提供し、サーバを受けとったメディア開発チームはAnsibleを使って最終セットアップをして利用、といったフローで作業を行っています。したがって、インフラチーム＋各メディアの開発チームそれぞれがAnsibleのリポジトリを保持し運用しています。

AnsibleとChefの比較

AnsibleとChefの違いは次のように考えています。

Chefのメリットとデメリット

▶Chefのメリット

❶可読性が高い
❷用意されているリソースが便利
❸構成がシンプル
❹ohai連携でサーバのデータが細かく取得でき、扱える
❺多様なサーバがあったら分岐しやすい

▶Chefのデメリット

❶初回導入に時間がかかる
❷chef-solo、chef-server、knife-soloなど覚えることが多い
❸client側にchef-dkのインストールが必要なので、初回のbootstrapに時間がかかる
❹knife-soloはroundsmanなどを入れないと、1コマンドで並列処理ができない
（ターミナルを2つ開くなどで並列処理はできる）

Ansibleのメリットとデメリット

▶Ansibleのメリット

❶シンプルで柔軟なディレクトリ構成
❷sshを叩いてるだけなので、ホスト側にソフトウェアなどは不要
❸格納されているスクリプトをssh越しに実行できる
❹並列処理

▶Ansibleのデメリット

❶Ansible独特な記法が多く、可読性が低い
（failed_when: result.rc not in [0, 1] など）
❷自由度が高い分、運用ルールを設けないと、日々複雑になっていく
❸オプションが多くてすぐ忘れる
❹シンプルな構成なので分岐に向いていない

以上よりインフラチームでは、

・OSセットアップはChefで管理／運用
・定常オペレーション、脆弱性対応、スポット作業はAnsibleで実施

と使い分けています。

開発チームでのAnsibleの利用

ここからは、筆者が所属している、ジョブセンスリンク開発チームのAnsibleの運用について紹介させていただきます。主な利用方法は次の4つです。

❶各ホストにMackerelの導入、設定変更
❷脆弱性対応などのスポット作業
❸インフラチームから渡されたOSに対して、アプリがデプロイできるまでの初期セットアップ
❹セットアップしたあとのserverspec実行

ansible.cfgは、**/etc/ansible/ansible.cfg**にシンボリックリンクを貼ってGitで管理しています。

```
$ more ansible.cfg
[defaults]
forks=100
host_key_checking = False
remote_port = 22
remote_user = hoge
ssh_args = -o ControlMaster=auto -o
ControlPersist=60s
ask_pass=True
```

Ansibleで最も悩まされるのがディレクトリ構成です。Ansible公式ドキュメントや他者のベストプラクティスを、理解せずそのまま利用して運用すると、大半が運用しづらくなり、「便利なのに使われないAnsible」ができあがるでしょう。

会社によってAnsibleでやりたいことが違うので、ベストプラクティスはあくまでも参考程度に留めておき、自社に合ったディレクトリ構成＋運用方法をしっかり議論し構築しましょう。弊社では**図1**のようなディレクトリ構成にしています。

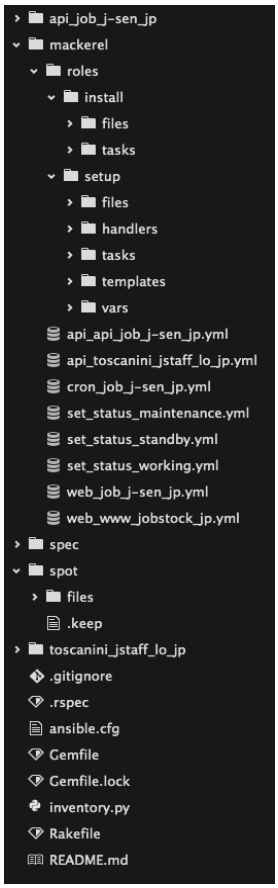
このようなディレクトリ構成を取った理由は次のとおりです。

❶基本はRoles構成。Roles構成のメリットは冪等性と役割の分担
❷1つのディレクトリでRoles構成を使うと、可読性が低くなる＋分岐が複雑になるので、サービスごとにディレクトリを分けてRoles構成を分離させたい
❸**spot**はスポット作業用。バージョンの更新などの冪等性は求めていないので、Roles構成から除外させる
❹**spot**で使われるファイル群は、すべて**files**に格納し、かつ相対パスで記述

Roles構成というのは、**表1**のようなディレクトリ構成でそれぞれの**main.yml**が読み込まれるようになっています。

よく使わないRolesのディレクトリを.keepファイルで保持するのを見かけますが、なるべくシンプルな構成を心がけるため、使わないディレクトリは削除しましょう。

▼図1　ディレクトリ構成



▼表1　Roles構成のディレクトリ構造

ディレクトリ	内容
files/	ホストに配布する、変数を持たないファイルを保持
templates/	テンプレートファイルを保持。変数を使って動的に変化させることができる。jinja2というテンプレート言語で構成
tasks/	タスク（メインの処理）実行
vars/	変数を定義
handlers/	notifyで定義されたハンドラ処理を実行
defaults/	変数を定義。一番優先度が低い（弊社ではvars/で事足りるため、使っていない）
meta/	roleの依存関係を定義

▼リスト1　Inventoryファイル

```
### Mackerel設定初回投入
$ vim hosts/test
192.168.1.1
$ ansible-playbook -i hosts/test mackerel/web_job_j-sen_jp.yml

### Dynamic InventoryでMackerelの設定更新
# ホスト一覧を調べる
# inventor.py ←http://gihyo.jp/magazine/SD/archive/2016/201601/support からダウンロードしてください
$ python inventory.py --list
# 実行結果サンプル： sample_inventory_lists.log
                    ↑http://gihyo.jp/magazine/SD/archive/2016/201601/support からダウンロードしてください
# ホストのIPを調べる
$ python inventory.py --host host1.example.com
# 実行結果サンプル： sample_host_info.log
                    ↑http://gihyo.jp/magazine/SD/archive/2016/201601/support からダウンロードしてください
# 実行
$ ansible-playbook inventory.py mackerel/web_job_j-sen_jp.yml

# host1.example.comだけに実行させたいとき
$ ansible-playbook inventory.py -l host1.example.com mackerel/web_job_j-sen_jp.yml
```

各ホストの情報取得については、Dynamic Inventory + Mackerelを利用しています。DynamicInventoryのメリットは、現在のホストステータスをAnsibleDynamicInventoryで定義されたJSON形式で出力させれば、それをInventoryファイルとして扱えるということです。ホスト情報を手動で管理しなくていいというのは、ホスト間違いによる作業ミス、という懸念が除外されるので、とても大事なことです。ただ、Mackerelの初回導入時のみ**hosts/**に**リスト1**のInventoryファイルを作り、ファイル指定でAnsibleを実行しています。

mackerel/web_job_j-sen_jp.ymlは**リスト2**のようになっています。

web_job_j-sen_jpに対して、roleにあるinstallとsetupの**tasks/main.yml**を実行しています。installがどのようになっているかは本誌のサポートページ[注1]からダウンロードして参考にしてください。

ymlの書き方のポイントは、いかに冪等性を担保するかです。必要であればwhenステートメントで分岐し、分岐しなくてよい部分はなるべく分岐しないように工夫しましょう。そして何度打っても同じ結果になるようにしましょう。

▼リスト2　mackerel/web_job_j-sen_jp.yml

```
- hosts: web_job_j-sen_jp
  user: hoge
  sudo: yes
  gather_facts: no
  roles:
    - { role: install }
    - { role: setup }
  vars:
    ### Mackerelのroleを指定する場合は以下を有効化
    mackerel_agent_roles:
      - "web:job_j-sen_jp"
      - "job_j-sen_jp:web"

    ### Mackerelで監視したい項目ファイルを記述
    mackerel_conf_file:
      - "web_job_j-sen_jp.conf"
```

最後に

以上、弊社のAnsible利用例でした。皆様もAnsibleを使って、オペレーションの自動化を目指してください！SD

注1）http://gihyo.jp/magazine/SD/archive/2016/201601/support

A.2 GitHubでの管理を考える Playbookの置き方とssh秘密鍵の暗号化

Author 上野 晶鋭（うえの あきと）　㈱IDCフロンティア

本稿では、筆者が執筆時現在かかわっている自社開発中のプロジェクトにおいて、なぜAnsibleを導入したのかという理由と、導入して得られた知見や感じたことを紹介します。

どんなプロジェクト?

筆者がかかわっているプロジェクトでは、「SSL終端を行うロードバランサを提供するサービス」を実現するJobQueueシステムを開発しています。

システムの構成は大まかに**図1**のようになっています。各コンポーネント（API、Controllerなど）は冗長化されていますので、サーバ台数としては1セットで24台の構成となっています。プロジェクトでは4セット利用して開発を行っているので、全体のサーバ台数は96台になります。この96台のサーバの構築や設定変更を簡単に行いたくてAnsibleを導入しました。

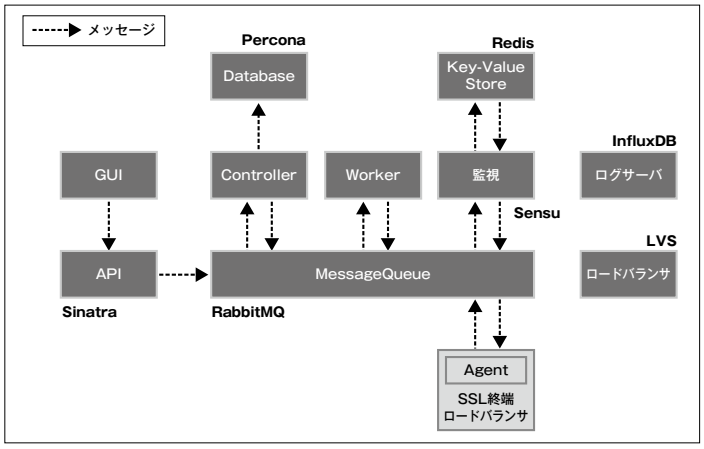
なぜAnsible?

なぜ数ある構成管理ツール（Chef、Puppet、最近だとItamaeなど）の中でAnsibleを選んだのか。理由はAnsibleの特徴である次の3点のためです。

・エージェントレス
・設定ファイルのように記載できる
・モジュール内で冪等性（べきとうせい）が確保されている

これらは第1章から第3章までの記事で詳しく述べられていますので、ここからは筆者のチームで行っているAnsibleの運用実例として、Playbookの管理方法とAnsibleのコマンドで行える暗号化の利用方法について紹介します。

▼図1　Ansible適用対象システム



[Ansible Tip1] Playbookの管理方法

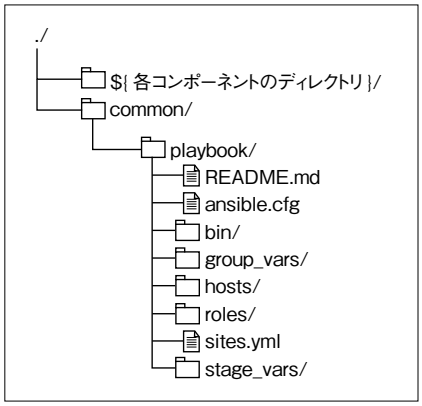
いつも思うのですが、プロジェクトを進めるときにアプリケーション（ソースコード）以外のものを「どこで、どんな構成で、どのように運用していくか」を迷いませんか？　筆者以外の人でも悩んでいる人がいるのかなと思い、ここに紹介させていただきます。

どこで

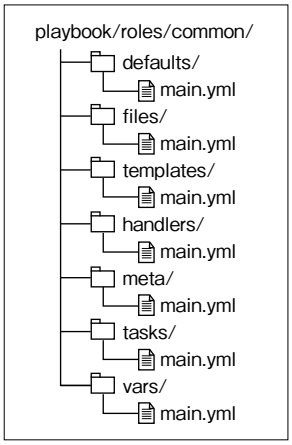
アプリケーションと同じGitHubのリポジトリで管理しています。リポジトリのトップディレクトリからPlaybookがあるディレクトリまでを図示すると**図2**のようになっています。

${各コンポーネントのディレクトリ}にはアプリケーションのコンポーネント単位のソースコー

▼図2　GitHubディレクトリ構成



▼図3　Playbookディレクトリ構成（common）



ドが格納されています。AnsibleのPlaybookはソースコードとは別のディレクリ（common/playbook）で管理しています。common配下にはPlaybook以外にもDeploy用のスクリプトなども配置しています。

どんな構成で

Playbookの実体は**図2**のplaybook/rolesディレクトリに**図3**の構成で配置しています。

Defaultsでデフォルト値を管理して、filesで対象ホストに転送するファイルを管理します。Templatesでは対象ホストに関連する変数置換が必要なファイルを管理して、handlersではPlaybookの最後に実行される処理を記載したファイルを管理します。Tasks配下でメインの処理を記載したファイルを管理して、vars配下でroleごとの変数を管理しています。

roleによって必要のないディレクトリは削除していますが、基本的にすべてのroleが上記のディレクトリ構成に従っています。

どのように運用しているか

ルールを決めてもなかなか守れないのが人の性ですよね。そこで筆者のチームでは、チームのメンバーが導入したadd_roleコマンドを**図2**のcommon/playbook/binに配置して運用しています。新しくroleを追加するときは、次のようにコマンドを実行することで**図3**のディレクトリ構成のroleを作成します。

```
$ ./bin/add_role ${new_role}
```

[Ansible Tip2]公開したくない情報の取り扱い

公開したくない情報をGitHubにアップロードするなという話でもありますが、管理先が複数になると管理が煩雑になりがちです。

筆者がかかわっている今の開発ではクラスタリングを組む際にssh接続が必要な個所があり、sshの秘密鍵を管理する必要があります。万が一のことを考えて鍵をそのままGitHubにアップロードするのは抵抗があります。そうは言ってもほかで管理したくない……。

そこで利用しているのが、ansible-vault[注1]コマンドです。ansible-vaultではパスフレーズを用いて対象ファイルの暗号化／復号をコマンド一発でできます。

運用としては次のように行っています。

1. GitHubからPlaybookをpull
2. ansible.cfgのvault_password_fileに指定したファイルの作成、パスフレーズの入力

注1）http://docs.ansible.com/ansible/playbooks_vault.html

例：vault_password_file に password.txt と指定し、password.txt にはパスフレーズのみを入力して保存

3. ansible-vault で復号
   $ ansible-vault decrypt ${target_file}
   復号した後はPlaybookを流したり、修正が必要の場合はPlaybookを修正する
4. ansible-vault で暗号化
   暗号化対象ファイルを編集した場合は次のコマンドで暗号化した後にGitHubにpush
   $ ansible-vault encrypt ${target_file}

結局ansible-vaultの暗号化パスフレーズを別管理しているじゃんって突っ込まれそうですが、sshの秘密鍵を別管理するよりは良いかなと思って運用しています。より良い管理方法があれば教えてください！

まとめ

まだシステムの開発中で、Ansibleは試験的に導入している段階のため正確には評価できていません（この点については別の機会があれば紹介します）。ただ「Ansible Tip1」で述べたように、異なるroleで同じディレクトリ構成をとっているおかげで、開発メンバー間の共有は楽にできていますし、ansible-vaultを利用して隠したい情報を簡単に暗号化／復号してGitHub上で管理することができています。

今回ご紹介したTipsが読者の皆さんのお役に立ち、Ansibleに興味を持っていただければ幸いです。SD

A.3 エージェントレスでWindowsのデプロイ自動化 AnsibleからWindowsを操作する

Author 廣川 英寿（ひろかわ ひでとし）　㈱リアルグローブ

Ansibleの強みの1つにネイティブでのWindows対応があります。筆者の所属する㈱リアルグローブでも、自社システムはすべてLinuxなのですが、Windowsサーバをメインで使用されているお客様も多く、そんな方のシステム運用自動化支援を実施するにあたり、対応当初からLinuxだけではなくWindowsのデプロイ自動化にもAnsibleを活用しています。ここでは、上記のような業務の中で得た経験をもとに、意外とハマりやすいAnsibleからWindowsに接続するための手順や、実際Ansibleでどのくらいのことができるのかを紹介します。

AnsibleからWindowsを操作するために

Windows側の準備

Ansibleと言えばエージェントレスが売りです。もちろん、Windowsを操作する場合でもそれは変わりません。ただし、WindowsにはLinux/UNIX系OSのようにSSHサーバが備わっていないため、AnsibleではWindowsの遠隔操作にWinRMを採用しています。

WinRMとは、Microsoft純正のWindowsリモート管理システムで、PowerShellなどと一緒にWindows Management Framework（以下WMF）に含まれているパッケージです。また、AnsibleからWindowsを操作するためには、Windows側にWMF 3.0以上がインストールされている必要がありますので、Windows 8、Windows Server 2012未満の場合は、あらかじめWMF 3.0以上をインストールしておく必要があります。

▼表1　Windowsのバージョンとインストール可能なWMFバージョン

WMFのバージョン	Windowsのバージョン
3.0インストール可	Windows Server 2008 Windows Server 2008 R2 Windows 7
3.0プリインストール済	Windows Server 2012 Windows 8
4.0プリインストール済	Windows Server 2012 R2 Windows 8.1
5.0プリインストール済	Windows 10 Windows Server 2016（予定）

なお、WMF 3.0を使う場合には、hotfix[注1]を忘れず適用してください。これが適用されていないとAnsible実行中、WinRM側で予期せぬタイミングのメモリエラーが発生してしまいます。

WMF 3.0以上がインストールされていれば、あとの設定は簡単です。Ansibleが公式に提供しているWindows設定用スクリプトをダウンロードして、PowerShellから実行してください。このスクリプトでは、WinRMサービス有効化、WinRMでHTTPS接続を受け付けるためのSSL証明書発行、ファイアウォール設定、PowerShellリモート実行許可などの処理をまとめて実行してくれます。

Administratorユーザでログインしている場合のPowerShell上からの設定スクリプトのダウンロード、実行手順は**図1**のようになります。

スクリプトを実行した結果、`Ok.`と出力されれば問題なく設定が完了しています。

これでWindows側の準備は整いました。IaaSや仮想環境を使っている場合、マシン立上げ後すぐにAnsibleを使い始められるように、ここまで設定済みの状態をイメージ化しておくとよいでしょう。最後に、WinRMのHTTPS通信用に、5986番ポートにAnsibleから接続できるように外部ファイアウォールなどを設定すれば、あとはAnsibleからの接続を待つばかりです。

Ansible側の準備

さて、次はAnsible側の準備です。Ansible自

注1）http://support.microsoft.com/kb/2842230

▼図1　PowerShellからスクリプトをダウンロードし実行する

```
設定スクリプトをダウンロード
PS C:¥Users¥Administrator> Invoke-WebRequest -Uri https://raw.githubusercontent.com/
ansible/ansible/devel/examples/scripts/ConfigureRemotingForAnsible.ps1 -OutFile
.¥Desktop¥ConfigureRemotingForAnsible.ps1
ダウンロードしたスクリプトを実行
PS C:¥Users¥Administrator> powershell -ExecutionPolicy RemoteSigned .¥Desktop¥
ConfigureRemotingForAnsible.ps1
```

▼リスト2　callback_plugins/fix-ssl.py

```
import ssl
if hasattr(ssl, '_create_default_https_context') and hasattr(ssl, '_create_unverified_context'):
    ssl._create_default_https_context = ssl._create_unverified_context

class CallbackModule(object):
    pass
```

体のインストールやpipの導入は済んでいるものとして、まずはWinRMをPythonから操作するためのライブラリ**pywinrm**をインストールします。

```
$ pip install pywinrm
```

次に、接続情報を記述したhostsファイルを用意しましょう(**リスト1**)。

そして、少し厄介なのがSSL自己証明書を使うための対応です(Python2.7.9未満では対応不要)。**リスト2、3**のようなファイルを用意してください。

最終的に、**図2**のようなファイルツリーになっていればOKです。

これで準備が整いました。それでは、実際にAnsibleからWindowsに接続ができるようになったか、**win_ping**モジュールを使って確認してみましょう。

```
$ ansible windows -i hosts -m win_ping
# 以下のように出力されれば成功
<<Windowsホスト名>> | success >> {
    "changed": false,
    "ping": "pong"
}
```

AnsibleからWindowsにできること

Ansibleには組み込みで使える便利なWindows専用モジュールが用意されています。

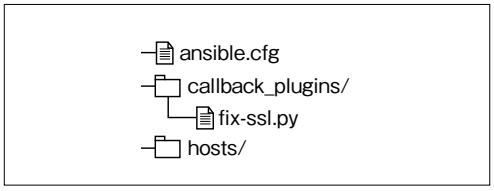
▼リスト1　hostsファイルの内容

```
[windows]
<<Windowsホスト名>>

[windows:vars]
ansible_ssh_user=<<ログインユーザー名>>
ansible_ssh_pass=<<ログインパスワード>>
ansible_connection=winrm
```

▼リスト3　ansible.cfg

```
[defaults]
bin_ansible_callbacks=True
callback_plugins = ./callback_plugins
```

▼図2　ファイルツリー



数の勝負ではLinux/UNIX用の充実には遠く及ばないものの、最新安定版1.9.4(2015年11月12日現在)では13種類であるWindows用モジュールが、現在ベータフェーズで近日リリース予定の2.0では29種類にまで増えており、Windows対応が急速に進んでいることがうかがえます。

すぐに使い始められるモジュール

実のところ、1.9.4の時点でも実用的なモジュールはかなり出揃っており、たとえば、基本的なファ

イル操作にURLからのファイルダウンロード、テンプレートの展開配置、ユーザやグループの編集、サービス起動管理、Windows Updateの操作、Windowsの機能のインストール、chocolateyによるパッケージ管理などは、今すぐに使い始めることができます。

そしてさらに2.0では、レジストリ編集、タスクスケジューラ管理、ファイアウォール操作、IIS管理、ファイルのACL管理などが増え、Windowsプロビジョニングに必要な大部分の操作がAnsible組み込みモジュールだけで完結できるようになったなという印象です。

ただし、一部利用時に注意が必要なモジュールもあります。その代表例が**win_copy**です。**win_copy**では通常の**copy**モジュールと同様にAnsibleローカルからリモートへのファイルコピーが実施できますが、WinRMの仕様上の制約から実用上1MB未満のファイルにしか使えません。3MBを超えるような場合はエラーになるばかりか、Windows側でWinRMの再起動が必要になってしまうような場合もあります。エラーにならないような場合でも、ものすごく転送が遅くなってしまうので、Windowsに大きなファイルを送りたい場合は、Windowsから参照可能なURLで公開してから**win_get_url**にダウンロードさせるといった工夫が必要です。

PowerShellとの連携も可能

このように現時点では利用に一部注意が必要なものもありますが、Windows用モジュールはかなり充実してきています。それだけではなく、もちろん**raw**モジュールでの生コマンド実行や**script**モジュールでのPowerShellスクリプト実行を組み合わせれば、操作の自由度に制限はありません。また、Windowsモジュール用のPowerShellヘルパーが用意されているので、PowerShellを触ったことがある人であれば自作モジュールの作成も簡単です。

その気になれば、UIAutomationと組み合わせてGUIアプリケーションの操作を自動化したり、レジストリ内に保存されたバイナリを書き換えてシェルからは編集できないWindowsの設定を更新したりといったところまで自動化できてしまいます。とくに後者は一度GUIから手動設定した際の値をリストアすれば状態を再現できるので、筆者はデスクトップ環境設定の自動化に多用しています。突き詰めるとAnsibleで自動化できない範囲などないのです！

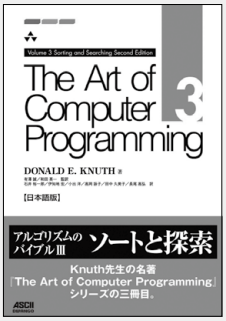
GUI自動化まではしていませんが、2.0で追加されたモジュールの使用例も含んだ簡単なWindowsサーバ・セットアップ用のPlaybookサンプル[注2]を公開してありますので、参考にしてください。

まとめ

AnsibleにWindowsサポートが付いた2014年8月の1.7のリリースから一年強の月日が経ちました。多くの操作を、結局は自前PowerShellスクリプトで実装しなければならなかった当初と比べると、モジュールも出揃い、安定性も上がり、Windowsに対してもAnsibleの美点である「シンプルさ」「冪等(べきとう)性」といったポイントを気軽に享受できるようになってきました。

本誌の読者さんは、筆者も含めてLinux使いの方がメインでしょうが、ときにはWindowsを相手にしなければならないという方も多いかと思います。Ansibleを使えば、WindowサーバとLinuxサーバを組み合わせたシステムでもワンストップ、ワンPlaybookでデプロイ可能になります。みなさん、ぜひともWindowsへのデプロイにもAnsibleを導入し、快適な運用ライフをお送りください！ SD

注2）https://github.com/h-hirokawa/ansible-windows-sample



The Art of Computer Programming Volume 3

Donald E. Knuth 著／有澤 誠、和田 英一 監訳／石井 裕一郎、ほか 訳
B5判／760ページ
4,800円＋税
ドワンゴ
ISBN＝978-4-04-869431-5

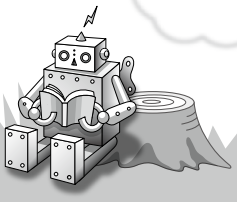
クヌース博士によるコンピュータアルゴリズムのシリーズのうち、第5章「ソート」、6章「探索」をまとめた1冊である。5章は大きく内部ソートと外部ソートの節に分けられる。前者はソートするレコードの数が十分に小さく、全処理をメモリ上で実行できる場合。後者は処理が遅い周辺記憶装置（テープ、ドラム、ディスク）において、ソートアルゴリズムが要求する素早い反応ができるようにデータ構造を調整する場合だ。6章で扱うのは、メモリ上から必要な情報のみを集めるためのアルゴリズム。逐次探索、キー比較による探索（二分木、バランス木）、デジタル探索（ハッシュ、副キー）などが学べる。余談ではあるが、博士はこの第3巻の第2版の制作中、組版システム「Tex」とフォント作成システム「METAFONT」を完成させたそうだ。

Apache Spark入門

株式会社NTTデータ、猿田 浩輔、土橋 昌、吉田 耕陽、佐々木 徹、都築 正宜 著／下垣 徹 監修
B5変形判／320ページ
3,200円＋税
翔泳社
ISBN＝978-4-7981-4266-1

Apach SparkはHadoopのエコシステムの上で動く、OSSの並列分散処理基盤。本書は、Sparkに初めて触れるエンジニアに向けて書かれた入門書であり、データ処理のための並列分散処理アプリを組み立てられるようになることを目標としている。Sparkは、RDDというイミュータブル（不変）・遅延評価という性質のデータ構造を持ち、「RDDを加工して新たなRDDを生成し、これを繰り返すことで目的の結果を得る」という動作モデルに基づいている。本書前半では、そういったSparkの基本、動作原理について解説している。本書後半では実践編として、基本的なAPI、Spark SQL、Spark Streaming、MLibの使い方について説明している。それぞれ具体的なデータを挙げながら解説しているので、実際の使い道を考えながら学べるだろう。

SD BOOK REVIEW



Android Wearアプリ開発入門

神原 健一 著
B5変形判／192ページ
2,580円＋税
技術評論社
ISBN＝978-4-7741-7749-6

腕時計向けのアプリを作ろうとしたとき、どうやってスマホアプリからの通知を受け取るのか、利用したいセンサーが腕時計側にない場合はどう対処するのか、円形の画面と四角形の画面があるが、UIデザインはどう切り替えるのかといったことは実装に欠かせない知識だ。本書はAndroidスマホアプリの開発経験者を対象に、これら腕時計型アプリ開発に必要な要点を集中して習得できるようコンパクトにまとめている。また、Android Studioの環境構築や効率を上げるためのTipsは、Eclipseからの移行にも役立つだろう。ハードウェア側の機能やデザインバリエーションは今後充実していくだろうが、スマートウォッチが普及するには何よりソフトウェアの力が必要だ。発展途上のいまこそ、新しい市場に挑戦してみてはどうだろうか。

ITエンジニアのための機械学習理論入門

中井 悦司 著
A5判／256ページ
2,580円＋税
技術評論社
ISBN＝978-4-7741-7698-7

機械学習のライブラリや優秀なツールが簡単に手に入るようになった昨今、参入へのハードルが低いと考えている方も多いのではないだろうか。否、全然そんなことはない、と本書を読んだ方は感じるだろう。本書を読み解くうえで必要とされるのは、理系大学1年～2年程度の高等数学だ。偏微分、重積分、確率統計、線形代数など、基本的に知っていなければならないことが多い。難解で厳しいと感じるだろう。しかし、本書のよいところは、計算過程をいっさい省いていないことだ。つまり、数式をじっくりにらみながら、式の変形をトレースできるのだ。この本は、難解な理論の手の内を、つまびらかに明かしてくれる家庭教師みたいなものだ。Pythonの例題コードもあるので、PCを傍らに置いて冬休みにじっくりトライしてほしい。

短期集中連載

開発に効く数字の測り方

クラウド時代のWebサービス負荷試験再入門

Author 仲川 樽八(なかがわ たるはち) ㈱ゆめみ Twitter@tarupachi

第2回

負荷試験と負荷試験ツール

はじめに

前回の記事で、クラウドの登場する前を「オンプレミス時代」、クラウド登場後を「クラウド時代」と定義し、とくにクラウド時代における負荷試験の概要やアンチパターンを説明しました。今回は負荷試験を行うためのツールの紹介と負荷試験実施にあたっての全体的な考え方を紹介します。

負荷試験ツールに共通の考え方

負荷試験にかかわる用語の説明

今回説明するものを含め、ほぼすべての負荷試験ツールには次の共通する概念があります。

- クライアント(HTTPのリクエストを同時に1つだけ実行できる単位)
- クライアントの同時起動数(負荷試験ツールから見て並列で発行するリクエスト数)
- RampUp期間(クライアントがすべて起動するまでの時間)
- シナリオ実行回数(または実行時間)
- スループット(単位時間あたりに処理されたリクエスト数)
- レイテンシ(負荷試験ツールからみたリクエストの処理時間)

負荷試験ツールからのリクエストの大まかなの流れは①～⑧のとおりです(**図1**)。

①リクエストが負荷試験攻撃サーバ上で生成される(同時起動数の設定まで)
②リクエストはHTTPリクエストとしてネットワーク上を移動
③リクエストはロードバランサによりWebサーバに振り分けられる
④リクエストがHTTPリクエストを持ってWebサーバに到達する
⑤Webサーバによって生成された、HTTPレスポンスを返答する
⑥レスポンスはロードバランサを経由して外に出る
⑦レスポンスはネットワーク上に戻る
⑧HTTPレスポンスを負荷試験攻撃サーバが受け取る→シナリオの実行回数が完了するまで①へ戻って繰り返す

負荷試験ツールで観測されるレイテンシはサーバの応答速度とは異なる

負荷試験ツールでのレイテンシは、ネットワークを転送される間に生じる遅延やSSLのデコードにかかわる遅延を含んだ数値です。サーバがいくら高速に応答しても、それが反映されているとは限りません。

各サーバの実際の応答速度はサーバ上でログを出力したり、ロードバランサで観測されるレイテンシを見る必要があります。

クライアントの同時起動数とサーバで処理される同時接続数は異なる

「負荷試験ツールから作成したリクエスト」の多くはネットワーク上や攻撃サーバ上にあり、実際に負荷をかける対象のサーバ上で処理中の

リクエストは全体の一部でしかありません。

とくにネットワークのレイテンシが大きいときには、負荷試験ツールで設定したクライアントの同時起動数と比較して、実際にサーバが処理中になる割合が小さくなりますので、攻撃サーバではクライアントの同時起動数を上げる必要があります。しかし、それは攻撃サーバ側の負荷となり、試験結果を不安定にする要因となります。

クライアントは前のリクエストが完了しないと次のリクエストを発行しない

サーバ上や、ネットワーク上のどこかでリクエストの応答が停滞すると、全体のスループットを大きく下げる要因となりますが、これは負荷試験特有の挙動であり、実際の挙動とは異なります。

負荷試験ツールによる負荷試験と実際のユーザアクセスの違い

負荷試験を行うにあたり、各種ツールを使うことで得られる試験結果と実際のユーザアクセスは異なります。そのことを理解しておくことが重要です。

リクエスト元サーバの台数、ネットワークの違い

負荷試験環境では、攻撃サーバの数は1台～数台の範囲です。しかし公開環境においてはアクセスするユーザ数分だけのリクエスト元が存

COLUMN 高負荷をかけたいときに攻撃サーバ上で変更しておきたいパラメータ

数百リクエスト／秒程度ならば問題ありませんが、それ以上の負荷を継続的にかけたい場合には、攻撃サーバでは次のパラメータを変更しておきます。そうしないと、より高い負荷をかけるために攻撃サーバのスケールアップを実施した場合であっても、途中でリソースが枯渇し、エラーが発生することがあります。

例）Linuxの場合

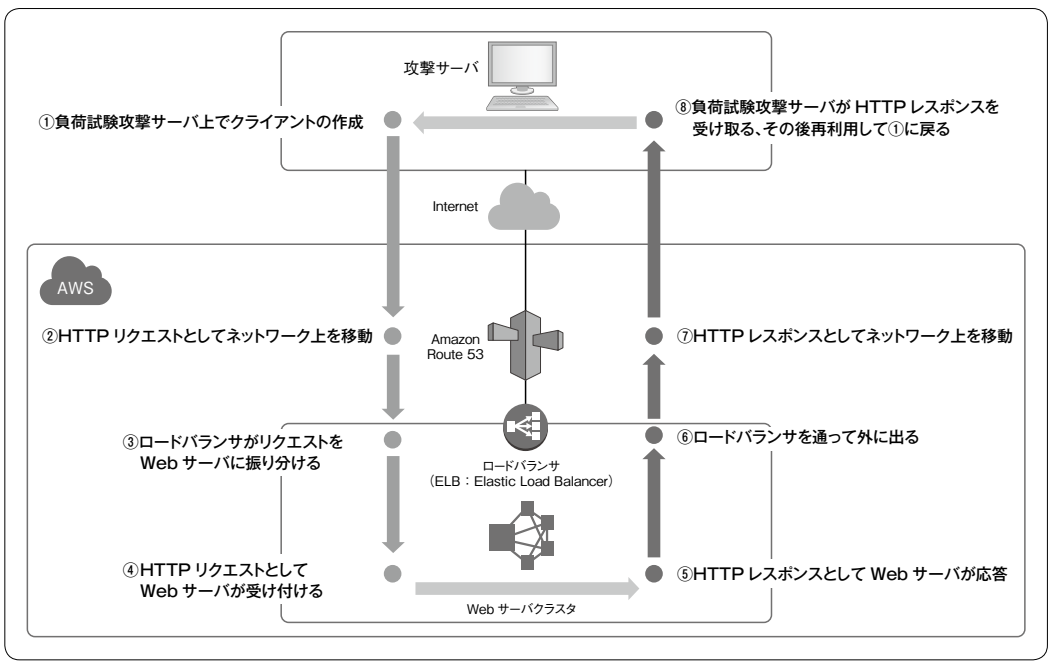
・ファイルディスクリプタ上限を上げる

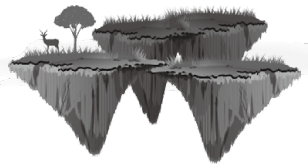
```
ulimit -n 65535
```

・tcp接続の上限数を調整する（/etc/sysctl.conf）

```
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 30
```

▼図1　クライアントが実際に攻撃をするイメージ





COLUMN

負荷試験は卓球のラリーである

筆者は、負荷試験の動作イメージを卓球のラリー（打ち合い）のようにとらえています（**図A**）。こう考えると、先ほどの「負荷試験で重要な3点」をイメージしやすくなります。

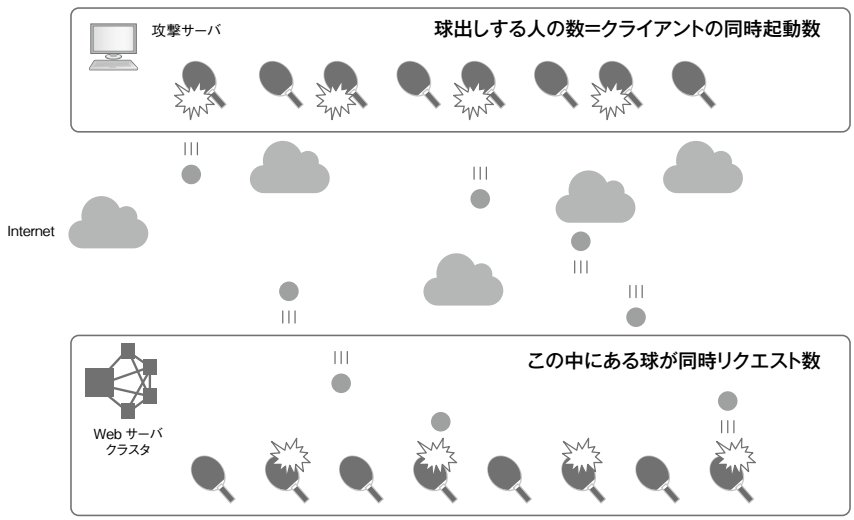
- 負荷試験ツールで観測されるレイテンシはサーバの応答速度とは異なる
- クライアントの同時起動数とサーバで処理される同時接続数は異なる
- クライアントは前のリクエストが完了しないと次のリクエストを発行しない

また、負荷試験サーバと対象システム間でより高速なラリーを行うためには負荷試験サーバをできるだけ対象システムに近づけて配置する必要があることがイメージできます。

負荷試験用語を卓球に当てはめて考えると次のようになります。

- クライアント（攻撃サーバ上で卓球をする人）
- クライアントの同時起動数（攻撃サーバ上で卓球をする人の人数）
- RampUp期間（クライアントを全員追加するまでの時間）
- シナリオ実行回数（ラリー回数またはラリー実行時間）
- スループット（単位時間あたりのラリー回数）
- レイテンシ（打った球が返ってくるまでの時間）

▼図A　卓球のラリーでたとえる負荷試験



在します。そのため、SSLを利用したサイトの場合、負荷試験環境においてはSSL接続の確立およびデコードのための負荷が攻撃サーバに集中してしまい試験ができません。加えてSSL接続をしない場合でも、httpリクエストのたびに通信が切断／再接続されるので、攻撃サーバにとって過剰な負荷となります。システムに効率的に負荷をかけるためには、Keep Aliveについても有効な状態で負荷試験をする必要があります（**図2**）。

また、同様にネットワークについても、負荷試験環境においては攻撃サーバにトラフィックが集中しますが、公開環境では分散します。そのため、攻撃サーバの性能がいくら高くても、ネットワークトラフィックが十分でない場合には試験になりません。また、環境によっては同一のIPからの連続アクセスを遮断する機構などが働いていることもありますので注意が必要です。

リクエスト先エンドポイントの違い

リクエスト先のエンドポイントのサーバが、負荷試験環境では攻撃サーバごとに一定時間

DNSキャッシュされてしまうことがあります。そのため、エンドポイントのサーバが自動的なスケールアウトに対応していたとしても、本来の性能を発揮できないことがあります(**図3**)。ただし、公開環境ではリクエスト元が分散していますので、この問題は発生しません。試験をするときは単一障害点となり得るエンドポイントを外すことで回避します。

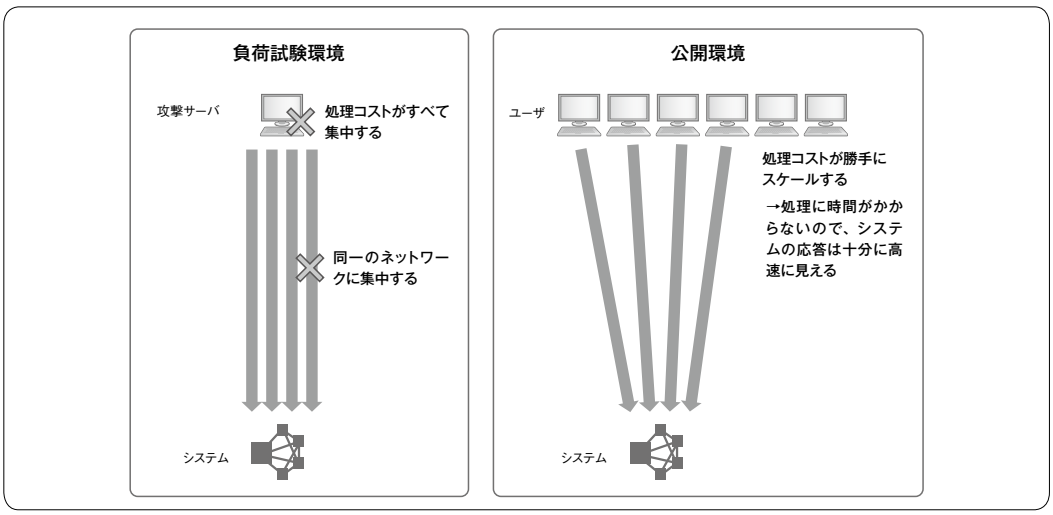
同時リクエスト数の違い

負荷試験環境では、攻撃サーバからのリクエストは、その結果が攻撃サーバに戻ってくることを待って、はじめて次のリクエストが投げられます。このためシステムの応答速度がいくら遅くても、あらかじめ指定したクライアント数のリクエストしか同時に発生しません(**図4**)。

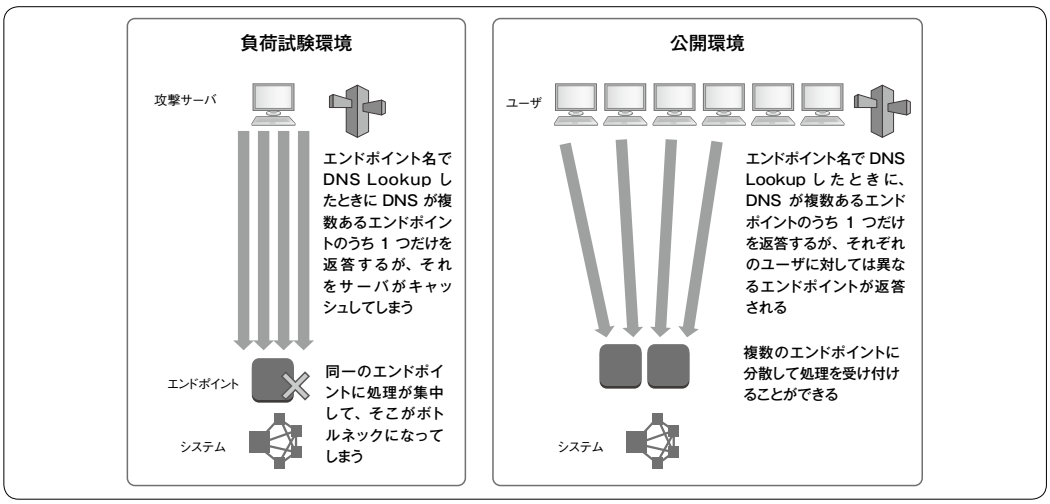
しかし、公開環境では特定のユーザの応答に時間がかかり、それらのリクエストの返答を待っている間であっても、ほかのユーザが新たに利用を開始し、別のリクエストをどんどん投げてしまいます[注1]。結果として、システムの応答時間が遅いと処理しなければならない同時リクエ

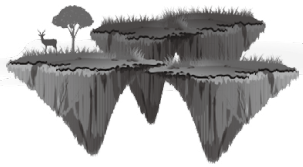
注1) コラムで説明した卓球の例を思い出してください。

▼図2 負荷試験環境と公開環境におけるサーバ台数・ネットワークの違い



▼図3 負荷試験環境と公開環境におけるリクエスト先のエンドポイントの違い





スト数はどんどん上昇していくという関係があります。同時リクエスト数の増加は、該当サーバにおけるメモリリソースの利用や外部サーバへの同時コネクション数などの数値に密接に影響を与えますので注意してください。

