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
8
2016
2016年8月18日発行
毎月1回18日発行
通巻376号
（発刊310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

》GitHub入門　》YumとAPTでわかるLinux

Special Feature 1

GitHub

はじめての Pull Requestから、チーム導入へ

さいしょの一歩

プルリクしていますか？

Special Feature 2

安全な
パッケージ管理

YumとAPTを通してOSとアプリの運用・管理を考える

案外知らなかった
パッケージの導入とそのしくみ

Extra Feature 1

Ruby on Railsへの
導入でわかった
RRRSpecによる
分散テストの効果

Extra Feature 2

短期集中連載
「乱数を使いこなす」力武健次

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）

8
2016
［目次］
Software Design

contents

Special Feature 1 第1特集

GitHub さいしょの一歩

プルリクしていますか？

8
2016
［目次］
Software Design

Special Feature 2 第2特集

案外知らなかった

YumとAPTのしくみと活用

安全にパッケージ管理していますか? 057

Extra Feature 一般記事

Catch up trend

àla carte アラカルト

contents

Column

Development

OS/Network

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

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

ISBN978-4-7741-8217-9
B5変形判／432ページ
定価（本体3800円+税）

【改訂新版】Spring入門

Javaフレームワーク・より良い設計とアーキテクチャ

長谷川裕一、大野渉、土岐孝平 著

いまやJavaを利用したシステム開発ではミドルウェアとしてSpring Frameworkが定番となりました。バージョン4.0以降、急速に普及しているクラウド環境への対応が当たり前のものになり、さらにはマイクロサービスへの対応もすでに当然の機能になっており、そうしたWebアプリケーションの作り方の枠組みすべてを含んでいる体系がSpringと言えます。本書はSpringの使い方を基礎の基礎から、応用的・実践的な開発まで広く解説します。

ISBN978-4-7741-8238-4
A5判／208ページ
定価（本体1980円+税）

Slack入門

ChatOpsによるチーム開発の効率化

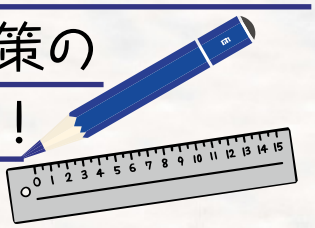
松下雅和、小島泰洋
長瀬敦史、坂本卓巳 著

Slackは親しみやすいUIとチャットコミュニケーションを活性化させる工夫がたくさんあり、いまやエンジニアだけでなく多くのビジネスユーザを獲得しています。
本書では「ChatOps」というソフトウェア開発におけるタスク管理の考え方に触れ、はじめてSlackを利用する方に向けて基本操作をじっくり解説します。さらにボットツール（Hubot）と連携したタスクの自動化やCIツールと連携したアプリの開発方法を紹介しています。

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

あなたを
合格へと導く
一冊があります！

効率よく学習できる
試験対策の
大定番！



金子則彦　著
A5判／688ページ
定価（本体3300円+税）
ISBN978-4-7741-7953-7

岡嶋裕史　著
A5判／424ページ
定価（本体1880円+税）
ISBN978-4-7741-7869-1

岡嶋裕史　著
A5判／624ページ
定価（本体2980円+税）
ISBN978-4-7741-7954-4

エディフィストラーニング株式会社　著
B5判／384ページ
定価（本体2980円+税）
ISBN978-4-7741-7955-1

岡嶋裕史　著
B6判／320ページ
定価（本体1680円+税）
ISBN978-4-7741-6942-2

左門至峰・平田賀一・山内大史・幸田廣信　著
A5判／320ページ
定価（本体2380円+税）
ISBN978-4-7741-8067-0

岡嶋裕史　著
A5判／672ページ
定価（本体2880円+税）
ISBN978-4-7741-7806-6

エディフィストラーニング株式会社　著
B5判／400ページ
定価（本体2980円+税）
ISBN978-4-7741-8240-7

大滝みや子・岡嶋裕史　著
A5判／744ページ
定価（本体2980円+税）
ISBN978-4-7741-7821-9

加藤昭・高見澤秀幸・矢野龍王　著
B5判／456ページ
定価（本体1780円+税）
ISBN978-4-7741-8215-5

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92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。

Eclipse OMR

ランタイムの開発をサポートする「Eclipse OMR」

「Eclipse OMR」（以下、OMR）は、任意のプログラミング言語のランタイムを開発するためのツールキットです。もともとはIBMが開発してオープンソースソフトウェアとして公開したもので、2016年3月よりEclipse Technology Projectのサブプロジェクトになりました。

独自のプログラミング言語を作る場合、ランタイムをどのように用意するかという点が大きな問題となります。理想的な方法は、各言語に最適化された専用のランタイムを開発することですが、それには膨大なコストが必要です。

そこで、より現実的な選択肢として、既存の言語のランタイムを活用するという方法があります。その代表的な候補がJVM（Java仮想マシン）です。現在、JVMはさまざまなプログラミング言語のための汎用的なランタイムとして採用されています。JVMを利用した場合、Javaプログラムとの相互運用性が高いというメリットもある反面、ランタイム自体の設計思想や提供される機能がJava言語と密接に関わっており、汎用性に欠けるという問題があります。

これに対してOMRは、言語開発者に対して新しい選択肢を提供するものです。OMRは独自のランタイムの開発をサポートするツールキットであり、それ自体はランタイムでもなければ、プログラミング言語を開発するためのライブラリでもありません。プログラミング言語の開発者は、OMRを使うことで、自前の言語に特化したオリジナルのランタイムを容易に実装できるようになります。

OMRで提供される機能

OMRでは、ランタイムの実装に必要となる共通的な機能が、特定の言語に依存しない形で実装されています。この共通部分の機能は、IBMが自社で開発・採用していたJava仮想マシン「J9」を元にして開発されたものです。IBMの開発者はJ9からランタイムを構成する共通的な機能をコンポーネントとして抽出・リファクタリングすることで、特定の言語に依存しないランタイムの基盤を作り上げました。これは同様の機能を自前で実装する場合に比べて、実績や性能の面で大きなアドバンテージがあることを意味しています。

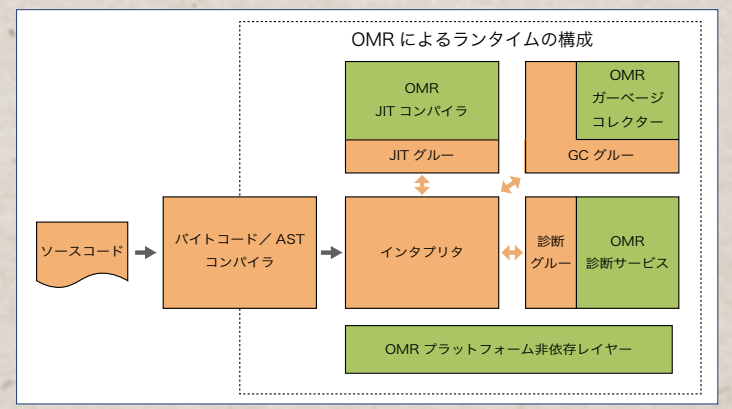
この共通機能と、開発する言語との間でブリッジの役割を果たすのが「ランゲージグルー」と呼ばれるパーツです。ランタイムの開発者はこのランゲージグルーを言語に合わせて独自に実装する必要があります。OMRでは、ランゲージグルーを実装するためのインターフェースも提供されます。

OMRによって提供されるおもな機能としては以下のようなものがあります（開発途中のものも含みます）。

- クロス・プラットフォーム対応のスレッド・ライブラリ
- 異なるプラットフォームに移植するためのポーティング・ライブラリ
- ガーベージコレクター
- IBM Health Centerと連携するトレース・ライブラリ
- 言語非依存のテストフレームワーク
- 診断支援サービス
- JITコンパイラ
- ツール・インターフェース

Eclipse OMRプロジェクトチームでは、技術検証としてOMRを用いたRubyやPython向けのランタイム実装を進めており、成果物がGitHubなどで公開されています。SD

▼図　Eclipse OMRの構成



Eclipse OMR
https://www.eclipse.org/omr

DIGITAL GADGET vol.212

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

IDEAから読み解くガジェット製品

プロダクトデザインの世界的アワード IDEA

SF作家、ウィリアム・ギブスンが「未来はここにある。それはまだ広くいきわたっていないだけだ。」と発言しています。昨今の先進的な家電製品などを見るに、まだ自分が知らなかったり持っていないだけで、すでにSF小説のような、さまざまな未来的機器が存在することに驚くばかりです。

IDEA（International Design Excellence Awards）は、米国工業デザイナー協会による、優れたプロダクトデザインに与えられる賞です。ビジネスや生活の質にかかわる工業デザインの価値をより広く伝えることを目的に1980年に設立された由緒あるデザイン賞です。製品として実際に販売されているものでなければ対象とならないため、夢のようなありえないデザインでなく、先進的でありながらも実現可能と証明されたものが評価されます。学生賞の部門では最近、実際に製品になる前のアイデアや試作品も扱われるようになりましたが、それにしても実現可能性のあるものばかりです。最近では、製品そのものが物体として形があるわけではない、デジタルデザインやコミュニケーションデザイン、イベントのデザイン、店舗やミュージカル、ホテルのデザインに関しても受賞の対象となっています。

※本記事で紹介しているものは
国内未発表・未発売のものを含んでいます。

IDEA 公式サイト
http://www.idsa.org/awards/

受賞作一覧
http://www.idsa.org/awards/idea/gallery

最近の傾向としては、単なる完成品としての製品の素晴らしさだけでなく、その製品が作られてきたストーリー、使うときのストーリー、ひとひねり効いた製品の工夫や細かなこだわりが評価される傾向が強くなってきています。その傾向の1つとして、商品とともに、その商品パッケージのデザインも評価の対象になっています。製品を購入し手に入れた後、初めてパッケージを開封して、製品を使い始める体験が重視されています。実際の製品は、使う前の期待、使い始め、使っている最中、使い終わった後、使っていない期間、さまざまな体験がその製品の評価に影響します。

SF映画に出てくる未来的な製品は、消費者を傷つけたり、ライバル製品としのぎを削ることもなく、ユーザサポートも関係なく、すべてがトラブルなくうまくいく前提で安全性やコストも関係なく、腕が疲れることもなく、火傷することもなく、ただただ「演出」のために理想的な製品が描かれています。実際の製品デザインはそのような都合の良いことばかりではなく、法規制も含め、さまざまな制約の中で最大限新しく便利なものが苦労して考えられています。そのうえ大量生産されて、多くの人に使われるべく世界中に広がっていっているのです。

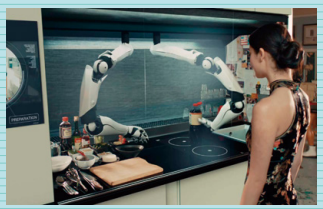
受賞作あれこれ

IDEA過去の受賞作から、いくつか先進的なものを紹介しましょう。

従来デジタル技術とは関係なかった領域のデジタル化

●Edyn Garden Sensor：庭の様子

1969年頃のキッチンコンピュータ（Computer History Musiumの公開資料より）。レシピを記憶させて利用できる形状の美しいコンピュータだが、使い方が難しすぎて1台も売れなかったと言われている。



Moley Roboticsが2017年に発売を予定している、ロボットシェフ。一流のシェフの動きを記録して再現する。

を栽培している植物に合わせて監視するセンサー(**写真1**)

●RYOBI Phone Works:スマートフォンと組み合わせる工事用ツール(連載193回でも紹介。**写真2**)

●Digilock:鍵や番号を覚えておかなくとも解錠が可能な、指紋認証の自転車やバイク用の鍵。コンセプトデザイン(**写真3**)

従来からある製品が技術革新によって新しい使い方ができるようになったもの

●Dolby Conference Phone & BT MeetMe with Dolby Voice:誰がどの方向からしゃべっているのかが把握できる音声会議システム(**写真4**)

●Flux Router:従来は筐体内に隠蔽されていたアンテナや部品を見える形にし、その働きを意識させるルータ機器(**写真5**)

●RE Camera:自撮りに適したハンディタイプのカメラ(**写真6**)

デジタルとは無縁だが、デジタル的考えの製品

●Roll-Di:ロールスクリーンをどちらに引っ張ったらいいのか瞬時にわかる印(**写真7**)

●Silk Road Enroute:血管疾患の手術に利用する機器。血液を電気のようにSTART/STOPし、流量をLOW/HIに切り替えられる(**写真8**)

●Clasp Burner:折り畳んで携帯可能な携帯用ガスバーナー。コンセプトデザイン(**写真9**)

製品そのものは従来型のものであるが、その利用方法にネットやデジタル技術を活用したもの

●Casper:ベッドのマットを自分の好みで試用したあとで購入できるサービス(**写真10**)

●Air Nut:室内と室外の環境を監視し、室内環境を快適に保つ空気清浄機。コンセプトデザイン(**写真11**)

●Monstas:関節疾患を持つ子供のリハビリ用のデジタルオモチャ(**写真12**)

従来からも同等のことができたが、より簡単に便利に使えるようにした製品

●v-alrt:緊急用の連絡ツール。病気や事故などの際、瞬時に緊急連絡するためのペンダント型のボタン(**写真13**)

●MI Router Mini:家庭用の小型Wi-Fiルータ。さまざまなスペースに設置できる(**写真14**)

●Intel WiDock:周辺機器やケーブル類をスッキリ収めることのできるケース。天板が操作パネルになっている(**写真15**)

プロダクトデザインの広がり

プロダクトデザインでは液晶画面の中で完結するデジタルなデザインとは異なり、素材感も重視されます。表面の素材感を担うのは、色、素材、表面仕上げの要素があり、同じ色と同じ素材でも、表面の加工の仕方や仕上げ方によって、手に持ったときの感覚や見た目の印象などが異なってきます。Color、Material、Finishの頭文字を

写真1 Edyn Garden Sensor

写真2 RYOBI Phone Works

写真3 Digilock

写真4 Dolby Conference Phone & BT MeetMe with Dolby Voice

写真6 RE Camera

写真7 Roll-Di

写真8 Silk Road Enroute

写真9 Clasp Burner

写真11 Air Nut

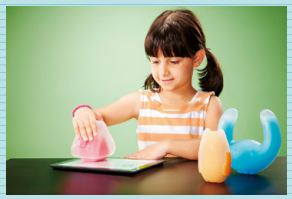


写真12 Monstas

写真13 v-alrt

写真14 MI Router Mini

とってCMFと呼ばれる工業デザインの分野です。

スマートフォンアプリやコンピュータ画面上のデザインの場合、CMFの真似ができるのは色までで、素材もどんなにリアルに映し出されたとしても、液晶画面やディスプレイ表示であることには変わりありません。デジタルデザインの分野でも、動きや色の変化、操作によって変化する事象で、質感的なものをよりリアルに表現しようする傾向が強くなってきており、その分野の表現も少しずつ豊かになってきています。また、振動や微弱な電流で、指で触ったときの質感を模倣しようという技術的アプローチもあります。

それでも、どんなに模倣しようとしてリアルに近づいたとしても、実物にはかなわないわけです。今後はデジタル表現ならではの文脈や振る舞い、印象、感覚の伝達、認識といった、従来の表現とはまた違う表現を開拓していくのが、これからの課題かもしれませんね。SD

写真5
Flux Router

写真10
Casper

写真15
Intel WiDock

Gadget 1

WeMo Insight Switch

http://www.belkin.com/us/Products/home-automation/c/wemo-home-automation/

リモート電源

IDEA受賞作。WeMo Insight Switchは普通の100V電源コンセントを、ネット経由でリモートコントロール可能な電源タップに変えてしまう製品です。このタップに接続することで、ネット対応していない家電製品でも電源のON/OFFが遠隔制御できてしまいます。WeMoを家やオフィスのWi-Fiに接続するとスマートフォン経由でコントロールすることができ、タイマー設定や電気使用量のモニタリングもできます。さらにオプションとしてモーションセンサーが用意されており、人の動きに反応してON/OFFを制御するようにもできます。日本での正式販売は未定。49.99ドル。

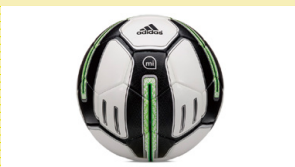
Gadget 2

MICOACH SMART BALL

http://www.adidas.com/us/micoach-smart-ball/G83963.html

スマートボール

IDEA受賞作。MICOACH SMART BALLは回転数・軌道・キックポイントを計測可能なスマートサッカーボールです。心拍数や走行距離データを活用したトレーニングツールとして利用できます。とくにフリーキックの際の癖や改善ポイントがわかるそうです。ボールの中に内蔵されているのは三軸の加速度センサーで、キックの際のスピード、蹴られた位置、ボールの回転、回転方向、ボールが飛ぶ際の軌道などのデータを取得し、スマートフォンの専用アプリと連動してボールの状態を把握することができます。一度の満充電で約1週間、約2,000回のキックで利用できます。



Gadget 3

Moto Hint

https://www.motorolastore.com/hint.html

耳栓型ヘッドセット

IDEA受賞作。Moto Hintは音声操作が可能な、耳栓型Bluetoothヘッドセットです。現在は2世代目の新製品Moto Hint+がリリースされています。赤外線スイッチが搭載されており、耳に装着するとONになり、音声による命令を待ち受ける状態になります。耳に装着した瞬間に、バッテリーの残量、残り使用時間を音声で読み上げて教えてくれます。本体は木製など、さまざまな色と材質を選択してカスタマイズすることができます。待ち受け時間は約100時間、本体だけで連続利用3.3時間、専用の充電ケースと合わせて17時間。149.99ドル。

Gadget 4

Post-it Plus App

http://www.post-it.com/3M/en_US/post-it/ideas/plus-app/

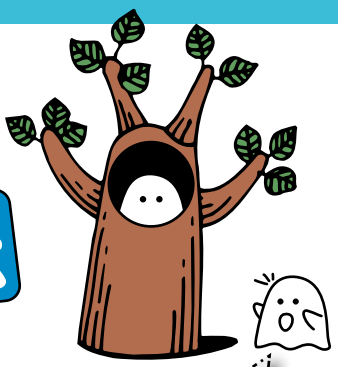
ポスト・イット撮影専用アプリ

IDEA受賞作。Post-it Plus Appは壁などに大量に貼り付けたポスト・イットを撮影することで、それぞれのポスト・イットを取り込み認識し、再利用することのできるポスト・イット撮影用アプリです。アイデアを発散させるためのアナログツールであるポスト・イットと、そこで生まれたアイデアをデジタル化して保存し再利用する、アナログとデジタルの間を橋渡しするツールです。ポスト・イットの色がビビッドな明るい色であることを活用し、うまいこと背景から抜き出して認識してくれます。

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

39

結城浩の再発見の発想法



デファクトスタンダード

デファクトスタンダードとは

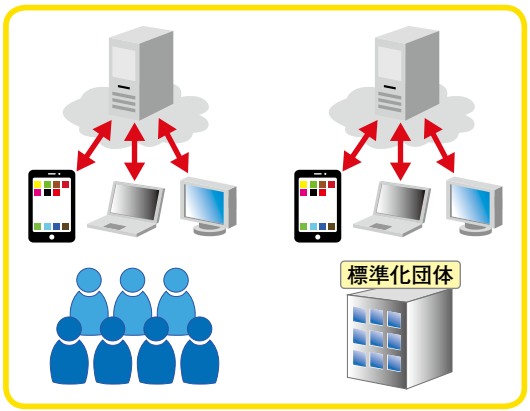
デファクトスタンダード(de facto standard)とは、標準化団体によって定められた標準規格ではなく、多くの人が従っている**「事実上の標準規格」**のことです(**図1**)。「デファクト」は、ラテン語のデ・ファクト「事実に基づいて(de facto)」という言葉から来ています。デファクトスタンダードとして有名なものにはTCP/IPやEthernetがあります。またPDFやTeXなどもデファクトスタンダードと言えるでしょう。

デファクトスタンダードではない、通常の標準規格のことは**デジュールスタンダード**(de jure standard)「法令上の標準規格」と呼ぶそうです。筆者はこの言葉を使った経験はありませんが、対比に便利なので以下でもこう呼ぶことにします。たとえば、電源を機器に供給するためのいわゆる「コンセント」は日本工業規格(JIS)として定められていますので、デジュールスタンダードと言えます。

デファクトスタンダードであれ、デジュールスタンダードであれ、多くの企業が標準規格を必要としています。そもそも標準規格が必要なのは、複数の企業が好き勝手に製品を開発していては不利益が大きいからです。企業が類似した内容を研究しなければならなかったり、せっかく製品を作ったのに会社が異なると互換性が確保できなかったりする不利益です。

そのように考えると「標準化団体によって標準規格を定め、各社はそれに従って製品開発をする」というデジュールスタンダードが唯一の正解のように見えます。しかしながら、実際にはそうではありません。デファクトスタンダードにもメリットがあるからです。

▼図1　デファクトスタンダード(左)とデジュールスタンダード(右)



メリット

デファクトスタンダードのメリットの1つは**スピード**にあります。新しい技術に対応した製品群を生み出そうとするときに、そのための標準化団体を新たに起ち上げ、規格そのものを検討／制定し、それが完了してから実際の製品開発に入るというのでは、たいへん時間がかかってしまいます。それに対して、標準化団体に頼らず、現時点ですでに広まっている規格にみんなが合わせるなら、スピーディに物事が進むでしょう。これはデファクトスタンダードの大きなメリットの1つで、技術的な変化が速いIT業界ではとくに重要な意味を持ちます。

デファクトスタンダードの別のメリットとして**相互運用性の確保**があります。これは通信の分野でとくに大事なことです。たとえ標準化団体が標準規格を定めても、実際にその規格が広まって、多くの会社で採用されなければ意味がありません。1社だけが標準規格に従ったとしても、通信相手の多くが別の規格に従っていては役に立たないのです。それならば、デファクトスタンダードになっている規格に合わせた開発をするほうが理にかないます。

さらに、デファクトスタンダードの大きなメリットは**実用性の担保**です。標準化団体が規格を定めても、それが本当に有効なのかは実際に試さなければわかりません。設計上はうまくいくはずだったのに、現実には役に立たず机上の空論に終わることはままあります。その点、デファクトスタンダードは違います。実際に動いている実績があって、みんなが使っているわけですから、机上の空論になる危険性は低いでしょう。

デメリット

デファクトスタンダードにもデメリットがあります。まず考えられるのが**競合の存在**です。デファクトスタンダードには強制力はありません。「多くの人から使われている規格」がもしも複数個存在したなら、そのあいだで競合が起こり、トラブルに発展することもあるでしょう。

また、逆に**独占的な状況**になる危険性もあります。デファクトスタンダードとなった規格を作り出したのが1社だった場合、改良が滞ったり、多数が望まない方向に規格が変化していく危険性です。業界としてこれを避けるには、デファクトスタンダードをデジュールスタンダード化していく活動が必要でしょう。たとえばAdobeのPDFはデファクトスタンダードとして広く使われていますが、現在はISO規格としてデジュールスタンダードにもなっています。

先ほどメリットのところで、デファクトスタンダードはスピーディに対応できると書きましたが、逆に**進歩が止まる**危険性もあります。そのような状況は、あまりにも多くの人が現在のデファクトスタンダードで満足しているため、大きな変更を好まない場合に起こります。

日常生活とデファクトスタンダード

デファクトスタンダードは、標準化団体による**「お墨付き」に頼らない**標準規格と言えます。デジュールスタンダードに比べてスピード感がありますが、場合によっては1社の思惑でふらふらする危険性があります。

私たちの日常生活でも、デファクトスタンダード的な発想が役に立つことがあります。たとえば開発会社で働くプログラマが、統一した開発手法を社内に浸透させようとするのはよくあることです。それは会社という小さな範囲ではありますが、「標準規格」を作ろうとしているものと見なせます。

デジュールスタンダード的な発想は、開発手法を社内に広めるときに、まず会社から「お墨付き」をもらい、強制力を持ってオフィシャルに展開するというものです。もちろん、これでうまくいけば問題はありませんが、「お墨付き」をもらうためにかかる時間でスピード感が鈍る場合があります。

それに対して、デファクトスタンダード的な発想は「お墨付き」を待つことなく、「事実上の標準規格」として現場にその開発手法を広めてしまうものです。これなら、技術上の変化にもスピード感を持って追従することが期待できますし、現場の実情と乖離（かいり）した結果になる危険性も少ないでしょう。その一方で、強制力がないためにうまく浸透できない危険性もあるでしょう。

あなたの周りを見回して、集団における問題解決の進め方を観察してみましょう。デジュールスタンダードのように、オフィシャルな「お墨付き」を確保してから進める場合が多いでしょうか。それともデファクトスタンダードのように、「事実上行われている解決策」を手がかりにして進める場合が多いでしょうか。

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10回　DragZoom

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

Author　増井 俊之（ますい としゆき）　慶應義塾大学

大規模データの検索と閲覧

世の中にはさまざまな大規模データが存在しますが、きちんとした管理者が存在するデータは階層的な構造で管理されているのが普通です。地名や電話番号やドメイン名などは厳格に階層的に管理されていますし、図書館の蔵書は分類番号を使って階層的に管理されています。個人的なデータであっても、現在のPCのファイルシステムでは、フォルダを使って階層的に管理しなければなりません。

一方、世界最大のデータベースであるWebは階層的に管理されていませんし、手持ちの書籍のような小規模データは手間をかけて階層的に管理するほどでもないので、適当に管理している人がほとんどだと思われます。しかし、管理された大規模階層データは数も種類も圧倒的に多いので、現在も将来もこれらを簡単に検索したり閲覧したりする方法が重要であることは間違いないでしょう。

明示的な階層が存在しないリストのようなものでも、あたかも階層が存在するかのように扱うことができる場合があります。たとえば辞書の場合、「aで始まる単語」「bで始まる単語」のように分類を行い、その下に「aaで始まる単語」「abで始まる単語」のような階層を考えれば階層データと同じように扱うことができます。つまり、ソートが可能なあらゆるデータは階層的なデータとして扱うことができると言えるでしょう。

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

階層データの閲覧

Webのようにリンクで自由に結合された構造のデータと比べると、階層的に管理された情報は対話的に閲覧／検索するのが比較的簡単です。図書館で分類に基づいて本を捜したり辞書で単語を捜したりといった作業は日常的なものですし、計算機で階層メニューやフォルダ階層などを操作することは一般的になっています。

階層的な構造を持つ大規模データを閲覧するためにさまざまなインターフェース手法が利用されています。小さな計算機画面に大量データを全部表示することは不可能ですから、なんらかの方法でデータの一部だけを表示する工夫が必要です。対話的に表示部分を変更しながら大規模データを閲覧するために次のような方法が広く使われています。

- 少しずつ順番に見る（e.g. スクロール、辞書や書籍をパラパラめくる）
- キーワードなどでフィルタリングして表示量を減らす
- 階層的に選択を繰り返してだんだん細部を表示する（e.g. ファインダー、階層メニュー）
- 注目点を部分的に拡大する
- なめらかに直線的にズーミングする（e.g. GoogleMaps）

フィルタリングにより絞り込んだデータをスクロールして閲覧するなど、これらを組み合わせた手法もよく用いられています。

ZUI

階層メニューを使ったりフォルダ選択を繰り返したりして大規模データを絞りこむ方法は、データの一部分の選択を繰り返すことによる非線型なズーミングの一種といえます。これに対し、CGでよく使われる「Perlin Noise」で有名なKen Perlinは、2次元画面上に表現したデータ全体をなめらかに直線的に拡大したり縮小したりすることによって大規模データの閲覧をできるようにする「Pad」という**ZUI**(Zooming User Interface)システムを1993年に提案し、研究者の間でかなり話題になりました(**図1**)。Padのような単純な方法を使えば拡大/縮小操作と移動操作だけであらゆる大規模な階層データを楽に閲覧できるわけですから、CLI、GUIのつぎに来るのがZUIだと期待され、さまざまなシステムや製品プロトタイプが作成されました。

ところが結果的にPadやその後継システムは世の中で流行ることがなく、ZUIの試みはほとんど忘れ去られてしまいました。流行らなかった理由はいろいろあるのでしょうが、2次元画面を自由に拡大縮小して目的の情報を得ることは普通のユーザには難し過ぎたということが大きな理由の1つでしょう。また当時はマウスホイールはまったく普及しておらず、ズーミングのための標準的な操作が存在しなかったことも関係しているかもしれません。

現在はGoogle Mapsなどでズーミング操作はお馴染みになっています。地図の場合は操作対象が具体的で誰にでも比較的わかりやすいためズーミング操作が問題なく利用されているようですが、地図以外への応用は進んでいませんし、スクロール操作とズーミング操作が衝突したり操作を間違えることはよくあります。2次元画面のズーミングを利用したインターフェースが今後流行する可能性は低そうです。

DragZoom

Padのような純粋なZUIは残念ながら普及に成功しませんでしたが、スマホのような小さな画面で大規模データを閲覧/検索するためにズーミングとフィルタリングが有効なことは間違いありませんから、これらを組み合わせて誰でも使える簡単なインターフェースを工夫すればズーミングインターフェースが日の目を見ることがあると思います。

私はスクロールバーを拡張してズーミングやフィルタリングを可能にした「LensBar」というシステムを長年提案しているのですが、これを簡単にした「DragZoom」というシステムを紹介します。

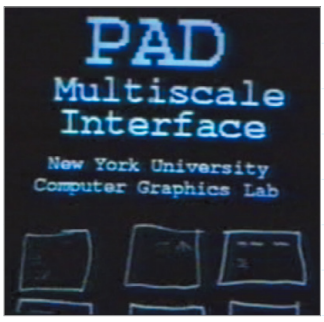
階層データのズーミング

図2は、フィルタリングとズーミング操作によってファイルを検索しようとしているところです。最初はUNIXのファイル構造のうちrootに近い部分だけが表示されています。灰色の横線は、エントリが隠れていることを示しています。

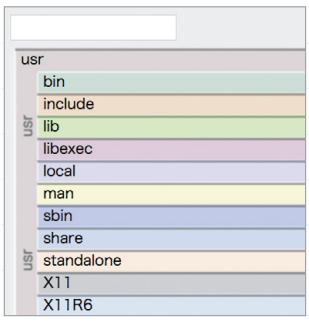
libexecをクリックして右にドラッグするか、スマホなどの場合は指でタッチしてから右にドラッグすると**図3**のようにリストがズーミングされます。

さらに右にドラッグを行うと、**図4**のようにすべてのファイルが

▼ **図1** ZUI(Zooming User Interface)システム



▼ **図2** DragZoom(灰色部分にはエントリが隠れている)



見えるようになります。

階層構造の要素を選択して拡大してブラウジングする方法はTreeViewや階層メニューなどでもお馴染みですが、DragZoomでは左右ドラッグでなめらかにズーミングのレベルを調整していることになります。

複数の指を利用できるスマホやタブレットではピンチ操作でもズームレベルを変えることができます。

ドラッギングによるスクロール

上下にマウスドラッグを行うと、スマホのスクロール操作と同じように画面をスクロールできます(**図5**)。

スクロール操作とズーミング操作は完全に可逆的なので、スクロールしたりズーミングしたりしたあとでカーソルや指をもとの位置まで戻すと画面は最初の状態に戻ります。

フィルタリングとズーミングの組み合わせ

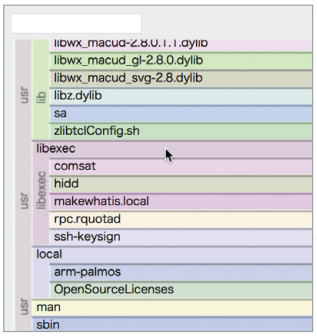
テキスト入力枠に「ruby」と入力すると、**図6**のように「ruby」を含むファイルやディレクトリだけが表示されます。ファイル名の左側にはフォルダ名も縦に表示されているので、どのフォルダのファイル名がマッチしたのかがすぐわかります。

この状態では「ruby」を含まないファイル名はまったく表示されていませんが、「/usr/bin/ruby」の部分をズーミングすると**図7**のように「usr」「include」なども見えるようになります。DragZoomでは、リストの各エントリに「重要度」を設定しており、マウスや指を移動したときは閾値(いきち)を変化させて閾値より大きな重要度を持つ行を表示するようになっています。「usr」や「include」のように階層のrootに近いファイルには大きな重要度を与えているため、キーワードにマッチされていなくても表示されるというわけです。

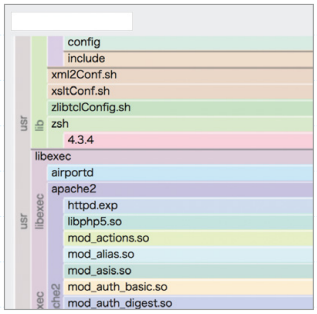
階層がない場合

辞書データのように階層構造が存在しないデータでもズーミング検索できるようにするため、DragZoomでは仮想的な階層を利用しています。たとえばa,b,c,d,e,f,gというフラットなデータがあるときは、それぞれに対して1,2,1,3,1,2,1のような重み(重要度)を与えておきます、ユーザが操作する閾値を超えたも

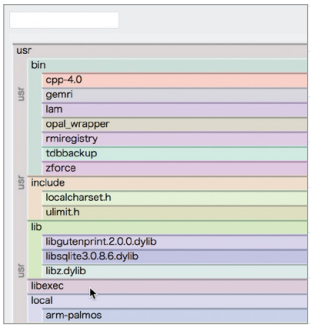
▼ 図3 リストがズーミングされる



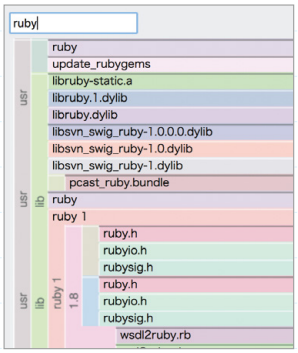
▼ 図4 すべてのファイルが見られるようになる



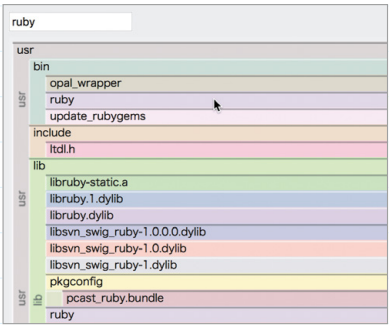
▼ 図5 マウスドラッグでスクロール表示できる



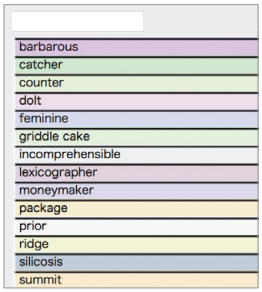
▼ 図6 「ruby」を検索



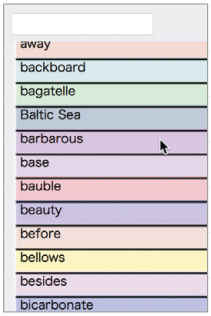
▼ 図7 ズーミングしながらファイルを探す



▼図8　フラットな英語辞書をブラウズしてみる



▼図9　ドラッグ操作



のだけ表示されますから、ユーザのズーミング操作で閾値を変えることによってd,b/d/f,a/b/c/d/e/fのように表示が変化することになります。

DragZoomでフラットな英語辞書をブラウズすると初期画面は**図8**のようになります。

ここでドラッグ操作でズームレベルを変化させると表示は**図9**のようになります。

さらにズームすると単語の意味が表示されます(**図10**)。

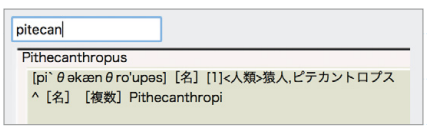
キーワードマッチングにあいまい検索機能を利用すると、多少間違ったキーワードを指定しても最も近いエントリが表示されます(**図11**)。

音楽ファイルの検索

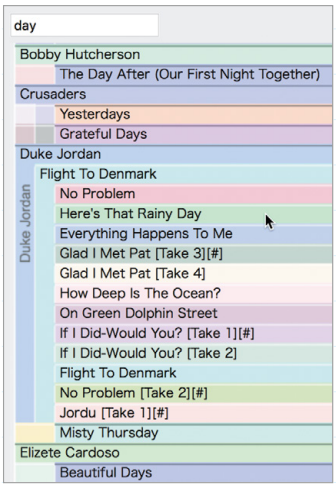
図12は私の手持ちの音楽ファイルで「day」を検索してみたところです。このように、ファイルでも辞書でも音楽ファイルでもDragZoomで簡単に検索できることがわかります。

▼図10　単語の意味が表示される

▼図11　あいまい検索



▼図12　「day」で検索



ズーミングシステムの課題

スクロールバーを使わずズーミングとフィルタリングだけで簡単に大規模データの閲覧や検索ができることはたいへんメリットがあるはずです。多くの大規模データは階層的に構造化されていますからDragZoomは広範囲なデータの検索に利用できるのですが、残念ながら私はまだこの手法を普及させることに成功していません。

有用性は十分なのに普及がうまくいかないのは、左右ドラッグでズーミングを行うというインターフェースに慣れるのが難しいことと、従来の方法でもとくにひどく困ることがないという理由によると思われます。DragZoomを使わなくても普通のテキスト検索や階層メニューなどで階層データをとりあえず眺めることができるのであれば、わざわざ新しい手法に乗り換えようとする人は少ないでしょう。

しかしDragZoomの方法は慣れればかなり便利なのはたしかです。いろいろなサービスで地味に使えるようにしたうえで、今後これをはじめとするさまざまなズーミングインターフェースが普及してほしいものだと願っています。

DragZoomのデモおよびソースコードは、http://DragZoom.com/およびhttp://GitHub.com/masui/DragZoomで公開しているのでご利用ください。SD

宮原徹の

オープンソース放浪記

第6回 OSC名古屋とイベント運営のキモ

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

宮原、酒やめるってよ

突然ですが、断酒を始めました。月1～2回開催されるオープンソースカンファレンス(OSC)に行くと、前日、当日、翌日と飲み食いして、気がつくと体重が！ そんな矢先、偶然Facebookで4年前の自分の写真が出てきて、「病気か？」というぐらい痩せているのを見て、一念発起し、体重が75kgになるまで酒断ちをすることにしました。

本稿執筆時点で断酒後2週間が経過し、83.6kgあった体重が81.6kgと2kg減しています。自分は基礎代謝が高いので、カロリーなどの摂取量を減らせば体重は自然と減っていきます。計算ではだいたい3ヵ月程度で75kgになる計算です。ただ、計算どおりにはいかないと思われるので、バッファとしてプラス1ヵ月程度で4ヵ月と見込んでおり、達成できるのは10月ぐらいになりそうです。10月1日(土)に酒所である新潟・長岡でのOSC開催があるので、このタイミングで美味しい日本酒が飲めるよう頑張ります(**写真1**)。

会場変更はいつもたいへん

さて、5月28日(土)にOSC名古屋が開催されました。OSC名古屋は昨年まで名古屋駅に近い名古屋国際センターにて開催してきましたが、会場が手狭になったため今回から吹上ホールという新しい会場での開催です(**写真2**)。

名古屋地域では、多くの地域コミュニティが日々活動しています。そのため、OSCではデモ展示のスペースがたくさん必要となります。今回は従来の1.5倍の展示スペースを用意できたので、ゆったりとした間取りにできて、出展者、来場者からも好評でした。本当は、各開催でゆったりした展示スペースを用意したいのですが、広い会場がなかったり、使用料が高額だったりするため、いつも展示ブースをぎゅうぎゅうに詰め込んでいる感じになっています。一方で、1つのブース(幅150cm程度)を2つのコミュニティで半分ずつ分け合ってもらうことは「袖すり合うも多生の縁」ではありませんが、出展者同士のつながりを強めるようで、「それはそれでいいのかな」とも思っています。

普通のイベントですと、セミナー中心でデモ展示がなかったり、あってもセミナーの添え物程度ということが多いですが、デモ展示あってのOSC。今後も頑張って展示スペースを確保していこうと思います。

セミナーを最大活用するには

OSCの企画はデモ展示を重要視していますが、もちろん最新情報が聴けるセミナーも重要です。では、セミナーを最大限活用するにはどうすればよいでしょうか？

セミナーで最も恩恵を受けるのは、実は発表者です。新ネタであれば、試行錯誤した内容をアウトプットとして発表資料にまとめられますし、持ちネタであればプレゼンをブラッシュアップできます。OSCでの発表はβテストと考えると、来たるべき本番のために少し冒険するよい機会でもあります。

東京などの大規模なOSCでは、

▼写真1 こちらは断酒前の、OSC名古屋打ち上げの様子。懇親会終了後、地元の地ビール屋さんに集合してお疲れ様会

▼写真2 スポンサーLTの様子。企業の人がコスプレのままプレゼンしても、誰も何も思わないようです(ちなみに艦これの何かだそうです)

スポンサー企業限定のLT大会を開催しています。45分間のセミナーほどのネタではない、あるいは出展のみでセミナー枠がないスポンサー企業のみなさんに活用していただいています。

一方、聴講者の立場でセミナーを活用するポイントは「共感度」にかかっています。もし現在進行形で取り組んでいる技術に関する内容なら、「あるある！」や「お！そんなことが？」といった共感を感じながら聴講できます。逆に、なんとなく興味がある程度だと、共感が得られず、実際に試してみようと思えず、結局そのまま……ということが往々にしてあります。共感度は人それぞれですから、共感度が高くないセミナーを無理に聴講せず、デモ展示をじっくりとみて回ることをお勧めします。きっと、何か新しい発見があるはずです。

立食形式は炭水化物重視で

OSC名古屋の懇親会は、会場の1Fにあるレストランで立食形式で開催されました（**写真3**）。

立食形式の懇親会では、毎回たいへんなのが食べ物の量です。時間帯も6時半から7時、しかもセミナーを聴いて脳が糖質を大量消費したあとですので、摂食中枢が刺激されまくりです。自然とお皿に取り過ぎてしまって食べ物が瞬殺、というのはIT系の懇親会でよく見る光景です。

そこでOSCの懇親会で工夫しているのが、炭水化物を多めに用意することです。種類としてはおにぎりや太巻き、パスタ、焼きそば、そしてみんな大好きカレーなどを用意して、かつ先にそれらを食べてもらうようにしています。

普通にケータリングを頼むと、上品にサラダやオードブルなど見栄えのよいメニューが提案されてきますが、種類少なめ、炭水化物多め、さらにたとえば唐揚げのように1個1個独立しているものにしてもらうと、テーブルごとに取ってきたものをシェアしやすくなり、全体の満足度が向上します。できるだけ食べ残しが少なくなるようにするのがポイントです。また、事前にクッキーなど（**写真4**）のおやつを配布して食べておいてもらうだけでもずいぶんと空腹感が変わってきます。懇親会を企画する人はぜひ参考にしてください。SD

▼写真3
懇親会も後半。食べるものはなくなっちゃったけど、カレーのおかげで満足感は高かったらしく、たくさんの人が話に花を咲かせていました

▼写真4
OSCといえばカントリーマアム。大阪の高校生がわざわざ神戸プリン味を差し入れしてくれました。スタッフで美味しくいただきました

Report

B級グルメを楽しむ ～名古屋編～

地元の名物を楽しむのも全国各地を廻る楽しみですが、B級グルメも含まれます。名古屋のB級グルメで食べたことがなかった「台湾ラーメン」を食べに行きました。

・台湾ラーメンの味仙

OSC会場から歩いて15分、台湾ラーメンの元祖「味仙」今池本店に。まずは普通に中華料理を楽しみ、シメに台湾ラーメンをいただきます。味は普通のものと、なぜか「アメリカン」という味を薄くしたものの2種類。けっこう辛いと事前に脅されていたのですが、筆者は辛いのは比較的大丈夫な方なので、アメリカンでは物足りないぐらいでした。でも、さすがに唐辛子たっぷりなので、あとからあとから汗が出てきます。

▲味仙 今池本店まで歩いて来ました。外観はちょっと大きめの中華料理屋さんという感じで、台湾ラーメンの元祖という感じはしません

▲こちら台湾ラーメン。たしかにスープの色が見ただけで汗が出てきそうな赤い色をしていますね

ツボイのなんでもネットにつなげちまえ道場

第14回

BBC micro:bit

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

読者のみなさんは、BBC micro:bit[注1]というマイコンボードをご存じでしょうか(**写真1**)。昨年 micro:bit は、BBC(British Broadcasting Corporation、英国放送協会)が、イギリスの子供たちがプログラミングやデジタルテクノロジに興味を持ってくれるよう、7学年(11歳)の子供たちに100万台配布するという計画とともに発表されました。BBCといえば、1980年代にBBC Microというコンピュータをリリースしていたことがあります。micro:bitは、昨年の夏～秋から配布され、ついに今年の7月くらいから一般販売が開始されそうです。

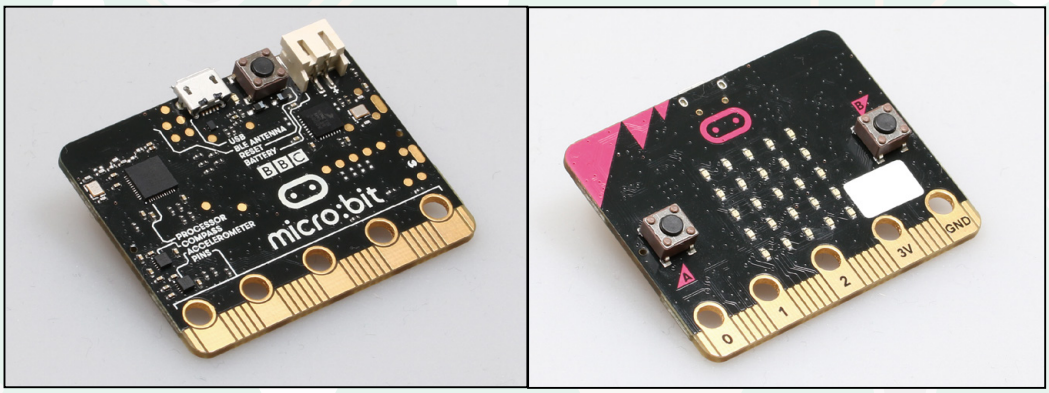
このBBC Microを開発したAcorn Computersという会社は、Acorn RISC Machineというプロジェクトでプロセッサを開発し、それはARM1と呼ばれる開発サンプルでした。最初の製品はARM2で、その後ARMアーキテクチャはmbedに搭載されているCortex-Mへとつながっています。

micro:bitには、Cortex-Mアーキテクチャのnrf51822というマイコンが搭載されています。このチップは、Bluetooth Low Energy(BLE)の無線機能も搭載していて、16MHzのCPUクロックで動作し、16KBのRAMと、256KBのFlashメモリを搭載しています。また、特徴的な5×5ドットのLEDマトリックスのほか、加速度センサと地磁気センサも搭載されています。基板の端のカードエッジコネクタには、マイコンからの信号線がつながっていますので、拡張性もあります。

このカードエッジコネクタを使うには、鰐口(わにぐち)クリップなどで接続するという方法(**写真2**)のほかに、カードエッジコネクタ用のコネクタ(**写真3**)を使って基板に接続するという方法もあります。鰐口クリップではうまく接続できずに、ほかの端子とショートしてしまう危険もありますので、このようなコネクタを使ったほうがよい

注1） https://www.microbit.co.uk

▼写真1　micro:bit(左：表面、右：裏面)



でしょう。

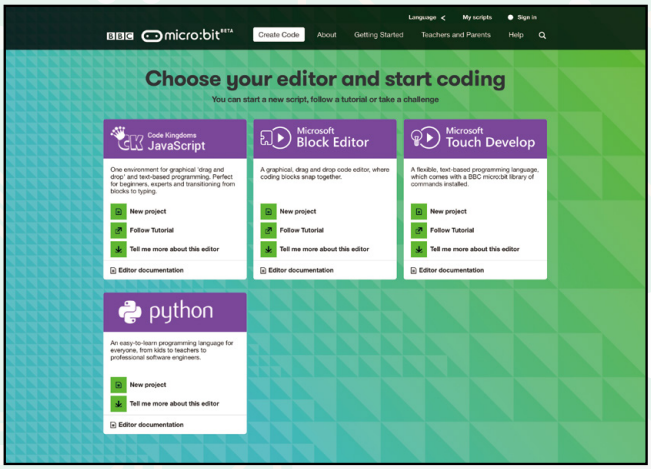
micro:bitは、mbedのプラットフォームとしても登録されています[注2]。micro:bitは、これまで紹介してきたmbedと同様に、mbedのオンラインコンパイラを使い、「コンパイル→ダウンロード→ドラッグ＆ドロップで書き込み」という手順で開発することができます。

しかし、今回はmbedの話でなく、micro:bitがどうおもしろいのかを説明していきたいと思います。

開発言語、環境

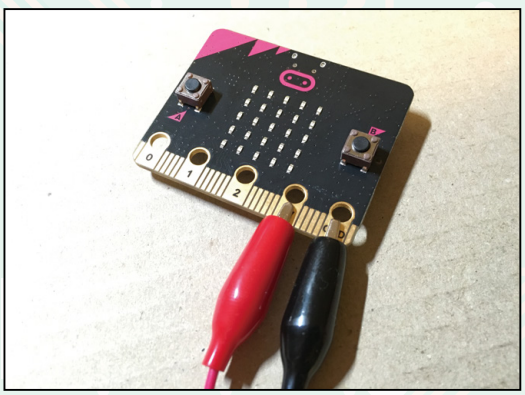
micro:bitはmbedですので、C++で開発できます。しかし、子供たちにいきなりC++でプログラミングを教えるのには無理がありますよね。実際、micro:bitには、ブロックをつなげてプログラミングを行うBlock Editor、Microsoft Researchが提供するタッチデバイス向け開発環境のTouch Develop、Code KindomsのJavaScript、Python、とさまざまな選択肢が提供されています[注3]（図1）。筆者もさっそく試しにBlock Editorを使ってみたのですが、これがなかなかよくできています。

▼図1　micro:bitのサイトによる開発環境の紹介

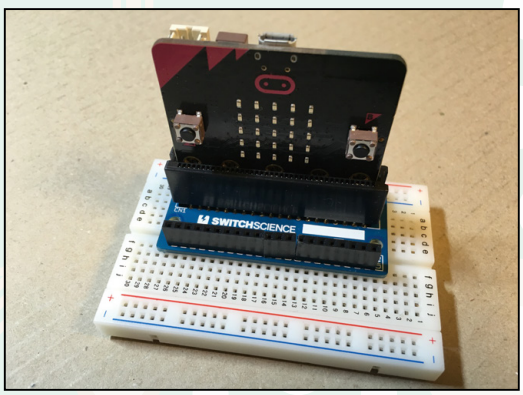


▼図2　BlockEditor

▼写真2　鰐口クリップを使ってみたところ



▼写真3　専用のピッチ変換基板



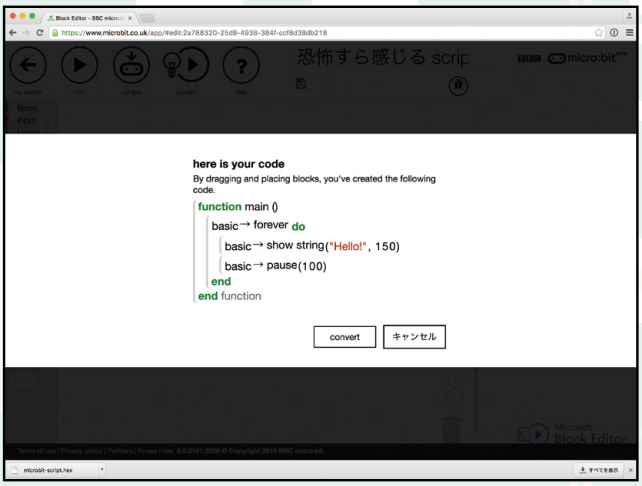
注2）https://developer.mbed.org/platforms/Microbit/

注3）https://www.microbit.co.uk/create-code

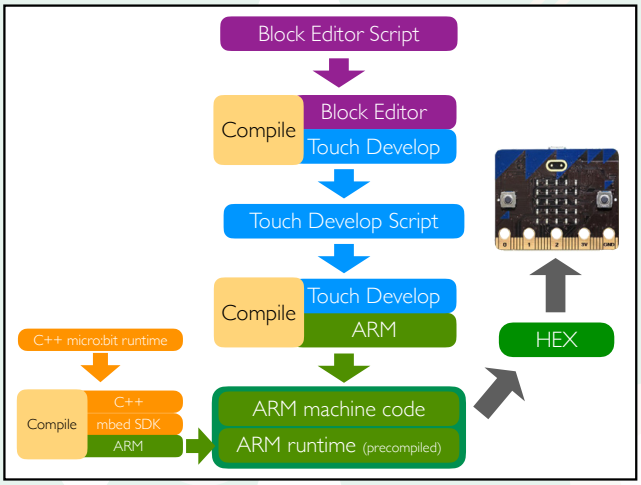
Block Editor

Block Editorは、ブロックを組み合わせて手軽にプログラミングを楽しめる開発環境です。mbedと同じようにWebブラウザで開発環境にアクセスします(図2)。ブロックを組み合わせて、「run」ボタンをクリックすると、画面中のシミュレーターでシミュレーションが始まります。ここで、「compile」ボタンをクリックすると、micro:bitに書き込むHEXファイル(バイナリを16進数の文字列としてテキストファイルにしたもの)がダウンロードされます。

▼図3　スクリプトへの変換



▼図4　Block Editorのしくみ



※出典：https://www.touchdevelop.com/microbit

Block Editorの「convert」ボタンをクリックすると、Touch Developのスクリプトに変換されます(図3)。Block Editorは、このTouch Developのフロントエンドのような位置づけのようです。このBlock Editorは、GoogleのBlocklyというオープンソースのWebベースなビジュアルプログラミング環境をベースに作られています。Blocklyは、GitHubで公開されています[注4]。

Block EditorやTouch Developのすごいところは、オンライン開発環境なのですが、オフラインでコンパイルができる点にあります。たとえばmbedの場合、Cortex-Mのバイナリをソースコードからコンパイルするとき、クラウドにあるサーバの上でC++コンパイラが走っています。Touch Developでは、一度Webブラウザがスクリプトを読み込んでいれば、サーバと通信することなく、ブラウザの上でコンパイラが走ります。もちろん、ビジュアルプログラミング環境やスクリプトで書かれた単純なコードなので、ブラウザで走るスクリプトだけでバイナリを作ることができるのでしょうが、コンパイラがスクリプトで書かれているというのは驚くべきことです。

筆者もワークショップなどでマイコンの開発方法を説明することがあるのですが、インターネット接続が前提だとトラブルが生じたりすることがあります。逆にオフライン開発環境では、IDE(統合開発環境)をインストールしてもらうときにトラブルが発生することもあります。micro:bitの開発環境のしくみは、もしかすると、こういった問題を回避できる、バランスのよいしくみかもしれません。

「ブラウザで走るスクリプト」と記したことからみなさんご想像のとおり、Touch DevelopはJavaScriptで書か

注4) https://github.com/google/blockly

れています。正確には、TypeScriptというMicrosoftによって開発されたオープンソースのプログラミング言語で、JavaScriptのスーパーセットです。TypeScriptは、コンパイルするとJavaScriptのソースコードを出力します。

話が逸れました。当初、Touch Developのスクリプトは、バイトコードにコンパイルされる仕様だったそうです。バイトコードというのは、特定のハードウェアでなく、仮想マシンで実行可能なバイナリです。Javaのバイトコードが有名どころでしょう。このころはコンパイル操作をすると、生成したバイトコードに、コンパイル済みのバイトコードインタプリタを結合して、ダウンロード用のHEXファイルを生成していました。

しかし、やはりバイトコード、スタック(メモリ)の利用を最適化していなかったために、メモリを多く消費してしまっていたそうです。このため、Microsoftのエンジニアは、バイトコードを生成するのではなく、micro:bitに搭載されているマイコンのコア、ARM Cortex-M0で直接実行可能なARM Thumb命令セットを生成するように作り替えたのです(**図4**)。こうして、JavaScriptによるThumb命令セットのコンパイラは誕生しました。

もうひとつ、Block Editorにはすごい機能があります。コンパイルして生成されたHEXファイルをインポートする機能です。Block Editorで「my scripts」ボタンをクリックすると、自分が作ったスクリプトの一覧のページに遷移できます。そこに「Import Code」というボタンがあります。これを使うと、HEXファイルを読み込んで、そのHEXファイルを作ったときのブロックを再現できます。リバースエンジニアリングの機能もついていると知ったときには、びっくりしました。これをどうやって実現しているかは、まだ調査できていません。

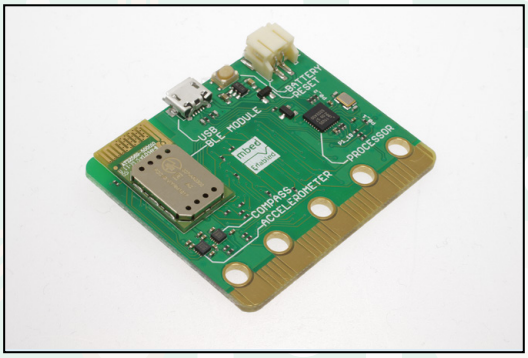
コンパイラが出力したファイルを元に戻せるしくみというのは、とても合理的だと思います。単一のファイルをやりとりするだけで、プログラムを見せ合うことができるのは、お互いに刺激し合って成長をする機会を提供できるように思えます。

技適

ここまで紹介してきたmicro:bitですが、冒頭で紹介したようにイギリスでの使用が前提になっています。このため、ヨーロッパの適合性評価は受けています。しかし、日本の電波法はもちろん、アメリカのFCC Part 15などのほかの国々での認証は受けていません。micro:bitは、ドラッグ&ドロップによるプログラムの書き込みのほか、OTA(Over-The-Air)、つまり無線通信を利用した書き込みにも対応しています。ですので、ユーザが書いたプログラムで無線機能を使わなくても、無線機能が有効になっています。

これだけおもしろいmicro:bitを、日本で適法に使うことができないのは残念です。とくに、子供にプログラミングの楽しさを経験してもらうのに、法を破らせるというのは教育的にもよくないように思います。そんなわけで、micro:bitに搭載されているマイコンの、技適マークつきモジュールを使用して、日本で適法に使えるmicro:bit互換機を作ってみました(**写真4**)。筆者にはスポンサーがいないので、micro:bitよりも高くなってしまいましたが、これで適法に遊ぶことができます。SD

▼写真4　筆者らが作ったmicro:bit互換機の試作品



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8月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

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

01

テレビ用高音質スピーカー Olasonic「TW-D770PT」 1名

普段使いのテレビ用に特化した音作りが特徴のスピーカーで、ニュースやドラマを観るときなど、人の声がクリアに聞こえるようになります。操作ボタンが付いた本体をテレビと光ケーブルでつなぎ、本体からミニケーブルでスピーカーをつなぎます。テレビのほかに、ラジオ、ICレコーダー、スマートフォン、携帯ゲーム機など、イヤホンジャック搭載の機器と接続できます。

提供元 **東和電子** http://www.olasonic.jp

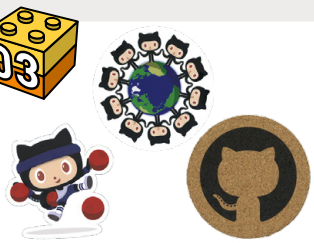
02

Type-C USBハブ「USH-C02」 3名

USB Type-Cコネクタから本体まで約10cmのケーブルでつないだ「ケーブル一体型」タイプのUSBハブで、複数の機器を接続した場合でも取り回しが快適です。USB2.0ポートが3つ、USB3.0ポートが1つの、計4ポートが搭載されています。

提供元 **ミヨシ** http://www.mco.co.jp

03



GitHub Tシャツ&ステッカ&コースター 1名

第1特集でも取り上げたGitのリポジトリサービス「GitHub」のノベルティセットです。Tシャツのサイズは「L」です。

提供元 **ギットハブ・ジャパン** http://github.co.jp

04

DevOps教科書

レン・バス ほか 著／長尾 高弘 訳

システムの開発と運用を一体化するDevOpsの実態とそのあり方を、ソフト・アーキテクチャの面から詳述した解説書です。マイクロサービスアーキテクチャ、AWSのケーススタディなどを扱います。

提供元 **日経BP** http://www.nikkeibp.co.jp **2名**

05

入社1年目からの「ネットインフラ」がわかる本

村上 建夫 著

機能的なシステム構築に欠かせないインフラとしてのネットワークについて、幅広く実用的な情報を提供しています。初学者向けに通信技術の基礎から解説し、セキュリティ対策までカバーしています。

提供元 **翔泳社** http://www.shoeisha.co.jp **2名**

06

機械学習と深層学習

小高 知宏 著

機械学習の諸分野をわかりやすく解説し、それらの知識を前提として深層学習とは何かを示す1冊。具体的な処理手続きやプログラム例（C言語）が適宜示されており、具体的な理解を助けてくれます。

提供元 **オーム社** http://www.ohmsha.co.jp **2名**

07

[改訂新版]Spring入門

長谷川 裕一、大野 渉、土岐 孝平 著

Javaを利用したシステム開発の定番になった「Spring Framework」の入門書。Springの使い方を基礎の基礎から、クラウド環境への対応といった応用的・実践的な開発まで広く解説します。

提供元 **技術評論社** http://gihyo.jp **2名**

第1特集

プルリクしていますか？

GitHub さいしょの一歩

はじめてのPull Requestから、チーム導入へ

自分が作ったソフトウェアをGitHubに公開する――ITエンジニアの間ではそんな文化が当たり前になりつつあります。しかし、成果物を公開することはGitHubの用途の一側面にすぎません。

公開されている他者のコードを修正し、その修正を取り込んでもらう「Pull Request」という機能を使って、複数のエンジニアが協力して1つのソフトウェアを作り上げる。それこそがGitHubの醍醐味ではないでしょうか。

ただ、このPull Requestが初心者には少々ハードルが高い。GitやGitHubを使いこなすには学ぶべきことは多いですが、まずはPull Requestを出せることを目標にしましょう。本特集で、はじめてのPull Requestを出す予行演習をしてみてください。

第1章

GitとGitHubとは何か

Author 丸山 晋平（まるやま しんぺい） ㈱リラク
Twitter neko_gata_s

「GitとGitHubって同じものでしょう?」そんな誤解をしている人はいませんか? GitHubを学ぶにあたり、まずは「Gitとは何か」「GitHubとは何か」をきちんと整理しましょう。

Git、GitHubの最初の一歩を踏み出そう

本特集のテーマは「GitHubさいしょの一歩」です。近年のソフトウェア開発では、バージョン管理システム（Version Control System、以後、VCS）は「当たり前」のように使われ、VCSを扱うスキルは必須スキルとなっていると言ってもいい状況かと思います。一方で、VCSには独特の概念が多く、使い始めてみるまでは「とっつきにくい」と思われてしまうのも無理もないことかと思います。しかし、いざ使い始めてみれば、「もうVCSなしでは開発できない」となってしまうほど便利であるのも、また事実ではないでしょうか。

この特集では、VCSの一種であるGitと、その関連サービスであるGitHubの使い方を平易なチュートリアル形式で説明していくことで、その「さいしょの一歩」をサポートしたいと思います。まずは、GitとGitHubの概要を眺めるところから始めましょう。

バージョン管理システムが解決する問題

さて、これからGitとGitHubの概要を見ていくのですが、そもそもGitとGitHubとはなんなのでしょうか。簡単に言ってしまうと、GitはVCS、GitHubはGitで扱うリポジトリのホスティングサービスです。とはいえ、これだけの説明ではちんぷんかんぷんだと思いますので、まずはGitに代表されるVCSとはなんなのかから見ていきましょう。

ソフトウェアの開発を行っていると、次のような状態が頻発します。

- AさんとBさんが同じファイルを編集する
- バグが発生した際、どの編集のタイミングで発生したのか調べるために昔のソースコードを参照したい
- 場合によっては昔のソースコードの状態に戻したい
- 誰がいつどのような変更を行ったのかを記録しておきたい

これらの状況に対し、VCSなしで立ち向かおうとした場合、「ソースコードを編集するときには必ずバックアップを取ってから編集する」「編集した個所には、いつ誰が変更したのかを必ずコメントで残す」など、人の手にたよった方法で立ち向かわなければなりません。

しかし、人間は間違える生き物ですので、人力でこのようなことを行っていると必ずどこかで事故が起こります。それだけではなく、このような方法では、AさんとBさんが同じファイルを編集したときに、Aさんがすでに変更していたことにBさんが気づかず、せっかくAさんが行った作業をBさんが上書きしてしまう、と

いうような状況は防ぐことができません。これでは困ります。

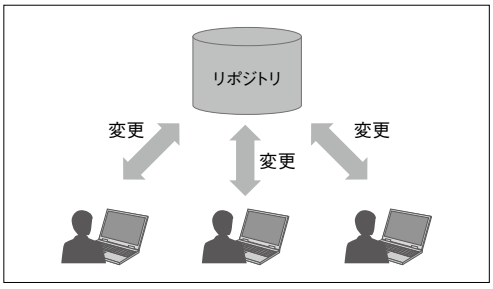
そこで、VCSの出番です。VCSのおもな機能として、「ファイルの履歴を残す」という機能と「競合(AさんとBさんが同じ部分を編集してしまう、などの状況)の検出」という機能が挙げられます。具体的な利用方法については第2章以降で触れますので、ひとまず今は「なるほど、VCSを利用すると、ファイルリネームによるバックアップ地獄や同時編集地獄、編集切り戻し地獄などから解放されるのだな」くらいに思っておいてください。

Gitの特徴

VCSの概要をざっと確認してきたところで、数あるVCSの中でGitにはどんな特徴があるのかということに話を進めます。Gitのほかにもさまざまなな VCSが存在しますが、その中でも、Git登場以前に主流だったSubversionというVCSがあります。今回は、Subversionと比べた場合のGitの特徴を見てみます。

Subversionと比べてみたとき、Gitの特徴として「分散VCSである」という点が挙げられます。多くのVCSは「リポジトリ」という「今までのソースコードのバックアップや変更履歴が格納されている場所」を持っています。GitもSubversionも例外ではないのですが、Subversionは、複数人で開発するような場合にもそのリポジトリは1つで、みんなが同じリポジトリに対して変更履歴やソースコードを格納していきます(**図1**)。

▼図1　Subversionの場合(リポジトリは1つ)



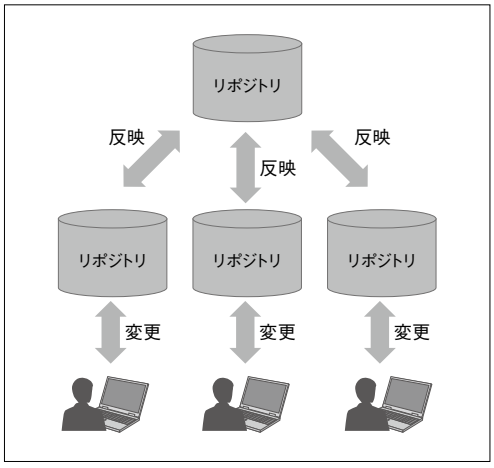
一方、Gitの場合、各人が自分のリポジトリを持っていて、各人はおのおののリポジトリに対して変更履歴やソースコードを格納していきます(**図2**)。

Subversionの場合、リポジトリに変更を格納すると、「共同のリポジトリ」にその変更が反映されるため、ただちにほかの開発メンバーに影響を与えてしまいます。そのため、気軽にトライ&エラーを繰り返すことが難しかったのですが、Gitの場合、自分のリポジトリに対して変更を格納しても、それがただちにほかのメンバーに対して影響を与えるということがありません。そのため、自分だけのリポジトリで気軽にトライ&エラーを繰り返すことができます。

トライ&エラーを繰り返したいときには、「前のコード」をとっておいて、いつでも気軽にそのときの状態に戻ったりしたいと思いますが、Gitのようにリポジトリを各人が持つようにしておけば、VCSの恩恵を受けながらトライ&エラーを繰り返すことができてうれしい、というわけです。

とはいえ、複数人で1つのソフトウェアを開発するなら、各人がそれぞれ自分のリポジトリだけで作業していては結局「みんなで作業」をすることはできません。各人がそれぞれ持っているリポジトリを統合していく必要があります。Gitでは、リポジトリ同士で「この変更内容を

▼図2　Gitの場合(各人のリポジトリとみんなのリポジトリがある)



取り込んでくれ」「取り込んだよ」というようなやりとりを行うことで、手元の「自分だけのリポジトリ」の変更内容を「みんなのリポジトリ」に対して反映したり、逆に「みんなのリポジトリ」の変更内容を手元の「自分だけのリポジトリ」に反映したりすることができます。

具体的な方法は第2章以降で触れていくので、ここでは「Gitを利用すると、みんなでリポジトリを共有する方法に比べると、気軽に自分だけのリポジトリに変更を格納していけて便利なんだな」程度に思っておいてください。

Gitを使う前提となるスキルや環境

さて、そんな便利なGitですが、Gitを扱うにはどのようなスキルや環境が必要なのでしょうか。Gitリポジトリを操作するためのアプリケーションとして、さまざまなGUIアプリが存在しますが、Gitは基本的にはUNIX系システムのコマンドラインツールとして開発されています。やはりここはUNIX系コマンドラインから利用するのが「素直」だと言えるでしょう。

この特集でもUNIXのコマンドラインの環境を使って解説していきます。そのため、基本的なファイル操作(`cd`、`pwd`、`ls`、`mkdir`など)をコマンドラインからできることが、Gitを利用するうえでの前提となります。

リポジトリホスティングサービスが解決する課題

Gitは、上述のとおり、VCSの一種、つまりソフトウェアです。一方、GitHubはGitリポジトリのホスティングサービスであると説明しました。ではリポジトリのホスティングサービスとはどんなものでしょうか。

リポジトリというのは、今までのソースコードや変更履歴が格納されているものである、と説明しました。ここで、複数人で開発するような場合についてちょっと想像してみましょう。自分1人で開発しているだけならば、リポジトリが自分のPCの中に存在するだけでかまいません。一方、複数人で開発する場合は、ほかの人からも最新のソースコードや変更履歴が見えている必要があるでしょう。そうなってくると、自分のPCの中だけにリポジトリがあっては困ります。開発者全員がアクセスできるようなサーバにリポジトリを置いておく必要が出てきます。先ほど言った「みんなのリポジトリ」が、みんながアクセスできるところに置いてある必要があるわけですね。

自分でそのためのサーバを立ててもいいのですが、そのためのサーバ管理もひと手間です。GitHubのようなホスティングサービスを利用すると、そのホスティングサービスにリポジトリを置いておくことができます。各開発者はGitHub上に置いてあるリポジトリを参照することで最新のソースコードや変更履歴にアクセスができる、という塩梅(あんばい)です。

GitHubの特徴

ホスティングサービスにはGitHubのほかにBitBucketなどがありますが、今回はGitHubに的を絞って説明しましょう。GitHubの特徴としては、次の点が挙げられます。

- 誰でも閲覧できる公開リポジトリは無料でいくつでも作成できる
- アクセス制限をかけたリポジトリも作成できるが、有料
- リポジトリごとに簡単なissue tracking system(課題管理システム)が付いてくる
- あるリポジトリに対して「こういう変更をしてみたから取り込んでくれない?」とお願いする「Pull Request(プルリクエスト)」という機能がある

さて、以上がGitとGitHubについての概要です。次の章からは、チュートリアル形式で実際に利用しながら、GitやGitHubの操作に慣れていきましょう。SD

第2章

GitHubを使うためのGitの基礎知識

Author 丸山 晋平（まるやま しんぺい） ㈱リラク
Twitter neko_gata_s

GitHubを使うためには、Gitの使い方を学ばなければなりません。ですが、数あるgitコマンドをいきなりすべて覚える必要はありません。本章ではひとまず、「stageして、コミットして、マージする」というGitでの開発における基本的な流れを行えるようにするためのコマンドを厳選して紹介します。

第1章では、GitやGitHubについての概要を見てきました。この章からは実際にGitを触りながら、その使い方を見ていきます。

Gitの準備

導入方法

何はともあれ、Gitをインストールしなければ話は始まりません。今回は誌面の都合上Mac OS XあるいはDebian系ディストリビューションに的を絞ってGitのインストール方法を説明します。Windowsの場合は、VirtualBoxなどを利用した仮想環境上にDebian系ディストリビューションをインストールして読み進めてください。

Mac OS Xの場合

Mavericks(10.9）以降のMac OS Xには標準でGitがインストールされていますが、実際に使い始めるためにはXcode Command Line Toolsというものが必要になります。まだXcode Command Line Toolsをインストールしていない場合は、ターミナルを開いて`git`コマンドを実行してみてください。すると、Xcode Command Line Toolsをインストールするように促すメッセージが表示されます。そのメッセージに従ってXcode Command Line Toolsをインストールしてください。これで、Mac OS XでGitが利用できるようになります。

Debian系Linuxの場合

パッケージ管理ソフトであるaptを利用してインストールするのが一番楽です。

```
$ sudo apt-get install git-all
```

を実行することでGitがインストールされ、`git`コマンドが利用できるようになります。

初期設定

Gitをインストールしたあとは、初期設定として、メールアドレスと名前の設定をしておきましょう。ここで設定したメールアドレスと名前は、VCSで管理する「作業履歴」に残るものです。「誰がいつどんな変更をしたか」の「誰が」の部分に使われる情報だと思っていただければ良いでしょう。

では、実際に設定をしてみましょう。次のコマンドで設定できます。

```
$ git config --global user.name ➥
'YOUR NAME'
$ git config --global user.email ➥
'yourmail@example.com'
```

`YOUR NAME`の部分と`yourmail@example.com`の部分については自分の名前とEmailアドレス

に置き換えてください。

Gitの基本的な操作方法

さて、Gitのインストールと初期設定が済んだところで、いよいよGitを利用してみましょう。今回は特定のプログラミング言語の知識に依存しないよう、単なるテキストファイルで構成されるプロジェクトのバージョン管理を題材にしながら、Gitの利用方法を学んでいきます。

リポジトリを作る

第1章で述べたとおり、Gitでは「リポジトリ」と呼ばれるところにソースコードや変更履歴を保存していきます。そのため、何はなくともまずはリポジトリを作成しなければ話は始まりませんので、そこからやっていきましょう。

最初に、プロジェクトのファイルを置くために、my_first_projectというディレクトリを作成し、そこに移動しましょう。今回は簡便のため、ホームディレクトリ直下にプロジェクト用ディレクトリを作成しますが、好きなところに作っていただいてかまいません。

```
$ cd $HOME
$ mkdir my_first_project
$ cd my_first_project
```

さて、このディレクトリが今回のプロジェクトの「作業ディレクトリ」になります。作業ディレクトリというのはその名のとおり、「作業中のファイルを置いておくところ」です。作業用のディレクトリができたので、このディレクトリで作業をしたファイルや、作業履歴を残すためのリポジトリを作成しましょう。リポジトリを作成するためのコマンドは、

```
$ git init
```

です。このコマンドを実行すると、

```
Initialized empty Git repository in
/path/to/my_first_project/.git/
```

というメッセージが表示されます。「/path/to/my_first_project/.git/に空のリポジトリを作りましたよ」くらいのメッセージですね。ここで、

```
$ ls -a
. .. .git
```

というようにカレントディレクトリの一覧を確認してみると、「.git」という隠しファイルができていることが確認できると思います。この.gitというディレクトリが、リポジトリの実体です。今後は、作業ディレクトリで作業した内容を、**git**コマンドを通じてこのリポジトリに格納していくことになります。

さて、これで「作業ディレクトリ」と「リポジトリ」の両方が用意できました。すぐに作業を開始してしまいたいところですが、ちょっと立ち止まって、「空っぽの作業ディレクトリ」と「空っぽのリポジトリ」をGitというソフトウェアがどのように認識しているかをたしかめてみましょう。

```
$ git status
```

というコマンドを実行すると、現在の作業ディレクトリとそのリポジトリをGitがどのように認識しているかを確認できます。出力は**図1**のようになるはずです。

「masterというブランチにいるよ」「最初のコミットだよ」「コミットすべきものがないよ(……)」といったような情報が出ています。ブランチについてはのちほど説明するので、今は無視しておいてください。では、「コミット」とはなんでしょうか。コミットというのは、「作業ディレクトリで作業した内容をリポジトリに格納すること」です。まだリポジトリを作成したばかりで、何もコミットしていないので、今

何かをコミットするとそれが最初のコミットになります。ですので、「最初のコミットである」という意味で`Initial commit`と表示されているのは筋が通っています。また、現在は作業ディレクトリも空っぽですので、「コミットすべきものがない」というのも筋が通っていますね。

はじめてのコミット

さて、話を戻します。今は、空の作業ディレクトリと空のリポジトリを用意したところまで進んでいるのでした。では今から、この作業ディレクトリにテキストファイルを作成して、それをリポジトリにコミットしてみましょう。

ファイルを作成

まずは、

```
hello! git
```

とだけ書いた、hello.txtというファイルをmy_first_projectディレクトリ直下に作成してみてください。これで、空だった作業ディレクトリに新しいファイルが追加されました。この状態で、再度`git status`コマンドを実行してみましょう。Gitは作業ディレクトリとリポジトリをどのように認識しているのでしょうか。**図2**のような出力が得られるはずです。

変化がありましたね。`Untracked files:`以下にいろいろな情報が出ています。詳しく見ていきましょう。

まずは一番下の行、`nothing added to commit ...`の部分です。「コミットするものは何も追加されていないけど、『untracked files』が存在するよ（trackするには云々）」というような情報が出ています。そして、`Untrakced files:`以下に、「untracked filesは以下のとおりだよ」という形で、hello.txtが挙げられています。

gitに変更を追跡させる——git add

作業ディレクトリにhello.txtファイルを作成したのに、「コミットするものがない」とはどういうことなのでしょうか？　そして、「untracked」や「track」というのはいったい何なのでしょうか？

ここで、`git status`が「Gitというソフトウェアが今の作業ディレクトリをどのように認識しているか」を表示するコマンドだったことを思い出しましょう。「trackされていないファイル」というのはつまり「Gitがtrackしていないファイル」ということです。trackというのは「追

▼図1　最初の「git status」の実行結果

```
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

▼図2　hello.txt作成後の「git status」の実行結果

```
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  hello.txt

nothing added to commit but untracked files present (use "git add" to track)
```

跡する」とかそういう意味ですね。つまり、これは「Gitは作業ディレクトリに変化があったということ自体は認識しているのだけれど、この変化をGitで追跡するようにはなっていないよ」ということです。

そこで、Untracked files:の下の行に書かれている(use git add <file>...)の部分に注目してみましょう。「コミットされるものに含めるためには、git add <file>...を使ってね」というようなことが書いてあります。

では、実際に作成したファイルをgit addしてみましょう。

```
$ git add hello.txt
```

これで、Gitがhello.txtに加えられた変更を追跡するようになったはずです。git statusで試してみましょう。図3のような出力が得られるはずです。

Changes to be commited:の下にnew file: hello.txt、つまり「新しくhello.txtというファイルができた、という変更がコミットされますよ」という情報が出てきたことがわかると思います。

コミットする——git commit

さて、これで変更をコミットする準備ができました。では、いよいよ実際にコミットしてみましょう。

```
$ git commit
```

というコマンドで、trackされている変更をコミットできます。しかし、このコマンドを実行すると、変更が即コミットされるわけではなく、なぜかエディタが立ち上がります。エディタ上には、リスト1のような表示が出ているはずです。

コミットメッセージ

最初、いきなりエディタが立ち上がってびっくりしてしまうと思うのですが、これは「コミットメッセージ」を記入するためです。では、コミットメッセージとはなんでしょうか。

リポジトリは今まで、「ソースコードや、変更履歴が保存される場所」だと言ってきました。このうち「変更履歴」について考えてみます。変更履歴を参照するとき、どのような情報がほしいでしょうか。よく5W1Hと言われるうち、「どこで」というのはソフトウェア開発におい

▼図3 hello.txtを「git add」したあとの「git status」の実行結果

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

  new file:   hello.txt
```

▼リスト1 「git commit」すると、エディタが立ち上がる

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   hello.txt
#
```

て重要ではありませんが、「いつ」「誰が」「何を」「なぜ」「どう」編集したのかは重要な情報です。このうち、「いつ」「誰が」編集したのかは、コミットするときに自動的に記録できます（最初に名前とメールアドレスを設定したのを思い出してください）。さらに、「何を」「どう」編集したのかは、編集前のコードと編集後のコードの差分を取ることで機械的に判断できます。しかし、「なぜ」そのような編集をしたのかについては、機械的に判断できません。コミットメッセージは、その「なぜ」を書いておくための場所です。

ところで、立ち上がったエディタにすでに書かれているテキストはどんな内容でしょうか。

・コミットメッセージを記入してください
・#から始まる行は無視されます（要するにコメントアウト）
・メッセージが空だとコミットされません

というようなことが書かれた下に、**Initial commit** や **Changes to be committed:...** といったように、今回コミットされるであろう変更点がまとめられています。

さて、実際にコミットメッセージを書いていきたいところですが、今回は最初のコミットですので「なぜ」の部分に書くべきことがそんなにありません。ひとまず1行目に「はじめてのコミット」とでも書いて、保存してエディタを終了させましょう。

```
[master (root-commit) 67260b7] はじめてのコミット
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
```

というようなメッセージが表示されていれば、無事にコミット成功です。念のため**git status**コマンドで現在の状況を確認してみましょう。

```
On branch master
nothing to commit, working directory clean
```

と表示されるはずです。作業ディレクトリで行った変更をリポジトリに格納したばかりですので、リポジトリに格納されている最新の状態と現在の作業ディレクトリには差異がありません。そのため、「コミットすべきものはないよ」「作業ディレクトリはきれいな状態だよ」という情報が表示されています。

◆ ◆ ◆

さて、コミットに成功したので、今までの流れをあらためて確認しておきましょう。

Gitでは、「作業ディレクトリ」で作業したソースコードや変更履歴を、「リポジトリ」というところに保存していきます。作業ディレクトリで何かしらの変更を行ったあと、**git add**コマンドを利用して「この変更はコミットしてほしい情報である」というのをGitに伝えておきます。変更点をすべて**git add**したら、**git commit**コマンドで実際に変更点をリポジトリにコミットします。その際エディタが立ち上がるので、「なぜこのような変更を行ったのか」をコミットメッセージとして残しておきましょう。

2度目のコミット

コミットまでの流れを一度体験したので、さらにコミットを重ねながらGitの便利さを体感していきましょう。

ファイルを編集

まずは作業ディレクトリのhello.txtを編集してみます。

```
hello! GitHub

Gitの操作方法について学んでいます。
```

上記のように変更しました。念のため、また**git status**で状態を確認してみましょう（**図4**）。

Changes not staged for commit:の下に、**modified: hello.txt**とありますね。まだ**git add**していないので、hello.txtの編集内容はコミットされない、というわけですね。

▼図4　hello.txtの編集後の「git status」の実行結果

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   hello.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

差分を確認——git diff

ところで、ソースコードを編集していると、「このファイルって編集前はどんなふうに書いてあったっけ」と思うことが頻繁にあります。Gitを利用していると、**git diff**というコマンドを利用することで、「以前のバージョン」からどのような変更を自分が行ったかを確認できます。ではやってみましょう。

```
diff --git a/hello.txt b/hello.txt
index a7b7233..7fd90d2 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1,3 @@
-hello! git
+hello! GitHub
+
+Gitの操作方法について学んでいます。
```

という出力が得られました。

注目すべきは、3行目以降です。**--- a/hello.txt**と書かれているのは、「以前のバージョンのhello.txtから、行が削除されているよ」という意味です。削除されたからマイナス記号、というわけですね。**+++ b/hello.txt**と書かれているのは、「以前のバージョンのhello.txtに行が追加されているよ」という意味です。追加されているからプラス記号です。

さらに下のほうを見ていくと、**-hello! git**という行、その直下に**+hello! GitHub**という行がありますが、これは「『以前のバージョンからhello! gitという行を削除したよ』『以前のバージョンにhello! GitHubという行を追加したよ』」という意味です。行を書き換えるというのは、言い換えれば「行を削除して新しく書くこと」ですね。さらに、その下の行を見ていくと、以前のバージョンに対して、空行と「Gitの操作方法について学んでいます。」という行を追加したということが見て取れます。

このように、Gitを利用すると、以前のバージョンからどこをどのように変更したのかを確認できます。**git add**する前に**git diff**を確認することで、「どんな変更をしたんだっけ」と確認したり、意図していない変更をしていないことを確認できたりして便利ですね。実は**git diff**では直近のバージョンからの変更だけではなく、さまざまなバージョン間の変更を確認できるのですが、そのあたりを詳しく知りたい方は、「Git - Book」[注1]を読んでみてください。

stageする——git add

さて話を戻して、2度目のコミットに進みましょう。編集した内容を**git diff**で確認した結果、「これで良し」と思ったとしましょう。**git add**でこの変更を「コミットする内容」としてGitに認識させたいところです。

ところで、この**git add**することをGit用語で「stageする」と言ったりします。プロダクション環境でアプリケーションを実行する前に動作確認するための環境を「ステージング環境」と言ったりしますが、Gitでも「本番コミット」する前にファイルを置いておくための「ステージング環境」があると考えるとわかりやすいでしょう。というわけで、変更をstageします。

```
$ git add .
```

注1）URL https://git-scm.com/book/ja/v2

今回はhello.txtというファイル名ではなく、カレントディレクトリを表す「.」を指定しました。実はgit addコマンドは、ディレクトリを指定するとそのディレクトリのファイルの内容すべてを再帰的に（つまり子ディレクトリ、孫ディレクトリも）stageしてくれます。一度にstageしたい場合などに便利ですので、覚えておくと良いでしょう。

さて、これでhello.txtへ行った変更がstageされました。念のためgit statusで状況を確認すると、図5のような出力が得られます。「hello.txtの変更がコミットされるよ」と書かれています。良いですね。

stageを取り消す——git reset

ところで、先ほど「stageするのはステージング環境で確認するようなものだ」と言いましたが、ステージング環境で確認したものは「やっぱりやーめた」とできなければステージング環境の意味がありません。ここで、一度stageした変更を「やっぱりやーめた」とするにはどうすればいいかということを見てみましょう。

実は、stageを取り消す方法はgit statusの出力結果に書いてあります。(use "git reset HEAD <file>..." to unstage)というのがそれです。stageを取り消すので「unstage」、というわけですね。書いてあるとおり、git reset HEAD hello.txt[注2]とすればhello.txtの変更をstageしたものは取り消され、stageされていない状態に戻ります。

unstageに限らず、git statusの出力にはしばしば「Xをしたいときには Yコマンドを利用してくださいね」といった情報が出ているので、「コマンド全部覚えなきゃ！」と思って気負わず、git statusの指示に従って利用していけば、そのうち自然にコマンドを覚えられます。こまめにgit statusで現在の状況を確認する癖を付けると良いでしょう。

▼図5　編集済みのhello.txtを「git add」したあとの「git status」の実行結果

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  modified:   hello.txt
```

コミットする——git commit

さて、unstageの方法を確認したところで、2度目のコミットをしてみてください。コミットの方法はもうわかりますね。今回のコミットメッセージは、

```
hello.txtの内容を修正

Gitの特集ではなくてGitHubの特集なので ↵
s/git/GitHub/した
```

くらいにしておきましょう。コミットメッセージの慣習として、1行目にEmailでいうところのSubject、3行目以降に本文を書くという慣習があります。このようにしておくと、後述するgit logやGitHubが“いい感じ”に履歴を表示してくれるので、なるべくこのフォーマットを守ると良いでしょう。

履歴を確認する

さて、これで無事2回コミットを行うことができました。Gitの機能の1つとして「履歴を残す」というのがありましたが、残した履歴はあとから参照できなければ意味がありません。ここでは、リポジトリに残った履歴を参照する方法を学んでいきましょう。

履歴を参照——git log

履歴を参照するコマンドは、git logです。実際にターミナルでgit logを実行してみてください。図6のような出力が得られます。

git logの出力はページャ（lessなど）で展開

注2）これはstageに乗っているhello.txtをHEADブランチの状態に戻すという意味です。HEADブランチについてはp.32で説明しますが、要するにこれでstageのhello.txtを直前のコミットの状態に戻せる、ということです。

されますので、ログを閉じたい場合はページャを終了させてください（lessの場合はqを入力）。

今まで行ったコミットの詳細が、新しいものから順に書かれています。commitの横に書かれている乱数のような文字列は「コミットハッシュ」と呼ばれるもので、そのコミットのIDのようなものです。その下に「誰がいつ行ったコミットか」が続き、その下にコミットメッセージが表示されています。このログを見ることで、「いつ誰がどのようなコミットをなぜ行ったのか」を知ることができます。

コミットの詳細を確認——git show

ところで、たとえば2回目のコミットで「実際にどのファイルにどういう変更を行ったのか」ということを詳しく知りたくなったときにはどうすればいいでしょうか。そういうときには**git show**コマンドが便利です。**git show <commit hash>**とすることで、そのコミットの詳細を見ることができます。今回の筆者の環境であれば、2回目のコミットのコミットハッシュは**a84d2a3fd0c7fe1a7b87adddea193a92cf32b041**ですので、次のコマンドを実行します。

```
$ git show a84d2a3fd0c7fe1a7b87adddea193a92cf32b041
```

すると、**図7**のような出力が得られるでしょう。

コミットログのほかに、どのファイルに対してどのような変更を行ったかも出力されました。

ファイルの変更を確認——git diff

単にファイルの変更だけ見たい場合は、**git diff <from commit hash> <to commit hash>**

▼図6 「git log」の実行結果

```
commit a84d2a3fd0c7fe1a7b87adddea193a92cf32b041
Author: shinpei maruyama <shinpeim@gmail.com>
Date:   Fri Jun 3 20:36:03 2016 +0900

    hello.txtの内容を修正

    Gitの特集ではなくてGitHubの特集なので s/git/GitHub/した

commit 67260b7e75f60e2f2e02c44827c7afd66dc148a9
Author: shinpei maruyama <shinpeim@gmail.com>
Date:   Fri May 27 16:35:41 2016 +0900

    はじめてのコミット
```

▼図7 「git show」の実行結果

```
commit a84d2a3fd0c7fe1a7b87adddea193a92cf32b041
Author: shinpei maruyama <shinpeim@gmail.com>
Date:   Fri Jun 3 20:36:03 2016 +0900

    hello.txtの内容を修正

    Gitの特集ではなくてGitHubの特集なので s/git/GitHub/した

diff --git a/hello.txt b/hello.txt
index a7b7233..7fd90d2 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1,3 @@
-hello! git
+hello! GitHub
+
+Gitの操作方法について学んでいます。
```

を利用できます。今回であれば、最初のコミットから2回目のコミットまでにどのような変更が行われたかを知るためには、

```
$ git diff 67260b7e75f60e2f2e02c44827c7afd66dc148a9 a84d2a3fd0c7fe1a7b87adddea193a92cf32b041
```

とすれば良いでしょう。

今回は最初と2回目のコミットを比べましたが、**<from commit hash> <to commit hash>**のところに好きなコミットハッシュを指定することで、任意のコミット間の変更を確認できます。

ブランチを利用する

さて、これまでで、コミットやコミット間の差異を確認する方法を学んできました。ここからGitの本領、「ブランチ」の利用方法を見ていきましょう。

ブランチが解決するもの

とはいえ、いきなり利用方法だけ見ても何が便利なのかわからないので、ブランチの利用方法を見ていくその前に、「ブランチを利用することでどんな問題が解決できるのか」ということを把握しておきたいと思います。

開発を行っていると、「Aの作業をしている途中、緊急対応でBの作業が必要になってしまった」というようなことが頻発します。このとき、Aの作業はまだ中途半端な状態ですのでリリースできません。Aの作業を始める前の状態に一度ソースコードを「巻き戻し」したあと、Bの作業を行ってリリースしたいですよね。そしてそのあと、まだ途中だったAの作業をまた手元に引き寄せて開発に戻る、ということをしなければなりません。しかし、これを手動でやろうと思うと、かなりたいへんです。たいへんなだけならともかく、事故が起こりそうです。ブランチは、このような作業をサポートしてくれます。

ブランチを一覧——git branch

では実際にブランチについて見てみましょう。簡単に言うと、ブランチというのはあるコミットを指すセーブデータのようなものです。百聞は一見にしかずと言いますので、実際に操作することでそれがどういうものなのか見てみたいと思います。

まずは、現在どのようなブランチ（＝セーブデータ）があるのかを確認してみましょう。

```
$ git branch
```

とすることで、ブランチの一覧を表示できます。

```
* master
```

という出力が得られたかと思いますが、このmasterというのは、いわば「デフォルトのセーブデータ」です。masterの左に「*」がついていますが、これは「今進めているセーブデータはmasterだよ」ということを表しています。

ブランチを作成・選択——git checkout -b

では、ここで、別のセーブデータ（＝ブランチ）を作成してみましょう。

```
$ git checkout -b branch_a
```

とコマンドを入力すると、

```
Switched to a new branch 'branch_a'
```

という表示が出たと思います。これは「新しくbranch_aというセーブデータを作って、そっちに移りました」というような意味です。

git branchで確認してみましょう。

```
* branch_a
  master
```

という出力が得られたのではないかと思います。branch_aとmasterというブランチがあり、現在選択されているのがbranch_aであることがわかります。

このとき、**git status**でGitの状態を確認すると

```
On branch branch_a
nothing to commit, working directory clean
```

となっています。今まで**On branch master**と書かれていたところが**On branch branch_a**となっています。「選択しているセーブファイルがbranch_aである」というような感じですね。

別のブランチでコミット

では、branch_aを選択している状態でhello.txtを編集してコミットしてみましょう。hello.txtの内容は**リスト2**にしましょう。この変更をコミットしてみてください。コミットメッセージは「branch_aで作業してみた」くらいにしておきましょう。**git log**で確認して、無事に新しくコミットできたことを確認してください。

ブランチにおける巻き戻し

ところで今、branch_aというブランチで新しくコミットをしましたが、これはbranch_aのセーブデータを「より進んだセーブポイント」でセーブしたようなものです。このとき、masterブランチはどのようになっているでしょうか。branch_aで作業を行ってコミットしましたが、masterブランチではコミットをしていません。つまり、masterブランチにはbranch_aで作業する前の状態が保存されているはずです。

これを確かめるために、実際にmasterブランチを選択してみましょう。ブランチを選択するには**git checkout**コマンドを利用します。

▼リスト2　hello.txtを編集

```
hello! GitHub

Gitの操作方法について学んでいます。
branch_aを新しく作ってbranch_aで作業しています。
```

```
$ git checkout master
```

さて、これでmasterブランチを選択できました。**git branch**で、masterの横に「*」が表示されていることを確認してください。その状態で、**git log**をしてみましょう。先ほどbranch_aでコミットした「branch_aで作業してみた」というコミットが見えなくなっているはずです。さらに、作業ディレクトリ内のhello.txtの内容を確認してみましょう。branch_aで作業を行う前の状態に戻っているのが見て取れると思います。

これはいったいどういうことでしょう？　状況を整理しましょう。

セーブデータのたとえで言うと、branch_aを選択したときにコミットした変更内容は、branch_aセーブデータにセーブされましたが、masterセーブデータにはセーブされていません。繰り返しになりますが、masterにはbranch_aで作業する前の状態が保存されています。ここでmasterブランチを選択するのは、そのmasterセーブデータをロードするようなものです。masterブランチにはbranch_aで作業する前の状態が保存されているので、結果的に、作業コピーの内容と**git log**の内容がbranch_aセーブデータで作業する前の状態に巻き戻せたということです。

ここで、ブランチが解決してくれる問題がどんなものだったか、思い出してみましょう。「Aの作業をしているときに緊急対応でBの作業が入ってきた、Aの作業を巻き戻してBの作業を行いたい」というような問題でした。

実は、今までの手順がまさにこの問題を解決する手順となっています。Aの作業をする前にbranch_aというブランチを作成し、そちらを選択しました。masterというブランチには「それまでの作業」が保存されています。そのあと、作業Aの内容をbranch_aに保存しましたが、

masterには以前の状態が残ったままです。ここで、masterブランチを選択すれば、以前の状態を手元に再現できたわけですね。

これで「Aの作業を巻き戻す」ということが実現できました。あとはここから作業Bを進めていけば良いです。作業Bを進めているあいだにも、branch_aには作業Aで行っていた内容が保存されていますので、branch_aで行った作業は無駄になっていません。

割り込み作業のためのブランチを作成

さて、それでは実際に作業Bを行っていきましょう。master上で作業Bを行っても良いのですが、作業Bを行っている途中でまた割り込みの作業が入ってくるかもしれません。念のため「作業B用のセーブデータ」をもう1つ作って、そのブランチを選択しておきましょう。

```
$ git checkout -b branch_b
```

さて、ここからは実際の作業です。作業Bとして、新しいファイル「b.txt」を作りましょう。内容は、

```
branch_bで作られたファイルです
```

くらいにしておきましょう。コミットメッセージは「branch_bでb.txtを作成」で良いでしょう。

これで作業Bは無事に終わりました。この内容をリリースするために、メインであるところのmasterブランチに、今branch_bで行った変更を取り込みたいところです。

ブランチに、別のブランチの変更を取り込む——git merge

あるブランチで行われた変更内容を選択しているブランチに取り込むためには、**git merge**というコマンドを利用します。今回ならばmasterブランチにbranch_bの内容を取り込みたいので、masterブランチを選択し、**git merge**を行います。

```
$ git checkout master
$ git merge branch_b --no-ff
```

git mergeコマンドを実行すると、エディタが立ち上がったかと思います。

git mergeは「あるブランチの変更を選択したブランチに取り込むコマンドである」と説明しましたが、これは言い方を変えれば「masterブランチにbranch_bで行った変更をコミットする」ということでもあります。このように、あるブランチで行った変更を取り込んでコミットするようなコミットのことを「マージコミット」と呼びます。VCSの能力の1つとして「履歴を残す」というものがあると言いましたが、「branch_bの内容をmasterに取り込んだよ」という変更履歴を残すためのコミットである、と言ってもいいでしょう。

というわけで、コミットメッセージは「緊急でbranch_bで作業した内容を反映」くらいにしておきましょう。これで、masterブランチに、branch_bで作業した内容を取り込むことができました。

ところで、**--no-ff**というオプションを付けていますが、この**--no-ff**というオプションは何でしょうか。実はこのオプションを付けないと、マージコミットが生成されず、単にmasterがbranch_bと同じところに進んでしまう場合があります。この**--no-ff**はちょっと複雑ですので、のちほどあらためて詳述しますが、今はひとまず「マージするときに**--no-ff**を付けると確実にマージコミットを残すことができるんだな」くらいに思っておいてください。

状況確認

さて、**git merge**を利用してmasterにbranch_bの内容を取り込んだマージコミットを作成するところまでやってきたのでした。**git log**でコミットの履歴を確認してみてください。branch_bで作業した内容のコミットと、そのコミットを取り込んだという内容のコミットが新たにmasterの履歴に生じているはずです。さらに、作業ディレクトリ内の内容を見てみてください。

branch_aで行った変更内容は反映されていない一方、branch_bで追加したb.txtはきちんと存在しているはずです。

ここで、現在の状況を整理しましょう。現在、master、branch_a、branch_bの3つのブランチが存在しています。そして、それぞれ、

- branch_aには途中だった作業Aが保存されている
- branch_bには緊急で行った作業Bの内容が保存されている
- masterには作業Bの内容を取り込んだ内容が保存されている

という状況ですね。

コミットグラフ

今回はそこまで複雑ではないですが、より状況が複雑になってきたとき、各ブランチにどんな内容が保存されているかを脳内だけで覚えておくのは現実的ではありません。そこで、各ブランチがどのように進んでいるのかを確認するのに便利なコマンドをこのあたりで覚えておきましょう。筆者のお勧めは**図8**に挙げるコマンドです。これを実行すると、**図9**のような出力が得られると思います。

各ブランチがどのように派生し、どのようにマージされたのか、線でつながれて表示されています。このようなものを「コミットグラフ」と呼んだりします。便利ですね。

しかし、毎回オプションだらけのコマンドを叩くのは少ししんどいです。そこで、「エイリアス」という機能を利用して**git graph**と打つだけで同様の出力を得られるようにしておきましょう。

図10コマンドを入力すると、今度から**git graph**と入力しただけで**図8**で入力したコマンドと同じ結果を得られます。

さて話を戻して、この形式のコミットグラフの読み方についてもう少し詳しく見てみましょう。「*」がコミットを表しています。その横には括弧があったりなかったりしますが、そのコミットを指すブランチがある場合、括弧でブランチ名が表示されます。その横にはコミットの概要が表示されています。

「HEAD」って何?

図9に**HEAD -> master**[注3]という見慣れない表示が出ていることに気づいたでしょうか。**master**はmasterブランチがここを指している、

注3) 環境によっては**HEAD, master**となっている場合があります。

▼図8 ブランチの状況（コミットグラフ）を表示するコマンド

```
$ git log --graph --date-order --all --pretty=format:'%h %Cred%d %Cgreen%ad %Cblue%cn
%Creset%s' --date=short
```

▼図9 「図8」のコマンド実行結果

```
*   3333d1c (HEAD -> master) 2016-06-07 shinpei maruyama 緊急でbranch_bで作業した内容を反映
|\
| * 6963c6d (branch_b) 2016-06-07 shinpei maruyama branch_bでb.txtを作成
|/
| * 393fc82 (branch_a) 2016-06-07 shinpei maruyama branch_aで作業してみた
|/
* 50189e4  2016-06-07 shinpei maruyama hello.txtの内容を修正
* 67260b7  2016-05-27 shinpei maruyama はじめてのコミット
```

▼図10 「図8」のコマンドを「git graph」だけで実行できるようエイリアスを設定

```
$ git config --global alias.graph "log --graph --date-order --all --pretty=format:'%h
%Cred%d %Cgreen%ad %Cblue%cn %Creset%s' --date=short"
```

ということでいいと思いますが、HEADとはいったい何なのでしょう。HEADというのは、簡単に言うと「今選択しているブランチ」のことを指します。現在はmasterブランチを選択しているので、HEADがmasterを指している、というわけですね。

HEADの動きを確認するために、実際に別のブランチを選択してみましょう。

```
$ git checkout branch_a
```

上記コマンドを実行したのち、再度git graphしてみると、HEADがbnrach_aを指しているのが確認できるのではないでしょうか。

割り込み作業を終了、本作業へ

さて、これで各ブランチの内容が一目で確認できるようになりました。branch_bはもういらないブランチになったので、このブランチは消してしまいましょう。ブランチを消すにはgit branch -dです。

```
$ git branch -d branch_b
```

これで、割り込みで発生した作業Bは完全に完了ですね。おめでとうございます。

さて、作業Bが完了したのは良いのですが、作業Aがまだ途中で放置されていますね。そこで、branch_aで引き続き作業を行っていきましょう。

```
$ git checkout branch_a
```

でbranch_aを選択し、hello.txtを次のように書き換えます。

```
hello! GitHub

Gitの操作方法について学んでいます。
branch_aの作業を完了しました
```

この内容をコミット（コミットメッセージは「作業Aを完了」にしましょう）したら、masterを選択してbranch_aの内容をマージし（マージコミットのメッセージは「branch_aの作業をマージ」でいいでしょう）、不要になったbranch_aを削除してみてください。コマンドは示さないので、今まで学んだコマンドを駆使して自分でやってみてください。最終的にコミットグラフが図11のようになっていれば成功です！

◆ ◆ ◆

さて、以上がブランチの説明となります。ブランチを利用することで、差し込みの作業などにも対応できることが見て取れたと思います。今回は、各ブランチが同じファイルの同じところを修正してしまった場合に起こる「コンフリクト」や、各ブランチで行ったコミットをあとから編集するようなことについては触れていません。そのあたりについて知りたい方は「Git - book」を参照してください。

落穂ひろい：--no-ff

ところで、マージについて説明するときに、--no-ffについてはのちほど詳述する、と言ったまま説明を続けてしまいました。このあたりで、

▼図11　最終的なコミットグラフ

```
*   d8c640c  (HEAD -> master) 2016-06-07 shinpei maruyama branch_aの作業をマージ
|\
| * a98e583  2016-06-07 shinpei maruyama 作業Aを完了
* |   3333d1c  2016-06-07 shinpei maruyama 緊急でbranch_bで作業した内容を反映
|\ \
| * | 6963c6d  2016-06-07 shinpei maruyama branch_bでb.txtを作成
|/ /
| * 393fc82  2016-06-07 shinpei maruyama branch_aで作業してみた
|/
* 50189e4  2016-06-07 shinpei maruyama hello.txtの内容を修正
* 67260b7  2016-05-27 shinpei maruyama はじめてのコミット
```

--no-ffについてきちんと説明しておきましょう。

実験のための環境準備

すでに軽く触れましたが、Gitではマージをするときに、マージコミットを作る場合と作らない場合があります。実際に試してみるのが一番ですので、ここで適当な場所（今回はホームディレクトリにします）に新しくff_merge_testというディレクトリとリポジトリを作成して、実験してみましょう。

ディレクトリにリポジトリを作成する方法はもう大丈夫ですね。

```
$ cd $HOME
$ mkdir ff_merge_test
$ cd ff_merge_test
$ git init
```

でOKです。まずはここにあたらしくhello.txtを作り、

```
Hello, ff-merge!
```

という内容で保存し、この内容をコミットしてください。コミットメッセージは「最初のコミット」でいいでしょう。ここでgit graphを確認すると、図12のようになっています。

さてここで、新しい作業を行いたいと思ってbranch_aを切ったとしましょう。

```
$ git checkout -b branch_a
```

さらにそのブランチでhello.txtを次のように編集します。

```
Hello, ff-merge!
ブランチAでの作業です。
```

この内容を「ブランチAで作業」というコミットメッセージでコミットしましょう。そのうえでgit graphを実行すると図13のようになっているはずです。

masterにはブランチAでの作業が保存されておらず、branch_aには保存されている、という状況を作ることができました。

--no-ffの検証

ここで、masterを選択したうえで、--no-ffを付けずにbranch_aをマージしてみましょう。

```
$ git checkout master
$ git merge branch_a
```

--no-ffを付けたときと異なり、エディタが立ち上がらず、

```
Updating 328870b..3b66d5b
Fast-forward
 hello.txt | 1 +
 1 file changed, 1 insertion(+)
```

というような表示が出てきたかと思います。ここでgit graphを確認してみましょう。図14のようになっているはずです。

マージコミットが作成されず、単にmasterの位置がbranch_aと同じところに移動しました。作業ディレクトリの内容を確認すると、hello.txtの内容はbranch_aと同じになっています。

▼図12　masterブランチでコミット後の「git graph」の実行結果

```
* 328870b  (HEAD -> master) 2016-06-15 shinpei maruyama 最初のコミット
```

▼図13　branch_aブランチでコミット後の「git graph」の実行結果

```
* 3b66d5b  (HEAD -> branch_a) 2016-06-15 shinpei maruyama ブランチAで作業
* 328870b  (master) 2016-06-15 shinpei maruyama 最初のコミット
```

▼図14 「--no-ff」を付けずにマージしたあとの「git graph」の実行結果

```
* 3b66d5b  (HEAD -> master, branch_a) 2016-06-15 shinpei maruyama ブランチAで作業
* 328870b  2016-06-15 shinpei maruyama 最初のコミット
```

Fast forwardマージ

このように、Gitは「単にブランチを進めるだけでマージしたいブランチにたどり着ける」といった場合、--no-ffオプションを付けなければマージコミットを作成せず、単にブランチをそこまで進める形で、マージしたいブランチの内容を取り込みます。このようなマージの方式を「Fast forwardマージ」と呼びます。--no-ffは「no fast forward」の略だったわけですね。

しかし、「あるブランチでの変更内容を取り込んだ」というのは立派な履歴ですので、普通はマージコミットがあったほうがうれしいはずです。あえて履歴を残したくないパターンというのはあるのでしょうか。

実はいくつかあります。そのうちの1つは第3章であらためて説明しますが、ほかにも次のようなシチュエーションを考えてみてください。

> あなたは、新機能の開発のため、masterブランチからnew_featureブランチを作成し、new_featureブランチでいくつか作業を進めていました。その途中で使ったことのないライブラリを使ってみようと思い立ちました。しかし初めて使うライブラリですので、本当に今回のユースケースにマッチするのかどうか、自信がありません。そこで、「実験用」ブランチとしてtest_new_libraryというブランチをnew_featureから作成し、test_new_libraryブランチで新しいライブラリを試してみることにしました。こうしておけば、もしも今回使ってみたライブラリが目的にうまくマッチしなかったとしても、new_featureブランチに戻り、test_new_libraryブランチを捨ててしまえば、なんのリスクもなく新しいライブラリを試すことができます。
>
> 一方「実験」がうまくいって、このままこのライブラリを利用したコードをnew_featureに取り込みたいとなったときのことを考えてください。プロダクトにとって、「新機能を作るブランチを切って、その内容を取り込んだ」というのは重要な情報ですが、「あなたが新しいライブラリを試行錯誤するためにブランチを切った」というのはあまり重要でない情報に思えます。ならば、その試行錯誤がうまくいったなら、new_featureブランチを単にtest_new_libraryブランチのところまで進めてしまえば良い、という考え方ができないでしょうか。

このようなときには、--no-ffを付けず、Fast forwardマージを行うという選択をすれば良いでしょう。

まとめると、

- マージコミットに情報を残したいならば--no-ffを付けてマージ
- マージコミットを残す必要がなければ--no-ffを付けずにFast forwardマージ

とすれば良い、ということです。

まとめ

この章では、Gitの基本的な使い方について見てきました。基本的に、

- 保存したい変更内容をgit addして
- git commitで保存
- 何か作業を行う前はブランチを切ってそこで作業する
- 作業を行ったブランチの内容をgit mergeで取り込む

という流れの繰り返しで、Gitを利用した開発は進んでいくことを確認してください。

次の章では、GitHubを利用して共同作業を行う方法を学んでいきます。SD

第3章

GitHubでPull Requestを出せるようになろう

Author 丸山 晋平（まるやま しんぺい） ㈱リラク
Twitter neko_gata_s

前章までで、1人でGitを利用する方法を一通り見てきました。本章では、GitHubを利用して、1つのプロジェクトを複数メンバーで進めるやり方を見ていきましょう。

GitHubの導入

1人でGitを使っているだけならば、自分のローカルにリポジトリを置いておくだけで十分ですが、同じプロジェクトを複数人で進めていくなら、「みんながアクセスできるリポジトリ」が必要となります。第1章で述べたとおり、自前でサーバを用意しても良いのですが、GitHubにリポジトリをホスティングすることで、手軽にオープンなリポジトリを立てられます。この章では、GitHubへのアカウント登録から、複数人でGitHubを利用するやり方について見ていきましょう。

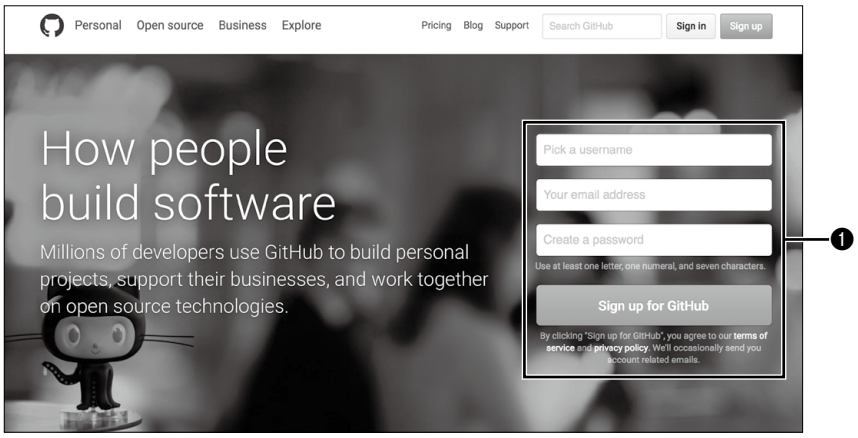
GitHubのアカウントの作成

GitHubを利用するためには、何はともあれアカウントを作成する必要があります。アカウントをすでに作成している方は、この節は読み飛ばしていただいてかまいません。

まずは、GitHub[注1]にアクセスしてみましょう。**図1**の❶のフォームに、取得したいユーザ名（英数とハイフンが利用できますが、ハイフンで始まったりハイフンで終わったりするユーザ名は利用できません）と、メールアドレス、好きなパスワードを入力し、「Sign up for GitHub」をクリックしてください。入力したメールアドレスに確認用のメールが届きます（**図2**）ので、「Verify email address」と書かれたボタン（❷）か、「Button not working? Paste the following link into your browser:」と書かれた下に書かれているURL（❸）にアクセスしてください。これでユーザ登録は完了です。

注1） URL https://github.com/

▼図1 GitHubのトップページ



▼図2　アカウント作成の確認用メール

▼図3　公開鍵を登録する

公開鍵の登録

さて、アカウントが無事に作られたところで、「ssh公開鍵」の登録をしておきましょう。公開鍵を登録しておかないと、リポジトリにアクセスするたびにパスワードを入力しなければなりませんが、一度公開鍵を登録しておけば、煩わしいパスワード入力なしで権限のあるリポジトリにアクセスできるようになります。公開鍵をすでに登録している方はこの節は読み飛ばしていただいてかまいません。

まずはsshキーペアを作成します（すでにキーペアを作成している場合は作成しなくてかまいません）。ホームディレクトリに「.ssh」というディレクトリを作成し、そのパーミッションを0700としてください。その後、**ssh-keygen**というコマンドを実行すると、そのディレクトリ内にid_rsaという秘密鍵と、id_rsa.pubという公開鍵を作成できます。途中で「どこに秘密鍵を作成するか」とパスフレーズを聞かれるので、お好きなパスフレーズを設定してください。パスフレーズを設定したくない場合は何も入力せずに Enter でOKです。

```
$ cd $HOME
$ mkdir .ssh
$ chmod 0700 .ssh
$ ssh-keygen
```

さて、これで秘密鍵と公開鍵のキーペアが作成できました。このうち、id_rsaというファイル、つまり秘密鍵は厳重に管理し、どこにも漏れないようにしてください。一方、id_rsa.pubというファイル、つまり公開鍵は、「公開」鍵というほどですので、公開しても良いものです。

このあたりの話を簡単に説明すると、公開鍵は「錠」、秘密鍵が「その錠を開けるための鍵」のようなものです。GitHubに公開鍵、つまり錠を預けることで、GitHubは機密情報に対してその錠を利用してアクセス制限をかけます。このアクセス制限を突破できるのは、秘密鍵（つまり錠の対となる鍵）を持っている人だけ、というわけです。錠をいくら他人に渡したところで問題にはなりませんが、鍵を他人に渡してしまったら、その錠は開け放題になってしまいます。そのため、繰り返しになりますが秘密鍵は厳重に管理してください。

では、GitHubに対して公開鍵を登録しましょう。https://github.com/settings/keysにアクセスすると（**図3**）、「New SSH Key」というボタンがあります（❹）。そこをクリックすると入力フォームが出現します。「Key」というところ（❺）に、~/.ssh/id_rsa.pubの内容をコピーして貼り付けてください。「Add SSH key」と書かれたボタン（❻）を押せば公開鍵の登録は完了です。

GitHub上で新しいリポジトリを作成

さて、これでGitHubを利用する準備が整いました。ここで、GitHub上に新しいリポジトリを作ってみましょう。第2章までで利用してきたmy_first_projectリポジトリをGitHub上でホスティングしたいと思います。

https://github.com/にアクセスし、「New Repository」と書かれたボタンをクリックしてみてください。これで新しいリポジトリの置き場を作成するページに遷移します。

ここで「Owner」とあるのは、「そのリポジトリの所有者は誰か」を設定する部分です。ここでは自分のユーザ名がOwnerになっていると思いますので、そのままでOKです。実はGitHubにはOrganizationという機能があり、それを利用していると、ここで自分のユーザ名以外を指定できます（Organizationについてはコラムを参照してください）。

「Repository name」には作成するリポジトリの名前（今回ならばmy_first_projectで良いでしょう）、「description」にはそのリポジトリを端的に表す説明を入力しましょう。今回は「Software Design 2016年8月号のチュートリアル」くらいにして、さりげなくSoftware Designの宣伝をしておけば編集の方が喜ぶのではないでしょうか（もちろん自由に設定していただいてかまいません）。

「Public」、「Private」という選択肢があります。GitHubの有料ユーザとなると、Privateを選択できるようになりますが、今回はPublicを選択しておきましょう。Privateを指定すると、リポジトリにアクセス制限をかけられます。具体的には、Ownerと、Ownerが指定したユーザ（コラボレータと呼びます）のみがリポジトリを閲覧したり編集したりできるようになります。

「Initialize this repository with a README」以下は、すでにローカルにリポジトリが存在する場合は無視しておきましょう。チェックを入れたり、None以外を選んだりすると、README やLICENSEが置かれたリポジトリが作成されてしまいます。

最後に「Create repository」ボタンをクリックすると、新しいまっさらのリポジトリ置き場が作成されます。

Column Organization

GitHubには「Organization」というしくみがあり、ユーザは複数のOrganizationに属すことができます。Organizationはリポジトリの所有者となることができ、そのOrganizationに属したユーザが、権限に応じた操作をそのリポジトリに対して行えるようになります。

たとえば、あるユーザはリポジトリの参照はできるがリポジトリに対して新しいコミットを登録できない、また別のユーザはそのOrganizationに新しいリポジトリを作成できるし、そのOrganizationのどのリポジトリにも自由にアクセスできるなど、細かいアクセス制御が可能になるわけです。組織的な開発をしたいときには便利な機能です。

ローカルのリポジトリをpushする

リポジトリを作成すると、そのリポジトリのトップへ遷移します（**図4**）。URLはhttps://github.com/ユーザ名/my_first_projectとなっているはずです。このページからは、このリポジトリの状況が確認できます。

現在はまっさらな状態ですので、「次にどうするべきか」が書かれています。今回は手元にあるリポジトリをインポートしたいので、「…or push an existing repository from the command line」の指示に従いましょう。

図4の❼に示したボタンをクリックすると、その左に書かれている一連のコマンドがクリップボードにコピーされます。ローカルのmy_first_projectに**cd**で移動したうえでそのコマンドを実行すると、my_first_projectリポジトリがGitHub上にホスティングされます。

クリップボードにコピーされているものをい

きなりターミナルに貼り付けるのが不安な方（まっとうな感覚です）は、一度なんらかのテキストエディタなどに貼り付けて内容を確認するのも良いプラクティスだと思います。

さて、これでGitHub上に1つの「リモートリポジトリ」と、手元のマシンに「ローカルリポジトリ」があるような状態となりました。

リモートリポジトリとは、その名のとおり「リモート」、つまり手元のマシンではなくてどこかのサーバに置いてあるリポジトリです。Gitを利用して複数人で共同作業するためには、ローカルリポジトリに対して行ったコミットをリモートリポジトリに反映したり、逆にリモートリポジトリに誰かが反映したコミットをローカルリポジトリに反映したりしながら作業することになります。第1章の内容を思い出してください。

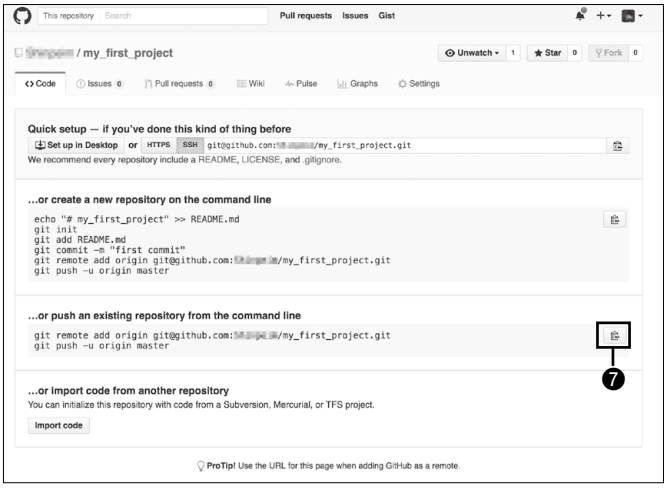
先ほど入力したコマンドを詳しく見ていくと、さらにその内容がよくわかります。まずは1つ目のコマンドから解説していきましょう。

```
$ git remote add origin git@github.com:ユーザ名/my_first_project.git
```

git remote addで、このリポジトリに対するリモートリポジトリを登録することができます。ここでは、**git@github.com:ユーザ名/my_first_project.git**という場所に存在するリポジトリ（つまり、先ほどGitHub上で作った空のリポジトリ）を、originという名前で登録しています。このコマンドを実行することで、GitHub上に作ったリポジトリとローカルに作ったmy_first_projectリポジトリが関連付けられたと思えば良いでしょう。

```
$ git push -u origin master
```

▼図4　ローカルのリポジトリをpushする



で、ローカルに存在するmasterブランチの内容を、origin（つまり、先ほど登録したリモートブランチ）に反映しています。**git push**コマンドの詳細については後述しますが、ひとまず今は「pushコマンドでローカルリポジトリのコミットの内容をリモートリポジトリに反映できる」くらいの理解をしておいてください。

複数人でGitを使うときの基本的な操作方法

さて、ここまでで、公開されたリポジトリと、それをリモートリポジトリとした手元のリポジトリが用意できました。ここからは、一人二役をこなしながら実際のチーム開発を体験していきます。

リモートリポジトリをcloneしてくる

今までmy_first_projectで作業してきたのをAさんとしましょう。今までは1人で作業していたAさんですが、Bさんがチームに加わったとしましょう。Bさんは、GitHub上にあるmy_first_projectに対して新しいコミットを反映したり、Aさんによる作業を手元に持ってきたりしたいはずです。

ここで、Gitではリモートリポジトリに対する変更をローカルリポジトリから反映したり、その逆にリモートリポジトリのコミットをローカ

ルリポジトリに反映したりしながら作業を行うということを思い出してください。Bさんは、なんとかしてGitHub上にホスティングされているmy_first_projectをリモートリポジトリとしたローカルリポジトリを作成しなければなりません。