平均スループットの考え方の違い

負荷試験環境では、非常に時間のかかるリクエストが少しでも混ざると、次のリクエストを投げることができず、全体のスループットが大きく落ち込むことがあります(**図5**)。しかし、公開環境においては、一部の時間のかかるリクエストは全体の処理のボトルネックとはならない場合があります。このときボトルネックとなる一部のリクエスト以外のリクエストには、十分な負荷をかけられなくなるという状況が発生します。このような場合は、時間のかかるリクエストは別スレッドからの攻撃にするか、シナリオからいったん外してください。

負荷試験ツールの紹介

次に挙げるのは、筆者が環境に応じて使い分けているツール群です。ここで紹介する以外にもさまざまなツールがあるのですが、今までに述べた基本的な考え方はそれぞれのツールに依存せず、ほぼ同じです[注2]。

- ApacheBench
- Locust
- Jmeter
- Tsung

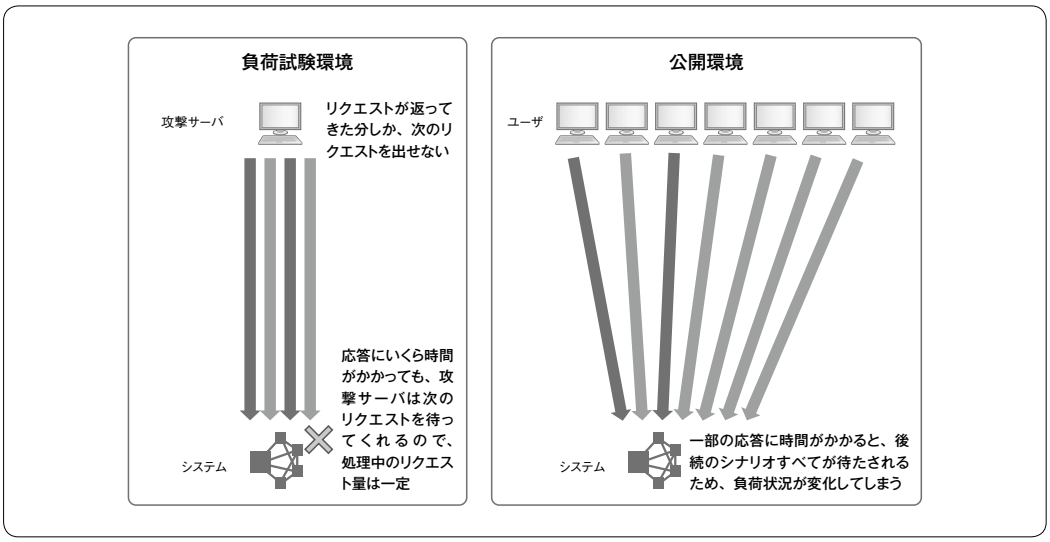
これらにはそれぞれ特徴があり、対象システムに合わせて適切なツールを選定せねばなりません。また、同じURLに対する負荷試験を行っているにもかかわらず、使うツールによって結果が異なることはよくあります。その場合は、対象のシステムにより負荷がかかり、結果としてより高いスループットが出ているほうがより良い負荷試験となります。そこで、想定した負荷がかからない場合には、ほかのツールを使用して数値を比較します。

[アンチパターン]対象のシステムに合わない負荷試験ツールを使用する

負荷試験はその規模や目的によって、使用するツールが変わってきます。たとえば、非常に

注2) オンラインでURLを入力することで負荷試験を行うサービスもありますが、ネットワーク的に近い場所からの試験ができませんので、適切な負荷試験の実施が困難になります。そのため本稿では触れません。

▼図4　負荷試験環境と公開環境における同時リクエスト数の違い



▼表1　負荷試験ツールの特性[注3]

	導入の容易さ	複雑なシナリオへの対応	結果の見やすさ	高負荷対応
ApacheBench	★★★★★	—	★	★
Locust	★★★	★★★★★	★★	★★
JMeter	★	★★★★★	★★★★★	★★★
Tsung	★★	★★★	★★★	★★★★★

高いスループットを提供するべきシステムがあり、それを試験するためにApacheBenchを利用すると、その特性のせいで結果として十分な負荷をかけきれないことがあります。試験を実施するにあたり、その特性に慣れたツールを1つ作ることも非常に重要ですが、ツールの選択も同様に重要です。

負荷試験ツール全体的な特性

システム応答の監視について、**表1**の各ツールに含まれるスループットの監視ツール、可視化ツールがありますが、全体のスループットが重要である場合にはロードバランサのモニタリングを利用すると、ツールに依存せずにモニタリングできます。また、ネットワークレイテンシを含まないシステムの処理レイテンシを計測するためにもロードバランサのモニタリングが必要です。

AWSの場合、ロードバランサのモニタリングは管理コンソールから簡単にモニタリングできます(**図6**、**図7**)。

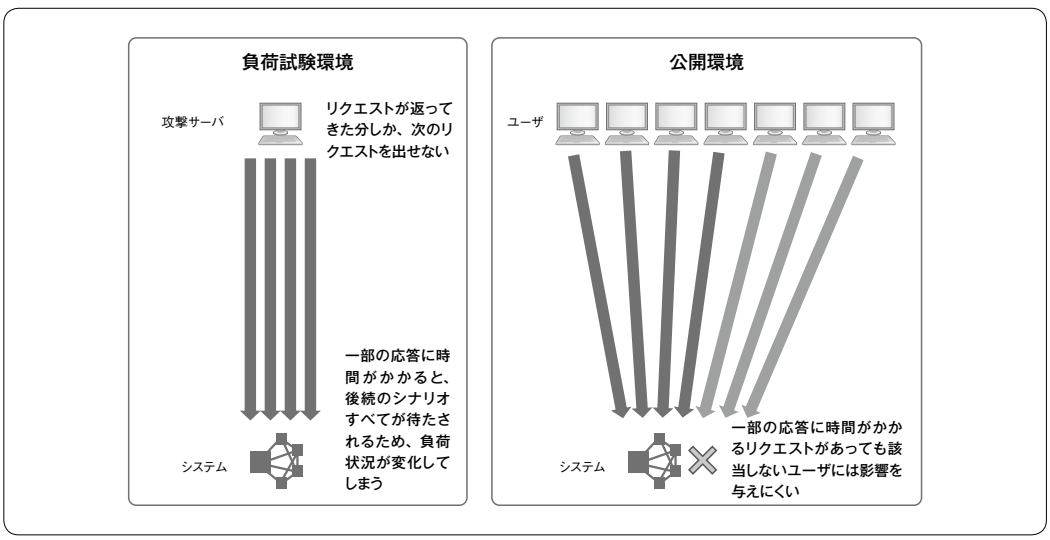
スループットは、**図7**のCloudWatchメトリックスの欄の[合計2xx(カウント)]、レイテンシとしては、[平均レイテンシ(ミリ秒)]を確認してください。

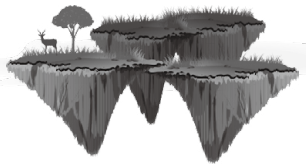
同様にELBの処理レイテンシを含まないサーバ上での処理を計測するためにはアクセスログに処理時間を出力する必要があります。Apache 2では、httpd.confのLogformat指定部分に%Dを追加してhttpdを再起動します。これで応答時間がmicrotimeで表示されるようになります。

```
LogFormat "%D %h %l %u %t ¥"%r¥" %>s ➡
%b" common
```

注3) これらは筆者の主観です。各ツールの機能を適切に使うことにより実際の評価は変動します。

▼図5 平均スループットの違い





使用リソース監視ツールについて

リソースの利用状況の監視ツールとしては、AWSを利用する場合は管理コンソールから利用できるCloudWatchでほぼ十分と筆者は考えています。ただし、RDSやELBでは1分間隔でのモニタリングをしますが、EC2インスタンスに関してデフォルトの監視タイミングは5分間隔です。そのため負荷をかけていない間のリソース使用状況と平均化されますので、15分程度の負荷をかけ続ける必要があります。そのためグラフ上ではリソースの使用状況が正しく反映されません。対応策として追加料金は発生しますが、負荷試験中はCloudWatchの[メトリックスの詳細モニタリングを有効化]オプションを付けることで、1分間隔のモニタリングが可能になります。スケールアウト対象のサーバの場合、最低でも1台のサーバに設定しておきます。

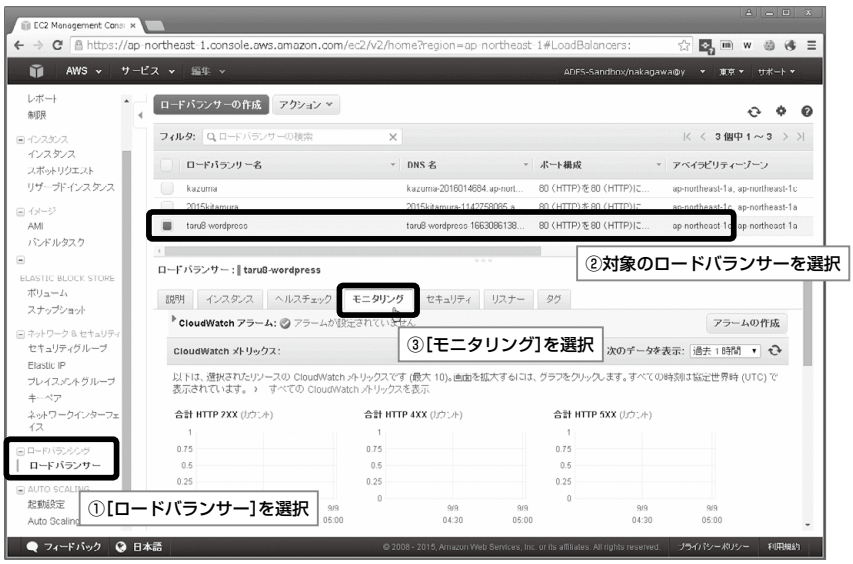
ApacheBench

特徴と注意点

インストールが平易なので時間をかけることができない場合など、ApacheBenchがお勧めです。ただし、シナリオを記載した試験を行うことができないので、シナリオを使用した試験

▼図6 AWSにおけるロードバランサのモニタリング①

▼図7 AWSにおけるロードバランサのモニタリング②



が必要なシステムの場合には、他のツールの利用を検討する必要があります。また、システムからリダイレクトヘッダが返答されてくる場合でも、リダイレクト先のURLに再アクセスしないことに注意してください。

特徴をまとめると次のようになります。

・単一のURLに対する試験を簡単に行うことができる
・POST/PUTの試験も可能（※DELETEはできない）
・リクエストごとにパラメータを変更できない
・シナリオ記載ができない
・攻撃サーバのCPUコアを1つしか利用できない

導入方法

Apacheがインストール済みのサーバならばすぐに利用できます。また、Apacheをインストールしたくない場合は、apr-utilパッケージまたはhttpd-toolsを導入すると利用できます。

```
$ sudo yum install apr-util
```

または、次のコマンドを入力します。

```
$ sudo yum install httpd-tools
```

重要なオプション

ApacheBenchでは先に述べたRampUP時間の設定項目はなく、最初のタイミングで-cオプションで設定した数のリクエストを同時に送付しようとします。そのため、攻撃開始時にサーバに処理が集中しますので、クライアント数の設定は攻撃先サーバの同時接続数の上限を超えないように注意してください。

```
% ab -h
```

でヘルプ表示します。重要なオプションを**表2**にまとめます。使い方は次のようになります。

```
Usage: ab [options] [http[s]://]ア
hostname[:port]/path
```

ApacheBenchの実行

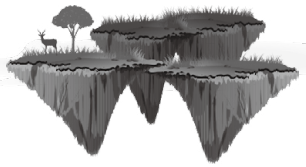
実行例を**図8**に示します。ここではローカルに設置したサーバのTOPページへのアクセスをしています。この結果の中で筆者がとくに注目するパラメータは先に述べた次の2つです。

・Requests per second(平均スループット)
・Time per request(平均レイテンシ)

負荷試験としては、このTime per requestがあらかじめ決められた許容範囲でどれだけRequest per secondを増やせるかを、性能の指標とすることが多いです。

▼表2　ApacheBenchの重要なオプション

オプション	意味	説明
-n	requests	総リクエスト数(※クライアントの同時起動数×シナリオ実行回数に相当)
-c	concurrency	同時リクエスト数(※クライアントの同時起動数に相当)
-p	postfile	POSTメソッドでリクエストする場合にbodyを記載したファイルを指定する
-u	putfile	PUTメソッドでリクエストする場合にbodyを記載したファイルを指定する
-i	—	HEADメソッドでのリクエストを行う
-C	attribute	cookieを送付する場合に使用
-A	attribute	Basic認証を利用する場合に使用
-k	keepalive	KeepaAliveを利用する場合に使用
-h	help	ヘルプ表示



▼図8　ApacheBenchの実行例

```
~% ab -n 1000 -c 100 http://localhost/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking virtualbox1 (be patient)
Completed 100 requests

...(中略)...

Completed 1000 requests
Finished 1000 requests

Server Software:        Apache/2.2.15
Server Hostname:        virtualbox1
Server Port:            80

Document Path:          /
Document Length:        4883 bytes

Concurrency Level:      100
Time taken for tests:   0.991 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      5185930 bytes
HTML transferred:       4912298 bytes
Requests per second:    1008.70 [#/sec] (mean)
Time per request:       99.137 [ms] (mean)
Time per request:       0.991 [ms] (mean, across all concurrent requests)
Transfer rate:          5108.46 [Kbytes/sec] received
...(以下略)...
```

Apache JMeter

特徴と注意点

非常に多岐にわたる機能が提供されているツールです。また、攻撃サーバのインスタンスタイプにもよりますが、数千リクエスト／秒(req/sec)までのシステムの場合は十分な負荷をかけられます。ただし、Apache JMeterは攻撃サーバ1台あたりのリソース使用量が比較的大きく、より高負荷をかけたい場合には、攻撃サーバとして複数のサーバを準備する必要があります。また、そのためには各攻撃サーバ間の通信ポートを開いておくなどのネットワーク設定が必要です。攻撃サーバを複数台準備する場合には各攻撃サーバと、シナリオを流し込む元のサーバのJMeterのバージョンが一致している必要があることも注意してください。JMeterを利用して高負荷をかける方法は、クラスメソッド㈱の記事によくまとまっています[注4]。

特徴を次にまとめます。

- Apache Benchでできない、DELETEメソッドの試験が可能
- リクエストごとに動的にパラメータを変更することが可能
- 複数のURLに対してシナリオを組んだ複雑な試験を行うことが可能
- シナリオはXMLで記述するがGUIが用意されており、比較的直感的なシナリオの記載が可能
- ProxyRecorderを利用したシナリオ作成も可能
- 試験結果の表示機能が豊富
- 複数のサーバを連携させることで比較的高負荷をかけることができる

注4)「SpotInstanceとJMeterを使って400万req/minの負荷試験を行う」http://dev.classmethod.jp/cloud/apache-jmeter-master-slave-100mil-req-min/

・HTMLコンテンツの場合、コンテンツ中で要求されるさまざまな静的リソースも同時に取得する試験を行うことが可能

導入方法

JDKのインストール後にソースファイルを展開するだけで利用できます[注5]。

実行結果サンプル

スレッドグループを作成し(**表3**)、どれくらいのクライアント数でどれくらいの回数の負荷試験を行うかなどを設定します(**図9**)。

TOPページコンテンツ中で呼び出される静的なリソースも自動で取得されていることが確認できます(**図10**)。**図11**では統計レポートを出力しています。

筆者がよく実施する方法は、シナリオの作成はローカルのWindows PCから[結果をツリーで表示]を有効にした状態で、少ないクライアント数の設定および攻撃回数で行い、シナリオが確定したら、別途構築した専用のインスタンスにシナリオを設置して本格的な攻撃を行う方法です[注6]。

Locust

Locustは英語でイナゴを意味するようです。大量のリクエストが同時に飛ぶイメージでしょうか。シナリオをPythonで記述できる負荷試験ツールです。

特徴と注意点

JMeterに近い機能があり、比較すると次のようになります。

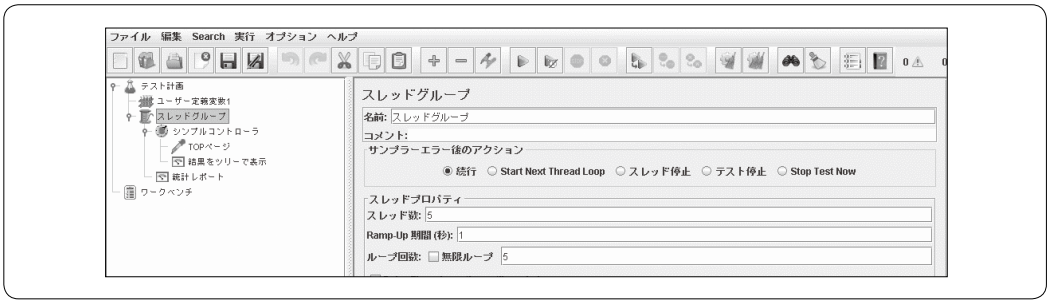
・Pythonで試験シナリオを作るので、柔軟な記述ができる
・シナリオがスクリプトであることから、Git管理などと相性がよい
・必要なサーバリソースが少ないので少数の攻撃サーバで負荷をかけやすい
・結果表示がシンプル

攻撃サーバ上でコマンドラインで攻撃ツールを起動し、それをWebのインターフェースから同時接続ユーザ数を指定して実際の攻撃を開始します。結果表示としての詳細な表示が不要なことも多いですので、筆者の場合、JMeterの代わりにこのツールをよく利用します。

▼表3　Apache JMeterのスレッドプロパティ

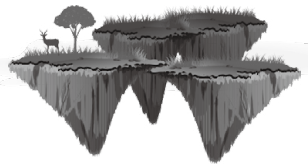
スレッド数	[クライアントの同時起動数]に相当
RampUp期間(秒)	[クライアントがすべて起動するまでの時間]に相当
ループ回数	[シナリオ実行回数]に相当

▼図9　Apache JMeter(スレッドグループの作成と設定)

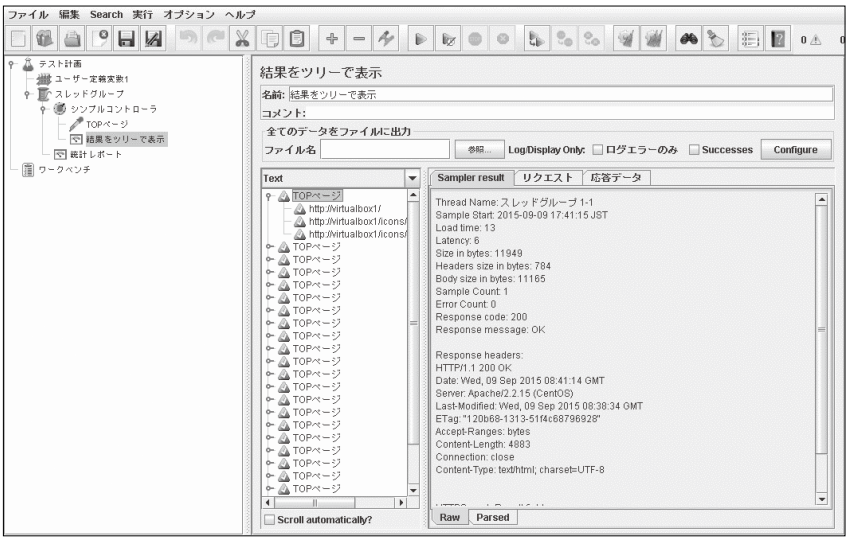


注5) http://jmeter.apache.org/　など

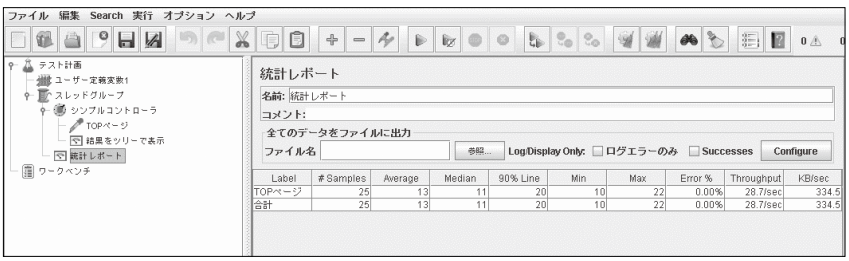
注6) 本格的な攻撃を行う際には[結果をツリーで表示]オプションは外す必要があります。ここで、[ログエラーのみ]のオプションを付けても実際にはかなりの負荷となり、対象のサーバに適切な負荷をかけることができなくなる原因となります。また、JMeterには非常に便利な機能がたくさんあるのですが、それらの中には[結果をツリーで表示]と同様に本格的な負荷をかける際には邪魔になる機能が多く含まれているので、随時調整してください。



▼図10　Apache JMeter（静的なコンテンツの自動取得の確認）



▼図11　Apache JMeter（統計レポートの出力）



インストール方法

公式Webページなどを参照し導入してください[注7]。

攻撃サーバ起動スクリプトサンプル

上記公式サイトにサンプルがありますが、run_10slaves.shなどのファイル名で**リスト1**を記述してから起動すると、同一サーバ上で複数の攻撃Slaveサーバを簡単に構築できます。

```
$ sh ./run_10slaves.sh
```

攻撃サーバを起動後に、Webインターフェースとして用意された画面からクライアントの同時起動数、追加率などを指定して実際の攻撃を開始します。

実行例

実行結果（**図12**）の項目の意味を次に示します。

①攻撃ユーザ数（クライアント数）：攻撃開始時に攻撃クライアントの増加率とともに設定する
②Slaveサーバ数：CPUコアに余裕がある場合などはSlaveサーバ数を増やすことで、負荷をかける上限をあげることができる
③総スループット：全体のスループットですので、シナリオとしてのスループットとは一致しないことに注意する

注7）http://docs.locust.io/en/latest/installation.html

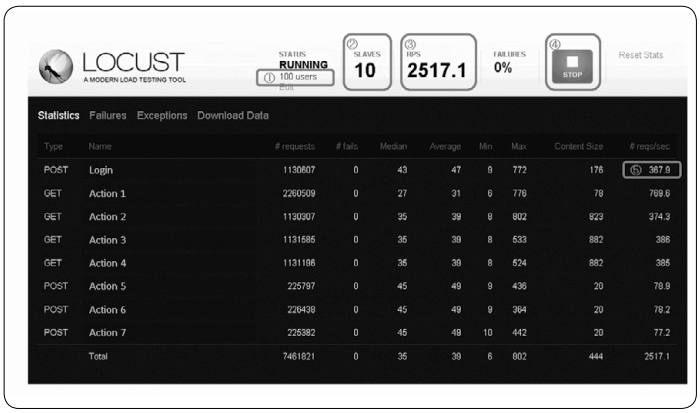
▼リスト1　起動スクリプト例（10 slaveサーバで起動する場合）

```
#!/bin/sh

LOCUST_FILE=${1:-src/locustScinario.py}
SCINARIO_CLASSNAME=WebsiteUser
sudo /bin/sh -c "
ulimit -n 65535
for slaves in {1..10}
do
    /usr/local/bin/locust -f ${LOCUST_FILE} ${SCINARIO_CLASSNAME} --slave &
done

/usr/local/bin/locust -f ${LOCUST_FILE} ${SCINARIO_CLASSNAME} --master
```

④Stopボタン：Locustでは起動時に総リクエスト数を指定せず、試験の中断はこのボタンから行う

⑤URL別のスループット：シナリオとしてのスループットを見るためには、各シナリオで一度だけ通るリクエストのスループットを見る。この場合は、ログイン処理を一度だけと定義しているため、1秒間に367シナリオをこなせるという見方をする

▼図12　Locust実行画面



Tsung

Erlangで記載された速度重視のツールです[注8]。

特徴と注意点

JMeterに近いのですが、比較すると少し癖があります。その特徴を次に挙げます。

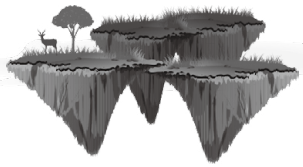
- シナリオをXMLで記載するのはJMeterと同様。むしろJMeterより比較的シンプルで理解しやすい記述が可能（ただしGUIによるシナリオの作成・閲覧ができないので、複雑なシナリオの管理にはやや不向き）
- JMeterと同じく、ProxyRecorderを利用してシナリオを作成可能
- 少ない攻撃サーバで高負荷をかけることが非常に得意
- 試験結果はJSON形式で出力されるが、それを可視化するためのWebインターフェースも完備

Locustとは異なり、コマンドラインで起動したタイミングから実際の攻撃を開始します。

用意された機能を正しく使用すれば、Tsungだけで詳細なシナリオを記載した負荷試験が可能ですが、XML記述の難易度が高いため、詳細なシナリオのチェックはLocustで行い、さらに高負荷をかけたいときにバックグラウンドでTsungで簡単なシナリオを流すなどの組み合わせで、筆者は利用しています。

ただし、このような方法をするときにはツール上で見る結果表示の信頼性が著しく下がりますので、前述したロードバランサにおけるスループットのモニタリングなどを利用し、各負荷試験ツールの提供する画面は利用しません。

注8）http://tsung.erlang-projects.org/user_manual/index.html



▼リスト2　Tsungのシナリオファイル例

```
<?xml version="1.0"?>
<!DOCTYPE tsung SYSTEM "/usr/local/Cellar/tsung/1.5.1/share/tsung/tsung-1.0.dtd" [] >
<tsung loglevel="notice" version="1.0">
  <clients>
    <client host="localhost" use_controller_vm="true" maxusers="2000" cpu="10"/>
  </clients>
  <servers>
    <server host="[server_host]" port="80" type="tcp"></server>
  </servers>
  <load>
    <arrivalphase phase="1" duration="10" unit="minute">
      <users maxnumber="2000" arrivalrate="200" unit="second"></users>
    </arrivalphase>
...（以下略）...
```

導入方法

公式サイトを参照し、次のように導入してください[注9]。

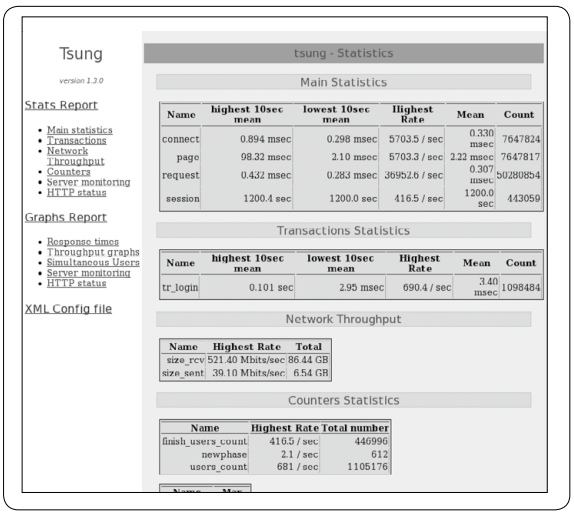
```
$ git clone https://github.com/processone/tsung.git
$ cd tsung
$ ./configure
$ make && sudo make install
```

シナリオ記載、試験実施

ここで紹介するにはボリュームが大きくなりますので、AccessTokenを取得して利用したページを呼び出す簡単なサンプルのみ一部紹介(**リスト2**)。このようなXMLファイルを、公式Webページなどを参考にしながら記述します。

注9) http://tsung.erlang-projects.org/user_manual/installation.html

▼図13　Tsungの統計レポート表示(公式サイトより)



実行結果サンプル

実行結果は、**図13**に示します。

次回予告

負荷試験は、今回紹介したようなツールを利用して進めていけばいいのですが、実際に負荷試験を進めるにあたっては効率的に行うための段取りがあります。それを無視して進めることは、結果としてシステムのプロファイリング(ボトルネックや改善点の分析)を難しくし、負荷試験を無駄なものにしてしまいます。

次回は効率的に負荷試験を進める方法を、次の項目で紹介します。SD

- 静的ファイルを叩くことで利用する負荷試験ツールや設定の試験を行う
- HelloWorldを叩く
- 参照系のページ、APIを叩く
- 更新の発生するページ、APIを叩く
- 外部サービスとの結合を含むページ、APIを叩く
- シナリオを組んで試験を行う
- 離れたネットワークから試験を行う
- 各リソースのスケールアップをして試験を行う
- ソースのスケールアウトをして試験を行う
- より強い負荷を与える

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2015年12月号

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

第2特集
ネットワーク・システム管理の定石
SNMPの教科書

短期連載
・クラウド時代のWebサービス負荷試験再入門

定価（本体1,220円+税）

2015年11月号

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

第2特集
攻撃を最前線で防ぐ
ファイアウォールの教科書

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

定価（本体1,220円+税）

2015年10月号

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

第2特集
Webメールの教科書
クラウドサービス利用か？　自社で構築か？

特別付録
・創刊300号記念　Vim&Emacsチートシート

定価（本体1,220円+税）

2015年9月号

第1特集
特講
正規表現・SQL・オブジェクト指向
苦手克服のベストプラクティス

第2特集
メールシステムの教科書
日本語もバイナリもちゃんと届くのはなぜか

特別企画
・なぜ俺の提案は通らないのか？

定価（本体1,220円+税）

2015年8月号

第1特集
Lispより始めよ、されば救われん！
なぜ関数型プログラミングは難しいのか？

第2特集
安全な通信を確保する
SSL/TLSの教科書

短期連載
・AWSで始めよう！　モダンなJavaアプリケーション開発

定価（本体1,220円+税）

2015年7月号

第1特集
あなたにもできる！
ログを読む技術［セキュリティ編］

第2特集
黒い画面（tmux）の使い方
プロになるためのターミナル活用術

第3特集
6人の先駆者に訊く
スペシャリストになる方法

定価（本体1,220円+税）

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN＆ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	渋谷区	紀伊國屋書店　新宿南店	03-5361-3315	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	書泉ブックタワー	03-5296-0051	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	千代田区	丸善　丸の内本店	03-5288-8881		広島市中区	丸善　広島店	082-504-6210
	中央区	八重洲ブックセンター本店	03-3281-1811	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322
	渋谷区	MARUZEN＆ジュンク堂書店　渋谷店	03-5456-2111				

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。

手がかりを探せ！

SMB実装をめぐる冒険 第3回

File System for Windowsの作り方

探す、調べる、ソフトを作る喜び

こんにちは。よういちろうです。「Windows共有フォルダをChromeOSのファイルアプリにマウントする」ことができるChromeアプリを開発してリリースしました。これを開発するためには、SMB（Server Message Block）と呼ばれるプロトコルを理解し、SMBプロトコルを話すクライアントコードをJavaScriptで書くことが必要でした。これは「File System forWindows」という名前でChromeウェブストアにて無料で公開していますので、Chromebookを持っている方はぜひ使ってみてください。今回は、短期連載2回目に引き続き探偵風に開発過程を紹介します。

Author 田中 洋一郎（たなか よういちろう） Blog https://www.eisbahn.jp/yoichiro Twitter @yoichiro

第5部 相手の資産を把握しろ

前回まで、ユーザ認証について、その認証方式を探りました。NTLMSSP認証方式を分析するにあたり、Type 1 Message、Type 2 Message、Type 3 Messageをそれぞれ調べ、ユーザ認証のしくみを明らかにしました。

ユーザ認証ができれば、接続先のサーバの資源を利用できそうですが、そう簡単にいかないのがSMBの複雑なところです。「共有リソース一覧を入手」し、「共有リソースのTree IDを入手」せねばなりません。ユーザ認証の壁の次は、共有リソースの壁でした。

共有リソース関連コマンドが見当たらない

ユーザに共有リソースの名前をクライアントの画面上で入力してもらえれば簡単なのですが、やはり自動的に共有リソース一覧をサーバから取得して提示し、ユーザが選択するだけで済むようにしておくべきでしょう。OS XやWindowsなど各種OSが当然のように共有リソース一覧を提示してくるので、もちろん方法があるはずです。

しかし、@ITの記事[注1]にも、Impementing CIFS本[注2]にも、そのやり方が掲載されていませんでした。さらに、MS-CIFS仕様書を読んでも、それらしい記述は見当たりません。インターネットで検索しようにも、「SMB shared resource list」といったキーワードで検索してみましたが、それらしい情報はヒットしませんでした。SMB_SHARED_RESOURCE_LISTというコマンドがあることを期待しましたが、残念ながらそれらしいコマンドは存在しませんでした。

数時間ほど探しに探したのですが、まったく情報が得られない状況でした。せっかくユーザ認証という大きな壁を超えたのに、敵はもう1つ大きな壁で行く手を阻んできました。

困ったときのパケットキャプチャ

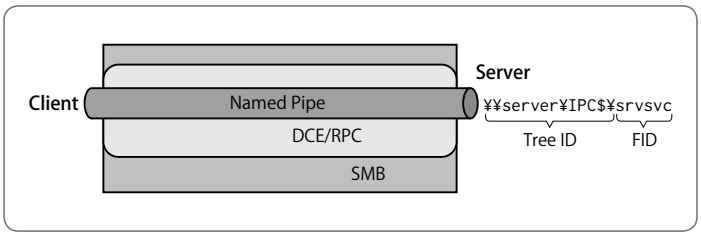
探しても情報が見つからなかった場合は、目の前に流れている正解を解析するしかありません。そう、パケットキャプチャの出番です。今回の場合は、SMB1/CIFSで定義されているコマンドの中にはズバリそのものがなかったため、NTLMSSP認証方式のときと同様に、きっとサブプロトコル的なものが適用されている予感がします。そんなときは、**tcpdump**コマンドではなく、Wiresharkが有益な情報をもたらしてくれそうです。

Wiresharkを起動しておいて、SMB1/CIFSプロトコルの設定を施しておいたSambaサーバにMac OS XのFinderから接続して、送受信さ

注1） http://www.atmarkit.co.jp/ait/articles/0410/29/news103.html
注2） Implementing CIFS（http://www.amazon.co.jp/dp/013047116X）もしくはImplementing CIFS—ubiqx.org（http://ubiqx.org/cifs/）

れるパケットを「盗聴」しました。その結果、共有リソース一覧を取得するために、予想していたよりも多くのメッセージが送受信されていることがわかりました。具体的には次の手順になります。

▼図1　共有リソース一覧取得のための概念



①SMB_COM_TREE_CONNECT_ANDXコマンドを使って、"¥¥[server_name]¥IPC$"という共有リソースに接続し、その結果のTree IDを入手する
②SMB_COM_NT_CREATE_ANDXコマンドを使って、"¥srvsvc"をオープンし、その結果のFID値を入手する
③SMB_COM_TRANSACTIONコマンドを使って、名前付きパイプを用いたDCE/RPCプロトコルによる先ほど入手したFIDへのバインドを行う
④SMB_COM_TRANSACTIONコマンドを使って、名前付きパイプを用いたDCE/RPCプロトコルによるNetShareEnumAllオペレーションをサーバに送信し、共有リソース一覧を入手する

簡単に言うと、**図1**のように「SMBプロトコル上で、DCE/RPCプロトコルを使って"¥¥[server_name]¥IPC$¥srvsvc"に対する名前付きパイプで共有リソース一覧取得を要求する」という内容です。SMBプロトコルは通信のための単なるペイロードであり、実際に行われていることはRPC(Remote Procedure Call)です。つまり、共有リソース一覧の取得に関してはDCE/RPC on SMBという多段プロトコルであり、SMBプロトコルの知識だけでは足りない、ということです。

DCE/RPCを勉強せずにDCE/RPCする

一般的にRPCは、利用したいプログラミング言語に応じたStubおよびSkeletonというコードを準備(ほとんどの場合IDLから自動生成されます)することで、通信内容をまったく知らなくても通信処理がコーディング可能になります。DCE/RPCについても本来はSDKなどを使って扱うものなのですが、今回の場合はDCE/RPCでの通信に必要なバイト列を自分で作って、それをSMBプロトコルのメッセージ内に含める必要があります。

「DCE/RPCは複雑なことが有名」ということがWikipediaに書かれていて、DCE/RPCを勉強して把握することは最初から諦めました。では、どうしたら良いでしょうか？

パケットキャプチャした結果を注意深く見ていったところ、あることに気がつきました。それは、

「動的に変更しなければならない値は、そう多くない」

ということです。DCE/RPCを成立させるために必要な値がほとんどを占めていて、それらはクライアントやサーバの動作環境に依存せず固定値で良さそうです。そして、パケットキャプチャから得られた内容に基づいて、動的に変化させなければならない値の指定方法を導き出すことができれば……、やりたいことを満たすことはできそうだと考えました。ユーザ認証のための処理ほどすべてを把握する必要はなさそうなので、この戦略で十分開発できそうな予感がしました。

事実、この戦略は正しかったのです。最終的に、さほど苦労することなく、共有リソース一覧の取得に成功しました。

IPC$への接続とTree IDの入手

では、具体的なメッセージのやりとりの説明に入っていきましょう。最初に行うことは、IPC$リソースへ接続し、Tree IDをサーバから入手することです。

IPC$(InterProcess Communication)とは、「そのサーバが外部に公開している共有リソース(ファイルやプリンタなど)一覧の取得などを行うためのプロセス間通信(IPC)に使用されるWindowsの機能」であり、Windows系OSやSambaでは必ずIPC$を共有リソースの1つとして公開しています。クライアントは、SMB_COM_TREE_CONNECT_ANDX(0x75)メッセージを使って、このIPC$共有リソースに接続を行います。サーバにて接続処理が成功すれば、その接続を示すTree ID値がレスポンスとして返ってきます。

SMB_COM_TREE_CONNECT_ANDXリクエストで使われるSMB_PARAMETERおよびSMB_DATAの構造は、**リスト1**のようになります。

先ほどすでにユーザ認証を行っていますので、SMB_COM_TREE_CONNECT_ANDXリクエストでのパスワードはなし(0x00値のみ)で大丈夫です。ここで注意すべき点としては、pathとserviceの文字コードの違いです。

・path：NULL終端文字付きUNICODE文字列
・service：NULL終端文字付きASCII文字列

serviceには"?????"を指定します。「え？　5文字であれば何でも良いという意味？」と思うかもしれませんが、違います。クエスチョンマーク5つです。これは「任意のリソースまたはサービスにマッチする」という指定になります。

▼リスト1　SMB_COM_TREE_CONNECT_ANDXリクエストの構造

```
SMB_PARAMETER {
  UCHAR word_count;         // 8
  words {
    struct {
      UCHAR command;        // 0xff
      UCHAR reserved;       // 0x00
      USHORT offset;        // 0x0000
    } ANDX;
    USHORT flags;           // 0x0000
    USHORT password_length; // 0x0001
  }
}
SMB_DATA {
  USHORT byte_length;
  bytes {
    ANY password;            // 0x00
    UCHAR[0 or 1] padding;  // この場合はなしでOK
    ANY path; // "¥¥[server_name]¥IPC$"
    ANY service; // "?????"
  }
}
```

サーバから返されるSMB_COM_TREE_CONNECT_ANDXレスポンスには、もちろんSMB_PARAMETERやSMB_DATAにいくつか値が含まれているのですが、それらはさほど重要ではありません。ヘッダに含まれるnt_status値が0(エラーなし)であれば、同じくヘッダに含まれるtree_id値を入手して、SMB_COM_TREE_CONNECT_ANDXメッセージでの目的は達成です。このtree_id値を使って、IPC$共有リソースにアクセスできます。

/srvsvcのオープンとFIDの入手

IPC$のTree IDを入手したあとは、そのIPC$共有リソースの中にある"/srvsvc"という名前付きパイプをオープンし、そのFID値を得ます。これには、SMB_COM_NT_CREATE_ANDX(0xa2)コマンドを使います。このコマンドのメッセージは、既存のファイルをオープンしてFID値を返す働きが基本ですが、もし指定されたファイル名のファイルが存在しなかったときに新規作成を指示したりできます。また、相手がファイルではなく名前付きパイプであったとしても、ファイルと同じように扱うことができるようになっています。

そのため、SMB_COM_NT_CREATE_ANDXリクエストが持つ値の数は、少し多めです。最初このメッセージに出会ったときは「うわ、面倒そうだな」と思ったのですが、よく見ていくと、ファイルを作成する際に必要な設定値としては当たり前のものばかりでした。SMB_COM_NT_CREATE_ANDXリクエストの構造は**リスト2**のようになります。

desired_accessは、オープンしたファイルや名前付きパイプに対して行われる可能性がある処理を指定するためのフラグ値です。本来は最低限の指定に留めるべきですが、次のようにほぼフルセットとなる値を指定しておけば問題はないでしょう。

▼リスト2 SMB_COM_NT_CREATE_ANDXリクエストの構造

```
SMB_PARAMETER {
  UCHAR word_count;
  words {
    struct {
      UCHAR command;           // 0xff
      UCHAR reserved;          // 0x00
      USHORT offset;           // 0x0000
    } ANDX;
    UCHAR reserved;            // 0x00
    USHORT name_length;
    UINT flags;                // REQUEST_OPLOCK(0x02)➡
| REQUEST_OPBATCH(0x04)
    UINT root_directory_fid;   // 0
    UINT desired_access;       // 0x2019f
    ULONG allocation_size;     // 0
    UINT ext_file_attributes; // EXT_FILE_ATTR_ATTR_➡
NORMAL(0x80)
    UINT share_access;         // READ(0x01) |
WRITE(0x02) | DELETE(0x04)
    UINT create_disposition;   // FILE_OPEN(0x01)
    UINT create_options;       // NON_DIRECTORY_➡
FILE(0x40)
    UINT impersonation_level; // SEC_IMPERSONATE(0x02)
    UCHAR security_flags;      // CONTEXT_TRACKING(0x01)
| EFFECTIVE_ONLY(0x02)
  }
}
SMB_DATA {
  USHORT byte_count;
  bytes {
    ANY file_name;             // Null-terminated UNICODE
  }
}
```

・FILE_READ_DATA (0x000001)
・FILE_WRITE_DATA (0x000002)
・FILE_APPEND_DATA (0x000004)
・FILE_READ_EA (0x000008)
・FILE_WRITE_EA (0x000010)
・FILE_READ_ATTRIBUTES (0x000080)
・FILE_WRITE_ATTRIBUTES (0x000100)
・READ_CONTROL (0x020000)

“IPC$/srvsvc”に対するSMB_COM_NT_CREATE_ANDXリクエストで重要な値は、file_nameおよびcreate_dispositionです。file_nameには“¥srvsvc”を、create_dispositionにはFILE_OPENを指定します。もちろん、ヘッダにuser_id値および先ほど入手したtree_id値を指定するのも忘れてはなりません。

サーバがリクエストの内容に基づいて正しく処理できれば、“¥srvsvc”に対するFID値が発行され、クライアントに返却されます。ヘッダにはFIDを入れる場所がないので、クライアントはSMB_COM_NT_CREATE_ANDXレスポンスの内容からFID値を取り出す必要があります。SMB_COM_NT_CREATE_ANDXレスポンスの構造は、**リスト3**になります。SMD_DATAはありません。

▼リスト3 SMB_COM_NT_CREATE_ANDXレスポンスの構造

```
SMB_PARAMETER {
  UCHAR word_count; // 68
  words {
    struct {
      UCHAR command; // 0xff
      UCHAR reserved; // 0x00
      USHORT offset; // 0x00
    } ANDX;
    UCHAR op_lock_level;
    USHORT fid; // FID
    UINT create_disposition;
    ULONG create_time; // SMBタイムスタンプ
    ULONG last_access_time; // SMBタイムスタンプ
    ULONG last_write_time; // SMBタイムスタンプ
    ULONG last_change_time; // SMBタイムスタンプ
    UINT ext_file_attributes;
    ULONG allocation_size; // Disk上のサイズ
    ULONG end_of_file; // 実際のサイズ
    USHORT resource_type;
    USHORT nm_pipe_status;
    UCHAR directory; // Directory=1, Otherwise=0
  }
}
```

DCE/RPCによるFIDへのバインド

“¥¥server¥IPC$¥srvsvc”に対応するTree ID値およびFID値を得ることができました。これでやっと関所と会話することができます。その会話方法は、先ほど紹介したDCE/RPCプロトコルです。

DCE/RPCプロトコルは、SMBプロトコルのSMB_COM_TRANSACTION(0x25)メッセージによって送受信されます。DCE/RPCプロトコルの前に、このSMB_COM_TRANSACTIONメッセージの内容を見ていきましょう。

SMB_COM_TRANSACTIONメッセージは、まさに名前付きパイプやプロセス間通信など、サブプロトコルを扱いたいときに使用するメッセージです。

そのため、サブプロトコルのバイト列を汎用的に運搬できるように設計されています。SMB_COM_TRANSACTIONは「trans_setup、trans_parameter、trans_data」という3つのペイロードがあります。サブプロトコルごとに、3つのうちのどれにどんな情報を持たせるかが異なっています。また、3つのうちの1つだけではなく、複数のペイロードを必要とするサブプロトコルも珍しくありません。

SMB_COM_TRANSACTIONリクエストの構造は、**リスト4**のようになります。

trans_parameterおよびtrans_dataについては、それぞれ「全部のバイト数は○○だけど、今回送ったのは△△」という指定ができるようになっていて、送りたい量が多い場合に複数のリクエストに分けて送信ができます。しかし、2回目以降は別のコマンド(SMB_COM_TRANSACTION_SECONDARYなど)を使う場合があり、注意が必要です。

SMBプロトコル側の説明ができたところで、DCE/RPCの話に移りましょう。バインドのためのDCE/RPCリクエストは、trans_dataとtrans_setupを使って送信されます。その構造は、**リスト5**のようになります。

リスト5を見てわかるとおり、重要な値はpacket_type、function、そしてfidくらいです。他の値は、DCE/RPCの都合で必要となる値ばかりに見えます。頑張ってDCE/RPCを勉強する

▼リスト4　SMB_COM_TRANSACTIONリクエストの構造

```
SMB_PARAMETER {
  UCHAR word_count;
  words {
    USHORT total_parameter_count; // Parameterの総サイズ
    USHORT total_data_count;      // Dataの総サイズ
    USHORT max_parameter_count;   // 1リクエストあたりのParameterの最大長
    USHORT max_data_count;        // 1リクエストあたりのDataの最大長
    UCHAR max_setup_count;        // Setupの最大長
    UCHAR reserved;               // 0
    USHORT flags;                 // 0
    UINT  timeout;                // サーバのタイムアウト時間(ms)
    USHORT reserved2;             // 0
    USHORT parameter_count;       // Parameterのサイズ
    USHORT parameter_offset;      // Parameterの開始位置
    USHORT data_count;            // Dataのサイズ
    USHORT data_offset;           // Dataの開始位置
    UCHAR  setup_count;           // Setupのサイズ
    UCHAR  reserved3;             // 0
    ANY trans_setup;              // setup_count分のバイト列
  }
}
SMB_DATA {
  USHORT byte_count;
  bytes {
    ANY name;                      // Null-terminated UNICODE文字列
    UCHAR[0...3] padding1;         // 4バイト境界に合わせるためのパディング
    ANY trans_parameter;           // parameter_count分のバイト列
    UCHAR[0...3] padding2;         // 4バイト境界に合わせるためのパディング
    ANY trans_data;                // data_count分のバイト列
  }
}
```

▼リスト5　バインドのためのDCE/RPCリクエストの構造

```
name = "¥PIPE¥";
trans_setup {
  USHORT function;             // TransactNmPipe (0x0026)
  USHORT fid;                  // FID
}
trans_data {
  UCHAR[2] version;            // 5.0 (0x0500)
  UCHAR packet_type;           // BIND (0x0b)
  UCHAR packet_flags;          // Last frag (0x02) | First frag (0x01)
  USHORT data_representation; // Little Endian (0x00000010)
  USHORT frag_length;          // trans_dataの長さ
  USHORT auth_length;          // 0
  UINT call_id;                // 1
  USHORT max_xmit_frag;        // 4280
  USHORT max_recv_frag;        // 4280
  UINT assoc_group;            // 0
  UCHAR num_ctx_items;         // 1
  USHORT context_id;           // 0
  UCHAR num_trans_items;       // 1
  UCHAR[16] interface;         // SRVSVC UUID (4b324fc8-1670-01d3-127
8-5a47bf6ee188)
  UCHAR[2] interface_version; // 3.0 (0x0300)
  UCHAR[16] transfer_syntax;  // 8a885d04-1ceb-11c9-9fe8-08002b104860
  UINT ver; //0x00000002
}
```

よりも、Wiresharkのパケットキャプチャの結果から重要な個所のみを把握したほうが理解しやすかった、という戦略を取った理由がわかっていただけたと思います。

SMB_COM_TRANSACTIONレスポンスも、SMB_COM_TRANSACTIONリクエストとほぼ同じ構成です。つまり、trans_setup、trans_parameter、trans_dataの3つのペイロードを持っています。バインド結果は、SMB_COM_TRANSACTIONレスポンスのtrans_dataに格納されてサーバから返却されます。そのtrans_dataには、DCE/RPCプロトコルにおけるサーバからのレスポンスが格納されていて、次の値をチェックすることでバインドがうまくいったかどうかを確認できます。

・packet_typeがBind_ack(0x0c)であること
・44バイト目のack_resultがAcceptance(0x00)であること

DCE/RPCによる共有リソース一覧の取得

やっと準備が整いました。共有リソース一覧の取得要求をサーバに出すことができます。再度DCE/RPCプロトコルを使って、"¥¥server¥IPC$¥srvsvc"に対して名前付きパイプで共有リソース一覧を取得します。バインドと同じように、SMB_COM_TRANSACTIONメッセージとDCE/RPCプロトコルの組み合わせを使います。

共有リソース一覧の取得要求は、先ほどと同じように、SMB_COM_TRANSACTIONメッセージのtrans_setupおよびtrans_dataの2つのペイロードを使用します。trans_setupに指定する内容は、バインド時のものと同じです。

▼リスト6　共有リソース一覧取得のためのDCE/RPCリクエストの構造

```
name = "¥PIPE¥";
trans_data {
  UCHAR[2] version;              // 5.0 (0x0500)
  UCHAR packet_type;             // Request (0x00)
  UCHAR packet_flags;            // Last frag (0x02) | First frag (0x01)
  USHORT data_representation;    // Little Endian (0x00000010)
  USHORT frag_length;            // trans_dataの長さ
  USHORT auth_length;            // 0
  UINT call_id;                  // 2
  UINT alloc_hint;               // alloc_hintを含むここから最後までの長さ
  USHORT context_id;             // 0
  USHORT opnum;                  // NetShareEnumAll (0x000f)
  struct {
    UINT referent_id;            // 0x00000001
    UINT max_count;              // ASCIIでのserver_uncの文字数
    UINT offset;
    UINT actual_count;
    ANY server_unc;              // Null-terminated UNICODE文字列
  } PointerToServerUnc;
  struct {
    UINT level;                  // 0x00000001
  } PointerToLevel;
  struct {
    UINT ctr;                    // 0x00000001
    UINT referent_id;            // 0x00000001
    UINT count;                  // 0x00000000
    UINT pointer_to_array;       // 0x00000000
  } PointerToCtr;
  UINT max_buffer;               // 0xffffffff
  struct {
    UINT referent_id;            // 0x00000001
    UINT resume_handle;          // 0x00000000
  } PointerToResumeHandle;
}
```

それに対して、trans_dataの内容が少し異なってきます。共有リソース一覧取得のためのDCE/RPCリクエストの構造は、**リスト6**のようになります。

今までで最も値が多いメッセージかもしれませんが、重要と思われる値は、次の3つだけです。

・packet_type：RPCの処理要求を示すRequest(0x00)値
・opnum：RPCの処理種別を示すNetShareEnumAll(0x000f)値
・server_unc：サーバ名を示すUNICODE文字列("\\[server_name]\")

たったこれだけの情報を渡すために、上記のようなメッセージを作らなければなりません。いかにRPCの内部で複雑なやりとりが行われているかが、バインドのときよりもよくわかり

ます。実際には、上記3つ以外の値は、ほぼ変更なしでサーバは動作します。

SMBプロトコルのヘッダにuser_idおよびtree_idをセットし、trans_setupにはバインド時と同じ内容を、そしてtrans_dataに上記の内容をセットしてサーバに送信すると、user_idで指定したユーザがアクセス可能な共有リソース一覧をサーバが返してきます。これがまた……複雑かつ余計な値がどっさり入ったレスポンスです。本質的ではない内容がリクエスト以上に多く含まれています。

関所に設けられた砦が突破されてしまった今、敵は最後の抵抗として「これでも食らえ!」と情報量で圧倒してきます。しかし、ここまで複雑かつ難解なSMBプロトコル、NTLMSSP認証方式、そしてDCE/RPCプロトコルを見てきた僕にとっては、恐るるに足りません。

では、共有リソース一覧がどのようにサーバから渡ってくるか、詳細を見ていきましょう。共有リソース一覧は、SMB_COM_TRANSACTIONのtrans_dataに格納されています。**リスト7**は、そのtrans_dataに含まれるDCE/RPCレスポンスの構造です。

packet_typeがResponse (0x02)になっていること以外は、前半はリクエスト時の内容とほとんど変わりません。後半は共有リソースの名前、種別、そしてコメント文字列がセットされています。**リスト7**では便宜上種別と名前、コメントを別の構造体(struct)で記載しましたが、実際には連続したバイト列です。

TYPE_INFOとNAME_COMMENTの2つに分けましたが、どう考えても「名前、種別、コメント」で共有リソース1つを構成するはずなのに、種

▼リスト7　共有リソース一覧を持つDCE/RPCレスポンスの構造

```
trans_data {
  UCHAR[2] version;                          // 5.0 (0x0500)
  UCHAR packet_type;                         // Response (0x02)
  UCHAR packet_flags;                        // Last frag (0x02) | First frag
(0x01)
  USHORT data_representation;                // Little Endian (0x00000010)
  USHORT frag_length;                        // trans_dataの長さ
  USHORT auth_length;                        // 0
  UINT call_id;                              // 2
  UINT alloc_hint;                           // NetShareEnumAllすべての長さ
  USHORT context_id;                         // 0
  UCHAR cancel_count;                        // 0x00
  UCHAR[0 or 1] padding;
  struct {
    UINT level;                              // 0x00000001
  } PointerToLevel;
  struct {
    UINT ctr;                                // 0x00000001
    UINT referent_id;                        // サーバで生成された値
    UINT count;                              // 共有リソースの個数
    UINT referent_id;                        // サーバで生成された値
    UINT max_count;                          // 共有リソースの個数
  } PointerToCtr;
  TYPE_INFO[count] types;                    // 共有リソースの種別値などの配列
  NAME_COMMENT[count] names_and_comments; // 共有リソースの名前およびコメントの配列
  UINT total_entries;                        // 共有リソースの個数
  UINT pointer_to_resume_handle;             // 0x00
  UINT windows_error;                        // エラーコード
}
---
struct {
  UINT name_referent_id;
  UINT type;                                 // 共有リソースの種別
  UINT comment_referent_id;
} TYPE_INFO;
struct {
  UINT name_max_count;                       // 共有リソース名の長さ
  UINT name_offset;                          // 0
  UINT name_actual_count;                    // 共有リソース名の長さ
  ANY name;                                  // 共有リソース名のNull-terminated
UNICODE文字列
  UCHAR[0...3] padding;
  UINT comment_max_count;                    // 共有リソースのコメントの長さ
  UINT comment_offset;                       // 0
  UINT comment_actual_count;                 // 共有リソースのコメントの長さ
  ANY comment;                               // 共有リソースのコメントのNull-
terminated UNICODE文字列
  UCHAR[0...3] padding;
} NAME_COMMENT;
```

別を共有リソースの個数分だけ並べたあとに、あらためて名前とコメントを共有リソースの個数分並べる、という謎の構造になっています。概念的には、**図2**のようになっています。

typeは、共有リソースの種類に応じて値が変化します。具体的には、**表1**に示した値の組み合わせとなります。

たとえば、一般的なファイルやディレクトリを持つ共有リソースであれば、STYPE_DISKTREEがtype値となるでしょう。それに対して、先ほどtree_idを取得したIPC$であれば、STYPE_IPCとSTYPE_HIDDENの論理和を取った値(0x80000003)がtype値となるでしょう。

これで共有リソース一覧が取得できるようになりましたが、本当はこれだけでは不十分です。1回のレスポンスに共有リソース一覧が収まらなかった場合に、サーバはnt_statusとしてSTATUS_BUFFER_OVERFLOW(0x80000005)を返すことで、クライアントに「もっと情報あるよ」と伝えてきます。続きの共有リソース一覧は、SMB_COM_READ_ANDX(0x2e)メッセージによって取得することになります。

さあ、これで敵が持っている関所をすべて突破できました。敵の資産にいよいよアクセスできます。ここまでくれば、ユーザ認証結果の権限の範囲内で、ディレクトリやファイルを自由に作成したり編集したりすることが可能になります。大きな2つの壁であった「ユーザ認証」と「共有リソース一覧取得」を超えました。もう敵は丸裸も同然、勝ったも同然です。

ひどかったエラーコード

ここで1つ愚痴を言うならば、共有リソース一覧取得を試行錯誤していた中で、最も辛くきつかったこと、それは「nt_statusとしてINVALID_PARAMETER(0xC000000d)しか返ってこなかったこと」でした。

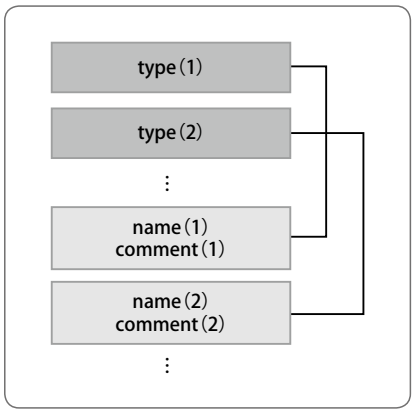
SMBプロトコルで規定された範囲内において何か間違っていたのであれば、その原因はnt_status値によってある程度推測できます。しかし、DCE/RPCプロトコルの規定範囲内で何か間違っていた場合、SMBプロトコルとしては「サブプロトコルの中で何か間違いが起きた＝サブプロトコルのバイト列のどこかが変」程度の認識となり、結果としてINVALID_PARAMETERしか返してくれない、というレスポンス内容になります。これだけ言われても、何がおかしかったのかさっぱりわかりません。

結局、自分が送ったリクエスト内容と、OS Xなどの「正しく動いているクライアント」の通信内容をパケットキャプチャして比較し、ひとつひとつ値をチェックしていきながら原因を探す、ということを日々行うしかありませんでした。

何かプロトコルを実装するには、とにかく「根気」が必要です。しかし、「辛いな」と思っていてはストレスが溜まる一方です。とくに仕様書がしっかりとそろっていない場合には、最初から「パケットキャプチャを楽しむ」という前向きな気持ちを持つことが大事だと痛感しました。

次号は解決編となります。そしてまとめて大団円へ！SD

▼図2　共有リソースの構造



▼表1　共有リソースの種類

定数	値	意味
STYPE_DISKTREE	0x00000000	Disk Drive
STYPE_PRINTQ	0x00000001	Printer Queue
STYPE_DEVICE	0x00000002	Communication Device
STYPE_IPC	0x00000003	Interprocess Communication
STYPE_TEMPORARY	0x40000000	Temporary Resource
STYPE_HIDDEN	0x80000000	Hidden Resource

新連載

コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

presented by
Japan Android Group
http://www.android-group.jp/

第1回 Android 6.0 Marshmallow誕生とコミュニティ

嶋 是一（しま よしかず）
NPO法人日本Androidの会 理事長

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

Androidは常に進化しています。2015年10月に、Androidの最新バージョン6.0である「マシュマロ（Marshmallow）」が正式に公開されました。Androidのコミュニティもまた、進化しています。最も普及したスマートフォンOSであるAndroidは、アプリケーションを開発する人、技術に興味を持つ人など、多くの人の注目を集めています。これらの人々が行う仕事や学校の枠を超えたコミュニティ活動により、より楽しくて面白い、エキサイティングなAndroidの世界が作り出されています。

今号から始まるこの連載では、このような「進化を続ける新しいAndroid」の情報を紹介するとともに、これを活用したさまざまな技術を、日本Androidの会で活躍するメンバーが中心となり紹介します。一緒になって、アプリケーション（以下、アプリと省略することもあります）やモノを「ともにつくりだして」いきましょう。

Android 6.0登場

今回のバージョンアップは「ファンを増やす？」

前述のとおり、Androidの最新バージョンである「Marshmallow」が登場しました。Nexus 5xとNexus 6Pという最新OS搭載の端末も発売されました（**写真1**）。また、最新の開発環境であるAndroid Studio 2.0が続いて11月に公開

▼写真1 Marshmallow対応端末（左：Nexus 5x、右：Nexus 6P）

▼図1 Marshmallowバージョンとは

されています。このMarshmallowを一言で言うと「より便利で手放せない存在」になるバージョンです（**図1**）。

Googleとしては、今回のバージョンは使う人の「使い心地の良さ」に響く機能を集中して拡張しています。AndroidのユーザはAppleのiPhoneに比べると、「信者」と呼ばれる人は少ないように思います。使いやすくすることでAndroidのファンとなるユーザを増やしたい、これが今回のバージョンの意図でしょう。多く

の人が手放せなくなるマシュマロとなるべくAndroidが進化したのです。確かにマシュマロ、食べると美味しいです。少しあぶって食べると、離れられなくなるようなおいしさです。しかし、注意が必要です。100gあたり326kcalもあり、継続的に食べ続けると、どうやら健康的に問題を来してしまいそうですが……。これは冗談として、Android 6.0で搭載した機能を紹介しましょう。

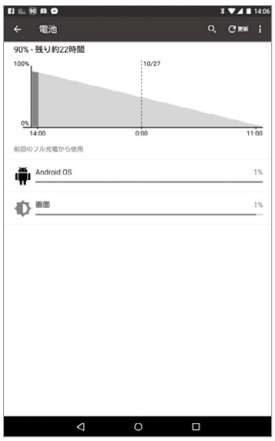
安心して使えるための機能

▶省電力待機機能「Doze」

スマートフォンをいつでも使えるように持ち歩くためには、電池持ちが良くなければなりません。にもかかわらず、フィーチャーフォン(二つ折りのガラケーに代表される携帯電話)に比べて、Androidのアプリは電池食いなものが多く、どうしても電池持ちの時間が延びません。電池食いのアプリが多くなる理由には、――たとえそれが自分以外のアプリの動作や端末全体の省電力を犠牲にするとしても――自分のアプリには電池を食ってしまうような機能やパワーをふんだんに使うことで利便性を上げ、高評価を得ることで多くの人にダウンロードさせたいと思う開発者の心理もありそうです。これぱかりは、「みんなでお行儀よく」にはならない状況が続いていました。

省電力の取り組みはAndroid 5.0から「プロジェクトボルタ」にて行われており、アプリごとの動作状況や電池消費量が確認できるようになっています(**図2**)。これにより、電池を消費しすぎているアプリを発見することができ、利用者が発見したときにはアンインストールの処置などを判断できるようにしていました。これはあくまでも開発者の改善や、利用者の判断に委ねているだけでした。

この野放しを改めて、Android 6.0からは「Doze(ドーズ)」機能が搭載されました。もっと積極的にAndroid OSが省電力に動作するようになったのです。この機能はAndroid 6.0で動作するアプリケーションすべてに影響します。ソース修正の必要なく機能は有効になりますが、開発者は正しく動作することを確認する必要があります。



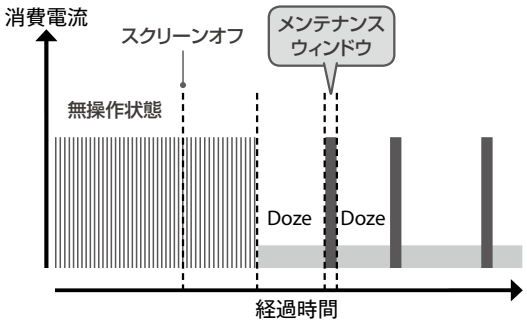
▶**図2**
アプリの電池消費確認画面

これまでの一般的な動作として、ユーザの操作がなくなり一定時間経つと、画面が消灯して黒くなります。消えることで、不使用時の無駄なディスプレイ電力を節約できます。さらに画面が消えた状態では、とくに処理するもの(ユーザ操作やタイマーによるアプリの処理、電話着信など)がない場合、CPUやTX/RX(電波の送受信回路)などもオフにして、「より高い省電力の状態」に移行します。この状態が長ければ長いほど、電池の持ちが良くなります。

しかし現実には、黒く消えた画面の裏でもメッセンジャー系のSNSアプリなど、数秒に一度ネットワークの確認を行うものもあり、期待したほどは端末が「より高い省電力の状態」に入れません。もちろんアプリ側も電池食いがばれると評価が悪くなる可能性があるため、省電力を考慮した設計がされてきています。しかし、定期的にネットワークを確認するアプリが複数入っていると、どうしても効果的にネットワークのアクセス頻度を減らすことができませんでした。

Dozeモードとは「寝ている状態」です。端末の操作がなくなり、充電しておらず、画面が暗く(スクリーンオフ)なると、一定時間でDozeモードに入り「より高い省電力の状態」なります(**図3**)。これまでのAndroidではこの状態で、

▼図3　Dozeモードの説明



http://developer.android.com/intl/ja/training/monitoring-device-state/doze-standby.html からの引用

アプリからネットワークなどの処理要求があると、都度「より高い省電力の状態」を解除して処理を行っていました。しかしAndroid 6.0のDozeモードでは、アプリから処理を要求されても動きません。その代わり、一定時間に一度だけ“メンテナンスウィンドウ”という時間を設けてあり、このタイミングに合わせて複数の処理をまとめて行います。終わったら再びDozeモードに入ります。積極的にアプリを止めることにより、省電力が実現します。電池の持ちが通常の2倍程度良くなると言われています。

たとえば、端末にインストールした2つのSNSアプリがおのおの10分に1回ネットワークに確認していた場合、これまでは平均して5分に1回端末が起き上がっていた計算になりますが、このDozeのしくみによって、10分ごとのタイミングでまとめて処理するため、10分に1回のネットワークアクセスで済むようになるというわけです。

▶実行時パーミッション

信頼できないアプリに電話帳のデータの利用許可を与えると、インターネットに送付して悪用される危険性があります。これまでのバージョンには、Androidのアプリをインストールするとき、「パーミッション」という機能でアプリが利用する機能やデータを提示し、利用者へ許可を求めるしくみがあります。信頼できないと判断すれば、インストールを中止させることができました。しかし多くのユーザは、内容を確認せずに「許可して」インストールしてしまい、セキュリティの脅威にさらされていました。

これを改善するために、センシティブなパーミッションだけは、実際にアプリがデータにアクセスしたり、センシティブな動作をするときに許可を求めるようになりました。たとえば、アプリが実際に電話を発信する直前に「電話発信(CALL_PHONE)」の許可を求められます。カレンダー、カメラ、電話帳、マイク、通話履歴、SIP、SMS、センサー情報、外部ストレージの利用などが、このような扱いになります。

会社で配布されるAndroid端末を、会社のカレンダー(スケジュール)と同期させている人もいると思います。あるいは他人に絶対知られたくない予定を管理している人もいるでしょう。旅行宿泊予定アプリや、グルメ予約アプリのような第三者のアプリからカレンダー同期を要求され、中身が読まれてしまうのを、これまで(利用断念以外に)防ぐ方法がありませんでした。この実行時パーミッションのしくみが加わったことで、個人情報を利用するアプリをより安心して使えるようになります。詳細の動作については次回の本連載で紹介予定です。

▶アプリケーションの自動バックアップ

Android端末を乗り換えたり、端末のオールリセットを行って問題になるのは、端末内部情報の消失です。多くのデータはクラウド側にあるため、Gmailやスケジュールデータなどで問題になることは少ないでしょう。また、端末にインストールしてあるアプリ本体もバックアップと復元が可能です。しかし“アプリのデータ領域”は対象外でした。Android 6.0からは、この領域もバックアップできるようになりました。ただし25MB上限で、ユーザのGoogle Driveの領域を用いるという制限があります。これにより、アプリ内部の設定や、一部のゲームのスコアなどもバックアップ可能になります。

使い勝手の向上

使い勝手を向上させる改善も行われています。主なものをかいつまんで紹介します。

▶指紋認証

Android標準で指紋認証の機能が搭載されました。アプリケーションから機能を呼び出して認証部分の利便性を向上させることができます。

▶アプリとリンクの紐付け

URLを指定する際に、起動するアプリケーションを選択可能になっています。これをドメインごとに設定できるようになります。

▶共有の向上

共有(Share)を行ったときに出てくる共有メニューの中に、特定のアプリのユーザ名やグループなどを表示させることができるようになります。これによって、共有先をより明確にした情報の共有ができるようになります。

▶音声操作

Voice Action APIが搭載され、アプリを音声コマンドで操作できるようになります。

▶アシストAPI

Android 6.0から搭載されたNow on Tap機能は、ホームボタンを長押しすることで、現在起動しているアプリの情報を読み取り、Google Nowがそのアシスト(検索や電話の促し)を表示する機能です。この有効/無効や、アシスト内容を設定するAPIです。

▶テーマ設定

テーマによる着せ替え機能が搭載されました。

歴史から見たAndroid 6.0

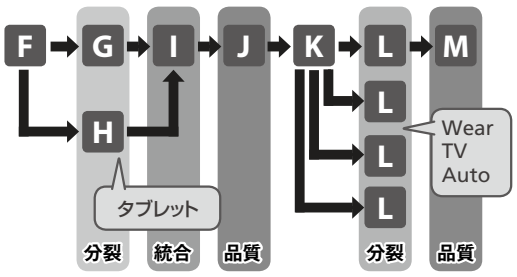
Androidのバージョン名(コードネーム)は、お菓子の名前で、かつ1文字目がアルファベットの順で決まっています。その、バージョンとバージョン名称の一覧は**表1**のとおりです。

これらのバージョンの歴史から見ると**図4**のとおりとなり、いくどもさまざまなプラットフォームに分裂、統合を繰り返しながら進化を続けています。とくに「Froyo」から、スマートフォン版の「Gingerbread」と、タブレット版の「Honeycomb」にOSが分裂したときがあり、次のバージョンの「Ice Cream Sandwich」で統合された経緯があります。そして、この次のバージョンの「Jelly Bean」では、プロジェクトバターと呼ばれる画面のスクロールの「ぬるぬるさ」を、iOSのスムーズさに負けないモノにするためにとり組んだバージョンとして有名です。つまり、統合したバージョンの後、使いやすさや品質向上を狙ったバージョンといえます。このあと「KitKat」の時代にAndroid Wearが発表されま

▼表1 Androidのバージョンとバージョン名称の一覧

頭文字	バージョン	バージョン名称(コードネーム)
C	Android 1.5	Cupcake
D	Android 1.6	Donut
E	Android 2.0/2.1	Eclair
F	Android 2.2	Froyo
G	Android 2.3	Gingerbread
H	Android 3.0/3.1/3.2	Honeycomb
I	Android 4.0	Ice Cream Sandwich
J	Android 4.1/4.2/4.3	Jelly Bean
K	Android 4.4	KitKat
L	Android 5.0/5.1	Lollipop
M	Android 6.0	Marshmallow

▼図4 Androidのバージョンの進化



した。「Lollipop」では、スマートフォンだけでなく、Wear(スマートウオッチ)、TV(テレビ)、Auto(自動車)にも対応するOSとして公開されました。その背景には、Androidに対抗するOS陣営が、スマートフォン以外の品目に向けてプラットフォーム化を広げていたことが一因となっています。そういう意味で、今回のMarshmallowは分裂後の、使い勝手向上や品質を向上するバージョンという位置づけになります。そのため、安心して使える機能や、ユーザビリティの向上を追加し、「より便利で手放せない存在に」となっていると思われます。

なお、アプリをこれまで作成してきた人にとって一番気になるのが、新しいAndroidバージョンでも問題なく動くかどうか、でしょう。基本的には新しいバージョンでも動作するはずですが、非推奨のAPIなどを利用していると、新しいAndroidバージョンでAPIが削除されて、機能が利用できなくなることがあります。

Lollipopのときには、かなり多くのAPIが追加、変更、削除、非推奨となりましたが、今回のMarshmallowでは、それが半分近くまで少なくなっていますので、影響は限定的です。しかし、その中でも気を付けるべきは、MarshmallowからHTTPを用いた通信を行うApache HTTP Client(クラスorg.apache.http)がまるごと削除された点です。org.apache.http自体はGingerbreadのバージョンから非推奨となっており、HttpURLConnectionを使うよう促されていました。無視してきた人にツケが回った感じです。とはいえ、どうしても利用したい場合は、org.apache.http.legacyのライブラリを利用するよう、libsに追加することもできます。

開発者とコミュニティ

コミュニティってどんなところ?

開発者にとってコミュニティは、どんなところでしょうか? またはどのようなことを期待されますか? 開発での悩みを聞いてもらいたい、壁にぶち当たったときに教えてもらいたい、自分の作品やアイデアを知ってもらいたい、自分の活動(Webページでもハッカソンでも)を宣伝して広めたい。あるいは一人でできないことを実現してみたい、仕事をもらいたい、最新の情報を知りたい、自分が団体を率いてみたい(隊長になってみたい)、などでしょうか。

これらすべて正しく、そして、すべてが正解というわけではありません。コミュニティへの参加の動機としてはいずれも正しいですが、コミュニティはすべてを与えてくれるものではありません。自分で活動した分に応じて、それらのいくつかは満たすことができます。しかし、動機や期待にかかわらずコミュニティに参画すると、それ自体が「楽しいこと」であるのがわかります。参加をするだけで、いきなり世界が広がります。技術の面や、アイデアの面や、時には他人の持つユーモアのあるバカさ加減で、自分の世界を広げてくれます。初めは驚きとともに、入ってくる情報に戸惑ってしまい、身動きができないかもしれません。しかし、そこから自分の興味の琴線に触れる情報や技術に対して、少しずつ活動(参画)を始めると、自分のモチベーションが上がってきます。つまり楽しくなりはじめます。そういう心のタッチポイントの機会に出会えるようになる、それこそがコミュニティのご利益ではないかと思っています。

世の中には数々のコミュニティがあります。その目的と性格、運営方法により、得られる内容も、より楽しいと思う方向性も違ってきます。開発を黙々と行う開発主体のコミュニティもあれば、新技術をどんどんと味見のように試していく初物好きのコミュニティもあります。いずれにしても、これらを体験するのに、はじめの一歩、コミュニティへの参加が必要です。勇気を出して、その一歩を踏み出してみませんか。

コミュニティ活動

Androidの進化と共に、Android界隈のコミュ

ニティ活動も進化してきました。Androidも登場して8年となりますので、さまざまなコミュニティでの活動が行われてきました。コミュニティ活動に参加する一番の方法は、コミュニティが公募しているメーリングリストや、SNSのグループに加わることです。しかし、もっとお勧めするのが、コミュニティが開催する、勉強会やイベントに参加することです。こちらの場合は、足を運ぶ必要があるので、少しだけの時間と勇気が必要となりますが、Androidの情報を得られるだけでなく、コミュニティの雰囲気や、今多くの人が持っている興味などを知ることもでき、より楽しむことができると思います。その後に、メーリングリストやSNSに加わり活動することもできます。

▼写真2　ABC 2015 Summer

コミュニティに参加して面白いのは、必ず「日本一」を目指す人がいるところです。たとえば、今回のようにAndroidのバージョンが発表されると、いち早くその機能を搭載したアプリを公開する人、または、いち早くその機能の解説記事を執筆公開する人がいます。そのような人の情報は、いち早くコミュニティの中を駆け巡り、その早さの技術力を称えられつつ、周りの人はその情報で、一般の人より早く技術を学べる場となっているのです。

これらAndroidコミュニティの1つである、日本Androidの会は、会員2万2千人を擁する、世界でも最大級のAndroidユーザコミュニティです[注1]。活動の多くは、メーリングリストの情報交換、そして首都圏で実施する毎月の無料勉強会、そして年2回実施するAndroidの祭典となるイベント「Android Bazaar and Conference（以下、ABC）」の開催です。無料月例の勉強会は100～200名程度で、毎月テーマを決めて、平日の夜19時～21時に講演会形式で実施します。ABCは、大学や商業施設の会場を棟ごと借り、休日1日を使って行うイベントです。Bazaar（展示会場）とConference（講演会場）の2つに分かれており、1,000～3,000名程度の来場者があります（**写真2**）。前回のABC 2015 Summer[注2]では、Android Mプレビュー版が登場した後だったため、技術解説講演があったり、ドローンを開発するための集中講座が催されました。また、会場では数々のIoTデバイスとAndroid端末の連携や、Pepperを利用したデモストレーション、Android Wearの作成アプリ紹介、Beacon技術の展示などにあふれかえり、おおいに盛り上がりました。

また全国に37地方支部があり、地域での勉強会やハッカソンなど、地域に根ざした活動も行われています。また、ワーキンググループ／部活動として、テーマを決めて活動しているメンバーもいます。「VR部」「福祉部」「学生部」「ロボットサミット」「Web部」「Unity部」「アド部」「Open Beacon Field Trial」「ドローン部」など、盛り上がっています。

また、もっと密接にコミュニティに参加したい人には、コミュニティ自体の運営や、企画されるイベントの実行委員に参加する方法もあります。コミュニティは自身がかかわればかかわるほどに、その面白さも情報も増えます。運営されている方に相談してみるのもよいでしょう。

コミュニティの進化

Androidのコミュニティも進化しています。単純にAndroidに関係するコミュニティが増えているとか、規模が大きくなっているとか、そ

注1）http://www.android-group.jp/

注2）http://abc.android-group.jp/2015s/

ういう進化の話もありますが、集まる人たちの持つ興味の進化も顕著です。Androidの創成期には、アプリ開発方法自体が興味となっており、とにかく情報が少なく、書籍も数えるほどしかありませんでした。また、海外のWebを探してもAndroidの情報が集まらないため、実際に開発したり、新しい機能を試したり、限られたGoogleやAndroidの情報を持っている人を求めて、人が集まってきていました。なんといっても、1年に2回も3回もバージョンアップが繰り返される状況だと、Android開発者としても食いついて振り落とされまいと必死に情報を求めるわけです。そのためハンズオンや勉強会がコミュニティの基本的な活動でした。

AndroidバージョンのGingerbreadのあたりから日本の書籍も増え、単純な開発のノウハウは共有されるようになりました。ドキュメントに書かれていない「新しい技術情報」や、Androidに取り入れられる「目立つ機能」に、多くの人の興味が向くようなりました。このあたりから、アイデアで自分のアプリを広く公開する人などが出はじめます。そういう、技術の牽引メンバーたちが、イベントや講演会などで活躍するようになりました。

最近の進化傾向としては、KitkatからはAndroid自体のバージョンアップのペースも落ち着いており、Android自体に搭載される機能の注目度も以前ほどはなくなってきました。そうすると、Android自体の興味だけでなく、その周辺技術の、Web、Beacon、IoTデバイス接続などに興味が移って来ました。とくに、Lollipopからは、Android Wear、Android TV、Android Autoのプラットフォームが登場し、“スマートフォンのAndroid”ではない領域の興味が大きくなっています。コミュニティはこのような興味に合わせて、新しい活動を作り出したり、終わらせたりして進化しています。

最近感じられるのは、人の興味が1つのコミュニティの枠に収まらなくなっている点です。その興味は時として、ほかのコミュニティの活動と重なることもあります。そのため複数のコミュニティに所属する人もいますし、あるいはコミュニティの人のつながりの中で、他の興味のあるジャンルのコミュニティとコラボレーションし、共同で勉強会を開いたり、交流を行うケースが増えてきています。まさに、これが新しいコミュニティの進化だと思います。Androidの進化、コミュニティの進化を体感することで、あなたのAndroidアプリ開発や技術的興味がより促されるでしょう。

まだAndroid未経験な方は、この魅力的なAndroid開発をはじめてみませんか？ SD

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 Spring

開催日：2016年3月12日
場所：東京都 青山学院大学 青山キャンパス

日本Androidの会が主催するABCの2016年春(spring開催)のイベント。表参道近辺で実施予定。開発者を中心にしたAndroidの総合イベント。最新のAndroid情報や、開発成果物を発表するイベント。また、Android周辺技術の展示も数多い。企業の展示もあり、端末から、アプリ、デバイス、サービス、クラウドまで、展示の多様と混沌さも楽しめる無料イベント。

http://abc.android-group.jp/

ABCD 2015 Kanazawa

- Android Bazaar and Conference Diverse -
開催日：2015年11月22日～23日(終了)
場所：石川工業高等専門学校 (石川県河北郡津幡町北中条タ1)

石川工業高等専門学校(石川県河北郡津幡町北中条タ1)日本Androidの会 金沢支部が開催するABCの地方開催イベント。展示と講演として、Androidだけでなく、VRについて内容が充実。クラウドファンディングも活用し目標達成！

http://abcd2015k.strikingly.com/

DroidKaigi 2016

開催日：2015年2月18日～19日
場所：東京工業大学 新岡山キャンパス

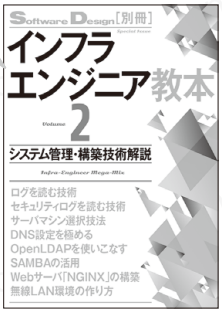
DroidKaigi実行委員会主催のエンジニアが主役のAndroidカンファレンス。Android技術情報の共有とコミュニケーションを目的に開催。学生割引あり。

https://droidkaigi.github.io/2016/

Software Design plus

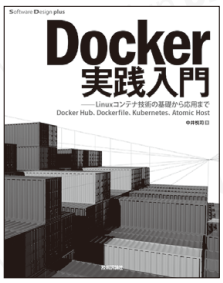
Software Design plusシリーズは、OSとネットワーク、IT環境を支えるエンジニアの総合誌『Software Design』編集部が自信を持ってお届けする書籍シリーズです。

最新刊！



Software Design 編集部 編
B5判・344ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-7782-3

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



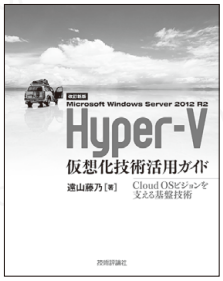
中井悦司 著
B5変形判・200ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-7654-3

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

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

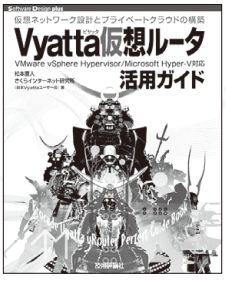
勝俣智成、佐伯昌樹、原田登志 著
A5判・288ページ
定価 3,300円（本体）+税
ISBN 978-4-7741-6709-1

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



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

寺島広大 著
B5変形判・440ページ
定価 3,500円（本体）+税
ISBN 978-4-7741-6543-1

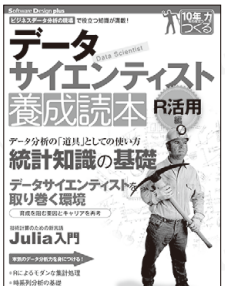


松本直人、さくらインターネット研究所（日本Vyattaユーザー会）著
B5変形判・320ページ
定価 3,300円（本体）+税
ISBN 978-4-7741-6553-0

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

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

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



養成読本編集部 編
B5判・164ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7057-2

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

Javaエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6931-6

JavaScriptエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6797-8

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

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

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

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

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

【改訂新版】サーバ／インフラエンジニア養成読本 管理／監視編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6424-3

【改訂新版】サーバ／インフラエンジニア養成読本 仮想化活用編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6425-0

【改訂新版】サーバ／インフラエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6422-9

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21回 定型文を瞬時に入力　yasnippetの実力

今回は前回少し紹介した、穴埋め式で定型文を入力できるパッケージ「yasnippet」について深堀りしていきます。スニペット作成のチュートリアルから、auto-yasnippetパッケージによる即席スニペットの使い方までを解説します。

スニペット展開の王道 yasnippet

ども、るびきちです。11月8日にJohn Wiegley氏がEmacsのメンテナに就任し、Emacs25リリースへ向けての動きで盛り上がっています。彼は前世紀からたくさんのelispプログラムにかかわり、長年熱心に活動している人物ですので、これからのEmacs界が楽しみです。執筆時点で25.1への仕様が凍結されたので、そう遠くない日にリリースされるでしょう。

前回は基本的な入力支援機能に触れてから、yasnippetという強力なスニペット(テンプレート)展開パッケージを紹介しました。yasnippetはMELPAダウンロードランキングでベスト10入りするほど定番になってきました。これを使えば穴埋め式で定型文を確実に入力できるので、コーディングや文書作成が捗ります。インストール後の初期設定は次のとおりです。

```
(yas-global-mode 1)
;;; スニペット名をidoで選択する
(setq yas-prompt-functions'(yas-ido-prompt))
```

スニペットを定義する

yasnippetはインストールした時点で各メジャーモード用にスニペットが用意されていますが、やはり自分で定義してこそ、使いこなしていると言えます。とくにコーディングの場面においては定型文入力の繰返しになります。関数、クラス、メソッドにはイディオムのような決まった使い方があり、それを登録することで確実に、効率よく入力できます。

スニペット登録例

前回(2015年12月号)登場したアドバイス定義のコードをスニペットの登録例にします。前回は完成形スニペットを例として示しましたが、そこに到達するまでの道のりを解説します。

リスト1は筆者が実際に使っている設定です。C-x v lでバージョン管理システムのログを表示し、そこでdを押したらM-x log-view-diffが実行されて前回のコミットとのdiffが表示されます。しかし、そのあとdiffを表示しているウィンドウを自動的に選択してしまうのが不満ですので、それを解消するアドバイスを書きました。アドバイスを定義するには、アドバイスの内容となる関数を定義し、advice-addでその関数を登録します。関数名は任意ですが、どの関数に対するアドバイスかを明確にするため、筆者は「**元の関数名--アドバイス名**」の形式にしています。その骨格だけを抜き出すと、

```
(defun log-view-diff--noselect (&rest them)
  )
(advice-add 'log-view-diff :after
            'log-view-diff--noselect)
```

のようになります。

登録の手順

◆3つのdirective

このリスト1をスニペットにしましょう。M-x yas-new-snippetを実行します。すると、「*new snippet*」バッファに切り替わり、name、key、bindingという3つのdirectiveが表示されます。この時点で新規登録用スニペットが展開されています(リスト2)。

nameはスニペット名なのですが、実際はスニペットの1行説明文で何を定義しているかを書きます。略語(key)を思い出せなくても、M-x yas-insert-snippetを使えばnameを手がかりにスニペットを展開できます。なので、スニペットに使われているキーワードをスペースで区切って羅列するというのは良いアイデアです。

keyはスニペットを展開する略語です。とくに使用頻度の高いスニペットに対しては、短くて覚えやすいものを設定しておくことで劇的に使い勝手が向上します。使用頻度が低いものについては、いずれ忘却の彼方へ追いやられるので適当に考えても良いでしょう。

bindingはそのスニペットを展開するキーバインドです。たとえばC-c C-i C-iを指定すれば、そのキーで展開できます。使わない場合はC-dで入力をキャンセルしてください。

◆スニペット本体を記述する

スニペット本体は# --以下の行に記述します。基本的にはここに記述した文字列がそのままスニペットになるのですが、スニペット展開の指令に使われる「$」と「`」、そして「\」そのものについては、それぞれ「\$」「\`」「\\」とエスケープする必要があります。リスト3の例ではエスケープ不要ですのでそのまま貼り付ければいいです。nameとkeyはそれぞれadvice-add、adviceと指定し、bindingは無指定にしました。

◆スニペットをテストする

ここでC-c C-tを押せばスニペットが正しく展開されるかテストができます。そのまま貼り付けた場合であってもエスケープ漏れの可能性があるので、テストすることをお勧めします。