どこかのサーバに存在するリポジトリをリモートリポジトリとしたローカルリポジトリを作成するには、**git clone**コマンドが利用できます。今回ならば、どこか適当なディレクトリ（今回はホームディレクトリとします）に移動したうえで**図5**のコマンドを実行してください。これは、

- git@github.com:ユーザ名/my_first_project.gitに存在するリポジトリを
- my_first_project_for_bという名前のローカルリポジトリにコピーし
- 元のリポジトリをoriginという名前のリモートリポジトリとして登録する

というコマンドです。リポジトリのコピーとリモートリポジトリの登録を一度にやってしまうコマンドだということですね。

さて、これであなたのマシンの手元にはAさんが作業するためのmy_first_projectディレクトリと、Bさんが作業するためのmy_first_project_for_bディレクトリができあがったわけですが、これだとちょっと一貫性がないので、my_first_projectをmy_first_project_for_aにリネームしておきましょう。ディレクトリをリネームしても、リポジトリに対して影響はありません（ディレクトリ内の.gitというディレクトリがリポジトリの正体だったことを思い出してください）。

ローカルリポジトリでコミットを重ねる

さて、ここで、Bさんになって新しい作業を行いましょう。このリポジトリにはREADMEが存在しないので、README.mdを作るという作業を行うこととします。Bさんの作業ディレクトリとローカルリポジトリはmy_first_project_for_bにありますから、まずはそのディレクトリに移動します。そして、何か作業するときにはまずは作業用のブランチを作成するのでしたね。

```
$ cd my_first_project_for_b
$ git checkout -b add_readme
```

でブランチを切って、次のようなREADME.mdを作成しましょう。

```
# MY FIRST PROJECT

チュートリアル用のリポジトリです
```

さて、この内容をコミットしておきましょう。コミットメッセージは「READMEを作成」くらいでいいでしょう。

コミットしたら、**git graph**で現在のコミットグラフを確認してみましょう（エイリアスを設定していない方は第2章を読み返してください）。**図6**のような出力が得られたはずです。

一番上に、今さっき行ったコミットが表示されていますね。add_readmeブランチもそのコミットを指していて、HEADもそこを指しているのは良いでしょう。ここがわからない方は第2章を読み返してください。

ところで、上から2つめのコミットに「**origin/master, origin/HEAD**」という見慣れない表示があります。これは、見てのとおりoriginというリモートリポジトリのブランチがどこを指しているかを表しています。

そのため、このグラフを読み解くと、

- ローカルリポジトリのadd_readmeブランチが最新のコミットを指している

▼図5　リポジトリのコピーとリモートリポジトリの登録を行う

```
$ git clone git@github.com:ユーザ名/my_first_project.git my_first_project_for_b
```

・HEADはadd_readmeを指している
・originというリモートリポジトリのmasterブランチが最新の1個前のコミットを指している
・originというリモートリポジトリのHEADはoriginのmasterブランチを指している
・ローカルリポジトリのmasterも最新の1個前のコミットを指している

ということがわかります。

ローカルリポジトリの内容をリモートリポジトリにpush

さて、現在の状況を確認しましたが、add_readmeブランチはローカルにしか存在しない状態です。これをほかの人が確認できるように、リモートリポジトリにもadd_readmeブランチがあるようにしたいですね。というわけで、ローカルリポジトリに反映したコミットをリモートリポジトリに対して反映してみましょう。ローカルリポジトリのコミットをリモートに反映するためには**git push**です。

git pushコマンドはオプションや引数をフルで利用する場合、**git push <options> <repository> <local branch>:<remote branch>**となります。今回であれば、ローカルのadd_readmeブランチと同じものをリモートのoriginにも作りたいので、

```
$ git push -u origin add_readme:add_readme
```

となります。しかし、**add_readme:add_readme**の部分が同じブランチ名の場合、省略可能となっており、一般的には省略された形式で

```
$ git push -u origin add_readme
```

とすることが多いでしょう。

ところで、この**-u**というオプションはなんでしょうか。これは、「リモート側のブランチをローカル側のブランチの追跡ブランチにする」というオプションです。追跡ブランチについては詳述しませんが、**-u**を付けることで、ローカルのブランチとリモートのブランチが関連付けられた状態になる程度に思っておいてください（詳しく知りたい場合は、「Git - Book」[注2]を参照してください）。

それでは、実際に上記のコマンドを実行し、ローカルリポジトリの内容をリモートリポジトリに反映させてみてください。

git pushでローカルリポジトリの内容をリモートリポジトリに反映したら、再度**git graph**を確認しましょう。**図7**のような出力が得られるかと思います。

origin/add_readmeという新しいリモートブランチができあがっているのが見えるかと思います。これで、ローカルリポジトリのコミットをリモートリポジトリに反映することができました。

注2）URL https://git-scm.com/book/ja/v2

▼図6　add_readmeをコミットした直後の「git graph」の実行結果

```
* 1f0f5e3  (HEAD -> add_readme) 2016-06-14 shinpei maruyama READMEを作成
* d8c640c  (origin/master, origin/HEAD, master) 2016-06-07 shinpei maruyama branch_aの作業をマージ
|\
| * a98e583  2016-06-07 shinpei maruyama 作業Aを完了
* |   3333d1c  2016-06-07 shinpei maruyama 緊急でbranch_bで作業した内容を反映
|\ \
| * | 6963c6d  2016-06-07 shinpei maruyama branch_bでb.txtを作成
|/ /
| * 393fc82  2016-06-07 shinpei maruyama branch_aで作業してみた
|/
* 50189e4  2016-06-07 shinpei maruyama hello.txtの内容を修正
* 67260b7  2016-05-27 shinpei maruyama はじめてのコミット
```

実際にGitHub上の表示がどうなっているか、リポジトリのトップページを見てみましょう。**図8**のように、ブランチの数が2になっていて(❽)、「your recently pushed branches：」というところに、add_readmeブランチが表示されており、「Compare & pull request」と書かれたボタンが出現したのが見て取れると思います(❾)。

さて、このまま手元でadd_readmeブランチをマージしてしまっても良いのですが、せっかくですのでGitHub上でPull Requestを出してみましょう。Pull Requestというのは、「こういうブランチを作ったので、マージしてください」というリクエストをGitHub上で出すことです。先ほど出現した「Compare & pull request」というボタンをクリックしてみてください。すると、**図9**のようなPull Request作成画面に移ります。

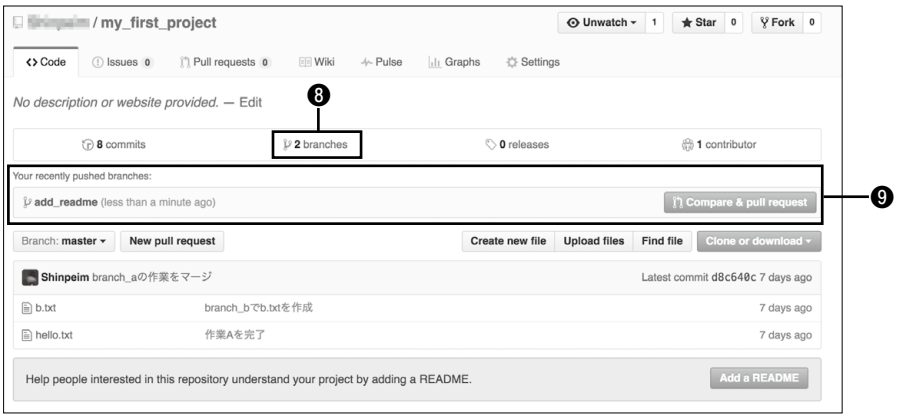
上から順に、どのブランチをどのブランチにマージしたいのか（compareと書かれている側のブランチをbaseと書かれている側のブランチにマージしたい、という意味です）という情報エリア(❿)、Pull Requestの内容の入力エリア(⓫)、このPull Requestに含まれるコミットの内容(⓬)、このPull Requestで変更されたファイルの差分(⓭)が表示されています。では、内容を「はじめてのpull requestです」と入力して、「Create pull request」ボタンをクリックしてみましょう。

Pull Requestが作成され、そのPull Requestのページに遷移したかと思います(**図10**)。⓮のタブに注目してください。このタブの「Commits」を選択すると、このPull Requestに含まれるコミットの一覧が表示されます。また、「Files changed」を選択すると、このPull Requestで変更されるファイルの差分一覧が表示されます。

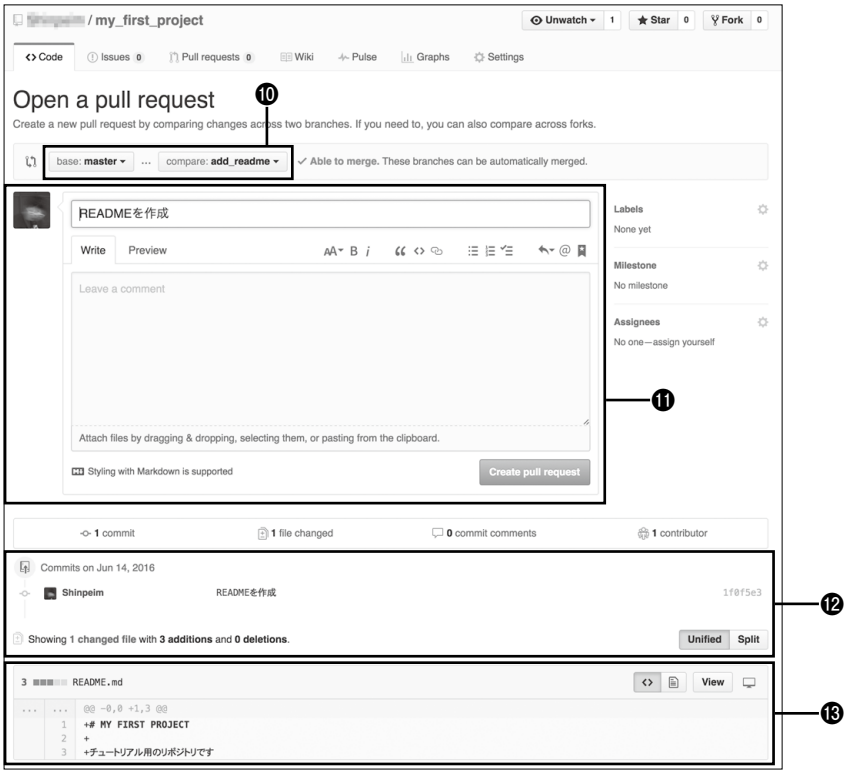
▼図7　add_readmeをリモートに反映した直後の「git graph」の実行結果

```
* 1f0f5e3  (HEAD -> add_readme, origin/add_readme) 2016-06-14 shinpei maruyama README作成
*   d8c640c  (origin/master, origin/HEAD, master) 2016-06-07 shinpei maruyama branch_aの作業をマージ
|\
| * a98e583  2016-06-07 shinpei maruyama 作業Aを完了
* |   3333d1c  2016-06-07 shinpei maruyama 緊急でbranch_bで作業した内容を反映
|\ \
| * | 6963c6d  2016-06-07 shinpei maruyama branch_bでb.txtを作成
|/ /
| * 393fc82  2016-06-07 shinpei maruyama branch_aで作業してみた
|/
* 50189e4  2016-06-07 shinpei maruyama hello.txtの内容を修正
* 67260b7  2016-05-27 shinpei maruyama はじめてのコミット
```

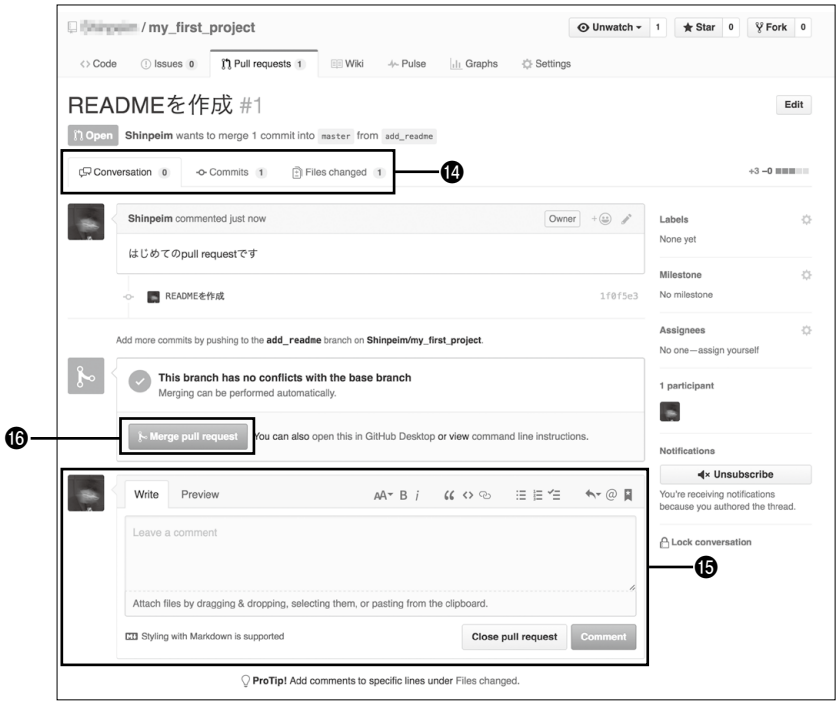
▼図8　add_readmeをリモートに反映した直後のGitHubの表示



▼図9　Pull Request作成画面



▼図10　作成したPull Requestのページ



Pull Requestを出された側は、GitHub上で「どんな変更内容が入ったブランチなのか」を確認できるわけですね。

また、⓯のフォームに入力すれば、このPull Requestに対してコメントを残せます。ここのコメントで「ここのコードはちょっと筋がよくない気がする」とか「ここの部分だけ直したらマージするよ」といったようなコミュニケーションを行うことで、ブランチごとのコードレビューがとてもやりやすくなっています。

今回は一人二役でやっているので、自らこのPull Requestをマージしてしまいましょう。⓰の「Merge pull request」をクリックすると、入力欄が出現します。ここには、このPull Requestのブランチをマージすることで生じるマージコミットのコミットメッセージを入力します。今回はデフォルトで入力されているものをそのまま送信してしまいましょう。これで、このPull Requestをマージすることができました。

この状態で、リポジトリのトップページに戻ってみてください。すると、masterにマージされた内容が反映されていると思います。また、README.mdの内容が整形されて表示されているかと思います。実は、GitHubには、README.mdをリポジトリのルートディレクトリに置いておくと、その内容をよしなに表示してくれるという機能があります。今、README.mdをコミットしたものをmasterにマージしたことによって、その内容が表示されるようになったわけです。

README.mdには、ライブラリならばそのライブラリのインストール方法や利用方法など、そのリポジトリを閲覧している人にとって有益なことを過不足なく書いておくと良いでしょう。

リモートリポジトリの内容に追いつく

さて、これで、GitHub上のリモートリポジトリではmasterにadd_readmeブランチの内容が反映された状態になりました。ここで、Aさんの気持ちになってみてください。Bさんが行った作業の内容が、リモートリポジトリに反映されました。AさんはBさんと一緒に開発をしているので、自分のローカルリポジトリにもBさんが行った変更内容を取り込みたいはずです。

というわけで、今度はAさんになって、これらの変更をローカルリポジトリに取り込んでみましょう。my_first_project_for_aに移動し、**git graph**でコミットグラフを確認してみてください。**図11**のような出力が得られると思います。

おや、おかしいですね。GitHub上のリモートリポジトリのmasterはadd_readmeの内容を取り込んでいるはずですが、origin/masterはadd_readmeが取り込まれる前のままの状態です。

実は、Gitのリモートリポジトリは、**git log**や**git status**を実行するたびに自動でサーバを見にいって「今ローカルが知っているリモートリポジトリの状況」を更新するというような気のきいたことをしてくれません。それもそのはずで、**git log**や**git status**を叩くたびにサーバにアクセスしないといけないとなれば、

▼図11　my_first_project_for_aでの「git graph」の実行結果

```
*   d8c640c  (HEAD -> master, origin/master) 2016-06-07 shinpei maruyama branch_aの作業をマージ
|\
| * a98e583  2016-06-07 shinpei maruyama 作業Aを完了
* |   3333d1c  2016-06-07 shinpei maruyama 緊急でbranch_bで作業した内容を反映
|\ \
| * | 6963c6d  2016-06-07 shinpei maruyama branch_bでb.txtを作成
|/ /
| * 393fc82  2016-06-07 shinpei maruyama branch_aで作業してみた
|/
* 50189e4  2016-06-07 shinpei maruyama hello.txtの内容を修正
* 67260b7  2016-05-27 shinpei maruyama はじめてのコミット
```

無駄なトラフィックがたくさん発生してしまいますし、ネットワークにつながっていないときには作業ができなくなってしまいます。せっかく手元にリポジトリがある分散型VCSなのに、リモートリポジトリにアクセスできない環境では何もできなくなってしまっては元も子もありません。

そこで、まずは**git fetch**コマンドを利用して、「ローカルが知っているリモートリポジトリの状況」を最新の情報に更新する必要があります。サーバから最新のリモートリポジトリの状況をフェッチしてくる（取ってくる）、というわけですね。**git fetch**コマンドにはとくに引数は必要ありません。単に

```
$ git fetch
```

と実行してみてください。その後、**git graph**でコミットグラフを確認すると、**図12**のような出力が得られるはずです。

origin、つまりGitHub上のリモートリポジトリのmasterは、手元のmasterより進んでおり、add_readmeの内容が取り込まれていることがわかります。

さて、ここで手元のmasterブランチもorigin/masterに追いつかせたいですね。そうしないと、いつまでたっても手元のmasterはadd_readmeの内容が反映されていないままの状態です。

ここで、第2章でやった**git merge**のときの**--no-ff**を思い出しましょう。**--no-ff**を付けずにマージすると、線をたどってそのブランチに追いつけるような場合には、マージコミットを作成せず、単に対象のブランチのところまでブランチを進めるのでした。今回ならば、masterブランチをorigin/masterブランチに追いつかせたいわけですが、masterとorigin/masterブランチの関係を見てみてください。一直線に上に進んでいけば、masterはorigin/masterに追いつくことができます。というわけで、

```
$ git merge origin/master
```

というコマンドを実行すると、手元のmasterブランチをorigin/masterブランチに追いつかせることができます。実際に実行してみて、その後**git graph**でブランチの状況を確認してみてください。

ちなみに、この一連の流れを自動で行う**git pull**というコマンドもあるのですが、**git pull**はあまりにいろいろなことを自動でやってくれ過ぎるので、慣れないうちは**git fetch**と**git merge**を利用して、リモートリポジトリとローカルリポジトリがどのようになっているのか逐一確認しながら作業を行うのをお勧めします。

▼図12　リモートの状況を更新した直後の「git graph」の実行結果

```
*   7113fb9  (origin/master) 2016-06-14 GitHub Merge pull request #1 from Shinpeim/add_readme
|\
| * 1f0f5e3  (origin/add_readme) 2016-06-14 shinpei maruyama README作成
|/
*   d8c640c  (HEAD -> master) 2016-06-07 shinpei maruyama branch_aの作業をマージ
|\
| * a98e583  2016-06-07 shinpei maruyama 作業Aを完了
* |   3333d1c  2016-06-07 shinpei maruyama 緊急でbranch_bで作業した内容を反映
|\ \
| * | 6963c6d  2016-06-07 shinpei maruyama branch_bでb.txtを作成
|/ /
| * 393fc82  2016-06-07 shinpei maruyama branch_aで作業してみた
|/
* 50189e4  2016-06-07 shinpei maruyama hello.txtの内容を修正
* 67260b7  2016-05-27 shinpei maruyama はじめてのコミット
```

ForkとPull Request

さて、以上で、1つのリポジトリに対して複数人で作業を行えるようになりました。

ところで、いくら公開しているリポジトリとはいえ、自分のリポジトリに他人が自由にコミットできるとなってしまったら、さすがにまずいですよね。ご心配なく。実はGitHubのPublicなリポジトリは、「閲覧は誰でもできるけど、pushするにはそのリポジトリの所有者であるかコラボレータである必要がある」というアクセスポリシーとなっています。

では、自分がオーナーでもコラボレータでもないリポジトリに対して何か変更をしたい場合、どのようにすればいいのでしょうか。その場合には、GitHubのForkという機能を利用できます。

まず、Forkしたいリポジトリにアクセスしてください（**図13**）。すると、右上に「Fork」というボタン（⓱）があるので、ここをクリックして進めていくことで、Fork元のリポジトリが自分のGitHub上にコピーされます。

これで、Fork元をまるっとコピーした、自分が所有者となったリモートリポジトリが作成されました！　自分が所有者になったので、あとはこのリポジトリを`git clone`してくれば、自分が所有しているリポジトリをリモートリポジトリとしたローカルリポジトリを手元にゲットできます。

リモートリポジトリを`git clone`する際には、「どこからcloneするのか」を指定する必要がありますが、Forkされてできた自分のリポジトリの「Clone or download」ボタンを押すと、そのリポジトリをどこからcloneすれば良いのかが表示されます（**図14**の⓲）。

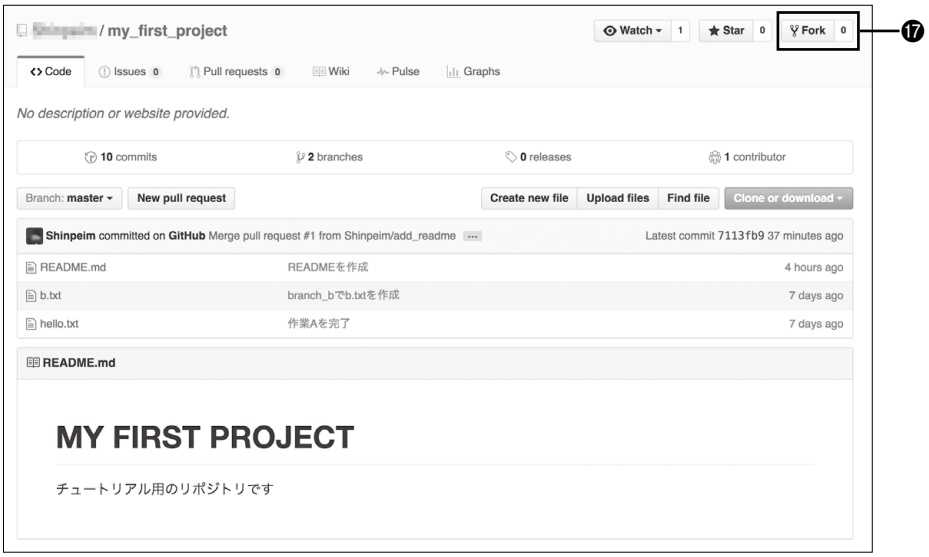
というわけで、

```
$ git clone ⑱で表示されたもの ローカルで保存したい場所
```

と実行してやれば、自分が所有者となったリモートリポジトリがローカルにcloneされます。

あとは、cloneされたローカルリポジトリに対して作業用ブランチを作成し、そこで行った作業をコミットし、それをリモートリポジトリに対してpushしてやれば、自分が所有している「Fork元のコピー」のリポジトリに対して

▼図13　「Fork」ボタンで他者のリポジトリを自分のGitHub上にコピーする



修正内容を反映させられます。

この「自分が所有しているリポジトリに存在するブランチ」を「Fork元のリポジトリ」に取り込んでほしい場合にも、Pull Requestを利用できます。

GitHubの自分のリポジトリ上から、Pull Requestを作成すると、**図15**のような画面が出てきます。base(⑲)にFork元のリポジトリを指定してやれば、自分が所有者でもコラボレータでもないリポジトリに対してPull Requestを送ることができます。あとは、元のリポジトリの所有者がこのPull Requestを受け付けてくれれば、無事にあなたのコミットが元のリポジトリに取り込まれる、というわけです！

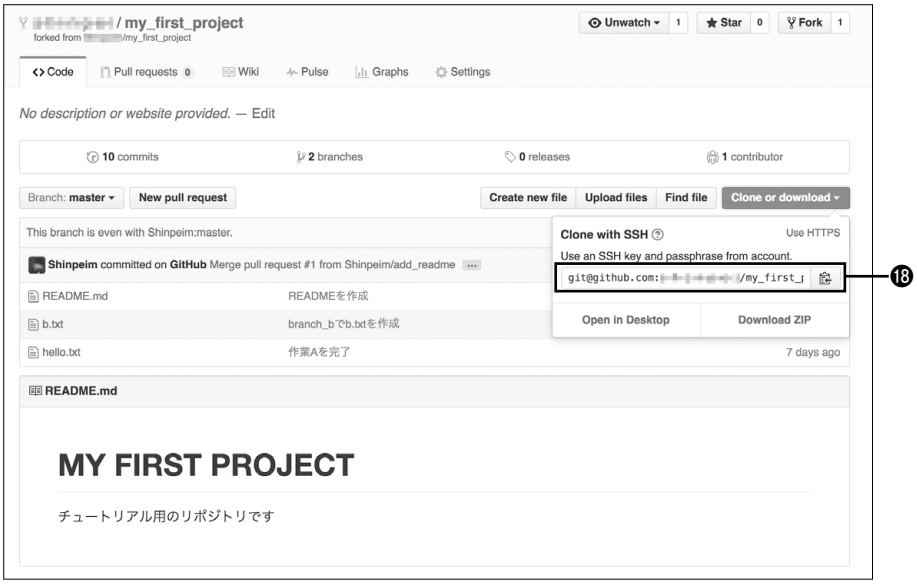
このとき、Pull Request内容には「なぜ、どのような作業を行ったのか」というのを端的に書いておくと良いでしょう。

まとめ

さて、以上で、GitHubを利用した複数人での開発の流れをざっと見てきました。これで、一通りGitとGitHubを利用した開発ができるようになったのではないでしょうか。より詳しいGitの利用方法については、何度も挙げていますが「Git - Book」を参照してください。公式のドキュメントであり、なおかつ非常にわかりやすく日本語でまとまっています。

より実践的に、チーム開発でどのようにGitHubを利用するのかについては、次の章をご覧ください。SD

▼図14 「Clone or download」ボタンでclone元を確認する



▼図15 Fork元のリポジトリに対してPull Requestを送る

第4章

[事例紹介]GitHubをチーム開発に導入するときに考えること

Author 福本 貴之（ふくもと たかゆき） ㈱ガイアックス
Twitter @__papix__

ここまでGitHubの基本的な使い方についてみてきましたが、「では企業やチームにおいてどのようにGitHubを利用すれば良いのか」を最後の章で紹介します。GaiaxでのGitHub導入・活用事例を参考に、ユーザとリポジトリの管理、ブランチの運用において気を付けておくことを解説していきます。

GaiaxとGitHub

第1章から第3章まで、GitHubやGitの基本的な使い方について説明をしてきました。本章では、筆者が所属するGaiaxという会社において、GitHubがどのように導入され、どのように活用されているかについて、具体的な例を紹介したいと思います。

今回紹介するGaiaxでの経験は、会社やチームの状況、文化など、読者のみなさんの環境と違うところも多々あるかと思います。とはいえ、本章で紹介するGitHub導入と活用の経験が、みなさんが会社やチームへGitHubを導入するにあたって、少しでも参考になればうれしいです。

Gaiaxの状況

「GitHubをどのように導入したか」について述べる前に、まず簡単ではありますがGaiaxの状況について説明をしておこうと思います。

Gaiaxは「主軸」と言えるような大きなプロダクトを持たず、10以上の小規模から中規模のプロダクトを開発、運用して利益を出すタイプの企業です。そして、その事業のそれぞれにエンジニアによる開発チームがあり、日々プロダクトの開発と運用に取り組んでいます。プロダクトの規模にもよりますが、開発チームは1～5人程度のエンジニアによって構成されることが多く、事業規模と同じく開発チームの規模も小規模～中規模です。

一方、Gaiaxには各事業の開発チームとそこに所属するエンジニアを横断的に支援するための「技術開発部」という部署があり、その中にエンジニア文化の醸成（じょうせい）を担う「技術推進室」というチームがあります。今回のGitHub導入も、この技術推進室のメンバーと、社内の有志エンジニアが中心となって進めていきました。

GitHubを使い始めるまで

Gaiaxは、バージョン管理システムとしてGitを採用しており、GitHubを導入する前は、リポジトリをホスティングするSaaSとしてBitBucketを活用していました。しかしながら、とくに若手のエンジニアを中心として、GitHubへ移行したいという声が挙がり始めました。

その理由としては、日常的に個人プロジェクトやOSSなどでGitHubを使っていて、BitBucketよりも使い慣れていること、Travis CIやCircleCIのように、GitHubにしか対応していないSaaSが多く存在すること、利用事例や便利なツールなども、GitHubにのみ対応しているものが多いこと、などがありました。

これらの声を受け、技術推進室が中心となってGitHubへの移行プロジェクトがスタートしました。とはいえ実のところ、Gaiaxにおける GitHubへの移行プロジェクトは、現時点でま

だまだ移行途中です。これは、社内に開発チームが多数存在しているため、それぞれの都合を調整して一気に移行することが困難だったためです。そのため、現在は各々の開発チームの状況に合わせて、順次BitBucketからGitHubへの移行を進めています。

とはいえ、GaiaxとしてはBitBucketとGitHubを併用し続けるつもりはありません。締め切りを定めつつ、GitHubをうまく活用するための社内ルール作りはもちろん、各開発チームへのアドバイスも行いながら、ゆっくりとGitHubへの移行を進めているという状態です。

ルール

これまで説明してきたような、「BitBucketからGitHubへの移行」といったSaaSの載せ替えという取り組みは、適当になりがちなSaaSの利用ルールを整理する良い機会と言えるでしょう。

Gaiaxの場合、GitHubを導入する前に利用していたBitBucketにおいて、ユーザの集合を意味する「User Group」の設定がかなり適当になってしまっていたりと、いくつかの問題が存在していました。そこで、GitHubへの移行を進めるにあたって、事前に移行計画や利用ルールを用意して、これに従いながら移行を進めていきました。

個人的には、会社やチームで利用するSaaSについては、利用ルールなどは定めず、各々のチームやエンジニアの裁量に任せて自由に使ってもらうというのが理想だと思っています。

とはいえ、ルールが定まっていたほうが「ルールに従って進めれば良い」という意味で、移行への障壁が下がる部分があります。また、あとからルールを定めてそれに適応するように設定を変更していくより、最初はしっかりルールを定めておいてから、チームやエンジニアがGitHubに慣れてくるのに応じてルールを緩くしていくほうが良いのではないか、という判断があり、今回はこのような方針でGitHub移行を進めていくことになりました。

◆ ◆ ◆

ここからは、具体的にBitBucketからGitHubへの移行を進めるにあたって、どのようなルールを定めたかについて解説していきたいと思います。読者のみなさんがGitHubへの移行を考えるにあたっては、これから紹介する例はあくまで参考にしつつ、それぞれの会社やチームの状況と文化に沿ったルール策定を進めていくのが大事になるでしょう。

GitHubの管理

ここでは、GaiaxがGitHubにおけるアカウントやリポジトリをどのように管理しているのかを紹介していきます。

アカウント

GitHubを会社やチームに導入するにあたっては、

- GitHubのOrganizationアカウントを用意する
- GitHub Enterpriseを導入する

の2つの方法があります。

GitHub Enterpriseを導入する場合、AzureやAWSといったIaaS、もしくはオンプレミスのサーバでGitHub Enterpriseを運用しなければならないので、今回はGaiaxのOrganizationアカウントを用意し、このOrganizationアカウントに各エンジニアのGitHubアカウントを所属させる形で導入を進めました。

また、各エンジニアのGitHubアカウントについては、すでに趣味やOSSなどでGitHubを活用しているエンジニアにはそのアカウントを使ってもらうようにして、まだGitHubのアカウントを持っていないエンジニアには個人でアカウントを用意してもらうことにしました。

各エンジニアのGitHubアカウントについては、趣味やOSS用のアカウントと、会社やチームの業務のためのアカウントを別に用意すると

いう選択肢もあります。ただ、GitHubには公式で複数のアカウントを切り替える機能はありませんし、業務の一環としてOSSに関する活動を行うこともありますので、今回はどちらも同じアカウントを利用できるようなルールにしました。

とはいえ、たとえばエンジニアの退職時に、そのエンジニアのGitHubアカウントがOrganizationアカウントに所属したままではいけません。そのため、アカウントの管理については、技術開発部の社内情報システムチームと連携して、Qiita:TeamやSlackなど、GitHub以外のSaaSと併せて管理を行うようにしています。

ユーザのロール

GitHubのOrganizationアカウントには、GitHubアカウントを持つユーザをOrganizationの「Member」として所属させることができます。そして各ユーザには、「Owner」と「Member」という役割（Organization role）を割り当てることができます。

「Owner」の権限を持つユーザは、「Member」の権限を持つユーザとは異なり、Organizationの設定と、Organizationが持つすべてのリポジトリに関する設定を変更できます。そのため、「Owner」のロールを持つユーザの数は、なるべく少なくなるように制限しましょう。

Gaiaxの場合、基本的に部長などの管理職と技術推進室のユーザのみ「Owner」にしており、それ以外のユーザはすべて「Member」のロールに設定しています。

リポジトリの権限

GitHubでは、リポジトリ単位で、各ユーザに対して「Admin」「Write」そして「Read」という3つの権限を設定できます。リポジトリに対して行える操作は、その操作を行うユーザが操作対象となるリポジトリに対してどの権限を持っているかによって変化します（**表1**）。

Organizationに所属する各ユーザに、各リポジトリに対する権限を与える方法は3つあります。

- 「Default Repository Permission」を設定する
- ユーザをリポジトリの「Collaborators」にする
- ユーザの集合である「Team」を作り、Teamにリポジトリの権限を割り当てる

Default Repository Permission

GitHubのOrganizationアカウントでは、Organizationの設定（Settings）の「Member privileges」で、「Default repository permission」を設定できます。ここで設定した権限は、Organizationに所属するすべてのユーザが、Organizationに存在するすべてのリポジトリに対してデフォルトで与えられる権限になります。

「Default repository permission」では、先ほど紹介した「Admin」「Write」「Read」に加え、何も権限を与えない「None」のいずれかを選択できます。この「None」を設定することで、「Read」以上の権限を別途付与しない限り、Organizationに存在するリポジトリをGitHub上から閲覧できないようにできます。

ちなみに、リポジトリの作成の許可については、「Member privileges」で「Allow members to create repositories for this organization」のチェックを入れることで、Organizationに所属するすべてのユーザにリポジトリの作成を許可することができます（後述の、リポジトリの

▼表1　GitHubにおけるリポジトリに対する権限

権限	できること
Admin	GitHub上でのリポジトリの閲覧、Gitでのclone、pull、pushの操作、リポジトリの設定（Settings）の変更
Write	GitHub上でのリポジトリの閲覧、Gitでのclone、pull、pushの操作
Read	GitHub上でのリポジトリの閲覧、Gitでのclone、pullの操作

Collaboratorsによって関連付けられたOrganizationに所属していないユーザを除く)。

Gaiaxでは、リポジトリの作成は全ユーザに許可し、「Default repository permission」については「Read」を設定しています。そして「Write」以上の権限を与える場合は、後述の「Team」を利用して権限を設定するという方針にしています。

Collaborators

GitHubでは、リポジトリ単位で「Collaborators」を追加できます。これは、Organizationに所属するユーザはもちろんのこと、そうではないユーザをリポジトリに関連付けることができ、ユーザごとに「Admin」「Write」そして「Read」の権限を与えることができます。

Gaiaxでは、Organizationに所属するユーザ(Gaiax社員)については、なるべく後述のTeamを使って権限を与えるようにしており、「Collaborators」はGaiax社員ではないユーザに権限を与えたい場合にのみ使うようにしています。

実際にGaiax社員ではないユーザへ「Collaborators」を使ってリポジトリへの権限を付与した例としては、エンジニア新人研修を他社と合同で開催した際、その相談や資料作成のためのリポジトリをGaiaxのGitHub Organizationアカウントで作成し、他社の研修担当者をリポジトリのコラボレータとして追加して、共同作業を行ったということがありました。

Team

GitHubのOrganizationアカウントでは、Organizationに所属するユーザの集合を「Team」として設定できます。そしてOrganizationアカウントに存在する各リポジトリに対して、チーム単位で権限を割り当てることができます。

たとえば、社内に「Reactio」という事業があり、その事業は「Reactio」と「Reactio-WebSite」というリポジトリをOrganizationアカウント上に保有しているとします。このとき、あるユーザにこれら2つのリポジトリ権限を与える場合、前述の「Collaborators」を使って権限を設定することもできます。ただしこの場合、「Reactio」と「Reactio-WebSite」という2つのリポジトリに対して、それぞれ権限を設定しなければなりません。

しかし、あらかじめ「Reactio-Developers」のようなチームを用意して、「Reactio」と「Reactio-WebSite」リポジトリでこのチームに対して適切な権限を設定しておけば、「Team」にユーザを追加するだけで、そのユーザは「Reactio」と「Reactio-WebSite」リポジトリに対して、設定された権限で操作を行うことができます。

同様に、「Team」からユーザを削除すれば、「Reactio」と「Reactio-WebSite」リポジトリに対して適用されていた権限は、すぐさま無効になります。

Gaiaxでは、事業に所属する開発チームや有志のチーム(たとえば、社内で利用するツールの開発チームや、新人研修の担当チームなど)ごとに、GitHub上で「Team」を設定しています。そして、それらのチームが利用するリポジトリに対して、適切に「Admin」や「Write」などの権限を与えるようにしています。

GitHubの使い方

ここまで、GaiaxがGitHubのOrganizationアカウントをどのように管理しているかについて話してきました。ここからは、筆者が所属しているReactio[注1]という事業の開発チームを例にして、どのようにGitHubを活用しているかについて、より具体的にお話したいと思います。

リポジトリ

つい先日、GitHubの料金プランの変更が発表されたのは、記憶に新しいことだと思います。

注1) URL https://reactio.jp

この新しいプランでは、リポジトリ数の制限が撤廃された代わりに、Organizationアカウントはユーザ1人につき毎月9ドル（ただし、5人までの場合は毎月25ドル）という料金体系になりました。

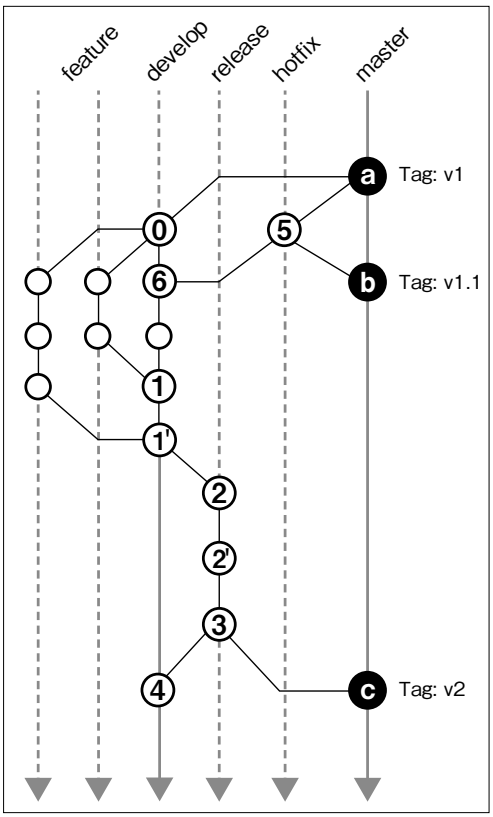
Gaiaxは、この料金プランの変更が行われる前からGitHubを利用しており、現時点では古い料金プランのまま利用を続けています。旧料金プランでは、ユーザ数は無制限である代わりにリポジトリ数によって料金が変化するため、リポジトリの数はなるべく減らしたいところです。

とはいえ、プロダクトを構成するさまざまなコード（Webアプリケーション本体のコード、iOS/Androidアプリケーションのためのコード、ItamaeやAnsibleなどのインフラのためのコード……）を1つのリポジトリにまとめてしまうと、IssueやPull Requestの扱いがとても複雑になってしまいます。そのため、プロダクトを構成するコードを役割ごとに分離して、そのひとつひとつにリポジトリを割り当てていくことが重要です。

次は、Reactioチームにおけるリポジトリ構成の一例です。実際にはこのほかにも、いくつかのリポジトリがあります。

- Reactio……Webアプリケーション本体のコード
- Reactio-WebSite……ReactioのWebサイトのコード
- Reactio-Template……AWSのAMIを生成するためのPacker用テンプレート

▼図1　Git Flow



ブランチ運用

GitやGitHubを活用して開発を行っていくにあたって、ブランチの運用は非常に重要です。ここでは、代表的なブランチ運用の手法である「Git Flow」と「GitHub Flow」について解説します。

Git Flow

Git Flowでは、次の5種類のブランチが登場します。

- master
- develop
- feature
- hotfix
- release

これらのブランチとそのコミットを図にすると、**図1**のようになります。この図を使いながら、Git Flowの流れを見ていきましょう。

Git Flowにおいて、新しい機能の実装は、developブランチとfeatureブランチを中心に行います。開発者はdevelopブランチ（**0**）からfeatureブランチを作成し、featureブランチでプロダクトに追加する機能の実装を行います。図のように、featureブランチは複数生成され、同時にいくつかの実装が進行することがほとん

どです。

featureブランチでの実装が終わると、featureブランチからdevelopブランチへマージし、実装をdevelopブランチに取り込みます（1）（1'）。

developブランチの内容をリリースする場合、まずはdevelopブランチからreleaseブランチを作成します（2）。細かな修正があればreleaseブランチで修正を行い（2'）、リリース準備が終わった段階でreleaseブランチをmasterブランチにマージして、リリースを行います（3）。またこのとき、masterブランチのコミットには、適切にバージョニングしたタグを付与するようにします（c）。

releaseブランチで修正した場合、その変更はdevelopブランチへ取り込みます（4）。このdevelopブランチから、また新たなfeatureブランチを作成して、機能の実装を進めていきます。

一方、リリースにバグがあった場合、そのリリースのコミット（a）からhotfixブランチを生成します（5）。バグ修正が終われば、hotfixブランチで行った変更はmasterブランチに取り込み、リリースとタグ付けを行います（b）。また、同時にdevelopブランチに対しても変更の取り込みを行います（6）。

これがGit Flowによるブランチ運用と、開発の流れです。

GitHub Flow

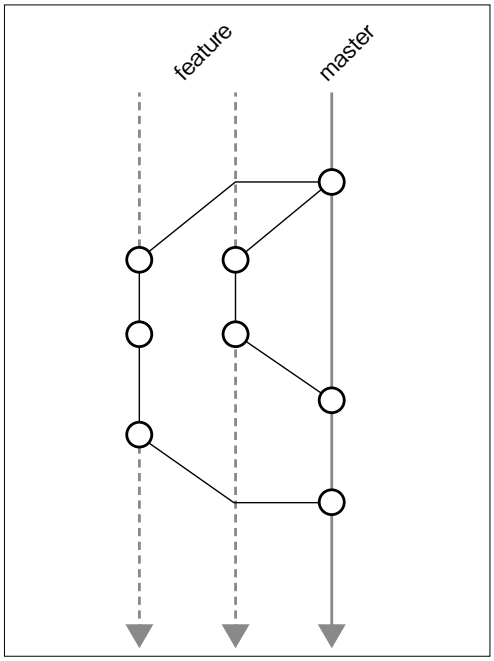
GitHub Flowは、GitHubの開発チームが用いているブランチの運用方法です。Git Flowとは異なり、GitHub Flowでは次の2種類のブランチしか登場しません。

・master
・feature

これらのブランチとそのコミットを図にすると、**図2**のようになります。この図を使いながら、GitHub Flowの流れを見ていきましょう。

GitHub Flowでは、featureブランチはmasterブランチから生成し、Git Flowと同じくそれぞれのfeatureブランチで実装を進めます。そして、masterブランチにfeatureブランチをマージしたら、なるべくすぐさまリリースを行います。これによって、Git Flowのdevelopブランチやreleaseブランチが必要なくなり、シンプルなブランチ運用を実現できます。

▼図2　GitHub Flow



◆ ◆ ◆

なお、実際にブランチを運用していくにあたっては、Git FlowやGitHub Flowなどの手法に無理に合わせるのではなく、チームやプロダクトにとって最適な運用方法を見つけていくことが重要です。

実際Gaiaxでも、Reactioチームは基本的にGitHub Flowを採用していますが、Git Flowを採用しているチームもありますし、これらに当てはまらない独自のブランチ運用ルールを採用しているチームも存在しています。

Git FlowやGitHub Flowを参考にしつつ、チームの文化や状況に合わせて、適切なブランチ運用の方法を考えていきましょう。

Pull Request

第3章で、GitHubからPull Requestを出す方法について解説をしました。ブランチをマージする際、GitHub上でPull Requestを利用して手続きを進めることで、マージ前にほかの開発者にレビューしてもらったり、アドバイスをもらったりすることが簡単にできます。

開発チームのメンバーと協力してプロダクトの開発を進めていくにあたって、Pull Requestを作成する際の「Leave a comment」のテキストボックスに書き込む内容は非常に重要です(**図3**)。

ここにはPull Requestの詳細を記入できますが、ここをしっかり書いておくことで、ほかのチームメンバーがPull Requestをレビューする際、その背景や概要をすぐ理解できるようになります。結果として、レビュアはPull Requestのレビューに、スムーズにとりかかれるようになります。少し手間ではありますが、詳細はしっかり書くようにしましょう。

Reactioチームでは、次のような内容を記入することが多いです。

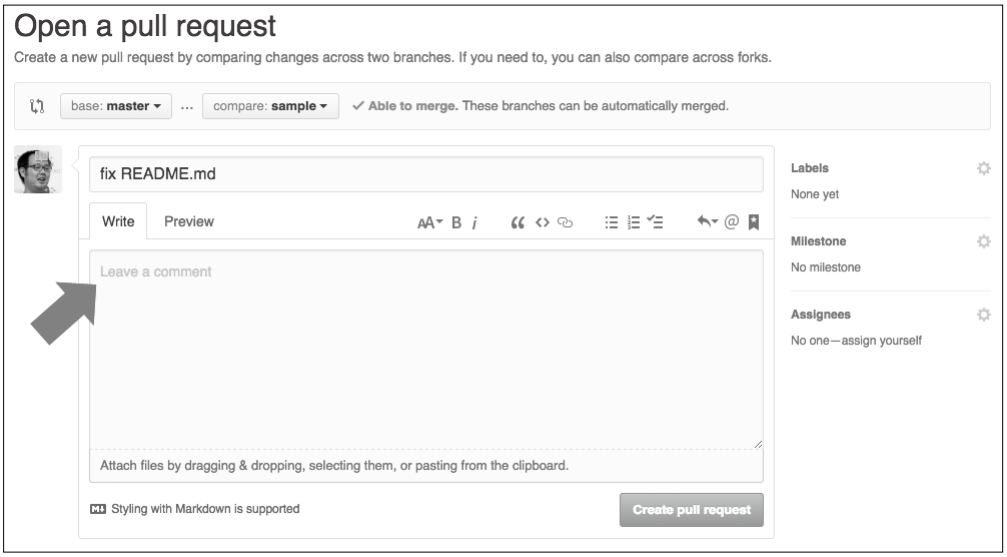
- 作業の目的(なぜその実装を行ったか)
- 作業の詳細(どのような実装を行ったかについての概要)
- とくに見てほしいポイント(実装をしていて悩んだところ、困っているところ)

「作業の目的」については、関連するGitHubのIssueやRedmineのチケットがあれば、そのURLを添付するとさらにわかりやすいでしょう。ちなみに、Reactioチームでは開発する機能やバグの管理は、GitHubのIssueではなく、esa.io[注2]を使ってまとめることが多いです。そのため、esa.ioの該当記事へのリンクを記入することがほとんどです。

さらに、Pull Requestではほかにも「Labels」「Milestone」そして「Assignees」を設定できます。Reactioチームでは、MilestoneについてはGitHub以外のツールで管理しているため、基本的にPull RequestではLabelsとAssigneesのみを活用しています。その具体例を説明していきます。

注2) ドキュメント共有サービス URL https://esa.io/

▼図3 Pull Requestを作る



▼図4　ラベルの例

▼図5　アサインの表示

Labels

Pull RequestとIssueには、リポジトリごとに任意のラベルを定義して、これを割り当てることができます。適切にラベルを割り当てることで、Pull RequestやIssueの性質やステータスをわかりやすく示すことができます。

具体的に、Reactioチームでは次のようなラベルを用意しています。

・作業状況を示すラベル
- 作業中
- レビュー待ち
- リリース待ち

・実装の内容を示すラベル
- 新機能
- バグ
- リファクタ
- 改善

たとえば、あるPull Requestのラベルが**図4**のようになっている場合、このPull Requestは「新機能を実装したPull Requestで、チームメンバーによるレビューが終わって、リリース待ちの状態である」ということがすぐにわかります。

Assignees

Pull RequestとIssueには、複数人のユーザをアサインできます（**図5**）（2016年5月末のアップデートまでは、1つのPull Requestにつき1人のユーザしかアサインできませんでした）。

Reactioチームでは、Pull Requestを出す際、Pull Requestを出すユーザ以外の全員をAssigneesに指定してレビューを依頼するようにしています。そしてレビューが終わったタイミングで、レビュアはAssigneesを、Pull Requestを出したユーザに変更するというルールにしています。

GitHubにこだわらない

せっかくGitHubを導入するのだから、なるべくGitHubが提供する機能だけを使って開発を進めていきたいと思うかもしれません。実際、GitHubにはGitリポジトリのホスティングやPull Requestだけでなく、タスク管理などに活用できる「Issue」、IssueやPull Requestの期限を定めることができる「Milestone」、情報をまとめることができる「Wiki」など、さまざまな機能が用意されています。

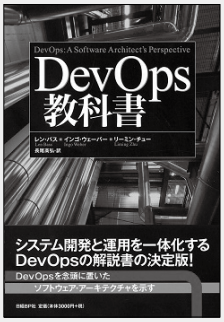
とはいえ、これらの領域はGitHub以外にも特徴のあるOSSやSaaSが多く存在しており、それらの多くはGitHubとの連携にも対応しています。

GitHubを導入するにあたっては、開発のすべてをGitHubだけで完結させようと考えず、適材適所でほかのOSSやSaaSを組み合わせていくことが大事です。

さいごに

GitHubを会社やチームに定着させるためには、ただ単にOrganizationアカウントを用意すれば良い、というわけにはいきません。何よりも大事なのは、会社やチームに「GitHubがある文化」を定着させていくことではないでしょうか。そのためには、GitHubの導入を推進するメンバーが、GitHubの良いところや良い使い方を、会社やチームに率先して示していくことが大事だと思います。

このGitHub特集が、これらを試みる際に、少しでも参考になれば幸いです。SD



DevOps教科書

レン・バス、インゴ・ウェーバー、リーミン・チュー 著／長尾 高弘 訳
A5判／480ページ
3,000円＋税
日経BP
ISBN＝978-4-8222-8544-9

DevOpsと言えば、「開発と運用が連携する」「継続的インテグレーションを実践する」など手段の面から語られることが多いが、本書の中では「高品質を保ちつつ、システムに変更をコミットしてからその変更が通常の本番システムに組み込まれるまでの時間を短縮することを目的とした一連の実践」と目的ベースで定義されている。そしてその目標を達成するためのDevOps的な手法・文化を、どのように導入・定着させるかといった課題を、「クラウド」「ITIL」「デプロイパイプライン」「マイクロサービス」といったキーワードを使って解説していく。1～3部は特定のツールやサービスに依らない内容だが、第4部「ケーススタディ」では実際の企業での具体的なシステム構成やChefのレシピなども紹介され、参考にしやすい。

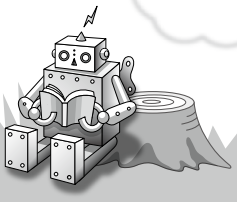
入社1年目からの「ネットインフラ」がわかる本

村上 建夫 著
A5判／368ページ
2,380円＋税
翔泳社
ISBN＝978-4-7981-4609-6

ネットワーク技術の全体像をインターネット、イントラネット、音声系ネットワークに分け、それぞれを構成する技術要素を広く解説している。「入社1年目からの」と書名にあるとおり基礎からの説明が心掛けられているが、ひとつひとつの技術に対して概要にとどまらない解説がされており、読み応えがある。ネットワーク系の書籍では疎かにされがちな音声通信についても、加入者電話網と携帯電話網、そしてIP電話について、それぞれで用いられている技術と問題点が細かく説明されている。

社内ネットワークの運用を任された若手社員にとって大いに役立つ1冊だ。また、今後IoTが普及していく中で、アプリエンジニアといった上流の開発者がネットワーク技術について知っておくためにも、丁度良い本だと感じた。

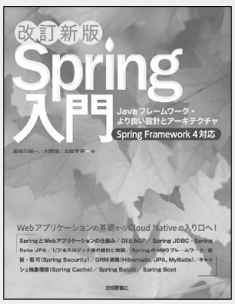
SD BOOK REVIEW



機械学習と深層学習

小高 知宏 著
A5判／232ページ
2,600円＋税
オーム社
ISBN＝978-4-274-21887-3

本書は、2011年に刊行された同じ著者の『はじめての機械学習』の続編で、機械学習をよりわかりやすくするためにC言語でのソースによる解説を試みたものである。C言語そのものの説明はほとんどないが、ポインタや構造体などは使わず、配列を使って解説されているので最低限のC言語の知識があれば理解できる。機能的学習、Q学習、遺伝的アルゴリズム、ニューラルネット、深層学習などのプログラムと動作の解説も載っているので、ソースリストに目を通すだけでも理解が深まる。ただ、掲載プログラムのインデントが浅かったり、文字が細長くなってしまって見にくいところがあるのが残念だ。書籍中にサンプルのダウンロードについての記載はないが、出版社の該当ページからダウンロードできるので、用意してから本書に向かうといいだろう。



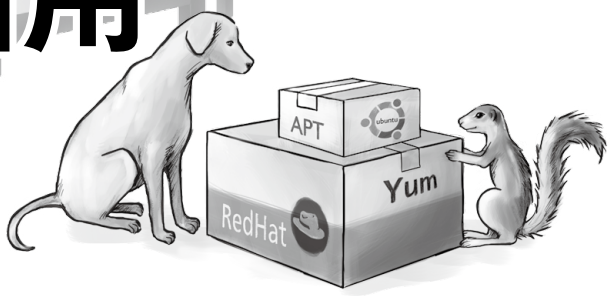
[改訂新版] Spring入門

長谷川 裕一、大野 渉、土岐 孝平 著
B5変形判／432ページ
3,800円＋税
技術評論社
ISBN＝978-4-7741-8217-9

全盛期にたくさんあったJavaのミドルウェア。生き残ったのはJBossとSpring Frameworkだけになってしまった。SpringはVMwareから今話題のPivotalに買収され、同社のPaaSであるCloudFoundryとともに重要な製品としてビジネス展開するようになった。Java 9のリリースに合わせて、今も精力的に開発が進められている。

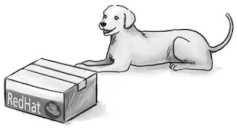
本書は2005年の『Spring入門』をSpring4に合わせて改訂したもの。初版から10年以上が経過しているのだ。しかしWebシステムの進化はクラウドという環境が変わっただけで、基本的な技術は変化していない。本書でもSpring Bootをはじめとして最新技術への対応を行っている。WebとJava、変わらぬコア技術を学ぶために本書を勧めたい。

案外知らなかったYumとAPTのしくみと活用

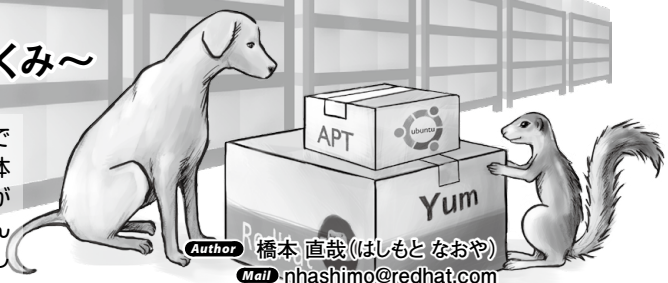


Red Hat Enterprise LinuxやCentOSでは「Yum」、DebianやUbuntuなどでは「APT」によって容易にパッケージ管理ができます。簡便にアプリケーションやツールがインストールできるので、どんな動作をしているのかを意識せずにお使いの方も多いのではないでしょうか。

本特集では、この2つのパッケージ管理システムの背景やしくみ、そしてより実践的な管理方法について紹介します。



序章 パッケージ導入の知識
～開発の背景とRPMパッケージのしくみ～



パッケージ管理ツールが何をやっているかご存じでしょうか。YumやAPTの各論に行く前に、本章で全体像をつかんでおきましょう。パッケージ管理ツールができるに至った経緯と、そのしくみを理解して、なんとなく使っていたツールをしっかりと自分のものにしましょう。

Author 橋本 直哉（はしもと なおや）
Mail nhashimo@redhat.com
URL http://blog.hashnao.info/
レッドハット株式会社

はじめに

Linuxディストリビューションにおけるパッケージ管理と言えば、現在ではDebian系ディストリビューションにおけるAPT（Advanced Package Tool）[注1]やRed Hat系ディストリビューションにおけるYum（Yellowdog Updater, Modified）[注2]が主流です。それぞれ低水準のツールdpkg（Debian Package）[注3]やrpm（RPM Package Manager）[注4]をベースとしています。APTやYumは複雑なパッケージ間の依存関係を解決してくれるため、dpkgやrpmコマンドを用いてパッケージをインストールする機会は非常に少なくなったと言えるでしょう。

この特集では、パッケージ管理の背景やしくみを理解することで、より実践的なパッケージ管理の知識やスキルを習得しやすくすることを目的とします。

- ソフトウェアの配布形態やパッケージ管理システムの歴史から、パッケージ管理のしくみや課題、APTやYumが生まれた背景を理解する
- Yumを正しく使いこなせるようにYumリポジトリのしくみやYumのしくみ、基本的なYumコマンドの使い方を理解する
- rpmとyumコマンドを使い分ける際のポイント、yumコマンドが推奨される理由や状況に応じたさまざまなYumのトラブルシュート例を通して、適切なパッケージ管理方法と実用面を意識した対処方法を習得する

まずはソフトウェアの配布形態の歴史を簡単におさらいしましょう。その次に、本題であるパッケージ管理のしくみを確認していきます。

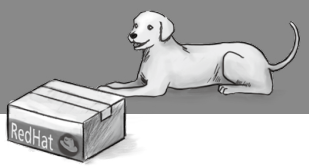
ソフトウェアの配布形態とパッケージ管理システムの歴史

APTやYumなどのパッケージ管理ツールが生まれる前、一般的にUnixや初期のLinuxではソフトウェアのソースコードをtarなどのアーカイブファイルとしてまとめ、配布していました。ソフトウェアのインストールにはソースコードのビルドなどの手動のオペレーションが必要でした。

1991年にLinus Torvalds氏がLinuxをリリースし[注5]、その後、さまざまなLinuxディストリビューションが世の中に広まります。すると多くのユーザはLinux Kernelだけでなく Unixユーティリティやテキストエディタ、Apache httpdなど、目的に応じて必要なソフトウェアを選択してインストールするようになりました。その結果、次のような要望や課題が出はじめま

注1）https://wiki.debian.org/Apt
注2）http://yum.baseurl.org
注3）https://wiki.debian.org/dpkg
注4）http://www.rpm.org

注5）https://groups.google.com/forum/#!msg/comp.os.minix/dlNtH7RRrGA/SwRavCzVE7gJ



した。

・ソフトウェアはソースコードで配布されており、ビルドの必要があるので、導入の敷居が高い
・ソースコードのビルドに手間と時間がかかる
・ソフトウェアのバージョン情報やパッチの適用、ソースコードのビルドと導入のプロセスを標準化し、管理を容易にしたい

これらの課題を解決し要望を満足するために、ビルド済みのバイナリパッケージが配布されるようになります。そしてその管理のために、1993年に各Linuxディストリビューションでパッケージ管理ツールの開発が始まり、Debianにおけるdpkgや Red Hat Linuxにおける RPMが開発されることとなります(パッケージ管理システムの歴史に関する詳細は、dpkgの歴史(**表1**)、RPMの歴史(**表2**)とPMS、RPP、PMの特徴と課題(**表3**)にまとめました)。

RPMパッケージ管理の課題とゴール

これまでバイナリパッケージとパッケージ管理ツールが生まれた背景などを見てきましたが、パッケージ管理ツールが目指すゴールとは何でしょうか。ここではRPMの設計上のゴールを詳しく見てみましょう。RPMのサイト(http://rpm.org)[注6]やFedora RPM Guide[注7]では"RPM Design Goals"というタイトルで設計上のゴールが公開されています。これらを総合的に解釈すると、RPMの設計上のゴールは次のように考えられます。