場合によってはスニペット自体は正しくても、テスト展開でエラーになることがあります。スニペットにはelispの式を埋め込めるのですが、テスト時と実運用時では環境が異なるためです。たとえば、ファイル名を表す変数、関数のbuffer-file-nameはテストバッファではnilとなるため、elisp式展開部分ではerrorと表示されます。それでも、ほかの部分ではテストができるので役立たずではありません。テストでエラーが起きたときには元のバッファで展開してください。これでうまくいくのであれば問題ありません。

◆スニペットを登録する

無事にテストがうまくいったらC-x bなどで

▼リスト1　diffを表示したあとの挙動に関する設定

```
(defun log-view-diff--noselect (&rest them)
  (other-window -1))
(advice-add 'log-view-diff :after
            'log-view-diff--noselect)
```

▼リスト2　新規登録用スニペット

```
# -*- mode: snippet; require-final-newline: nil -*-
# name:
# key:
# binding: direct-keybinding
# --
```

▼リスト3　スニペット本体を記述

```
# -*- mode: snippet; require-final-newline: nil -*-
# name: advice-add
# key: advice
# --
(defun log-view-diff--noselect (&rest them)
  )
(advice-add 'log-view-diff :after
            'log-view-diff--noselect)
```

▼リスト4　スニペットに穴埋めを設定

```
# -*- mode: snippet; require-final-newline: nil -*-
# name: advice-add
# key: advice
# --
(defun ${1:log-view-diff}--${2:noselect} (${3:&rest them})
  $0)
(advice-add '$1 :${4:after} '$1--$2)
```

スニペットのバッファに戻り、C-c C-cで登録します。すると、「Choose or enter a table」というプロンプトが出て、登録するメジャーモードを尋ねてきます。多くの場合M-x yas-new-snippetを実行したバッファのメジャーモードとなるので、そのままRETで確定します。次に新規作成したスニペットについては「Looks like a library or new snippet. Save to new file」と尋ねてきますが、これもそのままyで確定します。これでスニペットの登録が終わり、元のバッファに戻ります。

◆穴埋めを設定する

この時点でadvice Tabと入力することで貼り付けたスニペットがそのまま展開されます。これはこれで使用例を貼り付けられるので役立つのですが、機能的には略語展開となんら変わりありません。スニペットがスニペットらしくあるためには穴埋めを設定してナンボです。

とはいえ穴埋めを設定するかどうかは、そのスニペットの使用頻度と相談すべきです。あまりにも使用頻度が低いと、穴埋め設定が面倒に感じてしまい、yasnippetに悪い印象を持ちかねないからです。使用例を貼り付けただけのスニペットでも、十分な場合があることも事実です。今回のアドバイスのスニペットのように、これからも使用されることが予想される場合は迷わず穴埋めを設定してください。

穴埋めは「**$数字**」($1、$2〜)あるいはデフォルト値付きで「**${数字:デフォルト値}**」と指定します。スニペットを展開すると、Tabを押すたびに数字の順番でカーソル位置が穴埋め位置に移動し、入力できるようになります。圧巻なのは、同じ番号の穴埋めを複数個置いたときで、入力するたびに該当する穴埋めの文字列が同時に変化することです。スニペット登録時、nameを入力すると同時にkeyにも同じ文字列が入力されたのもこの現象です。また、$0は特別な意味があり、スニペット展開終了後に置かれるカーソル位置を示します。

これらをふまえたうえで穴埋めを設定しましょう[注1]。コードから生まれたスニペットの場合は、デフォルト値はそのまま保持しておくと記憶をたどりやすいです。advice-addの行にも穴埋めに設定したlog-view-diffとnoselectが登場するので、それぞれ$1、$2と記述します(**リスト4**)。ほかにも「`」で囲んでelisp式を埋め込んだり、「**${数字:$$(yas-choose-value 文字列リスト)}**」で文字列の選択肢を表示できたりします。

auto-yasnippetで即席スニペット

普段の文字入力で起こる同じパターンの入力

yasnippetは入力をとても効率良くしてくれますが、それだけではあらかじめ定義されたスニペットでしか有効ではありません。普段の文字入力でも同じようなパターンを入力することはよくあります。たとえば次の3行を入力する場合を考えてみましょう(これは筆者が関わったelispプログラムの一部です)。

```
(key (plist-get args :key))
(switch (plist-get args :switch))
(before (plist-get args :before))
```

おそらく共通部分だけを書いてコピーし、異なる部分をあとで入力することを真っ先に思い付くことでしょう。

注1) 1度作成したスニペットの編集に移るには、M-x yas-visit-snippet-fileが便利です。

auto-yasnippetを使う

こういう場合にyasnippetの展開が使えれば便利ですが、たった一度の入力のためにスニペットを定義するのはあまりにもめんどうです。そこでauto-yasnippetパッケージによる即席スニペットを使えば、普段の文字入力においてもyasnippetの展開の恩恵が受けられます。M-x package-install auto-yasnippetでインストールしましょう。ついでにmykieパッケージもインストールしておけば、対になるコマンドを1つのキーに割り当てられて便利です。次の設定をしましょう。

```
(setq aya-create-with-newline t)
(mykie:global-set-key "C-x C-y"
  :default aya-expand :C-u! aya-create)
```

yasnippetのスニペットは読みやすいですが、即席で使うにはいささか煩雑です。そこでauto-yasnippetではより入力しやすいシンプルな構文を採用し、内部でスニペットに変換しています。

お手軽1行即席スニペット

auto-yasnippetには2つのタイプの即席スニペットがあります。お手軽タイプは現在行を即席スニペットにする機能限定版です。regionが設定されていない状態、かつ穴埋め部分が1つ、かつその行に「~」が含まれていないときに使えます。先ほどのplist-getの場合がまさにこのケースです。お手軽タイプは穴埋め部分を「$」と入力して使います。次のように入力して、

```
($ (plist-get args :$))
```

行末にてC-u C-x C-y(aya-create)を実行すると「$」が消えてスニペット展開状態になるのでkey Tabと入力すればいいです。すでにこの状態で即席スニペットが登録されたので、次の行にてC-x C-y(aya-expand)で展開できます。同じように展開されるのでswitchと入力し、同様にbeforeも入力します。

フルバージョン即席スニペット

お手軽タイプは確かに便利ですがキーボードマクロに毛が生えた程度のものにすぎません。穴埋め部分が複数個ある場合や即席スニペットが複数行に渡る場合はフルバージョンを使う必要があります。フルバージョンは穴埋め部分の前に「~」を付けるか「`'」で囲みます。「~」のあとは英数字、ハイフン、アンダーバーまでが穴埋め部分とみなされます。それら以外を含む場合は「`'」を使います。plist-getをフルバージョンにすると、次のどれかになります。

```
(~key (plist-get args :~key))
(`key' (plist-get args :`key'))
```

C-u C-x C-yを押すと穴埋め部分を指定する記号が削除され、即席スニペットが登録されます。あとは同様にC-x C-yで展開していきます。

このように穴埋め部分の文字列が同じ場合は、同じものが入るとみなされます。フルバージョンでは穴埋め部分を複数個指定できるようになった代償として、同じ文字列を入力する必要があります。明らかに穴埋め部分が1つの場合はお手軽タイプが楽です。フルバージョンが本領を発揮するのは、複数行に渡る即席スニペットです。この場合はregionを指定してからC-u C-x C-yで登録してください。

◆ ◆ ◆

筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指すべく日々更新しています。手元でgrep検索できるよう全文をGitHubに置いています。またEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでも御答えします。「こんなパッケージ知らない？」「挙動がおかしいからなんとかしてよ！」はもちろんのこと、自作elispプログラムの添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD　登録はこちら➡http://www.mag2.com/m/0001373131.html

一歩進んだ使い方のためのイロハ

Vimの細道

第4回

mattn
twitter:@mattn_jp

VimでWeb開発

Web開発にはさまざまな言語が必要ですが、言語ごとにいちいち環境を変えるのは面倒ですよね。今回は、Webアプリを Vim 上だけで開発することをゴールに、JavaScript → HTML → Riot.js → CSS → PostgreSQLの順で、それぞれの段階における便利なVimプラグインを紹介していきます。

昨今のWeb開発事情とVim

前回、前々回と、VimでJavaを扱う方法やTipsを紹介しました。Vimの強力なカスタマイズ性によって、プラグインを導入したり簡単な設定を行ったりするだけで、IDE に匹敵する機能性を保ちながらも、重くならない開発環境を得ることができました。Eclimを使うか、それとも選び抜かれたプラグインを導入するかはみなさんしだいです。自分にあったほうを選んでみてください。

さて、昨今ではWebアプリケーション(以下Webアプリ)の開発者の数が世界的に多くなっているようです。石を投げればWeb系エンジニアに当たると言っていいほどWeb開発の需要が増えてきました。筆者もいろいろな仕事に携わっていますが、その中でもWebアプリ開発は大きな位置を占めています。今やWebアプリ開発のスキルは当然のものとなってきており、中にはWeb開発スキルだけでエンジニアをやっている、なんて人も意外と多いようです。

JavaScriptといえば、昔はHTMLを制御するための記述言語でしかありませんでしたが、Node.jsの登場により、今ではサーバサイドもクライアントサイドも開発できるプログラミング言語となりました。極論で言ってしまえば、JavaScriptだけ覚えておけばWebアプリ開発ができる、そんな時代になったと言ってもいいでしょう。巷にはaltJSと呼ばれる「JavaScriptで実装された別言語」が流行し、JavaScript自身もES6(ECMAScript 6th Edition)へと進化が進んでいます。

今回はJavaScriptをメイン言語としたWebアプリの開発方法を、Vimを使って説明していきたいと思います。

開発開始！

準備として、Node.jsをインストールしておいてください。執筆時点ではバージョン5.0.0がリリースされていますが、とくに新しいバージョンである必要もありません。筆者は0.10.40で開発しています。何を作ろうか迷いましたが、SPA(Single Page Application)のtodoアプリを作りたいと思います。今回は「Riot.js」というユーザインターフェースライブラリを使います。

VimでJavaScriptを書く

まずexpress[注1]でファイルサーバを作ります。もちろんVimはデフォルトでJavaScriptのシンタックスハイライトやインデントをサポートし

注1) URL http://expressjs.com

ています。しかし、デフォルトのインデントはあまり賢くありません。またシンタックスハイライトにおいてもいくらかのキーワードがハイライトされないという問題があります。

そこで「pangloss/vim-javascript」をインストールします。プラグインマネージャ「vim-plug」をお使いであれば、**リスト1**をvimrcに追加し、再起動後に:PlugInstallコマンドを実行します。本記事では以降、vim-plugの設定方法のみを記述します。そのほかのプラグインマネージャをお使いの方は個々のマニュアルを参照してください。

▼リスト1　vim-javascript導入のための設定

```
" https://github.com/pangloss/vim-javascript
Plug 'pangloss/vim-javascript', { 'for': 'javascript' }
```

▼リスト2　ファイルサーバのコード

```
var express = require('express'),
    app = express();

app.set('port', (process.env.PORT || 5000));
app.use(express.static(__dirname + '/assets'));

app.listen(app.get('port'), function() {
  console.log('app is running on port', app.get('port'));
});

// vim:set et sw=2 cino=>2,j1,J1:
```

▼リスト3　myhere/vim-nodejs-complete導入のための設定

```
" https://github.com/myhere/vim-nodejs-complete
Plug 'myhere/vim-nodejs-complete', { 'for': 'javascript' }
```

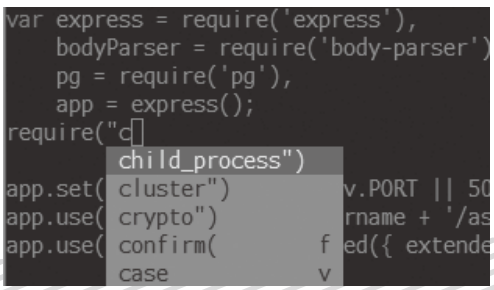
まずapp.jsをVimで開き、ファイルサーバのコード(**リスト2**)を書きます。一番下の行にあるおまじないのようなもの、これはモードラインと言い、このファイルを開いたときにVimの設定を行う機能です。この例では次のように設定されます。

- et(expandtab)：タブ文字をスペースで置き換える
- sw=2(shiftwidth)：インデントを2スペースとする
- cino=>2,j1,J1(cinoptions)：改行後のインデントを2スペース、無名関数や無名クラスの後を段下げ

詳しくは:help modelineで参照してください。JavaScript、とくにNode.jsのようにコールバック関数が多く出現する言語では、タブ文字[注2]の代わりにスペースでインデントすることが多く、また無名関数によってネストされることが多いので、なるべく少ないインデント幅が好まれます。

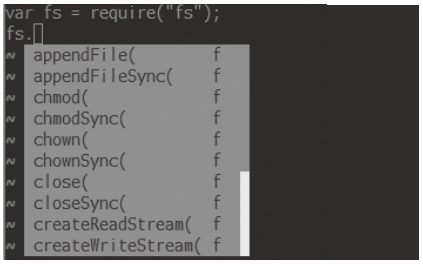
また、**リスト2**のような簡単なスクリプト程度であれば、入力補完が登場する場面はそれほどなかったかもしれませんが、今後このソースコードがどんどん膨らんでいったとしても効率的にプログラミングするためには、補完機能があったほうが便利です。JavaScriptに特化した入力補完プラグインもありますが、「myhere/vim-nodejs-complete」を導入するとNode.jsに適した入力補完が得られます。**リスト3**を設定してインストールしてください。これ以外の設定はとくに必要ありません。このプラグインを導入すると、**図1**のように、requireするパッケージ名を補完できます。また、このプラグインはrequireしたパッケージのメンバ補完もできるため、たとえば**図2**のようにfsモジュールだけに絞った入力補完も行えます。

▼図1　Node.jsの入力補完



注2）設定により、見た目の幅を変えることもできます。

▼図2　requireしたパッケージのメンバ補完



VimでHTMLを書く

次にindex.html(**リスト4**)を書きます(とは言っても、Riotのカスタムタグを読み込むための土台でしかありませんが)。今回は誌面の都合上、リッチなHTMLは書きませんが、もう少し凝ったHTMLを書くのであれば「**mattn/emmet-vim**」を導入することでサクサクとHTMLを書けるようになります(**リスト5**)。Emmetは HTML/CSSをコーディングする際にとても便利な記法で、たとえば次のように入力し、

```
ul>li*3>a{hello $$$}
```

<C-y>,(Ctrl-Yのあとにカンマ)をタイプすると、**リスト6**のように展開されます。さらにビジュアル選択でテキストの一部分、たとえば**リスト6**の001をビジュアルモードで選択し<C-y>,をタイプすると「Tag:」というプロンプトが表示されます。bを入力すると、

```
<li><a href="">hello <b>001</b></a></li>
```

のように、選択していた部分がで囲まれます。もう少し応用してみましょう。

```
桃太郎
浦島太郎
寿司太郎
```

▼リスト4　index.html

```
<!doctype html>
<html>
<head>
  <title>Riot todo</title>
</head>
<body>
<todo></todo>
<script src="todo.tag" type="riot/tag"></script>
<script src="https://cdn.jsdelivr.net/g/
riot@2.0.14(riot.min.js+compiler.min.js)"></script>
<script>
riot.mount('todo');
</script>
</body>
</html>
```

を行で選択(Vをタイプしてjjで3行選択)し、<C-y>,をタイプ、「Tag:」のプロンプトで、

```
ul>li*>a.link[http://www.google.com/search?q=$#]{$#}
```

のように入力します。すると、**リスト7**のようなリンク集が簡単に作れてしまいます。慣れてくると、このEmmet構文がスラスラと出てくるようになるので、興味のある人はぜひ使ってみてください。Emmet記法について詳しく知りたい方はオフィシャルサイト[注3]を参照ください。

注3) URL http://emmet.io

▼リスト5　mattn/emmet-vim導入のための設定

```
" https://github.com/mattn/emmet-vim
Plug 'mattn/emmet-vim'
```

▼リスト6　Emmet記法の展開

```
<ul>
    <li><a href="">hello 001</a></li>
    <li><a href="">hello 002</a></li>
    <li><a href="">hello 003</a></li>
</ul>
```

▼リスト7　リンク集を簡単に作成

```
<ul>
    <li><a class="link" href="http://www.google.com/search?q=桃太郎">桃太郎</a></li>
    <li><a class="link" href="http://www.google.com/search?q=浦島太郎">浦島太郎</a></li>
    <li><a class="link" href="http://www.google.com/search?q=寿司太郎">寿司太郎</a></li>
</ul>
```

▼リスト8　nicklasos/vim-jsx-riot導入のための設定

```
" https://github.com/nicklasos/vim-jsx-riot
Plug 'nicklasos/vim-jsx-riot'
```

▼リスト10　JulesWang/css.vim導入のための設定

```
" https://github.com/JulesWang/css.vim
Plug 'JulesWang/css.vim', { 'for': 'css' }
```

Vimで Riot.jsを書く

さて、次はRiot.jsのカスタムタグを作っていきます。入力ボックスと追加ボタン、「対応済み」を示すチェックボックスが付いたtodoリストを**図3**のように配置した構成とします。

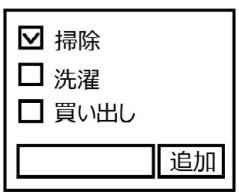
まずはRiotによるビューを作成します。Riot.jsはRiotタグの中にJavaScript/JSXを埋め込んだものですので、JSX専用のプラグインでは機能しません。そこで「**nicklasos/vim-jsx-riot**」というRiot.js専用のプラグインを導入します(**リスト8**)。これにより、tagファイルがシンタックスハイライトされるだけでなく、カスタムタグ内に書かれているスクリプトがJavaScriptとして認識されます。先に紹介したmyhere/vim-nodejs-completeを導入していれば、インラインで書かれたスクリプトに対して入力補完ができます(**図4**)。

基本的な動作をtodo.tag(**リスト9**)に記述しました。これだけでも追加ボタンやチェックボックスが機能するため、紙芝居の確認ができます。

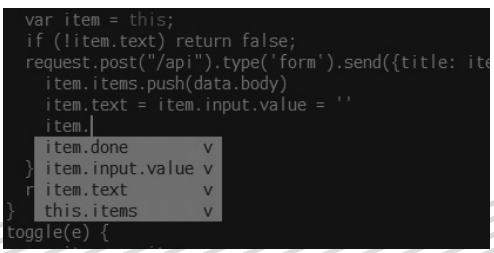
▼リスト9　Riotのタグファイル

```
<todo>
  <h3>{ opts.title }</h3>
  <ul>
    <li each={ items }>
      <label class={ completed: done }>
        <input type="checkbox" checked={ done } ➥
onclick={ parent.toggle }> { title }
      </label>
    </li>
  </ul>
  <form onsubmit={ add }>
    <input name="input" onkeyup={ edit }>
    <button disabled={ !text }>Add #{ items.➥
length + 1 }</button>
  </form>
  <script>
    this.items = opts.items || []
  edit(e) {
      this.text = e.target.value
  }
    add(e) {
      var item = this
      if (!item.text) return false
      item.items.push({ title: this.input.value })
      this.input.value = ''
      return false
    }
    toggle(e) {
      var item = e.item
      item.done = !item.done
      return true
    }
  </script>
</todo>
```

▼図3　todoリスト



▼図4　Riotのtagファイルでも入力補完



Vimで CSSを書く

次に、見た目を変えるためにCSSを作成します。ただし、ここで問題が発生します。Vimに同梱されているCSSシンタックスハイライトは、CSS3をすべてサポートしていないのです。CSS3のシンタックスハイライトを有効にするには、「**JulesWang/css.vim**」を導入します(**リスト10**)。開発者のJules Wang氏は、Vimに同梱されているCSSシンタックスハイライトのメンテナで、このプラグインはVim 7.0がリリー

▼リスト11　hail2u/vim-css3-syntax導入のための設定

```
" https://github.com/hail2u/vim-css3-syntax
Plug 'hail2u/vim-css3-syntax', { 'for': 'css' }
```

▼リスト12　gorodinskiy/vim-coloresque導入のための設定

```
" https://github.com/gorodinskiy/vim-coloresque
Plug 'gorodinskiy/vim-coloresque', { 'for': 'css' }
```

スされたあとも、細かくCSS3への対応を行った成果が含まれています。

また、「hail2u/vim-css3-syntax」を導入すると（**リスト11**）、W3Cにて現在ワーキングドラフトになっているCSSについてもシンタックスハイライトされるようになります。

これでもかなり便利になったのですが、もう少しだけ便利にしたいと思います。たとえばCSSを編集していて配色を変更したときに、毎回ブラウザをリロードするのは面倒ですよね。かと言って色見本を開くのも面倒ですし、ブラウザにフォーカスを移すのすら面倒なときもあります。そこで「gorodinskiy/vim-coloresque」を導入します（**リスト12**）。このプラグインを導入すると**図5**のようにCSSの配色指定の部分が実際の色で表示されます。色の値を変更すると、保存せずともリアルタイムで配色が変更されるため、デザインに要する時間を節約できます。

VimでSQLを書く

さて、この紙芝居を実際にデータベースと接続して機能させるためにapp.jsを編集します。データベースにはPostgreSQLを使います。DDL（Data Definition Language）としてschema.sql（**リスト13**）を作成します。一発で決まれば良いのですが、そう簡単には行きません。何度かテーブルを壊しつつ、完成に近づけることになります。Vimでschema.sqlを開きつつ、できたと思ったら「thinca/vim-quickrun」（**リスト14**）を使い、Vimから直接実行します。

▼リスト13　schema.sql

```
--- drop table todo;
create table todo (id serial, title text, done bool default false, createat timestamp default 'now');
```

▼表1　vim-quickrunのための環境変数

環境変数名	設定する内容
PGUSER	ユーザ
PGPASS	パスワード
PGHOST	ホスト
PGPORT	ポート
PGDATABASE	データベース

▼図5　CSSの配色指定の部分が、実際の色にハイライト

```
todo ul li { list-style-type: none; }
todo h3 { color: #335533; background-color: red; }
todo ul { padding-left: 10px; }
~
```

```
:QuickRun sql/postgres
```

ただし直接実行するには、**表1**の環境変数が設定済みである必要があります（いくらかは省略可能）。データベースによってユーザとパスワードを切り分けたい人は、$HOME/.pgpass（Windowsでは、%APPDATA%\postgresql\pgpass.conf）に次のように設定しておくと便利です。

```
*:5432:*:postgres:postgres
server1:5432:develop:user1:PaSsWd
server2:5432:production:user2:pAsSwD
```

編集したら自動でリロード

todoテーブルができたらapp.jsに処理を書きます。その際、何度かアプリケーションを再起動します。できれば自動で再起動してほしいですよね。gulpで状況にあった環境を作成するのも良いのですが、こういった小さいプロジェクトではめんどうです。筆者は自作のツール「mattn/goemon」を使っています。まず**リスト15**

▼リスト14　thinca/vim-quickrun導入のための設定

```
" https://github.com/thinca/vim-quickrun
Plug 'thinca/vim-quickrun'
```

▼リスト15　goemon.yml

```
# Generated by goemon -g
livereload: :35730
tasks:
- match: './assets/*.js'
  commands:
  - minifyjs -m -i ${GOEMON_TARGET_FILE} >
${GOEMON_TARGET_DIR}/${GOEMON_TARGET_NAME}
.min.js
  - :livereload /
- match: './assets/*.css'
  commands:
  - :livereload /
- match: './assets/*.html'
  commands:
  - :livereload /
- match: './assets/*.tag'
  commands:
  - :livereload /
- match: 'app.js'
  commands:
  - :restart
  - :livereload /
```

▼リスト16　CtrlP導入のための設定

```
" https://github.com/ctrlpvim/ctrlp.vim
Plug 'ctrlpvim/ctrlp.vim'
```

▼リスト17　Unite導入のための設定

```
" https://github.com/Shougo/unite.vim
Plug 'Shougo/unite.vim'
```

▼リスト18　todo一覧を取得するコード

```
app.get('/api', function(req, res) {
  pg.connect(function(err, client) {
    if (err) throw err;
    client.query("SELECT * FROM todo ORDER BY id",
function(err, result) {
      if (err) throw err;
      res.set('Content-Type', 'application/json');
      res.send(result.rows);
    });
  });
});
```

のファイルgoemon.ymlをプロジェクトフォルダに置いておきます。次にliverelodaを有効にするために、index.htmlの<head>タグ内に次の行を追加します。

```
<script src="http://localhost:35730/
livereload.js"></script>
```

そして、端末上でgoemonを実行します。

```
$ goemon node app.js
```

以降、todo.cssやtodo.tag、index.htmlを編集するとブラウザが自動でリロードします。またapp.jsを編集するとnodeコマンドが再起動します。つまり、開発者はソースコードを編集するだけで良く、コーディングに専念できるようになります。

HTMLもJavaScriptもSQLも同時に

vim-quickrunでSQLが抽出する結果を確認しながらapp.jsのJavaScriptを実装し、HTML/CSSのデザインを修正します。複数のバッファを切り分けながら編集する場合、「CtrlP」(**リスト16**)や「Unite」(**リスト17**)を導入すると便利になります。筆者はCtrlPを使っています。

この記事を執筆しているときも、原稿、app.js、assets/todo.tagを切り替えながら編集していました。ちなみにtodo一覧を取得するコードは**リスト18**のようになりました。

クライアント側ではsuperagentを使い、**リスト19**のように実装すれば画面表示時にtodo一覧が表示されます。

そのほか、「/api」へのPOSTでtodo追加、「/api/:id」へのPOSTで更新を行う処理を書けば、一応のtodoアプリができあがります。誌面の都合ですべてのソースコードは載せられませんが、リポジトリ[注4]にできあがったものを置いておきますので、興味のある方は遊んでみてく

▼リスト19　クライアント側でtodo一覧を表示させる

```
<script>
var request = window.superagent;
request.get("/api").end(function(err, data) {
  riot.mount('todo', { title: 'やることリスト', items: data.body });
})
</script>
```

注4）URL https://github.com/mattn/node-riot-example

ください。

Vimで JSONを編集する

Node.jsのアプリはpackage.jsonというファイルでパッケージングします。ファイル名のとおりJSONファイルなのですが、JSONファイルを編集しているとインデントがずれたり改行位置がまばらだったりと、本質的ではない部分で多くの時間を取られてしまいます。さらにWebサービスのAPIが返すJSONには改行がないことが多く、可読性が低くて編集に苦労する人もいるでしょう。

そんな場合はjq[注5]というコマンドが便利です。jqは本来、JSONにクエリを与えて抽出／整形するプログラムですが、全体を抽出するクエリ「.」を引数に与えてやれば、改行付きできれいなJSONを得ることができます。jqはUbuntuであればapt-getでインストールできます。インストールしたあと、VimでJSONファイルを開き次のコマンドを実行します。

```
:%!jq .
```

バッファの内容をjqコマンドの標準入力に書き込み、整形した結果でバッファを入れ替えています。改行のないJSONファイルもきれいにインデントされます(**リスト20**)。

jqコマンドの出力結果は可読性が高く、起動も高速ですので筆者も重宝しています。

◆ ◆ ◆

今回はVimを使ってNode.jsアプリを開発し、その行程で必要となるプラグインやコマンド、そしてテクニックを紹介しました。今回紹介した方法がすべてではありませんし、正解でもありません。自分で別のプラグインを見つけても良いですし、自作しても良いでしょう。自分なりの拡張方法を見つけ、自分に一番合ったVimを作り込んでいってください。Vimの拡張性をどう使いこなすかは、みなさんしだいなのです。

SD

▼リスト20　jqでJSONを整形

実行前

```
{"foo": "bar", "bar":[{"baz": true}]}
```

実行後

```
{
  "foo": "bar",
  "bar": [
    {
      "baz": true
    }
  ]
}
```

注5）URL https://stedolan.github.io/jq/

Vim自報

Vimをマスターしたいのならば、言葉のように使え

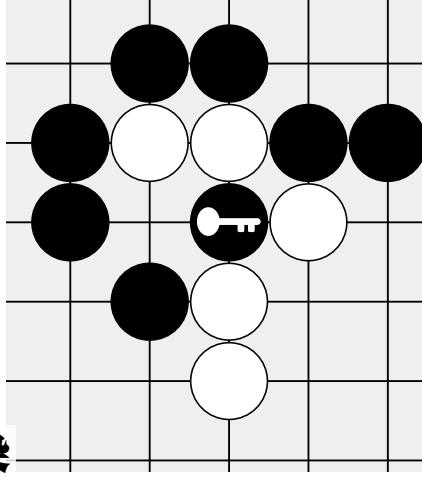
～Vimがもっとも華麗だと言われている理由、それはおそらく、使うという行為が考え始めることと同義になっている、ということであろう。Vimは名詞、動詞、副詞で完結する、謂わば言語のように機能するよう設計されている。～

これはDaniel Miessler氏が書いたVimの入門記事「A vim Tutorial and Primer」[注A]で語られている文章です。本連載の第1回で筆者も同様のことを言いましたが、Vimがほかのテキストエディタと大きく異なる点は、「何を」「どのように」「どうする」をユーザ自身がコマンドとして操作することにより、自由自在に操れるところにあります。頭で覚えるのも良いですが、本当にVimをマスターしたいのならば、喋るようにVimを操ることを念頭に入れて練習すべきです。

注A）URL https://danielmiessler.com/study/vim/

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

【第二八回】開発環境からのマルウェア汚染

2015年11月、Baidu社が提供しているMoplus SDKにバックドアが組み込まれていることが発表されました。このSDKを利用したアプリには脆弱性が含まれてしまうことが、報告されています。開発者にとっては考えたくない事態ですが、似た事例はほかにも確認されています。注意喚起の意味も込めていくつか紹介します。また、この機会に、オープンソース開発の安全性についても考えてみましょう。

マルウェアを埋め込む開発環境

今回は、ここのところ話題になっている、開発環境からのマルウェア汚染について考えてみたいと思います。開発者の立場からすると、自分で作ったプログラムがいつの間にかマルウェアに変身していたとしたら、それは悪夢としか言いようがない事態でしょう。開発者がどんなに気をつけても、開発環境自体が、作ったアプリケーションにマルウェアを埋め込むメカニズムを持っていたならば、完全にお手上げです。そして、そのようなアプリケーションが世の中に広く出回ってしまえば、開発者にとってもユーザにとってもたいへんに困った事態になります。

バックドア機能が組み込まれたMoplus SDK

この事件は最初、Bin Ma氏[注1]が、中国国内検索サービス最大手の会社で、日本でも知名度の高い、Baidu(百度)社が提供しているAndroidアプリ「Baidu Map」のバックグラウンドで、40310/TCPのポートをオープンしていることに気づいたことがきっかけのようです。この内容はMa氏の最初のドキュメント[注2]に詳しく書かれています。

このドキュメントには、「Baidu Mapは、バックグラウンドで特定ポート番号のソケットを作りデーモンとして外部からの通信を受け付けることができるようになっており、それがDoSとして使える脆弱性となっている」という重要な指摘が書かれています。そして、Baidu Mapだけではなく、ほかの似たようなアプリケーションでも同様な脆弱性があることも指摘しています。

それを氏はWormHoleと呼び、のちにBaidu Moplus SDKに起因していることに気がつきます[注3]。

この情報をベースに中国国内のNVD(National Vulnerability Database)である、CNVD(China National Vulnerability Database:国家信息安全漏洞共享平台)は2015年11月5日に、CNVD-2015-07217として脆弱性の注意喚起を行っています[注4]。危険度を表す指標(おそらくCVSSと同等な評価)は9.3となっており、危険度は「高」としています(**図1**)。

Baiduからは10月31日付けで、脆弱性に対するアップデートが行われたという説明が、中国最大手のブログWeibo上でなされています(**図2**)[注5]。

注1) http://www.weibo.com/u/1146576392
注2) Bin Ma, "Research on Open Socket Apps" http://vdisk.weibo.com/s/zo_33fRAzXCZK
注3) http://drops.wooyun.org/papers/10180
注4) http://www.cnvd.org.cn/flaw/show/CNVD-2015-07217
注5) 日本国内では「Moplusに関する本社の発表について」として、Baidu Japanがプレスリリースを次のURLで公開している。内容はWeiboの内容と同じ。 http://www.baidu.jp/info/press/report/151112.html

それは意図的なもの——Trend Microの指摘

2015年11月1日付けのTrend Micro社のセキュリティブログ[注6]でMoplus SDKの分析結果として、次のような極めて重要な指摘がなされます。

しかしながら、トレンドマイクロがこの脆弱性について調査を進めたところ、Moplus SDK自体にバックドア機能が備わっており、必ずしもそれが脆弱性に由来または関連しているわけではないことが明らかになりました。　トレンドマイクロ(株)の日本語訳サイト[注7]より引用

繰り返しになりますが、これは脆弱性ではなく、最初から意図的に組み込まれたバックドア機能であるという指摘です。Trend Micro社は、バックドアのリスクとして「フィッシングサイトへの誘導」「任意の連絡先の追加」「偽のショート・メッセージ・サービス(SMS)」「送信リモートサーバへのローカルファイルのアップロード」「アプリをAndroid端末にインストール」を挙げています。

また、Moplus SDKを組み込んだアプリ(SHA-1シグニチャの違い、バージョンの違いなどを含む)は14,112あり、そのうち4,014がBaiduの提供している公式アプリだそうです。そこから1億人のAndroidユーザが影響を受けているであろうと見積もっています。

Trend Micro社の分析はたいへん詳しく説明されており、この内容に関しては十分に信頼できるものと筆者は判断しています。

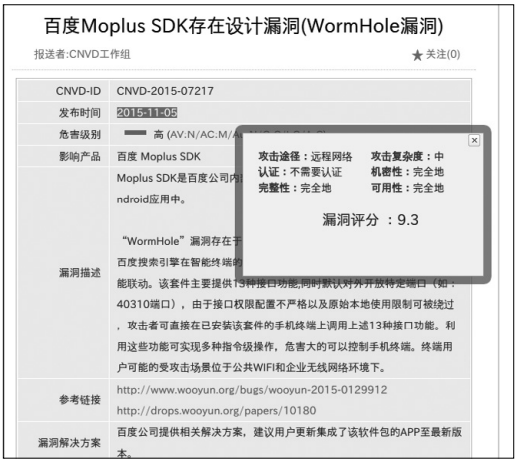
知らぬ間にバックドアが設置される

このバックドアの目的は、短く言ってしまえば「Baidu社はユーザの持つAndroid端末を自分の支配下におき、自分が望むようにコントロールしたかった」と考えるしかありません。その目的が善意なのか悪意なのかは、ここでは関係ありません。問題とすべきはユーザにどれだけのリスクを与えたか、という部分です。CNVDが脆弱性として評価したとおり、危険度は極めて高いのです。そのリスクをユーザに意図的に与えたことは大きな問題です。

ここで「Baiduが特別なのだろうか」「これまでに、このような問題はなかったのか」といった、疑問がいくつか出てくるはずです。

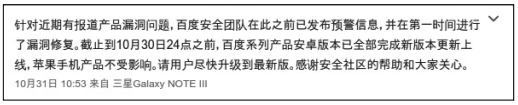
答えを先に言いますと、すでに前例があります。今から10年前の2005年に発覚した[注8]、いわゆるSony BMG rootkit(あるいは"Sony BMG copy protection rootkit scandal")の問題です。

◆図1　CNVDのWebサイトで公開されたWormHoleの脆弱性情報



脆弱性の問題は世界共通で、中国も脆弱性データベースを公開している。

◆図2　Weiboで公表されたBaiduからの説明



日本語訳：
弊社Androidアプリの脆弱性問題に関する一部の報道について、百度セキュリティチームは上記報道以前にアラートを発しており、そして即座に脆弱性を修復しました。10月30日24時までに、すべての百度Androidアプリはバージョンアップを完了しております。また、iOSについては本件とはいっさい関係がありません。ユーザ様に最新バージョンへの更新をお願い申し上げます。セキュリティコミュニティのご協力とみなさまのご理解を心から感謝しております。

注6) Seven Shen, "Setting the Record Straight on Moplus SDK and the Wormhole Vulnerability"
http://blog.trendmicro.com/trendlabs-security-intelligence/setting-the-record-straight-on-moplus-sdk-and-the-wormhole-vulnerability/

注7) http://blog.trendmicro.co.jp/archives/12540

注8) The "Sony rootkit" case (NEWS FROM THE LAB - Tuesday, November 1, 2005)
https://www.f-secure.com/weblog/archives/00000691.html

これは、ある種の音楽CDを、Microsoft社のWindowsが入っているPCのCD-ROMリーダーに挿入すると、ユーザの知らない間にXCP(Extended Copy Protection)、あるいはMediaMax CD-3といったソフトウェアがインストールされてしまう、という問題です。これらのソフトウェアはrootkitの技術が使われており、インターネットに接続していると外部から任意のコードが実行される潜在的なリスクの原因となりました[注9]。

当時、Sony BMG rootkitが含まれていた音楽CDの数は、EFF(Electronic Frontier Foundation:電子フロンティア財団)のWebサイトの情報によれば250タイトル以上とのことです[注10]。

これらの顛末(てんまつ)に関しては、世界的に著名なセキュリティ専門家Bruce Schneier氏のブログサイト「Schneier on Security」の記事[注11]が参考になります。

このように企業がユーザに知らせることなく、ユーザのリスクも考えず、平気でセキュリティを侵害しているようなケースは、昔から存在しています。もちろんSony BMGに対してもSchneier氏のブログのように厳しい声がありましたし、今回のBaiduに関しても厳しい声があるのは当然です。

ベンダがこのようなことをするのは、ユーザのコンピューティング環境をユーザの許可を取ることなく勝手に利用することが当たり前のように考えているからなのか、それともこれがユーザに対する便宜だと思っている一種の誤ったパターナリズムから派生しているのか、はたまた別の理由からこうなってしまったのか、まったく見当がつきません。しかしながら、ユーザ視点に立てば、「企業がユーザの信頼を裏切り、堂々とセキュリティを侵害していると考えざるを得ない」としか言いようがありません。

開発環境でバックドアを埋め込むアイデア

開発環境からバックドアを忍ばせる、というコンセプトも実は古くからあります。もっとも有名なのは、Ken Thompson氏の「コンパイラが自動的にlogin.c(loginコマンド)にバックドアをしかける」というもので、世間ではThompson hackと呼ばれています[注12]。

UNIXの開発者として知られるThompson氏とDennis Ritchie氏は1983年にACM Turing Awardを受賞します。Thompson氏はそのときの受賞記念講演で“Reflections on trusting trust”というタイトルの講演をします。

一応、確認のために書いておくと、Turing Award受賞した理由は、“for their development of generic operating systems theory and specifically for the implementation of the UNIX operating system.(汎用オペレーティングシステム理論の構築とUNIXオペレーティングの実装に対して)”です。直接的にセキュリティは関係していません。

このThompson hackですが、たとえばlogin.cをコンパイルするときに、自動的にパスワード認証回避のコードを加えバックドアを作るようなコンパイラを用意するというアイデアです。ちまたのブログを読むとlogin.cにバックドアを埋め込むコンパイラが存在した、ということを書いているものもありますが、少なくともThompson氏の論文“Reflections on trusting trust”にはアイデアが書かれているだけで、それを実装したという記述はありません。

論文中でも指摘されていますが、コンパイラがバイナリで提供されている環境では、どんなにlogin.cのコードを眺めていようとも、バックドアは見つけられません。もし見つけようとするならlogin.cをコンパイルしたバイナリを直接解析するしかありません。問題があれば、まずlogin.cのソースコードを疑うでしょうし、login.cのみにバックドアをしかけることに特化していれば、ほかのソースコードをコンパイルしても何の異常も出ません。もちろんバイナリの実行コードを逆アセンブルすれば見つか

注9) Vulnerability Note VU#312073 https://www.kb.cert.org/vuls/id/312073
注10) Updated Sony BMG DRM Spotter's Guide https://www.eff.org/deeplinks/2005/12/updated-sony-bmg-drm-spotters-guide
注11) Sony's DRM Rootkit: The Real Story https://www.schneier.com/blog/archives/2005/11/sonys_drm_rootk.html
注12) Thompson, Ken (1984). "Reflections on trusting trust" https://dx.doi.org/10.1145%2F358198.358210

るでしょうが、そこまで思いつくかどうか……。

そのうち誰かが見つける可能性もないわけではないでしょうが、脆弱性ですら何年も見つからないのですから、ずっと見つからずに存在し続ける可能性のほうがずっと高い、というネガティブな考えが筆者の中では強いです。

この論文の最後のAcknowledge(謝辞)のセクションで、「トロイの木馬の考え方はMultics実装の初期段階に米空軍のドキュメントに書いてあったのを読んだが、それが何だったか思い出せない。誰か知っていたら教えてほしい」ということが書かれています。このようにバックドアの考え方は、1960年代からすでにあったことがわかります。新しいようで極めて古くから懸念されている問題なのです。

診断なのかバックドアなのか

2014年に、iOSに、パスワードなしに個人情報データにアクセスできる隠し機能があることが発覚しました[注13]。Apple社は、これを診断のための機能であると説明しています。しかし、何であれ、本来は守られるべき個人情報データに、ベンダからアクセスできる手段があることを隠していたことは事実です。

悪夢が現実に——XcodeGhost

以前、Apple社のオリジナルXcode開発環境に、中国国内で第三者が変更を加えて同国内だけで配布していた特殊なXcode開発環境がありました。実はそこにマルウェア(XcodeGhost)が組み込まれていたことが、2015年9月に中国で発見されました(**図3**)。

XcodeGhostはAlibaba社のセキュリティチーム[注14]や360 Nirvan Team[注15]が詳しく分析しています。ちなみに、これらの分析でも先述のThompson hackについて言及していました。

最新のXcode 7は無料でダウンロードできるようですが、以前は有料でDeveloper Programに登録しなければ、ダウンロードできませんでした。それが中国国内で海賊版Xcode開発環境が広がっていた原因のようです。

その海賊版Xcode開発環境にXcodeGhostが加えられていたため、その開発環境で開発したアプリケーションにはバックドアなど悪質なマルウェアの機能が入ってしまうという状況になってしまいました。

CAC(Cyberspace Administration of China)のWebサイトには、2015年9月21日の京華時報から提供された記事が掲載されており[注16]、360 Nirvan Teamが145,000アプリをスキャンし、うち344アプリが影響を受けていた、と書かれています。

のちにiOSのUnityライブラリも同様にマルウェアが組み込まれていることが判明しており、そちら

◆**図3 XcodeGhostの最初の発見だと言われるWeiboのメッセージ**

iOS開発のエキスパートJoeyBlue_氏が、最初にネット上で言及したと言われる。
http://www.weibo.com/1650375593/CAV5fqdo3

注13) Undocumented iOS functions allow monitoring of personal data, expert says
http://arstechnica.com/security/2014/07/undocumented-ios-functions-allow-monitoring-of-personal-data-expert-says/
Roundup of iOS Backdoor (AKA "Diagnostic Service") Related Tech Articles http://www.zdziarski.com/blog/?p=3522
注14) Alibabaセキュリティチームによる XcodeGhost分析 http://jaq.alibaba.com/blog.htm?id=82
注15) 360 Nirvan TeamによるXcodeGhost分析 http://www.freebuf.com/vuls/78945.html
注16) http://www.cac.gov.cn/2015-09/21/c_1116620258.htm

はUnityGhostと名付けられています。

日本国内でよく使われるであろうアプリケーションらしきものは、リストの中に見当たらなかったので、日本で感染しているiOSアプリはごくわずかであると思われます。

いずれにしろ、コンセプトとして以前から知られていたThompson hackが現実に行われ、しかも、スマートフォンという大きなサイズの市場に影響を与えた歴史的な意味は大きいと言えます。

FLOSS開発環境なら安全か

FLOSS（Free/Libre Open Source Software）開発環境ではソースコードが提供されているので安全である、とは必ずしも言えません。それにはいくつもの安全性を確保するための前提が必要だからです。

①汚染されていないコードを入手する

これには、第三者によるマルウェアコードが入っていないことを確実にする必要があります。公式の配布サイトからソースコードをダウンロードする場合のみ、安全性が保てます。しかも、その配布サイトが安全である必要があり、侵入されコードが書き換えられているような事態にならないことが求められます。途中でのすり替えがないようにサイトとクライアント間はHTTPSで保護している必要があります。現在では、GitHubのようなサイトが使われる時代になっていますから、以前のようにFTPでダウンロードする時代からは、格段に安全性は高まっていると言えるでしょう。

②コードは電子署名されている

①とも絡むのですが、ディストリビューションの中のパッケージとして配布されているような場合は、コードは事前に電子署名されているので安全性はぐっと高くなっていると言えるでしょう。

③コードは精査されている

誰かが何の目的で作ったかわからないようなコードがいつの間にかマージされるようなことでは、セキュリティを確保できません。悪意がなくても、品質の低いコードがいつの間にかマージされているようなことも要注意です。もちろん、悪意のあるコードであろうとなかろうと、品質のためにきちんと精査されたうえでマージされるべきです。

◆ ◆ ◆

これらの条件が満たされないとき、FLOSSでもXcodeGhostのようなことが十分に発生します。

ただし、FLOSSの場合、ベンダが意図してバックドアを付けたMoplus SDKやSony BMG rootkitのようなケースは避けられるでしょうし、Appleのような診断用アクセス権限はユーザの判断によって機能を入れる／入れないといった選択をさせることもできるでしょう。確かに1行や2行で済むようなバックドアなら見つけづらいかもしれませんが、それなりの機能を加えるならば、相応のコード行数となり目立ってしまい、誰かに見つけられる可能性はかなり高いと思われます。

まとめ

ソースコードを公開しておらず、ブラックボックスとしてベンダ側に頼っているソフトウェアに、ベンダが意図的にバックドアに相当するものを入れていると、たいへん発見が難しいと言えます。しかしながら、いつまでも秘密にしておくことは、これまた意外と難しいことが、今回の事例から見えてきました。遅かれ早かれ誰かがどこかで見つけるものなのかもしれません。

健全なアプリケーションに、開発環境側からマルウェアを組み込むアイデアは非常に古くからあって、それが今回、意外とわかりやすい形で我々の目の前に現れました。これまで、その効果は（悪い意味で）非常に大きいと言われていましたが、実際に大規模な影響を与えることが実証されました。

また、FLOSSもソースコードを公開しているということだけで、安全性が保たれるわけではありません。そのソースコードの中身がきちんと見られていて、ソースコードが流通する部分が安全な場合のみ、安全性が高いと考えるべきでしょう。SD

Erlangで学ぶ並行プログラミング

Author 力武健次技術士事務所 所長 力武 健次(りきたけ けんじ) http://rikitake.jp/

第10回 OTPのデータベースMnesia

この連載ではプログラミング言語Erlangとその並行プログラミングについて紹介していきます。今回はOTPのデータベースMnesiaについて紹介します。

OTP最新版の状況

本稿入稿時のErlang/OTPの最新版は11月13日のパッチリリースである18.1.4です[注1][1]。inetsモジュールへのソケットオプションを指定可能にするなどの機能拡張とバグ修正が行われています。

OTPのディスク版KVSであるDETSとその課題

前回はErlang/OTPでのプロセス間データ共有のしくみとして、仮想マシンBEAM上に展開するErlangの項を検索したり保存したりするKey-value型データストア(KVS)であるETSを紹介しました。ETSはBEAMのメモリしか使えませんから、BEAMが停止すればデータは失われます。そこでディスクを使ってBEAMが停止してもデータが残るようにしたKVSとして、DETS[2]が用意されています。**図1**にETSとDETS相互の情報転送の例を示します。ETSとDETSそれぞれで書き換えた内容が反映されていることがわかります。

DETSではKVSのテーブルごとにファイルを作ります。テーブルの大きさには最大2GBという制限があります。テーブル名を知っていればDETSのテーブルは複数プロセス間で共有されます。ETS同様、テーブルをオープンしたプロセスが停止すると、そのテーブルにはアクセスできなくなります[注2]。

ETSとDETSをうまく組み合わせれば、たとえばKVSのデータをメモリのETSとディスクのDETS双方のテーブルに置き、読み出しはメモリ内のみに別途インデックスのETSテーブルを作って高速化し、プロセスやBEAMが停止するなどの障害が発生したときはディスク上のDETSテーブルから復旧するといった構成が可能になります。さらに複数BEAMノード(以下「ノード」)間の連携も可能になるでしょう。

しかし、ETSやDETSを連携させたプログラムを組むのは簡単ではありません。データの書き込みだけを考えてもETSとDETS両方への操作が必要ですし、損傷したデータを修復するにはテーブルを明示的にロックして、終了後解除するなどの煩雑な作業が必要です[3]。そこでこれらの機能を統合した分散データベースと

注1) 最新版はGitHubリポジトリを使いタグを指定することでビルドできます。詳細は「kerlでGitHub版のErlangをインストールする」(http://qiita.com/jj1bdx/items/4f7d7b5a53fcec32ab8d)を参照してください。

注2) 実際にはファイル名がわかっていれば再度テーブルを開くことができます。また、他にテーブルをオープンしたプロセスが残っていれば、そのプロセスからはアクセスし続けることができます。

▼図1　実行例1（ETSとDETSのテーブル間情報転送の例）

```
ETSとDETSの間の相互転送の例を示す
Eshell V7.1  (abort with ^G)
DETSのファイルtest.detsに対しDETSテーブルtestを対応づけて開く
1> dets:open_file(test, [{file, "test.dets"}]).
{ok,test}
DETSテーブルtestにデータを加える
2> dets:insert(test, {a, 1}).
ok
3> dets:insert(test, {b, 2}).
ok
DETSテーブルtestの情報を取得する
4> dets:info(test).
[{type,set},
 {keypos,1},
 {size,2},
 {file_size,5485},
 {filename,"test.dets"}]
DETSテーブルtest全体の情報を出力する。dets:traverse（テーブル名、関数）で、関数をテーブル全体に適用する
5> dets:traverse(test, fun(X) -> io:format("~p~n", [X]), continue end).
{a,1}
{b,2}
[]
ETSテーブルtest_etsを名前付きで作る
6> ets:new(test_ets, [named_table]).
test_ets
DETSテーブルtestの中身をETSテーブルtest_etsに加える（insert操作が行われる）
7> dets:to_ets(test, test_ets).
test_ets
ETSテーブルtest_etsの中身を見ると加えられているのがわかる
8> ets:tab2list(test_ets).
[{b,2},{a,1}]
ETSテーブルtest_etsにデータを加える
9> ets:insert(test_ets,{c,3}).
true
10> ets:insert(test_ets,{d,4}).
true
ETSテーブルtest_etsの中身を見ると加えられているのがわかる
11> ets:tab2list(test_ets).
[{d,4},{c,3},{b,2},{a,1}]
今度はETSテーブルtest_etsの中身をDETSテーブルtestに加える（insert操作が行われる）
12> dets:from_ets(test, test_ets).
ok
DETSテーブルtest全体を見てみると加えられているのがわかる
13> dets:traverse(test, fun(X) -> io:format("~p~n", [X]), continue end).
{a,1}
{b,2}
{c,3}
{d,4}
[]
```

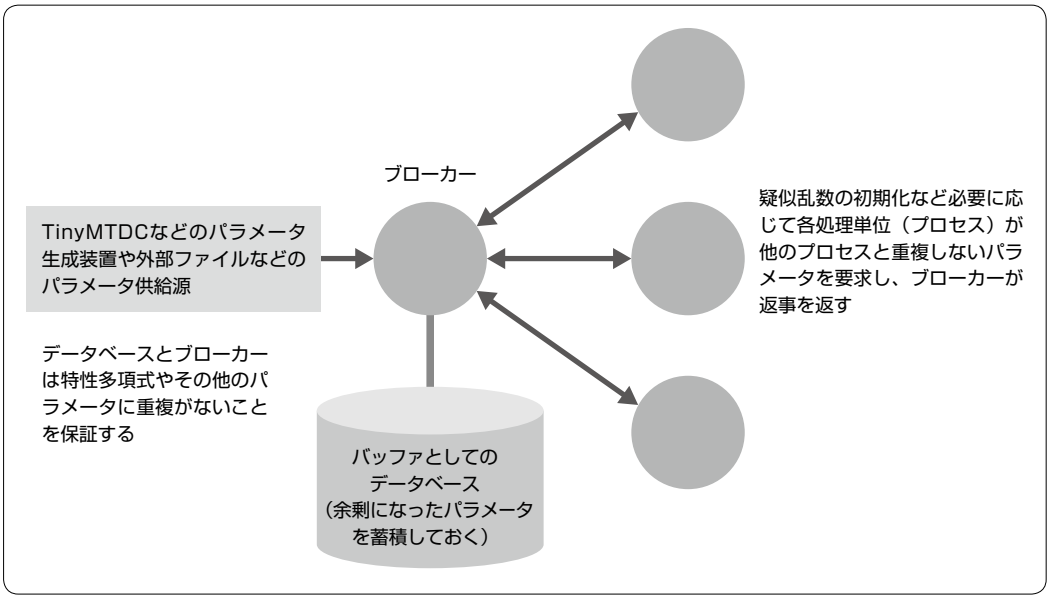
して、Mnesia（エムネジア）[注3]が考案されました[4]。

注3） Mnesiaの語源となったであろう英語のamnesia（アムネジア）には「記憶喪失」という意味があり、Mnesiaという名前はamnesiaにひっかけた「駄洒落」であろうと筆者は推察します。記憶喪失してしまうデータベースというのはかなり皮肉の効いたジョークだと思いますが（笑）。（詳細は次の文献にあります：Joe Armstrong, "Programming Erlang", Second Edition, Pragmatic Bookshelf, 2013, ISBN-13: 978-1-93778-553-6, p. 322, コラム "Why is the DBMS Called Mnesia?"）

Mnesiaの特徴

Mnesia[5]では、ETSやDETSを構成部品として使いつつ、複数のKVSの連携や、トランザクション、複数ノード間のデータ複製などの機能を使うことができるデータベースです。

▼図2　疑似乱数のパラメータ生成と配布のモデル



Mnesiaではetsやdets同様にキーとバリュー双方として取り扱うのはErlangの項を想定しています。Mnesiaは次の例に示した用途に適しています。

- Erlangのリストやタプル、マップなどのデータ構造をKVSのキーや値として持ちたい
- 複数のノードからも同じようにKVSにアクセスしたい
- 同じKVSの複製を複数ノードに持ち耐障害性を高めたい
- 同じKVSの複数をメモリとディスク双方に持ち、ノードやプロセスの停止に耐えられるようにしたい
- 複数ノード間の整合性を厳密に保ちたい[注4]

一方、Mnesiaに向いていないデータの例としては次の例が挙げられます。

- 小規模かつマップ（連載第7回を参照）やdictモジュール（単純な連想配列を実現）[6]で済むもの
- KVSの持つ値のサイズが大きい（ファイル、バイナリなど）
- ログやアーカイブなどデータが増え続けるもの（これらは専用のライブラリや外部データベースで処理すべき）
- 数GB以上の大規模なKVS[注5]
- 複数ノード間の可用性を優先するために、結果整合性（eventual consistency）を自動処理したい[注6]

Mnesiaはejabberd[7]やRabbitMQ[8]といった

注4）Mnesiaは複数ノードに存在するデータの複製（replica）の間にネットワーク分割（partitioned network）による不整合が発生した場合は、エラーイベントが発生します。この場合mnesia:set_master_nodes/{1,2}でマスターとなるノードを指定する必要があります。また、テーブル作成時にmajorityオプションを指定することで、ネットワーク分割が起こった場合多数側でない場合は、書き込みができなくなるようにすることもできます。

注5）MnesiaにはDETSの2GB制限による単一DETSファイルの場合の容量制限がありますが、これは分割（fragmentation）を行うことで回避できます。詳細は（https://erlangcentral.org/wiki/index.php/Mnesia_Table_Fragmentation）を参照してください。なお、実運用で2008年に2億5千万件のエントリを512個のDETSに分割して動かしていたという報告があります（http://erlang.org/pipermail/erlang-questions/2008-March/033951.html）。

注6）一般にデータベースの可用性を整合性よりも優先する場合は、あるキーに対して複数の矛盾する値の複製をすべて読み出して集めたうえで、最終的な値をどれにするかを決定する必要があります。
　このような手法の1つとして、Bob Ippolitoによるstatebox（ https://github.com/mochi/statebox ）があります。

▼リスト1　Mnesiaのテーブル定義レコードファイル（tinymt_params.hrl）

```
TinyMT（32ビット版）の疑似乱数生成パラメータを定義するレコード。32ビット符号なし整数の型
-type uint32() :: 0..16#ffffffff.
レコードを定義する
-record(tinymt32param, {
    特性多項式を示す128ビットの一意な数
    characteristic :: 0..16#ffffffffffffffffffffffffffffffff,
    mat1, mat2, tmat の組で疑似乱数生成が一意にできる
    mat1 :: uint32(),
    mat2 :: uint32(),
    tmat :: uint32(),
    これらは特性多項式の性質を示す数
    weight :: 0..127,
    delta :: 0..31
    }).
```

メッセージングシステムに使われています。またチャットサービス大手のWhatsAppではMnesiaを独自に改造することで2TBのメモリを16分割して180億のレコードを記憶するのに使っていたという2014年の報告があります[9]。

Mnesiaを疑似乱数の生成パラメータ取得に使ってみる

Mnesiaは他のKVS同様、いろいろな用途に応用可能です。今回は疑似乱数生成アルゴリズムのパラメータ取得という例を考えてみることにします。

疑似乱数の1つであるTinyMT[10, 11]注7は、同一のアルゴリズムでも与えるパラメータを変えることで、最大2の58乗個という非常に多くの独立した疑似乱数列を得ることができます[12]。この性質を活かすことで、TinyMTは並行あるいは並列計算にも有効です。たとえば並行あるいは並列計算で疑似乱数によるシミュレーションを行う際は、それぞれの計算単位（Erlangならばプロセス）で違うパラメータを使って独立した疑似乱数列を生成することで、生成された疑似乱数が重複することを防ぐことができます（図2）。

筆者は過去TinyMTのパラメータ生成ソフトウェアTinyMTDCを長期間にわたって動かし、約2億5千万通りの組み合わせを得ています[13]。このパラメータは次の特徴を持っています。

- 特性多項式のパラメータは127ビットの2進数で表現でき、重複はない
- 状態遷移関数を決定するパラメータは32ビットの2進数が2つである
- 出力関数を決定するパラメータは32ビットの2進数が1つである
- 状態遷移関数と出力関数が決まれば、疑似乱数列の生成アルゴリズムが確定する

TinyMTDCにより生成されたパラメータ列[13]は次のとおりです。

```
# charactristic, type, id, mat2, mat2, ➥
tmat, weight, delta
d8524022ed8dff4a8dcc50c798faba43,32,0,➥
8f7011ee,fc78ff1f,3793fdff,63,0
……以下略……
```

これを次のUNIXコマンドでErlang/OTPのfile:consult/1で読める形に変換します（先頭が#の行は取り除いています）。

```
#!/bin/sh
awk 'BEGIN{FS=",";}{print "{tinymt32param, ➥
16#" $1 ", 16#" $4 ", 16#" $5 ", 16#" $6 ➥
", " $7 ", " $8 "}.";}'
```

変換された結果内容はtinymt32param（レコード名）、characteristic、mat1、mat2、tmat、weight、deltaの順番で次のようになります。file:consult/1

注7）筆者は2012年にErlang/OTP用のTinyMTの実装について論文発表を行いました[11]。

はこのタプル列を読んでリストに変換します。

```
{tinymt32param, 16#d8524022ed8dff5a8dcc50c7
798faba43, 16#8f7011ee, 16#fc78ff1f,
16#3793fdff, 63, 0}.
……以下略……
```

このように事前にテキスト処理をしておくことで、file:consult/1という関数でタプルを要素とするリストとして外部のデータをまとめて読み込むことができるようになります。

今回の例では、あらかじめ計算したパラメータ群をMnesiaのテーブルとして用意し、プロセスが疑似乱数の計算を初期化する際に、必要に応じてそのテーブルからパラメータを読み出せるようにします。一度使ったパラメータはテー

▼リスト2　Mnesiaのテーブル操作のためのモジュール（tinymt_params.erl）

```
-module(tinymt_params).
-include("tinymt_params.hrl").
-export([first_time/0,
         init/0,
         add_record/1,
         select_record_tmat_delete/1,
         pickup_random_param/0]).
最初にMnesiaのスキーマを作る関数
-spec first_time() -> ok.
first_time() ->
    ok = mnesia:create_schema([node() | nodes()]).
Mnesiaのテーブルを作る関数
-spec init() -> {atomic, ok}.
init() ->
    {atomic, ok} = mnesia:create_table(
      tinymt32param,
      [{disc_copies, [node() | nodes()]},
       {attributes,
           record_info(fields, tinymt32param)}]).
Mnesiaのテーブルにレコードを加える
-spec add_record(#tinymt32param{}) -> {atomic, ok}.
add_record(R) ->
    mnesia:transaction(
      fun() -> mnesia:write(R) end).
Mnesiaのテーブルにあるレコードのうちtmatが引数と同じか大きいものを1つ選び、その選んだレコードを返すと同時に消去する
-spec select_record_tmat_delete(uint32()) -> #tinymt32param{}.
select_record_tmat_delete(Tmat) ->
マッチスペックを定義：tmatフィールドが与えられた値以上の場合、characteristicフィールドを返す
    Matchhead = #tinymt32param{
        characteristic = '$1', tmat = '$2', _ = '_'},
    Guard = [{'>=', '$2', Tmat}],
    Result = '$1',
    F = fun() ->
         {[V], _} = mnesia:select(tinymt32param,
                      [{Matchhead, Guard, [Result]}], 1, write),
         [Rec] = mnesia:read(tinymt32param, V),
         ok = mnesia:delete(tinymt32param, V, write),
         Rec
        end,
    {atomic, R} = mnesia:transaction(F),
    R.
上記の関数を使いランダムに1つパラメータを返す。選ばれたパラメータはMnesiaのテーブルから消える
-spec pickup_random_param() -> #tinymt32param{}.
pickup_random_param() ->
    select_record_tmat_delete(rand:uniform(16#100000000) - 1).
```

▼図3　2つのノードを1つのホストで立ち上げてMnesiaデータベースをセットアップするまでの手順

```
以下ホスト名"bigmac"というOS Xを実行している機器の例。コンパイルとディレクトリの初期化を行う
端末A: rebar compile
端末A: mkdir ./mnesia-alpha
端末A: mkdir ./mnesia-bravo
仮想マシンのノードをそれぞれ起動
端末A: erl -sname alpha -pa ebin -mnesia dir '"./mnesia-alpha/"'
端末B: erl -sname bravo -pa ebin -mnesia dir '"./mnesia-bravo/"'
以下はErlangシェルで分散ノード間接続のためのクッキーを設定
端末AとB: erlang:set_cookie(node(), mnesia_test_cookie).
分散ノード間の疎通を確認(これをしないと、Mnesiaを扱うノードが確定しない)
端末A: net_adm:ping(alpha@bigmac).
端末B: net_adm:ping(bravo@bigmac).
コンパイルしたオブジェクトとレコードのパターンを読み込む
端末AとB: l(tinymt_params).
端末AとB: rr(tinymt_params).
最初の一度だけはMnesia起動前にMnesiaのスキーマを決める必要がある
端末A: tinymt_params:first_time().
スキーマが決まったらMnesiaを起動できる
端末AとB: mnesia:start().
Mnesiaのテーブルを作る。分散ノード接続ができていればノードalphaとbravo双方にテーブルができる
端末A: tinymt_params:init().
Mnesiaのテーブルに情報を書き込む。TinyMTのパラメータを一度変数にすべて読み込む
端末A: {ok, Tablelist} = file:consult("tinymt32dc-rawtuples.txt").
読み込んだレコードのリストをすべてMnesiaのテーブルに加える
端末A: [tinymt_params:add_record(R) || R <- Tablelist].
```

ブルから消去します。この作業だけであればMnesiaである必要はまったくないのですが、今回は書き込むノードと読み出すノードを別にし、かつ両者のノードそれぞれのメモリとディスク上双方にテーブルを持つことにして、どちらかに事故が起こっても復旧できるための準備をしておきます[注8]。

MnesiaはETSやDETS同様、KVSをErlangのレコードの組として表現します。一般的にはレコード名がMnesiaのテーブル名、またレコードの最初の要素がKVSのキーに相当します。今回は特性多項式の情報をKVSのキーとすることで、キーが重複しないことを保証しています。**リスト1**にMnesiaのテーブルを表すレコード定義を示します。

Mnesiaでは、データベースの構造を示すスキーマ(データベース中のテーブル定義や各テーブルの関連情報を保持するデータ構造)を、mnesia:create_schema/1で最初に作る必要があります。スキーマは関連するすべての分散ノード上で作られる必要があります。スキーマができると、mnesia:create_table/2でテーブルを作ることができます。テーブル作成時のオプションとして、どこに実体を置くか(disc_copiesではメモリとディスク双方に置かれます)、またどのようなレコード定義を持つかなどを指定できます。

データベースの操作を行う際に使われるMnesiaの関数は、「トランザクション」[注9]の単位で実行されることを前提にしています。mnesia:transaction/1という関数を実行することで、この関数の引数として与えられる関数を、トランザクションの単位として実行する

注8) 具体的な復旧の手段については、どれがより確実な情報かを判断する方法が別途必要です。Mnesiaはデータベースの一貫性を重視するため、実際の運用ではマスターノードを決めるか、ノード数を増やして多数決で決定するのが現実的だと筆者は考えます。

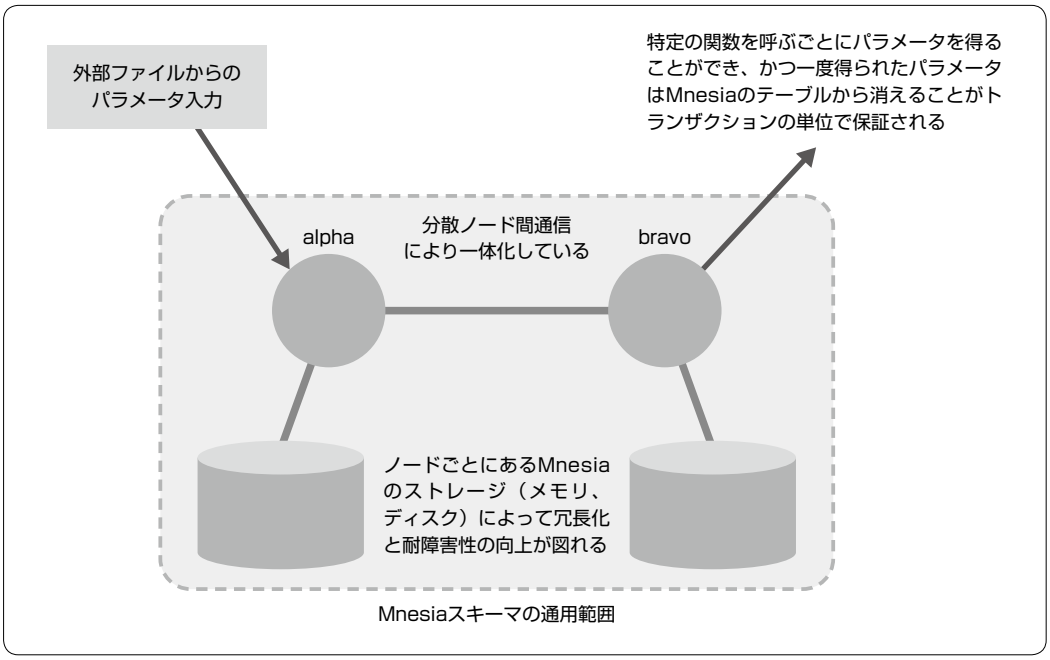
注9) トランザクションの定義はMnesiaでも他のデータベースと同様です。トランザクションに含まれる操作は全部行われるかまったく行われないかのどちらかです。一度実行されたトランザクションの結果は確定して消えることはありません。トランザクションの前後でデータベースの一貫性は失われません。また、各トランザクションの間に他のトランザクションが割り込むことはなく、直列に実行されます。これらの特性をACIDといいます。

ようになっています。この引数の関数は何度も再試行される可能性があるため、外部への副作用があると予期せぬ動作につながることがあります。また、mnesia:read/{1,2,3}やmnesia:write/{1,3}などの操作関数はトランザクション中以外の場所で実行することを想定していません[注10]。

リスト2にMnesiaのテーブル作成や操作のための関数をモジュールにまとめました。テー

注10) Mnesiaには dirty_ で始まるトランザクションの制約を受けない関数群がありますが、これらはより高速である代わりに、データベースの一貫性を損なう可能性があります。本稿では扱いません。

▼図4　Erlang分散ノードとMnesiaによる疑似乱数パラメータ配布



▼図5　Mnesiaデータベース操作の実行例

```
プロンプトの最初にノード名を記している。時系列で示すため2つのノードが混ざっている。テーブルのサイズを返す
(alpha@bigmac)66> mnesia:table_info(tinymt32param, size).
65536
テーブル中のフィールド名を返す
(alpha@bigmac)67> mnesia:table_info(tinymt33param, attributes).
[characteristic,mat1,mat2,tmat,weight,delta]
ここでノードbravoにてパラメータを1つ取得する
(bravo@bigmac)59> tinymt_params:pickup_random_param().
#tinymt32param{characteristic = 188318017379614859055003800087298981289,
               mat1 = 646644945,mat2 = 2875222745,tmat = 4294437887,
               weight = 71,delta = 1}
この時点でテーブルのサイズを見ると1つ減っているのがわかる
(alpha@bigmac)68> mnesia:table_info(tinymt32param, size).
65535
さらにノードbravoにてパラメータを1つ取得する
(bravo@bigmac)60> tinymt_params:pickup_random_param().
#tinymt32param{characteristic = 234226369015328067279219598290313905135,
               mat1 = 575472713,mat2 = 633915763,tmat = 3960389631,
               weight = 61,delta = 0}
さらにサイズが1つ減っているのがわかる
(alpha@bigmac)69> mnesia:table_info(tinymt32param, size).
65534
```

ブルへのレコード追加を行う関数のほかに、テーブルの検索→読み出し→該当レコードの消去という3つの作業を1つのトランザクションにまとめた関数を用意しています。こうすることで、読み出し中の内容を他のプロセスによって変えられてしまうことを防ぐことができます。

本連載では個々のErlangシェルでの実行例を紹介していますが、今回は2つのノードが関与するため、**図3**にそれぞれのノードに端末を対応させたセットアップ手順を別に記します。MnesiaもErlang/OTPのアプリケーションであり、`mnesia:start/0`で実行を開始するようになっています。セットアップしたシステムの構成を**図4**に示します。

実行の様子を**図5**に示します。ノードbravoで操作した結果は、ノードalpha上にも遅滞なく反映されていることがわかります。この実行例では65536個のレコードを読み込ませていますが、筆者のMac mini(Late 2012：2.6Ghz Intel Core i7/主記憶16GB/256GB SSD/OS X 10.11.1/Erlang/OTP 18.1.4)では2つのノードに対するデータベースの読み込みに20秒弱かかりました。SSDへの複製をせずメモリだけで動作するようにすればより高速になるものと思います。また、1,048,576個のレコードを読み込ませた時も、今回紹介したコードで問題なく動作しています。

まとめ

今回はErlang/OTPのデータベースMnesiaについて紹介しました。Mnesiaの機能の詳細を理解するには、他のOTPライブラリ同様リファレンスマニュアル[5]とユーザーズガイド[14]を熟読することをお勧めします。とくにユーザーズガイドでは、今回割愛したテーブル間連携やデータベースの分割、ディスクレスノードの使い方など、運用のノウハウが詰め込まれています。

次回はErlangから他のプログラムやライブラリを利用する方法について紹介する予定です。

ソースコードとサポートページ

連載の記事で紹介したソースコードなどGitHubのリポジトリに置いています(https://github.com/jj1bdx/sd-erlang-public/)。どうぞご活用ください。SD

参考文献

[1] http://erlang.org/pipermail/erlang-questions/2015-November/086722.html

[2] http://erlang.org/doc/man/dets.html

[3] Francesco Cesarini, Simon Thompson, "Erlang Programming", O'Reilly Media, 2009, ISBN-13: 978-0-596-51818-9, pp. 213-243 (Chapter 10: ETS and Dets Tables).

[4] Hakan Mattsson, Hans Nilsson, and Claes Wikström, "Mnesia - A Distributed Robust DBMS for Telecommunications Applications", In Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL '99), Springer-Verlag, London, UK, 1999, pp. 152-163. PDF URL:http://www.erlang.se/publications/mnesia_overview.pdf

[5] http://www.erlang.org/doc/man/mnesia.html

[6] http://www.erlang.org/doc/man/dict.html

[7] http://docs.ejabberd.im/

[8] https://www.rabbitmq.com/

[9] http://highscalability.com/blog/2014/3/31/how-whatsapp-grew-to-nearly-500-million-users-11000-cores-an.html

[10] http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/index-jp.html

[11] Kenji Rikitake, "TinyMT Pseudo Random Number Generator for Erlang",Erlang'12: Proceedings of the 2012 ACM SIGPLAN Erlang Workshop,pp. 67-72 (2012).(実装は https://github.com/jj1bdx/tinymt-erlang/ を参照)

[12] 斎藤睦夫、松本眞、「高速並列計算用の状態空間の小さな高品質疑似乱数生成器」、情報処理学会研究報告 Vol. 2011-HPC-131, No. 3, pp. 1-6.

[13] https://github.com/jj1bdx/tinymtdc-longbatch/

Sphinxで始めるドキュメント作成術

小宮 健 Komiya Takeshi Twitter @tk0miya

第10回 ドキュメントに図を入れよう ——テキストマークアップから図を生成する

今回のテーマ

前回は、さまざまなツールで作成した図をSphinxドキュメントに埋め込む方法を紹介しました。今回は応用編として、テキストマークアップから図を生成し、Sphinxドキュメントに埋め込む方法を紹介します。

Graphviz

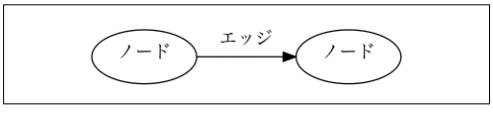
Graphviz[注1]は、AT&T研究所によって開発されたグラフ描画ツールです。DOT言語というマークアップ言語で定義したグラフから画像を生成します。ここでいうグラフとは、ノード(頂点)と、ノード同士を結ぶエッジ(辺)によって構成される図のことです(**図1**)。

はじめに、Graphvizを単体で動作させる手順を紹介します。Graphvizを利用するにはインストールする必要があります。GraphvizはYumやAPT、Homebrewなどのパッケージ管理システムを利用してインストール可能です。

```
Graphvizのインストール(Homebrewの場合)
$ brew install graphviz
```

注1) http://www.graphviz.org/

▼図1 グラフの例



それではサンプルを見ながらDOT言語の書き方を学びましょう。**リスト1**は4つのノードで構成されたシンプルなグラフです。

1行目のdigraphはDOT言語のキーワードで、定義するグラフが有向グラフである(エッジが向きを持つ)ことを表しています。2行目では「reST」と「HTML」の2つのノードと、reSTからHTMLに向かうエッジを->記号を使って定義しています。行末には文の区切りを示すセミコロンがあります。セミコロンは省略することも可能ですが、意図せぬ解釈を防ぐために、記述することをお勧めします。3行目、4行目も同様にノードとエッジを定義しています。最後の5行目はグラフの終了を表す閉じ波括弧です。

グラフ定義は拡張子.dotのファイルに保存します。グラフ定義に日本語を利用する場合は、文字コードをUTF-8にしてください。

グラフ定義ファイルから画像を生成するにはdotコマンドを利用します。dotコマンドには-Tオプションで画像形式を、-oオプションで出力ファイル名を指定します(**図2**)。

リスト1のグラフ定義をdotコマンドで変換すると、**図3**の図が生成されます。グラフ定義から生成した画像では、ノードは丸、エッジは矢

▼リスト1 DOT言語によるマークアップ

```
digraph {
    reST -> HTML;
    reST -> ePub;
    reST -> PDF;
}
```

印として描画されます。

Graphvizはグラフ定義から画像を生成する際に、ノードとエッジを計算に基づいて自動的にレイアウトします。このため、ノードやエッジを追加するにはDOT言語で書かれたグラフ定義を更新するだけでよく、GUIの作図ツールのように手動で位置を調整する必要がありません。

SphinxとGraphvizを組み合わせる

SphinxドキュメントにGraphvizのグラフを埋め込むには、Sphinx付属のsphinx.ext.graphviz拡張[注2]を利用します。

sphinx.ext.graphvizを利用するには、あらかじめGraphvizをインストールしておき、conf.pyで拡張を有効にします。

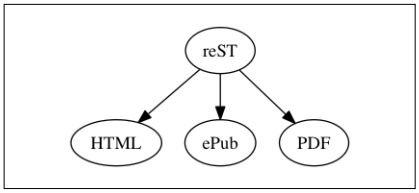
sphinx.ext.graphvizの設定(conf.py)
```
extensions = ['sphinx.ext.graphviz']
```

注2) http://docs.sphinx-users.jp/ext/graphviz.html

▼図2 dotコマンドの書式

```
$ dot -T png [入力ファイル名.dot] -o [出力ファイル名.png]
```

▼図3 リスト1の定義から生成されたグラフ



COLUMN

グラフの見た目を変更する

読みやすいグラフを作るには、ノードやエッジを装飾すると良いでしょう。**リストA**のようにノードやエッジに属性を付与することで、画像に変換した際の見た目を装飾できます(**図A**)。

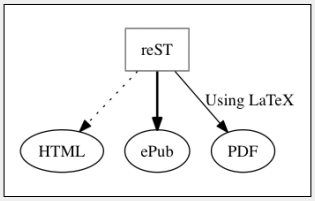
ノードやエッジの装飾には**表A**の属性を利用できます。ここで紹介した属性のほかにも、グラフの出力を変更するオプションなどが数多く提供されています。詳しくはGraphvizのサイト[注A]を参照してください。

注A) http://www.graphviz.org/content/attrs

▼リストA ノードやエッジに属性を付与する

```
digraph {
    ↓reSTノードを長方形に、枠の色を赤にそれぞれ変更
    reST [shape = box, color = "#FF0000"];

    reST -> HTML [style = dotted];      ←エッジを点線に変更
    reST -> ePub [style = bold];        ←エッジを太線に変更
    reST -> PDF [label = "Using LaTeX"]; ←エッジにラベルを追加
}
```

▼図A 装飾されたノードやエッジ(リストAの定義から生成)



▼表A ノードやエッジに指定できるおもな属性

オプション名	概要	取りうる値
shape	ノードの形を変更する	box(長方形)、diamond(ひし形)など
style	ノードやエッジの線のスタイルを変更する	dashed(破線)、dotted(点線)、rounded(角丸)、invis(非表示)など
color	ノードの枠やエッジの線の色を変更する	カラーコード(例:"#FF0000")
fillcolor	ノードの背景を変更する	カラーコード
label	ノードやエッジのラベルを追加する	任意の文字列(例:"Hello world")
dir	エッジの矢印の向きを変更する	back(逆方向)、both(双方向)、none(なし)

グラフをドキュメントに埋め込むにはgraphvizディレクティブを使います。graphvizディレクティブには2種類の記述方法があります。1つはdotファイルを引数として指定する方法、もう1つはグラフ定義をディレクティブのコンテンツとして記述する方法です(**リスト2**)。

ドキュメントの生成には、これまでどおり`make html`を実行します。sphinx.ext.graphvizが自動的にdotコマンドを実行して、グラフを生成し、ドキュメントへ埋め込みます(**図4**)。

外部ツールで生成したグラフを取り込む

Graphvizはさまざまなツールの出力形式に使われています。HashiCorp社のTerraform[注3]もGraphvizに対応しているツールの1つです。Terraformでは定義したインフラ構成を、Graphvizを使って可視化できます。

こうした外部ツールで生成したグラフを継続的に取り込むには、Makefileを書き換えると良いでしょう(**リスト3**)。そうすればドキュメントの変換時にグラフが生成され、最新の状態が自動的にドキュメントへと反映されます(**図5**)。

注3) インフラ構成をコードで記述することで、環境の自動構築やインフラ構成のバージョン管理などを可能にするツール。https://terraform.io/

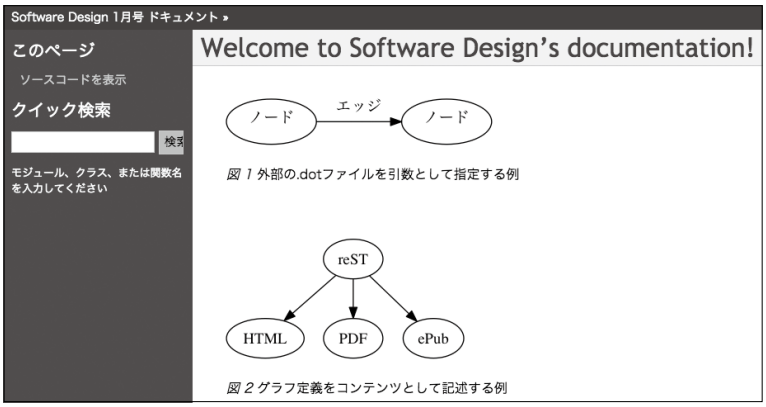
seqdiag

拙作seqdiagは、シーケンス図の作成を目的とするツールです。DOT言語に似た、独自のマークアップ言語を使って定義したシーケンスから画像を生成します。

seqdiagはマルチバイト文字の出力にlibfreetype[注4]を利用するため、シーケンスの定義に日本語を利用する場合は事前にインストールして

注4) フォントのラスタライズなど、さまざまなフォント関連の機能をサポートしたライブラリ。http://www.freetype.org/

▼図4　sphinx.ext.graphvizの出力結果



▼リスト2　graphvizディレクティブの使用例

```
.. graphviz:: example.dot
   :caption: 外部の.dotファイルを引数として指定する例

.. graphviz::
   :caption: グラフ定義をコンテンツとして記述する例

   digraph {
       reST -> HTML;
       reST -> PDF;
       reST -> ePub;
   }
```

▼リスト3　HTML変換の際にterraformを実行する設定(Makefile)

```
html:
    terraform graph > network.dot  ←Terraformでグラフを生成
    $(SPHINXBUILD) -b html $(ALLSPHINXOPTS) $(BUILDDIR)/html
```

おきます。libfreetypeはYumやAPT、Homebrewなどのパッケージ管理システムを利用してインストール可能です。

```
libfreetypeのインストール（Homebrewの場合）
$ brew install freetype
```

その後、pipコマンドでseqdiagをインストールします。

```
seqdiagのインストール
$ pip install seqdiag
```

それではサンプルを見ながらseqdiagの書き方を学びましょう。**リスト4**はブログのコメント投稿の流れを表すシーケンスの定義です。

1行目のseqdiagはseqdiagのシーケンス定義であることを表しています。2行目では->記号を使って「browser」ノードから「webserver」ノードへのメッセージを定義しています。また、label属性を指定してメッセージにラベルを付けています。5行目では<--記号を使って「webserver」ノードから「browser」ノードへの戻り値を定義しています。

シーケンス定義は拡張子.diagのファイルに保存します。シーケンス定義に日本語を利用する場合は、文字コードをUTF-8にしてください。

シーケンス定義ファイルから画像を生成するにはseqdiagコマンドを利用します。seqdiagコマンドには-fオプションでフォントファイルへのパスを、-oオプションで出力ファイル名を指定します（**図6**）。seqdiagコマンドが対応しているフォント形式はTrueTypeフォントおよびOpenTypeフォントです。

リスト4のシーケンス定義からは**図7**の図が画像として生成されます。

seqdiagにもシーケンス図の出力を変更するさまざまな属性や記法が提供されています。詳しくはseqdiagのサイト[注5]を参照してください。

注5）http://blockdiag.com/ja/seqdiag/

▼図5　Terraformの出力結果を取り込んだ例

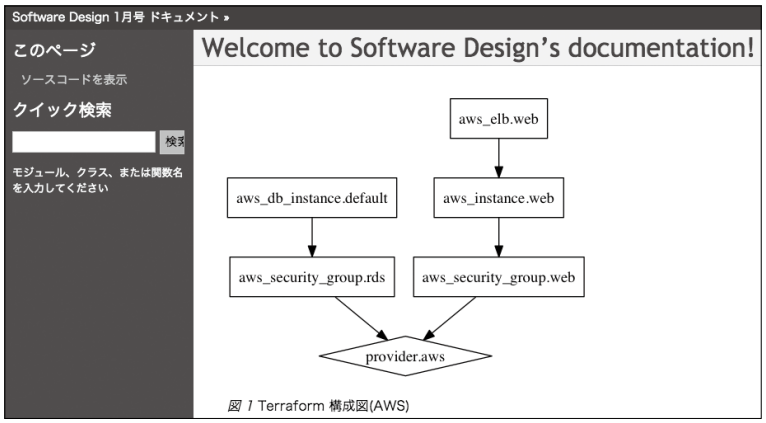
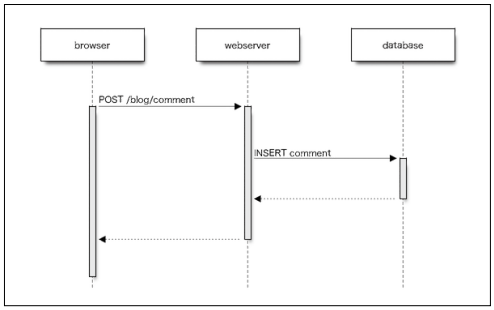


図1 Terraform 構成図(AWS)

▼リスト4　seqdiagによるマークアップ

```
seqdiag {
  browser  -> webserver [label = "POST /blog/comment"];
  webserver  -> database [label = "INSERT comment"];
  webserver <-- database;
  browser <-- webserver;
}
```

▼図7　リスト4の定義から生成されたシーケンス図



▼図6　seqdiagコマンドの書式

```
$ seqdiag -f [フォントファイルへのパス] [入力ファイル名.diag] -o [出力ファイル名.png]
```

Sphinxとseqdiagを組み合わせる

Sphinxドキュメントにシーケンス図を埋め込むには、sphinxcontrib-seqdiag[注6]を利用します。

sphinxcontrib-seqdiagはサードパーティ製のSphinx拡張です。利用するにはインストールする必要があります。

```
sphinxcontrib-seqdiagのインストール
$ pip install sphinxcontrib-seqdiag
```

次にconf.pyでsphinxcontrib-seqdiagを有効にします。同時にフォントファイルへのパスをseqdiag_fontpathに指定します(**リスト5**)。

シーケンス図をドキュメントに埋め込むには、seqdiagディレクティブを使います。graphvizディレクティブと同様、seqdiagディレクティブでも、シーケンス定義ファイルを引数に指定する方法と、シーケンス定義をseqdiagディレクティブのコンテンツとして記述する方法が利用可能です。**リスト6**はシーケンス定義ファイルを引数に指定した例です。

seqdiagディレクティブはfigureディレクティブと互換性があり、scaleやaltなどのオプションにも対応しています。

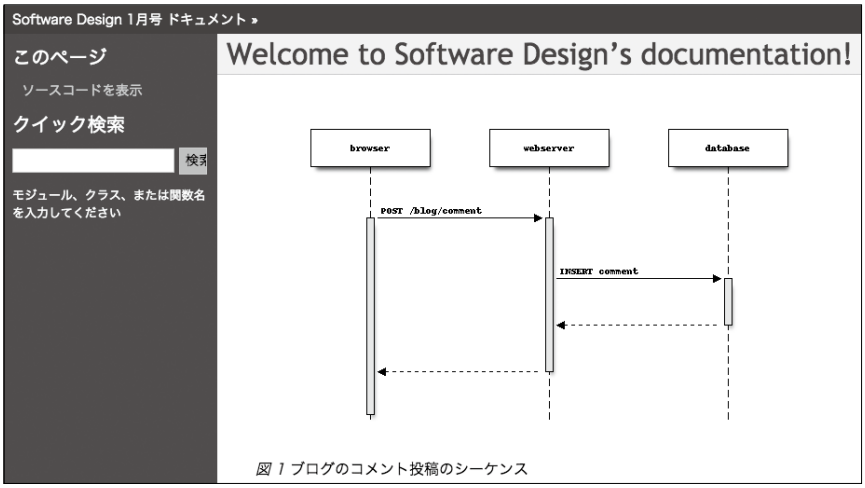
ドキュメントの生成には、これまでどおり`make html`を実行します。sphinxcontrib-seqdiagが自動的にseqdiagを実行して、シーケンス図を生成し、ドキュメントに埋め込みます(**図8**)。