・ソフトウェアの追加や削除を容易かつ安全に管理するため、バイナリやデータファイルなどのファイルの集合体を1つのパッケージとして一元管理できる
・バージョン情報を持ち、インストール済みのパッケージを安全にアップデートできる
・あるソフトウェアが複数のライブラリを必要とする場合、パッケージ間の依存関係を定義でき、依存関係を定義することで誤って特定のソフトウェアを削除することを防げる

注6) http://www.rpm.org/max-rpm/s1-intro-to-rpm-rpm-design-goals.html
注7) https://docs.fedoraproject.org/en-US/Fedora/24/html/System_Administrators_Guide/ch-RPM.html

▼表1　dpkgの歴史

1993年8月	Ian Murdock氏がDebian Gnu/Linuxを開始(Debian Linux Release)※1。Matt Welsh、Carl Streete、Ian Murdock氏がPerlでdpkgを開発※2
1994年	Ian Jackson氏が開発を引き継ぎ、コア部分をC言語に置き換え※3
1995年3月	Debian 0.93R5をリリース。dpkgをベースシステムインストール後のパッケージ管理として利用※4

※1 https://groups.google.com/forum/#!msg/comp.os.linux.development/Md3Modzg5TU/xty88y5OLaMJ
※2 https://anonscm.debian.org/cgit/dpkg/dpkg.git/plain/scripts/perl-dpkg.pl?id=1b80fb16c22db72457d7a456ffbf1f70a8dfc0a5
※3 https://anonscm.debian.org/cgit/dpkg/dpkg.git/plain/main/main.c?id=1b80fb16c22db72457d7a456ffbf1f70a8dfc0a5
※4 https://www.debian.org/doc/manuals/project-history/ch-releases.en.html

▼表2　RPMの歴史

1993年12月	Rik Faith、Doug Hoffman、Kevin Martin氏が開始したBOGUS Linuxにて、パッケージ管理にPackage Management System(PMS)を採用※1
1994年11月	Red Hat Commercial Linux(Red Hat Linuxの前身)をリリース※2。パッケージ管理にRed Hat Software Program Packages(RPP)を採用
1995年5月	Rik Faith、Doug Hoffman氏がRed HatとのRPPとPMSの重要な機能をベースにPackage Manager(PM)を開発
1997年2月	Erik Troan、Marc Ewing氏がRed Hat Linux向けにPerlでRPMを開発※3。RPM 1.0をリリース※4

※1 http://bogus.org
※2 https://groups.google.com/forum/#!topic/comp.os.linux.announce/Optn4pqWFsw
※3 http://rpm5.org/roadmap.php
※4 http://rpm5.org/docs/max-rpm.html

・インストール済みのパッケージの名称、ライセンス、配布元、概要説明などのソフトウェア情報を簡単に確認できる
・RPMパッケージのインストールやアップデートの際、RPMパッケージの妥当性（改ざんの有無）を検証できる
・パッケージが持つファイルの属性情報[注8]をデータベースに記録することで、パッケージのインストール後に、パッケージが所有するファイル情報の整合性を検証できる
・バイナリとソースの２種類のパッケージを生成することができ、バイナリパッケージはコンパイルしたソフトウェアのインストールや実行が可能であること。ソースパッケージはバイナリパッケージを生成するためにソースコードをコンパイルする手順を定義した文書を含めること

ここまではソフトウェア配布形態の歴史や課題と、RPMが生まれた背景やパッケージ管理が目指すゴールを見てきました。それでは、RPMパッケージの構造はどうなっていて、どのような情報がパッケージに含まれるのでしょうか。たとえば、rpmコマンドにオプションを与えて、`rpm -qi <package>`を実行するとパッケージ情報を参照することができますが、この情報はどこに含まれるのでしょうか。ここからはRPMパッケージの構造やパッケージに含まれる情報を見ていきましょう。

まず、RPMパッケージファイル名の構造を見てみましょう[注9]。

```
name-version-release.architecture.rpm
```

RPMパッケージファイルは4つのフィールドで構成されており、ファイルの拡張子は.rpmとなります。各セクションの要素は定義のとおり前から、nameはパッケージの名称、versionはバージョン番号、releaseはバージョンに対応するリリース番号、architectureはパッケージが

注8） グループ、所有者や権限モードなど。

注9） この構造はバイナリRPMパッケージを前提としています。ソースRPMパッケージの場合、ファイル名の末尾が.src.rpmとなります。

▼表3　PMS、RPP、PMの特徴と課題※1

	特徴	課題
PMS	●Bogus Linuxで採用される ●RPMで最も重要な要素の1つであるPristine Sourcesと呼ばれるコンセプトを採用している※2 ●すべてのソフトウェアはオリジナルのソースからビルド、変更点はパッチで管理し、ビルド中にパッチを適用する ●変更個所をパッチとして管理することでリリース管理が容易になり変更点を迅速に参照できる	●検索機能が弱い ●パッケージの検証機能がない ●複数のマシンアーキテクチャに対応していない ●パッケージ管理のためのデータベースのデザインが弱い
RPP	●Red Hat Commercial Linuxで採用される ●多くの特徴をRPMに引き継ぐ ●1つのコマンドでパッケージをインストール／アンインストールできる ●パッケージをインストール／アンインストールする前後でスクリプトを実行できる ●パッケージの検証や検索ができる	●Pristine Sourcesのコンセプトを採用していない ●複数のCPUアーキテクチャに対応していない ●実行可能なファイルをパッケージに含めた際、ソースからビルドできるか保証できない
PM	●商用ではリリースしていない ●前身のPMSやRPPの主要な機能を採用している	●複数のCPUアーキテクチャに対応していない ●パッケージ管理のためのデータベースのデザインが弱い

※1 http://www.rpm.org/max-rpm/s1-intro-to-rpm-package-management-how.html
※2 http://ibiblio.org/pub/historic-linux/distributions/bogus-1.0.1/bogus-1.0.1/notes/Announce

サポートするCPUアーキテクチャとなります。それぞれのフィールドが意味する内容はRPMパッケージファイル名の構造(**表4**)にまとめました。

次にRPMパッケージファイルのフォーマットを見てみましょう。

RPMパッケージファイルは主に4つのセクション(**図1**)で構成されています。各セクションの役割は次のようになります。

- File Identifier：ファイルがRPMパッケージであることを示す識別子
- Signature：RPMパッケージを検証するための電子署名
- Header：パッケージ名称やバージョン番号などの基本情報を含むヘッダ
- Payload：データ、実際にインストールするファイルの集合体

File IdentifierセクションはファイルがRPMパッケージであることを示す識別情報を含んでいます。

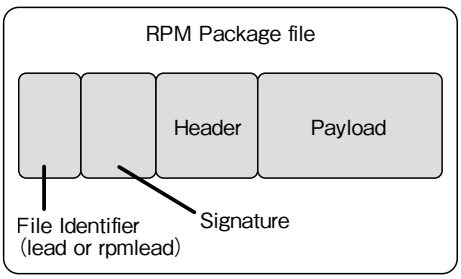
Signatureセクションは取得したファイルが破損していないか、パッケージ自体が第三者により改ざんされていないかを確認するために利用されます。PGP(Pretty Good Privacy)を利用し、公開鍵で署名したことを保証することができます[注10]。

Headerセクションは RPMパッケージを構成する際、パッケージ名称やバージョン番号などのメタデータをタグデータとして保持しています。

Payloadセクションは、RPMパッケージをインストールする際に実際に指定パスに配置され

注10) RPMのPGPによる署名は開発者そのものの信頼性までは保証しません。あくまでRPMパッケージに署名したことを保証するため、ユーザが署名者を信頼するかどうかはユーザ自身の判断によります。

▼図1　RPMパッケージファイルの構造



▼表4　RPMパッケージファイル名の構造

name	●一般的にパッケージ対象のソフトウェア、またはその一部のコンポーネントを差し示すような名称を定義する ●ソフトウェアの名称がパッケージ名に合致している例：systemd(systemd)、NetworkManager(NetworkManager) ※()内がRPMパッケージ名 ●RPMパッケージに含むソフトウェアの名称を示す ●ソフトウェアの名称がパッケージ名に合致していない例：Linux(kernel)、FirewallD(firewalld) ※()内がRPMパッケージ名
version	●RPMパッケージに含むソフトウェアのバージョン番号を示す ●バージョン番号の連番の規則はパッケージに依存する※1 ●例：firewalld-0.3.9、kernel-3.10.0、systemd-219
release	●RPMパッケージに含むソフトウェアバージョンのリリース番号を示す ●リリース番号は該当するバージョンに対し、バグフィックスのパッチを適用した場合などにパッケージ管理者がインクリメントする ●例：firewalld-0.3.9-14.el7、kernel-3.10.0-229、systemd-219-19.el7_2.7
architecture	●RPMパッケージファイルが対応するCPUアーキテクチャやマシンタイプを示す ●例：firewalld-0.3.9-14.el7.noarch、kernel-3.10.0-229.el7.x86_64、systemd-219-19.el7_2.7.x86_64

※1 nameの命名規則が必ずしもソフトウェアの名称と合致していないように、version、releaseやepochいずれもRPMのバージョンを示し、必ずしも含まれているソフトウェアのバージョンと一致しているとは限りません。たとえば、kernel-3.10.0-229 releaseというreleaseバージョンのkernelはupstreamでは存在しません。RPMのversion、release、epochはあくまでもRPMそのものの属性を示し、たまたまupstreamと一致しているケースが多いだけであるとも言えます。場合によっては、glib2やgtk3のようにRPMのnameにversionを含める例もあります。

るバイナリファイルや設定ファイルなどを含みます。各セクションのうち、一般的に最もサイズが大きくなるため、Payloadセクションに含むデータ自体は圧縮されています。

セクションの詳細は**表5**にまとめました。

まとめ

この章では、ソフトウェア配布形態の歴史を追いながら、パッケージ管理が生まれた背景やRPMパッケージが目指すゴールやパッケージの構造を見てきました。

現在でもソフトウェアを配布する際の方法として、たとえばアップストリームでは最新版を迅速に提供するため、従来どおりソースコードをアーカイブファイルに含めてバージョンごとに保持する、ビルド済みバイナリファイルを含むRPMパッケージとして提供する、などがあります。

一方、近年はパッケージをインストールする際、rpmコマンドを使うケースは少なくなり、Yumなどが主流になっています。第1章ではYumにスポットを当て、RPMの課題とYumが生まれた背景、Yumリポジトリのしくみ、yumとrpmコマンドをどのように使い分けるべきか、実践を想定したyumコマンドの使い方やケース別に見るYumのトラブルシュート例を紹介します。また、最後に複雑な依存関係を解決するYumがどのような課題を抱えているのか、そしてYumの後継であるDNFについても紹介します。SD

▼表5　RPMパッケージファイルのセクション

File Identifier	● ファイルがRPMパッケージであることをマジックナンバーを用いて証明する ● fileコマンドでセクションの最初の数バイトを参照し、マジックナンバーのデータベース（/usr/share/magic）と結果の値を比較することで、RPMファイルであることを検証する ● RPMファイルのタイプを認識するためのフラグがあり、パッケージにバイナリやソースパッケージを含むかを定義する ● File Identifierはleadもしくはrpmleadとも呼ばれる
Signature	● File Identifierセクションの次にSignatureセクションが現れる ● Signatureはパッケージの完全性（取得したアーカイブが正当であることを保証）を検証するために利用でき、オプションとして信頼性の検証も含まれる ● GNU Privacy Guard（GnuPG）を用いてRPMパッケージに電子署名ができる
Header	● パッケージ名、バージョン番号、ライセンスなどのメタデータをタグデータとして含む ● Headerセクションはさらに Header record、Header index record structures と Index record structures の3つのパートで構成する※1
Payload	● RPMパッケージをインストールする際に実際にインストールするデータやバイナリファイルなどを含むアーカイブファイルを構成する ● 領域を節約するためPayloadセクションに含むデータはGNU gzipなど※2で圧縮している ● 圧縮したアーカイブデータはcpioフォーマットで構成、cpioフォーマットのため、rpm2cpioとcpioコマンドでファイルを抽出することができる※3 ● Payloadセクションはさらにcpio header、File name、Padding、File Data、Paddingの5つのパートで構成する※4

※1 RPM Headerであることを識別し、Index recordの件数やデータサイズを含むHeader record、パッケージ名やバージョン番号などのタグIDを含むHeader index record structures、これとタグ付けしたデータの実体であるIndex record structuresで構成します。
※2 Fedora 24からPayloadの圧縮データフォーマットをデフォルトでXZ（LZMA）に変更する予定です。
https://fedoraproject.org/wiki/Features/XZRpmPayloads
※3 cpioフォーマットの制約により、RPMパッケージに含む個々のファイルサイズの制限は2GBでしたが、RPM 4.5.9の機能拡張により4GBとなりました。http://www.rpm.org/wiki/Releases/4.5.90
また、RPMパッケージファイル自体の容量制限が2GBから64bit上限まで緩和されたことも大きな変更です。これにより、たとえば2GBを越えるKVMゲストイメージの配布などが可能となりました。
※4 Payloadセクションの主要な構造として、cpio headerはファイルの権限やパーミッション、File Dataは実際にインストールするファイルを含みます。

第1章

Yumを使いこなそう

～構造や動作を知って管理の不安を解消～

本章ではYumに焦点を当て、リポジトリからのパッケージ検索や依存関係の解決がどのように行われているのかを解説します。実践面では、Yumの便利な使い方やRPMの使いどころ、トラブルシュートが役立つはずです。また、Yumの後継ツールと目されているDNFも押さえておきましょう。

Author 橋本-直哉（はしもと なおや）
Mail nhashimo@redhat.com
URL http://blog.hashnao.info/
Author 佐藤 暁（さとう さとる）
Mail ssato@redhat.com
URL https://github.com/ssato/
レッドハット株式会社

RPMの課題とYumが生まれた背景

YumはYellow Dog Linux向けに開発されたYellowdog Updater（YUP）を前身として、Seth Vidal[注1]氏がPythonで実装しました[注2]。Yumの歴史（**表1**）を参照すると2002年から開発が始まり、2003年にFedora Core 1にYumパッケージが含まれました。YUPはMacintosh向けのYellow Dog Linuxディストリビューションだけで動作する一方、YumはRPMベースのシステムを対象とし、RPMパッケージやリポジトリに対する高レベルなインターフェースを提供することで、パッケージのインストールや管理を自動化することを目的としました。

一方、APTの歴史（**表2**）はYumより古く、1997年3月に当時のDebian Release ManagerであるBrian White氏がdpkgのフロントエンドツールとして動作するdselect[注3]をリプレースするためのプロジェクトを発足した後、1999年3年にDebian 2.1に含まれました。

Yum登場以前はユーザは依存するRPMパッケージ群を自分でリポジトリから検索し、手動でダウンロード、インストールする必要がありました。Yumはこの手間を省き、パッケージ間の依存関係の解決や依存パッケージ群のダウンロード・インストールまでの一連の手順を自動化し、パッケージ管理を簡単にすることを目的として開発されました。YumリポジトリでRPMパッケージの集合を一元的に管理、配布することで、大学や企業などで分散したLinuxホストを管理者が容易に管理できるようになりました。

注1）Seth Vidal氏は2013年7月8日にノースカロライナ州ダーラムで交通事故により、享年36歳の若さで亡くなりました。Fedora CommunityやRed Hatは氏の訃報と功績をそれぞれ伝えています。https://fedoraproject.org/wiki/User:Skvidal/Friend、https://www.redhat.com/ja/about/blog/thank-you-seth-vidal

注2）Duke大学の物理学部はパッケージ管理にYUPを利用しており、YUPは依存関係の解決に課題がありました。Seth Vidal氏はDuke大学の物理学部でシステム管理者を担当、YUPに代わるYumの開発をはじめ、Michael Stenner氏はPRMパッケージやheaderファイルを取得するurlgrabberの開発を担当、両名でYumの基礎開発をはじめました。

注3）https://wiki.debian.org/dselect

▼表1　Yumの歴史

2002年6月	Seth Vidal氏がYumのRPMをパッケージング※1
2003年11月	Fedora Core 1をリリース。Yumパッケージが含まれる※2、3
2003年12月	Robert G. Brown氏が“YUM: Yellowdog Updater, Modified”※4を発表
2006年3月	Fedora Core 5をリリース。up2dateは廃止され、Yumの利用を推奨※5

※1 ChangeLogやSpecfile（yum.spec）を参照した結果からの推測となります。
http://yum.baseurl.org/gitweb?p=yum.git;a=blob;f=yum.spec;h=854baf3dd3fd7302a7e6b45a5602a6a1302330d3;hb=HEAD
※2 https://docs.fedoraproject.org/en-US/Fedora_Core/1/html/Release_Notes_for_32-bit_x86_Systems/
※3 http://www.fedorafaq.org/fc1/#InstallSoftware
※4 http://www.phy.duke.edu/~rgb/General/yum_article/yum_article.pdf
※5 https://docs.fedoraproject.org/en-US/Fedora_Core/5/html/Release_Notes/ar01s06s09.html

Yumリポジトリのしくみ

たとえば、yumコマンドでパッケージをインストールする場合、その裏側ではどのようなしくみでリポジトリからパッケージを検索し、依存関係を解決しているのでしょうか。Yumの動作を理解するにあたり、Yumリポジトリのしくみについて**表3**をもとに順番に見ていきましょう。

Yumリポジトリは「RPMパッケージファイル」と「リポジトリメタデータ（以降はメタデータと呼称）」で構成されます。メタデータを作成する方法はおもに次の2通りがあります。

- 上流となるYumリポジトリやインストールメディア内のrepodata/をそのまま参照、もしくはHTTPやFTPミラーにreposyncコマンドを実行する
- createrepoコマンドを実行し、RPMをスキャン、RPMパッケージ情報を抽出する

Yumリポジトリを作成するには、まずreposyncで外部のYumリポジトリを同期、もしくはインストールメディアをマウントするなどしてRPMパッケージを取得します。そして、createrepoコマンドでメタデータを生成します[注4]。リポジトリの作成はreposync(1)やyum.baseurl.orgの“RepoTools[注5]”を参照、メタデータの生成はcreaterepo(8)や同じくyum.baseurl.orgの“RepoCreate[注6]”やFedoraの“Deployment Guide[注7]”を参照するとよいでしょう。

Yumリポジトリは、ftp、nfs、http[s]などで外部に公開できます。

そして、たとえばhttp[s]の場合、Basic認証やSSLクライアント証明書による認証などを利用できます。

クライアントホストは.repoファイル（/etc/yum.repos.d/<repository_name>.repo）に設定したリポジトリの情報を元に、指定したプロトコル経由でリポジトリにアクセスします。

注4） リポジトリメタデータの一部はRPMパッケージファイル内の情報を元に再構成できますが、できないものもいくつかあります。たとえば、comps.xml（RPMグループ定義）やupdateinfo.xml（エラータ情報）が該当します。これらはRPMパッケージファイル内に存在しない情報を含むため、外部から取得しリポジトリに加える必要があります。
注5） http://yum.baseurl.org/wiki/RepoTools
注6） http://yum.baseurl.org/wiki/RepoCreate
注7） https://docs.fedoraproject.org/en-US/Fedora/15/html/Deployment_Guide/sec-Creating_a_Yum_Repository.html

▼表2　Aptの歴史

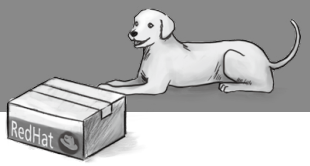
1997年3月	dselect replacement projectが発足※1
1998年3月	ソフトウェアの名称がA Package Tool（APT）に決まる※2。Scott K. Ellis氏がApt 0.0.1をリリース※3
1999年3月	Debian 2.1にAptが含まれる※4

※1 https://lists.debian.org/debian-user/1997/04/msg00786.html
※2 https://lists.debian.org/deity/1998/03/msg00105.html
※3 ChangeLogからの推測となります。
https://github.com/Debian/apt/blob/master/debian/changelog
※4 https://www.debian.org/doc/manuals/project-history/ch-detailed.en.html#s4.1

▼表3　リポジトリを構成するための要素

RPMパッケージ	●クライアントホストに配布するRPMパッケージファイル ●ディストリビューションのインストールメディアをマウントする、reposyncコマンドで外部のYumリポジトリをローカルに同期するなど
リポジトリメタデータ	●メタデータは複数のファイルから構成。フォーマットはXMLとSQLiteがあり、現在のデフォルトはSQLiteとなる※1 ●一連のファイルの属性やパスを定義したインデックス（repomd.xml）、リポジトリに含まれるRPMパッケージの一覧（primary.xml）やRPMパッケージに含まれるファイルの一覧（filelists.xml）など ●createrepoコマンドを利用

※1 createrepoコマンドを実行するとXML（.xml）とSQLite（.sqlite）の双方を生成し、デフォルトでデータベースにSQLiteを利用します。古いバージョンのYumを利用している場合、SQLiteを無視し、XMLを参照します。http://yum.baseurl.org/wiki/RepoCreate



メタデータに含まれる要素

リポジトリ内のrepodata/に含まれるリポジトリメタデータのファイルには、依存関係を解決するための重要な情報が含まれています。それぞれの要素を詳しく見ていきましょう。実際のYumリポジトリに含むメタデータはCentOS Mirror List[注8]などから参照できます。

メタデータに含まれるファイル名や用途を**表4**にまとめました。ここではCentOS 7.2のリポジトリを参考に、それぞれのメタデータに含まれる要素を順番に見ていくことにします。

まず最初に、repomd.xmlを見てみましょう。repomd.xmlファイルはCentOS Mirror Listから理研をたどり、リンク先[注9]を参照してください。

注8） https://www.centos.org/download/mirrors/

注9） http://ftp.riken.jp/Linux/centos/7.2.1511/os/x86_64/repodata/repomd.xml

▼表4　メタデータファイルの概要

repomd.xml	● repomdはそのほかのメタデータファイルを定義するインデックスの役割を持ち、たとえばGPGで署名することでリポジトリの完全性を証明する意味合いを持つ ● クライアントがrepomdをダウンロードすることでメタデータファイルの変更を認識できる ● 要素にファイルパス(location)、タイムスタンプ(timestamp)、チェックサム(checksum)、サイズ(size)などを含む
primary.xml (or .sqlite)	● リポジトリに含まれるRPMパッケージのデータを含む ● 要素にパッケージ名(name)、ファイルパス(location)、バージョン(version)、チェックサム(checksum)やフォーマット(format)などを含む ● フォーマットの要素に、より詳細なRPMの情報(license、vendor、group、provides、requiresなど)を含む
filelists.xml (or .sqlite)	● RPMパッケージに含まれるファイルやディレクトリパスのデータを含む ● primary.xmlに記載されているパッケージ名をfilelists.xmlから参照する際、チェックサム(pkgid)、パッケージ名(name)、アーキテクチャ(arch)やバージョンを元に認識
other.xml	● RPMパッケージのChangelogに関するデータを含む ● パッケージ名の参照はfilelists.xmlと同様
comps.xml	● RPMパッケージを機能的なグループ名で分類 ● リポジトリによってはcomps.xmlを含まない場合もある
updateinfo.xml	● CVE、セキュリティアドバイザリやBugzillaのErrataを含む ● リポジトリによってupdateinfo.xmlを含まない場合もある

▼リスト1　repomd.xml(抜粋)

```
<data type="primary">
  <checksum type="sha256">0e54cd65abd3621a0baf9a963eafb1a0ffd53603226f02aadce59635329bc937
</checksum>
  <open-checksum type="sha256">52f37980031df49b1d185dff22af6de919981fdd50a9805c3e591caa4a3e
c5d8b</open-checksum>
  <location href="repodata/0e54cd65abd3621a0baf9a963eafb1a0ffd53603226f02aadce59635329bc93
7.xml.gz"/>
  <timestamp>1449700524</timestamp>
  <size>2624357</size>
  <open-size>24141104</open-size>
</data>
```

▼表5　repomdの要素

checksum	● locationに定義しているファイルのチェックサム値
open-checksum	● ファイルをgzipで圧縮した場合のチェックサム値
location	● メタデータファイルのパス
timestamp	● メタデータファイルのタイムスタンプ
size	● メタデータファイルのサイズ(Byte表記)
open-size	● メタデータファイル圧縮時のサイズ(Byte表記)

repomd.xmlには<data>タグ内にfilelistsやotherなど、ほかのメタデータファイルの情報が記述されています。Yumはこのファイルを最初に読み込み、ほかのメタデータファイルのURLを解決し、必要に応じてダウンロードします。一例として、primaryセクションを抜粋します(前ページ**リスト1**)。それぞれの要素の名称と概要は前ページの**表5**に記載しました[注10]。

primary.xmlにはRPMパッケージ一覧が含まれ、パッケージ名やバージョン、依存関係などの基本的なパッケージ情報が記述されています。RPMパッケージの依存関係に関する重要な情報はformatタグ内に記述されています。formatタグ内の主な要素を**表6**に記載しました。タグの詳細な定義は、rpm.orgに公開している"Tags: Data Definition[注11]"や"Manual Dependencies[注12]"、Fedora Draft Documentationの"RPM Guide[注13]"に記載されています。

次にfilelists.xmlを見てみると、各RPMパッケージに含まれるファイルやディレクトリパス情報の一覧が記述されています。続いてother.xmlを見てみると、各RPMパッケージの変更履歴の情報が記述されています。続いてcomps.xmlを見てみると、RPMパッケージを機能的なカテゴリなどで分類したグループ定義が記述されています。この情報はインストーラなどでパッケージをグループ単位でインストールする際に利用されます[注14]。パッケージのグループ定義の中の主な要素を**表7**に記載しました[注15]。

次に、updateinfo.xmlを見てみると、CVE[注16]、セキュリティアドバイザリー[注17]やBugzilla[注18]

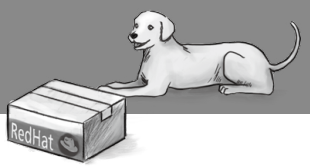
注10) 各メタデータファイルを取得し、shasumコマンドなどでチェックサム値を圧縮／未圧縮の状態でそれぞれ取得するとchecksumやopen-checksumの値と合致するはずです。
注11) http://www.rpm.org/max-rpm/s1-rpm-inside-tags.html
注12) http://www.rpm.org/max-rpm/s1-rpm-depend-manual-dependencies.html
注13) https://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/index.html
注14) yum groupinstallやインストーラを起動後、パッケージグループを選択する際などに参照されます。
注15) comps.xmlの詳細はyum.baseurl.orgの"YumGroups"やFedora Hostedの"Fedora Comps"を一読いただくことをお勧めします。http://yum.baseurl.org/wiki/YumGroups、https://fedorahosted.org/comps/
注16) https://access.redhat.com/security/security-updates/#/cve
注17) https://access.redhat.com/security/security-updates/#/security-advisories
注18) https://bugzilla.redhat.com/

▼表6　primaryに含まれるformatの要素

rpm:provides	●パッケージが提供する仮想パッケージ(virtual package)を定義 ●たとえばクライアントのアプリケーションがどのパッケージに依存するか意識せずに任意の名称を定義したい場合に利用
rpm:requires	●依存するRPMパッケージの名称、バージョンやリリース番号などを定義
rpm:conflicts	●コンフリクトする対象のパッケージを定義 ●同時にインストールすると競合するパッケージを定義する場合などに利用
rpm:obsoletes	●廃止(Obsolete)する対象のパッケージを定義 ●新しいバージョンをインストールする場合やパッケージの名称が変更になった場合などに利用

▼表7　compsに含まれる要素

id	●グループIDを定義
name	●表示するグループ名を定義
description	●グループの説明を定義
default	●パッケージグループを選択する際、デフォルトでグループを有効にするか定義
uservisible	●グループをユーザに表示するか定義
packagelist	●グループに属すパッケージ名を定義 ●type属性に次のいずれかを選択できる ● optional：デフォルトでパッケージは有効にならず、GUIで選択したグループからパッケージの要否を指定できる ● default：デフォルトでパッケージは有効になり、GUIで選択したグループからパッケージの要否を指定できる ● mandatory：グループを選択すると必ず含まれる ● conditional：依存するパッケージをインストールする場合に含まれる



に関するバグ情報などが含まれていることがわかります。Yum security plugin[注19]はセキュリティアップデートやエラータの表示、また該当するパッケージのアップデート時にこのファイルの情報を利用しています。

Yumのしくみとプラグインによる機能拡張

Yumコマンドの流れ

これまでYumリポジトリのしくみやメタデータファイルの内容などを見てきました。次に、Yumのしくみや設定ファイルの記述方法を確認していきます。

まずyumコマンドを実行する際の流れを簡単に追いながら、そのしくみを見ていきましょう。ここでは、yum listとyum installを実行する例をもとにYumの動作を解説します[注20]。

例1. すべての利用可能（available）なRPMとインストール済みRPMの一覧を表示：yum list

①/etc/yum下の設定ファイルと/etc/yum.repos.d/内のYumリポジトリ定義ファイルを読み込み、初期化
②/var/lib/rpm/内のRPMデータベースファイルを読み込み
③有効なYumリポジトリについて順番にリポジトリメタデータのキャッシュファイルを読み込み（もしキャッシュファイルが古ければリポジトリからメタデータをダウンロード、キャッシュファイルとして保存し読み込み）
④読み込んだRPMデータベースとリポジトリメタデータを参照、インストール済みRPMと利用可能なRPMを計算してそれぞれの一覧を出力

例2. RPMをインストール：yum install <package>

①～③はyum listを実行する場合と同様
④リポジトリメタデータを参照し、インストールするRPMを特定（バージョン指定がなければ利用可能な中の最新バージョンとなる）
⑤リポジトリメタデータからインストールするRPMが依存するRPMを順番に解決し、特定
⑥依存するものも含めてインストールするRPMをすべてダウンロードし、インストール

Yumプラグイン

次にYumプラグインの特徴を見てみましょう。

Yumはプラグインのしくみを実装しており、機能を拡張できます。たとえば、yum-plugin-security[注21]やyum-plugin-downloadonly[注22]などのプラグインがあります。プラグインはyumコマンドのオプション（--noplugins）やyum.confの設定パラメータ（noplugins）で有効、無効化できます[注23]。

Yumの内部構造について、デザインパターンにSingletonパターンを採用しており、Yum PluginsはPythonモジュール（.py）としてロードされ、importフックを用いて呼び出されます[注24]。

Yumの設定ファイル

続いてYumの設定ファイル（yum.conf）とYumリポジトリの設定ファイル（.repo）を見てみましょう。まず、設定ファイルは/etc/yum.conf、/etc/yum.repos.d/<repository_name>.repoの順に参照します。yum.confの[repository]セク

注19）RHEL7はyumパッケージに含まれ、RHEL6はyum-plugin-security、RHEL5はyum-securityパッケージの名称です。https://access.redhat.com/solutions/10021
注20）ソースコードの概要はyum.baseulr.orgの"YumCode Snippets"に公開しています。http://yum.baseurl.org/wiki/YumCodeSnippets
注21）https://access.redhat.com/solutions/10021
注22）https://access.redhat.com/solutions/10154
注23）Yumプラグインの利用方法に関する詳細は"RHEL 7システム管理者ガイド - 5.6. YUMのプラグイン"に公開しています。http://red.ht/1UdK6Oi
注24）Yumプラグインの概要はyum.baseulr.orgの"Writing YumPlugins"に公開しています。http://yum.baseurl.org/wiki/WritingYumPlugins

ションにリポジトリの設定を含めることもできますが、個々のリポジトリ固有の設定は/etc/yum.repos.d/ディレクトリ配下の.repoファイルに記載します[注25]。

詳細はyum.conf(5)やFedoraの“System Administration Guide”を参考にするとよいでしょう[注26][注27]。

YumとRPMのユースケース

それでは、実際にyumコマンドを利用するにあたり、サブコマンドやオプションとrpmコマンドを実行するケースを見てみましょう。Yumのユースケースとして、知っていると便利なサブコマンドとオプションを解説します[注28]。

■ 利用可能なパッケージを対象にパッケージ名に‘yum-plugin*’が含まれる一覧を取得

```
yum list available 'yum-plugin*'
```

完全なパッケージ名がわからない、もしくは類似パッケージも含めて検索する場合に利用します。listサブコマンドのオプションとしてavailableを指定しているので、利用可能なパッケージのみが検索対象となります[注29]。

■ 指定したファイルを含むパッケージと該当するリポジトリを表示

```
yum provides '*bin/yum'
```

おおまかにわかっているファイル名やパスから、それを提供するパッケージを探す場合に利用します。providesをinstallに置き換えて、そのままそのパッケージをインストールすることもできます。

■ パッケージのURLを指定しインストール

```
yum install <URL>
```

URLを指定することで依存するパッケージを含めて直接リモートのパッケージをインストールする場合に利用します。

■ 有効にしているリポジトリからリポジトリのメタデータを取得しキャッシュ

```
yum makecache
```

リポジトリのメタデータをキャッシュして、後でオフライン環境で問い合わせを行う場合などに利用します。

■ 指定のリポジトリのみを参照し、パッケージをインストール

```
--disablerepo='<repoidglob>' ➡
--enablerepo='<repoidglob>'
例) yum --disablerepo='*' ➡
    --enablerepo='dvd' install <package>
```

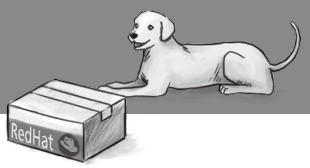
--enablerepoオプションで指定したリポジトリ（たとえばインストールメディアをマウントしたDVDメディア）だけを参照させたい場合に利用します。<repoidglob>にはリポジトリIDもしくはリポジトリ名を指定します。

■ オフライン環境で問い合せ

```
--cacheonly
例) yum --cacheonly list <package>
```

オフライン環境で、リポジトリのメタデータのローカルのキャッシュのみを参照して問い合わせを行う場合に利用します。事前に前述のyum makecacheコマンドを実行し、キャッシュを更

注25) Fedoraの場合、epel-releaseパッケージにepel.repoが、RHELの場合、subscription-managerにredhat.repoがといったようにリポジトリ設定ファイルが含まれる場合もあります。
注26) https://docs.fedoraproject.org/en-US/Fedora/18/html/System_Administrators_Guide/sec-Configuring_Yum_and_Yum_Repositories.html
注27) proxyを設定する場合、RHELはsubscription-managerを利用します。http://red.ht/1S2cG0q。FedoraやCentOSの場合は、yum.conf(5)を参考にするとよいでしょう。
注28) Yumコマンドのチートシートも参考になります。https://access.redhat.com/ja/articles/1354533
注29) 検索対象を絞るほかのオプションについてはyum(8)のLIST OPTIONS節を参照ください。



新しておく必要があります。

rpmコマンドを実行するケース

次に、yumでなくrpmコマンドを実行するケースを見てみましょう。rpmコマンドを利用する典型的な例として、Yumでは行えない複雑な問い合わせを行う場合などが該当します。

クエリフォーマットで複雑な書式を指定して問い合せ結果を出力

```
rpm -qa --qf '%{n},%{e},%{v},%{r},
%{arch},%{buildhost}¥n'
```

出力結果を整形し、標準では出力されないepochなどの情報を取得しています。

インストール済みパッケージに含まれるファイルの一覧を取得

```
rpm -ql yum
```

ファイルのパスからインストール済みのRPMパッケージ名を取得

```
rpm -qf $(which yum)
```

Yumを推奨する理由

yumでなくrpmコマンドを利用するケースをいくつか見てきましたが、Yumが導入されてからRPMパッケージ管理にrpmコマンドを利用することは推奨されていません。Fedoraの“System Administrator's Guide[注30]”に推奨しない理由を公開しています。この文書を総合的に解釈すると、おもに次の理由であると考えられます。

- RPMは低レベルで処理を実行するため、パッケージの依存関係やシステムの整合性を考慮せず強制的に削除するといった危険な操作を実行できてしまい安全ではない[注31]。一方、Yumは複雑なシステムの整合性を記録し、パッケージをインストールやアンインストールする際、強制的にシステムの整合性をチェックすることができる
- たとえばローカルにあるRPMパッケージをインストールする際、yumとrpmどちらのコマンドでも対象のパッケージをインストールすることはできるが、yumは依存関係を解決する一方、rpmは解決しない[注32]
- RPMパッケージをインストールやアップグレードする際、rpmコマンドに--aidオプション[注33]を使い、RPMトランザクションに指定したパッケージを加えることである程度は依存関係を解決するが、yumと同じレベルでは解決しない[注34]
- “rpmコマンドを実行するケース”で提示した例などを除く大半の場合はyumでカバーでき、rpmコマンドを実行する必要性がない

Yumのトラブルシュート

今までに、パッケージをYumでインストールする際、依存関係を解消できずにインストールに失敗し、rpmコマンドでパッケージを強制的に削除した結果、今度は削除したパッケージの依存関係を解決できないといった事象に直面した方もいるかもしれません。次にyumコマンドを利用する際のトラブルシュートについても状況別に順に確認していきましょう。

なお、RHELを利用している場合、まずサポー

注30) https://docs.fedoraproject.org/en-US/Fedora/21/html/System_Administrators_Guide/ch-RPM.html

注31) http://red.ht/1UdNj0l
注32) yumリポジトリ上にあるRPMパッケージのURLを指定し、rpmコマンドでインストールする場合も同じことが当てはまります。
注33) http://linux.die.net/man/8/rpm
注34) RPM 4.9.0からaidオプションは廃止されています。http://www.rpm.org/wiki/Releases/4.9.0

トに問い合わせてみるという観点も必要です。

"No package <package> available"が出力され、パッケージがインストールできない

(a) yum list available '*package_name*' を実行し、パッケージが利用可能どうか確認
(b) yum repolistを実行、実際に有効なリポジトリの一覧を出力
(c) .repoに定義したYumリポジトリが有効(enabled=1)になっているか

(a)の確認には、Yumリポジトリの設定ファイルを生成するオンラインツールも参考になるかもしれません[注35]。

リポジトリの定義に問題はないが、何らかの理由でパッケージを取得できない

(a) リポジトリにアクセスができるかどうか
(b) キャッシュを無効化、最新のキャッシュを取得
(c) 複数のリポジトリを有効にしており、リポジトリ間で依存関係の競合が発生していないかどうか

まずはクライアントからリポジトリにhttp[s]でアクセスができるか確認します(a)。次にyum clean allでキャッシュをクリアし、yum make cacheでキャッシュを取得します(b)。キャッシュクリアで解決できない場合、特定のリポジトリを順番に有効にした状態でパッケージの取得に問題がないか確認します[注36](yum repolist --disablerepo='*' --enablerepo='7-server-extras-rpms')(c)。

パッケージの依存関係を解決できず、特定のパッケージをインストールできない

(a) リポジトリにパッケージや該当するバージョンのパッケージが存在するか
(b) package-cleanupコマンドを実行し、依存関係を解消
(c) 依存関係に競合しているパッケージを除外

有効にしているリポジトリに要求するパッケージ自体がない、もしくは該当するバージョンのパッケージが存在しないなどの可能性が考えられます(a)。あるいはトランザクションが失敗、パッケージの作りが悪いなど、何らかの理由が想定されるため、たとえば次のオプションを利用して、依存関係を解決できるか確認します(b)。

--orphans：有効なリポジトリから有効でないインストール済みのパッケージを出力
--problems：ローカルのrpmdbから依存関係の問題を出力
--cleandupes：ローカルのrpmdbから重複しているパッケージの有無をスキャン、古いバージョンを削除

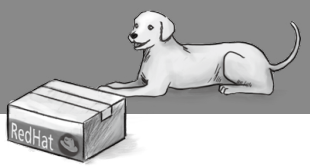
yum.confや.repoファイルの除外リスト(exclude=<package_name>)に記載しているパッケージがアップデートもしくはインストールされても問題がないという前提のもと、指定したリポジトリの除外を無視し、インストールすることもできます(yum --disableexcludes='<repository_id>' install <package_name>)(c)。

Yumのトランザクションが正常に完了せず、トランザクションが中止もしくは失敗

(a) yum-complete-transactionを実行し、中断したトランザクションを解消

突然の電源断などでyumの実行を強制的に中断させた場合は(a)によって中断されたトランザクションをyumに再度実行させます。

注35) https://access.redhat.com/labs/yumrepoconfighelper/
注36) 特定のリポジトリをどうしても利用したい場合、.repoファイルにpriority=<num>を記載し、リポジトリの優先度を設定することもできます。詳細はCentOSの"yum-plugin-priorities"やyum.conf(5)を参照ください。https://wiki.centos.org/PackageManagement/Yum/Priorities



Mirror listを定義しているが、パッケージのダウンロードが速くならない

(a) Mirror Listに定義したリポジトリのロケーションに問題がないか

リポジトリの参照先が物理的に遠い場合などが考えられるため、Mirror Listの設定先を確認します。

yum-plugin-fastestmirrorを有効にしたが、パッケージのダウンロードが速くならない

(a) yum-plugin-fastestmirrorを無効にする

CDNとの相性が悪い場合があるため、プラグインを無効にした後、問題が解決するか確認します(`yum --disableplugin='fastestmirror'`)。

プラグインのコンフリクトやプラグイン自体に問題がある

前述で説明した方法で該当するプラグインを無効にします。

Yum Cacheに何らかの不具合が起きている

(a) yum clean expire-cacheを実行

(a)を実行することで過去に取得したメタデータの記録を削除し、ローカルのYum Cacheを再検証します。expire-cacheで解決しない場合、yum clean allですべてのキャッシュをクリアすることで解決するかもしれません。

Yumの後継「DNF」

長くパッケージ管理ツールとして実績のあるYumもいくつか課題をかかえています。たとえばYumはRPMパッケージの依存関係を解決するため、個々の依存先を順番にたどり解決しますが、この処理のコードは非常に複雑で見通しが悪くなってしまっています[注37]。また、大部分がやや古いスタイルのPython 2のコードで書かれているため、Python 3への移行が非常に困難です。そしてpluginや内部APIなど、一部は文書化されています[注38]が、残念ながら全体として完全なものとはなっていません。

Yumのこれらの問題点を改善するために過去にいくつか試みがなされてきました[注39]が、ついに課題を解決し、Yumを置き換えつつある[注40]のがDNFです。DNFはユーザ向けにできるかぎりYumとの互換性を残しつつ、コードは抜本的に見直して一部のYum由来のコードを除きほぼ一から開発されています。

DNFはYumと異なりCLIとは直接関連しない処理などをほかのライブラリに移譲しています。

- RPMパッケージの依存関係はhawkey[注41]経由でlibsolv[注42]が解決
- RPMパッケージファイルやリポジトリメタデータはlibrepo[注43]がダウンロードし、処理
- Compsデータ(RPM Group定義データ)はlibcomps[注44]が処理

中でも一番特徴的なのはlibsolvによるパッケージ間の依存関係の解決でしょう。libsolvはもともとOpenSUSEのRPMパッケージ管理ツールzypperで、Yumと同様の問題の解決のために開発されたSATソルバベースのライブラリです[注45]。libsolvはRPMだけでなくdebなどの

注37) 該当コード部分はhttp://yum.baseurl.org/gitweb?p=yum.git;a=blob;f=yum/depsolve.py;hb=HEAD#l870 などで非常に複雑で入り組んだものとなっています。
注38) http://yum.baseurl.org/wiki/5MinuteExamples や http://yum.baseurl.org/wiki/WritingYumPluginsなど。
注39) https://people.freedesktop.org/~hughsient/zif/など。
注40) Fedora 22から標準パッケージ管理ツールとしてyumを置き換え：https://fedoraproject.org/wiki/Releases/22/ChangeSet#Replace_Yum_With_DNF
注41) https://github.com/rpm-software-management/hawkey/
注42) https://github.com/openSUSE/libsolv/
注43) https://github.com/rpm-software-management/librepo/
注44) https://github.com/rpm-software-management/libcomps/
注45) https://en.opensuse.org/openSUSE:Libzypp_satsolver、とくにリンク先のFosdem 2008 presentationなどを参照のこと。

ほかのパッケージ形式をサポートし、またrpmmd（Yumが利用しているリポジトリメタデータ形式）だけでなくOpenSUSEやArch Linuxなどで使われているほかのリポジトリメタデータ形式にも対応しています[注46]。libsolvはよく研究され、実績もあるSATソルバアルゴリズムを利用することで依存関係解決の高速化などを実現しています。

またDNFはYumと比較するとコードも非常にわかりやすく[注47]、利用するライブラリなども含めて文書が充実しています。

- DNF：http://dnf.readthedocs.io
- Hawkey：http://hawkey.readthedocs.io
- librepo：http://rpm-software-management.github.io/librepo/

DNFはYumとは少し挙動が違う部分などもあります[注48]が、Fedoraで実績を重ね徐々に成熟が進んでいるため、日常の基本的な用途ではほぼYumと同じ感覚で使うことができます。SD

注46）libsolvはsolvという独自の形式でリポジトリメタデータを扱い、各種形式のリポジトリメタデータは内部的にsolv形式に変換される。
注47）ただし関数型プログラミングのスタイル（map、curry化など）を多用しているので、そのスタイルへの慣れが必要。
注48）主なCLIの違いについては、http://dnf.readthedocs.io/en/latest/cli_vs_yum.htmlにまとまっている。

column ソフトウェアパッケージ管理の未来

Linux Kernelのコンテナ関連機能をうまく活用したDockerやソフトウェアコード履歴管理ツールGitの急速な普及、そしてinitのみならずLinuxシステム管理の基本的で重要な部分を整理統合しつつあるsystemdの浸透などの影響も受け、ソフトウェアのパッケージ形式や管理システムにも旧来の方法を補完する、または置き換えも視野に入れた他の方法が現われはじめています。

NixOS[注A]とその影響を強く受けたOSTree[注B]では、パッケージ操作（インストールなど）の際にRPMなどのように既存ファイルを置き換えません。NixOSはこのしくみを『純粋な関数型パッケージ管理システム』であり、アップグレードなどの操作はロールバック可能で安全だと述べています。

Flatpak[注C]はコンテナ関連機能やD-Bus、OSTreeなどの最新技術を活用し、ランタイムとアプリケーション本体を分離可能にしながら、同時にランタイムの複数バージョンの同居やアプリケーションの安全な実行を実現しています。もともとGNOMEでのアプリケーション配布時の問題解決などを目的に開発され利用されていますが、GNOMEだけでなくKDEでもflatpak形式でのソフトウェアの配布が検討されはじめています[注D]。

SSDS（Server Side Dependency Solving）プロジェクト[注E]ではDNFと同様にhawkey/libsolvを利用し、依存関係解決をYumリポジトリサーバ側で行うという試みを行っています。これによりたとえばDockerイメージなどで、内部では最小限のSSDSクライアントプログラムだけでRPM管理を行うといったことが実現できるようになります。

注A）https://nixos.org
注B）https://ostree.readthedocs.io
注C）http://flatpak.org（http://flatpak.org/developer.htmlにより技術的な詳細な情報）
注D）https://community.kde.org/Flatpak
注E）http://developers.redhat.com/blog/2016/02/18/project-remote-dependency-solving/、https://github.com/rh-lab-q/server-side-dependency-solving/

第2章

APTを使いこなそう ～Debian/Ubuntu編～

APTはDebianやUbuntu、およびそれらの派生ディストリビューションで利用しているパッケージ管理ツールです。本章では、APTが提供するツールやライブラリ、さらにはAPTに対応したリポジトリやバックエンドで動くdpkgツールも含めたパッケージ管理システムとしてのAPTについて説明します。

Author 柴田 充也（しばた みつや）
mail mty:shibata@gmail.com
Ubuntu Japanese Team／株式会社 創夢

Debian/Ubuntuの「要」であるAPT

APT（Advanced Packaging Tool）はパッケージ管理システムとして次の機能を提供します。

・リポジトリからパッケージのダウンロード
・パッケージのインストール・削除
・パッケージ間の依存関係の解決
・リポジトリ上のパッケージの検索・情報表示
・新しいバージョンへの更新
・ディストリビューションのバージョンアップ

DebianやUbuntuではパッケージのフォーマットとして「Debianパッケージ」を採用しています。リポジトリはこのDebianパッケージを保存している場所です。APTでパッケージをインストールするということは、依存関係に従って必要なパッケージを自動的に選別し、リポジトリから取得しインストールするということなのです。

APT登場前後の歴史

序章でも紹介されているように、太古のGNU/Linuxシステムでは自分で必要なソースコードを取得し、コンパイルしてインストールすることが当たり前でした。今でもそのやり方をもっとスマートなしくみのうえで、踏襲しているディストリビューションも存在します。しかしながら皆が皆、豊富な計算資源を持っているわけではありません。各自でコンパイルした結果がほぼ同じでほかのマシンでも流用できるのであれば、ぜひとも流用したいところです。

1994年1月に公開されたDebian 0.91には、パッケージ管理スクリプトとして「dpkgコマンド」を同梱していました（**表1**）。これはDebianの創設者であるIan Murdockが作ったシェルスクリプトです。このdpkgコマンドは、コンパイル済みのバイナリパッケージをインストールや削除するだけでなく、インストール済みのパッケージのリストアップやコンテンツの表示といった、まさしくパッケージ管理システムと言える機能も有していました[注1]。

注1）dpkgのスクリプトには「StopALOPにインスパイアされた」とあります。このStopALOPはWettstein博士が開発したパッケージングシステムで、名前の由来は「脱毛症の阻止」なんだそうです。ソフトウェアのインストールは、当時から毛髪に優しくない作業だったのかもしれませんね。

▼表1　DebianとUbuntuの年表

1994年1月	Debian 0.91リリース
	dpkgコマンドの採用
1994年3月	パッケージングガイドライン作成
1995年10月	Debian 0.93R6リリース
	deslect採用
1996年6月	Debian 1.1 Buzzリリース
	パッケージフォーマットが2.0に
1997年	deslectに変わるdeityの開発開始
1998年4月	aptの最初のリリース
1999年3月	Debian 2.1 Slinkリリース
	aptの正式採用
2004年	Ubuntuの開発開始
2004年10月	Ubuntu 4.10リリース
2014年4月	apt 1.0のリリース

当時のDebianパッケージは拡張子こそ「.deb」ではあるものの、そのフォーマットは現在とまったく異なります。a.out形式のバイナリや/var/adm/dpkg以下のメタデータなどをcpioで固めただけだったのです。dpkgコマンドはcpioを展開してファイルをインストールするスクリプトでした。

0.93系の正式版である0.93R6が1995年の10月にリリースされます[注2]。0.93R6ではdpkgのフロントエンドであるdselectが追加されました。0.93R6のdebファイルはtar.gzファイルの前に独自ヘッダを追加したDebian固有のバイナリに変わりました。また個々のパッケージに責任を持つメンテナー制も導入しています。

1996年にはELFバイナリへの移行を行ったDebian 1.1 Buzzがリリースされました[注3]。パッケージングシステムの有用性を証明するかのようにパッケージの数が474個まで増えています。debファイルも独自形式からar形式となり、メタデータに依存関係などが追加されるなど、より今風なフォーマットになっています。

1999年3月のDebian 2.1 Slinkでは、dselectに代わる新しいパッケージ管理ツールとして「apt」を使用するようになりました[注4]。Aptはもともと「Deity」という名前で1997年あたりから開発が始まっていました。dselectは依存関係の解決をユーザが行う必要があるなど初心者には使いにくいUIでした。その問題を解消するために立ち上がったのがDeityチームなのです。

Deityは直訳すると「神」です。宗教的リスクを伴う名前ということで、改名することになります。いくつかの名前が提案されたようですが、結果的に「APT（A Package Tool）」になりました。そう、Advanced Package Toolではなかったのです。これが1998年4月ごろの話です。

それから18年。APTはDebianだけでなくその派生ディストリビューションでも広く使われるようになりました。APTの利便性に惹かれてDebian系を使っている読者も多いことでしょう。さらにAPTはパッケージ管理用ライブラリも提供しており、より高機能なフロントエンドとしてのaptitudeや、GUI向けのSynapticなども開発されました。aptのリリースから16周年を迎えた2014年の4月には、Deistyチームから「apt 1.0」のリリースがアナウンスされました[注5]。機能カテゴリごとにわかれていたコマンドが「apt」という統一されたインターフェースになるなど、18年たった今も進化を続けています。これからもapt先生のご活躍にご期待ください[注6]。

DebianとUbuntuの関係

UbuntuはDebianの「派生ディストリビューション」の1つです。つまりDebianと同じDebianパッケージフォーマットを使用し、パッケージ管理システムとしてAPT/dpkgを使用し、そのパッケージの多くはDebianと同じものを使用しています。ただし特定のバージョンのDebianとのバイナリ互換性を謳っているわけではありません。DebianとUbuntuの間で同じバイナリパッケージを使えるとは限りませんし、ソースパッケージレベルですら修正が必要な場合もあります。あくまでDebianの成果物を流用した別のディストリビューションなのです。

Ubuntuでは半年ごとの開発期間の最初の段階で、その時点におけるDebianの開発版のリポジトリと同期します（**図1**）。ソースパッケージレベルで差分がないものについては新しいバージョンを自動的にコピーしますし、差分があるものについては手作業でマージ作業を行います。同期したソースパッケージはすべて、Ubuntuの開発インフラ上でバイナリパッケージとしてビルドされます。1つ前のリリース日からおよそ数

注2）https://lists.debian.org/debian-announce/1995/msg00007.html
注3）https://lists.debian.org/debian-announce/1996/msg00021.html
注4）https://lists.debian.org/debian-announce/1999/msg00005.html
注5）https://lists.debian.org/debian-devel/2014/04/msg00013.html
注6）Debianリリースの歴史は「Debian小史」が詳しいです。https://www.debian.org/doc/manuals/project-history/index.ja.html

日程度で、これらの作業を完了しUbuntuの次期開発版の環境を整えています。

こんな短期間でディストリビューション間の同期作業を行えるのは、ひとえにDebianパッケージの品質の高さのおかげです。品質が高く充実したパッケージインフラがあるからこそ、Debianには数多くの派生ディストリビューションが存在するのです。

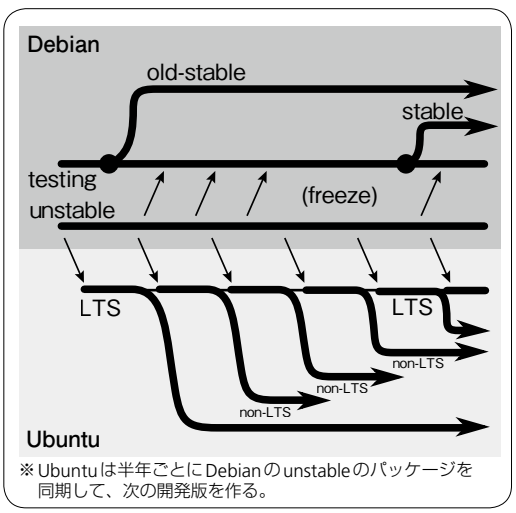
aptコマンドの使い方

本節ではUbuntu 16.04 LTSを元に、パッケージ管理ツールであるaptコマンドやその周辺ツールの使い方について解説します。Debianやより古いUbuntuでは事情が異なる部分も存在しますが、おおよそ同じように使えるはずです。

パッケージの検索とインストール、削除

aptコマンドを使う最大の用途は、パッケージの検索とインストールでしょう。ここでは削除方法と併せて、パッケージのライフサイクルについて説明します。

▼図1　DebianとUbuntuのリリースサイクルの簡易模式図



※Ubuntuは半年ごとにDebianのunstableのパッケージを同期して、次の開発版を作る。

リストの更新（update）

パッケージを検索する前にあらかじめやっておくべきことが、パッケージリストの更新です。/var/lib/apt/lists/以下にあるパッケージリストには登録済みのリポジトリからインストールできるパッケージの情報がリストアップされます。パッケージリストが古いと、リポジトリには存在しないバージョンのパッケージをインストールしようとしたり、アップデートが通知されなかったりします。

更新するには「sudo apt update」を実行してください。Ubuntuではsystemdもしくはcronを用いて、最大で1日に2回程度パッケージリストを更新するようにしています。よってユーザが明示的に更新する必要はほとんどありません。リポジトリを変更したりシステムの構築を自動化する際に、ソフトウェアのインストールを行う前にやっておく、ぐらいのつもりでよいでしょう。

更新に失敗した場合はできるだけすぐにはパッケージをインストールせず、時間を置いてもう一度リストを更新したうえでインストールしてください。もしくは別のミラーリポジトリを試すという方法もあります。

検索と情報表示（search、show）

パッケージを検索する場合は「apt search 単語」を実行します。管理者権限は不要です。検索対象はパッケージ情報のうちパッケージ名とパッケージの説明（Description）になります。単語は複数並べるとAND検索となります。また単語に正規表現を使うこともできます。インストール済みのパッケージは「インストール済み」と表示されます。なおインストール済みのパッケージをリストアップしたい場合は「apt list --installed」と実行します。

パッケージの詳細な情報を表示したい場合は「apt show パッケージ名」を実行します。パッケージ名やバージョン、説明だけでなく依存関係や開発元のURL、パッケージメンテナーなども表

示されます。**図2**をもとに、重要なフィールドをいくつか紹介しましょう。

Package、Version、Descriptionはそれぞれパッケージの名前、バージョン、説明です。バージョンの一般的な書式は次のとおりです。

```
オリジナルバージョン[-Debianリビジョン]
[Ubuntuリビジョン]]
```

オリジナルバージョンはソフトウェア本体のバージョンです。Debianリビジョンは Debian側での修正回数になります。Ubuntuオリジナルのパッケージなら0です。Ubuntuリビジョンは「ubuntu数字」の形で、Ubuntu側の修正回数を示します。Debianリポジトリから同期しただけのパッケージの場合は存在しません。パッケージによっては書式が異なる場合もあります。Descriptionは1行目が短い概要で、2行目以降は詳細な解説です。

Depends、Conflicts、Breaks、Replacesは依存関係を示すフィールドです。Dependsがそのパッケージが依存しているパッケージを示します。Depends以外は競合状態を表現するために使用しており、ユーザが意識することはほぼないでしょう。例以外にも、一般的に同時に利用することが多いパッケージを示すRecommendsや、同時に利用することで便利になるパッケージを示すSuggestsも存在します。なおUbuntuでは、とくに指定しないとRecommendsもインストールします。

Maintainerはパッケージのメンテナーの名前と連絡先を示すフィールドです。Ubuntuは原則としてメンテナー制を採用していないため、Ubuntu開発者のメーリングリストが入っています。Debianのリポジトリから修正せずに同期したパッケージについては、オリジナルのメンテナーが記載されていることもあります。修正した場合は、Debianのメンテナーは Original-Maintainerフィールドに記載されます。

▼図2　パッケージの情報（一部抜粋）

```
$ apt show hello
Package: hello
Version: 2.10-1
Maintainer: Ubuntu Developers <...>
Original-Maintainer: Santiago Vila <...>
Depends: libc6 (>= 2.14)
Conflicts: hello-traditional
Breaks: hello-debhelper (<< 2.9)
Replaces: hello-debhelper (<< 2.9), ...
Description: example package based on...
```

■ インストール（install）

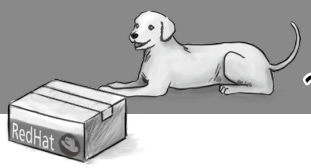
パッケージのインストールは「sudo apt install パッケージ名」です。Depends、Recommendsをもとに同時にインストールするパッケージや使用するストレージ容量を表示しますので、問題なければ「y」（もしくは Enter ）を入力してください。指定したパッケージのみインストールする場合は、とくに問い合わせは行いません。

「-y」オプションをつけると「y」と答えたと判断します。非対話的にインストールしたい場合に便利でしょう。「--no-install-recommends」オプションは、Recommendsをインストールしたくない場合に指定します。容量を節約したい場合に使うとよいでしょう。

インストール時はまずリポジトリから/var/cache/apt/archives/以下にパッケージをダウンロードします。ダウンロードしたパッケージからメタデータを/var/lib/dpkg/info/以下に展開し、メンテナースクリプトであるpreinstの実行、コンテンツの展開、postinstの実行という手順をたどります。/var/lib/dpkg/info/以下にあるメンテナースクリプトにはパッケージのインストール・削除前後に実行するコマンドが記載されています。インストール後にデーモンが起動しないなど、何かトラブルが発生したらそちらも調べてみるとよいでしょう。インストールが成功するとダウンロードしたパッケージファイルは削除されます。

■ パッケージの削除（remove、purge）

パッケージの削除は「sudo apt remove パッケージ名」です。指定したパッケージと、指定し



たパッケージに依存しているパッケージが削除されます。依存関係によって一緒にインストールされたパッケージは削除されませんので注意してください。それらのパッケージを削除したい場合は「sudo apt autoremove パッケージ名」とします。パッケージ名を指定しなかった場合は、自動的にインストールされたものの必要なくなったパッケージを一通り削除します。このとき、パッケージのメンテナースクリプトのprerm、postrmがそれぞれ実行されます。

removeは「設定ファイル」として指定されたファイルは削除しません。設定ファイルとはおもに/etc以下のユーザが変更し得る設定ファイルです[注7]。誤ってパッケージを削除したとしても、再度インストールしたら同じ設定で利用できるように設定ファイルは削除しないのです。もし設定ファイルも含めてすべて削除したい場合は「sudo apt purge パッケージ名」とします。なおソフトウェアが自動的に生成したファイル、とくにホームディレクトリ以下のファイルは、パッケージ管理ツールの管轄外ですので、purgeであっても削除されません。

パッケージの更新（upgrade、full-upgrade）

インストール済みパッケージに対して新しいバージョンが存在する場合、「sudo apt upgrade」でパッケージを更新できます。どのパッケージが更新されるかは「apt list --upgradable」で確認できます。

カーネルの更新時は「新しいカーネルパッケージをインストール」します。これは何かあったときに古いカーネルパッケージに簡単に戻すための措置です。古いカーネルが不要になったら、autoremoveで削除してください。とくに/bootを小さめのパーティションにしている場合、古いカーネルパッケージが残っていると/bootが溢れる可能性があります。

更新に伴いほかのパッケージを削除しなければならないとき、upgradeコマンドはそのパッケージの更新を「保留」します。通常リリースの状態ではまず起こり得ませんが、開発版のDebian/UbuntuやPPAなどサードパーティのリポジトリを使っているなら、それなりの頻度で発生します。保留状態のパッケージを更新したい場合は、「sudo apt full-upgrade」を実行してください。

debファイルのインストール

手動でダウンロードしたdebファイルをインストールしなければならないことがあるかもしれません。比較的新しいバージョンのaptであれば、「sudo apt install ./ファイル名」でインストールできます。パス名であることを明示するためにファイル名はフルパスないし「./」が必要です。この方法なら依存パッケージも一緒にインストールしてくれます。

その他の便利なコマンド

ここまで説明したコマンドを使えば、日常的な作業は一通り行えるはずです。ここではさらにいくつかの便利なコマンドを紹介します。

・パッケージ更新履歴の取得

/usr/share/doc/パッケージ名の下にはパッケージの更新履歴が保存されています。ただし容量の都合上、記録されているのは最新の数バージョンです。もし完全な履歴がほしい場合は「apt changelog パッケージ名」としてください。

・パッケージキャッシュの削除

/var/cache/apt/archivesにはダウンロードしたパッケージファイルが残っています。このキャッシュを削除したい場合は「sudo apt clean」とします。また最新のキャッシュのみは残しておきたい場合、「sudo apt autoclean」も使えます。

注7）設定ファイルのリストは「dpkg-query --control-show パッケージ名 conffiles」で取得できます。

・依存関係の逆引き

あるパッケージが依存しているパッケージはそのパッケージのDependsを見ればわかります。逆にあるパッケージに依存しているパッケージのリストを取得したい場合は「apt rdepends パッケージ名」とします。

・ファイルからパッケージを特定

あるファイルが属するパッケージを探したい場合、「dpkg -S ファイル名」で検索できます。見つからなかった場合は、動的に生成されたファイルか、シンボリックリンクかもしれません。インストールしていないパッケージに属するファイルを検索したい場合はapt-fileコマンドを使用してください。

・ソースコードの取得

あるパッケージのソースコードを取得したい場合は「apt source パッケージ名」を実行します。Debian由来のパッチが適用された状態で展開されますので、アップストリームとの違いを確認する際に便利でしょう。ただしsources.listにdeb-srcタイプのリポジトリを登録している必要があります。

・パッケージ構築に必要なパッケージ

あるパッケージを構築するために必要なパッケージは、そのソフトウェアをビルドする場合に必要になるソフトウェアのリストでもあります。自前でビルドしたい場合は「sudo apt build-dep パッケージ名」とすると、構築に必要なパッケージ一式をインストールできます。

・スーパー牛さんパワー

スーパー牛さんパワーを得たい場合は「apt moo」とします。「moo」の数を増やすとパワーが増大します。

GUIでのパッケージ管理ツール

デスクトップ環境とセットで使っている場合は、GUIからパッケージ管理システムを使えると便利です。APTのフロントエンドとしていくつかのソフトウェアが存在します。最近はGNOMEにおけるGNOME SoftwareやKDEにおけるPlasma Discoverなど、各デスクトップ環境ごとにAppStreamに対応したストアアプリを用意する傾向にあるので、まずは最初からインストールされているものを使ってみるのがよいでしょう。

Ubuntu 16.04 LTSでは、「Ubuntuソフトウェア」というストアアプリが導入されました。これは従来のUbuntuソフトウェアセンターを廃止し、GNOME Softwareに置き換えたものです。GNOME Softwareがソフトウェア名としては

column リポジトリの公開鍵

リポジトリにはパッケージファイルのほかにパッケージリストやパッケージ情報の翻訳といったさまざまなデータが置かれています。aptはどのようにこれらのデータを安全に取得しようとしているのでしょうか。

まずPackagesファイルは「apt show」で表示されるデータの一部やパッケージのハッシュ値などが記録された重要なファイルです。インストール時のパッケージファイルの妥当性はPackagesに記録された情報をもとに検証します。パッケージリスト更新時に最初にダウンロードされるのは、InReleaseと呼ばれるGPG鍵で署名されたファイルです。このInReleaseにはPackagesを含むメタデータファイルのハッシュ値がすべて記録されています。よってInReleaseをGPG鍵で検証し、そこにあるハッシュ値からほかのメタデータファイルの正当性を検証するという手順を踏んでいるのです。

InReleaseを検証するGPG鍵は、最初からOSに含まれています。Ubuntuの場合はubuntu-keyringパッケージがそれです。またapt-keyコマンドで、リポジトリ用の公開鍵を管理できます。

「Software」という一般名詞を採用しており、混乱を避けるために「Ubuntuソフトウェア」という名前に変更しています。ただし開発途上のソフトウェアを導入したので、まだまだ荒削りです。

よりAPTよりのツールとしては、Synapticが存在します(**図3**)。こちらはアプリストアというよりは、APTのGUIフロントエンドという実装です。よってアプリストアでは表示されないような、個別のライブラリパッケージなどもSynapticなら検索できます。

APTの設定

APTの設定は、/etc/apt以下の次のファイルを編集することで行います。

・sources.list、sources.list.d

リポジトリ(パッケージの取得元)を設定するファイルです。詳細は「man sources.list」を参照してください。

・apt.conf、apt.conf.d

APTツール全般の設定が格納されたファイルです。詳細は「man apt.conf」を参照してください。

・preferences、preferences.d

インストールするパッケージのバージョン制御を行うファイルです。詳細は「man apt_preferences」を参照してください。

リポジトリの設定を行うsources.list

sources.listではパッケージファイルの取得元であるリポジトリを設定します。sources.list.dディレクトリには複数の設定ファイルを置くことができます。書き方自体はsources.listと同じです(**図4**)。何かリポジトリを追加したい場合は、できるだけsources.list.d以下に追加し、sources.listはオリジナルのままにしておきましょう。

sources.listの書式は新旧2つの書式が存在します。古い書式は拡張子が.listであり、1行に1リポジトリの情報を記述します。新しい書式は拡張子が.sourceであり、複数行に渡ってキーと値を「:」で区切りながら記述します。世に出回っているDebian系ディストリビューションのほとんどは古い書式で、今後は少しずつ新しい書式に移行していく予定です。今回は古い書式のみを紹介します。

▼図3　Synapticパッケージマネージャー

▼図4　sources.listの書式

```
書式：
type [ option1=value1 option2=value2 ] uri suite [component1] [component2] [...]

例：
# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.
deb http://jp.archive.ubuntu.com/ubuntu/ xenial main restricted universe multiverse
deb-src http://jp.archive.ubuntu.com/ubuntu/ xenial main restricted universe multiverse
```

「#」で始まる行はコメントとして解釈されます。debタイプはバイナリパッケージの、deb-srcはソースパッケージの取得元を記述します。typeの後ろには角括弧でオプションを記述できます。たとえば特定のアーキテクチャの情報のみを取得したい場合にこのオプションを使用します。uriはリポジトリのURIです。httpやhttpsはもちろんのこと、fileやcdrom、sshなどのスキームも使用できます。suiteには多くの場合、ディストリビューションのコードネームが入ります。Debianの最新リリース版ならstableですし、開発版ならsidやunstableになります。Ubuntu 16.04 LTSならxenialになります。特定のディレクトリをリポジトリとして使うなら、URIの下のパス名を指定します。componentはリポジトリによって異なります。DebianやUbuntuの公式リポジトリであれば、ライセンス・サポート状況によって異なるコンポーネントを用意しています。**図4**の例だと「main restricted universe multiverse」はいずれもコンポーネントです。

sources.listを更新した場合は、「sudo apt update」でパッケージリストを再取得してください。

APT全般の挙動を変更するapt.conf

apt.confではAPTの挙動に関する設定を行います。この設定はaptコマンドだけでなくAPTをバックエンドとするすべてのツールに影響します。

▼リスト1　apt.confの書き方

```
// 1行で記述する
APT::Authentication::TrustCDROM "true";

// ブロックで記述する
APT
{
  NeverAutoRemove
  {
        "^firmware-linux.*";
        "^linux-firmware$";
  };
};
```

「//」で始まる行はコメントとして扱われます。また行中の「/* コメント */」も同様です。APTの設定はカテゴリごとのツリー構造となっています。**図5**の2行目の例のようにノード間を「::」で区切って1行で設定することもできますし、**リスト1**の「// ブロックで記述する」以下のように「{}」を用いて複数行に渡ったブロック構造として設定することもできます。

ただし、apt.confの設定がすべてAPTで解釈されるというわけではありません。たとえば50unattended-upgradesにあるUnattended-Upgradeカテゴリは、unattended-upgradeプログラムでのみ解釈します。ちなみにこの設定ファイルはunattended-upgradesパッケージのメンテナースクリプトである/var/lib/dpkg/info/unattended-upgrades.postinstによってインストールされるファイルです。

APTの設定を確認したい場合は、apt-configコマンドを使います。「apt-config dump APT::Install-Recommends」で指定したキーの値を表示します。「apt-config shell VAL APT::Install-Recommends」とするとVAL変数に指定したキーを代入するシェルスクリプトを表示します。後者はスクリプトからAPTの設定を取得したい場合に便利です。具体的な使用例は/usr/lib/apt/apt.systemd.dailyなどが参考になるでしょう。なお、apt-configに設定を変更する機能はありません。

特定のバージョンを固定化できるpreferences

APTは基本的にすべての有効なリポジトリの中から、一番新しいパッケージをインストールしようとします。しかしながら、あるバージョンからアップグレードしたくない場合や、特定のリポジトリからは明示的に指定したパッケージのみインストールしたい場合もあるでしょう。preferencesでは、そのようなバージョンやリポジトリの優先度を設定できます[注8]。

注8）https://help.ubuntu.com/community/PinningHowto

たとえばパッケージfooのバージョンを1.0に固定したい場合は**リスト2**のように設定したファイルを/etc/apt/preferences.d/fooとして作成します。ここでPackageにはパッケージ名を、Pinには適用したいルールを設定しています。今回はバージョン番号で固定ですので「version バージョン番号」としています。「*」でワイルドカードにすることもできます。Pin-Priorityにはルールに合致したときのパッケージの優先度を指定します。大きな数字であれば優先的にインストールされます。一般的なリポジトリにあるパッケージの優先度が500ですので500より大きければ優先してインストールされますし、小さければ明示的に指定しない限りはインストールされません。1000より大きい場合はバージョンのダウングレードを許容します。今回はバージョンが小さくなったとしてもそのまま維持してほしいので1001としています。

「apt policy foo」と実行すると、優先度と実際にインストールされるバージョンが表示されますので設定が正しいかどうか確認しておきましょう。

▼リスト2　preferencesの例

```
Package: foo*
Pin: version 1.0*
Pin-Priority: 1001
```

▼図5　ソフトウェアとアップデート

GUIによる設定

システム設定にある「ソフトウェアとアップデート」は、上記で説明したsources.listやapt.confの一部をGUIから設定できるツールです(**図5**)。リポジトリのミラーの変更や光学ディスクからのインストール、アップデートのタイミングぐらいであればこのツールから設定できます。

たとえばUbuntuの場合、16.04からはセキュリティアップデートに対して自動的に適用するようになりました。自動適用されると困る場合はアップデートタブの「セキュリティアップデートがあるとき」を「ダウンロードを自動的に行う」ぐらいに変更しておきましょう。

APTを使った便利な機能

APTそのものの機能以外にも、APTシステム上で使うと便利なツールやしくみがいくつも存在します。本項ではその一端を紹介します。

PPA

PPA(Personal Package Archive)はLaunchpad上に簡単にパッケージリポジトリを作るためのシステムです。Launchpadのアカウントを持っていて、規約に同意した人であれば誰でもFLOSSなソフトウェアのリポジトリを作ることができます。PPAではDebianパッケージの

書式にしたがって作ったパッケージをアップロードします。するとLaunchpadがバイナリパッケージへとビルドし、署名を行い、アカウントやプロジェクトごとに異なるリポジトリにて公開してくれます。

ユーザはPPAを追加するだけで、公式リポジトリよりも新しいバージョンのパッケージやそもそも公式リポジトリにないパッケージ、テスト版のパッケージなどを試せるということもあって、今日では広く使われるようになりました。ただしPPAはUbuntuマシン上でビルドするため、Ubuntu専用の機能となっています。

必要なPPAを見つけたら、あとはコマンドで追加するだけです(**図6**)。「ソフトウェアとアップデート」でも追加できますが、PPAを必要とするユーザであればコマンドの方が手っ取り早いでしょう。

add-apt-repositoryがやっていることは/etc/apt/sources.list.d/以下にリポジトリを追加し、リポジトリの公開鍵を追加しているだけです。なおPPAは非公式なリポジトリですので、追加する場合は本当に必要かどうか慎重に判断したうえで利用してください[注9]。

32bit版ライブラリのインストール

DebianやUbuntuは多くのパッケージがマルチアーキテクチャ(MultiArch)に対応しています。たとえば64bit版のシステム上でも32bit版のパッケージをインストールして動かすことができるということです。とくにMultiArchに対応したライブラリパッケージであれば、複数のアーキテクチャ用パッケージを同一システム上で共存できます。これはとくに32bit版のバイナリしか提供していなプロプライエタリなソフトウェアを動かしたい場合に便利です。

システムとは異なるアーキテクチャのパッケージをインストールしたい場合は、「package:i386」のようにパッケージ名の後ろにアーキテクチャ名を指定します。

Backportパッケージ

DebianもUbuntuも、一度リリースしたら原則としてパッケージの不具合修正のみを行い、機能を大きく変更するようなバージョンアップは行いません。機能追加よりも安定性を重視することによる措置です。しかしながら、特定のパッケージのみより新しいバージョンを使いたいこともあります。DebianもUbuntuもその用途のために「Backportsリポジトリ」を用意しています。これは開発版も含めてより新しいリリースのリポジトリにあるパッケージを、古いリリースにも提供するしくみです。ただし十分にテストされているわけではないので、扱いには注意してください。

Ubuntuは最初からBackportsリポジトリが有効化されています。Debianの場合は、明示的にリポジトリを追加する必要があります[注10]。原則としてどちらも「コードネーム-backports」がsources.list上のsuiteになります。たとえば16.04(xenial)上でBackportsからfooパッケージをインストールするには、**図7**のように実行します。依存パッケージをどこから取得するかによって書式が若干異なります。

パッケージ設定システムdebconf

Debianパッケージには、インストール時にユーザに設定を求めるパッケージが存在します。

注9) PPAの注意点は次のURLでも紹介しています。 http://gihyo.jp/admin/serial/01/ubuntu-recipe/0419

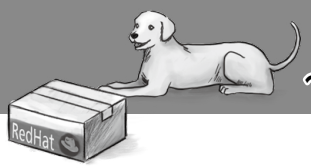
▼図6　PPAの追加方法

```
$ sudo add-apt-repository ppa:ユーザ名/PPA名
$ sudo apt update
```

注10) https://backports.debian.org/Instructions/

▼図7　Backportsからのインストール

```
そのパッケージのみをBackportsから取得する方法
$ sudo apt install foo/xenial-backports

依存パッケージも含めてBackportsから取得する方法
$ sudo apt install -t xenial-backports foo
```



たとえばmysql-serverパッケージは、インストール時にデータベースのrootパスワードの設定を要求します。また、このような設定をインストール後に再度行いたい場合もあります。このようなパッケージごとの設定を管理するシステムがdebconfです。

設定自体は単なるスクリプトとデータファイルの組み合わせです。debconf用の設定ファイルがあるパッケージは、/var/lib/dpkg/info/以下に「パッケージ名.config」という名前で設定スクリプトを保存します。このスクリプトはインストール時に実行されます。インストール後に再度設定したい場合は、「sudo dpkg-reconfigure パッケージ名」と実行してください。

各設定ごとにプライオリティが用意されています。普段は妥当なデフォルト値が存在しないhigh以上のプライオリティのみが表示されますが、mediumないしlowプライオリティの設定も行いたい場合は、「--priority プライオリティ」のオプションを渡してください。medium以下であればデフォルト値が設定されていますので、必要な設定だけ変更して残りはそのままにしておくとよいでしょう。

ローカルリポジトリを作る

dpkg-devのdpkg-scanpackagesコマンドを使うと、debファイルが保存されたディレクトリを簡易的なパッケージリポジトリとして利用できます(**図8**)。複数のバイナリパッケージファイルをAPTから一元的にインストールしたい場合に便利でしょう。

dpkg-scanpackagesは指定したディレクトリにあるバイナリパッケージのリストを作るコマンドです。ローカルリポジトリにおいてリストは圧縮されている前提ですのでgzipに渡しています。なお第一引数はリポジトリURIからの相対パスであることが推奨されています。最後にパッケージリストを更新したら、aptコマンドでローカルパッケージをインストールできます。なお、apt-ftparchiveコマンドを使うとより簡単にリポジトリと同等のメタデータファイル群を生成できます。

リポジトリの簡易ミラーを作る

複数のホストでDebian/Ubuntuを運用しているとき、個々のホストが個別にリポジトリまでパッケージを取得しにいくのは非効率です。とくにアップデート時は同じパッケージを取りにいくわけですので、1台のホストがパッケージを取得したら、残りのホストはローカルネットワーク内部で取得できた方が便利でしょう。そのような目的のためにAPT用のプロキシサーバを構築するツールがapt-cacher-ngです。

導入手順は非常に簡単です。まずプロキシサーバとなるホストにapt-cacher-ngパッケージをインストールします。このホストは初期設定だと3142番ポートで待ち受けることになります。次に、プロキシサーバを利用したいホスト側の/etc/apt/apt.conf.d/以下に適当な名前の設定ファイルを作って、次の内容を記述するだけです。

```
Acquire::http::Proxy "http://プロキシのアドレス:3142/";
```

ちなみに上記のアドレスにWebブラウザからアクセスすると、apt-cacher-ngの管理画面が表示されます。

▼図8　ローカルリポジトリの作成

```
debファイルのあるディレクトリを"/usr/local/debs"とします
$ cd /usr/local/debs
$ dpkg-scanpackages . | gzip -9c > Packages.gz
$ echo "deb file:///usr/local/debs ./" | sudo tee -a /etc/apt/sources.list.d/local.list
$ sudo apt update
```

パッケージ管理システムは、ソフトウェアのインストールに関わる諸々の作業からユーザを解放してくれます。ただし残念ながらトラブルがまったくなくなるわけではないのも事実です。

パッケージリスト更新時のエラー

パッケージリスト更新時「ハッシュサムが適合しません」などのエラーメッセージとともに、更新に失敗することがあります。たとえばリポジトリから取得したファイルがハッシュサムリスト（Releaseファイル）の値と異なるときなどです[注11]。原因はいくつか考えられます。

・リポジトリ内部で不整合が起きている

独自に運用しているリポジトリやミラーリポジトリの場合、コンテンツの更新時ないし上流との同期時にReleaseファイルの生成が正しく行われないことがあります。この場合はリポジトリ側で解決するまで待つか、別のミラーリポジトリを使うしかありません。

・経路の途中に問題がある

経路の途中でどこかで改竄（かいざん）されているというケースもあります。あまりに頻発するようならこちらを疑ってください。プロキシを経由しているのであれば、**リスト3**のようにapt.confでキャッシュさせない設定にすると状況が改善する可能性があります。

「以下のパッケージは認証されていません」

公開鍵が正しく登録されていないリポジトリからパッケージをインストールしようとすると、「以下のパッケージは認証されていません」と警告が表示されます。そのまま警告を無視してインストールすることも可能ではありますが、中間者攻撃によって不正なパッケージに差し変わっている可能性も否めません。安全な方法でリポジトリの公開鍵を取得し、apt-keyコマンドで取り込めば解決します。一般的には、そのリポジトリを運用しているサイトに公開鍵の情報が記載されているはずです。

「未解決の依存関係があります」

dpkgコマンドでdebファイルを直接インストールしたものの依存パッケージはインストールされていない場合、「未解決の依存関係があります」と表示されます。これは依存パッケージのインストールに失敗した場合も同様です。たいていの場合は「sudo apt install -f」を実行することで自動的に解決します。このコマンドを実行しても同様のエラーが出る場合は、端末に表示されるエラーをもとに解決してください。

運がよければ「sudo apt install --reinstall 問題のパッケージ名」で再インストールするだけで解決する場合もあります。

「弱いアルゴリズム」

特定のリポジトリにアクセスしたとき、署名が弱いアルゴリズムを使用している旨の警告が表示されることがあります。より新しいaptコマンドでは、MD5/SHA1といった比較的弱いアルゴリズムを使った署名のサポートを廃止するための措置です。現時点でこの警告は無視できますが、今後も表示されるようであればリポジトリの管理者に対応を依頼してください。SD

▼リスト3　キャッシュしない設定

```
Acquire::http::No-Cache "true";
Acquire::https::No-Cache "true";
Acquire::ftp::No-Cache "true";
```

注11）新しいAPTとリポジトリの組み合わせでは、ハッシュサムごとにURLを変えること（by-hash機能）によりこの問題のほとんどを解消しています。

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第30回 圧縮ファイルあれこれ

圧縮と聞いてLHAとかMAGとか思い出した担当はまさに老害のお手本。老害タッグが生み出す『ひみつのLinux通信』が読めるのは本誌だけ！

to be Continued

日本語のファイル名を含むZIP圧縮ファイルを、Linuxのunzipで展開すると文字化けするんですよね。unzipにこだわらなければ日本語ファイルに対応できるunarが便利ですよ。簡単に調べたらいくつかの最新ディストロではaptやyumでインストールできるみたい。これで窓の人と仕事しても平気だね。最近の若者は圧縮ファイル内の1つのファイルを取り出すためにとりあえず全部展開する人が多いみたい。unzipもtarも、「**unzip 圧縮ファイル名 欲しいファイル**」のように、展開対象圧縮ファイル名の後に取り出したいファイルを指定すればそれだけ「抽出」できるから時間とストレージを大切に使い……なんか老害みたいだね……あっち行きますね。

短期集中連載

乱数を使いこなす

author 力武健次技術士事務所　所長 力武 健次(りきたけ けんじ)
URL http://rikitake.jp/

第1回「コンピュータと乱数」

はじめに

この連載ではシミュレーションやセキュリティ確保に欠かせない乱数に関する技術について紹介します。連載は次の内容で3回を予定しています。

- 第1回「コンピュータと乱数」：乱数、とくにシミュレーション用の疑似乱数についての基礎的事項の説明
- 第2回「物理乱数ハードウェアを作る」：物理乱数を生成するためのハードウェアの製作
- 第3回「物理乱数をOSで使ってみる」：OSでの乱数生成の機能としくみ、そしてそこに第2回で作った物理乱数ハードウェアをどう組み合わせるかについての方法の紹介

本連載を通して、乱数はどう作り、どう使えばいいのかについて親しんでいただければ幸いです。

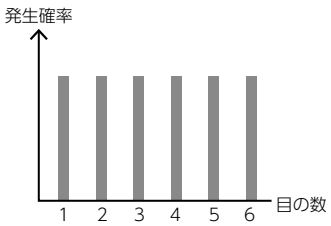
乱数とは何か

乱数とは一言でいえば「規則性のない(ランダムな)数」であり、それを並べたものを「乱数列」と言います。「規則性がない」(あるいは「ランダムな」)という表現にはいろいろな意味が含まれていますが、具体的には次の性質にまとめることができます[注1]。

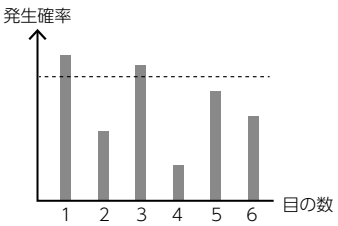
乱数の偏り

乱数列に含まれるそれぞれの数がどのような確率で出てくるかは決まっています。たとえば1から6までの整数がそれぞれの面に書いてある直方体のサイコロを振ったとき、1から6までのそれぞれの数が出てくる確率は、1/6です[注2]。この確率に偏りが生じると、規則性が出てきてしまうため、シミュレーションで誤った結果が出たり、セキュリティ上の脆弱性となることがあります(**図1**)。

▼図1　サイコロに見る乱数の偏り



理想的なサイコロでは、すべての目の発生確率が同じ



しかし現実のサイコロでは、発生確率が目によって変わる。この例では明らかに奇数の目が出やすくなっている。このような偏りは、ゲームでは不正につながるため好ましくない

乱数列の性質

しかし、具体的にどの数がいつ出てくるかは事前に予測することが非常に困難か、予測する方法がないかのどちらかです。この性質を利用して、乱数列を使って第3者に予測されにくい情報を必要とするセキュリ

注1) 本連載ではコンピュータで扱う乱数列を説明しますので、とくに注釈のない場合は、乱数列の各要素は有限かつ離散的な値を取る集合と仮定します。具体的には、0と1の2つの値を取るビットの列や、0から255の整数値を取るバイト列などを想定して話を進めます。

注2) 本連載では問題を簡単にするため、とくに注釈をしない場合は、離散一様分布、つまり乱数として取り得るそれぞれの数の出てくる確率がすべて同じであることを仮定して話を進めます。サイコロを振ったときに出てくる目はこの分布に従っているとみなすことができます。

ティ確保のための秘密情報(パスワードなど)を得ることができます。別の言い方をすれば、乱数列が十分予測困難でない場合、パスワードを攻撃者に見破られてしまうなどの実害が生じることがあります。

物理乱数

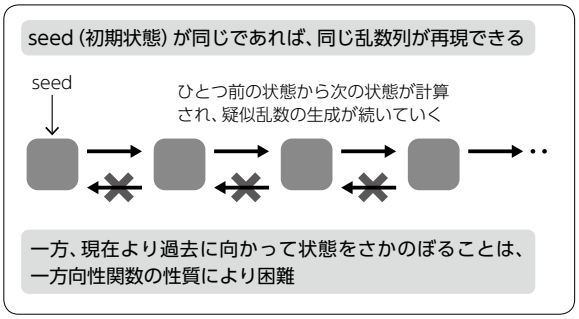
乱数列の中で、物理現象のみによって生成されたもの(物理乱数)については、再現する方法が存在しません。この性質は、秘密を守るうえでセキュリティ確保の目的に役立ちます。

一方向性関数で決まる乱数の性質

一方、コンピュータで一方向性関数[注3]を計算して生成する疑似乱数の場合は、関数に与えたパラメータ(seed[注4])が再現できれば、過去の乱数列そのものの再現ができます。この性質から、シミュレーションの追試や再現を行うために、疑似乱数を使うことができます。また、一方向性関数に高い暗号学的強度を持たせれば、セキュリティ確保のうえでより予測されにくい秘密情報を生成できます(**図2**)。

今回は、シミュレーション用の疑似乱数(以下単に「疑似乱数」)について紹介します。今回扱う疑似乱数は、偏りを可能な限り抑えていますが、暗号学的強度は考慮していません。また、シミュレーション用のため、同一のseedを与えれば同じ乱数列を再現できることを仮定しています。

▼図2　一方向性関数で決まる乱数の性質



疑似乱数の応用分野

疑似乱数を使うと、予測の難しい現実社会の諸要素について、コンピュータでシミュレーションを行えるようになります。よくある例としては、ゲームがどのように進むかを決めるのに疑似乱数を使うという方法があります。そのほかにも次に述べる応用例があります。

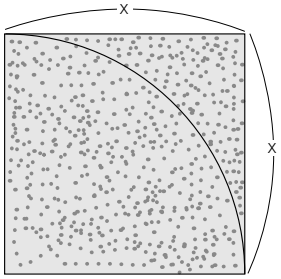
モンテカルロ法

乱数を使って各種パラメータを無作為に変えることで、最適解を見つけていく「モンテカルロ法」は、機械学習などで効率良い処理を行うのに使えます。一例として、ランダムに多数の点を正方形の中に置いて、それが円の中に含まれているかどうかを調べることで、円周率の近似推定ができます(**図3**)。

乱択アルゴリズム

クイックソートやスキップリストなど、リス

▼図3　モンテカルロ法の例



一辺の長さがxである正方形の中に半径xである円の4分の1を含めた場合、両者の面積の比は4:π (円周率) である。仮にこの正方形の中に多数の点を乱数で座標を決めてランダムに取ると、円の中に含まれる点の個数を全体の個数で割れば、結果はπ/4に個数が増えるほど近づいていく。このような乱数によってランダムに問題を解決していく手法を「モンテカルロ法」と呼ぶ

注3) 一方向性関数とは、関数fについてy = f(x)としたときに、x = g(y)となるようなfの逆関数gが存在しないか、あるいは存在したとしても計算量が膨大になり、計算することが困難なもののことを言います。

注4) 疑似乱数を生成するための関数に与えられるパラメータのことをseed(種)といいます。同じアルゴリズムに同じseedを与えれば、同じ疑似乱数列を作り出すことができます。

トの並べ替えや検索に際しランダムな要素を取り入れることで、平均処理時間の高速化を行うアルゴリズムもあります。モンテカルロ法を含め、これらを「乱択アルゴリズム」と呼んでいます。

疑似乱数のD/A変換

雑音をサンプリングしてデジタル値に直すと乱数同様の性質を持ちます。これを逆に応用すれば、疑似乱数をD/A変換することで、デジタル信号処理を使った高速な雑音の生成が可能になります。

属性テスト

プログラムのテストを行う場合、一定の値の範囲や性質を満たしたランダムな値をパラメータに設定して正常な動作をしているかどうかを検証する「属性テスト」という手法があります。商用ツールとしてはErlangでよく使われるQuickCheck[1]があります。

疑似乱数の品質評価と検定

疑似乱数のような「ランダムな数列」を、あらかじめ決められた方法で数学的に判断するというのは自己矛盾を含みます。予測できてしまうものは本物の乱数とは言えないからです。ですから疑似乱数の品質評価は簡単ではありません。しかしながら、乱数が満たすべき数学的性質を計算によって調べることで、偏りなどの既知の問題を見つけることはできます。具体的には次の方法で乱数列の検定[注5]を行います。

統計的に疑似乱数を検証

疑似乱数生成で得られた乱数列に対して平均値や分散などの統計的特徴値を計算します。これらは分布[注6]に固有の値を取ります。乱数列の要素の数が大きくなるほど、そこから出てくる特徴値は理論的な値に近付いていきますから、その収束していく様子を見ることで、乱数かどうかの判断ができます。

疑似乱数の距離分布

一様分布の場合の検定手法としては上記の統計的特徴値だけではなく、より発見的手法で乱数としての性質を維持しているかを調べる手法が多数提案されています。一例として、乱数の値を座標として2次元や3次元の点を多数作り、複数点間の距離の分布を調べるなどの手法があります。

データ圧縮による検定

乱数には規則性がないので、無損失のデータ圧縮によってサイズが変わることはないという特徴があります。現在普及しているgzip、bzip2、xzなどのデータ圧縮ツールを使えば、サイズが変わるかによって乱数の検定ができます。

◆ ◆ ◆

一様分布の乱数の検定を行うためのツールとしては、ent[2]、Diehard[3]、Dieharder[4]、TestU01[5]などが普及しています。また、これらの背景にある理論の説明は参考文献[6][7]を参考にするとよいでしょう。

疑似乱数ライブラリ設計の失敗例

現在普及している言語の大多数はそれぞれ疑似乱数生成を行うためのライブラリを備えています[注7]。しかし、昔からある疑似乱数ライブラリには古いアルゴリズムを使っているものがあ

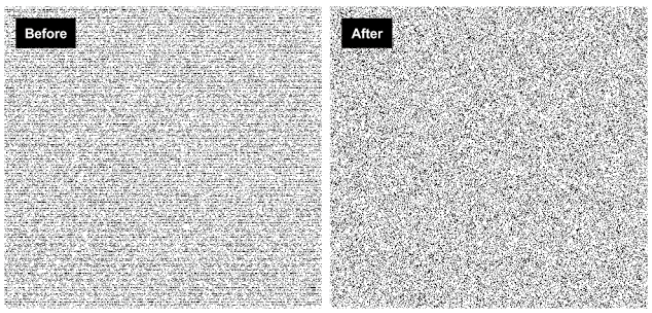
注5) 検定とは、ある数列が何かの法則に従っているかについて、その法則に従っていない確率がある小さな値(有意水準、一般には1%あるいは5%)以下であることを示すことで、結果としてその法則に従っている可能性が十分高いことをテストするための統計的手法です(「検定」というのは統計固有の難しい響きの言葉ですが、「テスト」と読み替えても良いでしょう)。

注6) 分布(あるいは確率分布)とは、数列の中に出てくる値と、それぞれの出現確率との関数のことです(離散値と連続値では定義は違いますが、本稿では詳細は割愛します)。

注7) プログラミング言語に限らず、OSの機能として予測の困難な暗号学的強度の高い疑似乱数を生成するデバイスを用意するのも一般的になりつつあります。たとえばLinuxやFreeBSDでは/dev/randomあるいは/dev/urandomといった名前のデバイスを用意しています(詳細は連載第3回で解説予定)。

り、周期[注8]が短か過ぎて全数探索による攻撃(結果の予測)が可能になったものもあります[注9]。たとえば、Cの標準ライブラリに入っているrand()(周期231)やrandom()(周期2^{35})、rand48()(周期2^{48})といった関数やJavaのクラスjava.util.Random(周期2^{48})は周期が短く、シミュレーション用には適していません。後述するSFMTなどの長周期の疑似乱数を使う必要があります。周期の長い疑似乱数はseedである内部状態の量も多く必要とし、計算が遅くなるという欠点はありますが、大規模シミュレーションでは可能な限り偏りのない疑似乱数が必要であることを考えると、できるだけ周期の長いものを使っておいたほうが無難でしょう。

▼図4　JavaScript V8エンジンで起こった問題の例(参考文献[8])



Randomness visualization before PRNG update (V8 4.7) and after (V8 4.9) (visualization by Mike Malone, http://bl.ocks.org/mmalone/bf59aa2e44c44dde78ac, CC-BY-4.0)

また、検定が不十分な疑似乱数ライブラリを放置していた例もあります。JavaScriptのV8エンジンでは、バージョン4.7のクラスMath.random()で使用していたアルゴリズムMWC1616(周期2^{32})が、2次元の点の座標を描画する検定で一見して目で見てわかる横筋が出てしまうという問題が発見され(**図4**)、バージョン4.9でXorshift128+(周期2^{128}-1)に改められました[8]。同様に2008年にWindows上のPHPバージョン5でも同様の問題が報告されています[9]。

より良い疑似乱数ライブラリを選ぶには

シミュレーション用の乱数は、長い周期であると同時に、高速・大量に使われるため速く生成できることが必要です。その意味で1997年に登場したMersenne Twister(メルセンヌ・ツイスター、MT)[10]は、2^{19937}-1という超長周期を実現し、かつ高速な生成を意識した画期的な乱数生成アルゴリズムでした。MTはRやPythonを始めとする多くのオープンソースの言語やライブラリに採用されています。現在はMTを改良したSFMT[11]が、大規模シミュレーションの分野で広く使われています。SFMTはBSDライセンスを採用しており、多くの言語で使うことができます。

一方、MTやSFMTほどの長周期を必要としない分野でも、より高速で偏りの少ない乱数が開発されています。本稿で紹介したXorshift*/+というアルゴリズムは、2003年に発表されたXorshiftアルゴリズムを元にしています。これはXorshiftのアイデアであったシフト演算とXOR演算だけを使った生成法から、MT/SFMTの作者達が考案したXSadd[12]による改良を経て、TestU01検定のBigCrushテストによって発見された問題項目の数を可能な限り減らしつつ、乱数列の生成速度を向上させるという2つの難しい目標を同時に追求した結果として生まれたものです。現在Xorshift*/+は、さらにビット回転演算を追加しXoroshift+アルゴリズムを加えて実験されています[13]。Xorshift*/+およびXoroshift+はリファレンスコードがCC0(パブリック

注8) アルゴリズムによる疑似乱数では、有限長のseedである限り、乱数列の生成を続けていけばいつかは同じ値が再度現れる可能性があります。同じ値が再度現れるまでの試行回数のことを「周期」といいます。

注9) 筆者はErlangのrandomライブラリで使われているAS183というアルゴリズムの全数検索(周期2^{43})に成功しています(https://github.com/jj1bdx/as183-c)。この全数探索結果の公開後、Erlangではバージョン18.0からXorshift*/+をベースとしたより周期の長い乱数ライブラリのrandモジュールが用意されました。バージョン19.0ではAS183を使ったrandomモジュールはdeprecated(非推奨)の扱いになりました。バージョン20では廃止される予定です。

ドメイン相当)のライセンスで公開されており、自由に使うことができます。

また、SFMTやXorshift*/+といった最近のアルゴリズムでは、特定の回数乱数生成を行ったあとのseedをより高速に求めるためのジャンプ関数が用意されています。このジャンプ関数を使えば、同じアルゴリズムを使っても結果が重複しないように複数のseedを設定できるため、並行処理あるいは並列計算のときに結果の独立性を保証できます(**図5**)。

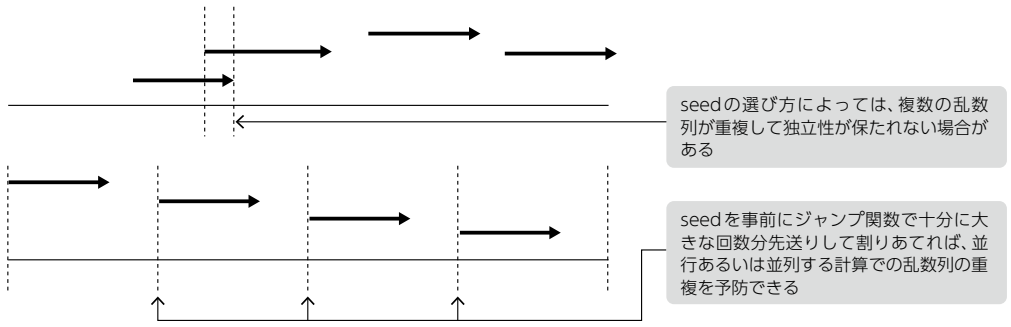
疑似乱数ライブラリの出力で注意すべきこと

疑似乱数生成ライブラリの多くは、既定の動作として一様分布の疑似乱数を生成します。出力は整数あるいは実数(浮動小数点数)のいずれかです。この両者を変換する際は、変換する値の範囲に注意が必要です[注10]。

また、離散一様分布の場合、値の範囲の広い離散乱数列(1からP)から値の範囲の狭い離散乱数列(1からQ)を作る場合は、PがQで割り切れない限りそのまま変換してしまうと確率が均等にならないため、PがQの倍数であることを満たす範囲にある場合のみ変換を行い、余った部分は捨てて使わないという工夫が必要になります(**図6**)。

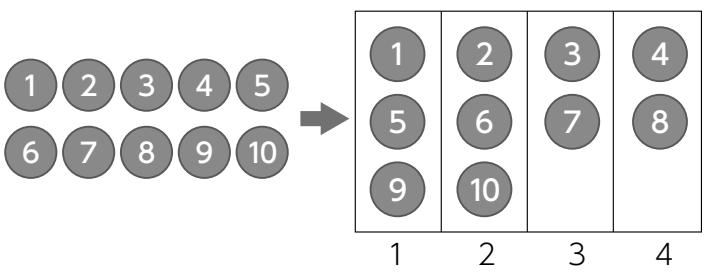
実際の用途では、さらに出力の確率分布を変えることが必要になる場合があります。たとえばゲーム中のアイテムの当選確率をアイテムごとに変えるという例を考えてみます。簡単な方法としては、乱数の出力の取り得る値の範囲を

▼図5　並行処理あるいは並列計算でのseedの選び方



左から右に進んでいく矢印は、内部状態の変化の様子を表しており、横軸の位置が同じだと内部状態も同じで結果の独立性を失うことを示している

▼図6　要素数の違う離散乱数列の変換



1から10までの離散乱数列から、1から4までの離散乱数列を新たに作り出すことを考えてみる。
この場合10は4では割り切れないため、新しい乱数列の各要素の生成確率を偏りなく同じにすることはできない。
左の図のような変換を行う場合は、元の乱数列で9と10が出てきた場合、それらを捨てて、1から8までの要素が出てくるまで再試行を繰り返す必要がある。この方法を使えば、変換結果の偏りをなくすことができる。

注10) 離散乱数列と実数の0から1の乱数列を相互に変換する際、実数の側には4とおりの選択肢(結果Xの範囲が $0.0 \leqq X \leqq 1.0$, $0.0 < X < 1.0$, $0.0 \leqq X < 1.0$, $0.0 < X \leqq 1.0$)があります。このため、実装に応じてすべての可能性を網羅しないと、小さいながら変換誤差のバグが発生し、大規模シミュレーションで利用する場合などの問題となることがあります。

単純に比例分割して大小比較をするという方法があります。しかしこの方法では乱数の発生ごとに比較演算を行わなければならないため、n個に対して最悪でO(n)、二分探索でもO(log (n))の演算が必要です。この作業をより効率的にするために、分割のしかたを工夫して、検索キーの整数部と小数部を分割してO(1)の探索を実現したもの(Walker's Alias Method[14]、**図7**)が実現されています。また、より一般化して各種分割手法を比較した論文もあります[15]。

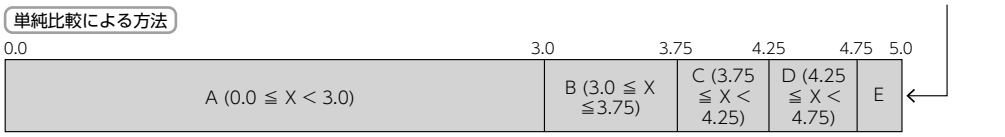
次回予告

今回は乱数のうち、シミュレーション用の偏りの小さい疑似乱数について紹介しました。次回は物理乱数を生成するハードウェアの製作手法について紹介する予定です。SD

▼図7　一様分布の乱数から任意の重みを付けて選択する方法

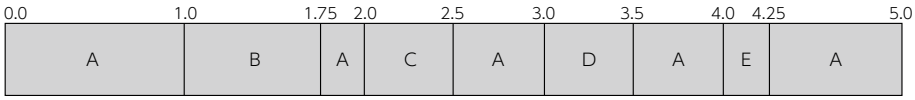
A～Eの5つの選択肢を、実数の疑似乱数X (0.0 ≦ X < 5.0) を使って選ぶことを考える。
一例として出現確率は A:B:C:D:E = 60%:15%:10%:10%:5% = 3.0:0.75:0.5:0.5:0.25 を考える。

単純比較による方法



この場合最大で4回の比較演算が必要である。選択肢n個の場合は最悪O(n)、二分探索でもO(log (n))になる。

Walker's Alias Methodを使った方法



X の整数部 = 0 ならば A
X の整数部 = 1 で X の小数部 < 0.75 なら B, ≧ 0.75 なら A
X の整数部 = 2 で X の小数部 < 0.5 なら C, ≧ 0.5 なら A
X の整数部 = 3 で X の小数部 < 0.5 なら D, ≧ 0.5 なら A
X の整数部 = 4 で X の小数部 < 0.25 なら D, ≧ 0.25 なら A
整数部の値をインデックスとして小数部の閾値をテーブルから求めることで、比較演算は一度で済む。
この場合、閾値のテーブルを作るのはO(n)だが、整数部によるテーブル検索の比較演算はそれぞれO(1)で済む。

参考文献

[1] http://www.quviq.com/products/erlang-quickcheck/
[2] "ENT: A Pseudorandom Number Sequence Test Program",http://www.fourmilab.ch/random/
[3] "The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness", http://stat.fsu.edu/pub/diehard/
[4] Robert G. Brown, "Dieharder: A Random Number Test Suite", https://www.phy.duke.edu/~rgb/General/dieharder.php
[5] Pierre L'Ecuyer, Richard Simard, "TestU01", http://www.iro.umontreal.ca/~simardr/testu01/tu01.html
[6] 平岡和幸、堀玄、『プログラミングのための確率統計』、オーム社、2009、ISBN-13: 978-4-274-06775-4
[7] 伏見正則、『乱数』(UP応用数学選書12)、東京大学出版会、1989、ISBN-10: 4-13-064072-0
[8] "There's Math.random(), and then there's Math.random()", http://v8project.blogspot.jp/2015/12/theres-mathrandom-and-then-theres.html
[9] "PHP rand(0,1) on Windows < OpenSSL rand() on Debian", http://cod.ifies.com/2008/05/php-rand01-on-windows-openssl-rand-on.html
[10] 松本眞、『あなたの使っている乱数、大丈夫?—危ない標準乱数と、メルセンヌ・ツイスター開発秘話—』、http://www.math.sci.hiroshima-u.ac.jp/~m-mat/TEACH/ichimura-sho-koen.pdf
[11] 松本眞、齋藤睦夫、「SIMD-oriented Fast Mersenne Twister (SFMT)」、http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index-jp.html
[12] 松本眞、齋藤睦夫、「XORSHIFT-ADD (XSadd)」、http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/XSADD/index-jp.html
[13] Sebastiano Vigna, "xoroshiro+ / xorshift* / xorshift+ generators and the PRNG shootout", http://xoroshiro.di.unimi.it/
[14] 「重み付きランダム抽出:Walker's Alias Method」、http://qiita.com/ozwk/items/6d62a0717bdc8eac8184
[15] Marsaglia, G., Tsang, W., & Wang, J. (2004). Fast Generation of Discrete Random Variables. Journal of Statistical Software, 11(3), 1 - 11. doi:http://dx.doi.org/10.18637/jss.v011.i03

アプリケーションテストに
時間がかかりすぎてませんか?

Ruby on Railsへの導入でわかった RRRSpecによる分散テストの効果

Author 後藤 優一(ごとう ゆういち)
ピクスタ㈱
Twitter @_yasaichi

はじめに

RRRSpec(トリプルアールスペック)は、クックパッド㈱が開発したオープンソースの分散テスト実行システムで、Rubyのライブラリ(gem)として提供されています。

筆者が所属するピクスタでは、サービスが成長するにつれてRuby on Railsを使ったアプリケーションの自動テストの実行に時間がかかるようになり、日々のリリースが滞ってしまう課題を抱えていました。この課題を解決するためにRRRSpecを導入したところ、テストの実行時間を大幅に短縮することができ、併せてコスト削減を実現できました。しかし、導入にあたって参考となる事例や情報がまだまだ少なく、苦労した点もありました。

そこで本記事では、ピクスタでの導入事例を紹介したうえで、RRRSpecの概要とその有効性、および環境構築や実運用時のポイントを説明したいと思います。

導入の経緯

ピクスタでRRRSpecを導入するに至った背景、導入時の検討事項、導入後の効果を説明します。

背景

ピクスタは、写真・イラスト・動画のデジタル素材をオンライン上で販売するマーケットプレイスサイト「PIXTA[注1]」を開発・運営しています。同サイトは、「本体」と呼ばれるモノリシックなRailsアプリケーションと、そこから切りだされた複数のマイクロサービスで構成されています。そして、これらの自動テストをRubyのテスティングフレームワークの1つであるRSpecを用いて記述しています。

サービスが成長するにつれてアプリケーションの自動テストの規模も大きくなり、現在、「本体」だけで3万を超えるテスト項目が存在します。これをマシン1台で実行すると完了まで数時間かかるので、CI(Continuous Integration)サービスの並列実行機能を利用していましたが、それでもなお70分程度かかっていました。これではリリース可否の判断に遅れが生じます。サービスの機能追加・改善を素早く継続的に行うために、テストの高速化が課題となっていました。

アプローチの検討

テストの高速化にあたって、次の3つのgemの利用を検討しました。

注1) URL https://pixta.jp

・parallel_tests[注2]
・test-queue[注3]
・RRRSpec[注4]

これらはすべてテストを並列実行することで高速化を実現しますが、アプローチがそれぞれ異なります。

parallel_testsは3つの中で最初に開発されたgemで、実行時間が均等になるようにテストファイルを各プロセスに割り振ったあと、テストを並列実行します。初回実行時はファイルサイズをもとに割り振りを行いますが、それ以降は前回の実行時間を記録したログを用います。シンプルで良いアプローチなのですが、実行前にテストファイルの割り振りを行うので、プロセス間で実際の実行時間にどうしても偏りが出てしまうことが欠点です。

test-queueは後発のgemで、いわゆるMaster-Workerパターン[注5]で実装されています。Workerはキューからテストファイルを取り出して順次実行していくので、parallel_testsよりも効率的に並列実行することができます。よって、数十分程度で終了する規模のテストなら、test-queueを導入することで簡単に高速化できます。しかし、マシンのCPUのコア数という制約がある以上、高速化の度合いには一定の限界があります。

RRRSpecはtest-queueと同様にMaster-Workerパターンを採用していますが、Workerとしてリモートのマシンを使い、テストを分散実行する前提で設計されている点が異なります。したがって前述の2つのgemよりも導入に手間がかかりますが、一度導入してしまえばテストの規模に応じて柔軟にスケールアウトすることができます。

また、それ以外にも、

・マシンのシャットダウンやプロセス停止からの自動復帰
・失敗したテストの自動再実行
・テスト実行順序の最適化
・無反応のプロセスに対するタイムアウト

などの機能を備えています。

test-queueにも分散実行のためオプションがありますが、リモートマシンに障害があった場合などが考慮されておらず、分散実行においてはRRRSpecのほうが優れていると感じました。

ピクスタではまず、test-queueを導入して1台のマシンでの高速化を試みましたが、期待する実行時間を達成できませんでした。そこで分散実行によるアプローチを採用することにし、RRRSpecを導入しました。

導入効果

RRRSpecを導入することで、「本体」のテスト実行時間を70分から15分程度まで短縮できました。これによって大幅にリリースサイクルが短縮され、サービスの機能追加・改善を素早く行えるようになりました。またAmazon EC2のスポットインスタンスを利用することで(後述)、CIサービス利用時よりもコストを10%程度削減できました。

RRRSpecの概要

いくつかの用語を定義したあと、RRRSpecのシステムアーキテクチャとgemの構成について説明します。

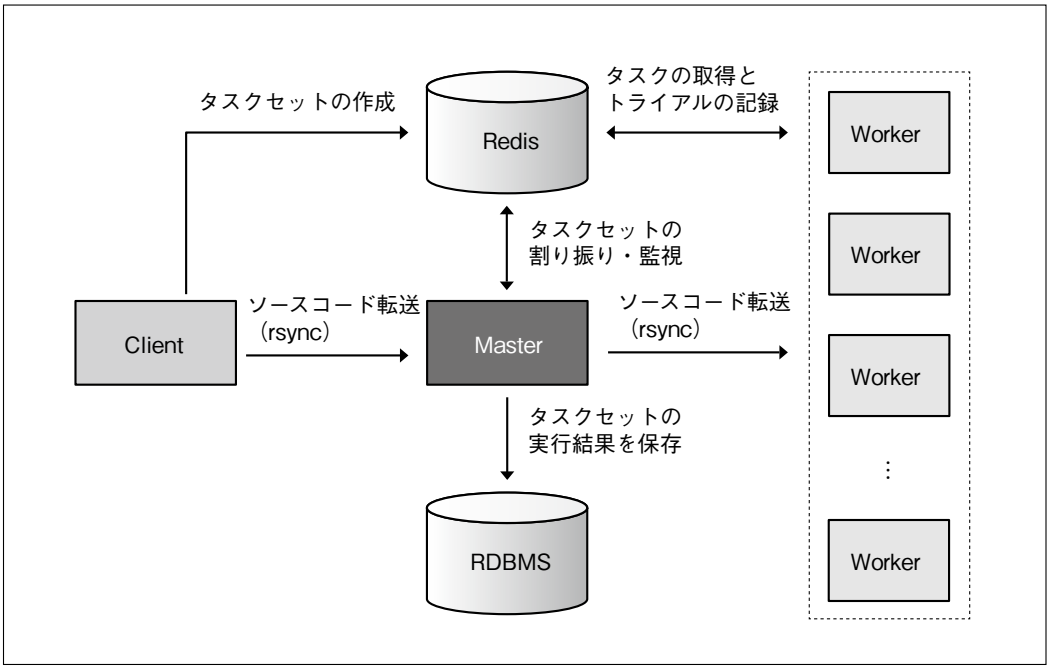
用語の定義

RRRSpecのシステムが取り扱う対象として、次の3つを定義します。

・タスク：1つのテストファイル(＝*_spec.rb)
・タスクセット：タスクの集合で、RRRSpec

注2) URL https://github.com/grosser/parallel_tests
注3) URL https://github.com/tmm1/test-queue
注4) URL https://github.com/cookpad/rrrspec
注5) 一般的にはMaster-Slaveパターンという名称で呼ばれることが多いと思われますが、ここでは後述するRRRSpecのシステム構成に合わせてこちらの名称を用いました。

▼図1　RRRSpecのシステム構成



の実行単位

・トライアル：タスクの実行結果

システム構成

「アプローチの検討」の項でも述べたように、RRRSpecはMaster-Workerパターンで実装されています。

これに、タスクセットの実行を依頼するClientと、その実行結果を保持するデータベースを加えたシステム構成図を**図1**に示します。図中の矢印はタスクセットの処理を開始してから終了するまでに生じるデータの流れを示しています。

図1からわかるように、RRRSpecではテスト対象のソースコードの共有にrsyncが用いられています。また、実行中のタスクセットに関する情報はRedisで保持され、その実行結果はRDBMSを用いて永続化されます。

ワークフロー

RRRSpecにおいて、タスクセットは次のようなフローに沿って処理されます。

❶テスト対象のアプリケーションからRRRSpecのコマンドを実行すると、rsyncによってソースコードがMasterに転送され、Redis上に対応するタスクセットが新規作成される

❷Masterはタスクセットを処理するWorkerを決定する

❸WorkerはMasterからrsyncによって対象のソースコードを取得する

❹Workerは（デフォルトで）マシンのCPUのコア数だけ子プロセスを生成する

❺各プロセスはタスクセットからタスクを取り出して実行し、トライアルの情報を記録する

❻トライアルが失敗し、かつ失敗回数が指定の値未満であれば、Masterは対応するタスクをタスクセットに戻す

❼❺～❻をタスクセットが空になるまで繰り返す

❽Masterはタスクセットの終了処理を行い、実行結果をRDBMSを用いて永続化する

このワークフローの制御にもRedisが用いられています。Redisのリスト型にはRPUSH(Right PUSH)とBLPOP(Blocking Left POP)というコマンドがあります。RRRSpecではこれらを組み合わせて用いることで、Redis上にいくつかのキューを構成しています。

そして、MasterとWorkerはこれらのキューにアイテムがエンキュー[注6]されるのを待ち、アイテムが追加されたらすぐにデキュー[注7]して処理するというプロセスを繰り返し行っています。このプロセスの中で、次に実行したい処理のキューにアイテムを適宜エンキューすることで、前述のようなフローを実現しています。

gemの構成

「はじめに」の節でも述べたように、RRRSpecはRubyのgemとして提供されています。gemは機能別に次の3つに分割されています。

- rrrspec-client
- rrrspec-server
- rrrspec-web

rrrspec-clientはクライアント用のgemで、テスト対象のアプリケーションにClientとしての機能を提供します。rrrspec-serverはサーバ用のgemで、MasterとWorkerを動作させるのに用います。最後のrrrspec-webは、タスクセットの実行状態や結果を表示するWeb UIを提供するためのgemです。ClientのCLIから同様の情報を確認することもできますが、Web UIのほうが使いやすいのでこちらをお勧めします。

環境構築

「アプローチの検討」の項でも述べたように、RRRSpecは分散実行を前提に設計されていますが、1台のマシンで動作させることもできます。RRRSpecのGitHubリポジトリでは、Dockerを用いて1台のマシンで動作させるサンプル[注8]が提供されています。しかし、このサンプルから実運用時の構成をイメージするのは難しいと思います(実際に筆者はここで苦労しました)。

そこで本記事では、より実運用時の状態に近くなるよう独自に再構成したものを用いて、チュートリアル形式で環境構築を行います。お手元の環境で試してみてください。なお、完成したコードは筆者のGitHubリポジトリ[注9]にホストしてあります。紙幅の都合上、一部のコードを省略していますので、適宜こちらのリポジトリを参照しながら読み進めてください。

必要なもの

本節では、次の4つがマシンにインストールされていることを前提としています。

- Ruby(2.3.1)
- Bundler(1.12.4)
- MySQL(5.7.12)
- Redis(3.2.0)

すべてのコードは上記がインストールされたMacBook Pro(OS X El Capitan)で動作確認を行っています。なお、括弧内の数字は動作確認を行った際のバージョン番号を示しています。

Client

テスト対象となるサンプルアプリケーションを作成し、これにClientとしての設定を行います。まず、clientというディレクトリを作成し、この中でbundle initコマンドを実行します。生成されたGemfileを次のように編集し、bundle installコマンドを実行します。

```
source "https://rubygems.org"

gem "rspec"
gem "rrrspec-client", "0.4.3"
```

注6) キューにデータを格納すること。
注7) キューからデータを取り出すこと。
注8) URL https://github.com/cookpad/rrrspec/tree/master/local_test
注9) URL https://github.com/yasaichi/rrrspec-tutorial

▼リスト1　client/.rrrspec（一部抜粋）

```
RRRSpec.configure(:client) do |config|
  #❶ rsyncのパス
  config.rsync_remote_path = ENV["RRRSPEC_RSYNC_REMOTE_PATH"]

  config.rsync_options = %w(
    --compress
    --times
    --recursive
    --links
    --perms
    --inplace
    --delete
    --cvs-exclude #❷ バージョン管理関連のファイルを除外する
  ).join(" ")

  #❸ トライアル回数の上限
  config.max_trials = 1
end
```

必要なgemがインストールできたので、テスト対象のアプリケーションを作成しましょう。最初に、次のコマンドを実行してRSpecのセットアップを行います。

```
$ bundle exec rspec --init
```

生成されたspecディレクトリ内に、samplesというディレクトリを作成します。この中に、前述のリポジトリを参考に次の3つのファイルを配置してください。

・fail_spec.rb：必ず失敗するタスク
・success_spec.rb：必ず成功するタスク
・timeout_spec.rb：必ずタイムアウトするタスク

サンプルアプリケーションの作成ができたので、Clientとしての設定を行いましょう。clientディレクトリの直下に、.rrrspec（**リスト1**）という設定ファイルを作成します。

「ワークフロー」の項でも述べたように、Clientはrsyncによってアプリケーションのソースコードを Masterに転送します。この転送先のパスを環境変数を介して設定しています（**リスト1**の❶）。転送時のオプションはrsync_optionsで指定でき、転送時間短縮のためにバージョン管理関連のファイルを除外しています（**リスト1**の❷）。なお、動作確認の際には必要ありませんが、実運用の際にはパスワードなしでMasterにSSH接続できるように鍵の設定を行っておく必要があります。

rsync以外にもさまざまな設定項目があり、その1つがトライアル回数の上限値であるmax_trialsです（**リスト1**の❸）。今回は「1」としていますが、実運用の際にはWorkerに障害が起きることも考慮して、2以上の値を設定しておくといいでしょう。

MasterとWeb UI

MasterとWeb UIを提供するアプリケーション（以下、Web）を動作させるための環境を構築します。まず、serverというディレクトリを作成し、この中でbundle initコマンドを実行します。生成されたGemfileを次のように編集し、bundle installコマンドを実行します。

今回はWebのアプリケーションサーバとしてUnicornを用いていますが、Rackアプリケーションに対応しているものなら何でも構いません。

```
source "https://rubygems.org"

gem "mysql2"
gem "rrrspec-server", "0.4.3"
gem "rrrspec-web", "0.4.3"
gem "unicorn"
```

必要なgemがインストールできたので、MasterとWebの設定を行いましょう。configというディレクトリを作成し、この中に、前述のリポジトリを参考に次の3つのファイルを配置してください。

・database.yml：データベースの接続設定ファ

▼リスト2　server/config/master.rb（一部抜粋）

```
#❶ MasterとWebで同じ設定を適用する
%i(server web).each do |type|
  RRRSpec.configure(type) do |config|
    config.persistence_db = db_config[env].symbolize_keys
    config.execute_log_text_path = root.join("tmp/rrrspec-log-texts")
  end
end

RRRSpec.configure(:server) do |config|
  #❷ デーモン化した際のPIDファイル
  config.pidfile = root.join("tmp/pids/rrrspec-master.pid")
end
```