次回予告

今回紹介したGraphvizとseqdiagは、いずれもテキストベースの定義から図を生成します。どちらのツールも図の要素を自動的にレイアウ

注6) https://github.com/blockdiag/sphinxcontrib-seqdiag

▼図8　sphinxcontrib-seqdiagの出力結果



▼リスト5　sphinxcontrib-seqdiagの設定(conf.py)

```
extensions = ['sphinxcontrib.seqdiag']
seqdiag_fontpath = u'/Library/Fonts/ヒラギノ角ゴ Pro W3.otf'
```

▼リスト6　seqdiagディレクティブの使用例

```
.. seqdiag:: example.diag
   :caption: ブログのコメント投稿のシーケンス
```

トしてくれるので、利用者は図の中身を書くことに集中できます。前回紹介したCacooやVisioも含め、適材適所で作図ツールを選択すると良いでしょう。

次回はSphinxの全文検索機能について紹介します。SD

COLUMN

その他のテキストベースの作図ツール

今回紹介したGraphvizとseqdiagのほかにも、テキストベースの作図ツールはいくつも存在します(**表B**)。それぞれマークアップ方式や出力できる図が異なっているため、用途に合わせてツールを使い分けると良いでしょう。どのツールもSphinx拡張が提供されているため、簡単にSphinxと連携できます。

▼表B　テキストベースの作図ツール

ツール名	Sphinx拡張	概要
PlantUML	sphinxcontrib-plantuml	各種UML図
yUML	sphinxcontrib-yuml	UML図(クラス図、アクティビティ図、ユースケース図)
mscgen	sphinxcontrib-mscgen	シーケンス図
blockdiag	sphinxcontrib-blockdiag	ブロック図
actdiag	sphinxcontrib-actdiag	アクティビティ図
nwdiag	sphinxcontrib-nwdiag	ネットワーク関連の図
gnuplot	sphinxcontrib-gnuplot	2次元グラフ／3次元グラフ

COLUMN

Sphinxワークショップ@関西

Author 清水川 貴之

本連載執筆陣の1人、清水川です。

2015年10月31日に、日本UNIXユーザ会さん主催のイベント「Sphinxワークショップ@関西」[注B]が大阪の梅田で開催されました。Sphinx-users.jpから講師として、筆者を含む2名が参加してきました。

参加した13名のうち、半分以上の方がSphinxを初めて使うということでしたが、中にはSphinxをかなり使い込んでいる方もいらっしゃいました。

ワークショップでは、前半でSphinxを紹介し、後半でチュートリアルを使ったハンズオンを行いました(**写真A**)。チュートリアルは、Sphinxのユーザ会のサイトに掲載している「Sphinxをはじめよう[注C]」という記事をもとに、sphinx-quickstartや、記法の練習などを各自進めてもらい、質問があれば講師が答えるというスタイルで行いました。

今回のワークショップは、日本UNIXユーザ会さんの企画です。Sphinx-users.jpでは、要望に応じてSphinxのワークショップやハンズオンの講師を派遣していますので、お気軽にご相談ください。

▼写真A　ハンズオンの様子

注B）http://sphinxjp.connpass.com/event/22023/
注C）http://sphinx-users.jp/gettingstarted/index.html

Mackerelではじめるサーバ管理

Writer 松木 雅幸（まつき まさゆき）㈱はてな
Twitter @songmu

第11回 mackerel-check-pluginsで柔軟なチェック監視

本連載の第5回でMackerelのチェック監視の紹介をしましたが、その後機能が拡充され、公式チェックプラグインパッケージの提供も開始されました。今回はあらためてチェック監視についての説明を行うとともに、公式プラグインを用いた実践的な監視方法について取り上げます。

メトリック監視とチェック監視

Mackerel[注1]に限らず、サーバ監視手法は「メトリック監視」と「チェック監視」の2つに大別されます。

メトリック監視は、継続的に何らかの数値（メトリック）を取得し、その変化を観察、サービスの傾向の把握や異常値の検出を行うものです。Mackerelで言うと、mackerel-agentによるCPUやメモリ使用率などの取得と可視化、閾値（しきいち）設定がそれにあたります。

チェック監視は、定期的に何らかの状態が正常か異常かをチェックすることで、システムの状態を確認するものです。たとえば、ログのエラー文字列を検知するログ監視や、あるプロセスが正常に起動しているかを確認するプロセス監視などが挙げられます。Mackerelでは、mackerel-agentの設定ファイルに設定を追加することで、各種チェック監視ができるようになっています。また、Webサイトが正常なレスポンスを返すかどうかをチェックする「外形監視機能」も、チェック系の監視と言えるでしょう。

チェック監視は必要か

以前は、監視と言えばチェック監視が中心でした。たとえば、古くから使われているOSSの監視システムであるNagios[注2]は、そのようなチェック監視を中心に据えた設計になっています。そのころはサーバシステムへの要求も現在ほどには高くなく、チェック監視でダウンを検知したらサーバを再起動したり、せいぜい待機系を用意しておいてそちらに切り替えたりするなどの素朴な運用が、多くの場合まかり通っていた時代でもあります。

現実問題として、膨大な量のメトリックを時系列データとして定期的に長期保存するためには、相応のディスク領域や性能が必要になります。しかし、当時はハードウェア・ソフトウェアの両面で制約がありました。メトリックを取得しても、その場で閾値と比較するチェック監視を行うのみで、時系列データとして保存しておくのは一部の限られたメトリックに限定せざるを得ませんでした。

しかし、近年ではサーバシステムへの可用性への要求が向上しています。とくにインターネットの発展により、Webシステムは単にシステムを冗長化しておくだけではなく、急激なアクセス負荷向上にも備えなくてはいけません。単なる死活監視だけではなく、キャパシティプランニングのための監視も必要となってきたので

注1） URL https://mackerel.io
注2） URL https://www.nagios.org

す。そのためにも、リアルタイムにサーバの状態を可視化することが求められます。

同時に、時系列データ技術の向上やハードウェア性能の向上、ストレージコストの低下により、多くのメトリックを時系列データとして保存しておくことは、以前ほど苦ではなくなってきています。それにより、以前はその場でのチェック監視しかできなかったメトリックも継続的に保持し、可視化できるようになりました。Mackerelもメトリック監視を中心に据えた設計になっています。

それでは、チェック監視はもう必要ないのでしょうか？　けっしてそんなことはありません。ログのキーワード監視や、死活監視、URL外形監視などは、メトリックと関係のない監視と言えますし、数値を取るにしても継続的に可視化する必要がないものもあります。また、何より、NagiosやSensu[注3]などのチェック監視資産は非常に豊富であり、かつ枯れて(安定して)いるため、これらを活用しない手はありません。

チェック監視プラグインの共通仕様

多くの監視システムで使われている、チェック監視プラグインのための共通仕様があります。それはプログラムの終了コードで、監視対象の状態を表現するものです。これはPOSIXで、終了コードが0だと正常終了、それ以外だと異常終了であるという仕様を拡張したものです。終了コードと状態の対応は次のとおりシンプルな仕様になっています。(プラグイン実行時の標準出力は、補助的なメッセージとして利用されます)。

終了コード	状態
0	OK
1	WARNING
2	CRITICAL
0,1,2以外	UNKNOWN

実際のところ、チェックプラグインは単なるコマンド実行ですので、BashやPerlやPythonやRubyなど、あらゆる言語で記述ができます。

たとえば、Mackerelの公式チェックプラグインに含まれるcheck-httpを利用して、はてなのトップページの動作チェックを次のように実行してみます。

```
% check-http -u http://www.hatena.ne.jp
HTTP OK: HTTP/1.1 200 OK - 126263 bytes in 0.276851 second respons time
```

この場合、正常に200レスポンスが返ってきた旨が出力されています。正常終了ですので終了コードもOK(0)となっています。このチェック監視プラグインの共通仕様はMackerelのチェック監視だけではなく、次のソフトウェアでも採用されています。

- Nagios NRPE(Nagios Remote Plugin Executor)
- Sensu check plugin
- Consul script check

つまり、NagiosやSensuのチェックプラグインをそのままMackerelのそれとして利用できるのです。これは既存の監視システムからMackerelに監視設定を移行したいといった場合に、非常に便利です。

Mackerelにおけるチェック監視の設定

Mackerelでチェック監視の設定をするためには、mackerel-agent.confに次のような項目名と、チェックプラグインのコマンドを指定します。

```
[plugin.checks.http]
command = "/usr/local/bin/check-http -u http://localhost:5000"
```

項目名は、plugin.checks.で始まっている必要があり、含まれるドットの数はちょうど「2」である必要があります。2つめのドット以降は

注3) URL https://sensuapp.org

監視設定の名前として利用されます。コマンドはmackerel-agentにより1分間隔で実行され、その終了ステータス／標準出力が監視結果として使用されます。これは前述したチェックプラグインの仕様に沿って動作する必要があります。この例では、ローカルのアプリケーションサーバが動作しているかどうかの死活監視を行っています。

公式チェックプラグイン集を活用する

Mackerelでは公式チェックプラグインのパッケージを提供しています。

インストール

インストールには、お使いの環境に合わせてMackerel公式のyumリポジトリもしくはaptリポジトリを使うことを推奨しています。リポジトリの設定が行われていれば、次のようにコマンド1つでチェックプラグイン集をインストールできます。

```
rpmパッケージの場合
% yum install mackerel-check-plugins
debパッケージの場合
% apt-get install mackerel-check-plugins
```

リポジトリの設定方法は公式のヘルプ[注4]をご参照ください。rpmファイルやdebファイルを直接取得したい場合は、GitHubのリリース情報[注5]を参照してください。各プラグインは/usr/local/binにインストールされますので、利用する場合にはmackerel-agentの設定ファイルに、利用するプラグインに合わせて設定を追加してください。設定の反映には、mackerel-agentの再起動が必要です。以降は、公式プラグイン集を使って、各種監視設定を行う方法を説明していきます。

check-procsでプロセス監視

● シンプルなプロセス監視

check-procsを使うことでプロセス監視を行えます。たとえばcronの監視をするには次のように指定します。

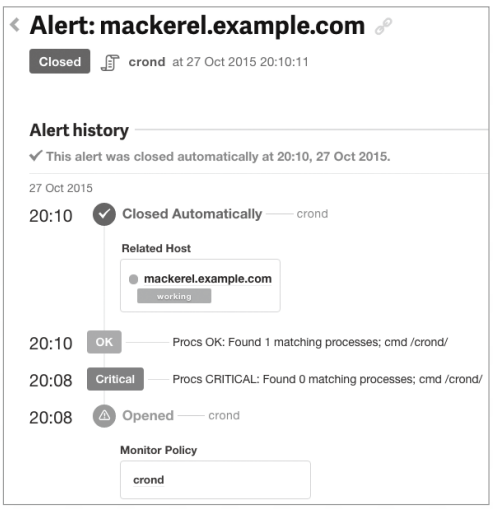
```
[plugin.checks.check_cron]
command = "/usr/local/bin/check-procs
--pattern crond"
```

--pattern(-p)オプションには対象のプロセスにマッチさせる正規表現を指定します。この状態でcrondが動作を停止すると、Criticalアラートが発生し、プロセス復旧後に自動で閉じられます。**図1**はその様子です。

● プロセスの個数も含めて監視

単にプロセスの死活監視だけではなく、ワーカーのプロセス数などが適正に保たれているかどうかの監視をしたい場合もあるでしょう。check-procsには次のようなオプションがあり、プロセス数に閾値を設定できます。

▼図1　check-procsによるプロセス監視



注4）「Mackerelエージェントをインストールする」URL https://mackerel.io/my/instruction-agent
注5）URL https://github.com/mackerelio/go-check-plugins/releases

・-w, --warn-over
　設定値を上回ったらwarning
・-c, --critical-over
　設定値を上回ったらcritical
・-W, --warn-under
　設定値を下回ったらwarning
・-C, --critical-under
　設定値を下回ったらcritical

たとえば、Nginxのワーカー数を含めた監視をしたい場合は次のように指定します。

```
[plugin.checks.check_nginx_worker]
command = "check-procs -p nginx -W 8 -w 10 ➡
-C 1 -c 30 --user nginx"
```

この場合、Nginxのワーカー数が8未満もしくは10より大きい場合にWarning、1未満もしくは30より大きい場合にCriticalとなります。さらにこの例では、--userオプションで実行ユーザを指定し、より正しくワーカー数が取得できるように設定しています。

そのほか、check-procsにはプロセスの実行時間やプロセスの状態に対して監視を行うようなオプションもそろっています。詳しくはGitHubのREADME[注6]をご覧いただくか、% check-procs --helpを実行して確認してください。

check-logでログ監視

●シンプルなログ監視

check-logを使うことでログの監視を行えます。

```
[plugin.checks.access_log]
command = "/usr/local/bin/check-log --file ➡
/var/log/access.log --pattern FATAL"
```

--fileオプションに監視対象のファイルを、--patternオプションにエラー文言を検出したいパターンを正規表現で指定します。この場合、ログファイルにFATALという文字列が出現した場合にCriticalアラートが発生します。ログのチェックは1分ごとに行われ、事前に読んだ行のチェックはスキップされます。

●発生頻度に対する閾値や除外パターンを指定する

キーワードの出現頻度に閾値を設定したい場合や、除外条件を指定したい場合もあるかと思います。

たとえば**リスト1**の設定ではNginxのアクセスログを監視して、4xxや5xx系のエラーの発生をチェックしています。ただし、--excludeを指定することで「robots.txt」へのアクセスは除外しています。また、--warning-over、--critical-overを指定することで、1分間に3回より多く出現したらWarning、10回より多く出現したらCriticalになるように設定しています。最後に--returnオプションが付いていますが、これはパターンが出現したエラー行を標準出力に出力する設定です。このオプションを入れることで、エラー行の内容もMackerelに送られます。このオプションは有用ですが、ログに秘匿情報が含まれる可能性がある場合には、その点を考慮したうえで利用してください。

●1つのログに対して複数のチェック監視を設定したい

check-logは前回のチェックまでに読んだロ

▼リスト1　Nginxのアクセスログを監視して、エラー発生をチェック

```
[plugin.checks.access_status]
command = '''
  /usr/local/bin/check-log                    \
    --file /var/log/nginx/access.log          \
    --pattern 'HTTP/1¥.[01]" [45][0-9][0-9] '  \
    --exclude 'GET .*?robots\.txt HTTP/1\.[01]'\
    --warning-over 3 --critical-over 10       \
    --return
'''
```

注6）URL https://github.com/mackerelio/go-check-plugins/blob/master/check-procs/README.md

▼リスト2　ステートファイルの保存場所を指定

```
[plugin.checks.access_status5xx]
command = '''
  /usr/local/bin/check-log                  \
    --file /var/log/nginx/access.log        \
    --pattern 'HTTP/1\.[01]" 5[0-9][0-9] ' \
    --state-dir /var/mackerel-cache/check-log2
'''
```

グの位置をステートファイルに保存しています。ステートファイルは、デフォルトでは/var/mackerel-cache/check-log下に保存されます。同じログファイルに対して複数のチェック監視を設定する場合には、**リスト2**のように--state-dirを指定して、別の場所にステートファイルを保存するようにしてください。

check-logのそのほかのオプションに関しては、GitHubのREADME[注7]をご覧いただくか% check-log --helpを実行して確認してください。

check-tcpでTCP接続の監視

check-tcpを使うことにより、TCPサーバの接続確認やレスポンスのチェックを行えます。

リスト3ではHTTPのアプリケーションサーバに対して、GET / HTTP/1.0\r\n\r\nというリクエストを送り、レスポンスにOK Farmが含まれているかどうかの確認をしています。--escapeオプションを指定することで、--sendに指定した¥rや¥nなどの文字列が改行文字として扱われるようになります。

また、接続にかかった時間に対して閾値が設定でき、**リスト3**の場合ですと、3秒以上かかった場合にWarning、10秒以上の場合にCriticalが発生します。

もちろん、HTTP以外のサーバのチェックもできます。いくつかのサービスに関しては--serviceオプションを指定することで、複雑な指定なしに標準的なチェックができるようになります。たとえばFTPの場合だと**リスト4**のようになります。--serviceオプションに指定できる設定は、FTP、POP、SPOP、IMAP、SIMAP、SMPT、SSMTPとなっています。check-tcpのそのほかのオプションに関しては、GitHubのREADME[注8]をご覧いただくか、% check-tcp --helpを実行して確認してください。

▼リスト3　TCP接続の監視

```
[plugin.checks.tcp_app]
command = '''
  /usr/local/bin/check-tcp                  \
    --hostname localhost                    \
    --port     5000                         \
    --send     'GET / HTTP/1.0\r\n\r\n'     \
    --escape                                \
    --expect-pattern 'OK Farm'              \
    --warning 3 --critical 10
'''
```

▼リスト4　FTPサーバの接続監視

```
[plugin.checks.ftp]
command = "/usr/local/bin/check-tcp ➥
--service=ftp -H localhost"

↓上記と等価の設定
[plugin.checks.ftp]
command = '''
  /usr/local/bin/check-tcp                  \
    -H localhost --port 21                  \
    --expect-pattern '^200'                 \
    --quit QUIT
'''
```

mackerel-check-pluginsの開発について

mackerel-check-plugins[注9]はGitHubで開発しており、OSSとして提供しています。リポジトリ名に「go」がついていることからわかるように、Mackerelのそのほかのツール群と同様にGolangで開発しています。また、「mackerel」を入れていないのは、チェック監視プラグインとして汎用的に使えるようなツールとして開発

注7）URL https://github.com/mackerelio/go-check-plugins/blob/master/check-log/README.md
注8）URL https://github.com/mackerelio/go-check-plugins/blob/master/check-tcp/README.md
注9）URL https://github.com/mackerelio/go-check-plugins

しているからです。

Golang製であるため、パフォーマンスと可読性のバランスに優れています。実際、各種監視ツールにおいて、チェックプラグインは1分間隔程度の頻繁な頻度で実行されるため、スクリプト言語などで書かれたものは場合によってはパフォーマンスに影響を与えることがあります。たとえば、Sensuのプラグインのほとんどは Rubyで書かれていますが、一部動作がやや重いのです。それらをgo-check-pluginsに差し替えることにより、プラグイン実行によるパフォーマンスへの影響を低減できるでしょう。

go-check-pluginsはもちろんみなさんのpull requestをお待ちしています。書き方は、既存のプラグインのソースコードを参考してください。

まとめ

今回はチェック監視の概念の説明や、公式チェックプラグインを用いた実践的なチェック監視についての紹介を行いました。Mackerelは既存のチェック監視ツールとの互換性を考えて開発しているため、既存のシステムからMackerelへの乗り換えも容易に行えるようになっています。実際、自前で運用していたNagiosを撤廃してMackerelに監視を任せたというユーザもいるくらいです。

チェック監視は歴史がある監視方法であり、多くのプラグインはオープンソースとして公開されているので、実装を読んでみると勉強になることも多かったりします。シンプルながら奥が深い領域だと言えるでしょう。みなさんもぜひチェックプラグインを書いてみてください。go-check-pluginsへのコントリビュートもお待ちしています。SD

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第27回 ❖bhyveでOpenBSDファイアウォール on FreeBSDを構築(その2)

◉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。

ファイアウォール pf(4)

前回はbhyveを使う例として、OpenBSDをゲストオペレーティングシステム(以降、OSと略記)としてインストールする方法を紹介しました。シナリオとして取り上げたのは、FreeBSDをハイパーバイザとして運用し、OpenBSDをファイアウォールとして利用するというものです。今回は仮想環境で動作しているOpenBSDでpf(4)を使う方法を紹介します。

pf(4)はOpenBSD 3.0からシステムに同梱されるようになったTCP/IPトラフィックフィルタリング機能です。いわゆるファイアウォールということになります。OpenBSD 3.0よりも前のバージョンではIPFilterが使われていましたが、IPFilterのライセンス変更の影響を懸念したOpenBSDプロジェクトはIPFilterの使用を止めてpf(4)の開発に取り組みました。OpenBSD 3.0以降、OpenBSDではIPFilterはサポートされておらず、pf(4)が使われています。

pf(4)はTCP/IPトラフィックフィルタリング、NAT、TCPトラフィック帯域制御、TCPトラフィックパケットプライオリティ化機能などを提供します。OpenBSDのみならずFreeBSD、NetBSD、Mac OS Xなどに移植され活用されるようになったほか、FreeBSDベースのファイアウォール／ルータソリューションであるpfSenseやOPNsenseで使われている人気の高いファイアウォールです。

設定ファイルと基本操作 pf.conf(5)&pfctl(8)

ここでは例として、執筆時点で最新リリース版となるOpenBSD 5.8を扱います。前回も紹介しましたが、最新のリリース版はOpenBSDのミラーサイト[注1]などから取得してください。

OpenBSDではpf(4)はデフォルトで有効になっています。設定は/etc/pf.confに記述するしくみになっていて、この設定ファイルは起動時に読み込まれるほか、rcスクリプト経由でも読み込まれます。OpenBSD 5.8のデフォルトの/etc/pf.confは**リスト1**のようになっていて、ループバックネットワークデバイスにはフィルタリングをかけず、それ以外はステートレストラフィックとX11へのリモートコネクションのみをブロックします。このルール以外はすべて素通りする設定になっています。ルールの詳しい読み方については以降の連載で詳しく説明していく予定です。今はなんとなくその雰囲気だけでも感じてもらえればと思います。

pf(4)の制御はpfctl(8)コマンド経由で行います。たとえば、次のようにpfctl(8)コマンドを実行することで、任意のファイルに記述したpf(4)のルールを適用させることができます。

```
# pfctl -f /path/to/mypf.conf
```

設定ファイルを指定してルールを適用した場合、

注1　http://www.openbsd.org/ftp.html

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。

それまでのルールはクリアされ、設定ファイルに記載されたルールに置き換わります。現在適用されているフィルタリングルールは次のように確認できます。

```
# pfctl -sr ↵
block return all
pass all flags S/SA
block return in on ! lo0 proto tcp from
any to any port 6000:6010
```

/etc/pf.confに記述されたフィルタリングルール(**リスト1**)と、pfctl -srで表示されるルールが異なる記述になっていることが確認できると思います。ですが、それらが表現しているルールは同一のものです。

pf(4)のルールセットは人間が表記しやすいようにいくつかのシンタックスシュガーのような機能(リスト、マクロ、テーブルなど)が導入されており、pfctl -srでは最終的に展開された状態のルールが表示されています。

一応、機能を無効化する方法も紹介しておきます。pf(4)の利用が前提になっていますので次の設定を使うことはないと思いますが、pf(4)の機能を無効にしたい場合には/etc/rc.conf.localファイルに次の設定を追加してシステムを再起動してください。

```
pf=NO
```

pf(4)の有効化／無効化はpfctl(8)コマンドを使って次のように動的に切り替えることもできます。

pf(4)を有効化
```
# pfctl -e ↵
pf enabled
```

pf(4)を無効化
```
# pfctl -d ↵
pf disabled
```

このあたりがOpenBSD pf(4)で基本となる設定ファイルpf.conf(5)と制御コマンドpfctl(8)の使い方です。/etc/pf.confファイルを編集して、pfctl(8)コマンドでルールを適用して試して、といった操作を繰り返して目的とするルールを書き上げることになると思います。

ルールセット:シンタックス

pf(4)のシンタックスは基本的に**リスト2**のようになっています。最初に、対象となるパケットを通すのかブロックするのかを、passおよびblockで指定して(actionの部分)、以降はその動作に関する修飾や、どのパケットを対象とするのかの修飾が続きます。インターフェース、プロトコル、行き先、送信元、ポート番号などを指定します。

pf(4)のシンタックスはだいぶわかりやすいので、勘のよい方ならこの段階ですでにそれなりに記述できると思います。シンタックスの詳しい内容は次回以降で説明するとして、今回はシンタックスの詳しい説明に入る前にマクロ、リスト、テーブルと呼ばれる記述方法を説明しておきます。このあたりがわかっていると、ルールの理解やルール記述の効率が変わってきます。

ルールセット:マクロ

pf(4)ではルールセットにマクロと呼ばれる機能が用意されています。これは変数のようなもので、=で割り当て、$で参照を行います。次のようにルー

▼リスト1　OpenBSD 5.8のデフォルトの/etc/pf.conf

```
#       $OpenBSD: pf.conf,v 1.54 2014/08/23 05:49:42 deraadt Exp $
#
# See pf.conf(5) and /etc/examples/pf.conf

set skip on lo

block return    # block stateless traffic
pass            # establish keep-state

# By default, do not permit remote connections to X11
block return in on ! lo0 proto tcp to port 6000:6010
```

▼リスト2　pf(4)シンタックスの基本

```
action [direction] [log] [quick] [on interface] [af] [proto protocol] \
   [from src_addr [port src_port]] [to dst_addr [port dst_port]] [flags tcp_flags] [state]
```

ルを記述した場合、

```
if = "fxp0"
block in on $if from any to any
```

pf(4)は次のルールに展開して処理を行います。

```
block drop in on fxp0 all
```

if = "fxp0"で、ifにfxp0を割り当てています。$ifのように使用すると、これがfxp0に置き換わるというしくみです。なお、fxp0はネットワークインターフェースの名称です(fxpはIntel EtherExpress PROのデバイスドライバを表しています。この名称はFreeBSDと同じです)。

マクロは再帰的に利用することもできます。再帰的に使用する場合、展開させたい部分はダブルクォーテーションの中に含めないように記述します。たとえば、次のようにマクロを使います。

```
host1 = "192.168.1.1"
host2 = "192.168.1.2"
hosts = "{" $host1 $host2 "}"
```

この場合、ルールの中で$hostsのように使用すれば{ 192.168.1.1 192.168.1.2 }と書いたことと同じになります。IPアドレス、ポート番号、ネットワークインターフェース名などをマクロで定義するなどして、ルールセットの複雑性を軽減してメンテナンスを容易にする効果があります。

ルールセット：リスト

pf(4)ではプロトコル、ポート番号、アドレスなどを複数同時に指定できます。リストは{ }で囲まれた対象で表現され、{ }の中はスペース区切り、またはカンマ区切りで値が使われます。たとえば、**リスト3**のようにIPアドレスを{ }で囲ってカンマ区切りで配置すると、pf(4)としては**リスト4**のように個別のルールセットに展開してフィルタリングを行います。

リストは1つのルールセットに複数同時に使うこともできます。**リスト5**のように複数のリストを使った場合、**リスト6**のように互いの要素を掛け合わせたルールに展開したものが使用されます(なお、行末にバックスラッシュを指定することで、1行に書くべきルールを複数行に改行して記述できます)。

リストはネストしていても使用できます。**リスト7**のようにルールでリストがネストしている場合、**リスト8**のようにすべて展開された結果がルールとして使われます。

これはマクロと併用する場合に便利な機能です。**リスト9**のルールは先のルール(**リスト8**)と同じ結果に展開されます。ネストしているリストもフラットなリストに展開されるので、こうした利用が可能になっています。

ルールセット：テーブル

リストと似た機能にテーブルがあります。これはIPアドレスのグループを表現するもので、大量のアドレスのグループを表現したい場合などに使われます。リストよりもメモリの消費量が少なく、処理

▼リスト3　IPアドレスをリストで指定

```
block out on fxp0 from { 192.168.1.1, 192.168.1.2 } to any
```

▼リスト4　リストは展開して使用される

```
block drop out on fxp0 inet from 192.168.1.1 to any
block drop out on fxp0 inet from 192.168.1.2 to any
```

▼リスト5　1つの行に複数のリストを指定

```
block out on fxp0 proto { tcp, udp } \
    from { 192.168.1.1, 192.168.1.2 } to any
```

▼リスト6　すべてのリストが展開されて使用される

```
block drop out on fxp0 inet proto tcp from 192.168.1.1 to any
block drop out on fxp0 inet proto tcp from 192.168.1.2 to any
block drop out on fxp0 inet proto udp from 192.168.1.1 to any
block drop out on fxp0 inet proto udp from 192.168.1.2 to any
```

▼リスト7　リストがネストしているケース

```
block out on fxp0 \
    from { 10.0.0.0/8, { 192.168.1.1, 192.168.1.2 } } to any
```

速度が速いという特徴があるとされています。

テーブルはリストで表現できないルールを考えるとよくわかります。たとえば**リスト10**のルールを見てください。これは特定のIPアドレスを除く、ほかの全体に一致してほしいという「希望」が見て取れる書き方ですが(!はNOTを意味しています。192.168.1.1以外の192.168.1.0/24をブロックしてほしい、というニュアンスを込めています)、リストは指定されているものを展開するだけの機能ですので、処理前に**リスト11**のように展開されます。これでは全部ブロックされてしまいますので、書いた方の思惑とは違う動作をすることになります。希望通りには動作してくれません。

こうした場合に利用する機能がテーブルです。**リスト12**のようにテーブルを使用すると、希望通りに192.168.1.1以外の192.168.1.*をブロックするようになります。リストのように個別に展開するのではなく、テーブルとしてひとつのまとまりとして処理するため、こうしたことが可能になっています。

テーブルはtableで宣言し、必ず< >で囲って使用します。適用されているテーブルの内容は次のようにしてpfctl(8)コマンドで確認できます。

```
# pfctl -t hosts -T show
   192.168.1.0/24
  !192.168.1.1
```

テーブルの内容はpfctl(8)コマンドを使ってリアルタイムに変更することもできます。テーブル名を指定して、**図1、2**のようにpfctl(8)コマンドを実行することで行います。

テーブルに含めたいアドレス一覧は**リスト13、図3**のようにファイルから取得させることもできます。テーブルの内容を動的に変更してほしくない場合にはconstの指定を使えますし、ルールがすべて消えた場合でもテーブルとして存在させ続ける場合にはpersistという指定を加えます。テーブルはpf(4)を使いこなすうえで基本となる機能です。

次回はpf(4)のもっと踏み込んだ使い方を紹介します。SD

▼リスト8　ネストしたリストはフラットなリストとして展開される

```
block drop out on fxp0 inet from 10.0.0.0/8 to any
block drop out on fxp0 inet from 192.168.1.1 to any
block drop out on fxp0 inet from 192.168.1.2 to any
```

▼リスト9　マクロとリストのネストを組み合わせた書き方

```
host1 = "192.168.1.1"
host2 = "192.168.1.2"
host = "{" $host1 $host2 "}"
block out on fxp0 from { 10.0.0.0/8, $host } to any
```

▼リスト10　楽観的推測：192.168.1.1以外の192.168.1.0/24をブロック

```
block out on fxp0 from { 192.168.1.0/24, !192.168.1.1 } to any
```

▼リスト11　展開されたルールは無慈悲にも全部をブロック

```
block drop out on fxp0 inet from 192.168.1.0/24 to any
block drop out on fxp0 inet from ! 192.168.1.1 to any
```

▼リスト12　テーブルを使って希望のルールを実現

```
table <hosts> { 192.168.1.0/24, !192.168.1.1 }
block out on fxp0 from <hosts> to any
```

▼図1　pfctl(8) - テーブルへアドレスを追加

```
# pfctl -t hosts -T add !192.168.1.2
1/1 addresses added.
```

▼図2　pfctl(8) - テーブルからアドレスを削除

```
# pfctl -t hosts -T delete !192.168.1.2
1/1 addresses deleted.
```

▼リスト13　ファイルからアドレス一覧を取得してテーブルに設定

```
table <hosts> file "/etc/iplist"
block out on fxp0 from <hosts> to any
```

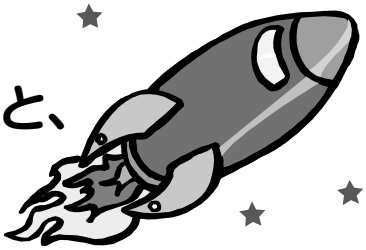
▼図3　ファイル内容の確認と設定されたルールの確認

```
# cat /etc/iplist
192.168.1.0/24
!192.168.1.3
!192.168.1.4
!192.168.1.5
!192.168.1.6
!192.168.1.7
# pfctl -t hosts -T show
   192.168.1.0/24
  !192.168.1.3
  !192.168.1.4
  !192.168.1.5
  !192.168.1.6
  !192.168.1.7
```

31

Debian Developer　やまねひでき　henrich@debian.org

DebConf15レポート（中編）と、Debian Live終了騒動



Debian Hot Topics

DebConf15レポート（つづき）

前回に引き続いて、DebConf15のセッションの内容をかいつまんで紹介します。

「派生ディストリビューション」セッション

DebConfは、DebianやUbuntuをベースとする派生ディストリビューションの話が、実際に開発／運用している当事者から聴ける貴重な機会です。紹介された各ディストリビューションをざっと見てみましょう。

★LiMux

「Linux in the City of Munich（AKA LiMux）- A 2015 status update」というセッションでは、ミュンヘン市におけるLinuxの採用とその状況について説明が行われました。

ミュンヘン市は150万人の住民を擁する、ドイツで3番め・ヨーロッパで12番めに大きな都市で、33,000人の職員を抱えています。2001年に、Windows NT 4.0のサポート終了を視野に入れて移行プロジェクトが始まりました。プロジェクトのゴールは80%のPC（この時点で15,000台中12,000台）がLinuxへ移行することで、2013年にプロジェクトは終了しましたが、結果として15,000台のPCが移行を完了しています。そして、2015年現在も、まだLinuxの利用は進んでおり、18,000台が移行したとのことです。

現在、利用している独自ディストリビューション「LiMux 5.0」は、Kubuntu 12.04ベースでPPA（Personal Package Archive）からKDE 4.12を追加したものです。利用ソフトウェアとして、Firefox と Thunderbird は ESR[注1] を、Libre Officeは4.1をベースに300以上のパッチを当てたものを利用しています（ミュンヘン市は、LibreOfficeの開発母体である「The Document Foundation」のアドバイザリーボード（顧問委員）でもあります）。

今後の予定として、Ubuntu 14.04ベースにアップデートし、さらにLiMux 6.0ではUbuntu 18.04をベースにする予定だそうで、その足取りは確かなもののようです。

★Lernstick

「Lernstick - A Debian derivative for Schools in Switzerland」というセッションでは、同様に公共団体である学校が利用するディストリビューションとして、スイスで開発／利用されている「Lernstick」[注2]の紹介が行われました。

Lernstickは次の目標をめざして開発されているそうです。

- 学校で便利に使えること
- 持ち運びができてセキュアな学習環境であること

注1）Extended Support Releaseの略で、長期サポート版のこと。URL http://www.mozilla.jp/business/downloads/ を参照。DebianもIceweaselやIcedoveはESRをベースに採用している。

注2）URL http://imedias.ch/lernstick/

- 10年前の古いマシンでも動作すること
- 技術的な知識の乏しいエンドユーザでも利用できること
- 管理負担が少ないこと
- BYOD(Bring Your Own Device)が可能であること
- 安定したベースシステムに最新のアプリケーションを載せられること

開発は大学から資金援助を受けてパートタイムで2名が実施しており、そのほかの資金調達としては利用する学校からサポート契約を得ているとのこと。

技術的には、Debian stable(Debian Live)をベースに、backports＋独自のbackports＋サードパーティリポジトリ＋独自パッケージという構成で、「なるべくDebianと差分が最小になるように」という方向性を採っています。独自に作成したソフトウェアとして、USBメモリにインストールするためのツール「DLCopy」や「Lernstick Welcome」、グラフィカルブートを実現するソフトウェアgfxbootの設定をしやすくするツール「xmlboot」などがあります[注3]。

標準バージョンは4GB弱のサイズに、複数のデスクトップ環境(GNOME、KDE、XFCEなど)を含み、デフォルトでは管理者パスワードなしでsudoが可能になっています。

別バージョンとして、試験での利用を念頭に置いた制限版(Lernstick Exam Environment)があります。また、契約者にはプロプライエタリ・ソフトウェアを含むカスタムビルドも提供しています。Debianでは導入しづらい変更点として、知識がないユーザでも使えるように、Microsoft社に署名をしてもらってSecure Bootをサポートしていることや、non-freeなドライバが最初から導入できるようになっている、などの違いがあるそうです。

今後は「Debian Edu[注4]との協業を行う」「開発したツール類をDebianへアップロードする」「WikiやTODOリストを公開して開発のプロセスを可視化する」などの改善を検討しているとのことです。

★AIMS Desktop

別の学校系ディストリビューションとして、「AIMS Desktop」というセッションで、AIMS(アフリカ数学科学研究所)[注5]で利用されている「AIMS Desktop」[注6]の案内がされました。

AIMSは、2003年に南アフリカのケープタウンで設立された数学とコンピュータサイエンスのための新しい大学院生センターです。開設当初からLinuxを使用しており、アプリケーションとしてSagemath、R Studio、Octave、Scipy、Spyder、TexMakerなどを利用しています(とくにSagemathは、Debian/Ubuntu向けのPPAを作成するなど力を入れているようです)。

最初はDebian testingを利用しようとしていたのですが、2004年に発表されたUbuntuにスイッチし、その後、Ubuntuをカスタマイズしたものをずっと利用してきています。しかし、近年発表されたCanonical社のIPポリシー[注7]に抵触しないように、AIMSのライセンスとして利用できる対象を大幅に制限しているそうで、その結果としてDebConfでも自由に配布ができない状態となっています。これは良くないので、今後はDebianへスイッチすることも検討しているそうです。権利関係周りはややこしいですね。

◆ ◆ ◆

日本の場合、地方自治体や学校などの団体がディストリビューションを作るまでに至るような話はいっさい聞かれません。その代わりに、

注3) こちらのツール類は URL https://github.com/imedias で公開されている。

注4) 学校向けのディストリビューションで、おもにノルウェーでの活動が活発だが、台湾などでも活動がある。
URL https://wiki.debian.org/DebianEdu/

注5) URL http://www.aims.ac.za/english/

注6) URL https://launchpad.net/aims-desktop

注7) Interectural Property、知的財産。

不得手な担当者がぼんやりとした仕様とカツカツ予算でSIerに丸投げし、SIerは工数がないので仕様を詰め切れずに開発して、結果的に誰も使わないようなシステムができあがる……という話をチラホラと耳にします。

独自ディストリビューションを作った海外の事例でも、潤沢に予算があるわけではなく苦肉の策として始めた……というようなところも多くあります。ですが、そんな中でも当事者として試行錯誤して工夫を重ねているのが見て取れます。

日本の自治体や学校は、いくつかの例外を除いては縮小傾向にありますので、潤沢な予算は期待できません。今後は、海外の事例のような方向性を模索していくようになっても良いのではないでしょうか。

★Tails

「Tails - a technical overview」は、アメリカ国家安全保障局(NSA)による個人情報収集の手口を告発したエドワード・スノーデン氏が利用していることで有名になった「Tails」のコントリビュータらによるセッションです。

TailsはTorネットワーク[注8]の利用(**図1**)に表されるように、ユーザのプライバシーと匿名性を第一に考えて開発されているLiveOSです。現在は、Debian 7「Wheezy」がベースになっています。特徴として、「どのソフトウェアがネットワークに接続するかを把握できないため、Torネットワーク以外への接続をすべてブロックする」「シャットダウン時にメモリ上の痕跡を残さないようにきちんと上書き消去する」などの機能があります。

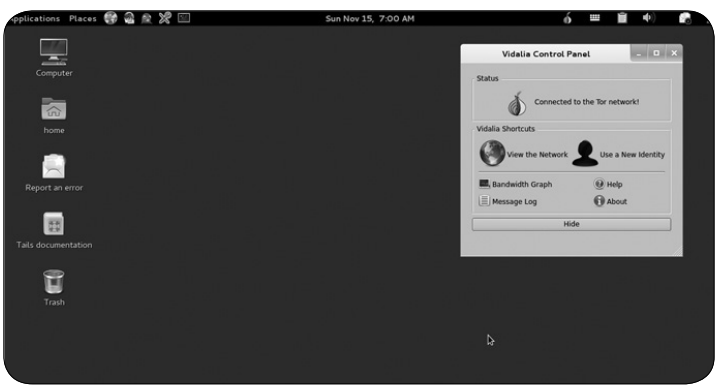
LiveOSであるものの、USBメモリにインストールしてある場合はデータの保存についても考えてあり、LUKS[注9]で暗号化したパーティションに保存ができるようになっています。

セッション中では、このような「ある分野に特化したディストリビューション」はたいてい早期に終了する(とくにセキュリティ分野)との認識が示されており、原因として「チームが小さい」「長期のコミットメントがない」「メンテナンスやユーザサポートの負荷が高い」「NIH症候群」[注10]などが挙げられていました。この対策として、なるべくDebianで開発されているツール類を利用するようにしており、その結果、Tails固有のコードが減少し、各コンポーネントのつなぎ合わせ(glue code)が多くなり、活動もTailsの要望を取り入れてもらうようにupstreamとの対話が多くなっているとのことです。

開発は、Vagrant[注11]を使った継続的ビルドを実施し、Sikuli、libvirt、cucumber[注12]を利

注8) ルータを経由するたびに接続経路の匿名化のために暗号化を施すことで、何重にも暗号化が施されるという米海軍調査研究所由来の暗号化通信方式(「オニオンルーティング」と呼ばれる)を実装しているソフトウェア「Tor」により構築されたネットワーク。

▼図1　Tailsのデスクトップ画面
(デフォルトでTorネットワークへ接続される)



注9) Linux Unified Key Setup-on-disk-formatの略で、ディスク暗号化の仕様。Linuxにおいては通常dm-cryptが利用される。ブロックデバイス全体を暗号化するため、脱着可能なストレージメディアやノートPCのディスクドライブなどに適する。

注10) "Not Invended Here(我々が作ったものではない)"と言って他者が作ったものを拒否して作りなおそうとする態度。

注11) HasiCorp社製の仮想環境構築を容易にしてくれるツール。バックエンドとして各種仮想化技術が使えるが、おもにVirtualBoxが使われることが多い。

注12) Sikuliは OpenCVをベースに画像認識を利用してGUI操作の自動化を行うスクリプト。libvirtはXen、KVM、VMWareなど各種仮想マシンの制御を抽象化したライブラリ。cucumberはRubyで実装されている受け入れテストのためのテスティングフレームワーク。

用した自動テストを実装中。限られた人的リソース(15名弱)で、6週間ごとの継続したリリースを行っています。2016年にはDebian 8「Jessie」ベースのTails 2.0をリリースする予定だそうです。2.0では、さらなるセキュリティ面での強化も予定しているが、Tailsそのものに限らずサイトの国際化やバグトラッカーの設置などを検討している、とのこと。このあたり「ユーザビリティとセキュリティのバランス」も念頭に置いているのが見て取れます。Tailsプロジェクトでは、UI/UXの改善や翻訳者なども必要としているとのことですので、興味のある方は一度ダウンロードして利用してみると良いでしょう注13。

◆ ◆ ◆

これらのほかにも「Cumulus Linux: Debian for Network Switches」や「hLinux: HP's Debian derivative a year later」などの商用ベンダによる派生ディストリビューションのセッションも開かれていました。

Debian Live終了騒動

前節で紹介した各種の派生ディストリビューションでも利用されている「Debian Live」ですが、突然、開発者が「プロジェクトはハイジャックされた」と言って終了を宣言しました。周りの人には唐突に見えた終了ですが、さて、いったい何があったのでしょうか?