▼リスト3　server/Rakefile（一部抜粋）

```
#❶ 設定ファイルのパスを環境変数を介して渡す
ENV["RRRSPEC_CONFIG_FILES"] = File.expand_path("../config/master.rb", __FILE__)

#❷ rrrspec-serverに定義されたRakeタスクを利用できるようにする
load "#{Gem::Specification.find_by_name('rrrspec-server').gem_dir}/tasks/db.rake"
```

イル[注10]

・master.rb：MasterとWebの設定ファイル
・redis.rb：Redisの接続設定ファイル

master.rbの一部を抜粋したものを**リスト2**に示します。

persistence_dbにはデータベースの接続情報、execute_log_text_pathにはトライアルのログを保存するディレクトリのパスを設定します。これらの値はMasterとWebで同一でなければならないので、ループを使って同じ設定を適用しています（**リスト2**の❶）。

また、動作確認の際には必要ありませんが、実運用の際にはMasterのプロセスをデーモン化して常に起動しておく必要があります。そのため、pidfileに書き込み権限のあるディレクトリのパスを設定しています（**リスト2**の❷）。

続いて、WebをUnicornで起動するための設定を行います。serverディレクトリの直下に、前述のリポジトリを参考にconfig.ruというファイルを配置してください。これはRackアプリケーションの設定ファイルで、Unicornの起動時に読み込まれます。

これで必要な設定が終わったので、最後にMySQL上に必要なデータベースとテーブルを作成しましょう。専用のコマンドがRakeタスクとしてgemに定義されているので、これを利用するためにRakefile（**リスト3**）を作成します。そして、次のコマンドを実行してください。

```
$ bundle exec rake rrrspec:server:db:
create rrrspec:server:db:migrate
```

RRRSPEC_CONFIG_FILESは設定ファイルのパスを渡すための環境変数です（**リスト3**の❶）。実行時に省略できるように、コード上で値を設定しています。そして、loadメソッドを呼び出すことで、rrrspec-serverに定義されたRakeタスクを読み込んでいます（**リスト3**の❷）。

Worker

Workerを動作させるための環境を構築します。Masterの作業と重複する部分が多いため、新たに作成する必要があるのはWorkerの設定ファイルであるserver/config/worker.rbのみです。このファイルの一部を抜粋したものを**リスト4**に示します。

注10）パスワードなしのrootユーザを想定しています。当てはまらない場合には適宜usernameやpasswordを追加・修正してください。

▼リスト4　server/config/worker.rb（一部抜粋）

```
RRRSpec.configure(:worker) do |config|
  #❶ rsyncのパス
  config.rsync_remote_path = ENV["RRRSPEC_RSYNC_REMOTE_PATH"]

  #❷ --cvs-excludeオプションは不要
  config.rsync_options = %w(
    --compress
    --times
    --recursive
    --links
    --perms
    --inplace
    --delete
  ).join(" ")
end
```

▼図2　タスクセットの一覧

「ワークフロー」の項でも述べたように、Workerはrsyncによってテスト対象のソースコードをMasterから取得します。この転送元のパスを環境変数を介して設定しています（**リスト4の❶**）。なお、ClientからMasterへの転送の際にバージョン管理関連のファイルは除外されているため、--cvs-excludeオプションの指定は不要です（**リスト4の❷**）。

また、Clientの場合と同様に、実運用の際にはあらかじめ鍵の設定を行う必要があります。

動作確認

正しく環境構築ができたかどうかを確認するために、RRRSpecを起動してテストを実行してみましょう。まず、serverディレクトリに移動し、Redisを起動します。

```
$ redis-server
```

また、rsync用のディレクトリを作成し、そのパスを環境変数に設定しておきます。

```
$ mkdir -p "${PWD}/tmp/rrrspec-rsync"
$ export RRRSPEC_RSYNC_REMOTE_PATH=
"${PWD}/tmp/rrrspec-rsync"
```

準備ができたら、次の3つのコマンドを実行してMaster、Worker、Webを起動します。

```
$ bundle exec rrrspec-server server
--config config/master.rb --no-daemonize
$ bundle exec rrrspec-server worker
--config config/worker.rb --no-daemonize
$ bundle exec unicorn -p 3000
```

この状態でclientディレクトリに移動し、次のコマンドを実行して処理を開始します。

```
$ bundle exec rrrspec-client start
--rsync-name=sample_app
```

http://localhost:3000にアクセスすると、Active Tasksetsに--rsync-nameで指定した名前のタスクセットが追加されています（**図2**）。

2分程度経つと、処理が終了してRecent

▼図3　タスクセットの実行結果

Tasksetsにタスクセットが移動します[注11]。タスクセットのキー名をクリックすると詳細ページに遷移し、実行結果を確認することができます(**図3**)。

サンプルアプリケーションは必ず失敗するタスクを含んでいるので、タスクセットの実行結果はFAILEDになっています。成功した場合の画面も見てみたい方は、これらのタスクをテスト対象から除外して再実行してみてください。

実運用時におけるポイント

RRRSpec導入時に直面した問題とその対処法や、実運用時における工夫を紹介します。

データベースに関する設定

「環境構築」の節では、データベースのない簡単なアプリケーションを用いて説明を行いました。しかし、通常はデータベースを用いるアプリケーションを扱うことがほとんどだと思います。この際、いくつかの注意が必要です。

「ワークフロー」の項でも述べたように、Workerはマシンの CPUのコア数だけ子プロセスを生成してテストを並列実行します。よって、マルチコアのマシンでは、2つ以上の子プロセスが生成されることになります。このとき、各プロセスで同じデータベースを用いると、データの競合が発生してテストが不安定になってしまいます。そのためプロセスごとに異なるデータベースを用いる必要があります。

RRRSpecでは、これらのプロセスに一意な非負整数が割り振られており、その値はSLAVE_NUMBERという環境変数から参照できます。これを利用することで前述の問題を回避できます。たとえばRailsアプリケーションの場合、データベースの接続設定(config/database.yml)を次のように書き換えます。

```
test:
  <<: *default
  database: rails_app_test<%= ENV["SLAVE_
NUMBER"] %>
```

そして、.rrrspecのslave_commandにデータベースの初期化処理を加えます。

```
config.slave_command = <<-SLAVE
  bundle exec rake db:migrate:reset --trace
  bundle exec rrrspec-client slave
SLAVE
```

Workerのスケーリング

「アプローチの検討」の項でも述べたように、RRRSpecはテストの規模に応じてWorkerを柔軟

注11) 2分以上経っても処理が終了しない場合、キャンセル用のコマンド「bundle exec rrrspec-client cancel キー名」を実行後、再試行してみてください。

にスケールアウトできることが長所の1つです。この長所を十分に生かすために、Workerのデプロイをいかに簡略化するかがポイントになります。

ピクスタでは、アプリケーションの環境構築とデプロイの自動化にAWS(Amazon Web Services)のOpsWorksを用いています。

Workerも環境構築はOpsWorksで行いますが、デプロイはせずにインスタンスのAMI(Amazon Machine Image)を作成しています。そして、AWSのAuto Scalingを用いて、このマシンイメージからスポットインスタンスを作成しています。Auto Scalingにはインスタンスの起動時に任意のスクリプトを実行できる機能があるので、これを利用してWorkerのデプロイに必要な処理を行っています。

起動するマシンの数やスペックの変更、起動時間のスケジューリングなどはすべて管理画面上から行えるので、使用状況に応じて柔軟に運用することができます。また、「導入効果」の項でも述べたように、スポットインスタンスを用いることで高速化と同時にコスト削減ができました。

SSHにおける同時接続要求数の制限

Auto Scalingを用いることでWorkerを簡単にスケールアウトできるようになりましたが、今度はマシンを一定以上の台数にすると、rsyncが頻繁に失敗するようになってしまいました。

原因を調査したところ、ブルートフォースアタック対策のために、同時にSSH接続を要求できるマシンの数が制限されていることがわかりました。この制限は`sshd_config`ファイル内の`MaxStartups`という項目から変更できるので、Workerマシンの数に応じて適切な値を設定しておきましょう。

CIサービスからの移行

現在ほとんどのCIサービスには、GitHubをはじめ、さまざまなホスティングサービスとの連携機能があります。これを利用すれば、リモートブランチへのpushをフックにして、自動でテストを実行できます。しかし、RRRSpecには同様の機能がないので、CIサービスからの移行時には何らかの対応が必要です。

ピクスタでは、Node.jsで専用のアプリケーションを実装し、Gitフックでこの APIを呼び出すことで対処しました。APIでは次のような処理が行われます。

❶テスト対象のソースコードを取得し、push時の状態にする
❷必要なgemをインストールする
❸RRRSpecのコマンドを実行し、テストを開始する

❸のコマンドの実行時には`--rsync-name`というオプションを指定でき、指定した値がWebでの表示に用いられることは「動作確認」の項で述べたとおりです。この値にアプリケーション名とコミットIDを含めることで、各タスクセットとの対応がわかるようにしています。

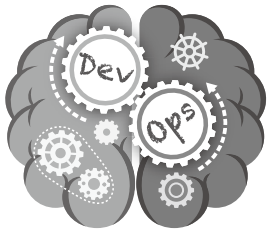
まとめ

本記事では、次のような内容を取り上げました。

・ピクスタでのRRRSpecの導入事例
・テスト高速化のためのライブラリの比較
・RRRSpecのシステムアーキテクチャ
・RRRSpecのgemの構成
・環境構築の手順とポイント
・導入時に直面した問題とその対処法
・実運用時における工夫

本記事の内容が、RSpecを用いた自動テストの高速化に取り組んでいる方や、他の言語やフレームワークにおいて同様の課題を抱えている方の参考になれば幸いです。SD

アプリエンジニアのための［インフラ］入門



Author 出川 幾夫（でがわ いくお） レバレジーズ株式会社 teratail開発チーム Twitter @ikuwow

第2回 ネットワーク入門

「現場でDevOpsを実現させるには、まずアプリエンジニアがインフラを知る必要がある」という前提に立ち、アプリの視点からインフラを広く学んでいく本連載。第2回では、インターネットを支えるネットワーク技術の全体像を、「階層構造」「プロトコル」「アドレス」をキーワードに概観してみましょう。

エンジニアの必須科目

今やあらゆるアプリケーションがインターネットに接続され、何らかの形で外部のサーバなどと通信をします。またIoTという言葉も一般的に使われるようになり、パソコンだけでなくありとあらゆるものがインターネットにつながる時代が来ています。

とくにWebの技術は、多くのネットワーク技術から成り立っています。非常に複雑なWebの世界を理解するためには、ネットワークの基礎知識は必須です。

筆者はWebアプリケーションのパフォーマンス分析の際に、プロトコルの仕様を詳細に調べたり、パケットキャプチャをして通信のモニタリングを行ったりしたことがあります。また大きなネットワーク構成を構築する場合や、トラブルが起こった際の原因の特定も、ネットワーク階層やプロトコルの理解があると便利です。

ネットワークの技術は非常に広範囲に渡ります。本稿では基本的な概念と身近な部分を中心に紹介していきます。

ネットワークの階層モデル

ネットワークの大きな特徴が階層構造です。ネットワーク全体の理解のためには、階層モデルと通信の流れをはじめに理解しておくのが近道です。

OSI参照モデル

ネットワーク技術はおもに「プロトコル」と呼ばれる通信規則を基に成り立っています。ネットワークを学ぶことはプロトコルを学ぶことと同義と考えても良いかもしれません。

プロトコルには、HTTP、FTP、SMTP、DNS、SIP、TCP、IEEE802.11など本当にさまざまな種類があります。これらを整理するために便利なのが「OSI参照モデル（Open System Interconnection reference model）」です。OSI参照モデルでは、ネットワークとその技術を7つの階層に分類しており、それぞれの層に、通信における役割が定義されています（**表1**）。プロトコルの役割も、これらの層のいずれかに当てはめることができます。

この中でセッション層とプレゼンテーション層はアプリケーション層との役割分担があいまいで、TCP/IPの範疇（はんちゅう）では、この3つを合わせてアプリケーション層と呼ぶこともあります。

パケットの流れ

インターネットでは基本的にデータを「パケット」という小さな区切りに分割して伝送します。

▼表1　OSI参照モデル

階層	名称	役割	プロトコル
第7層	アプリケーション層	特定のアプリケーション間の通信	HTTP、FTP、SMTP、DNS、SNMPなど
第6層	プレゼンテーション層	機器やデータに固有のフォーマットの違いを吸収	
第5層	セッション層	通信のコネクションの確立・切断	
第4層	トランスポート層	送信元アプリケーションから宛先アプリケーションへデータを伝送する。アプリケーションごとにポート番号というアドレスが割り振られる	TCP、UDPなど
第3層	ネットワーク層	送信元ホストから宛先ホストへデータを伝送。経路を管理し、IPアドレスを用いて場所を識別	IP、ARP、ICMPなど
第2層	データリンク層	スイッチングハブ間など、直接接続された機器間での通信	Ethernet、PPP、IEEE802.11など
第1層	物理層	電気信号の物理的な通信。ケーブルやコネクタの形状を策定。ビット列を電気信号へ変換する	

パケットはこの7階層を上から下へ降りながら、順にその階層の通信に必要なヘッダを付与されていき、さらに下の層へ送られます。ヘッダはプロトコルごとに固有の形式をしていて、宛先やデータのサイズなど、そのプロトコルの通信に必要な情報が含まれています。最終的に物理層で情報が電気信号まで変換され、宛先へ流れていきます。

逆に受信側では、そのパケットのヘッダを1つ読み取って外しては上の層に渡します。このようにして受信データは最終的にアプリケーションに到達し、通信が完了します(**図1**)。

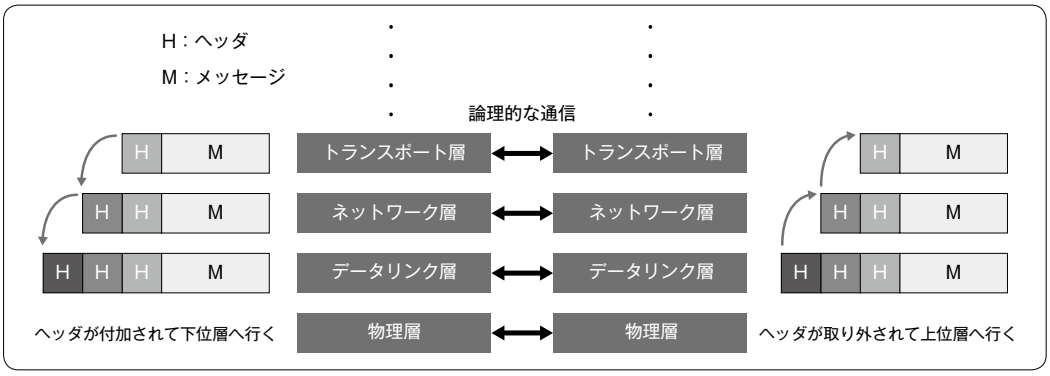
この階層構造の良いところは、階層それぞれの役割がはっきり分かれていて、それぞれ独立した通信として扱うことができる点です。**図2**のように、クライアントPCとサーバはトランスポート層以上の通信に関してはルータのことを意識する必要はなく、一対一の通信として扱うことができます。

また、ルータはネットワーク層までの通信を制御する装置ですが、トランスポート層以上でどのような通信が行われようが、その役割や処理は変わりません。

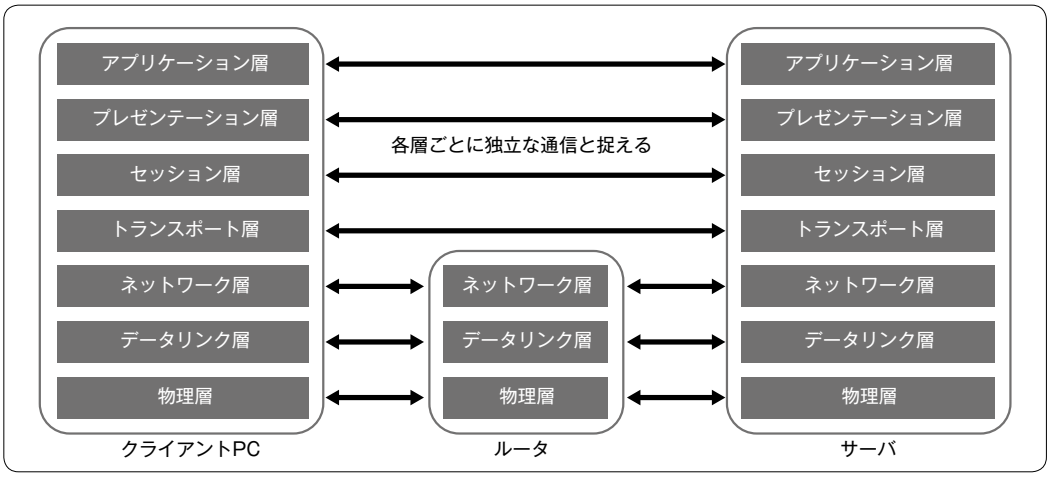
同じインターネットというしくみでも、まったく同じ形状のケーブルでも、この階層構造があることによって、さまざまなアプリケーションがさまざまな種類の通信を行うことができるというわけです。この柔軟性と適度な独立性が、インターネットを柔軟で強力なものにし、世界中のあらゆる人々のインフラとして普及させた理由の1つではないかと筆者は思います。

プロトコルや機器の名前を聞いたら「どの階

▼図1　パケットの流れ



▼図2　層ごとに独立して実現するインターネットの通信



層の技術か？」という問いを立てて整理することで、個々の技術の理解が深まります。ネットワークの議論はこの階層モデルを前提として行われることが多いので、OSI参照モデルの7階層はぜひ暗記しておきましょう。

TCP/IP

TCP/IPという言葉を一度は聞いたことがあると思います。これはインターネットに関わるプロトコルの総称のことです。TCPとIPはそれぞれ個別のプロトコルの名前なのですが、インターネットの根幹を成すプロトコルのため、このような名前になっています。

インターネットのしくみは、すべてこのTCP/IPで説明できると言っても過言ではありません。ここでは最も重要なIP、TCP、UDPの3つについて簡単に説明していきます。

IP

IP（Internet Protocol）は第3層のネットワーク層のプロトコルで、host-to-hostの通信を実現します。「Internet」という名前のように、ネットワーク間を飛び越えて通信を行うことができます。終端のクライアントPCやサーバのほかに、ルータがこのネットワーク層の通信を制御します。

IPは経路制御といい、宛先IPアドレスまで到達可能な経路を探索して、その方向にパケットを送り出すのがおもな役割です。現在広く普及しているのはIPバージョン4（IPv4）ですが、32bitで表すIPv4アドレスの数が枯渇したため、アドレスを128bitで表すIPバージョン6（IPv6）へ少しずつ移行が進んでいます。

IPはコネクションレス型で、通信の開始の際に宛先ホストと接続の確立を行いません。ベストエフォート型とも呼ばれ、「精一杯がんばるが結果は保証しない」というプロトコルです。そのためパケットを紛失した場合に再送するしくみなどはありませんし、宛先サーバの電源が落ちていても、問答無用でデータを送信するという単純な動作をします。

TCP

TCP（Transmission Control Protocol）は第4層のトランスポート層のプロトコルです。IPの上位層に位置します。

TCPの通信は「コネクション型」です。通信の初めに3ウェイハンドシェイクという処理を行って接続を確立してから、データの送受信を開始します。送信中にパケットの順番が入れ替わってしまった場合の順序制御や、パケットに

紛失があった場合の再送制御、流量制御や輻輳(ふくそう)制御など、通信の品質やネットワークの効率を上げるしくみが多数用意されています。HTTPやFTP、SMTPなど、多くのプロトコルがTCPの上に成り立っています。

UDP

UDP(User Datagram Protocol)も第4層のトランスポート層のプロトコルの1つです。UDPはTCPとは対照的に、「コネクションレス型」の通信をします。パケットの構造が非常に単純なため、高速な動作をします。これはDNSやSNMP、SIPなど、おもに信頼性よりも高速な通信が求められるプロトコルで利用されます。動画や音声のストリーミングなどもUDPが得意な分野です。

TCPとUDPはどちらが良い悪いという話ではなく、それぞれ得意な分野があるのです。

場所を表すさまざまなアドレス

インターネット上の場所を表すのに、さまざまなアドレスが利用されます。URLもアドレスの1つと言えますし、アプリケーションからデータベースサーバを参照するなど、インターネット上の場所を指定する機会は多々あります。どのアドレスもそれぞれ属している階層があり、役割が違います。

ホスト名、ドメイン名

たとえば今、それぞれfirst.example.com、second.example.comという名前が割り当てられたホスト(ここではそれぞれ1台のサーバ)があるとします。ドメイン名とはexample.comの部分を指します。これは個人や会社が購入して取得するもので、ICANNという組織とそこから委託された組織が管理しています。

ホスト名はfirstやsecondの部分を指します。同じドメイン内で、どのホストかを区別するための名前です。またfirst.example.comやsecond.example.com全体をホスト名と呼ぶこともあります。このような形式は、FQDN(Fully Qualified Domain Name)とも呼ばれます。

IPアドレス

ホスト名は人間が認識しやすくするための形式ですので、通信を行う際にはDNSというしくみによってIPアドレスに変換され、通信を行います。

IPアドレスはネットワーク層のプロトコルのIPにおけるアドレスで、ネットワーク上の1ノードを表すのに用いられます。

IPv4ではIPアドレスは202.241.189.71のように、3つのドットで区切られた4つの数字で表されます。ネットワークの範囲を表すサブネットマスクを合わせて「CIDR」という形式(192.168.252.52/24)でも表現されます。

IPアドレスには、ローカルIPアドレスとグローバルIPアドレスの2種類があります。ローカルIPアドレスは、CIDRで表すと10.0.0.0/8、172.16.0.0/12、192.168.0.0/16の範囲のアドレスを指し、ローカルネットワーク内で自由に用いていいアドレスです。グローバルIPアドレスは世界中で通用するIPアドレスで、IPアドレスを指定できれば世界中のどこにあるホストでも通信を始めることができます。

ポート番号

ポート番号とは、TCPとUDPにおいて、アプリケーションを区別する番号のことを言います。送信元ポート番号と宛先ポート番号があります。ポート番号はホスト名やIPアドレスと合わせてexmample.com:8888や192.168.252.52:9200という形式で表すこともできます。

ポート番号は、0～65535の範囲で表されます。とくに0～1023まではウェルノウンポートといい、ICANNによってそれぞれの用途が定められています。たとえば、HTTPは「TCP80番」、SMTPは「TCP25番」、DNSの名前解決リクエストは「UDP53番」、というふうにです。

http://example.comというアドレスをブラウザのアドレスバーに入力してEnterを押すと、暗黙の内にexample.comというホストのTCP80番ポートに、HTTPのリクエストを送っていることになります。

MACアドレス

MAC(Media Access Control)アドレスとは、ネットワークインターフェースごとに一意に付けられているアドレスのことで、ハードウェアアドレスとも呼ばれます。たとえば、有線LANコネクタと無線LANのあるPCは、MACアドレスを2つ持っているということになります。

IPアドレスはネットワーク層のアドレスですので、データリンク層以下の通信しか行わないスイッチングハブなどを指し示すことには使えず、どの機器を通れば宛先のIPアドレスを持つホストに到達できるかがわかりません。ARP(Address Resolution Protocol)によって、宛先IPアドレスからどのMACアドレスを持つ機器に送るべきかを探します。

アドレスを調べてみよう

MACアドレスやIPアドレスは、Unix系のOSではifconfigコマンドを使うことで簡単に調べられます。

▼図3 ifconfigコマンドの例

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0D:5E:50:E6:9D
          inet addr:192.168.252.52  Bcast:192.168.252.255  Mask:255.255.255.0
          inet6 addr: fe80::20d:5eff:fe50:e68d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:50016781 errors:0 dropped:0 overruns:0 frame:0
          TX packets:71994847 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13032908299 (12.1 GiB)  TX bytes:89221435291 (83.0 GiB)
          Interrupt:18
...(略)...

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
...(略)...
```

(1) (2) (3) (4)

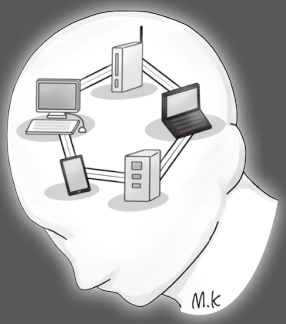
図3におけるHWaddrの後の12桁の16進数で表されるものがMACアドレスです(1)。また、inetの行のaddr:の後がIPv4アドレス(2)、inet6の行のaddr:の後がIPv6アドレス(3)をそれぞれ指します。

loと書かれているのはループバックインターフェースと言い、ホスト自身を指すインターフェースです。そのIPアドレスはIPv4だと127.0.0.1となっています(4)。また複数のネットワークインターフェースがあるマシンではもちろん、仮想マシンを立ち上げている場合でも、その数だけ仮想的なインターフェースが作られ、それぞれにIPアドレスやMACアドレスが割り振られます。

おわりに

ここで紹介したネットワーク技術はほんの一部です。インターネットは基本的に世界のどこにも中心がない分散型の構造になっており、小さくシンプルなプロトコルの組み合わせで成り立っています。実用のためだけでなく教養としても非常におもしろい分野だと思います。興味のある人はぜひ『マスタリングTCP/IP[注1]』などの書籍などを参考にしてみてください。SD

注1) 竹下隆史, 村山公保, 荒井透, 苅田幸雄 著, オーム社, 2012.



仮想化の知識、再点検しませんか?
使って考える仮想化技術

本連載は「仮想化を使う中で問題の解決を行いつつ、残される課題を整理する」ことをテーマに、小さな仮想化環境の構築・運用からはじめてそのしくみを学び、現実的なネットワーク環境への実践、そして問題点・課題を考えます。仮想化環境を扱うエンジニアに必要な知識を身につけてください。

Author
笠野 英松(Mat Kasano)
有限会社 ネットワーク・メンター
URL http://www.network-mentor.com/indexj.html

第3回 仮想環境の構築(その2)~仮想マシンの作成

今回は、ホストシステムとして前回構築したデフォルト設定のCentOS 6.7上のKVM(コラム参照)を使って、仮想マシン(ゲストシステムのハードウェア)の作成とゲストシステムのOSとネットワーク環境のインストールを行います。

仮想マシン/ゲストシステムの作成

仮想マシンの作成は(今回は)仮想マシンマネージャーで5ステップで行います。

CentOSのメニューから、[アプリケーション]→[システムツール]→[仮想マシンマネージャー]で仮想マシンマネージャーを起動します。これは前回説明しているので、ここで問題が発生したならば前回の記事を参考にしてください。

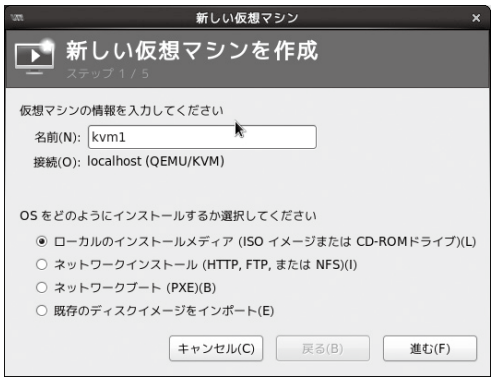
最初に、仮想マシンマネージャー画面の「localhost(QEMU)」上でマウスを右クリックして「新規」を選ぶか、またはメニューアイコンの「新しい仮想マシンの作成」ボタン(**図1**)をクリックして仮想マシンの作成を開始します。

▼図1 新しい仮想マシンを作るウィザードを起動

ステップ1:仮想マシン名とOSメディア

最初の画面(ステップ1)では仮想マシン名とOSのインストール元/メディアを設定・選択します(**図2**)。

▼図2 新しい仮想マシンを作るステップ1



▼図3 新しい仮想マシンを作るステップ2

Column サーバ仮想化製品「KVM」について

KVM※1はOVA※2で注目を浴びており、RHEL6からOSパッケージに同梱されています。完全仮想化マシンHVM※3を提供するもので、パッケージのインストールを除けば、運用管理はサーバ上で、Xenとほぼ同様なコマンド・操作を「仮想マシンマネージャー（virt-manager）」で行えます（仮想マシンのI/Oパフォーマンスを上げるvirtioという、仮想マシン上の準仮想化ドライバもある）。なお、KVMではハードウェア仮想化支援機構（VT機構）が必要になります。Xenと同様にリモートからの運用管理機構はなく、VNCなどを利用します。KVMはLinux OSに同梱されており、仮想化（KVM）パッケージ（パッケージグループ：仮想化、仮想化クライアント、仮想化プラットフォーム）の同時インストールが可能です。KVMとXenにほぼ共通な運用管理については、本連載のKVM実践の中で解説していく予定です。

※1）Kernel-based Virtual Machine（http://www.linux-kvm.org/page/Main_Page）
※2）Open Virtualization Alliance（2011年5月17日発足。https://openvirtualizationalliance.org/）
※3）Hardware Virtual Machine（http://www.linux-kvm.org/page/FAQ#What_is_Intel_VT_.2F_AMD-V_.2F_hvm.3F）

仮想マシン名は、仮想マシンマネージャーや仮想マシン管理ユーザインターフェース（仮想マシン管理コマンド）の管理対象（「ドメイン」という）の名前で、仮想マシンの設定ファイル（/etc/libvirt/qemu中のxmlファイル）の名前にもなります。本連載では仮想マシンのOSとしてWindows 7とFreeBSD 10.3をインストールし、それぞれkvm1、kvm2という仮想マシン名をつけることにします。

インストールメディアとしては**図2**のように、CD/DVDドライブもしくはISOイメージファイルというローカルデバイス／ファイル、URL指定のネットワーク先、PXEネットワークブート、そして既存のディスクイメージ（.imgファイル）の4種類があります。手持ちのものにあわせて選択しますが、今回は後述するようにWindows 7はDVD、FreeBSD 10.3はISOイメージファイル[注1]でインストールを行います。

ステップ2：メディア場所とOS

次のステップ2で、そのインストールメディアの場所とOSの種類／バージョンを指定しま

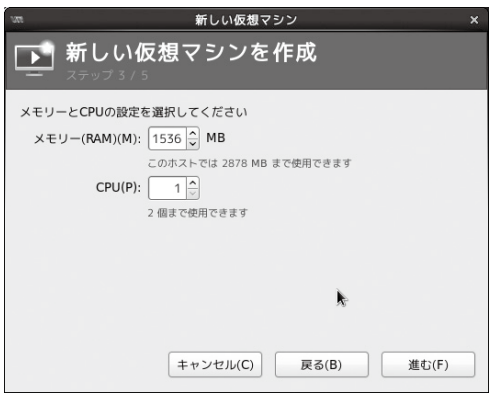
注1） FreeBSD 10.3の提供元
https://www.freebsd.org/ja/where.html

▼表1　KVMで指定可能なOS一覧

OS種別	バージョン
Windows	Microsoft Windows 2000/XP/Vista/7以降
	Microsoft Windows Server 2003/2008以降
Unix	OpenBSD 4.x以降
	FreeBSD 6.x/7.x/8.x以降
Solaris	Sun Solaris 9/10以降
	Sun OpenSolaris以降
Other	Novell Netware 4/5/6以降
	MS-DOS
	Generic
Linux	Red Hat Enterprise Linux 2.1/3/4/5/5.4/6/7以降
	Fedora Core 5/Core 6/7/8/9/10/11/12/13/14/15/16/17/18以降
	Debian Etch/Lenny/Squeeze/Wheezy以降
	Mageia 1以降
	Mandriva Enterprise Server 5.0/5.1以降
	Mandriva Linux 2009 or earlier/2010以降
	Suse Linux Enterprise Server 11以降
	openSuse 11/12以降
	Ubuntu 8.04 LTS/8.10/9.04/9.10/10.04 LTS/10.10/11.04/11.10/12.04 LTS/12.10以降
	Generic 2.4.xカーネル/2.6.xカーネル/2.6.25以降（virtioによる）

▼図4　ISOイメージファイルの指定手順

▼図5　新しい仮想マシンを作るステップ3



す(**図3**)。インストールメディアの場所は「ネットワークブート(PXE)」を選択する以外は指定する必要があります。

OSの種類とバージョンは前ページにある**表1**のものが選択可能です。OSの種類では「全般」「Windows」「UNIX」「Solaris」「Other」「Linux」の中から、バージョンは「全般」から多数のOSが選択できます。OSの種類で「Show all OS options」を選択すると全表示されます[注2]。

OSの種類は仮想マシンのACPIやAPIC、マウスドライバ、I/Oインターフェースなど OSの機能特性をOSメディアから取得して最適化するのに使用されます。もしOSの種類を(バージョンも)「全般」にした場合、この最適化が行われません。なお、URL指定(「ネットワークインストール」)の場合のみ、「インストールメディアに応じてOSの種類を自動判別する」によりこれを自動検出することができます。

今回は、Windows 7のDVDインストール(**図3**)とFreeBSD 10.3のISOイメージインストールを選択します。ISOイメージインストールではそのISOイメージファイルの物理ホストのファイルシステム内のパスを設定する必要があります(**図4の(a)～(f)**)。

ステップ3:メモリとCPU

ステップ3は仮想マシンに割り当てるメモリサイズとCPU数の設定です(**図5**)。

指定可能な数は灰色文字で選択欄下に表示されていますが、並行稼働させる仮想マシン台数と物理ホストを考慮して決めます。なお、今回の処理では、並行処理ではなくWindows 7または、FreeBSD 10.3のどちらか一方を稼働させ、他方は停止しておく前提の設定です。

注2) 詳細なOSの一覧は、端末コマンドのman virt-installまたは、virt-install --os-variant listで調べることもできる。

▼図6　新しい仮想マシンを作るステップ4

ステップ4：ストレージ

ステップ4は仮想マシンストレージの設定です(**図6**)。ストレージは物理ホスト内のハードディスクや接続デバイスから選択します。

ディスクイメージを作成する場合、この時点でディスク全体を割り当てるかどうかを選択可能です(「今すぐディスク全体を割り当てる」をチェック)。ここで大きなサイズを割り当てた場合、仮想マシン作成時に時間がかかりますが、OSインストール時にはその時間が不要になります。これを選択しない場合、ディスクサイズを割り当てる仮想マシン利用時に時間がかかります。また、割り当てサイズが利用可能なサイズを越えていると、このオプションを選択している場合には仮想マシン作成時にエラーとなります。逆にこのオプションを選択していない場合には、仮想マシン利用時に問題が発生する可能性があります。

ステップ5：最終確認

ステップ5では今までの設定の最終確認を行います(**図7**)。ここではデフォルト設定で作成するため、「インストールの前に設定をカスタマイズする」を選択していません(運用管理の回に「設定のカスタマイズ」として取り上げる予定です)。また、「詳細なオプション」で各種オプションの追加設定が可能ですが、ここもそのままにしています。

▼図7　新しい仮想マシンを作るステップ5

▼図8　新しい仮想マシンを作成中

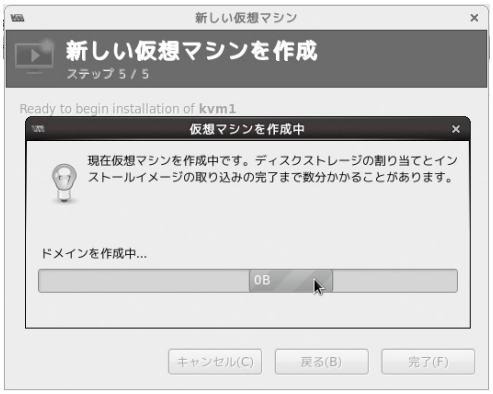


図7では「仮想ネットワーク 'default' : NAT」しか有効になっていませんが、ほかの接続、たとえばブリッジも仮想マシンマネージャーの「接続」に加えることで利用可能になります(別の回で解説します)。

ほかには、MACアドレスや仮想化の種類(KVM)、32/64ビットアーキテクチャの設定・選択が可能です。なお、MACアドレスは変更可能ですが、基本的にKVM(仮想マシン)ではベンダー識別子(OUI)[注3]は「52:54:00」を使用し

注3）Organizationally Unique Identifier。管理組織(ベンダー)識別子の詳細はhttp://standards-oui.ieee.org/oui/oui.txtを参照。IEEEが規定するMACアドレスの先頭3オクテットがOUI。KVMのOUIはIEEEに登録されていないが、qemu/kvmでは「52:54:00」と規定されている。

ています(デフォルトで自動設定される)。

以上で仮想マシンの作成設定が終了し、「完了」をクリックすると仮想マシンが実際に作成されます(**図8**)。

◆ ◆ ◆

このあと、ゲストシステムのOSインストールを開始します。

ゲストシステムのOSとネットワーク環境のインストール

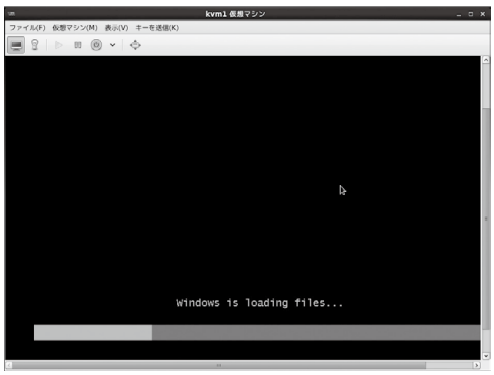
ゲストシステムのOSインストールは、仮想マシンの作成後、「完了」ボタンで自動的に引き続いて開始されます。

ステップ2で指定したゲストシステム用OSの所在場所から、そのOSのDVDやISOファイルイメージなどによりインストール実行となります。本稿では、Windows 7(**図9**)とFreeBSD 10.3(**図10**)の2つをそれぞれ単独でインストールしています。

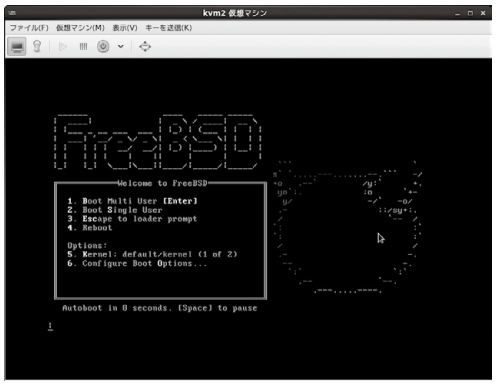
・仮想マシン1(kvm1):Windows 7
・仮想マシン2(kvm2):FreeBSD 10.3

紙幅の都合でインストール手順は割愛しますが、FreeBSD 10.3はデフォルトインストールとしています。ですのでCUI(Character User Interface:テキスト表示・入力)のシステムとして作成されます。

▼図9 Windows 7をインストール



▼図10 FreeBSD 10.3のインストール



ゲストOSインストール後の状態

ゲストOSのインストール後、仮想マシンマネージャー画面で仮想マシンの状態を見ると**図11**のように「実行中」であることがわかります。これは次のコマンドでも確認できます。

```
[root@vm1 ~]# virsh list
 Id    名前                         状態
----------------------------------------------------
 1     kvm1                         実行中
```

仮想マシンイメージ

インストール完了後、仮想マシンのイメージが**図12**のように作成されています。その所在パス名は仮想マシンの設定xmlファイル(/etc/libvirt/qemu/kvm1.xmlおよびkvm2.xml)中のdomain/devices/disk/sourceセクションのfileに記述されています。

▼図11 仮想マシンマネージャーによるゲストマシン状況の確認

▼図12　仮想マシンイメージファイル

```
[root@vm1 ~]# ls -al /var/lib/libvirt/images
合計 20971724
drwxr-xr-x. 2 root root        4096  6月 11 19:14 2016 .
drwxr-xr-x. 9 root root        4096  6月 11 22:17 2016 ..
-rw-------. 1 root root 10737418240  6月 12 18:24 2016 kvm1.img
-rw-------. 1 root root 10737418240  6月 11 22:10 2016 kvm2.img
```

仮想マシン作成中に起こる問題と対策

仮想マシン作成中に起こる問題はインストール元が見つからないとか、OS選択の詳細画面が表示されないなどの問題です。概して、処理の手順や操作の誤りからくるものです。

仮想インストールのDVDメディアが選択できない

DVDインストールの場合は、あらかじめそのDVDメディアを物理ホスト上にマウントしておく必要があります。さもないとDVDメディアが選択できないので、**図3**の「CD-ROMまたはDVDを使用」の選択欄を使おうとしても「メディアがありません（/dev/sr0）」と表示されてしまいます。

マウントしてあれば**図13**のようにデスクトップ上にDVDメディアのアイコンが表示されるので、この状態にしてから仮想マシンの作成へと進んでください。

▼図13　DVDメディアがマウントされた状態

仮想インストールのISOイメージファイルが選択できない

このISOイメージファイルは仮想マシン作成開始前にあらかじめ指定するパスに提供元からダウンロードして、適当な場所に格納しておかなければなりません。

ゲストOSが選択できない

「OSの種類」の選択欄では、最初は、全般、Windows、Linuxだけしか表示されませんが、「Show all OS options」をクリックするとすべてのOSの種類が表示され、選択できるようになります。バージョンの「全般」以外は、OSの種類を「全般」以外を選択することで表示されるようになります。

次回予告

次回は今回作成／インストールした仮想マシンを仮想マシンマネージャーから使用する方法を解説します。また、仮想マシンの初期状態を確認して、問題がないかを調べます。SD

連載各回では、読者皆さんからの簡単な「ひとくち質問（QA）コーナー」や「何かこんなこともしてほしい要望（トライリクエスト）コーナー」を設けて「双方向連載」にできればと思っていますので、質問や要望をお寄せください。

宛先：sd@gihyo.co.jp
件名に、［仮想化連載］とつけてください

RDB性能トラブルバスターズ奮闘記

原案 生島 勘富(いくしま さだよし) info@g1sys.co.jp ㈱ジーワンシステム
構成・文 開米 瑞浩(かいまい みずひろ) イラスト フクモトミホ

第6回 「テーブル設計は後まわし!」にするやり方もある

ウォーターフォール・モデルの新規開発案件で悩ましいのが、開発の後半でUIが変更になった場合に、DBのテーブル設計およびシステム開発全体に手戻りが発生すること。五代さんと大道君も新規開発に取り組むようですが、やはりそのことに頭を悩ませているようです。

登場人物紹介

生島氏
DBコンサルタント。性能トラブルの応援のため大道君の会社に来た。

大道君
浪速システムズの新米エンジニア。素直さとヤル気が取り柄。

五代氏
大道君の上司。プロジェクトリーダーでもある。

新規開発やりますよ!

大阪を中心に20年間システム開発に携わったあと、現在は東京で仕事をしている「SQLの伝道師」ことジーワンシステムの生島です。いつもは既存のあるいは開発中のシステムに関してパフォーマンスの相談を受ける取引先、浪速システムズの五代さんから今回は新規開発案件の相談を受けました。

「生島さん、ウチが依頼を受けたお料理レシピ投稿サイトの開発について、技術アドバイザという立場でご協力お願いしたいんですが」

今でいうならクックパッドのようなサイトですね。料理のカテゴリ(和・洋・中など)、使っている食材、所要時間や用途(パーティ用、お弁当用、子供向け、減塩・糖質制限など)といったさまざまな条件で検索できるもの。コンシューマー向けサービスですので、当たれば利用者数は100万単位で増えることが予想されます。ちなみにクックパッドは現在有料会員が100万人を超え、月間ユーザ数も5,000万人に達する巨大サービスに成長しています。

当然、下手な作り方をすると性能問題を起こすでしょう。

「性能トラブルを起こしてから手を打つより、最初から意識して作ったほうがえぇと思いまして、生島さんにご協力してもらいたいんです」と五代さん。確かにそのとおりなので、大道君を含む開発チームに対して私がコンサルタントとしてサポートする、ということで依頼を受けることにしました。

そこで、真っ先に提案したのがこの方針です。

「極めて複雑なというほどじゃないにしても、そこそこ複雑なユーザインターフェース(以下、UI)が必要ですね……でしたら、テーブル設計は後まわしにしましょ!」

「えっ」「えっ」大道君と五代さんが同時に驚きの声を上げました。無理もないことで、ほとんどの会社では普通そういうやり方はしないはずです。しかし、あるやり方をすれば、このほうがうまくいくのです。どういうことか、詳しく説明しましょう。

テーブル設計は後まわし!の真意とは?

典型的なウォーターフォール・モデルの開発工程は**図1**のようになり、基本設計の段階でテー

ブル設計とUIを含む機能設計を行います。現在では早めにプロトタイプを作って利用者に操作感を確認してもらい、必要に応じて修正を加えながら進めるアジャイル的なスタイルを取り入れている開発案件も増えているとはいえ、基本はウォーターフォールというケースがまだまだ多く、五代さんたちもそう考えていたようです。しかしすでによく知られているように、この方式ではどうしても「手戻り」が多発します。

「新人ITエンジニアへの教育をやってるときだったら、手戻りが起きるのは要件定義や基本設計をいいかげんにやってるからや！　きっちりやれ！　とうるさく言うことにも意味がありますけど、実際のところそれで手戻りがなくなると思いますか？」

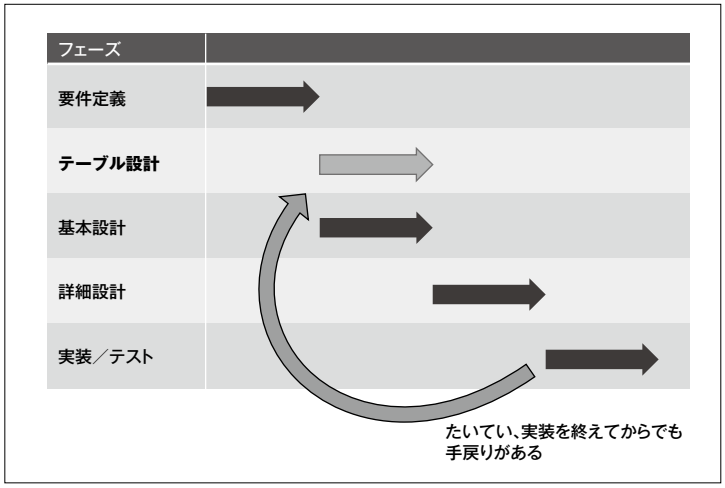
「お客さんが仕様をきっちり決めてくれたら……」

「確かにそうなんですけど、お客様も、自分でもわからへんものは決められへんでしょ」

「確かに、やってみんとわからんもんはいろいろありますね。こういうコンシューマー向けサービスはとくにそういう面がキツイし……」

「問題は、UIの仕様変更があとになるほど、その修正がシステム全体に影響することが多くなるってことです」

▼図1　典型的ウォーターフォール・モデルの開発工程



テーブル設計の変更はシステム全体に影響する

図2に挙げたとおり、本来(1)バックエンドの仕様はフロントエンドのニーズに応じて決めるものです。ここで言うフロントエンドとはユーザ体験(UX：User experience)およびそのために用意されるUIのこと、バックエンドとはビジネスロジックモジュールやデータベースのことです。ところが、(2)フロントエンドには仕様変更がどうしてもよくあります。UXなんか使ってみないとわかりませんから、動くシステムができて使ってみてようやく「ああしたい、こうしたい」という具体的なニーズが出てくるわけです。

「現実的にはそうやねぇ……」と五代さん。

困るのは、それによって(3)テーブル設計も変更が必要になり、(4)その影響がシステム全体に波及しやすいということです。画面のボタンの位置を変えるといった修正は一画面だけにとどまりますが、新しい情報項目を増やすような修正はそのテーブルを使う全モジュールに影響します。

「そうなりますね……」と大道君。

「これを防ぐには、どう

したらいいですか？」

「要件定義でしっかりヒアリングを……」

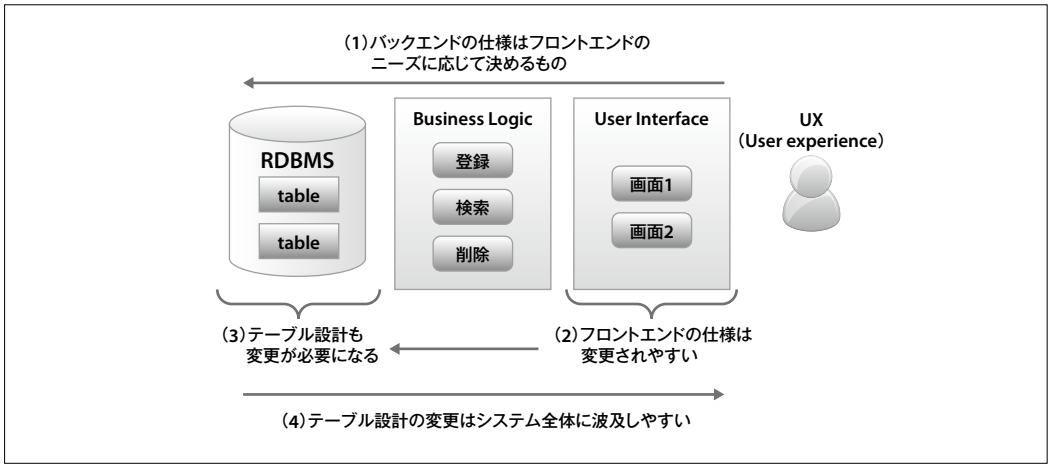
「無理言うてますやん……客のせいにしない方向で！」

「ガハハ。結局のところ、使ってみんとUI/UXが決まらんいうことは、とっとと使わせて決めてもらうしかないんとちゃいますか。要するにプロトタイピング、アジャイル的なやり方で」

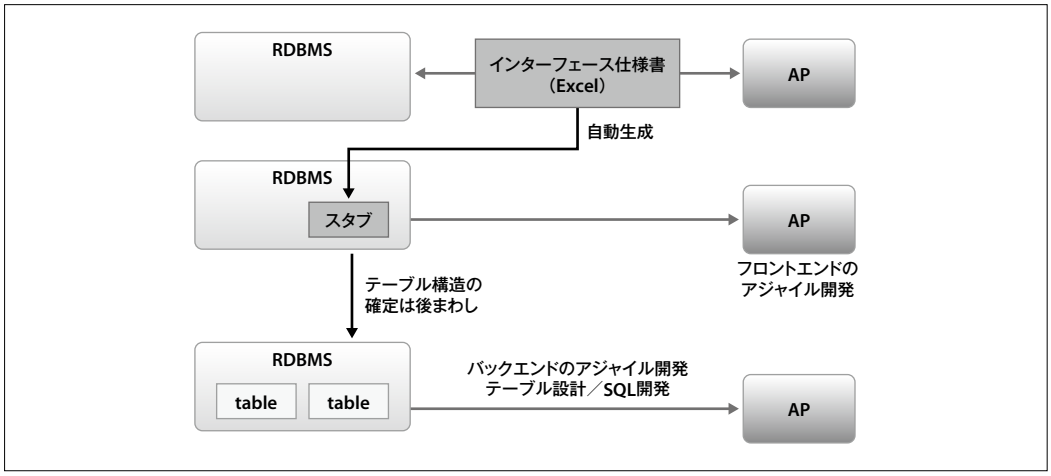
「そう、それをやろうってことです。そのためにウチで採用してきた実績のある簡単な方法があるんですよ」

「おお、どうやるんですか？」

▼図2　UIの仕様変更がシステム全体に影響する理由



▼図3　Excelでインターフェース仕様書を書いてDBスタブを自動生成



I/F仕様書を書いてスタブ自動生成

おおまかな手順は**図3**です。まず1段目、AP(アプリケーション)側からDB(データベース)へのインターフェース仕様書(以下、I/F仕様書)をExcelで書きます。

「DBへのI/F仕様書？」

「どんなパラメータを渡すとどんな結果が返ってくるか、を定義するわけ。詳しくはあとで説明するね」

そして、そのI/F仕様書からDBのスタブを自動生成します。自動生成はマクロでやります。スタブはDBのストアドプロシージャとして実装するもので、AP側からは本当のDB呼び出しと同様に使えます。で、**図3**の2段目ですが、このスタブを使ってアジャイル式にAPの画面を作り、フロントエンドの仕様を確定させます。ユーザはこの段階で、データはダミーですが実際に動く画面を使ったうえでいろいろな要望を出せますから、ウォーターフォール型よりもずっと早くUI/UX要件を固めることができます。

そして3段目、フロントエンドの仕様が固まったら、バックエンドを作ります。テーブル、ストアドプロシージャ、SQL文などを確定させていくわけです。

「それで……うまくいく、と？」

「ウチでは実際これでやってきましたんで」

「I/F仕様書ってどういうものを書くんですか？」

実例としては**図4**のようなものです。プロシージャ名とその機能概要、引数、戻り値、ダミーデータをそれぞれ書きます。これを使って**リスト1、2**のようなスタブを自動生成します。これを使って生成したスタブはストアドプロシージャになり、AP側からはCALL文で呼び出せます(SELECT文と同じ扱い)。本番時にはそのストアドプロシージャの中身を、実テーブルを使うように置き換えればいいので、AP側の修正は不要です。

「これは……SQLの知識なくても書けます？」

「そのとおーり!!　そこが大事なとこなんよ！」

▼図4　Excelで作るDBインターフェース仕様書

(a)プロシージャ名、機能概要

プロシージャ名	機能概要
TEST_PROC	名前と住所で顧客を検索する

(b)引数定義

項	引数名	区分	必須	型	テーブル名	フィールド名	比較演算子	備考
1	PARM1	IN		TEXT		NAME	LIKE	1つ目のパラメータ
2	PARM2	IN		TEXT		ADDRESS	LIKE	2つ目のパラメータ

(c)戻り値定義

戻り値			
項	フィールド名	型	備考
1	ID	NUMBER	主キーです
2	NAME	TEXT	名前
3	Bdate	DATE	誕生日
4	ADDRESS	TEXT	住所
5	TEL	TEXT	TEL
6	FAX	TEXT	FAX

(d)ダミーデータ定義

ID	NAME	Bdate	ADDRESS	TEL	FAX
1000	山田 太郎	1970/11/7	大阪府大阪市住之江区1	06-6666-7777	06-6666-8888
1001	佐藤 二郎	1970/11/8	東京都大田区蒲田2	06-6666-7777	06-6666-8888
1002	鈴木 一郎	1970/11/9	愛知県名古屋市中区4	06-6666-7777	06-6666-8888

スタブを生成したあと、AP側からは次のようなコードで呼び出す

```
CALL TEST_PROC("佐藤","東京")
```

DB関連の設計／開発工程を合理化できる

図1のようなウォーターフォール型でよくあるやり方をDB屋から見ると、こんな欠点があります。

①テーブル設計がなかなか確定しない
②全工程でDBのスキルが必要

①については先ほど書いたとおり、ユーザが実際に使えるのがあとになってしまうのが原因なので、プロトタイピングをすることによって解決できます。

②はAP側から生のSQLを使うことによって起きる問題です。当連載でもこれまで書いてきましたが、SQLは通常フロントエンド側で使われる言語とは設計思想が違うため学びにくく、苦手としているエンジニアが多いものです。にもかかわらず生のSQLを使うと、苦手だからシンプルなSQLで済まそうとして「ぐるぐる系」と呼ばれる無駄に複雑な手続き型コードを書くようになり、開発工数もかさむし性能も落ちるしバグも出やすくなる結果を招きます。

ではどうしたら良いのか？　その答えの1つが本稿で紹介するExcelインターフェース仕様書→スタブ生成方式です。

大道君が言うように、**図4**のようなI/F仕様書はSQLの深い知識がなくても書けます。スタブは自動生成できるので、フロントエンドの開発工程にはDBのプロはいりません。それがある程度進んでI/F仕様書がそろってくると、「AP側がどのようにDBを使うのか」が具体的にわかります。生のSQLではパラメータと戻り値がSQLの中に埋もれてしまってわかりにくくなりますが、Excelで分離して書いてあれば一目瞭然です。それをDBのプロが見れば、適切なテーブル設計をしたうえで合理的なSQL文を作ることができるわけです。

「DBのプロって、なかなかいないですからね……」

「僕も勉強はしていますけど、まだまだですし……」

「だから、DBとAPの開発をきっちり分離したほうがいいんですよ。これ

▼リスト2　スタブ（プロシージャ）

```
DROP PROCEDURE IF EXISTS TEST_PROC;

DELIMITER $$
/* ■□■□ TEST_PROC    名前と住所で顧客を検索する    */
CREATE PROCEDURE TEST_PROC
    (
    PARM1 TEXT  -- 1つ目のパラメータ
    , PARM2 TEXT  -- 2つ目のパラメータ
)
BEGIN

    -- 本番時は以下のSQLを修正し、このコメントを削除する。
    SELECT
        ID AS ID
        , NAME AS NAME
        , Bdate AS Bdate
        , ADDRESS AS ADDRESS
        , TEL AS TEL
        , FAX AS FAX
    FROM xTEST_PROC_VIEW
    WHERE
        1 = 1
        AND (NAME LIKE PARM1 OR PARM1 IS NULL)
        AND (ADDRESS LIKE PARM2 OR PARM2 IS NULL)
    ;

End
$$
DELIMITER ;
```

▼リスト1　スタブ（ダミーデータ定義）

```
CREATE OR REPLACE VIEW xTEST_PROC_VIEW AS
SELECT '1000' AS ID, '山田 太郎' AS NAME, CONVERT('1970/11/7', date) AS Bdate, '大阪府大阪市➡
住之江区1' AS ADDRESS, '06-6666-7777' AS TEL, '06-6666-8888' AS FAX
UNION ALL SELECT '1001', '佐藤 二郎', CONVERT('1970/11/8', date), '東京都大田区蒲田2', '06-6666-➡
7777', '06-6666-8888'
UNION ALL SELECT '1002', '鈴木 一郎', CONVERT('1970/11/9', date), '愛知県名古屋市中区4', '06-6666-➡
7777', '06-6666-8888'
;
```

はそれを可能にする方法の1つなんです」

ほかにはO/Rマッパーを使うことによっても似た効果が得られますが、O/Rマッパーは、手続き型(オブジェクト指向)がSQL(RDB)を取り込む形で作られています。そのため手続き型から見たら効率が良くなりますが、DB側の性能については考慮されておらず、RDB本来の性能を出してくれるわけではありません。DB側にインターフェースを作り、そのインターフェースのマッピングをするためにO/Rマッパーを利用すれば、手続き型とRDBの両方の性能を十分に活かせます。

なぜDBのI/F仕様書を書かないの?

この手法の1つのカギが、DBのI/F仕様書を書くことです。

「こういうドキュメント、書いたこと/見たことありますか?」

「ありませんね……」

「私も、ウチ以外で作ってるところを見たことがありません」

DBはAPに対してデータ管理機能を提供するプラットフォームです。プラットフォームには普通APIがあり、インターフェース仕様書があります。そしてAPIはAP側の必要に応じてだんだんと進化していくものです。それはたとえばJavaのAWTやSwingをイメージしてもらえばわかると思います。

ところが、なぜかDBについてはインターフェース仕様書を書かずに使う会社がほとんどです。RDBMSはSQLとストアドプロシージャでかなり複雑な機能を提供できるので、「テーブルはこうなってるから、あとは自由に使ってや」とSQLを生で使わせるのではなく、APにとって使いやすい形のAPIを提供してあげたほうが、トータルでの開発効率が良くなります。

本稿で紹介したようなI/F仕様書を書けば、「ああ、このAPは、こんな形でDBを使いたいんだな」ということがわかります。わかったら、それに合わせてDBのエキスパートがテーブル設計をし、どんなに複雑であっても最適なSQL/ストアドプロシージャを組んでAPIとして提供できます。そのためにはUI/UXのプロトタイピングを先行させ、あとでテーブル/SQL/ストアドプロシージャの設計/開発をしたほうが良いわけです。

「なるほど……」

「ということです。こんな形でやってみませんか?」

「僕はやってみたいです」

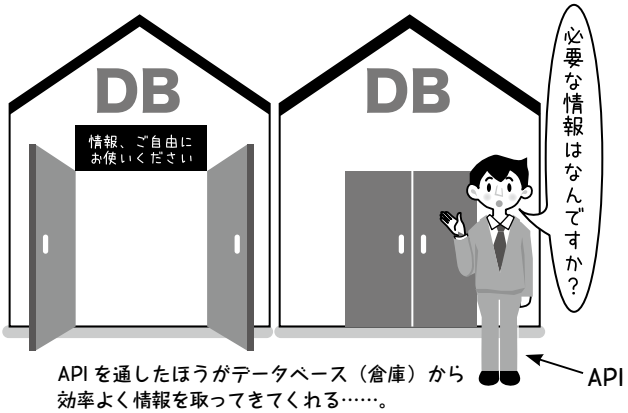
「お客さんの了解も取らなアカンので、今すぐ結論は出せませんけど……」

「要件定義は『張りぼて』ですが動くプログラムで行います、と言われて嫌がるお客様は少ないと思います」

「そやな……わかりました! その方向で話してみましょう!」

そうして始まったお料理レシピ投稿サイトについてもまたいろいろと性能問題が起きるのですが、それについてはまたの機会に書くことにします。SD

DBをAPIを通して利用させるイメージ(右)と
DBを直接利用させるイメージ(左)



APIを通したほうがデータベース(倉庫)から
効率よく情報を取ってきてくれる……。

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8回 Google I/O 2016で注目すべき開発環境の進化

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

fkm
日本Androidの会

嶋 是一（しま よしかず）
NPO日本Androidの会
理事長

Google I/O

今年もGoogle I/Oが開催されました（**写真1**）。Googleが開催する開発者向けイベントのGoogle I/Oは、毎年Googleからの新しい製品や技術の情報が発表される場所であり、エッジの効いた開発者にとって重要な情報元となるイベントです。今年は例年開催されていたサンフランシスコではなく、なんとGoogle本社のあるマウンテンビューで開催されました。しかも、まるで夏フェスのような屋外での開催となり、多くの人を驚かせました。

キーノートでは数々の発表が行われました。Google HOMEという宅内家電を連携させて音声認識で動かすデバイスや、コミュニケーションツールのAlloとDuo、Android Nの新しいプレビュー情報、Android Wear 2.0の発表、アプリをインストールせずに実行することができるAndroid Instant Appsなどの新技術が紹介されています。

▼写真1　Google I/O 2016でのワンショット

それら中でも開発者としてインパクトが大きいのが、Android Studio 2.2のリリースと、バックエンドシステムであるFirebaseのAndroid対応でしょう。今回はGoogle I/Oに参加したfkm氏から、この2つを掘り下げて紹介してもらいます（著：嶋是一）。

Android Studio 2.2 Preview

Android Studio 2.1の正式版が4月26日にリリースされたばかりですが、Google I/O 2016では次バージョンとなるAndroid Studio 2.2のPreview版が発表されました。主な変更点を**表1**に示します。とくに、新しいレイアウトエディタとConstraintLayoutはAndroidアプリ開発の方法を大きく変えるものとなりそうです。

新しいレイアウトエディタとConstraintLayout

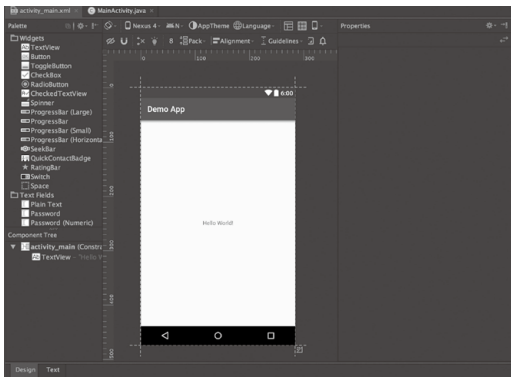
Android Studio 2.2で追加された機能のうち、一番大きなものはこの新しいレイアウトエディタとConstraintLayoutでしょう。ConstraintLayoutはサポートライブラリとして提供され、API Level 9以上で利用可能なレイアウトです。特徴は子Viewの配置を制約[注1]を基に決定する

注1） ここでいう「制約（Constraint）」とは「指定した条件」といった意味合いです。

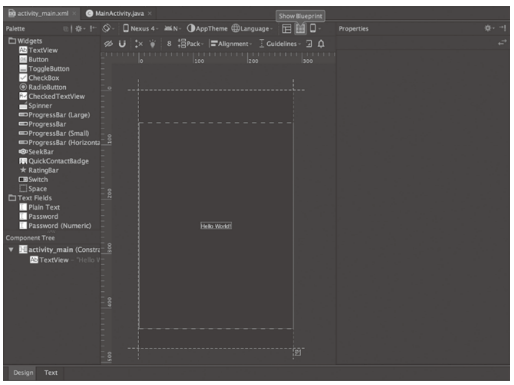
▼表1　主な変更点

デザインに関する変更点	
新しいレイアウトエディタ	設計図モードが追加され、後述のConstraintLayoutのための機能強化が行われた
ConstraintLayout	子Viewを、指定した条件を満たすように配置するレイアウト。レイアウトのネストを浅くしたまま、フレキシブルな配置が可能となった
レイアウトインスペクタ	アプリ実行中のレイアウト状態を取得し、レイアウトの階層や属性値を確認できるようになった
開発・ビルドに関する変更点	
Firebaseプラグイン	Firebaseの機能（Analyticsなど）をすぐアプリに実装できるようになった
サンプルブラウザ	サンプルコードの確認が容易になった
C++サポートの強化	デバッガがJavaとC++の両方を同時に扱えるようになった
Jackコンパイラの強化	アノテーション処理がサポートされた
Merged Manifestビューワ	マージ後のAndroidManifestを確認するためのビューワが追加された。これにより、どのライブラリがどのパーミッションを追加したかなどの判別が容易になった
テストに関する変更点	
Espressoテストレコーダー	UI操作を記録し、Espressoテストコードを生成する機能が追加された
APKアナライザー	apkファイルに含まれるクラスやリソースなどがサードパーティのツールなしで確認できるようになった

▼図1　新しいレイアウトエディタ



▼図2　設計図（blueprint）モード

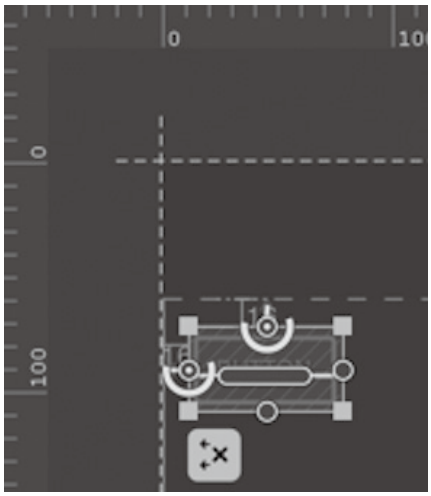


点で、iOSのAuto Layoutによる Viewのレイアウトに近いものになっています。

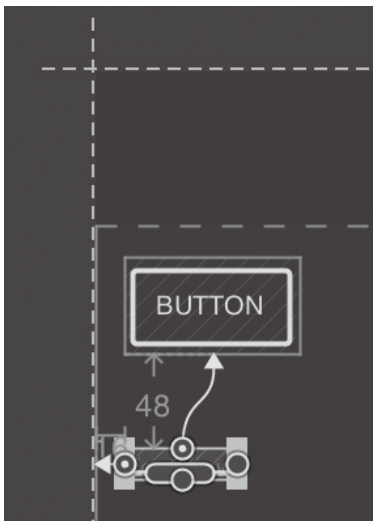
「iOSのAuto Layoutのようなもの」と書くと、「使うのが非常に難しいのでは？」と心配する方もいると思います。Android Studio 2.2では、レイアウトエディタもConstraintLayoutによる変更に合わせて大幅に強化されました。これまでのレイアウトエディタは使い勝手が良いとは言えず、レイアウトXMLを直接編集したほうが思ったとおりのレイアウトを実現しやすいという力不足の面もありました。そのため、UIデザイナーもXMLの文法を習得する必要があり、AndroidアプリのUI設計はハードルの高いものとなっています。Android Studio 2.2ではこの問題を解決し、制約に基づいたレイアウトを実現するための新しいレイアウトエディタが導入されます。もちろん、ConstraintLayoutを用いない従来のレイアウトに対するサポートも強化されています。

図1はAndroid Studio 2.2の新しいレイアウトエディタの画面です。「Language」の2つ右にあるボタンをクリックすると、**図2**のように

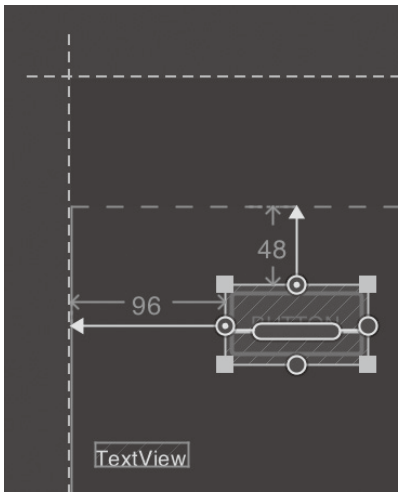
▼図3　アニメーションで上端と左端に制約が追加されようとするのを表す



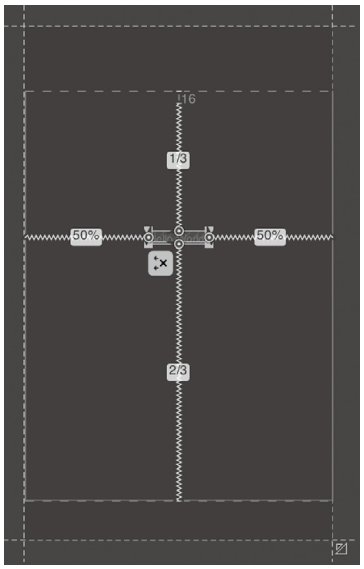
▼図4　制約は矢印で表される



▼図5　ボタンをドラッグで移動させると、下のTextViewも制約にしたがって移動する



▼図6　上から1/3の位置に配置してみる



設計図モード(blueprint mode)になります。

試しに画面の左上にボタンを配置してみましょう。するとAndroid Studioは「ボタンの左端は親Viewの左端から16dpにする」という制約と、「ボタンの上端は親Viewの上端から16dpにする」という制約を自動で設定してくれます。このとき、(誌面では伝わりづらいですが)**図3**のようにボタンの該当する位置にアニメーションで「ここに制約を追加するよ」を教えてくれます。

次に、このボタンの下に48dpほど間隔をあけてTextViewを配置してみましょう。すると、Android Studioは左端に関する制約と、「このTextViewはボタンの下に配置し、マージンは48dp」という制約を自動で追加します。追加された制約は**図4**のように矢印で示され、制約の有無が一目でわかるようになっています。

ConstraintLayoutは制約を基にレイアウトを行うので、たとえばボタンをドラッグし、**図5**

▼表2 Firebaseに追加された主な機能

機能	説明
Analytics	アプリ内分析機能。分析結果からユーザセグメントを作成し、Push通知などのサービスに利用することが可能
Cloud Messaging	Push通知サービス。GCM(Google Cloud Messaging)がFCM(Firebase Cloud Messaging)に改名された。iOSのAPNsもサポートされている
Notification	Cloud Messagingの機能を用いて、指定した時刻やユーザセグメントに通知を送るサービス。開封率の測定なども行ってくれる
Authentication	認証機能。Googleだけでなく、Facebook/Twitter/GitHubによる認証もサポートされている
Remote Config	アプリ内の設定をサーバ側で制御するためのサービス。たとえば新機能を最初は一部のユーザにのみ解放するといった使い方ができる
Test Lab	Androidアプリのテスト支援サービス。Cloud Test Labと呼ばれていたサービスがFirebase Test Labに改名された
Crash Reporting	アプリクラッシュ時のレポート収集サービス

のように移動させた場合には、下のTextViewも制約にしたがって自動で移動します。

ここまでの説明ではRelativeLayoutとの違いがないように見えますが、ConstraintLayoutでは「バイアス」という制約を用いることができます。**図6**はバイアスを用いて、TextViewのy座標を親Viewの上から3分の1の位置に配置した例になります。

Firebaseの機能強化

Firebaseとは、2014年10月にGoogleが買収したBaaS(Backend as a Service)です。買収された当時はデータのリアルタイム同期を得意とするサービスでしたが、Google I/O 2016で多くの機能を追加したことが紹介されました。**表2**に追加された主な機能を示します。もちろん、リアルタイムデータベースやストレージ機能は引き続き提供されます。

表2で示した機能はどれもモバイルアプリ開発において重要な機能です。そしてその多くが無料で利用可能なため、多くのアプリがFirebaseの機能を利用することになると思われます。

Cloud MessagingとNotificationを使ってみた

Cloud MessagingとNotificationの機能を確認するため、新規AndroidアプリにFCMを導入し、Notificationを使って通知を送ってみました。結論から先に言うと、Javaプログラムを1行も書かずに通知機能を実装することができました。

▼図7 Firebaseにプロジェクトを作成する

まず、Android Studioで新規アプリプロジェクトを作成します。Application IDをcom.mokelab.sddemoとし、最もシンプルなEmpty Activityのテンプレートでプロジェクトを作成しました。

次に、Firebaseのコンソール[注2]で新規プロジェクトを作成します(**図7**)。プロジェクト作成にはGoogleアカウントが必要となることを

注2) https://console.firebase.google.com/

お忘れなく。プロジェクトを作成した後は「AndroidアプリにFirebaseを追加」をクリックします。ここで、先ほど新規作成したAndroidアプリのApplication ID(パッケージ名)を入力します(**図8**)。「アプリを追加」ボタンをクリックすると、自動でgoogle-services.jsonのダウンロードが始まるので、Androidアプリプロジェクトのappフォルダに入れてください。これでFirebaseコンソールでの準備は完了です。

google-services.jsonの配置が完了したら、Androidアプリプロジェクトに設定項目を追加していきます。まず、プロジェクト直下のbuild.gradleを開き、buildscriptのdependenciesに“classpath 'com.google.gms:google-services:3.0.0'”を追加します(**リスト1**)。

次に、app配下のbuild.gradleを開きます。まず、dependenciesに“compile 'com.google.firebase:firebase-messaging:9.0.2'”を追加。また、build.gradleファイルの最下行に“apply plugin: 'com.google.gms.google-services'”を追加します(**リスト2**)。本稿執筆時点ではライブラリのバージョンは9.0.2でしたが、実際に導入する際は最新バージョンを指定してください。ライブラリ名とバージョンはFirebaseのAvailable libraries[注3]で確認できます。

最後に、build.gradleファイルを編集してい

▼図8　AndroidアプリにFirebaseを追加する

注3) https://firebase.google.com/docs/android/setup#available_libraries

▼リスト1　プロジェクト直下のbuild.gradleを編集する

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.0-alpha2'
        classpath 'com.google.gms:google-services:3.0.0' // この行を追加

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

▼リスト2　app配下のbuild.gradleを編集する

```
apply plugin: 'com.android.application'

android {
    // 中略
}

dependencies {
    // 中略
    compile 'com.google.firebase:firebase-messaging:9.0.2' // この行を追加
}

apply plugin: 'com.google.gms.google-services' // ファイルの最下行にこの行を追加
```

▼図9　最初のメッセージを送信

▼図10　メッセージ文とラベル、ターゲットとしてアプリを設定

るので、Android Studioの「Sync Project with Gradle Files」で編集内容をAndroid Studioに反映させましょう。このとき「firebase-messagingが見つからない」という旨のエラーメッセージが表示されたら、SDK Managerを起動し、Google Repositoryを更新してください。Firebaseのライブラリは jCenterではなく、Google Repositoryで提供されているためです。

Androidアプリ側の準備は以上になります。実機でアプリを起動し、端末のバックボタンでバックグラウンド状態にします。この状態でFirebaseのコンソールを開き、左ナビゲーション内の「Notifications」を選びます。すると、**図9**の画面が表示されるので「最初のメッセージを送信」をクリックします。

次に、通知メッセージ文とラベルを入力し、ターゲットとしてアプリを選択します(**図10**)。ラベルはFirebaseコンソール内で識別するためのものなので適当なものを入力してかまいません。

▼図11　通知が表示された

詳細オプションで通知音や有効期限などを指定することもできます。入力が完了したら「メッセージを送信」ボタンをクリックすることで、先ほど起動したアプリに対し、通知が表示されます(**図11**)。なお、アプリがフォアグラウンドにいる場合はライブラリの機能で通知は表示されないので、送信を試す場合はバックグラウンドに移動させておいてください。

まとめ

FirebaseのAndroid対応が発表されたことにより、Androidを用いた開発が、バックエンドまで含めてますます行いやすくなった印象です。Google I/Oでは、Android Nの「Androidの自身の進化」も発表されていますが、それ以上に開発者のための「開発しやすさ」の進化のほうが大きい進化を遂げていると感じます。また来年のGoogle I/Oに向けてこの傾向は進むでしょう。この動向を追える開発者のみ、進化の御利益を得ることができます。楽しんで開発をしましょう。SD

Sphinxで始めるドキュメント作成術

川本 安武　KAWAMOTO Yasutake　Twitter @togakushi

第17回 ドキュメントの種類に応じた表現ができる「Sphinxドメイン」

ドメインとは

Sphinxには、プログラム言語ごとにクラスや関数のリファレンス(説明文)を記述するために、「ドメイン」と呼ばれるしくみが用意されています。ドメインは、Sphinxが提供するディレクティブとロールを分野ごとに分けて整理し、まとめたものです[注1]。たとえば、Pythonの関数説明やクラス説明はPythonドメインを、C言語の説明はC言語ドメインを使用してそれぞれ記述します。

ドメインが存在しないことによる問題点

ドメインが導入される前のSphinxには、次のような問題がありました。

Sphinxの開発当初は、クラスを説明するclassディレクティブや、関数を説明するfunctionディレクティブなど、一般的な用語を使用したディレクティブはすべてPythonのために提供されていました。その後、C言語向けのディレクティブが追加されることになりましたが、cfunctionやctypeなどの名前で追加されています。これらのディレクティブは、Python向けのディレクティブとの名前の重複を避けるため、「c」というプレフィックスが付けられています(**リスト1**)。同様に、ほかの言語のドキュメントを書くにはディレクティブやロールの名前の衝突を避けるための工夫が必要な状態でした。

この問題を解決するためにドメインというしくみが導入されました。ドメインによって名前空間が分けられ、どの言語向けに作られたディレクティブでもclassやfunctionなどの一般的な用語を使用できるようになりました(**リスト2**)。Sphinxでは、Pythonのほかには C言語、C++、JavaScript、reStructuredText(以下、reST)のドメインを組み込みドメインとして提供しています。

ドメインがもたらすメリット

名前空間が分けられたことにより、新しい分野の説明を記述するためのディレクティブやロールを追加することが容易になりました。ドメイン単位での拡張が可能になり、言語リファレンス以外の用途で利用することも簡単になっています。Ruby、Goなどの各種言語向けのドメインや、HTTP(REST API)や運用ドキュメントなど、プログラム言語以外の分野のドメインなどが公開されています。

ドメインを使用したドキュメントの記述

ドメインに属するディレクティブとロールは「ドメイン名:ディレクティブ」のように、コロン(:)で区切って記述します。Pythonドメインのドメイン名はpyですので、ディレクティブは「`.. py:function:: keyword`」(**リスト3**)、ロールは「`:py:func:`keyword`」(**リスト4**)となり

注1) ドメインに属さないディレクティブもあります。

ます。

ドメインによって提供されるディレクティブやロールは、その分野の説明のためだけに追加されます。たとえば、関数の説明を書くためのディレクティブでは、その言語の文法規約に従って解釈が行われます。Sphinxでは通常、*は強調マークアップとして解釈されますが、引数に*が含まれていてもエスケープすることなく記述できます。

また、ドキュメントを変換した際にもその言語の文法を解釈します。**図1**の例では、関数名の識別(❶)、モジュール名の識別(❷)、引数やキーワードの識別(❸)、引数やキーワードの説明(❹)が行われています。クロスリファレンスを生成する際には、関数名の最後に引数を含まない形で括弧を追加します(**図2**の❺)。

ドメインで提供されるディレクティブとロールは、それぞれのドメインによって異なりますので、各ドメインのドキュメントを確認してください。

▼リスト1　関数用ディレクティブの記述例（ドメイン導入前）

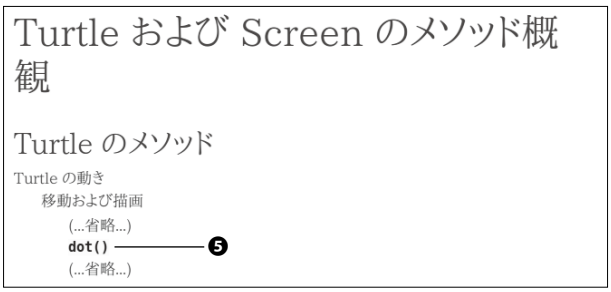
```
Pythonの場合

.. function:: sqrt(value)
   ↑functionというディレクティブを使う

C言語の場合

.. cfunction:: sqrt(value)
   ↑cfunctionというディレクティブを使う
```

▼リスト2　関数用ディレクティブの記述例（ドメイン導入後）

```
Pythonの場合   どちらでもfunctionというディレクティブが使える
.. py:function:: sqrt(value)

C言語の場合

.. c:function:: sqrt(value)
```

▼図1　リスト3をHTMLに変換した様子

▼図2　リスト4をHTMLに変換した様子



▼リスト3　ドメインを利用したディレクティブの記述例

```
================================================
:py:mod:`turtle` --- Tkのためのタートルグラフィックス
================================================

.. py:module:: turtle
   :synopsis: Tkのためのタートルグラフィックス

.. py:function:: dot(size=None, *color)

   :param size: 1 以上の整数（与えられる場合には）
   :param color: 色を表わす文字列またはタプル

   直径 *size* の丸い点を *color* で指定された色で
   描きます。
   *size* が与えられなかった場合、pensize+4 と
   2*pensize の大きい方が使われます。
```

▼リスト4　ドメインを利用したロールの記述例

```
Turtle および Screen のメソッド概観
==================================

Turtle のメソッド
-----------------

Turtle の動き
   移動および描画
      | (...省略...)
      | :py:func:`dot`
      | (...省略...)
```

デフォルトのドメイン名

ドメインを利用してC++ライブラリのドキュメントを書いている場合、cppドメインのディレクティブやロールを中心にして記述していくことになります。そのため、cpp:functionのように何度もcpp:と書くことになります。そこで、デフォルトドメインを設定すると、ディレクティブ、ロールに指定するドメイン名を省略して記述でき、書き手の負担を減らせます。

デフォルトドメインは次の2通りの方法で設定します。

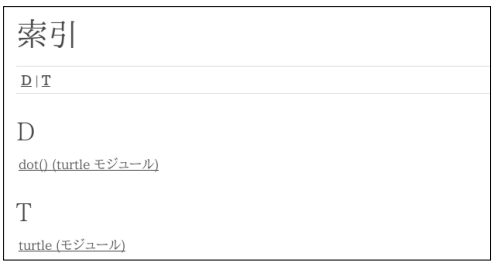
■ primary_domain

primary_domainは、conf.pyに設定するオプションです。

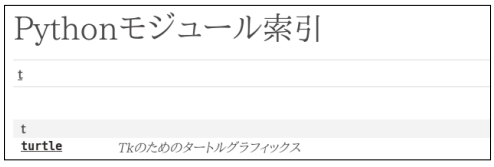
ドメインは1つのプロジェクトの中で、いくつもの種類が利用できます。primary_domainには、複数あるドメインの中から主となるドメインを指定します[注2]。ディレクティブなどを記述する際にドメイン名が省略されている場合は、この設定値がデフォルトドメインとして使用されます。primary_domainのデフォルト値はPythonドメインのpyです。sphinx-quickstartが生成するconf.pyにはprimary_domainが記載されていませんので、追記してください。

注2）ドメインを1つしか利用していない場合でも、primary_domainは指定できます。

▼図3　リスト3によって追加される全体インデックス



▼図4　リスト3によって追加されるドメイン専用インデックス（Pythonドメインによるモジュールインデックス）



■ default-domainディレクティブ

default-domainディレクティブは、ページ内のデフォルトドメインについての設定を行います。このディレクティブで指定したデフォルトドメインは、primary_domainより優先されます。

ドメインのインデックス（索引）

ドメインで提供されるディレクティブには、定義した用語を自動的に索引に追加するものがあります。**リスト3**で使用しているPythonドメインのfunctionディレクティブであれば、**図3**のようになります。

また、一部のドメインはドメイン独自の索引を作成します。たとえば、Pythonドメインでは、moduleディレクティブによってモジュールインデックスが作成されます（**図4**）。

Sphinxでは、indexディレクティブやglossaryディレクティブ、前述のドメインが提供するディレクティブによって追加される索引（全体インデックス）と、ドメイン専用のインデックスを生成します。それぞれのインデックスへのリンクは、HTMLテーマによって若干異なる位置にあったりしますが、基本的にはドキュメントの右上と右下にあります。なお、全体インデックスへのクロスリファレンスは、:ref:`genindex`で作成できます。ドメイン独自のインデックスへのクロスリファレンスはドメインごとに異なりますが、Pythonドメインのモジュールインデックスであれば、:ref:`modindex`で作成できます。

標準ドメイン

標準ドメインは、どのドメインにも属さない汎用的なディレクティブを集めたものです。標準ドメインの名前はstdです。標準ドメインは、デフォルトドメインに設定しなくても、ドメイン名を省略して記述できます。

標準ドメインのprogramディレクティブは、コマンドラインで利用するプログラム(コマンド)の説明を書く場合に使用します。optionディレクティブと組み合わせて使用し、そのプログラム(コマンド)のオプションの説明を記述できます(**リスト5、図5**)。全体インデックスにはプログラム(コマンド)と、そのオプションについて双方向で追加されます(**図6**)。

Sphinx拡張によるドメインの追加

ドメインはSphinxの拡張機能で追加できます。PyPIを検索すると多くの拡張ドメインが見つかります。PHPを始め、Go、Erlang、Scala、Common Lispなどの言語ドメインや、HTTPドメインのようにWeb APIのドキュメントを書きやすくするドメインもあります。

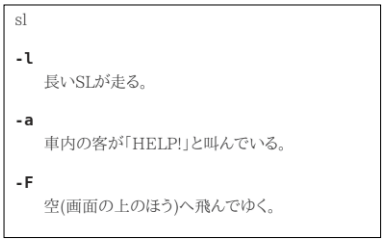
ドメインの追加は、過去の連載で紹介した拡張と同様にpipコマンドを使用します(**図7**)。

ここまでは言語リファレンスを作成するためのドメインばかりが登場しましたが、Sphinxでは言語リファレンス以外のドキュメントも書けます。次節では運用ドメインを例に、言語リファレンス以外のドメインの利用例を紹介します。

▼リスト5　標準ドメイン(programディレクティブとoptionディレクティブ)の使用例

```
sl

.. program:: sl

.. option:: -l

   長いSLが走る。

.. option:: -a

   車内の客が「HELP!」と叫んでいる。

.. option:: -F

   空(画面の上のほう)へ飛んでゆく。
```

▼図5　リスト5をHTMLに変換した様子



▼図6　リスト5によって追加される全体インデックス

COLUMN

全体インデックスの分割

全体インデックスは、conf.pyの設定で分割されたインデックスページも生成できます。html_split_indexオプションがTrueに設定されると、Sphinxはすべての索引が載っている全体インデックスとアルファベットごとにページを分けた全体インデックスの2通りを作成します。

また、ドキュメントをHTMLに変換する際に、html_use_indexオプションがFalseに設定されていると全体インデックスが、html_domain_indicesオプションがFalseに設定されているとドメイン専用のインデックスページが、生成されなくなります。html_domain_indicesオプションは、生成すべき索引のリストを持たせることもできます(HTMLだけではなく、EPUBやLaTeXにもインデックスに関するオプションが存在します)。

インデックスにある情報が多くなると逆に見にくくなるので、必要に応じて分割された全体インデックスも利用すると良いでしょう。

運用ドメイン

運用ドメインは、筆者が仕事上で利用する運用ドキュメントをSphinxで活用するために作成しました[注3]。筆者の職場環境では顧客に納品するドキュメントとは別に、内部で利用する目的で書かれたドキュメントが数多く存在します。これらのドキュメントはプロジェクトごとや担当者ごとに分かれて作成、管理されています。

これは意図的に分けたのではなく、プロジェクト固有の情報が記載されていたり、担当者がドキュメント化する必要があると感じる部分が異なっていたりするため、自然と分かれて管理されるようになりました。このような状況ですので、プロジェクトAで作成したOracleの導入手順書と、プロジェクトBで作成したOracleの導入手順書が存在したりしています[注4]。

内部で使う技術的なドキュメントはすべてSphinxで管理すると決めたとき、前述のように「同じことについて書かれているドキュメント」が大量に発掘されました。さらに「プロジェクト固有の情報が記載されている」ため、一ヵ所に集約して管理するには不都合なことも多く、また汎用的な内容になるように修正する必要もありました。ひとつひとつドキュメントの内容を精査して重複を排除したり、汎用的に利用できるように書き換えたりするには多くの時間が必要な状況でした。

この状況から抜け出すために、まず、どこに何があり、何が書かれているのかを把握する必要がありました。筆者はこれを運用ドメインを作成することで解決しました。

運用ドメインではinstall、setting、command、howtoというディレクティブが追加されます。これらのディレクティブを使用することで、そのドキュメントが「インストール手順」や「設定手順」、「コマンドの使い方」「その他の雑多な手順」であることを明示します。また、ディレクティブによってそれぞれの専用インデックスが作成されます。つまり、運用ドメインによってドキュメントの種類が分類され、Sphinxで管理した際にインデックスで整理されることになるのです。

ドキュメントのreST化をするにあたって、まずはメンバーにreSTの文法を覚えてもらいました。運用ドメインで記述する内容をテンプレート(**リスト6**)として用意し、ドキュメントにはそれを追加してもらいました。

インストール手順はinstall、コマンドの使い方を述べたものはcommand、その他雑多なものはすべてhowtoにするという、大雑把なルールで既存のドキュメントのreST化を進めました。幸いテキストで書かれたドキュメントが多く、reST化にはそれほど苦労はしませんでしたが、時折、体裁が大きく崩れ、reSTとしてSphinxが

注3) インストール方法と設定方法は、https://bitbucket.org/togakushi/sphinxcontrib-operationdomain/を参照。

注4) この2つのドキュメントの差分はインストール対象のホスト名が違う、作業に使うSSHクライアントやユーザ名が違う、といった程度なのです！

▼図7　pipコマンドでHTTPドメインを追加する例

```
$ pip install sphinxcontrib-httpdomain
（..中略..）
Collecting sphinxcontrib-httpdomain
  Downloading sphinxcontrib_httpdomain-1.5.0-py2.py3-none-any.whl
（..中略..）
Installing collected packages: sphinxcontrib-httpdomain
Successfully installed sphinxcontrib-httpdomain-1.5.0
```

▼リスト6　ドキュメントを整理するためのテンプレート

```
xxxxxx導入手順書
================

.. op:install: xxxxxx導入手順書
   :synopsis: このドキュメントに書いてある概要を超簡単に。省略可能。(例:xxxxxxのインストール)
   :platform: RHEL,Solarisなど、特定プラットフォーム向けの内容なら、対象を書く。省略可能。

   :書いた人: 誰が書いたか書く(?は必ず1行空ける)
   :書いた日: いつ書いたか書く(わからなければ空欄でよい)
```

処理できないものもありました。そのようなものは、時間短縮のためにreST化する前のドキュメントを丸ごとliteralincludeで取り込むという乱暴な変換も行っています。

このやり方で、プロジェクトごとに分かれたディレクトリ構成のままドキュメントをreST化しました。運用ドメインによってドキュメントが分類され、それぞれの分類ごとに専用インデックスが作成されるので、どのプロジェクトでどのようなドキュメントが作成されたのかが把握できるようになりました。

重複が多いものは、利用頻度が高い傾向にあるドキュメントですので、優先的に汎用的なドキュメントへ書き直していきます。汎用的になったものは、プロジェクトごとのディレクトリとは別に管理し、構造化を進めています。前回の本連載で紹介された運用ドキュメントの部品化までは進められていませんが、運用ドメインとSphinxによって、ドキュメントを探す時間、書く時間を減らすことに成功しています。

まとめ&次回予告

ドメインで提供されるマークアップ(ディレクティブとロール)により、ドキュメントで表現できることが増えます。Sphinxはドキュメントを汎用的に書くことができるツールですが、ドメインを使用すれば目的に特化した専用のドキュメントも作成できます。

次回はPythonコードから自動的にリファレンスドキュメントを生成する、autodoc機能について紹介します。SD

COLUMN

anyロールとデフォルトロールによるクロスリファレンス

Sphinxには指定されたキーワードがref、doc、term[注A]やドメイン(組み込みドメイン、拡張ドメインを含む)などのロールとして解釈できるかを自動的に探索して試すanyロール[注B]があります。

anyロールは、「:any:`キーワード`」と記述するだけでキーワードがラベルなのかrstファイルを指しているのかなどを自動的に判別し、クロスリファレンスを生成します。指定されたキーワードがロールとして解釈できない場合や、複数の候補が見つかった場合には警告を出力します。

また、ロールに関する設定としてデフォルトロールがあります。デフォルトロールの設定は、default_role(conf.pyで指定)[注C]、または、default-roleディレクティブ(reST内に記述)[注D]で行います。デフォルトロールを設定しておくと、「`keyword`」のように「:ref:」や「:doc:」などを省略して記述した際に、デフォルトロールが指定されているものとして解釈します。default_roleはプロジェクト全体、default-roleディレクティブは記述されているreST内で有効です。

この2つを組み合わせ、デフォルトロールにanyを指定することで、クロスリファレンスを生成するためのロールを省略でき、書き手の負担を大幅に減らせます。

ただし、Sphinx-1.4.2未満のバージョン[注E]をPython 2.7で利用している環境では、anyロールに日本語が指定された場合、UnicodeEncodeErrorが発生してドキュメントの変換が失敗します。これは、組み込みドメインのC++ドメインが日本語のロールを受け付けないために発生します(Sphinx-1.4.2で修正済み)。また、使用している拡張ドメインによっては、日本語を受け付けずに同様の問題が発生する可能性があります。UnicodeEncodeErrorが発生した場合は、明示的にロールを指定してエラーを回避し、ドメインの作成者に報告してあげてください。なお、Python 3ではこの問題は発生しません。

注A) refは指定されたラベルへリンクするときに使うロール。docは指定したファイルパスにリンクするときに使うロール。いずれも本連載第2回(本誌2015年5月号)を参照。termはglossaryディレクティブで作った用語集へリンクするときに使うロール。本連載第5回(本誌2015年8月号)を参照。

注B) http://www.sphinx-doc.org/ja/stable/markup/inline.html#cross-referencing-anything

注C) http://www.sphinx-doc.org/ja/stable/config.html#confval-default_role

注D) http://docutils.sourceforge.net/docs/ref/rst/directives.html#default-role

注E) 執筆時点(2016年6月)の最新版はSphinx-1.4.4。

書いて覚える Swift 入門

第17回　WWDC2016の誤算

Writer 小飼 弾（こがい だん）　twitter @dankogai

静かなること WWDC2016のごとく

フロリダ銃乱射事件[注1]の犠牲者への黙祷から始まった今年2016年のWWDCは、例年にもまして静かでした。ハードウェアはおろか、ソフトウェアも毎年恒例のOSアップデートを除けば「新製品」はなし。macOSはiOSが10になることを考えれば、OS X（オーエステン）という名前をそのままにしておけないのは自明というものでしょう。Apple製品用のOSにマッチする正規表現も`((watch|tv|i)OS|OS ?X)`だったのが`(watch|mac|tv|i)OS`となってずいぶんとすっきりしましたが、あくまでヴァージョンは"10.12"。灰色のバックグラウンドにアスキーアートのリンゴというロゴは、そんな地味なWWDCを実によく象徴しています。開発者ではないプレスの皆さんはさぞ退屈されたのではないでしょうか。

しかし本誌の読者にとって、WWDC2016はあくびしてスルーするにはあまりに重要な知見に満ちています。

Learn Different＝ Differencial Privacy?

その最たるのが、AppleのAIに対する姿勢です。キーノートでも"Deep Learning"というバズワードが、ご丁寧にもLSTM[注2]というアルゴリズムの名前まで含めて登場しましたが、重要なのは「Appleも取り組んでますよ」ということではなく、Appleは何をしないのかというメッセージです。

OS X改めmacOSではずいぶん前から標準搭載されていた顔認識が、iOS 10でついに標準装備されます。それも顔だけでなく背景なども含めた包括的な画像認識という形で。そこで使われるのがAIで、そのこと自体はすでにGoogle PhotoやFacebookなどを通して日常触れているであろう本誌の読者にとってはこれまた「何をいまさら」といったところでしょう。問題は「何をやるか」ではなく「どこでやるか」。

Appleは、端末にそれをやらせると言っています。クラウドではなくて。

これは2つの点においてかなり不利な選択です。1つは処理能力。クラウドであれば必要な処理能力を必要なときに必要なだけ増やせますが、端末はそうも行きません。すでにiPhone 6/SEやiPad ProがMacBookに匹敵する処理能力を備えている以上、現在のMacのPhotoアプリケーションに標準装備されている程度の画像認識処理であれば問題なくそれをこなしそうですが、同様のことをたとえば動画に対して行うとしたら？

しかし、それ以上に重要なのは、データの蓄積。プログラムの質はソースコードが書かれた時点で決定しますが、AIの質を決めるのは、デー

注1）https://ja.wikipedia.org/wiki/フロリダ銃乱射事件
注2）https://en.wikipedia.org/wiki/Long_short-term_memory

タの質量。どれほど優れた母と父のもとに生まれた赤子でも生まれたてでは母国語すら話せないように、どれほど計算資源を用意しようが生まれたてのAIはそれ以上に非力です。生まれで決まるプログラムに対し、育ちで決まるAIというのはその点においても「ウェットウェア」たる我々に似ているのですが、そのAIを育てるデータが「オレ」の分しかないのと「オレタチ」全員分あるのでは勝負にすらならないように感じられます。

しかしプライバシーを考慮に入れると、その印象は逆転します。Facebookでは他人が撮った写真に自分がタグづけされていることがしばしばあって、友人同士のパーティの写真だとかなり重宝するのですが、もし同じことが街角の監視カメラで行われていたとしたら？　GoogleもFacebookも当然のごとく「悪用はしない」と主張していますが、それを証明しようとしても悪魔の証明になってしまいます。何しろすでに彼らの手元でデータは解析されているのですから。ちなみに、単にデータを保管している場合は「盗み見していない」証明は簡単です。伝送系路とストレージが暗号化されていることさえ示せば良いのですから。クラウド**に**データを上げるのとクラウド**で**データを処理することには本質的な違いがあるのです[注3]。

パーソナルコンピューター製造販売会社として産声を上げたAppleは、今後も変わらずパーソナルということにおいて首尾一貫しています。

しかし、プライバシーというのは“All or nothing”なものなのでしょうか？　いくらクラウドAIが不安でも、パーソナルAIだけでは不便なのは前述のとおり。両者のいいところ取りはできないのでしょうか?

それを成そうというのが、“differencial privacy”。The Vergeは“probably the most bewildering part of Apple's WWDC Keynote[注4]”と言っていますが、とまどったのは筆者も同様です。微分プライバシー？　差分プライバシー？

1つの例として、文字変換を挙げます。かな漢字変換のクラウド化は今や日常茶飯事となっていますが、どうすれば文章をクラウド側にさらさずに実現できるでしょうか？

「誰が」変換しているのかは知らせずに、「何を」変換しているかだけ知らせて、その「何」だけを集めて統計処理し、変換中の「みんな」に候補全部を送ってしまえばいい。iOS 10には欧文からの絵文字変換も搭載されるのですが、まさにそのように実装されています。

プライバシーの本質は、「誰」と「何」の紐付けなのですから、この紐さえ切ってしまえば、「何」だけクラウド処理するようにすればいいのではないか。

> *differential privacy*[注5] *aims to provide means to maximize the accuracy of queries from statistical databases while minimizing the chances of identifying its records*

「記録自体を特定される可能性を最小化しつつ、統計的情報精度を最大化する」という言葉の定義に確かに合致しています。

Virtual Private Cloud ——もう1つの方法

誰かが特定できる情報はなるべく端末で処理し、誰かが特定できないようにできる情報は（differencial privacyで）クラウド処理する。それが、Appleのクラウドに対する方針ということでよさそうです。実にAppleらしい落としど

注3）「iTunes Matchはどうよ？」という読者は鋭い。確かに「パーソナルデータをクラウド処理」しているように見えます。が、マッチした楽曲自体は、楽曲の著作権保持者が使用権をライセンスしている意味でパーソナルなデータではなく、パーソナルなのはマッチしない楽曲と楽曲のリストというメタデータというのがポイントです。

注4）http://www.theverge.com/2016/6/17/11957782/apple-differential-privacy-ios-10-wwdc-2016

注5）https://en.wikipedia.org/wiki/Differential_privacy

ころではありますが、不満もあります。たとえばSiri。現在Siriは、iPhoneやiPadやApple TVやApple Watch（さらにmacOSからはMac）の持ち主以外の声にも反応してしまいますが、differencial privacyの観点からはそれは当然ということになります。声色を聞き分けられるとしたら、クラウドの中のSiriが「誰」を知っているということなのですから。

differencial privacy以外に、プライバシーとクラウドのいいとこ取りをする方法はないのでしょうか？

たとえば、こんな方法もありえます。クラウド処理したいユーザの問い合わせがきたら、その都度クラウド上でそのユーザ専用の仮想マシンインスタンスを立ち上げるのです。その際仮想マシンのイメージをユーザごとに暗号化しておけば、クラウド提供者はユーザのことをいっさい知ることなくユーザにクラウド資源を提供できます。

しかし理論的に可能なこの方法は、現在のマシン仮想化ではまだ高くつき過ぎるかもしれません。Siriに話しかける都度、仮想マシンが起動しては会話が終わるやいなやシャットダウンというのは確かに重過ぎるように感じます。しかし仮想マシンではなくコンテナであれば、セッションごとに起動しては終了しても十分な速度を確保できるかもしれません。実際Webブラウザでコードを実行する、いわゆるWeb REPLサービスの多くは、IBM Swift Sandbox[注6]を含めそのように実装されています。

しかし十分なパフォーマンスだけではこの方法を導入する十分な理由にはなりません。この方法は「こちら側」のAppleデバイスごとに「あちら側」にももう1台のデバイスを用意するようなものですから、クラウドという名の（見かけ上は）1台のデバイスを共有するよりはるかにコストがかかります。AWSやAzureやGoogle Cloud Platformはそれ自体がれっきとした商品なのに、基本「ハードウェアのおまけ」であるiCloudで現時点でそこまでできるかは疑問ですが、損益分岐点を下回るのは時間の問題でしょう。

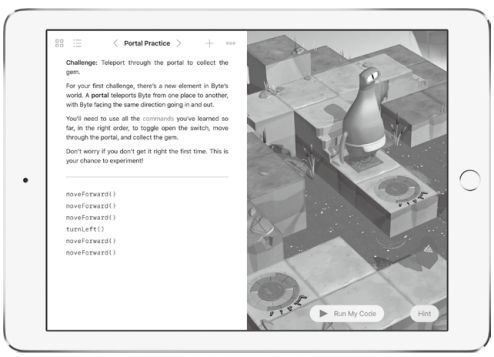
しかしどのような実装を採用するにせよ、その生い立ちから「パーソナル」を売り物にしてきたAppleが、「ネットワーク」を売り物にしている他者よりもプライバシーを気にするのは自然かつ必然な帰結だと筆者は感じます。

Swift Playground for iPad

「新製品なき」WWDCにおいて、もっともそれに近いのがSwift Playground for iPadでしょう（**図1**）。Xcode for iPadでないところが、実にAppleらしい。「タブレットでコードを書く」というのであれば、タブレットとPCを統合しようとした――そしてWindows 8で痛い目にあった――Windowsタブレットですでに実現しているとも言えます。C#が好きなら、今すぐSurface Bookを買ってXamarin三昧すべきだと筆者でも認めるのにやぶさかではないのですが、しかしそこにおける開発環境は、あくまでPCであってタブレットではないというのもたしかです。

では、Swift Playground for iPadは何なのか。「enchantMOON[注7]じゃん！」と読者の皆さんには

▼図1　Swift Playground for iPad



注6） https://swiftlang.ng.bluemix.net/#/repl
注7） http://enchantmoon.com/ja/

お馴染みの清水亮さんの心の声が聞こえてきました。ヴィジュアルな言語ではないSwiftを、ヴィジュアルなScratch[注8]やMOONBlock[注9]のように動かせたり、課題がiBooksのように本棚に登録されていたり、変数名をタップするとその変数のプロパティが補完されたり……。これなら、キーボードがない環境でもかなり使えそうです。

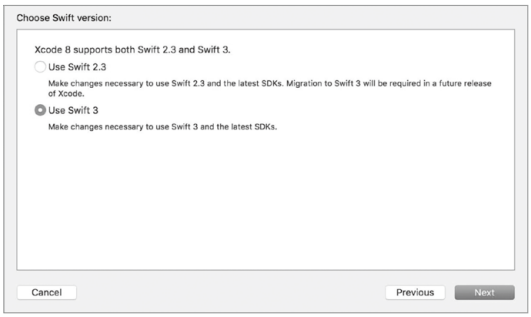
Swift Playground for iPadを見てしまうと、欲も出てきます。JavaScriptCoreに相当するSwiftランタイムがモジュール提供されないか、と。実はiBooksは、すでにJavaScriptの実行をJavaScriptCore経由でサポートしています。これと同様のことができれば、ブラウザで<script lang="swift">とかまではあと一歩ですし、shell scriptを超えた真のスクリプト言語にSwiftはなり得るでしょう。

Swift 2.3からSwift 3へ

なんだかSwift 3の話題に入る前に誌面が尽きてしまいそうですが、一番重要なことだけ今号で。Swiftの次のバージョンは、3だけではありません。2.3も出ます。Xcode 8は、どちらもサポートしています(**図2**)。

で、Swift 2.3とは何かというと、Swift 2.2 + New SDKとのこと。Swift 1からSwift 2への移行は問答無用だったのが、Swift 2から3への移行は、プロジェクト全体ではなくソースファイルごとに段階的に行えます(**図3**)。

▼図2　Xcode 8はSwift 2.3もSwift 3もサポートする



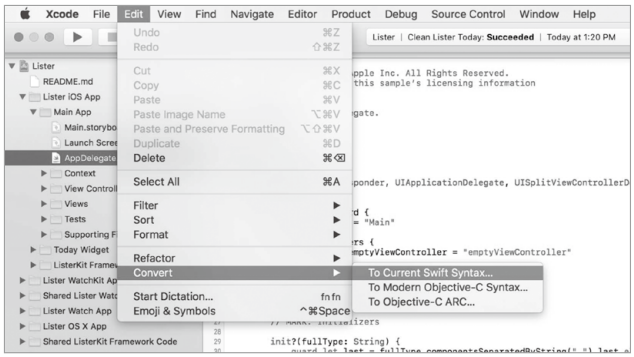
あと、Swift 2.2から#if swift(>=version)が使えるので、ソース内での切り替えも一応できます。

```
func xyz(x:Double, y:Double, z:Double)->Double {
    return x*y*z
}

#if swift(>=3)
xyz(x:2, y:3, z:4)    // more swifty
#else
xyz(2, y:3, z:4)      // like Objective-C
#endif
```

そのようにしたのは、Swiftの資産が往時より格段に増えたからでしょう。過去をバッサリ捨てるにはあまりにも。Xcode 7登場時にはいまだObjective-C主でSwift従だったのが、今やすっかり逆転している感があります。Swift 2のコードベースは、すでに技術的遺産となりつつあるのです。その遺産を段階的に継承できるという点で、Swift 2.3の発表は筆者にとって今回のWWDCでもっともうれしい誤算でした。

次回はいよいよSwift 3で何が変わったかを解説します。SD

▼図3　Swift 2からSwift 3へのコード変換



注8）https://scratch.mit.edu/
注9）http://moonblock.jp/

一歩進んだ使い方のためのイロハ

Vimの細道

第10回

mattn
twitter:@mattn_jp

Vimからgitを使い倒す

今回はVimからバージョン管理ツール「git」を便利に使う方法を解説。fugitive、gitv、agit、vim-gita、unite-gitといった、Vimの中だけでgitの操作を行えるパッケージをどんどん紹介していきます。気に入ったものはぜひインストールして試してください。

Vimでgitを使うには

GitHubの登場とともにgitの利用ユーザが爆発的に増え、今や開発者でなくてもgitを使うようになってきました。デザイナさんと協業する場合でも、最近は開発者・デザイナともにgitを使って作業し、各々が加えた変更をマージしながら作業しています。しかしもちろん、端末での作業が苦手な人もいます。そのため、世の中にはgitを扱うためのフロントエンドがいくつかあります。

・gitk（git付属）
・TortoiseGit
・SourceTree

ほかにもいくつかありますが、詳しくは公式サイト[注1]を参照してください。

このようにいろいろなツールがありますが、Vimにおいては、その使い手によって十人十色のやり方が存在します。まず、gitで管理されたプロジェクトをVimから扱う場合、次に挙げるようなパターンがあります。

①別の端末でgitコマンドを実行する
②:shでシェルを起動し、そこからgitコマンドを実行する
③Vimを終了してコマンドを実行する
④Vimからgitコマンドを実行する

まず①ですが、編集中のVimの状態を何も変えずにいられるという点では最高の方法だと思います。ただし画面を切り替える必要があり、編集中のVimで変更内容を確認しながらコミットメッセージを書くといった場合には向かないかもしれません。Linux Desktopできれいにウィンドウ分割したり、tmuxを使って端末を分割したりする人もいます。ただ、Vimのウィンドウを分割して複数のソースを一度に編集する場合には、その領域すらも惜しく感じてしまうこともあります。

②はVimのウィンドウがいったん消えてしまいますし、Vim本体がサスペンドしてしまうのであまりシームレスではありません。

③もウィンドウが消えてしまううえに、Vimの再起動が必要になるのでVimの起動が遅い人には向きません。

そこで④の出番ですが、みなさんはVimからgitの機能を使う方法をいくつご存じでしょうか？　実はいろいろ存在し、目的別に問題を解決する何種類もの方法があります。

注1）URL https://git-scm.com/downloads/guis

gitのためのパッケージ

fugitive

fugitive[注2]は、Vim界隈で知らない人はモグリと言われるくらい有名人のTim Pope氏が開発しているVimプラグインです。:Gitというそのままのコマンド名が用意されており、基本的にすべてのgitコマンドをVimから実行できます。git logを実行するには、次のようにコマンドします。

```
:Git log
```

コマンドは必要に応じてページャで結果が表示されます。

Gwrite

gitの変更をステージングするにはgit addを使いますが、fugitiveでは:Gwriteコマンドを使います。少し理解しづらいかもしれませんが、ファイルに対して書き込むコマンドが:writeであり、gitのステージに書き込むのが:Gwriteと覚えると、そのほかのコマンドも理解できるようになります。

引数なしで実行するとカレントバッファが、引数を指定すると対象のファイルがステージングされます。ディレクトリ名を渡すと、その配下が一括でステージングされます。

Gread

Gwriteの意味を理解いただけたら、こちらもわかるかと思います。最終コミットから変更を加えたあと:Greadを実行すると、行った変更が破棄されて最終コミットのバージョンに戻ります。gitではチェックアウトの意味になります。

▼図1 「:Gdiff」の実行結果

Gdiff

git diffを分割ウィンドウでわかりやすく表示します(**図1**)。Vimのdiff機能をすでに使っており、横方向に分割表示したい方は、diffoptオプションにverticalを追加します。

```
:set diffopt+=vertical
```

また、変更された位置を行き来するにはdiffの表示内で[c(上方向)と]c(下方向)をタイプします。Gdiffコマンドは引数にブランチ名を指定できるので、トピックブランチと現行のブランチとの差分も簡単に表示できます。

```
:Gdiff feature-branch
```

Gcommit

Vimからコミットが行えます。:Gwriteコマンドを使ってあらかじめステージングしておく必要があります。いったんどのファイルがステージングされているかを確認したい場合は、次に説明する:Gstatusコマンドで状況を把握し、Cをタイプしてコミット画面へ移ることもできます。エディタを開いたままコミットできるので、とてもシームレスに開発を進められます。

Gstatus

現在の状態を表示するコマンドです。ファイル名部分にてDをタイプするとdiff差分が表示さ

注2) URL https://github.com/tpope/vim-fugitive

▼図2　ステージング予定

```
# On branch master
# Your branch is up-to-date with 'origin/ma
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what
#   (use "git checkout -- <file>..." to dis
#
#       modified:   201608/201608.md
#
```

れます。またCをタイプするとコミット画面が表示されます。なお、ステージングするファイルを登録または解除する場合はファイル名部分で-をタイプします。すると、追加と解除がトグルします(**図2、3**)。

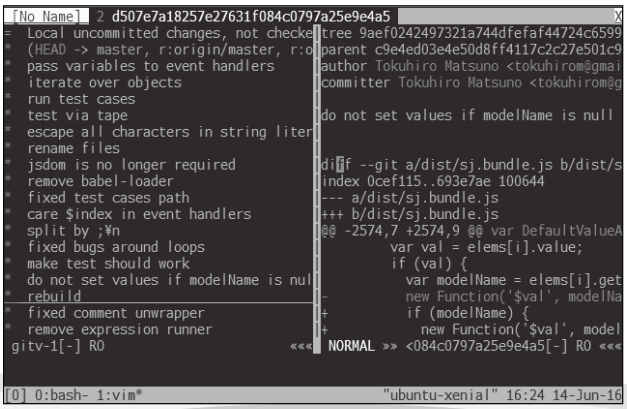
Gblame

fugitiveを有名にさせた一番の機能はこのGblameだと思っています。

コミットの詳細を確認したいファイルを開き、:Gblameを実行します。すると左側に分割ウィンドウが作成され、その行を編集した際のコミット名がコミット作者の名前とともに表示されます。

分割されて画面が狭く感じるかもしれません。その場合Cをタイプすると可能な限り分割ウィンドウが小さくなります。元に戻す場合はDをタイプします。分割ウィンドウ側に移り、コミットID部分でEnterをタイプするとそのコミット時点でのソースコードが右側のウィンドウに表示されます。

▼図4　gitv



▼図3　ステージング解除

```
# On branch master
# Your branch is up-to-date with 'origin/ma
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unst
#
#       modified:   201608/201608.md
#
# Untracked files:
#   (use "git add <file>..." to include in
```

ステータス行連携

現在編集中のブランチがわかりやすいように、fugitiveではステータス行で使用するための関数を用意しています。statuslineオプションに次を追加することで表示できます。

```
%{fugitive#statusline()}
```

gitv

gitにはgitkというGUIツールが付属します。現在のリポジトリのログや変更内容をGUIで確認できます。

このgitkの動作をVimで実現したのがgitv[注3]です(**図4**)。:Gitvで起動します。gitvはfugitiveの拡張プラグインですので、動作にはfugitiveが必要です。

gitvの操作は単純明快です。コミットログビューアですので、基本はコミットログから差分を閲覧するのがメインの機能です。oやO、sでウィンドウの分割方法を変えられます。詳しくは:help gitvを参照してください。

このコミットログ画面ではcoをタイプすると閲覧対象のブランチを閲覧できます。ただし起動時に、:Gitv!のように!(バングと言います)を指定して起動した場合には、閲覧モードではなく変更モード(ファイルモードと呼ばれています)となり、coをタイプした際は、実際にカレントブランチを変更できます。

とても便利なプラグインなのです

注3) URL https://github.com/gregsexton/gitv

▼表1　agitでの操作

キー	説明
C	指定のブランチをチェックアウト(git checkout)
cb	新しいブランチを作成する(git checkout -b)
D	ブランチを削除する(git branch -d)
rs	ソフトリセット(git reset --soft)
rm	リセット(git reset)
rh	ハードリセット(git reset --hard)
rb	リベース(git rebase)
ri	(対話的に)リベース(git rebase -i)

が、Windowsでは日本語交じりのコミットログを正しく表示できない、また遅い、さらにカスタマイズが容易ではないなどの問題点があります。

agit

gitvの問題点を解決し、不足している機能を足したものがagit[注4]です。gitvに比べて高速に動作し、j/kによるコミットログの閲覧もサクサクと動作するようになっています。

agitもgitv同様にコミットログビューアですので、操作は簡単です。起動は:Agitもしくはカレントバッファのファイルに対して履歴を閲覧する:AgitFileのどちらかで起動します。

:Agitで起動した場合はリポジトリ全体の履歴一覧が表示され、カーソルの移動(j/k)によって、右側のウィンドウにそのコミットで行われた変更がdiff形式で表示されます。

現在表示しているファイルに対して:AgitFileで起動すると、そのファイルの履歴が表示されます。:AgitFileで実行したモードではコミットログのカーソル行が示すリビジョンで加えられた変更をdiff形式で表示します。

gitvでは、リポジトリに対して行える変更コマンドがチェックアウトだけでしたが、agitでは**表1**の操作が可能です。

このように、gitを使った一連のブランチ制御が一通りできるので、コマンドラインを使うよりもサクサクとgit操作が行えます。

▼図5　「:Gita status」の実行結果

vim-gita

先に紹介したfugitiveは、Tim Pope氏のプラグインということもあり多くのユーザから評価を得ているプラグインですが、その反面次のような不満を持っているユーザもいます。

・インターフェースが統一されていない
・コマンドが直感的ではない(例：Gread＝git checkout、Gwrite＝git add)
・コマンドが覚えられない

そこで登場したのがvim-gita[注5]です。vim-gitaでは、各コマンドがそれぞれの機能と連携しています。たとえば差分一覧からのblameやコミットなど、ユーザが期待している多くのgit機能が連携されています。いったんgitaのコマンドを起動すれば、git作業のほとんどをgitaの中で完結でき、かつそれぞれにマッピングが用意されています。

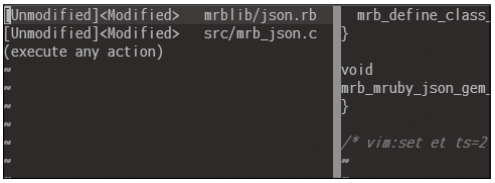
vim-gitaを使った開発ワークフローは次の手順です。

❶新しくファイルを編集する
❷:Gita statusを実行する(**図5**)
❸<<をタイプしてステージングする(>>で解除)
❹<C-^>(Windowsのコマンドプロンプトでは

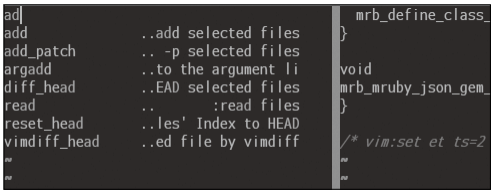
注4)　URL https://github.com/cohama/agit.vim

注5)　URL https://github.com/lambdalisue/vim-gita

▼図6　「:Unite giti/status」の実行結果



▼図7　unite-gitiで「ad」と検索



<C-6>)でコミットメッセージ編集

❺さらにほかのファイルも編集

❻:Gita diffで変更内容確認

❼:Gita statusから:Gita commitでコミット

このように、おおよその操作をgitaの中で完結できるようになっています。

基本的なインターフェースはfugitiveに似ているので、fugitiveを使っていて不満のある方、とくにインターフェースが気に入らないという方はgitaを試してみてはいかがでしょうか。

unite-giti

unite-giti[注6]は、Shougo氏が開発しているUniteというプラグインで使用できるgit向け拡張プラグインです。Uniteを常用している方であれば、操作感がイメージできるはずです。

```
:Unite giti/status
```

このコマンドでステータスを確認し、アクション(TABをタイプ)を選択し追加(git add)や削除(git rm --cached)、リセット(git reset)やコミット(git commit)が行えます(図6、7)。それぞれのコマンドを手打ちすることなくUniteの候補選択で実行できるので、覚えないといけないことが少なくなります。

CtrlPとgitの連携

CtrlPにはUniteほど充実したgit連携プラグインは存在しませんが、gitで管理されたプロジェクト内のファイルを選択する場合、簡単な設定でそれを実現できます(**リスト1**)。

この設定を使うと、gitの管理フォルダ内であればgit管理しているファイルのみを、gitで管理されていないフォルダであれば通常のファイル一覧を表示します。何も考えずに<leader>f[注7]をタイプすれば、良い感じにファイル一覧を開

注6) URL https://github.com/kmnk/vim-unite-giti

注7) mapleaderを設定していないのであれば\f。

▼リスト1　CtrlPのgit連携設定

```
nnoremap <Leader>f :call <SID>CtrlPFilesWithGit()<Return>

function! s:CtrlPFilesWithGit()
  if exists('b:ctrlp_user_command')
    unlet b:ctrlp_user_command
  endif
  call system('git rev-parse --is-inside-git-dir')
  if v:shell_error == 0
    let b:ctrlp_user_command = ['.git', 'cd %s && git ls-files']
    exe 'CtrlP'
  elseif v:shell_error == 128
    exe 'CtrlPCurFile'
  else
    exe 'CtrlP'
  endif
endfunction
```

いてくれるので便利です。

このように、Vimからgitを扱う方法は、紹介しきれなかったものも含めるとかなり多いです。実を言うと筆者はこれらのプラグインを常用せず、tmuxで端末を分割してgitをコマンドラインで操作しています。しかしこれは単に、筆者がそのほうが速く操作できるからであって、gitaやuntie-gitiのほうが速いという方はそちらを使うべきだと思います。

GitHubを探せば、まだまだたくさん見つかると思います。自分に合ったgitの連携方法を見つけ出してください。SD

Vim目録

Vim scriptにおける制約

Vim scriptを一度でも触ったことがある人ならわかると思いますが、Vim scriptはほかの言語と異なり、非常に厄介な問題を持っています。

ひとつは関数リファレンスを格納するための変数名が小文字で始まってはいけないというルールです。

```
" エラーになる
" E704: Funcref variable name must ➤
start with a capital: f
let f = function('system')
```

小文字で統一したい開発者にとっては、なんともモヤモヤするルールです。なぜこのようなルールがあるかというと、Vimのビルトイン関数(たとえばsystem)を勝手に書き換えてしまうことができ、ほかのプラグインで誤動作しかねないためです。こればっかりはしかたないと諦めてください。

もうひとつは、一度値を代入された変数が型拘束を持ってしまうということ。