これは、Bug#804315で新規パッケージとして「live-build-ng」が登録されたことに、live-buildパッケージのメンテナDaniel Baumann氏が驚いたことに始まります。彼のlive-buildパッケージはまだ終わってないのに、第三者が「-ng(new generation)」などというパッケージを登録するとはどういうことだ!?となったわけです。これだけだと、「すでに前任者がいるのにネゴも取らないでひどいな」というだけの話なのですが、これに至るまでにはそれなりのわけがあったのです。

背景には、Debian LiveがほぼDaniel氏1人で運営/開発されているプロジェクトであったことが根本にあります。彼は他者からのパッチや要望の受け入れには消極的であるにもかかわらず、突然思いつきで大きく仕様を変更することが珍しくありませんでした(これは、UEFIサポートを切望していたdebian-CDチーム側のバグ登録を2013年から放置したままであったことからも見て取れます)。

CDチーム側からすると、機能面では、作成のための設定も複雑で壊れやすく扱いが難しいのが難点でした。さらに、運用面においてもオフィシャルLiveイメージの作成も通常のフローから外れてDebian Live側で実施したがったうえに、毎回リリースが遅れる……そのようなことから、CDチームのメンバーにはかなりのフラストレーションだったようです(何度かDaniel氏に参加してもらおうとも試みたが、すべて失敗に終わり、最終的にCDチームはDaniel氏に頼ることなく、自身でビルドするようになったとのこと)。

そして、最終的にDaniel氏のlive-buildに頼らない別実装ができたのでそれをパッケージ化しようとした……のですが、その際にあてつけみたいな名前を付けてしまったので議論になった、というわけです。

現状では、「live-build-ng」は「live-wrapper」と名前を変えて登録作業が続けられているようです。この後、live-buildの作業へDaniel氏が復帰することは難しそうで、ほかの人が引き継ぐことになりそうです。これまで、live-buildをベースに作業をしてきた派生ディストリビューションが、live-buildを続けるのか、それともlive-wrapperベースに移行するかについては、今後を注視してみたいと思います注14。SD

注13) URL https://tails.boum.org/download/からダウンロードし、URL https://tails.boum.org/contribute/を参照。

注14) ソースの取得については、live.debian.netが利用できない場合は、URL https://github.com/debian-liveの利用が推奨されている。

第69回 Ubuntu Monthly Report

Ubuntu 15.10で修正された日本語関連のバグ

Ubuntu Japanese Team　あわしろいくや

今回は第67回に書けなかったUbuntu 15.10のバグとその修正についてのレポートです。

第67回の補足

本連載第67回（2015年11月号）で『Ubuntu 15.10とそのフレーバーについて』と題して執筆しましたが、締め切りの関係で、本誌発売の1ヵ月前の内容の割には、現時点でも、付け加えるべき補足はあまり多くありません。KDE関連の各種ソフトウェア（KDE Software Compilation）のバージョンがKDE Framework 5.15とKDE Plasma 5.1.2にアップデートされたのと、ライブイメージからの起動でも日本語が入力できるようになったことくらいです。

どうしてそれだけの精度になったのかというと、未確定のことは書かなかったというだけのことです。といいますか、執筆時点のUbuntu 15.10開発版は、とても世に出せるようなものではありませんでした。しかし、ひとつひとつバグ報告をしてすべて修正してもらい、リリース前日にようやくまともになった（とその時点では思っていた）ということです。今回はその内容の具体的な解説です。

Kubuntuについてはさらに補足が必要でしょう。Kubuntu創始者にしてKubuntu Community Councilのメンバであり、リリースマネージャーであったJonathan Riddellは、15.10のリリースサイクルでUbuntu Community CouncilによりKubuntu Community Councilの座を辞任するよう要求され、それに従いました。なお、Kubuntu Community Councilにリーダー職はないので、彼は正確には実質的なリーダーでした。15.10リリース後リリースマネージャーも辞任し、Kubuntuプロジェクトから去りました[注1]。

その後、2名のリリースマネージャーが就任し[注2]、Kubuntuは引き続きリリースされることが正式に決定しました。一安心といったところです。

ここまで話がこじれるからにはいろいろあったわけですが、要約するとJonathanが知的財産（ライセンス）やポリシーがオープンではなくなっているという意見を表明し、Ubuntu Community Councilと話を進めていく中で問題のある態度を取り、それをとがめられたということのようです。UbuntuにはCode of Conduct[注3]があり、常にこれを順守しなくてはなりません。たしかに礼儀正しさは必要なことではあるのですが、有力開発者が何人もいなくなるという事態を引き起こしてまで守らなくてはいけないものなのかは、疑問の余地があるように思いますし、Jonathanの指摘はあながち的外れともいえません。

注1）https://lists.ubuntu.com/archives/ubuntu-devel/2015-October/038939.html
注2）https://lists.ubuntu.com/archives/ubuntu-release/2015-November/003443.html
注3）http://www.ubuntu.com/about/about-ubuntu/conduct

10月時点で修正されていなかったバグ

正確にいえばwishlist（要望）ですが、話がややこしくなるので、BTS（Launchpad）に登録された内容はすべてバグと呼ぶことにします。インプットメソッドがIBusからFcitxに代わったのはUbuntu 15.10開発開始まもなく（5月9日）[注4]のことであり、当初はfcitx-anthyが採用される予定だったものの、その話の中でfcitx-mozcの方が良いのではないかということになりました。そこで“Use fcitx-mozc as default ja_JP input method”[注5]というバグレポートが書かれました。

ここで1つポイントなのが、Ubuntuのデフォルトでインストールされるパッケージは、すべてmainリポジトリになくてはならないということで、fcitx-mozcも例外ではありません。変換エンジンが独立しているfcitx-anthyと違って、fcitx-mozcはmozcソースパッケージに含まれるため、Mozcそのものがmainにある必要があります。

さらにもう1つポイントがあります。mainのパッケージになるためには、ビルドに必要なパッケージもすべてmainになくてはなりません。Mozcは、高機能かつ自前でいろいろな機能を持っているため、依存するパッケージもまた多いのです。

このmainに入れるための作業を“MIR”（Main Inclurion Request）といいます。ディスプレイマネージャーの“Mir”と同じつづりですが、まったくの別物ですので注意してください。

“[MIR] mozc”[注6]が報告されたのが8月20日であり、fcitx-mozcなどMozc関連パッケージがデフォルトでインストールするようになったあとのことです。普通に考えるとMIR通過後に、デフォルトでインストールされるようにするべきではないかと思うのですが、そうはならなかったということです。なかなか興味深いです。

というわけで、10月時点で修正されなかったのは実質1つ、“[MIR] mozc”だけということになります。しかもこれは手続きですので、手伝おうにも手の出しようがありません。

日本語でインストールしたにもかかわらず、メニューが英語になってしまうバグ

では、ここから具体的に修正されたバグを見ていきましょう。驚くべきことに、少なくとも9月の間は15.10を日本語でインストールしてもメニューが英語になってしまうバグがありました。ほかの言語でも同様のはずで、誰かがバグ報告して直してくれるかなと思ったのですが、そんなことはなかったので報告しました。

メニューが英語になる場合に、真っ先に疑うべきは環境変数です。というわけでenvコマンドで見てみると、“LANGUAGE=en”という環境変数が設定されていました。これが原因で間違いありません。Ubuntu GNOMEやKubuntuでは同様にならなかったため、これらの違いを考えてみると、デスクトップマネージャーが原因なのではないかと考えました。Ubuntuのデスクトップマネージャーは LightDMです。Ubuntu GNOMEはGDM、KubuntuはSDDMです。

少し補足すると、LANG=ja_JP.UTF-8とLANGUAGE=enが同時に出力される場合、メニューの表示には後者が優先されます。LANGUAGEがない場合はLANGによって決定されるため、日本語で表示されるというわけです。

LightDMのソースコードをgrepしても、該当しそうな部分は見つかりませんでしたが、GDMもSDDMも環境変数LANUAGEは出力していません。というわけで、LANGUAGEを出力しなければいいのではないかと考えてバグ報告しました[注7]。すると、環境変数の内容はaccountservicesパッケージで決定しており、このバグを修正することによって正しくLANGUAGE=jaを出力するようになりました。とはいえ、ほかのデスクトップマネージャーを見てもわかるとおり、基本的にLANGUAGEは不要ですし、なぜかGDM_LANGという環境変数も出力していま

注4） https://bugs.launchpad.net/bugs/1439006
注5） https://bugs.launchpad.net/bugs/1468105
注6） https://bugs.launchpad.net/bugs/1486772
注7） https://bugs.launchpad.net/bugs/1502921

す。これは見たらわかるようにGDMのための環境変数ですが、Ubuntuでパッケージになっている GDMはこの環境変数を出力しないパッチが当たっています。そのような奇妙な状態になっているにもかかわらず、結局この2つの環境変数は、現在のバージョンでも出力されています。

Ubuntuフレーバーで Fcitxが起動しなくなるバグ

これはUnityではないデスクトップ環境であり、かつCJKV(日中韓越)ではないロケールでキーバインドがインプットメソッド(IBus、ないしFcitx)の制御にならないようにim-configパッケージを修正したつもりが、CJKVすべてを巻き込んでインプットメソッドの制御から逃れてしまい、Fcitxが起動しなくなったというバグでした[注8]。つまりはリグレッションです。

これではわかりにくいので、実際に動いているコードを見てください(**リスト1**)。"IM_CONFIG_DEFAULT_MODE=cjkv"がximになっていたのがバグであり、IBusもFcitxも起動しなくなったのです。

どうして"xim"になるとインプットメソッドが起動しなくなるのかは、/usr/share/im-config/data/79_xim.confと/usr/share/im-config/data/79_xim.rcを見ると一目瞭然です。

そもそもどうしてこんな事態になり、かつこれで修正されるのかがよくわからないのですが、この手の謎ハックは結構あります。

fcitx-frontend-qt5 パッケージが削除されるバグ

Ubuntu 15.04から、fcitx-frontend-qt5パッケージはインストールイメージの中に含まれています。15.10でも同様ですが、なぜかインストール完了直前に削除されてしまっていました[注9]。なお、このパッケージがないと、FcitxでQt5アプリケーションに日本語を入力できなくなります。

インストーラ(Ubiquity)でインストールした場合、インストール終了直前に不要なパッケージをアンインストールします。必要か不要かは、何かしらからか依存されているか否かで判断されます。fcitx-frontend-qt5はlanguage-selectorから呼ばれており、日本語を選択した場合はUbuntuだとアンインストールされずに残り、Ubuntuフレーバーだとインストールイメージには含まれていないため、リポジトリから取得してインストールします。しかし、アンインストールするコードにバグがあり、fcitx-frontend-qt5が必要であるにもかかわらずアンインストールされてしまったということです。

修正されたコードを見ると、"fcitx-frontend-qt5:amd64"のような":"のあとにアーキテクチャが入るパッケージだとアンインストールしてしまっていたようです。15.04以前はどうして問題にならなかったのか不思議です。

fcitx-mozcが インストールされないバグ

前述のとおり、インストール時に日本語を選択すると、fcitx-mozcとMozc一式がインストールされるはずなのですが、Ubuntuフレーバーだとそのとおりに動いていたものの、肝心のUbuntuではそうではなかったのです[注10]。これは完全に原因不明で途方に暮れていたのですが、ある日[注11]突然解決します。原

注8) https://bugs.launchpad.net/bugs/1481025

注9) https://bugs.launchpad.net/bugs/1503297
注10) https://bugs.launchpad.net/bugs/1506502
注11) ある日というか、リリースの2日前になのですが。

リスト1　im-config 0.29-1ubuntu7の/etc/default/im-config

```
if [ -n "$XDG_CURRENT_DESKTOP" -a "$XDG_CURRENT_DESKTOP" = 'Unity' ]; then
    # Start best input method unless overridden below
    IM_CONFIG_DEFAULT_MODE=auto
else
    # Start best input method only if CJKV environment and not overridden below
    IM_CONFIG_DEFAULT_MODE=cjkv
fi
```

因はfcitx-mozcがMIRを通過しておらず、mainにないことでした。割にやっつけ的なコメントとともにfcitx-mozcがmainに入ると、インストーライメージに収録されました。よって、fcitx-mozcがインストールされないというのは正しくなく、正確にはインストールイメージに収録されていなかったのです。ただ、筆者もインストールイメージに収録されるとは聞いておらず、寝耳に水でした。完全に狐につままれたコメントを、該当のバグに寄せています。

なお、この副作用としてライブイメージからでも日本語の入力が可能になっています。

fcitx-mozcのパッケージが空になってしまうバグ

「やった、fcitx-mozcがインストールされるようになった！」と喜んだのも束の間、実際にインストールテストをしてみると、たしかにインストールはされているものの、どうやっても有効になりません。そこでいろいろと調査したところ、パッケージがインストールされているという情報はあるものの、パッケージを構成するファイルが根こそぎ何もないということに気づきました[注12]。このバグ報告に添付した画像を再掲します（**図1**）。

ここは1つのファイルがないことだけを示していますが、実際にはすべてのファイルがありませんでした。フォルダはすべてあったのですが。

原因はfcitx-frontend-qt5と同じで、インストール完了直前に削除する処理のバグでした。このバグが修正されたのがリリース前日で、それをもとにインストールイメージが作成されました。これがリリース版になるのかなと思ったのですが、次のインストールイメージも作成され、それがUbuntu 15.10としてリリースされました。本当にギリギリの修正であったことがよくわかります。

余談ですが、筆者はリリースまでにインストールテストを、Ubuntuを11回、Ubuntu GNOMEを6回、Xubuntuを4回行っています。もちろん過去最多です。

注12） https://bugs.launchpad.net/bugs/1508121

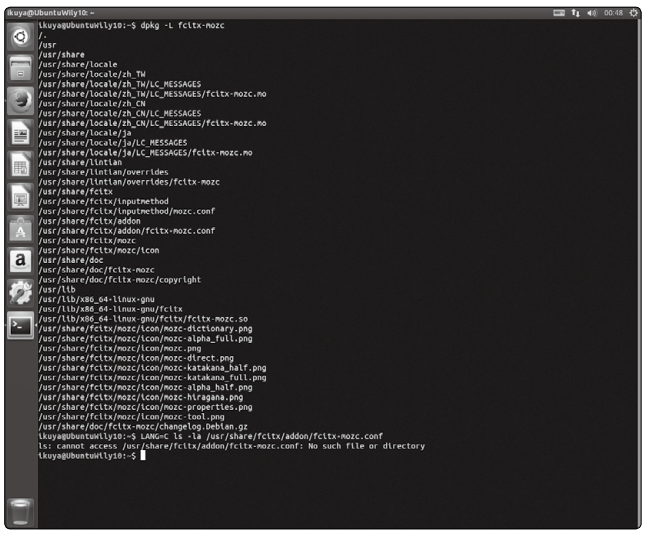
修正されなかったバグ

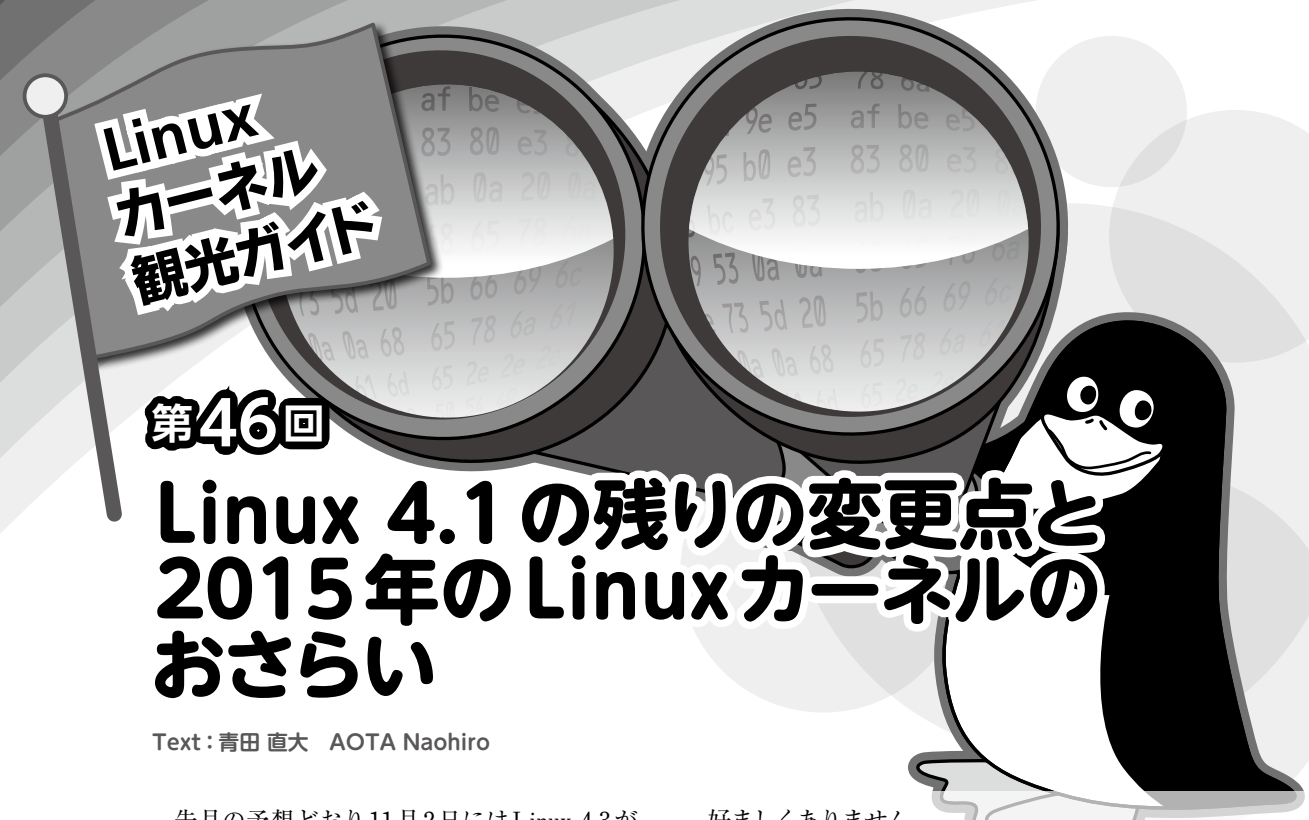
こうして15.10のリリース日を迎えたわけですが、この前後に初回ログイン時にFcitxがfcitx-mozcを自動的に認識して入力メソッドとして追加するはずが、そうはならないというバグが見つかりました[注13]。すでに開発版の16.04では修正されています。まともな動作テストが可能になったのがリリースの前日で、為す術がなかったと言えばそうなのですが、初めてインプットメソッドを採用したUbuntu 6.06から、ずっと日本語を選択してインストールするとインプットメソッドが正しく起動していたのですが、今回、初めてそうはならないリリースとなってしまったのがとても残念です。また現在、存在しない英語キーボードを認識するという仕様になっているのですが、これだと困るということで修正が進んでいます[注14]。なお、いずれも先月号の当連載を担当された柴田さんにより進められています。

いずれも日本語Remixでは修正されているため、日本語関連のトラブルを回避したいという場合にはお勧めです。SD

注13） https://bugs.launchpad.net/bugs/1465535
注14） https://bugs.launchpad.net/bugs/1514544

図1　パッケージの情報はあるものの、ファイルがない





第46回

Linux 4.1の残りの変更点と2015年のLinuxカーネルのおさらい

Text：青田 直大　AOTA Naohiro

先月の予想どおり11月2日にはLinux 4.3がリリースされました。Linux 4.4の開発も順調に進み、すでにマージウィンドウがクローズされ、11月23日にはLinux4.4-rc2がリリースされています。

今月は先月に引き続き、Linux 4.1の残りの変更点を紹介していきます。

Linuxのアクセス制御：capabilities

Linuxでは、基本的にはアクセス制御として、長く使われているuser/groupシステム、およびrootへの特権付加を実装しています。さらにLinux 2.2以降では、capabilitiesというシステムも導入しています。これはrootに一括して付与されていた特権を分離するものです。これによって、必要な権限だけを付与したプロセスを作ることができ、万が一プロセスが乗っ取られた場合のシステムへの影響を限定できます。

たとえば、“ping”を送るためにはRAWソケットを使う必要があり、これには特権が必要です。pingは一般ユーザでも実行したいプログラムなので、“ping”にset-uidしておいてroot権限で実行されるようにしてあります。しかしこれでは、rootが実行し得ることはなんでもできてしまい好ましくありません。

そこでpingは、main()の冒頭で**リスト1**に示すlimit_capabilities()という関数を呼び出しています。この関数内では、まずプロセスの現在のcapabilitiesを取得（cap_get_proc()）し、cap_cur_pに保管しています。このcap_cur_pをcap_get_flag()を使って、Permitted capabilitiesのCAP_NET_ADMINおよびRAWソケットの作成を許可するCAP_NET_RAWが立っているかどうかを調べます。Permitted capabilitiesは、プロセスが取得できるcapabilitiesを示しています。一度、Permitted capabilitiesから落ちたcapabilityは、再取得できません。すなわち、ここではCAP_NET_ADMINおよびCAP_NET_RAW以外のcapabilityを使えないようにしていることになります。

そのうえで、socketを作成する前後でenable_capability_raw()、disable_capability_raw()をそれぞれ呼び出しています（**リスト2**）。CAP_NET_RAWを必要なときだけ有効にすることで、万一の場合のシステムへの影響を限定できます。

なお、最近のシステムであれば“ping”プログラムがset-uidではなく、file capabilityによって権限付与されていることもあります。file capabilityとはset-uidのcapability版のようなも

▼リスト1　iputils-s20150815/ping_common.c

```
void limit_capabilities(void)
{
#ifdef CAPABILITIES
    cap_t cap_cur_p;
    cap_t cap_p;
    cap_flag_value_t cap_ok;

    cap_cur_p = cap_get_proc();   /* 現在の capabilities を取得 */
    if (!cap_cur_p) {
        perror("ping: cap_get_proc");
        exit(-1);
    }

    cap_p = cap_init();
    if (!cap_p) {
        perror("ping: cap_init");
        exit(-1);
    }

    cap_ok = CAP_CLEAR;           /* CAP_NET_ADMIN が permit されていれば, 続けて permit を取得 */
    cap_get_flag(cap_cur_p, CAP_NET_ADMIN, CAP_PERMITTED, &cap_ok);

    if (cap_ok != CAP_CLEAR)
        cap_set_flag(cap_p, CAP_PERMITTED, 1, &cap_admin, CAP_SET);

    cap_ok = CAP_CLEAR;           /* 同様に CAP_NET_RAW の permit も取得 */
    cap_get_flag(cap_cur_p, CAP_NET_RAW, CAP_PERMITTED, &cap_ok);

    if (cap_ok != CAP_CLEAR)
        cap_set_flag(cap_p, CAP_PERMITTED, 1, &cap_raw, CAP_SET);

    if (cap_set_proc(cap_p) < 0) {  /* 新しい capabilities の適用 */
        perror("ping: cap_set_proc");
        exit(-1);
    }

    if (prctl(PR_SET_KEEPCAPS, 1) < 0) { /* 次の setuid で capabilities がクリアされないようにする */
        perror("ping: prctl");
        exit(-1);
    }

    if (setuid(getuid()) < 0) {
        perror("setuid");
        exit(-1);
    }

    if (prctl(PR_SET_KEEPCAPS, 0) < 0) {
        perror("ping: prctl");
        exit(-1);
    }

    cap_free(cap_p);
    cap_free(cap_cur_p);
#endif
    uid = getuid();
    euid = geteuid();
#ifndef CAPABILITIES
    if (seteuid(uid)) {
        perror("ping: setuid");
        exit(-1);
    }
#endif
}
```

ので、プログラム実行時にroot権限の代わりに、指定したcapabilityのみを付与することが可能になります。“getcap”を使ってpingのfilecapabilityを見てみると、たしかにCAP_NET_RAWが有効(“effect”)で、許可(“permit”)されていることがわかります(**図1**)。

Single User Linux

このようにLinuxのアクセス制御は柔軟にできていますが、そうした権限分離が必要でない場合もあります。たとえば、Linuxを動かすような組み込みシステムでは、そのほとんどの機能をroot:rootで動作するinitプロセスに押し込めていることがあるようです。こうした環境ではアクセス制御のコードは無駄にしかなりません。

そこでLinux 4.1ではuser/groupおよびcapabilityシステムを、カーネルコンパイル時の設定で削除できるpatchがマージされました。削除した場合、すべてのプロセスがroot:root (UID: 0, GID: 0)で、さらにすべてのcapabilitiesを持った状態で動作するようになります。また、setuid()やcapget()といったuser/groupまたはcapabilitiesに関するシステムコールがビルドされなくなります。こうした機能削除によって、およそ25KBほどカーネルのテキスト領域が小さくなります。

TraceFS

システムの挙動を調べるにあたって、Linuxではtracingという機能を使うことができます。これはカーネルおよびユーザのプログラムのさまざまな場所にhookを仕込んで、その時点での変数の内容をダンプしたり、あるいはどの関数がどの関数を呼んでいるかを「トレース」するためのシステムです。

この機能のインターフェースは/sys/kernel/debug/tracing/下のファイル群へのアクセスで実現されています。ここで/sys/kernel/debugというのがprocファイルシステムのと同様の疑似ファイルシステムであるdebugfsのmountポイントになっています。debugfsはLinuxカーネルのさまざまなデバッグ機能へのインターフェースを提供しているファイルシステムで、tracing機能のインターフェースもその一部として実現されていたわけです。

この実装には2つの問題がありました。1つはdebugfsの下に実装されていることです。すなわち、tracingを使いたいだけなのにdebugfsの下のさまざまなdebug機能もアクセス可能にしてしまいます。もう1つの問題点はdebugfsがuserlandからのmkdirをサポートしていないことです。これはtracingのinstanceという機能にかかわってきます。

デバッグしていると複数の個所にトレースポイントを入れたくなることがあります。それらのダンプがすべて同じバッファに出力されると、その分類が面倒になってしまいます。そこでトレースの出力バッファを複数作成し、それぞれ、どのイベントのトレースが出力されるのかを設定できるようにするのが、tracing instanceの機能

▼図1　pingのfile capabilityの確認

```
$ sudo getcap /bin/ping
/bin/ping = cap_net_raw+ep
```

▼リスト2　iputils-s20150815からpingのsocket作成部分

```
ping.c:
    /* Create sockets */
    enable_capability_raw();
    if (hints.ai_family != AF_INET6)
        create_socket(&sock4, AF_INET, hints.ai_socktype, IPPROTO_ICMP);
    if (hints.ai_family != AF_INET)
        create_socket(&sock6, AF_INET6, hints.ai_socktype, IPPROTO_ICMPV6);
    disable_capability_raw();
```

です。instanceの作成には、“mkdir instances/<名前>”を使います(**図2**)。できたディレクトリの中には、トップレベルであるtracingと似たようなファイルが自動的に作られています。これらファイルがトップレベルの対応するファイルと同じような働きを行います。たとえば、instances/fooとinstances/barを作り、fooではsched_wakeupを、barではkmallocをそれぞれ有効にしてみます。するとたしかに、それぞれのinstanceに各イベントについてのトレースのみが入っていることがわかります。

さて、このtracing instanceには、**図2**のようにmkdirが使われています。ところが、debugfsはmkdirをサポートしていません。そのため、この部分は特別なhackを用いて実装されていました。

以上の2つの問題点を解決するために、Linux 4.1ではdebugfsからtracefsが分離することになりました。tracefsはその名が示すとおりに、tracing専門の疑似ファイルシステムです。debugfsから分離することで、ほかのdebug機能を見せることなく、tracingを使うことができるようになりますし、mkdirの部分もきれいに実

▼図2　tracing instanceを使う

```
# cd /sys/kernel/debug/tracing/
# ls
available_events            current_tracer         instances        per_cpu             set_event          ➥
  set_graph_notrace  trace_options        tracing_thresh
available_filter_functions  dyn_ftrace_total_info  kprobe_events    printk_formats      set_ftrace_filter  ➥
  snapshot           trace_pipe           uprobe_events
available_tracers           enabled_functions      kprobe_profile   README              set_ftrace_notrace ➥
  trace              tracing_cpumask      uprobe_profile
buffer_size_kb              events                 max_graph_depth  saved_cmdlines      set_ftrace_pid     ➥
  trace_clock        tracing_max_latency
buffer_total_size_kb        free_buffer            options          saved_cmdlines_size  set_graph_function ➥
  trace_marker       tracing_on
# mkdir instances/foo
# mkdir instances/bar
# ls instances/foo/
available_tracers  buffer_total_size_kb  events       per_cpu    set_ftrace_filter   snapshot  trace_clock  ➥
  trace_options  tracing_cpumask      tracing_on
buffer_size_kb     current_tracer        free_buffer  set_event  set_ftrace_notrace  trace     trace_marker ➥
  trace_pipe     tracing_max_latency
# echo 1 > instances/foo/events/sched/sched_wakeup/enable
# echo 1 > instances/bar/events/kmem/kmalloc/enable
# head  head instances/foo/trace_pipe
CPU:0 [LOST 17112 EVENTS]
     plasmashell-2465  [000] d..3 25361.127194: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
     plasmashell-2465  [000] d..3 25361.127200: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
     plasmashell-2465  [000] d..3 25361.127206: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
     plasmashell-2465  [000] d..3 25361.127212: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
     plasmashell-2465  [000] d..3 25361.127217: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
     plasmashell-2465  [000] d..3 25361.127222: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
     plasmashell-2465  [000] d..3 25361.127228: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
     plasmashell-2465  [000] d..3 25361.127234: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
     plasmashell-2465  [000] d..3 25361.127240: sched_wakeup: comm=QXcbEventReader pid=2467 prio=120 target_cpu=003
# head instances/bar/trace_pipe
CPU:3 [LOST 52416 EVENTS]
              X-2288  [003] ...1 25377.327571: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039f000 bytes_req=…
              X-2288  [003] ...1 25377.327572: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039c200 bytes_req=…
              X-2288  [003] ...1 25377.327574: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039c600 bytes_req=…
              X-2288  [003] ...1 25377.327575: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039e600 bytes_req=…
              X-2288  [003] ...1 25377.327576: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039ea00 bytes_req=…
              X-2288  [003] ...1 25377.327577: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039f800 bytes_req=…
              X-2288  [003] ...1 25377.327579: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039e000 bytes_req=…
              X-2288  [003] ...1 25377.327580: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039de00 bytes_req=…
              X-2288  [003] ...1 25377.327581: kmalloc: call_site=ffffffff815cc380 ptr=ffff88017039c800 bytes_req=…
```

装しなおすことができます。

分離はされましたが、ユーザからの見た目はほとんど変わりません。tracefsは/sys/kernel/debug/tracingの下に自動的にmountされています(**図3**)。またdebugfsと同様に、tracefsをmountするためのディレクトリである/sys/kernel/tracingが自動的に作成されるようになっています。

そのほかのLinux 4.1の新機能

Linux 4.1のそのほかの新機能について簡単に紹介します。指定したサイズのファイル書き込み領域を確保するfallocate()というシステムコールがあります。このシステムコールは、領域の解放ができるようにするFALLOC_FL_PUNCH_HOLEや、指定した領域を0埋めするFALLOC_FL_ZERO_RANGEなど拡張が続いていました。Linux 4.1ではファイルの指定したoffsetに指定した長さの領域を挿入できるFALLOC_FL_INSERT_RANGEフラグが登場しました。すなわち、ファイルの途中に(0埋めに見える)holeを挿入し、挿入位置より後ろにもともとあったデータはholeの後ろにシフトされます。

ほかにもファイルシステム関連としてProject Quotaサポートの VFSへの導入が挙げられます。これまでVFSでは ユーザ単位あるいはグループ単位でのQuotaをサポートしていました。XFSでは、これら以外に「プロジェクト」単位のQuotaをサポートしています。これは各ファイルに「プロジェクトID」を割り当て、そのIDごとにQuotaをかけるというものです。ディレクトリ内の新規ファイルにプロジェクトIDを継承させるフラグもあります。つまり、大雑把にいえばディレクトリツリー単位でのQuotaがかけられるようになる、ということです。Project Quotaは、これまではXFSでしか使われていませんでしたが、ext4にも導入する足掛かりとしてかVFS上にProject Quotaに対応する一般化されたコードが導入されています。

2015年のLinuxカーネル

今回は1月号なので、2015年のLinuxカーネルについて振り返ってみましょう(**表1**)。2015年最初のLinuxカーネルは2月にリリースされたLinux 3.19です。その後、4月にLinux 4.0、6月に4.1、8月に4.2、11月に4.3とリリースが行われています。この最近は、9週間か10週間でのリリースが続いているので、Linux 4.4を見るのは来年ということになりそうです。

今年のLinuxカーネルの最も大きな変更点といえば、一目瞭然でそのバージョン番号の変化でしょう。LinusがGoogle+で投票を行った結果、Linux 3.20はLinux 4.0とバージョン番号を変えてリリースされることとなりました。そんな理由ですので、もちろん3.19と4.0との間で大きな変化があったわけではありません。Linux 3.0が出たのが2011年7月、Linux 4.0が出たのが2015年4月ということでまた4年後5.0が出てくるのでしょうか?

2015年のLinuxカーネルの機能面での変化を、分野ごとに見てみましょう。まず、セキュリティ関連ではメモリのアクセス範囲を検査する機能として、ハードウェアの支援を使ったIntel MPXのサポート、ソフトウェアで実現したものとしてKASANが実装されました。また、セキュリティパッチの適用という面で考えれば、カーネルを再起動することなく関数を置き換えるlivepatchが導入されました。複雑なpatchの場合にどうやって一貫性を保つかはまだ議論が続いているようです。一方でモジュールロード時

▼図3　tracefs として mount されている

```
$ mount|grep trace
tracefs on /sys/kernel/debug/tracing type tracefs (rw,relatime)
$ ls -ld /sys/kernel/tracing
dr-xr-xr-x 2 root root 0 Nov 23 17:20 /sys/kernel/tracing
```

のコードを使うことで、アーキテクチャ依存部分をなくしx86以外にもlivepatchを使えるようにする方向にも開発が進んでいます。

次に仮想化・コンテナ関連を見てみましょう。いずれもまだ紹介できていませんが、KVM、Xen関連ではVirtio GPU、コンテナ関連ではPIDs cgroupとblkiocgroupのwritebackサポート、両方にかかわるものとしてuserfaultfdといった機能が挙げられます。Virtio GPUは、virtioを使ってGPUを仮想化するものです。PIDs cgroupは新しいcgroupのsub-systemでグループ内のプロセスIDの数を制限します。userfaultfdはユーザ空間でpage faultの発生をキャッチし、faultに対応できるようにするものです。仮想マシンやコンテナのlivemigration時に、必要なpageをon-demandに持ってくるシステムに使われることが予想されます。

ファイルシステム関連の分野では、Ext4、F2FSに暗号化機能が実装され、ファイルシステムレベルでの暗号化が共通化されつつあります。また、NVDIMMといった新しいデバイスに対応するため、pageを迂回したデータアクセスを可能にするDAXや、NVDIMMへのアクセス用のライブラリとなるlibnvdimmが作られています。さらに、高速なブロックデバイスに対して割り込みではなく、pollingでデータを受信することを可能にするpatchも導入されています。またほかにもタイムスタンプの更新を遅延することでmetadataの更新を削減するlazytime機能も実装されました。

最後にカーネルやドライバ開発関連の機能を見ていきましょう。カーネル開発ツールに関しては、gdbのヘルパスクリプトが追加されデバッグに便利なコマンドが使えるようになったことが挙げられます。ほかにもDevice Mapperの新しいターゲットであるdm log-writeによって、ブロックデバイスへの書き込みを記録・再現ができるようになり、ファイルシステム開発に役に立つことでしょう。また、どちらかといえばシステム開発系となりますが、EFIなどのNVRAM領域にユーザが書き込んだデータを記録し、再起動後も見ることを可能にする機能である/dev/pmsg0も追加されています。

また、2015年はeBPFの適用範囲が広がった年でもありました。eBPFは、もともとはpacket filtering用の機能であるBPFが、カーネルのヘルパ関数の呼び出し、マップの読み書きという2つの機能面で拡張されたものです。eBPFはJITされ、高速かつ安全に動作することから、さまざまな場所にeBPFが導入され、より柔軟な機能を実現しています。具体的にはsocketに結びつけてフィルタリング、eBPFによる通信フローのクラス分け、kprobeに結びつけてイベントを記録するかどうかの判断が行えるようになっています。さらにはperfからCのBPFコードを指定することで、そのコードがclangによってBPFバイトコードにコンパイルされ、カーネルに読み込まれるといった機能も実装されています。SD

▼表1　2015年のLinuxカーネルの進化

	1月	2月	3月	4月	5月	6月	7月	8月	9月	10月	11月	12月
Linux		3.19		4.0		4.1		4.2			4.3	
セキュリティ		MPX		KASAN livepatch								
仮想化・コンテナ								Virtio GPU writeback cgroup			PIDs cgroup userfaultfd	
ファイルシステム				DAX lazytime		Ext4 暗号化		libnvdimm				polling
カーネル開発				gdb ヘルパ /dev/pmsg0		dm log-write						
eBPF		socket				cls_bpf kprobe						perf

January 2016

NO.51

Monthly News from jus

Japan UNIX Society

法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp　日本UNIXユーザ会
前野 洋史　MAENO Hiroshi　mahiro@smkwlab.net　九州産業大学大学院情報科学研究科
神屋 郁子　KAMIYA Yuko　yuko@smkwlab.net　九州産業大学情報科学部

今どきの女子大生のIT教育&今どきのインターネット研究

今回は、8月に京都で行った研究会と、10月に神戸で行ったインターネットコンファレンスの模様をお伝えします。

jus研究会 京都大会

■IT系女子大生の育て方
【講師】宮下 健輔（京都女子大学）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2015年8月8日（土）15:15～16:00
【会場】京都リサーチパーク1号館4階 会議室A

京都大会は京都女子大学の宮下先生を講師にお迎えし、女子大生へのIT教育についてご講演いただきました。参加者は24人でした。

■カリキュラムの概要

宮下さんが所属する京都女子大学現代社会学部は、2000年に開設された比較的新しい学部です。カリキュラムとしてはかなり幅広い分野の学問を扱っていますが、予備校では文系に分類されているらしく、理数系が得意な学生の入学は少ないようです。そこで大学側も、数学は苦手だがパソコン好きな学生に狙いを定め、いろいろなものを与えて能力を伸ばす方針を採っています。そんな現代社会学部ゆえ、カリキュラムも時代の要請に沿って徐々に変化していますが、情報系の授業はどの分野へ進むにしても必須の基本スキルであるとともに、現代社会を見る側面の1つであると位置づけられており、相当数の科目が実施されています。

現在の情報系の授業は3本柱で構成されています。1つめはアルゴリズムとプログラミングで、開設当初から一貫してRubyを教えており、エディタはEmacsを使っています。毎年約150人が受講していて、最初の授業でEmacsやCygwinなどの大インストール大会が行われる様子は阿鼻叫喚（あびきょうかん）だそうです。2つめはネットワーク関連で、こちらはCisco Networking Academyのプログラムを採用しています。UTPケーブルの作成からVLANやOSPFなどの設定までを学びます。3つめは情報文明論や知的財産権論といった、どちらかといえば社会的な側面からの知識を身につける科目が並んでいます。また、近年はさらに高度の専門性を身につける情報課程と呼ばれる科目群も用意されました。

■さかんな課外活動

さらに、宮下さんの研究室では課外活動をさかんに行っています。具体例としては、関西オープンフォーラムのネットワーク構築、オープンソースカンファレンス京都のボランティア、電子部品アクセサリー作り、Rails Girls[注1]などがあります。これらの活動はたいてい、宮下さんが「こんなんあるで」と言ってプロジェクトの紹介だけを行い、あとは学生たちに自主的にやらせているそうです。課外活動を楽しむためには勉強が必要になるので、それが彼女たちの能力を伸ばすことにつながっているようです。

最後に宮下さんから、理系ではないPC好きは少なからず存在する、活動を楽しみたいと思えば勉強も

注1）URL http://railsgirls.jp/

楽しくなる、そしてIT系の企業は学生を受け入れてくれる土壌が存在するのが良いところであるという3点のまとめが紹介されました。参加者にも大学の先生が何人かいて、学生の教育について熱心な質疑応答がなされました。宮下さんの説明はとてもわかりやすく、参加者にとっても有意義な講演だったと思います。

インターネットコンファレンス2015

■インターネットコンファレンス2015
【日時】2015年10月13日（火）～14日（水）
【会場】サンパル（ひょうご産業活性化センター ビジネスプラザ 兵庫ホール）

本学会は、jus、公益財団法人ひょうご産業活性化センター、日本学術振興会協力研究委員会インターネット技術第163委員会、日本ソフトウェア科学会インターネットテクノロジー研究会、WIDEプロジェクトの5つの主催団体と19の協賛団体で開催されました。論文発表は、コンテンツ配信、セキュリティ、インターネット基盤の3セッションで9件、WIP（Work in Progress）セッション3件、ポスター・デモ展示12件と招待講演が3件あり、参加者は66名でした。

■各種表彰

本学会では優秀な論文／発表を表彰しています。各賞の受賞者は次のとおりです。

論文賞：

- 「大規模HTTPライブストリーミング配信におけるサーバログを用いた視聴遅延の推測手法の提案」
  二宮 恵、長 健二朗
  （IIJイノベーションインスティテュート）

ポスター賞：

- 「DNSログ解析によるDGAを用いたマルウェア検知のための予備調査」
  渡辺 拳竜、池部 実、吉田 和幸（大分大学）

プレゼンテーション賞：

- 「Consumer-driven Adaptive Rate Control for Real-time Video Streaming in Named Data Networking」
  米田 孝弘（パナソニック）

学生奨励賞：

- 「拡張現実技術（AR）によるコンピュータネットワークの可視化システム」
  鄒 暁明（神戸情報大学院大学）
- 「Application Layer Multicastを用いたPub/Sub基盤と連携動作するOpenFlow Multicastの設計および実装とメンバ管理オーバーヘッドの評価」
  藤田 雅浩（京都産業大学）
- 「通信制御系に対するモデルベース縮退運転システム」
  佐々木 翼（電気通信大学）

■論文紹介

「大規模HTTPライブストリーミング配信におけるサーバログを用いた視聴遅延の推測手法の提案」

ライブストリーミングにおける視聴遅延にはさまざまな要因があります。本論文では、HTTPライブストリーミングにおける視聴遅延をモデル化し、Webサーバのログのみを用いてユーザごとの視聴遅延を推測する手法を提案していました。分析の結果、「セグメントの長さとクライアントが試聴開始時においてバッファリングするセグメント数が視聴遅延の長さを決める大きな要因となること」、「視聴ごとの視聴遅延のばらつきの幅が視聴遅延の平均を中心にして±2セグメントの範囲に収まること」、「プレイリストの長さを短くすることで視聴遅延を短くし、そのばらつきを小さくできる可能性」が示されました。

◆ ◆ ◆

発表された論文はインターネットコンファレンス2015のWebページ[注2]から参照できます。2016年度は鳥取大学の大森幹之氏を実行委員長に迎えて開催予定とのことです。SD

注2）URL http://www.internetconference.org/ic2015/

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

第49回 東北TECH道場の紹介

テクノロジーを使って復興を支援することができないか。そう考えてたどり着いた方針の1つが、東北のIT技術者の支援と、長期的な視野でITを地域に根付かせることでした。その人材育成の取り組みの1つ、東北TECH道場について紹介します。

東北の若者にプログラミングを教える

東北TECH道場[注1]は2012年の11月に開始されました。

2011年3月の東日本大震災の後、東北の復興と発展のために多くの方が引き続き精力的に活動されています。まだ傷は癒えていない地域が多くあることも事実ですが、テクノロジーを使ってそれらの活動を支援できないか、またテクノロジーを使って頑張っている開発者の皆さんを支援できないかと考えて始められた取り組みです。

Google、ゴーガ、イトナブ石巻の協力により運営されており、Hack For Japanからは筆者が講師、および運営スタッフとして参加しています。最初は宮城県は仙台、石巻、岩手県は滝沢という3つの場所でスタートし、その後、岩手の北上と釜石、盛岡(滝沢から移動)が加わり、不定期ではありますが福島の会津若松やいわきでも開催されてきました。そして2015年9月から始まった第9期では、新たに青森県の八戸と福島県の郡山が新たな道場として加わりました。

道場という名前が示すように、座学で講義をするというよりは実際に手を動かして開発していくことを重視しており、東北各地の道場でおもにAndroidアプリの開発を進めています。当初はAndroidアプリに限らず、Web、Google Maps APIなど、さまざまな技術要素を網羅する予定だったのですが、初回にさまざまな要素を盛り込みすぎたという反省と、スマートフォンのアプリ開発は自分の持っている手元の端末でアプリが動くため成功体験を得やすいという観点から、Androidでの開発に絞ることにした経緯があります。

2ヵ月から3ヵ月の期間を1つの区切りとして進めて、期の締めに成果発表会を行います。本誌発売時点では第9期が終了したところで、年が明けて2016年からは第10期が開催される予定です。この記事の時点で道場開始から3年が経過し、第1期から継続して参加してくれている道場生は今では一人で一通り開発を進められるまでに成長してくれており、新たに道場に参加してくれた初心者を指導できるようになりました。道場というスタイルをとっていることから役回りとしては師範代と言える存在かと思います。

そんな彼らも最初はアプリ開発はもちろん初めてで、プログラミングも学校で少しやったことはあるけれども……というところからのスタートでした。そこから積極的に道場に参加して、講師から教わるだけでなく自分自身でも書籍などで技術を吸収し、期ごとにアプリをバージョンアップして機能を増やしたり完成度を高めるなど研鑽を積んだ結果です。

筆者がかかわることになったきっかけは、この連載でも毎年レポートしている石巻ハッカソンの2012年7月の第1回目で行われたIT Bootcampにて石巻工業高校の高校生たちと出会い、継続してサポートをしていきたいと考えていたところに、Hack For Japanで一緒にスタッフとして活動していたGoogleの方から講師をやらないかと声をかけていただいたことから始まりました。

それまでもコミュニティ活動などでAndroidのアプリ開発について誰かに教えることはありましたが、この道場のように継続してサポートしていくということは初めてで、道場生たちの「作ったアプリが自分のスマートフォンで動いた！」というときの

注1　http://www.tohokutechdojo.org/

感動がこちらにも伝わってくるような場面に何度も遭遇できるのは講師冥利につきるというものです。

ある日の石巻道場

通常、道場は月に1、2回、土曜日の13時から18時にかけて開催されます。

オープニング

オープニングではお互いの状況報告を兼ねて今日やりたいことを簡単に発表します。初めて参加するという人がいる場合は自己紹介タイムも兼ねています。また、同時開催の他の道場がある場合はハングアウトで結んで情報交換をします（**写真1**）。

講義タイム

道場生全員に知ってほしいこと、何か新しい技術の発表などがあった場合には、30分から1時間程度の講義を行うときもあります。この日は「より良いコードの書き方」と題して、プログラムを書くにあたって気をつけるべきことを筆者が話しました。

“「アプリが動けばいい、コードの見た目のきれいさなんて関係ない……」と思っていた人は正直に手を挙げてください”という問いから始まり、“なぜきれいなコードを書く必要があるのか”ということを30分ほどかけて説明しました。

開発タイム

時間いっぱいまで道場生はひたすらアプリの開発をします（**写真2**）。基本的には各自それぞれがオリジナルのアプリを開発し、詰まったりわからないことがあれば、もちろん講師が（時には一緒に悩みながら）全力でサポートします。通常、講師は1人から2人体制で、道場によっては地元のエンジニアの方がチューターとして手伝ってくれる場合もあります。初参加の道場生が多い場合は、初心者グループでそろってハンズオン課題を進める日もあります。

頑張って開発して甘いモノがほしくなったり、お腹が減ってきたという人のために、おやつのお菓子も提供されます。

クロージング

クロージングではオープニングと同様にその日の成果を報告します。やったことを自分で整理して発表することで、次に向けての課題も見えてくるはずです。

オンラインでのコミュニケーション

月に1、2回の開催だけで補いきれない部分は、質問があるときは道場生と講師が集まるコミュニティがGoogle+に設けてあり、そこで質問を投稿すると講師が答えてくれるようになっています。また、オンライン道場と称して、ハングアウトで東京にいる講師とつないで質問、相談を受け付けることもあります。

成果発表会

各期の最後の日は成果発表会を行っています。参加者それぞれが頑張ってきた成果を発表するのです

◆写真1　ハングアウトで結んで他の道場とのやり取り

◆写真2　ひたすら開発

が、東北の各地の道場をハングアウトで結んで行います。アプリを完成させることができた人はGoogle Playストアにアップロードして世界に向けて公開します。また、その期に完成しなかった人もできたところまでをデモし、基本的に全員が発表用のスライドを作成して次の期に向けての抱負、苦労したところ、工夫したところなどを1人5分程度で発表します。

各回のオープニングとクロージング、そして発表会の様子は「ハングアウト オンエア」を使ってストリーミングされ、アーカイブとして動画が残されています[注2]。よろしければご覧ください。

お楽しみ

アプリ開発は決して敷居の高いものではないのですが、1つのアプリとして公開できるところまで仕上げるにはそれなりに苦労も伴います。しかし、頑張ってアプリを完成させた道場生には、賞や景品などのお楽しみ企画が出るときがあります。2013年の8月には、アプリを公開できた人への賞品として東京にあるGoogleの日本オフィスを見学してランチをいただき、さらにGoogleの社員の皆さんの前で自分の作ったアプリをプレゼンするという貴重な機会がありました。

道場主の役割

現在は東北各地に広がる道場ですが、道場運営において大事なことは、地元で道場主として取りまとめをしてくれる方々がいることです。会場を準備したり、時には大学や高校などを訪問して活動を紹介し参加者を募るということをしてくれています。このように熱心に取り組んでいただける道場主なしには東北TECH道場は成り立ちません。

道場生たちと将来

参加してくれた人全員がAndroidのアプリ開発者になるというわけではないと思います。アプリ開発はあくまでITの技術、プログラミングに触れるためのきっかけと捉えて、より深くAndroidの技術を探求していくのはもちろん良いですし、そこから派生して別の技術での開発、もしくはプロデューサーのような方向を目指していくというケースもあると思います。

最初はわからなくて当たり前で、小文字のエル(l)と大文字のアイ(I)を間違えて入力してビルドが通らないなど戸惑うことも多いと思いますが、少しずつ積み重ねていければ良いのです。どんなに経験を積んだエンジニアでも最初はみんな初心者だったのです。また、現在はどの道場にも東京などから講師を派遣していますが、将来的にはその地元の道場生の中から講師となる人が出てきて、地元だけで道場を回せるようになるのが最終目標です(**写真3**)。

講師を募集しています

プログラミングを学びたいと目を輝かせて参加してくれる道場生たちと時間を共にすることは教える側にとっても刺激になり、自分が書いたプログラムが初めて動いたときの初心にかえって新鮮な気持ちになることができると思います。東北の若者と触れ合ってみたい方、Androidのアプリ開発の腕に覚えのあるエンジニアの方はぜひとも講師として参加していただければと思います。ご興味のある方は、info_tohokutechdojo@googlegroups.comまでお知らせください。SD

◆写真3　ある日の石巻道場

写真提供：イトナブ石巻

注2　https://goo.gl/9lqkaw

道場生によるアプリ

これまでにリリースされたアプリを最近のものからいくつか紹介します。

ココイコ

盛岡道場の遠藤さんが開発した、第8期の発表会で多数の賞を総なめにしたアプリです。今いる場所から近くの飲食店を探して一緒に行ってくれる人を募ることができます。住所から周辺の飲食店を検索し、Twitterでご飯の友を募集します。マテリアルデザインを取り入れてアプリの見た目にもこだわっています。

pic gift ～簡単メッセージカード作成～

石巻道場のデザイナー、太田さんが開発したアプリで、写真を“大切な人を喜ばせるためのギフトにする”というコンセプトで、写真を簡単にかわいいメッセージカードに変身させることができます。

場所メモったーよ！

石巻道場の中塩さんが開発したアプリで、ふと立ち寄ったラーメン屋など、次に来ようと思ってもなかなか来られなかったり、道を忘れてしまったりすることのないように、座標(地図)、写真、感想をそのままメモすることができます。共有機能で友人にシェアすることもできます。

橋野鉄鉱山

釜石道場にて作成されたアプリで、岩手県釜石市にある世界遺産・明治日本の産業革命遺産の橋野鉄鉱山を楽しくナビゲートするアプリです。この遺産の歴史、概要などを知ることができます。

温故知新
IT むかしばなし

第50回

記録メディアのバックアップ ～古いデジタルデータが消える前に～

速水 祐（HAYAMI You）　http://zob.club/　Twitter：@yyhayami

はじめに

本連載を執筆するために、筆者は過去のデータを調べることが多くなりました。そのため、1980年代のフロッピーディスクなどを倉庫のロッカーから探し出し、データを読み出そうとしたのですが、メディアに問題があり読み出せないことがあります。10年以上経過したメディアは再生が怪しくなり、とくに記録密度が高いものほどその傾向が強くなります。データを保持するためには、メディアの変換を兼ねたバックアップが必要です。古いデータは、何年か経つと忘れ去られ、廃棄してしまうことがありますが、あとで考えると貴重なデータであることがあります。この機会に古い大事なデータのバックアップにチャンレンジしてみませんか。

▼写真1　カセットテープ（上：赤いケースの60分。下：緑のケースの90分）

今回は、記録メディアのバックアップについてお話します。

カセットテープのバックアップ

カセットテープは、1962年にオランダのフィリップ社が開発した録音／再生用の磁気記録テープの規格です。1979年に登場したウォークマンなどで使われ、1980年代前半のパソコンでも、デジタルデータであるプログラムやデータの保存メディアとしても使用されていました。

筆者は、1970年頃からカセットテープを使用しており、当時のテレビ番組を録音して聞いていました。そのとき録音したデータの中には、今ではほかで聞くことができない貴重なものもあり、そのバックアップに挑戦しました（**写真1**）。

まず、1993年まで使用していたカセットデッキに電源を入れて使用しようとしましたが、セットしたテープが回りません。分解して内部を調べてみると、テープを回すベルトが伸びてしまっていて、空回りしていました。ベルトをはずし、規格を調べて通販でほぼ同じものを見つけ、交換することで正常に回転するようになりました。再生すると、想像していたよりクリアな音が流れ、当時のオーディオ技術のレベルの高さを実感しました。

次に1970年頃のカセットテープを巻き戻したところ、動作が止まらず、ずっと回転し続けてしまいました。調べてみると、テープの先頭部分の茶色の録音テープ部と透明なリード部の接合部分が切れてました。古いカセットテープはこの部分が劣化していて切れやすいようです。ですから古いカセットテープを扱う場合、巻き戻しは慎重に行い、最後の部分は手回しで対応[注1]した方がよいでしょう。それでも切れてしまった場合には、ケースを一度分解して切れた部分を粘着テープで接着し、新しいカセットテープのケースに入れ直して再生すると回転もスムーズになり問題が解決しました。

1980年代前半の、パソコンのプログラム保存用のカセットを再生する場合、当時のデータレコーダをそのまま使うと問題が起こります。30年以上経過したレコーダーはゴムベルトはもちろん、テープを送る役目をする

注1） 昔はカセットテープの穴に、鉛筆を挿し込んで回していました。

記録メディアのバックアップ ～古いデジタルデータが消える前に～

ピンチローラが劣化してベタベタになっていて、そこにテープが巻きついてしまいます。

ピンチローラは、ベルトと同じく黒いゴム製で、これも間違いなく劣化しています。筆者が使用したデータレコーダーを内蔵したシャープの3台のMZシリーズ[注2]は、すべてピンチローラの交換が必要でした。プログラムを記録したテープのデータをMP3の音声データにしてノートPCに保存しておけば、ノートPCの音声出力をレトロPCのカセットインターフェースに接続することで、当時のテープより安定したロードが可能になります。

フロッピーディスクのバックアップ

古い3.5インチのフロッピーディスクから、ファイルが読めないトラブルにあった方は多いと思います。フロッピーディスクは、カセットテープ以上に外気と接触する磁気メディア部の面積が大きく、劣化の頻度が高くなります。また湿度が高い環境ですと、紙製のジャケットが格好の温床となりカビが生えてきます。ですから、古いディスクをフロッピードライブに入れる前に、カビの有無を確認して取り除くことが大事です。

古いフロッピーディスクの読み取りの失敗率は、1990年代の3.5インチHD[注3]ディスクが一番高く、80年代後半の5インチHDディスクと続き、80年代前半の5インチ2DD[注4]ディスクという順番になりました。記録密度が低い容量320KBの2D[注5]ディスクは、保存状態さえよければ案外正確に読み取れます。

FATフォーマット(MS-DOS標準フォーマット)のディスクなら、Windowsマシンで読み取りができますが、DISK BASIC、CP/Mなどのほかの DOSを読み取るには、そのシステムが動くマシン環境が必要です。異なるフォーマットを読み取るソフトウェアツールを利用することで読めるケースもありますが、プロテクト[注6]がかかっているディスクではお手上げです。

1970年代当初のパソコンでは、パラレルポートを経由し、CPUでフロッピードライブシステムを制御していましたが、簡単に確実に利用できるNEC μPD765などのフロッピーディスクコンロトーラ(以下FDC)が使用されるようになりました。しかし、FDCを介してのデータの読み書きは、その機能に依存し、制約から外れることができません。したがって、FDCの異なるドライブシステムで書き込まれた情報を読めないケースも出てきます。

しかし最近、この問題を解決するKryoFlux[注7]というフロッピードライブをUSBで外付けするためのボードが発売されています。KryoFluxは、FDCの役割りをソフトウェアで柔軟に行い、ディスクのフォーマットを設定することでデータをイメージデータとして取り込むことができます。

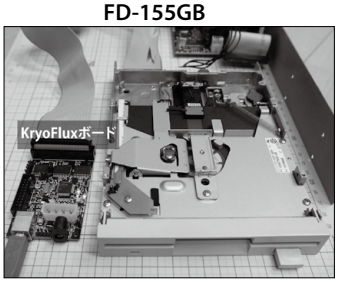
写真2のように外付け5インチフロッピードライブ(TEAC FD-155GF)とボードを接続して、USBケーブルはWindowsマシンとつなげて使います。FD-155GFは、2HD/2DD用のドライブですが両面倍密度2Dのディスクを読むこともできます。

さらにKryoFluxは、FDC制御ができるため「ビットストリームをまるごと記録できる」機能を持っており、フロッピーディスクのバックアップには有用なシステムボードと言えます。

おわりに

20年以上前に作成した、ここでは紹介しなかった映像データ(ビデオテープ、8mmフィルム)を含めて音声データ、アナログ／デジタルデータは、貴重なものが多くあるはずです。昔にどのようなものを作成したかを思い出して、データが消える前に早めのバックアップをお勧めします。SD

▼写真2　KryoFluxボードとFD-155GB



注2) 最近ではTVドラマ「掟上今日子の備忘録」でもMZ-700が出て一部で話題になっていました。

注3) High-density Double-side：両面高密度。

注4) 2-side Double-density Double-Track：両面倍密度倍トラック。

注5) 2-side Double-density：両面倍密度。

注6) ここでは書き込み保護ではなく、簡単にコピーされないための保護のこと。

注7) http://www.kryoflux.com/

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

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

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

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

Service

ガイアックス、「Reactio」と「Mackerel」の連携を発表

㈱ガイアックスは11月18日、同社の障害対応管理ツール「Reactio」と、㈱はてなが提供するサーバ監視サービス「Mackerel」との連携を発表した。

Reactioは、障害発生後の通知と対応内容を記録・管理できるツール。障害発生に気づいたユーザは「インシデント」[注1]を作成し、障害の内容や状況、対応方針などを記載して一斉通知ボタンを押す。すると、登録された関係者へ自動的に電話がかけられ、記載内容が合成音声で読み上げられるというしくみ。そのあとは、インシデント内にて、関係者同士で障害対応の議論ができる。

これまでReactioでは、Mackerelとの連携には独自プログラムの作成や、手動でのインシデント作成が必要だった。今回の連携により、Mackerelが障害を検知するとReactioにて自動でインシデントが作成され、関係者への日本語による一斉自動コールが可能となる。

ReactioとMackerelを連携利用する場合、年内申し込みに限り、6ヵ月間Reactioスタンダードプラン(月額6,000円)を無料提供するキャンペーン「障害対応完璧プラン」(**http://reactio.jp/campaign/mackerel**)が実施されるとのこと。

CONTACT
㈱ガイアックス URL http://www.gaiax.co.jp

注1) 本ツールにおいては、障害対応のためのチャットルームを指す

Software

アドバンスソフトウェア、「ExcelCreator 2016」を発売

アドバンスソフトウェア㈱は11月26日、Excel 2016に対応した高速Excelファイル生成コンポーネント「ExcelCreator 2016」を発売した。

ExcelCreator 2016は、Visual Basic .NET、Visual C#、ASP.NETで作成したプログラムの中から、Excelファイルを生成できるコンポーネント。実行環境にExcelがインストールされている必要はなく、独自の技術でファイルの生成を行うため高い処理速度を持つ。Excelファイルの新規作成や既存のExcelファイルの上書き、別ファイルへの保存ができ、目的のセルに対してデータ設定、書式、計算式、画像、オートシェイプなどの設定、シートのコピーや行、列の挿入／削除などがシンプルなコーディングで実現できる。生成したExcelファイルはPDFやHTMLファイルにも出力できる。

価格は1クライアント開発ライセンスあたり69,120円。本コンポーネントを組み込んだアプリをサーバに配置して使用する場合は別途、xlsx形式用／xls形式用のコンポーネントが必要となる(ともに129,600円)。両方の形式を扱える統合コンポーネントは194,400円(すべて税込)。

CONTACT
アドバンスソフトウェア㈱ URL http://www.adv.co.jp

Topic

マカフィー、2015年の10大セキュリティ事件、認知度ランキングを発表

マカフィー㈱は11月13日、2015年に起こったセキュリティ事件に関する意識調査の結果を発表した。本調査では、過去1年間に発生したおもなセキュリティに関する事件を30件選定、それら事件に対する認知度が測定された。対象者は国内企業の経営者、情報システム担当者、一般従業員など22歳以上の男女1,552人。

●セキュリティ事件に関する意識調査の結果

順位	セキュリティ事件(時期)	認知度
1	日本年金機構への標的型攻撃で125万件の年金個人情報が流出(2015年6月)	60.1%
2	振り込め詐欺／迷惑電話による被害(1年を通して)	56.8%
3	大手金融機関やクレジットカード会社などをかたるフィッシング(1年を通して)	42.1%
4	東アジアの国家元首を題材にした映画公開に際し、米Sony Pictures Entertainmentにサイバー攻撃(2014年11月)	37.0%
5	公衆無線LANのセキュリティ問題(1年を通して)	36.9%
6	Flash Playerの脆弱性(1年を通して)	35.3%
7	全国初のケースとなる、無線LANの「ただ乗り」による電波法違反容疑で男を逮捕(2015年6月)	32.9%
8	ソニー・コンピュータエンタテインメントの「PlayStation Network」にシステム障害(2014年12月)	30.7%
9	IP電話の乗っ取り被害(1年を通して)	28.2%
10	中央官庁の局長が、飲酒で寝過ごした電車内でカバン置き引きの被害に遭い、職員連絡網など流出(2015年6月)	24.9%

CONTACT
マカフィー㈱ URL http://www.mcafee.com/jp

Software

エクセルソフト、「Xamarin 4」を提供開始

エクセルソフト㈱は11月25日、モバイルクロスプラットフォーム開発環境「Xamarin」の最新バージョン「Xamarin 4」の提供を開始した。

Xamarinでは、開発環境「Xamarin Studio」を使って、Mac上でiOS、Android、Mac OS X向けのC#アプリをビルドできる。また、アドインを使用することで、Windows上のVisual Studioでも、シングルソリューションでiOS、Android、Windows、Windows Phone向けのアプリをビルドできる。

Xamarin 4には、UWP（Universal Windows Platform）のプレビューサポートを追加した「Xamarin.Forms 2.0」、iOSアプリ開発用の新しいMacビルドホスト「Xamarin Mac Agent」のほか、AndroidとiOSのUIデザイナの改良など多くの新機能が含まれる。また、本バージョンでは、モバイルアプリのテストツール「Xamarin Test Cloud」およびモバイルアプリの分析・モニタリングツール「Xamarin. Insights」と、IDEとのスムーズな統合を実現している。

●ラインナップ
（価格は年間サブスクリプション）

製品名	価格（税抜）
Xamarin.Android Business	127,800円
Xamarin.Android Enterprise	243,000円
Xamarin.iOS Business	127,800円
Xamarin.iOS Enterprise	243,000円

CONTACT
エクセルソフト㈱ URL https://www.xlsoft.com/jp

Topic

シマンテック、「ノートンサイバーセキュリティインサイトレポート」を発表

㈱シマンテックは11月25日、セキュリティに関する調査「ノートンサイバーセキュリティインサイトレポート」の結果を発表した。

本調査は、17ヵ国計1万7,152人の、モバイルデバイスを持つ18歳以上の成人に対して行われたセキュリティに関する調査。日本の調査結果は、1,009人からの回答に基づく。次のような結果となった[注1]。

- ネット犯罪はこの1年でさらに拡大し、被害者数は5億9,400万人に、被害総額は1,500億USドルとなった
- 国別ではインド、ブラジル、UAE、メキシコ、中国での発生率が高く、中国の被害額が全体の1/3を占める
- 日本におけるネット犯罪の被害者数は786万9,600人で、調査対象国ではもっとも低い
- 日本におけるネット犯罪の被害総額は2,258億円。被害者1人あたりの損失額は2万8,697円
- 日本のネット犯罪の発生率は7%と低いものの、金額別では11位にランクインしており、1人あたりの被害額が大きいことが明らかになった

注1) 被害者数＝オンライン利用成人数×過去1年のネット犯罪被害率
被害金額＝過去1年の被害者数×ネット犯罪の平均被害額

CONTACT
㈱シマンテック URL http://www.symantec.com/ja/jp

Service

ディアイティ、「NIRVANA改」の商用版、「WADJET」を販売開始

㈱ディアイティは11月26日、統合分析プラットフォーム「NIRVANA改」の商用版「WADJET（ウジャト）」製品群を発表、2016年1月から出荷開始する。販売価格は120万円から。

昨今、特定の組織に的を絞ったサイバー攻撃や、ファイアウォールやIPS/IDSといった既存の防御システムが突破されるインシデントが多発している。そのためセキュリティオペレーションの現場では、セキュリティ機器が発行する大量のアラートの処理に追われている。インシデント発生時には物理的な対応も必要となるため、迅速な対応は困難であり、相当な人的コストがかかり続けるといった問題があった。

そんな中販売開始されたWADJETは、NIRVANA改にディアイティが独自に開発した分析機能を搭載したもの。インシデント発生時、ルールに従ってファイアウォールやスイッチなどのネットワーク機器を自動的に制御し、異常通信の遮断を実現する自動防御機能を持つ。

NIRVANA改はNICT（情報通信研究機構）によって開発された、組織内ネットワークを流れる通信のリアルタイムな観測・分析や、各種セキュリティ機器からのアラート集約を実現するサイバー攻撃統合分析プラットフォームである。

CONTACT
㈱ディアイティ URL http://www.dit.co.jp

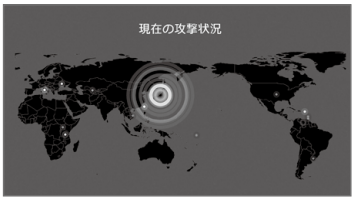
Service

サイバーセキュリティクラウド、「攻撃見えるくん」サービス提供開始

㈱サイバーセキュリティクラウドは11月11日、外部公開サーバへのあらゆる攻撃をリアルタイムに可視化するWebサービス「攻撃見えるくん」の提供を開始した。

同サービスは、自社がどこからどの程度攻撃を受けているかをリアルタイムに地図上やグラフに表示する攻撃可視化サービス。企業に自社への攻撃の存在を「知る」体験をしてもらうために、本サービスは無料で公開される。攻撃ログの表示件数は1,000件までだが、5,000円／月で表示件数を10,000件に上げることもできる。

同社はまた、サーバへのあらゆる攻撃を遮断するIPS+WAFクラウド型サーバセキュリティサービス「攻撃遮断くん」を提供している。本サービスはクラウド（IaaS）を含むほぼすべてのサーバに対応し、ネットワーク、OS、Webアプリケーションへの攻撃を防ぐ。「攻撃見えるくん」も、機能の1つとして含まれている。こちらは、初期費用30,000円、利用料100,000円／月となる。



▲「攻撃見えるくん」管理画面（一部）

CONTACT
㈱サイバーセキュリティクラウド URL http://www.cscloud.co.jp

Service

日本ヒューレット・パッカード、Dockerコンテナ向けに最適化された製品サービスを発表

日本ヒューレット・パッカード㈱は11月27日、Dockerエコシステム向けに構築された各種ソリューションを発表した。おもなラインナップは次のとおり。

- **「HPE Helion Development Platform 2.0」**
  マイクロサービスの展開を実現するためのアプリケーション開発プラットフォーム
- **「HPE StormRunner」、「HPE AppPulse for Docker」**
  Docker化されたアプリのテスト、展開、モニタリングを実施するためのソリューション
- **Remote Docker Swarmクラスタ監視**
  エージェントレス監視アプリケーション「HPE Site scope」を利用し、Docker Swarmクラスタ全体を監視
- **「HPE Codar for Docker」**
  ハイブリッド環境で、負荷に応じて継続的にリソースプールからのデプロイを行う
- **Docker Machine plugin for「HPE Composable Infrastructure」**
  「HPE OneView」をベースに、Dockerコンテナホストの展開を自動化する

CONTACT
日本ヒューレット・パッカード㈱
URL https://www.hpe.com/jp/ja

Topic

DMARC.orgの責任者が来日、送信ドメイン認証技術「DMARC」についての説明会開催

11月18日、「DMARC（Domain-based Message Authentication, Reporting and Conformance）」についてのメディア向け説明会が㈱TwoFive主催で行われ、DMARC.orgのExecutive Directorであるスティーブ・ジョーンズ氏が登壇した。

DMARCは、「SPF[注1]/DKIM[注2]の認証に失敗したメールを受信側がどう扱うべきか」のポリシーを、ドメイン管理者側が宣言するためのしくみ。すでに米国では多くのISPがDMARC対応を進めるなど普及率が高まっている。Twitter社では1日に1億1,000万通あったなりすましメールが1,000通に激減、PayPal社ではクリスマスシーズンに2,500万通もの迷惑メールを遮断するなど、非常に高い効果をあげている。

DMARC.orgは、DMARCの普及および迷惑メール撲滅を目指して、2012年1月にGoogle社やMicrosoft社をはじめとする電子メールサービスおよび技術関連企業など15組織によって立ち上げられた。

注1） Sender Policy Framework：送信元のIPアドレスを用いる送信ドメイン認証技術
注2） Domainkeys Identified Mail：電子署名を用いる送信ドメン認証技術

CONTACT
DMARC.org URL https://dmarc.org

Software

リバーベッド、ネットワーク／プロトコル分析ツール「Wireshark 2.0」を発表

リバーベッドテクノロジー㈱は11月10日、オープンソースのネットワーク／プロトコル分析ツール「Wireshark 2.0」のリリースを発表した。

新バージョンでは多くの機能が追加、改良され、操作性も向上した。新しいアプリケーションフレームワークの採用により、あらゆるプラットフォーム（とくにMac OS XやWindows）で、優れたユーザエクスペリエンスを実現するという。おもな新機能は次のとおり。

・英語、日本語を含む7言語に対応
・SSLディセクタの改良
・I/O、TCPストリーム、スループット、ウィンドウスケーリングの各種グラフの改良
・IPv6の統計情報の収集とアドレス圧縮
・関連するパケットと対話のスパンを最初の列に表示
・キャプチャのグラフや統計を簡単に比較できるウィンドウ整列機能
・ソートなどを高速化するバックグラウンド分析機能

Wiresharkは次のURLから無料でダウンロードできる。

▶**https://www.wireshark.org**

CONTACT
リバーベッドテクノロジー㈱ URL http://jp.riverbed.com

Software

トレンドマイクロ、「Trend Micro Endpoint Sensor」発売

トレンドマイクロ㈱は11月25日、エンドポイント型標的型サイバー攻撃対策製品「Trend Micro Endpoint Sensor」を発売した。

本製品は、各エンドポイントにインストールされるエージェントと、それらをコントロールするマネージャで構成されている。エンドポイントのエージェントは、各エンドポイント内でのレジストリの変更やプロセスの生成、権限昇格など、攻撃手法として利用され得る各種アクティビティを記録する。それら情報をはじめ、ネットワーク監視装置との連携により取得した不審な兆候の情報や、OpenIOC（脅威情報を共有するフォーマット）、YARA（不正プログラムの識別・分類ツール）などの情報を用いて記録したアクティビティを検索することで、関連する攻撃動作の可視化を実現する。この攻撃動作をIT管理者が解析し、エンドポイントにおける脅威がどのように行われていたかを把握できる。

さらに、このエンドポイント内部で知り得たファイル名やハッシュ値、攻撃手法として利用され得る各種アクティビティ情報など、攻撃に関連する情報を再び利用し、ネットワーク内のそのほかのエンドポイントを検索することで、ほかにも隠れた脅威を発見できる。

CONTACT
トレンドマイクロ㈱ URL http://www.trendmicro.co.jp

Software

Red Hat、Red Hat Enterprise Linux 7.2をリリース

Red Hat社は11月19日、「Red Hat Enterprise Linux」（以下、RHEL）の新バージョンであるRHEL 7.2の提供を開始した。おもに、Linuxコンテナ、セキュリティ、ネットワーク、システム管理の機能が強化された。

Linuxコンテナ

Dockerエンジンのアップデートに加え、Dockerコンテナの作成／管理ツール「Kubernetes」、Dockerコンテナ管理用のWebインターフェース「Cockpit」などのパッケージがアップデートされた。また、コンテナ向けの軽量OS「Red Hat Enterprise Linux Atomic Host」も最新の7.2になった。

セキュリティ

SCAP（Security Content Automation Protocol）で定められた形式で記述されたチェック項目に従い、システムを自動で検査できるツール「OpenSCAP」のAnacondaプラグインが追加。インストール中の検査が可能になった。

システム管理

システムアーカイブツール「Relax-and-Recover」が導入された。ISO形式のローカルバックアップの作成や、災害復旧作業の簡素化が可能となる。

CONTACT
レッドハット㈱ URL http://www.redhat.com/ja/global/japan

ひみつのLinux通信

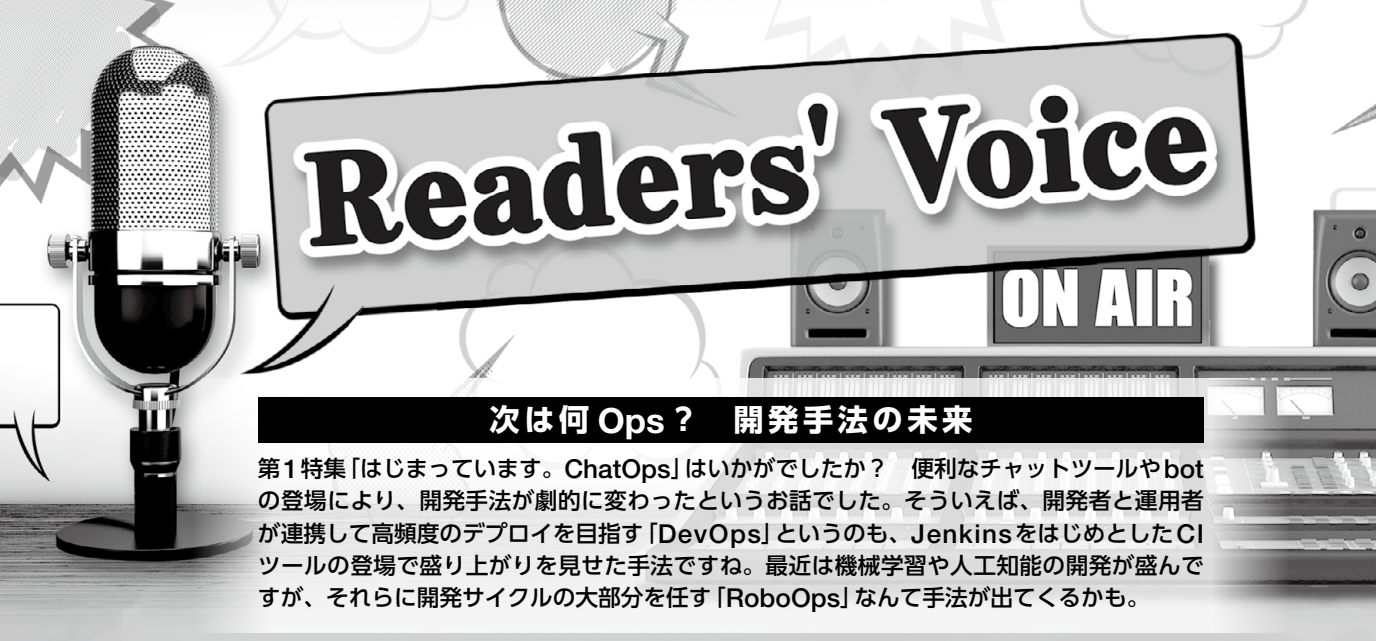
作)くつなりょうすけ　@ryosuke927

年末年始でループする人生も、また愉し……。でも物理年齢はインクリメントされますから！
仮想化は脳内だけにしとくといいよ！（豆蔵より）

恒例年末年始特番

Unix Wizard専門学校[円環の理]編

クリスマスを一人で過ごしていても、みんなそうだから安心しなよ！（44歳独身編集）



次は何Ops？　開発手法の未来

第1特集「はじまっています。ChatOps」はいかがでしたか？　便利なチャットツールやbotの登場により、開発手法が劇的に変わったというお話でした。そういえば、開発者と運用者が連携して高頻度のデプロイを目指す「DevOps」というのも、Jenkinsをはじめとした CIツールの登場で盛り上がりを見せた手法ですね。最近は機械学習や人工知能の開発が盛んですが、それらに開発サイクルの大部分を任す「RoboOps」なんて手法が出てくるかも。

2015年11月号について、たくさんの声が届きました。

第1特集 すいすいわかるHTTP/2

次世代のインターネットプロトコル規格「HTTP/2」の特集でした。HTTP/1.1からの移行の必要性から、HTTP/2のプロトコルの全体像、HTTP/2対応のサーバソフト（nghttp2、h2o、Nginx）による環境の構築、今後のインターネットの展望までを扱いました。

まさしく「そろそろ押さえておかないと」と思っていたのでタイミングがよかったです。1.1の問題点をふまえた形での解説、わかりやすくて理解しやすく感じました。　**romeoshaertさん／長崎県**

HTTP/1.1と下位互換になってくれるか心配です。ホームページ持ってるので。　**Tayuさん／千葉県**

ちょうど勉強したいと考えていたHTTP/2の基本を知ることができ、本格的な学習の入口になった。　**浅井さん／大阪府**

HTTP/2は、以前耳にして以来気になっていたので、勉強になった。保守している自社サービスにもWebアプリケーションがあるため、今後の参考になった。　**ほまれさん／千葉県**

HTTP/1.1との比較が、具体的でわかりやすかったです。　**すけさん。さん／東京都**

HTTP/2は、キーワードと簡単な特徴だけ知っていましたが、今回の記事を参考に実際に試してみることで理解が深まりました。今後必須技術になるでしょうから、非常に参考になりました。　**今井さん／千葉県**

「HTTP/2って何だろう」と思っていたので、買ってみました。　**護さん／新潟県**

HTTP/2について、言葉として聞いたことはあるけれど、具体的なしくみは知らなかったという声が多く寄せられました。インターネットにおいて、本格的に普及するのはまだ先かもしれませんが、知っておくと役立つこと間違いなしです。

第2特集 ファイアウォールの教科書

セキュリティ対策のはじめの1歩、ファイアウォールの特集でした。基礎概念の説明から、iptables・firewalldといった具体的なソフトウェアの解説を行い、最後の章ではWebアプリ専用のファイアウォール「WAF」を紹介しました。

新しいソフトウェアをサーバにインストールしてうまく動かないなと思うと、ファイアウォールが原因だったりして苦手意識がありました。克服できそうです。　**binaさん／東京都**

境界線のところは任せっぱなしだったので興味があった。　**みふさん／神奈川県**

最近はインターネットからの攻撃も多いので、防御は重要ですね。　**KKさん／愛知県**

firewalldは気になっていましたが手つかずだったので、タイミングよく勉強できました。　**山下さん／東京都**

ファイアウォールを設定しているときに、なんでだろうかと感じているのが解決できた。　**ももんがさん／静岡県**

セキュリティ対策は、他人任せ・ソフトウェア任せといった人が多いのではないでしょうか？　特集で紹介したiptablesやfirewalldはOS標準機能ですので、ぜひ手元で動かしてみてください。

11月号のプレゼント当選者は、次の皆さまです

①Parallels Desktop 11 for Mac Pro Edition
けん様（埼玉県）、福司久美子様（埼玉県）

②東芝1号機ものがたりⅡ &12digit premium calculator
匿名希望様（神奈川県）、古沢淳様（神奈川県）

③GitHub Tシャツ
上田裕己様（島根県）

④『入社1年目からの「Web技術」がわかる本』
にぽぽさん様（東京都）、taka様（福岡県）

⑤『初めてのSpark』
北海のクマ様（北海道）、おじさん様（東京都）

⑥『たのしいインフラの歩き方』
田中俊也様（京都府）、永作肇様（東京都）

⑦『データサイエンティスト養成読本　機械学習入門編』
八坂文規様（神奈川県）、山下高範（奈良県）

※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。

短期連載 SMB実装をめぐる冒険【前編】

Windows共有フォルダをChromeOSのファイルアプリにマウントするアプリ。その開発には「SMBプロトコル」の理解とJavaScriptでの実装が必要でした。連載第1回では、SMBプロトコルの情報集めと、そこでやりとりされるメッセージの解析までを追いました。

臨場感溢れていて、とてもおもしろい。次回も気になります……。
くま～～～さん／神奈川県

SMBの情報がそんな状態だったとは、知らなかった。
マシンみな古い／奈良県

なんだかんだ言ってWindowsが混在するので、SMBは大切です。
下平さん／東京都

SMB自体は新しい規格ではないので当然資料などが出回っていると思っていたが、そうでもないことに驚いた。そのような状態から1つずつ手探りで仕様を調べていく展開は手に汗握る感じでとてもおもしろかった。次回記事も期待したい。　tack41さん／愛知県

SMBプロトコルの通信の中身を、日本語マニュアルなしに解析していくのは一苦労。謎が1つずつ解き明かされていくさまは、まさにミステリ小説ですね。

短期連載　OS Xクライアント管理ツール「munki」【後編】

Windowsに比べると、これといった手段がないMacのクライアント管理。そんな中「munki」は"手軽な導入"、"Googleによる開発"で注目を集めるOS Xクライアント管理ツールです。後編では管理対象のクライアント状態、インベントリ情報の取得について説明しました。

こんな便利なツールがあるんですね。
出玉のタマさん／大阪府

munkiは気になっていたので、とても参考になりました。
n0tsさん／東京都

OS Xクライアントを簡単に管理できるmunki、注目度が高いようです。エンジニアさんやデザイナさんにはMacを好んで使う人が多く、需要のある企業も多いのではないでしょうか。

短期連載　Jamesのセキュリティレッスン【6】

新しいファイル形式「pcap-ng」も使えるようになったパケットキャプチャのツール「Wireshark」の使い方を紹介する短期連載です。今回はキャプチャファイルからポートスキャンの結果を推測する方法を、問題を出しながら解説しました。

マイナンバーの件もあり、あらためてセキュリティについて学べるのは良い。
YGさん／神奈川県

ポートスキャンの方法がとても参考になりました。　そうでげすさん／神奈川県

今回はポートスキャンに関する実践的な回でした。Wiresharkはサイバー攻撃を受けたときの調査用ツールとして非常に有用と言えますね。大きなアップデートがあれば、「Jamesのセキュリティレッスン」でまた紹介したいと思います。

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
February 2016

2016年2月号
定価（本体1,220円+税）
192ページ

1月18日発売

［第1特集］2大OSSデータベースの特異点を知れ!

MySQLとPostgreSQL徹底比較
導入時の「罠」を避ける現場ノウハウ

皆さんは、MySQL派ですか? それともPostgreSQL派ですか? 当たり前のように使っているOSS DBですが、案外知らないで使っていることばかり。プロセスタイプとスレッドタイプの違い、基本的なSQL文法の違い、機能や拡張性の違い、などなど特異点（罠）を知り、はまらずにシステム構築するポイントを本特集では解説していきます。OSS DBのメリット・デメリットを押さえて積極的に活用してみませんか?

［第2特集］永久保存版・現場必携

LANケーブリング／サーバラッキングの教科書
サーバラックを組む基本知識から応用技術まで

引き込み線で木版がいる?、床耐加重?、光ファイバーは折れる?、現場でやってみなければわからない配線の技術と知恵

※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

「Red Hat Enterprise Linuxを極める・使いこなすヒント.SPECs」（第18回）、「書いて覚えるSwift入門」（第11回）は都合によりお休みさせていただきます。

SD Staff Room

●かつてないほど高齢化が進んだ当編集部は、いわばTOKIOのようなもの（笑）。特集もダッシュ村を参考にしつつ、ゼロからデータセンターを作ってみるとか、銅線からCAT-5ケーブルを作ってみるとか、全部無料でシステム構築してみるとかとか、そういうのがいいかもしれない。いや、そうしようっと。（本）

●最近、野菜がべらぼうに安い。大根1本80円とか、キャベツ1個90円とか。特売だと白菜が1玉で100円ということもある。1ヵ月くらい前は、この3倍近くしたななどと思いつつ、年末にはまたべらぼうに高くなるんだろうなと推察する。毎年高騰寸前に購入できるかの想定が難しい。去年の統計とかないのかな。（幕）

●息子の同級生がトランペットを始めるということで、私のお古を貸し出すことに。が、大学卒業後ろくに手入れもせずに、うん十年。シルバーだったそれは青銅器みたいに（言い過ぎ）。あわてて重曹で磨いてなんとか銀色に見えるようにしました。こいつも使ってもらえることになって、うれしいに違いない。（キ）

●マイナンバー通知カードが届きました。ペラペラなカードなので、遅かれ早かれ失くしそうです。住基ネットのときも番号が通知されたはずですが、その文書をどこに保管したか見当もつきません。全然使っていないから不都合もないのですが。じゃあ、マイナンバーもそんなに気にしなくていいか。（よし）

●仕事柄、PC画面や紙の原稿とにらめっこする時間が長く、肩と首が猛烈に凝ります。整体に通うのは億劫なので、休憩中にいろいろなストレッチを試しています。お勧めのストレッチは、首を回しながら左右の肩を前後逆向きに回転させるというもの。ビジュアル的に、お外ではできないですね。（な）

●息子の学校の役員で広報誌を担当しています。配置決め前にまず聞かれたのは「広報誌が作れるようなソフトが入ったパソコンご家庭にありますか?」でした。今日びスマホやタブレットで事足りるせいか、手を上げたのはたったの二人。「若者のパソコン離れ」が囁かれていますが、実際はどの世代も、なのでしょうか。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6173

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2016年1月号

発行日
2016年1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。