```
let v = 2
let v = 'hello'            " これはOK
let v = 3.14               " これもOK

" 以下はエラーになる
" E706: Variable type mismatch for: v
let v = {}                 " これはNG
let v = []                 " これはNG
let v = function('system') " これもNG
```

Vimで扱えるスカラ値(数値、文字列)を格納した変数にはあとから辞書、配列、関数リファレンスを再代入できないのです。再代入するためには、一度unletを使って変数を削除しなければなりませんでした。この問題は、たとえば何が格納されているかわからない配列をforでループする際に問題となります。

```
let a = [1, ["foo", 3], {"foo": "bar"}]
for v in a
  echo v
endfor
```

変数vは1回目のループではスカラ値を格納しますが、2回目は配列が格納されます。よってこの問題に遭遇し、実行時にエラーとなります。これを回避するためには次のように書かないといけませんでした。

```
let a = [1, ["foo", 3], {"foo": "bar"}]
for v in a
  echo v
  unlet v
endfor
```

しかし、Patch 7.4.1546にてこの制約が外されました。これによりVimプラグイン開発者を長年悩ませていた問題が1つなくなりました。しかしながら、世界中にはいろいろなバージョンのVimを使っているユーザがいます。しばらくの間は、前述のようにunletで変数を削除するコードを書くことになりそうです。

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28回 シェルコマンドを活用しよう（発展編）

シェルコマンド4部作の最終回では、作業をさらに自動化するパッケージを紹介します。コンパイル時のウィンドウのポップアップを制御したり、ファイルを保存したときに特定のプログラムが動くよう設定したり、またアクセス権を自動で付与したり……。手間を少しずつ減らすことで、日々の作業を大きく効率化できます。

コンパイルをもっと便利にするパッケージ

コンパイルはEmacsでもっと便利に

前々回から、Emacsでコンパイルを行うM-x compileについて掘り下げてきました。開発中のプログラムをコンパイルするとエラーや警告が出るのは日常茶飯事ですので、M-g M-n/M-g M-pによるエラージャンプ機能は便利です。

一方、多くのプロジェクトに関わってくるとコンパイルコマンドが異なってくるので、それを切り替えるのが面倒という問題があります。compile-commandをファイルローカル変数・ディレクトリローカル変数にする方法もありますが、smart-compileパッケージによってファイル・プロジェクトごとにcompile-commandを設定するほうが管理しやすいでしょう。その発展形として、multi-compileは複数のコンパイルコマンドが登録でき、テストやデプロイなども可能になります。

*compilation*ウィンドウを自動で閉じる

Emacsでコンパイルすると、別ウィンドウに*compilation*バッファがポップアップしてきて、ウィンドウ構成が崩れてしまいます。*compilation*バッファの情報を見たいのは、エラーや警告が出ているときくらいのものです。正常にコンパイルが終了した場合は、わざわざウィンドウ構成を戻す必要があって面倒に思います。

そこで、bury-successful-compilationパッケージを使えば、コンパイルが正常終了した場合は自動で*compilation*ウィンドウが閉じられ、元のウィンドウ構成に戻るようになります。パッケージをインストールし、次の設定を加えれば有効になります。

```
(bury-successful-compilation 1)
```

こんな名前ですが、マイナーモードです。

ファイル保存後に処理を実行させる

ファイルの変更をシステムに反映させる

ファイルを保存したあとに、シェルコマンドを実行したいケースがあります。設定ファイルを編集したあとに、そのプログラムに設定を反映させたいといった場合などです。

保存後に自動で反映コマンドを実行させることで、反映忘れで余計な時間を消費することがなくなります。たとえば、個人用cron設定ファイル~/.crontabを編集後、crontab -e ~/.crontabを実行するといった具合です。

また、保存後にコンパイラやテストプログラムを実行させたいといった場合もあります。M-x compileなどを手動で実行する手間は省けます

▼リスト1　auto-save-buffers-enhancedにて、①と②を実現する設定

```
(require 'auto-save-buffers-enhanced)
(setq auto-save-buffers-enhanced-interval 5)
;; not-save-fileと.ignoreは除外する
(setq auto-save-buffers-enhanced-exclude-regexps '("^not-save-file" "\\.ignore$"))
```

▼リスト2　real-auto-saveにて、①を実現する設定

```
(require 'real-auto-save)
(setq real-auto-save-interval 5)
```

が、エラージャンプ機能が使えなくなるというトレードオフがあります。

自動保存との兼ね合い

ただし、自動保存パッケージと併用するときには注意が必要です。

第17回（2015年9月号）で紹介したauto-save-buffers-enhancedパッケージやreal-auto-saveパッケージは、ファイルを変更後一定時間後に自動で保存し、手動でC-x C-sする手間を省けました。これらの自動保存パッケージと、これから紹介する自動シェルコマンド実行パッケージを組み合わせた場合、ファイルの変更が中途半端な状態でシェルコマンドが自動実行される場合があります。その場合はエラーが通知されてしまいます。

自動保存パッケージを使いつつ、中途半端な状態によるエラーを防ぐには2つの選択肢があります。

①自動保存までの間隔を長くする（5秒など）
②自動シェルコマンド実行を設定しているファイルを自動保存対象外にする

auto-save-buffers-enhancedパッケージではどちらも可能です（**リスト1**）。

real-auto-saveパッケージには、執筆時点でファイル単位で自動保存を無効にする機能はありません（**リスト2**）。

auto-shell-command

auto-shell-commandパッケージは、ファイル保存後に自動的にシェルコマンドを実行させることができます。これだけだと、各自がafter-save-hookを設定すれば済む話のように思えますが、このauto-shell-commandはとてもうまくできています。次の特徴があります。

・非同期実行なのでシェルコマンド実行中でもEmacsを操作できる
・ファイル名の正規表現によってシェルコマンドを指定できる
・自動実行を一時的に中止することもできる
・特定のファイルのみ一時的にシェルコマンドを設定できる
・Emacs外のファイル書き換えには影響されない
・自動実行シェルコマンド実行中に実行命令が来たときは前の実行終了後に実行される
・対話的にシェルコマンドを設定でき、それを永続化する設定がキルリングに格納される
・シェルコマンドが異常終了したときのみ実行結果が表示される
・作者が日本人で日本語マニュアルがある

非同期に実行されるので、実行に時間がかかるシェルコマンドも指定できます。たとえば、テストやコンパイラを走らせることだって可能です。

ただし前述のとおり、自動保存パッケージにより中途半端な状態で保存された場合はエラーが通知されてしまいます。その辺はうまく折り合いをつけてください。

◆基本的な使い方

自動実行の設定はM-x ascmd:addで対話的に行うのが基本です。

たとえば、~/.crontab にcronの設定を書いて、

▼表1　M-x ascmd:addの自動実行を制御するコマンド

コマンド	機能
M-x ascmd:toggle	自動実行を一時的に無効化／有効化する
M-x ascmd:popup	自動実行の出力バッファがポップアップされる
M-x ascmd:exec	ファイル名を指定することで、ファイルを変更することなく自動実行シェルコマンドを実行する
M-x ascmd:remove	直前の自動実行の設定を解除する
M-x ascmd:remove-all	すべての自動実行の設定を解除する

保存後にcrontab ~/.crontabを実行する場合はM-x ascmd:add RET ~/.crontab RET crontab $FILE RETを実行します。なお、$FILEはファイル名、$DIRはディレクトリ名に置換されます。するとエコーエリアに、

```
(ascmd:add '("~/.crontab" "crontab $FILE"))
```

と出て、同じ内容がキルリングに格納されます。

init.el上でM-x ascmd:addを実行してC-yで貼り付ければ永続化することもできます。初期設定時には、その操作を繰り返すことになるでしょう。

ascmd:addは、自動実行の設定を格納する変数ascmd:settingにpushされていくので、あとから実行したほうが優先度が高くなります。

◆そのほか制御コマンド

M-x ascmd:addには、自動実行を制御するコマンドがいろいろと用意されています(**表1**)。ただ、使用頻度が低いのならば、わざわざキーに割り当てる必要はないでしょう。

実行属性を自動で付ける

実行可能なスクリプトファイルを作成するとき、わざわざシェルでchmod +xを実行していませんか？

実はEmacsでは#!行が含まれる場合に実行属性を付けてくれる機能があります。**リスト3**の設定で、保存後に自動で実行属性を付けてくれます。一度設定すれば、二度と実行属性について注意する必要がなくなります。

おわりに

これまで4回に渡り、シェルコマンドを扱うコマンド・パッケージを紹介してきました。

M-!など任意のシェルコマンドを実行するコマンド、M-x ifconfigなどの単発シェルコマンド実行のための専用コマンド、さまざまなコンパイルコマンド、保存時に実行するシェルコマンドの設定など、多くの種類がありました。ここでは紹介しきれませんでしたが、独自インターフェースを用意しているパッケージもあります。

M-x compileに似ていますが、バックグラウンドで実行し、異常終了したときのみ結果をポップアップしてくれるbprパッケージもあります。これは経過時間をリアルタイムに教えてくれます。筆者のサイト[注1]で紹介しています。

筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指しています。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お答えします。「こんなパッケージ知らない？」「挙動がおかしいからなんとかしてよ！」はもちろんのこと、自作elispプログラムの添削もします。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

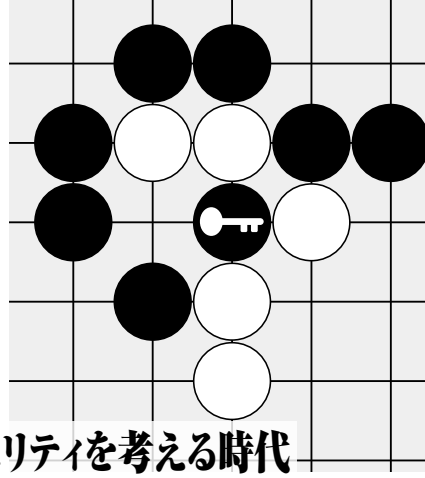
注1）URL http://rubikitch.com/2015/11/03/bpr

▼リスト3　ファイル保存後、自動で実行属性を付与する

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

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第三四回】電力施設から家電クーラーまでセキュリティを考える時代

「これだけ膨大な数のコンピュータがつながっているインターネットなのだから、自分のコンピュータに脆弱性があったとしても、それほど簡単に他者から見つかるものではない。」そう考えているとしたら、その考えは改めなければなりません。いまや、インターネットにつながるコンピュータの情報はすべて把握されています。今回はそれを示す事例をいくつか紹介します。

天網恢恢とはまさにインターネットのこと

「天網恢恢疎にして漏らさず」という中国の古い言葉があります。「悪いことを行えば必ず天罰を受ける」という意味で使われます。

「天網恢恢疎にして漏らさず」という言葉を直訳すると「天の網は広く大きくて目が粗いように思うが、その網から逃れることはできない」となります。

この言葉をこう解釈してはどうでしょうか。「天網」の意味は「天に張り巡らされている網」で、そして「恢恢」とは「広い」「大きい」というわけですから、「天網恢恢」とは、まさに世界中に張り巡らされているインターネットのことです。

「疎にして」とは「網の目が粗い」つまり、インターネットは広大なネットワークですので、どこに何がどうつながっているかはわからない、ということにします。「漏らさず」は、もともとの意味でもある「漏れなくすべては把握されている」と解釈します。つまり……、

「広大なインターネットの空間に接続されていても、アクセス可能となってる機器は漏れなくすべて把握されている」

疎にして漏らさず

2015年12月23日、ウクライナの電力会社がマルウェアを使った攻撃を受けて、イヴァーノ＝フランキーウシク地域の140万人住民の半分が停電被害に遭うという事件がありました[注1]。

この停電は、制御コンソールがマルウェアにより使えなくなったことが引き金になって起こりました。このマルウェアが侵入に使った脆弱性は、2014年1月にCVE-2014-0751として告知されている脆弱性で、10212/tcpポートに細工をしたメッセージを送ると任意のコードを実行できるという危険性の高い脆弱性です。

約2年前から脆弱性情報が出されている既知の脆弱性を修正もせずに動かしているのもどうかと思いますが、それ以上に、その制御コンソールがPCベースであること、さらにインターネットに接続していて外部から直接アクセス可能な状態になっていたということも信じがたい事実です。招いた結果は大規模な地域停電という重大なものですが、その背景は極めてお粗末です。

脆弱性情報が出ていても、このPCを誰かがわざわざ探したりするとは思わなかったのかもしれません。あるいは、広大なインターネット空間の中でこ

注1） 詳しくは本連載第32回（本誌2016年5月号）をご覧ください。

のようなマイナーなシステムを探し当てるのは、藁の山の中から1本の針を見つけるのと同じで、極めて稀なことだと考えたのかもしれません。もちろんウクライナのシステムを計画、設計して導入した人たちに直接聞いたわけではないので、本当のところはわかりませんけれども。

しかし、この広大なインターネットで見つけられる可能性は極めて小さい、あるいはずっと隠れていられると思う人は少なからずいるはずです。

20年前、30年前ならばいざ知らず、SHODAN[注2]やCensys（後述）のようなサービスが存在している2016年では、問題がある機材がインターネットに接続していれば、見つかるのは時間の問題でしかありません。現代のインターネットの世界は、まさに「天網恢恢疎にして漏らさず」なのです。

現状のIoT製品を見てみると

では、今後増えていくIoTデバイスはどうなのでしょうか。今でも、インターネット経由でモニタリング可能な監視カメラなどはビルや街角、あるいは家庭のレベルでも使われています。今後は、本格的に家電をインターネットにつなぎ、外部からコントロールするようになっていくでしょう。このとき、製品側としては、外部からの攻撃を想定したシステムにしたり、あるいは脆弱性があったときに自動的にセキュリティ・アップデートが行われたり、はたまたネットワーク的に安全にするためにわざわざ小規模ネットワーク向けのファイアウォールを用意したりするでしょうか。

現在、PCを買うとデフォルトでマルウェアを検知するようなセキュリティソフトウェアが付いてきて、最低限な機能といえども最初から安全性には配慮しています。しかし、昔は、マルウェア対策セキュリティソフトウェア、当時の呼び方ではウィルスチェッカーと言っていましたが、それを導入するのも別途自分で購入しなければなりませんでした。今はマルウェア対策が当たり前でも、昔はオプショナルで別途購入し、自らインストールするといった作業が必要でした。

今後、雨後のタケノコのようにIoTと呼ばれるような製品が、ぞくぞくと現れるはずですが、昔のPCのようにセキュリティは考慮されずに、「流行りに取り残されないように、とりあえず作って売ってみた」レベルのものが続出するはずです。そして、そのセキュリティの問題が社会一般に影響を及ぼし、大きな社会問題として表面化しない限り、その流れは止めようがありません。それはちょうど、PCに入れると処理が遅くなるウィルスチェッカーのようなソフトウェアにわざわざお金を払う必要性を感じなかったころと同じだと言えるでしょう。

しかし、IoT製品のセキュリティ問題が表面化したときには、すでに多くの製品が広がっており、それをアップデートするにしても時間がかかる、あるいはアップデートすることも難しくなる、そんな問題があるのが現実です。家庭向けインターネット接続のためのルータというネットワーク専用機器ですら脆弱性があった場合、十分にアップデートできない現状ですから、ましてや「とりあえず作ってみた」レベルの初期段階のIoT製品にどれだけ期待できるのかという不安は残ります。

スマホでエアコン操作

2012年の夏、大手家電メーカーが家庭向けのエアコンにスマートフォンからインターネット経由でエアコンをON/OFFできる機能を搭載しようとしていました。一方で、国内法の基準では、「危険が生ずる恐れがない」家電以外で、電源をネットワークを経由してONにすることは認められていませんでした。

たとえば、ビデオの録画予約などは火事になったりするような「危険が生ずる」ことはないので、このような規制の範囲には入りません。諸条件があるのですが、電力消費量もかなりあるエアコンをONにする場合は規制外とはなりませんでした。

注2） インターネットに接続されている機器を検索できるWebサービス。詳しくは本連載第18回（本誌2015年3月号）をご覧ください。

メーカー側が安全装置をビルトインするなどして「危険が生ずる恐れがない」ことを示せれば問題はありません。しかし、その解釈は作った側のメーカーと、経済産業省の家電製品の安全対策などを担当する製品安全課との間でずれがあり、2012年の時点では、メーカーはエアコンを遠隔操作でONにする機能を削除することになりました。当時、話題になったので、記憶にある方も多いかと思います。

この件に関して、2012年時点ですでに似たような機能を持ったスマートホームのHEMS(Home Energy Management System)が存在し、リモートからON/OFFできることも指摘されました。これは、HEMSが製品安全課ではなく情報経済課の担当であるという縦割りで、また電気用品安全法の外にあり、法整備がなされていない分野のため規制されませんでした。これもおかしい話です。

また、SNS上では、「新しい商品を認めないことでイノベーションを阻害している」という批判が出ていました。確かに、安全対策として何をすべきかの環境の整備が遅れているのは批判されるべきですが、一方で、安全の検討を十分にしていないことを見逃せば「イノベーションが進む」というわけではありません。むしろ、一定の安全基準をきちんと設けて、その基準を技術的にクリアすることこそ正しいイノベーションです。何の基準もなく検討もなく安全性を示すことなく、楽観的に、あるいは無邪気にとでも言うのでしょうか、「やってみた」のレベルでインターネットから接続して機器をコントロールできるようにするのは、ルーズになっただけであり、それをイノベーションとは呼ぶことには、筆者は抵抗を覚えます。

2013年に規制の条件が見直され、従来の「危険が生ずる恐れがない」から、「通常の使用状態において危険が生ずる恐れがない」かつ「動作が円滑である」という条件になりました。この件に関しては、規制緩和という報道がなされていますが、解説資料[注3]を読むかぎりにおいては、「規制緩和」ではなく「明確化」と言ったほうが正しい表現です。

また、2016年3月に、電気用品調査委員会から「『解釈別表第四に係わる遠隔操作』に関する報告書の追加検討報告書」[注4]などが、さらに追加され検討が続いています。

筆者の目からみると、ネットワーク経由での「安全である」というロジックを明確化したことによって、あいまいだった規制を強化しています。また、重要な点なのですが、遠隔操作について消費者にメリットだけではなくデメリットも明確に伝え、そして理解してもらうことを前提にしています。この点は非常に評価すべきものだと思っていますし、そのため経済産業省のWebサイト[注5]には次の文言が明示されています。

消費者の皆様におかれましては、警告表示等の内容を十分に確認するなど、配線器具の遠隔操作に伴うリスクを十分に理解された上で、安全にご利用いただきますようお願いいたします。

このエアコンの件から派生した「規制緩和」に関して、これまでいくつかのメディアで「規制緩和によりエアコンの遠隔操作が可能となった」という記事は見かけましたが、「安全を明確化した」と報道したり、「使う前にリスクがあることを十分に理解するべき」という啓発を行ったりしている記事は見かけた記憶がありません。その意味では、2013年の規制の見直しは本来の目的を達成したとは言えないのかもしれません。なぜならば言うまでもなく毎年、IoTのリスクは上がっているからです。

SHODANに続くCensysの登場

インターネットに接続しているサーバやクライアントといったコンピュータだけではなく、ルータなどのネットワーク機材、IoT機器と言われる情報家

注3) 「遠隔操作に対する技術基準の解釈について」電気環境安全研究所、平成25年5月
http://www.jet.or.jp/common/data/new/new143_semi_2.pdf
注4) 「『解釈別表第四に係わる遠隔操作』に関する報告書の追加検討報告書」電気用品調査委員会、平成28年3月22日
http://www.eam-rc.jp/pdf/result/remote_control_4_2.pdf
注5) http://www.meti.go.jp/policy/consumer/seian/denan/topics.html

電や組み込み機材、あるいは制御系システムなど、外部に公開する意図があるか否かにかかわらず、外部からアクセス可能なネットワーク機材の存在をスキャンしデータベース化して、それらの情報などをユーザに提供しているサイトが、現在、いくつか存在しています。最も有名なのが、これまでの連載でも何度か取り上げたSHODANです。

SHODANによりPCやサーバなど従来のインターネット接続機器だけではなく、IoTに分類されるようなインターネットに接続されている機材もくまなくスキャンされ、蓄積され、検索できるようになっています。

そしてもう1つ、Censysという、米国ミシガン大学と米国イリノイ大学アーバナ・シャンペーン校の研究者が作った、インターネット全体をスキャンするシステムと、その情報をデータベース化して提供するサービスがあります(**図1**)。

特徴はデフォルト設定のNmapより1,300倍も高速にインターネット空間をスキャンするZMapを開発して使っていることです[注6]。ZMapはC言語で書かれた約8,900行のサイズのプログラムでGNU/Linux上で動作します。その性能は、1Gbpsのネットワークに接続すればIPv4空間を45分ですべてスキャンしてしまいます。

◆**図1　Censysのサイト(https://censys.io)**

シンプルに検索できるだけの、まさに検索サイトとなっている。

現在、一般の人だけではなく、ある程度のネットワークの知識のある人でも、何の保護もなくネットワークに接続している機材がいつの間にかスキャンされるということを意識している人はそんなにいないと思います。

しかし、実際には、IPv4空間すべてを45分でスキャンする能力のあるスキャナーが存在しており、そのスキャン結果をデータベース化し、条件に合わせて柔軟に検索できるようになっています。今はもうそんな時代です。IoTセキュリティを考えると、脆弱性はもとより、設定ミスがあったままインターネットに接続していれば、確実にSHODANやCensysのようなサイトに捕捉され、データベース化され記録に残される、ということです。

SHODANやCensysの脅威の見える化

Censysのデータベース検索は、ゲスト(登録なし)では少ない回数しか使えませんが、登録を行いメールアドレスが確認できればログインして使えるようになります。

使い方は極めてわかりやすく、たとえばシンプルに

```
8.8.8.0/24
```

と検索すると、8.8.8.0/24に接続されているサーバと、標準的なポートに関してのスキャン結果を返してきます(**図2**)。ちなみに、**図2**の検索結果に表示されている8.8.8.8は有名なGoogleのネームサーバサービスです。

雑誌に結果の詳細を掲載することはさすがにためらわれるので、概要だけに留めておきますが、「ビルディングオートメーションと制御ネットワークのためのデータ通信プロトコル」であるBACnetプロトコルに反応を返しているサーバが日本国内にどれくらいあるか検索してみたところ、30件ほど検索されて出てきました。

BACnetプロトコルを使うサーバが見えるような形で運用することは、あまり良いことだと思いませ

注6) https://zmap.io/

ん。VPNで接続を確保し、内部ネットワークとしてアクセスする形を採るべきでしょう。

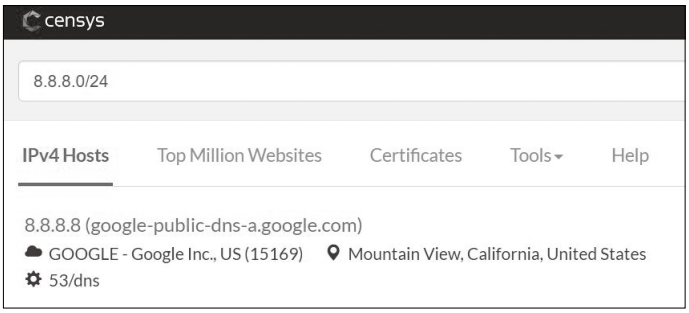
ウクライナの電力会社の話をしましたが、あれはウクライナの話で日本では関係ない、と強く言えるような状況ではないことがわかっていただけたでしょうか。

これもまた何度目かの繰り返しになりますが、SHODANやCensysはネットワークの脅威を見える化してくれており、その警告を我々が知ることができるたいへん有用なサービスであり、研究です。

SHODANやCensysと同様な、あるいはさらに強力な（あからさまに悪意のある攻撃目的のための、と言ったほうがいいのかもしれませんが）スキャンツールおよびデータベースが存在していると我々は考えるべきで、それを前提としたうえで、議論を始めなければいけない時代になっています。

問題のあるIoT機材を不用意にインターネットに接続してしまったら、確実に見つけられてしまうことを理解したうえで使うべきなのです[注7]。SD

◆図2　Censysでの検索結果



注7）現状のIoTノード端末機材類は安全性のモデルや実装が不十分です。そのため、グローバルに開かれているインターネットに直接アクセスすることでリスクを大きくするより、VPNなどを用いてプライベートな閉じたネットワーク空間で利用し、リスクを最小限にすべきだと、筆者は考えています。

SOURCES

レッドハット系ソフトウェア最新解説

新連載

第1回

Ansible Tower

Ansible Towerはオープンソースの Ansibleを基盤とした企業向けソリューションです。今回はAnsible Towerの基本を紹介します。

Author 小島 啓史(こじまひろふみ)
mail : hkojima@redhat.com
レッドハット㈱ テクニカルセールス本部 ソリューションアーキテクト

Ansible Towerの概要

Ansible TowerはオープンソースのAnsibleによる、構成管理／アプリケーションのデプロイメント／オーケストレーションといった各種自動処理を管理するための企業向けソリューションです。CUI/GUI/REST APIといったインターフェースのほかに、ジョブスケジューラ、ロールベースの権限管理、Ansibleによる自動処理の操作記録といった機能を備えています。執筆時点での最新バージョンは2.4.5であり、2016年4月にリリースされました。今回はこのバージョンをベースに技術的な側面からAnsible Towerをご紹介します。なお、Ansibleそのものについては過去の特集[注1]を含めさまざまな媒体ですでに紹介されていることもあり、本記事ではとくに紹介しません。

Ansible Towerはもともと Ansible社(旧名称はAnsibleWorks社)が開発・提供していたプロプライエタリ・ソフトウェアです。オープンソースのAnsibleプロジェクトを立ち上げたMichael DeHaanが中心となって、2013年7月にAnsible AWXという名前の製品としてリリースしたのが始まりです。このときのバージョンは1.2.2であり、2013年11月リリースの1.4.0まではAnsible AWXという名前でしたが、2014年2月リリースの1.4.5からAnsible Towerという名前に変更されました。しかし、2015年10月にAnsible社がRed Hatに買収されてからは、Red HatがAnsible Towerを提供するようになりました。2016年6月現在においてもAnsible Towerはプロプライエタリ・ソフトウェアですが、今後オープンソース化されていく予定です。

ちなみに、執筆時点ではRed Hat製品ドキュメント一覧[注2]にAnsible Tower by Red Hatという名前が記載されていますが[注3]、本記事では単にAnsible Towerと記述します。

Ansible/Ansible Towerは Chefや Puppetと比較すると、まだ歴史が浅いソフトウェアです。しかし、Ansibleの利便性や学習コストの低さのために注目度は高く、日本国内ではDMM.comラボ社[注4]、海外ではNASA[注5]やBinckBank(オンラインバンク)での採用実績が報告されています[注6]。また、Red HatでもOpenShift(PaaS基

注1) Software Design 2016年1月号の第2特集など。

注2) URL https://access.redhat.com/documentation
注3) Red Hat社内製品ページではRed Hat Ansible Towerとも記載されています。
注4) URL http://time.levtech.jp/article/a-dmmcom_labo_20150831
注5) URL https://www.ansible.com/blog/nasa-automation
注6) URL https://www.ansible.com/case-studies

盤を構築するための製品)のインストーラとして、Ansibleを活用しています。

Ansible Towerのインストール

Ansible Towerを入手するには、ほかのRed Hat製品と同様に、年次のサブスクリプション契約が必要です。しかし、執筆時点ではプロプライエタリ・ソフトウェアであるため、もし試用する場合には評価用ライセンスが別途必要になります。

まず評価版のAnsible Towerを入手します。Ansible TowerはLinux/Vagrant/Amazon EC2に対応したインストールプログラムが用意されていますが、今回はRHEL7へのインストールを想定し、Linux版のAnsible Tower[注7]を利用します。

Ansible Towerをインストールするには、まず最新版のRHEL7 for x86_64(執筆時点ではRHEL7.2 for x86_64)をインストールしておきます。インストール終了後、Ansible Towerのインストール準備をします。最初にターミナルを起動してsubscription-managerコマンドを利用してシステム登録を実行します。このとき、保有しているRHELサブスクリプションをシステムに自動付与しておきます。

```
# subscription-manager register --username=redhat_account --password=account_passwd --auto-attach
```

次に、Ansible Towerが必要とするAnsibleパッケージをインストールします。執筆時点ではAnsibleパッケージはRed Hat公式リポジトリではなくEPELリポジトリで提供していますので、EPELリポジトリを利用してAnsibleパッケージをインストールします。

```
# yum -y install http://dl.fedoraproject.org/pub/epel/epel-release-latest-7.oarch.rpm
# yum -y install ansible
```

注7) URL https://www.ansible.com/tower-trial

Ansibleパッケージをインストールしたあとに、入手したインストーラ(tarball)を解凍します。

```
# tar xf ansible-tower-setup-latest.tar.gz
# cd ansible-tower-setup-2.4.5
# ls
README.md  backup.yml  group_vars
install.yml  restore.yml  setup.sh
ansible.cfg  configure  host_vars
  licenses  roles
```

解凍したファイルの中身を見るとhost_vars、group_vars、ansible.cfgといったファイル名が確認できますので、Ansible Towerのインストールにも Ansibleを利用していることが分かります。そこでSSH鍵を作成して、ローカルホストに作成したSSHの公開鍵を登録しておきます。

```
# ssh-keygen -f /root/.ssh/id_rsa -N ''
# ssh-copy-id root@localhost
```

Ansible Towerインストールを実行するための設定を行います。インストール対象のホスト(ローカルホストを指定)、利用するデータベース(内部データベースを指定)、Ansible Towerのadminユーザと Munin(Ansible Towerと一緒にインストールされるサーバ監視ツール)のパスワードの入力が求められるので、入力していきます。その後、Ansible Towerのインストールスクリプトを実行します(**図1**)。

これでAnsible Towerのインストールが完了です。FirefoxなどのWebブラウザでAnsible TowerのWebページにアクセスできます。http://localhostにアクセスすると、ログイン画面が表示されるので、

- Username；admin
- Password；設定時(./configure実行時)に指定したパスワード

でログインできます。

最初にAnsible Towerにログインすると、ライセンス情報の入力画面が表示されます。**図2**の"Get a Free Tower Trial License"をクリッ

▼図1　Ansible Towerのインストールスクリプトの実行

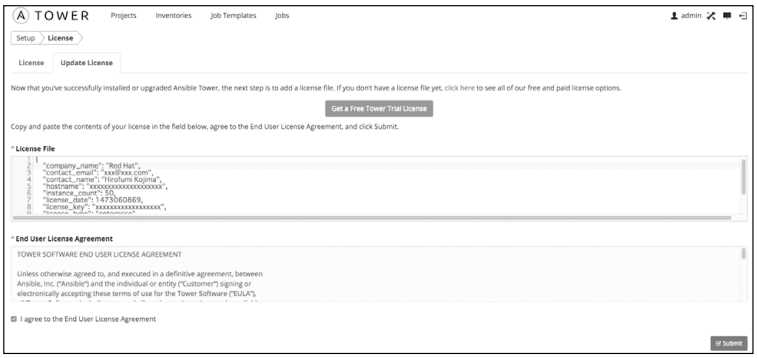
```
# ./configure
-----------------------------------------
Welcome to the Ansible Tower Install Wizard
-----------------------------------------
... snip ...
Enter the hostname or IP to configure Ansible Tower
(default: localhost): localhost
... snip ...
Will this installation use an (i)nternal or (e)xternal database? i
... snip ...
Enter the desired Ansible Tower admin user password:
Enter the desired Munin password:
... snip ...
FINISHED!
# ./setup.sh
... snip ...
The setup process completed successfully.
```

クすると、評価版ライセンスを申請できますので、“ENTERPRISE SUPPORT”付きのライセンスを申請してください[注8]。申請するとライセンス情報が記載されたメールが届きますので、ライセンス情報を入力してから[Submit]をクリックします。

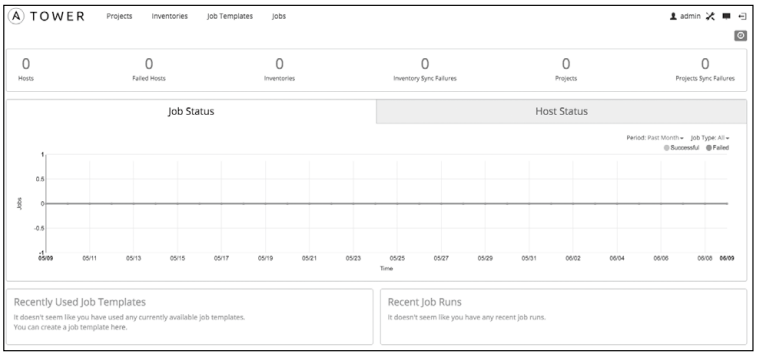
入力したライセンスの認証が正常に完了すると、次の**図3**が表示されます。以降、この画面がログイン時に表示されるWebページとなります。

まだAnsibleによる処理を実行していないため、ログなどは表示されていません。

▼図2　Ansible Towerのライセンス情報入力画面



▼図3　Ansible Towerのダッシュボード



次回は

今回はAnsible Towerのインストールまで紹介しましたが、Playbook/Inventoryの登録などを実施していないため、このままでは何もできません。次回はAnsible Towerのコンポーネントや Playbookの登録・実行などを解説していきます。SD

注8） 管理画面だけ少し触ってみるなどの場合は、“SELF SUPPORT”でもOKです。

Ubuntu 16.04 LTSで Cinnamon 3.0を使用する

Ubuntu Japanese Team／あわしろいくや

今回は、あえてLinux MintではなくUbuntuでCinnamonデスクトップ環境を使ってみます。

UbuntuとCinnamon

本誌2016年5月号の第2特集第2章でも解説しましたが、Ubuntuにはフレーバーという公式派生版が多数あります。主な違いはデスクトップ環境です。しかし、すべての著名なデスクトップ環境がフレーバーとしてリリースされているわけではありません。その中でも代表的なのはCinnamonデスクトップ環境（以下Cinnamon）でしょう。

Cinnamonは、Ubuntuの派生版であるLinux Mint向けに開発されていますが、もちろん単独でも使用でき、たくさんのLinuxディストリビューションでデスクトップ環境として採用されています。しかし、なぜかUbuntuのフレーバーとしてはリリースされていません。やはりLinux Mintがあるからなのかもしれませんが、Linux MintはUbuntuとリポジトリを共有しているものの、特徴がかなり異なるので、CinnamonをUbuntu上で使いたい場面も多いと思います。

UbuntuのリポジトリにCinnamonは存在しないのかと言われるとそのようなことはなく、Debian由来のパッケージが一式そろっています。16.04ではCinnamon 2.8相当のパッケージがありますが、現在のCinnamonの最新版は3.0であり、やはり新しいバージョンを使用したいものです。

最新版のCinnamonを配布しているリポジトリ（PPA）はいくつかあるようですが、筆者も独自のものを用意しています。純粋にDebianのリポジトリに入ったパッケージをリビルドしたもので、アップグレードがスムーズに行えるということにメリットを感じています。パッケージ数がさほど多くないので、管理が比較的容易なのもあります。

というわけで今回は、Ubuntu 16.04 LTSにCinnamon 3.0をインストールし、活用する方法を紹介します。

インストールとログイン

インストールは、Ubuntu Minimalを使用して一から行ってもいいのですが、今回は16.04にCinnamonを追加インストールすることにします。このほうが簡単なのと、同時にインストールしても干渉しないのと、通常最初からCinnamonを使用するということはなく、Unityを使用してみたものの、ほかのものも触ってみたいというリクエストのほうが多いだろうと考えてのことです。

16.04のインストール方法の紹介は省略します。不安があるなら最初は仮想マシンのゲストOSとしてインストールしてみるといいでしょう。

16.04のインストール完了後アップデートをすべて適用したうえで次のコマンドを実行します。

```
$ sudo apt install --no-install-recommends ↵
cinnamon-desktop-environment gksu
```

インストール完了後ログアウトし、名前の横にあ

るUbuntuロゴをクリックすると、Cinnamonが増えています(**図1**)。これを選択してログインするとCinnamonが起動します(**図2**)。

メニューが日本語になっていないのは意図しての動作です。Cinnamonは翻訳状況にかなりの問題があり、英語のまま使ったほうがいいからです。今回も英語のメニューを想定して解説しますが、日本語にしたい場合は次のコマンドを実行後ログアウトして再ログインしてください。

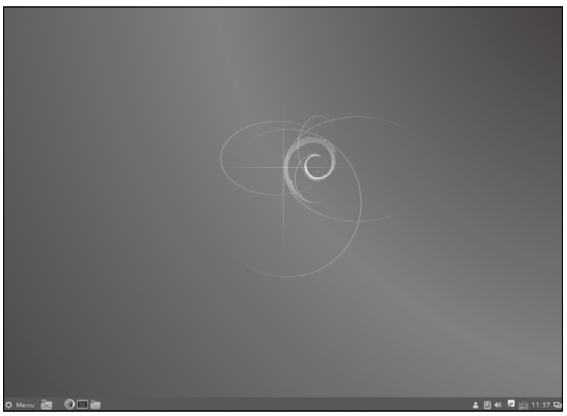
```
$ sudo apt install cinnamon-l10n
```

初歩的なカスタマイズ

続けて初歩的なカスタマイズを行います。可能な限りUbuntu(Unity)に似せるをポリシーにします。

図1　セッションの選択

図2　Cinnamonの初期表示



壁紙

デスクトップを右クリックして、[Change Desktop Background]をクリックします。[Wallpapers]に[Ubuntu]があるので、これをクリックします。

テーマ

Cinnamon用のUbuntuっぽいテーマ(Ambiance)はないことはないのですが、外部リポジトリからインストールしたほうがより「それっぽい」ので、そのようにします。次のコマンドを実行してください。

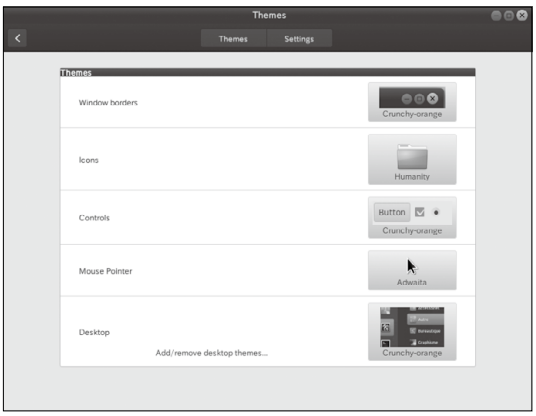
```
$ sudo add-apt-repository ppa:noobslab/themes
$ sudo apt update
$ sudo apt install ambiance-crunchy
```

その後システム設定(System Settings)を起動し、[Themes]をクリックします。[Window borders]を[Crunchy-orange]に、[Icons]を[Humanity]に、[Controls]を[Crunchy-orange]に、[Desktop]も[Crunchy-orange]にします(**図3**)。

インジケーター

Unityが対応しているインジケーター(App Indicator)を採用するデスクトップ環境が少しずつ出てきています。代表的なのがKDE SCですが、このCinnamonも対応しています。[System Settings]-[General]-[Enable support for indicators(Requires Cinnamon Restart)]を[オン]にし、指示どおり一度ログアウトして再ログインするとインジケーターに

図3　テーマの変更する個所



対応した表示になります。ログアウト前と再ログイン後にFcitxのキーボードアイコンをクリックしてみると、違いがわかりやすいでしょう。

操作

Cinnamonを使用する際に知っておくと便利な操作を解説します[注1]。

Ctrl + Alt +矢印キー

Cinnamonの操作の特徴の1つは、このCtrl + Alt +矢印キーではないでしょうか。Ctrl + Alt +左右矢印キーで仮想デスクトップの切り替えというのはよくありますが、Ctrl + Alt +上矢印キーでワークスペースの増減と名称の変更ができます(**図4**)。右端の[+]でワークスペースを増やすことができ、各ワークスペースにマウスのポインターを合わせたときに右上に表示される[−]でワークスペースを減らすことができます。ワークスペースの名称にマウスのポインターを合わせると、ワークスペース名を変更できます。

Ctrl + Alt +下矢印キーで、現在表示しているウィンドウ一覧を表示できます(**図5**)。矢印キーでアクティブなウィンドウを変更することも、右上のバツボタンでウィンドウを終了することもできます。

ワークスペースの切り替えはループせず、ひとつひとつ戻る必要がありますが、ループさせたい場合は[System Settings]-[Workspaces]-[Settings]タブ-[Allow cycling through workspaces]を[オン]にします。

Alt + Tab キー

Alt + Tab キーでウィンドウの切り替えができるのはわりと普通に見られますが、ウィンドウの切り替えの表示方法をカスタマイズできます。[System Settings]-[Windows]-[Alt-Tab]タブの[Alt-Tab switcher style]を表示してください。[Timeline (3D)]はWindows 7のAeroフリップ3D風の表示になります(**図6**)。[Coverflow (3D)]はCompiz風の表示になります。さらにシンプルなものもありますので、お好みに合わせてカスタマイズしてください。

タイル機能とスナップ機能

ウィンドウを特定の場所に移動させるとそのウィンドウを全画面化させたり、あるいは画面半分の大きさに変更させたりする機能がタイル機能です。ほとんどの場合はガイドが表示されるので、その場所でボタンを離すとその大きさになります(**図7**)。Cinnamonでは、さらにこの状態で数秒待っていると[Hold <Ctrl> to enter snap mode. Use the arrow keys to shift workspaces]という説明が表示され、その指示に従ってCtrlキーを押すと領域の色が濃くなり(透過度が低くなり)、スナップ機能に切り替わります。そしてその場所でボタンを離すとウィン

図4　ワークスペースの切り替えと増減

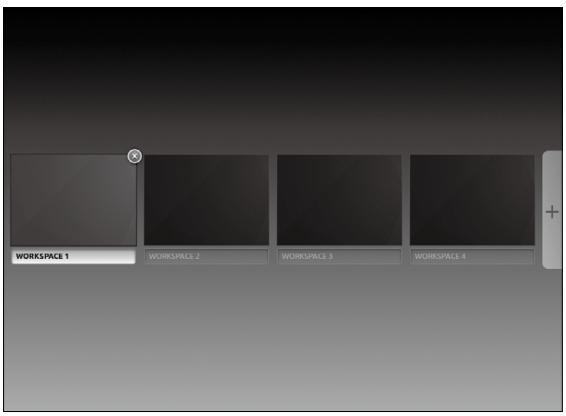


図5　そのワークスペースにあるウィンドウの一覧表示

注1) もうひとつおもしろい機能として[ホットコーナー]があるのですが、今回は紙幅の関係上省略します。

ドウの大きさが変更されます(**図8**)。

結果的にはウィンドウの大きさがガイドされている状態に変更されるという点ではタイル機能もスナップ機能も同様ですが、この2つには明確な違いがあります。タイル機能の場合、特定の条件でほかのウィンドウが邪魔をしなくなります。具体的には、スナップ機能で画面の上半分にウィンドウを表示させた場合、ほかのウィンドウを最大化すると画面の下半分に表示されるだけになります。タイル機能の場合は、最大化するとそのままウィンドウを画面いっぱいに最大化します。すなわち、スナップ機能を使用した場合はその領域を避けてくれるということです。

もうひとつの[Use the arrow keys to shift work spaces]は、そのとおりこの状態で矢印の左右キーを使用すると、ワークスペースを移動できます。タイル機能やスナップ機能を使用しない場合、Ctrl + Alt + Shift +左右矢印キーで現在アクティブなウィンドウをほかのワークスペースに移動することもできます。

図6　見覚えのあるウィンドウの切り替え

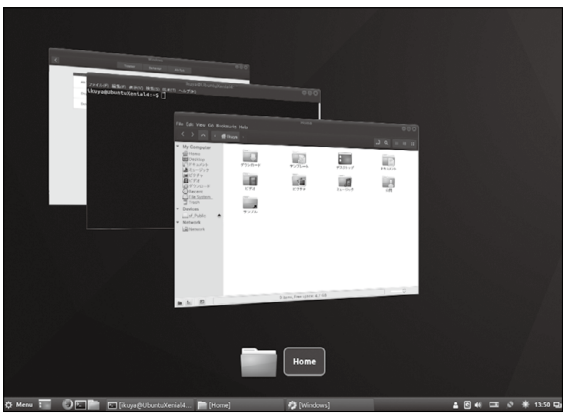
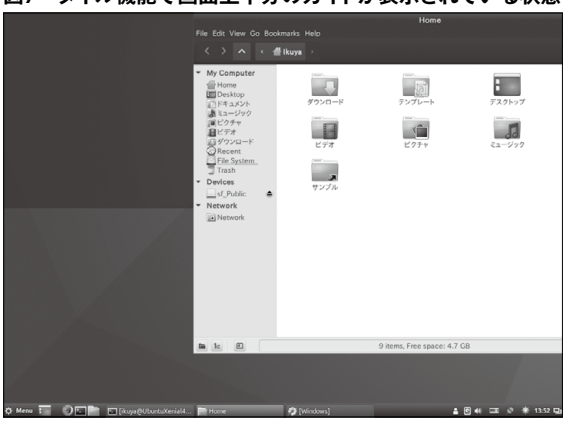


図7　タイル機能で画面上半分のガイドが表示されている状態



アプレットとデスクレット

Cinnamonの特徴として、アプレットとデスクレットがあります。アプレットはパネルに常駐するもの、デスクレットはデスクトップに常駐するものと区別するとわかりやすいでしょう。いずれもインターネット経由で追加できるのがおもしろいです。

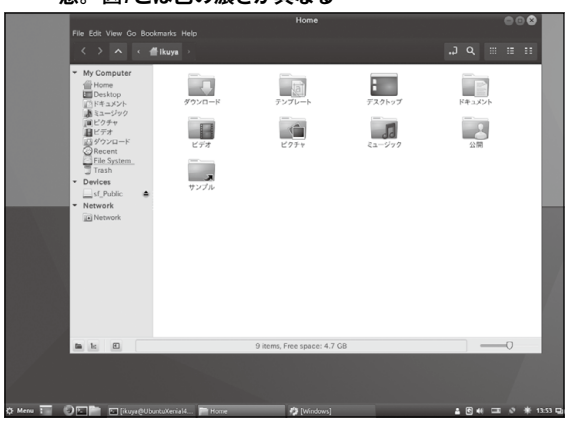
また、今回は使用しませんが拡張機能(Extensions)もあります[注2]。

アプレット

アプレットは[System Settings]-[Applets]-[Available applets (online)]で追加できます。今回は[SmallCalc]を追加してみましょう。その名のとおり小さな電卓のアプレットです。[SmallCalc]にチェックを入れ、[Install or Update selected Items]をクリックし、インストールします。その後[Installed applets]タブに移動して[SmallCalc]を選択して[Add to panel]をクリックするとパネルに追加されます。削除する場合は[Installed applets]タブを表示し、[SmallCalc]を右クリックして[Uninstall]をクリックします。

注2) 執筆段階で最新の拡張機能でもCinnamon 3.0と互換性がなかったので見送ります。

図8　スナップ機能で画面上半分のガイドが表示されている状態。図7とは色の濃さが異なる



デスクレット

デスクレットは[System Settings]-[Desklets]-[Available desklets (online)]で追加できます。今回は[Weather Desklet]を追加してみましょう。その名のとおり天気を表示します。[Weather Desklet]にチェックを入れ、[Install or Update selected Items]をクリックし、インストールします。その後[Installed desklets]タブに移動して[Weather Desklet]を選択して[Add to desktop]をクリックするとデスクトップに追加されます。デスクレットを右クリックして[Configure]をクリックし、[Location]に数値を入力します。この数値はBBC WeatherのWebサイト注3を開いて[Find a Forecast]の欄に表示したい地名を入れ、候補がある場合はそこから選択すると個別ページを表示します。そのURLにある数値が[Location]に入力する数値です。大阪の場合は[1853909]です。正確かどうかはよくわかりませんが、[Data service]から複数のサービスが選択できるので、いろいろと試してみてください。削除する場合は[Installed desklets]タブを表示し、[Weather Desklet]を右クリックして[Uninstall]をクリックします。

その他

パネルに日付を表示する

パネルに日付を表示したい場合、時計のアプレットを右クリックして[Configure]をクリックします。[Use a custom date format]にチェックを入れると好きなフォーマットに変更できます。GNOME Shellに準じて"**%B %e日 (%a) %H:%M**"としておくのがいいでしょう。

メニューの表示項目を変更する

メニューを表示すると[その他]が一番上に来て目障りなので、これを非表示にする方法です。メニューアプレットを右クリックして[Configure]をクリックします。一番下に[Open the menu editor]があるので、これをクリックしてメニューエディターを起動します。項目を非表示にする場合はチェックを外します。すなわち、今回は[その他]の左横にあるチェックを外せば非表示になります。

nemo-fileroller

NemoはCinnamonのファイルマネージャーですが、これをアーカイブマネージャーであるfile-rollerと統合するパッケージが提供されています。次のコマンドを実行してインストールしてください。

```
$ sudo apt install nemo-fileroller
```

一度ログアウトして再ログインすると使用できるようになっています。たとえばファイルやフォルダを選択した状態で右クリックすると、[圧縮]という項目が増えています。

ownCloudクライアントとの統合機能

16.04のリポジトリにあるownCloudクライアントにはNemoでも状態アイコンを表示する機能がありますが、残念ながらパッケージにはなっていません。半ば強引ではありますが、この機能を有効にしてみましょう。**図9**のコマンドを実行してください。

この状態で一度ログアウトして再ログインし、さらにownCloudクライアントを起動して設定完了後同期を開始すると、アイコンの右下にステータスを表示するアイコンが利用できるようになっています。

SD

注3) http://www.bbc.com/weather/

図9 Nemoで状態アイコンを表示する

```
$ sudo apt install nemo-python nautilus-owncloud
$ sudo cp /usr/share/nautilus-python/extensions/syncstate.py /usr/share/nemo-python/extensions/
$ sudo sed -i.org -e 's/autilus/emo/g' /usr/share/nemo-python/extensions/syncstate.py
```

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第33回 ❖Microsoft loves FreeBSD

Microsoft♥FreeBSD

2016年6月8日からカナダの首都オタワで開催された「BSDCan」に併設する形で、FreeBSD開発者会議が開催されました。今回の開発者会議では、MicrosoftとIntelから自社のサービスや技術においてどのようにFreeBSDをサポートしているかの発表が行われたのですが、とくにMicrosoftから行われたMicrosoft Azure（以下Azure）におけるFreeBSDのサポートに関する話に、参加者から高い関心が集まりました。

Microsoftは数年前から、同社の仮想化環境であるHyper-VにおけるFreeBSDのサポートを進めてきました。ここ数年では、AzureにおけるFreeBSDの正式サポートに向けて取り組みを進めてきたのですが、このタイミングでMicrosoftから正式に「FreeBSD 10.3仮想環境」の提供が始まるとともに、その正式なアナウンスが行われました。今回はこのあたりの事情を説明するとともに、AzureでFreeBSDをデプロイする方法を紹介します。

▼写真1　カナダで開催されたFreeBSD DevSummitでAzureについて発表するMicrosoft

◉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。

なぜ、MicrosoftがFreeBSDを……？

MicrosoftとFreeBSDコミュニティはこれまで、それほど仲の良い関係を築いていませんでした。Hotmailのサービスが始まった当初、サーバにはFreeBSDが使われていたものの、のちにWindows Serverに代わっています。当時のMicrosoftは、オープンソースコミュニティに対して比較的敵対的な態度だったと思います。

状況が変わり始めるのは、Microsoftが仮想化技術やクラウドプラットフォームサービスに注力し始めるあたりからです。現在Microsoftはオペレーティングシステムやオフィススイートを販売するというビジネスモデルから、クラウドサービスの提供といったビジネスモデルへの転換を進めています。これはアプライアンスベンダからの要望に応えるといった意味も持っています。

FreeBSDはさまざまな産業で活用されているのですが、そうした産業の1つに組込み産業があります。中でも高性能／高機能アプライアンス（ストレージ、ネットワーク機器など）を開発しているベンダが、そのプラットフォームを物理的なハードウェアからクラウドプラットフォームに移し始めて

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。

います。これまで物理的なハードウェアとソフトウェアをセットで販売していたものが、すべてクラウド上にセットアップしてサービスのみを提供するといった形に変わりつつあります。過渡期にはHyper-Vなどの仮想環境を利用するといったものでしたが、現在ではAzureへ移行する様子をみせています。

こうしたアプライアンスベンダの要求に応えるには、MicrosoftとしてはHyper-VでFreeBSDをサポートする必要があります。完全仮想化であればFreeBSDは動作しますが、それではHyper-Vの性能をフルに発揮できません。Hyper-Vでフル性能を発揮できるようにドライバを開発する必要があるほか、そのほかHyper-Vと連動して便利なサービスを提供するために各種サービス（デーモン）の開発と運用が必要になります。このあたりから、MicrosoftはFreeBSD FoundationやFreeBSDプロジェクト、FreeBSDを利用しているベンダと協力関係を構築するようになります。Microsoftにとって、FreeBSDのサポートは必要不可欠だったわけです。

Microsoftから直接提供されるFreeBSD仮想環境

Hyper-Vで性能を発揮するにはHyper-Vで必要になるバスに対応したドライバを実装する必要があるほか、準仮想化に必要になるドライバをいくつか持っておく必要があります。これに加えてユーザランドで動作するデーモンを動作させ、Hyper-Vのホストと通信して動作させる必要があります。

このあたりの成果物はすでにFreeBSDカーネルにマージされ、パッケージとしてサービスが提供されています。Microsoftは上海にオープンソース系のチームを抱えていますが、彼らはすでにFreeBSDコミッタにも就任しており、Microsoftから直接成果物がマージできる体制が構築されています。

ほかのサービスとAzureのもっとも大きな違いは、Microsoftがゲストオペレーティングシステムとして正式にFreeBSDをサポートしている点と、Microsoftから直接FreeBSD仮想環境のイメージが提供されている点にあります。Amazon EC2など、FreeBSDの仮想環境が提供されているサービスはほかにもありますが、それらはFreeBSDプロジェクトがイメージを提供していたり、個人が自分の責任でデプロイする必要があります。Azureではそのあたりのお膳立てをMicrosoftが行ってくれます。後発のサービスだけあってダッシュボードも扱いやすいものです。

FreeBSD開発者会議では、FreeBSD 10.3をAzureにデプロイする方法が紹介されましたが、今後のリリースでさらに性能が向上することがわかっています（ドライバの改善による性能向上、現在はまだサポートしていない機能の実装など）。すでに動作に必要になる基本的な部分はマージされていますので、FreeBSD Updateでバイナリアップデートが可能ですし、いったんデプロイしたFreeBSDはアップグレードして使い続けることができます。FreeBSDをサーバとして運用する場合、今後はAzureが重要な候補になることは間違いのない状況です。

FreeBSD on Azure

AzureでFreeBSDをセットアップする方法はとても簡単です。すでにアカウントを持っているなら、4つのステップを経て数分でデプロイできます。慣れたら1分といったところでしょう。

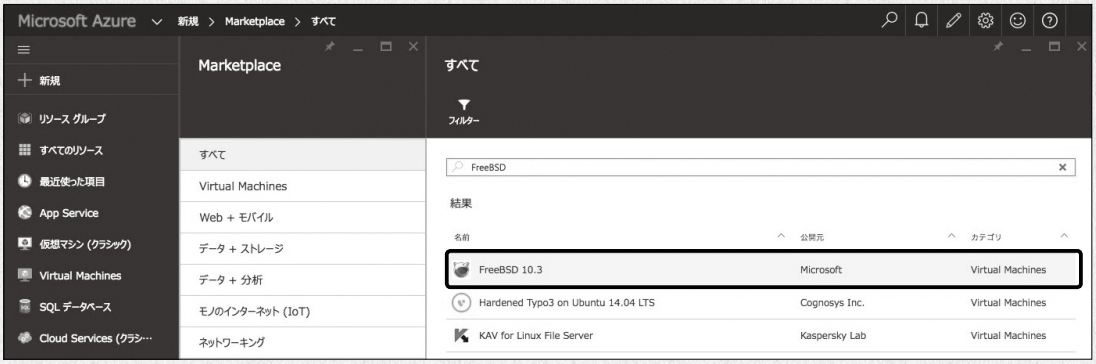
とりあえず試してみたいという場合には、無償サブスクリプションで利用できます。同様のサービスでの無償サブスクリプションには、利用期間に制限があったり、クーポンを使い切ると以降は料金が発生したりしますが、執筆現在（2016年6月）、Azureの無料サブスクリプションはそのまま支払なくサービスを利用し続けることができます。明示的に支払うサブスクリプションに変更しない限り、そのまま利用可能です（その代わり選択できる環境や性能は限られています）。

以降でAzureにFreeBSDをセットアップする方法を紹介します。

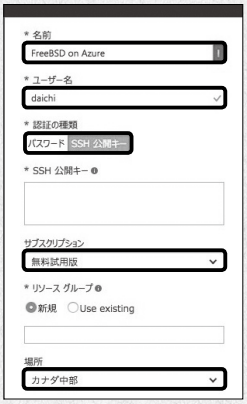
まず、Azureのアカウントのセットアップは各自終わらせておいてください。

チャーリー・ルートからの手紙

▼図1　FreeBSDの仮想環境を検索する



▼図2　基本情報を設定



▼図4　設定内容の確認

▼図3　利用する仮想環境リソースを選択

次に、ダッシュボードから検索をかけてFreeBSD仮想環境を探します(**図1**)[注1]。現在ならFreeBSD 10.3の仮想環境が見つかるでしょうし、今後はFreeBSD 11.0も現れることになるでしょう。

注1　ダッシュボードはタブレットデバイス(Surfaceを意識したものだと思います)を意識したUIにもなっていて、PCから操作するだけではなく、Surface ProやiPad Proからも便利に扱えそうです。

FreeBSD仮想環境を選択してセットアップを開始します。ステップ1では仮想環境の名前、最初に作成するユーザ名、そのユーザのパスワードまたはSSHの公開鍵、サブスクリプションの選択(無料サブスクリプションの場合にはここが「無料試用版」になっている)、クラウドの場所の選択、などを行います(**図2**)。パスワードを設定すればログインは簡単ですが、セキュリティ上好ましいとはいえません。SSH公開鍵のほうを選択して、利用しているSSH公開鍵を入力して、この鍵を使ってログインするようにしてください。

次に仮想環境のハードウェアを選択します。ここでさまざまなリソースの仮想環境を選択できるのですが、無料のサブスクリプションには上限がありますので、「お勧め」として表示されている中からどれかを選択してください(**図3**)。実質的に、設定することはこれで終わりです。

次にストレージやネットワークについて設定するページが表示されますが、ここはとくに変更することなく次の画面にいきます。

最後に設定内容を確認します(**図4**)。これでセットアップ完了です。

セットアップした環境にssh経由でログインすると**図5**のようになります。この環境はFreeBSD 10.3をデプロイしたあとに、FreeBSD Updateを実行してカーネルとユーザランドを最新のパッチセットにアップグレードしてあるほか、いくつかパッケージをインストールして利用しています。はじめからAzureのホストと連携するためのサービスが動

作しているほか、FreeBSD Updateでカーネルとユーザランドをアップグレードしても使い続けられる点がポイントです。

dmesg(8)でシステムメッセージを確認すると、「Hypervisor: Origin = "Microsoft Hv"」の出力を確認できます(**図6**)。

開発者はいつでもログインできるマイ・サーバをいくつか持っているものですが、Azureはその最新の候補になったと言えます。簡単にデプロイできるので、とりあえず作成して使ってみるというのはありだと思います。使い勝手が良ければ、本番環境への適用なども評価してみてはいかがでしょう。

変わり続けるベンダとコミュニティ

ユーザのニーズや産業業界が変わり続けるように、FreeBSDが求められる業界も時間とともに変化しています。もともとISPやエッジサーバでのニーズが高かったFreeBSDですが、現在では組込みや動画配信、仮想環境といった分野でのニーズが伸びていますし、協力関係を結ぶ企業もそういった業界の企業へシフトしています。

MicrosoftがFreeBSDのサポートへの取り組みを開始したことはすでに数年前から報道されていますが、今回MicrosoftからFreeBSD仮想環境の正式公開がアナウンスされたことは1つのマイルストーンとなるものです。今後、FreeBSDを利用する環境としてAzureはそのシーンを広げることになるものとみられます。実際、FreeBSD開発者会議ではMicrosoftから、FreeBSDのデプロイ率が増えているといった発表もありました。

ほかの*BSDへサポートが広がっていくのかはこれからの取り組みによりますけれど、すでにFreeBSDで取り組んだ成果物があることから、動作させること自体はそれほど難しくないように思います。セキュリティが重要視されるシーンではOpenBSDに高い注目がありますし、今後、ほかの*BSDにも適用が広がっていくかもしれません。今後のMicrosoftの取り組みに注目しておきたいところです。SD

▼**図5　Azureで動作するFreeBSD 10.3(セキュリティアップデート済み)**

```
3. ssh  /Users/daichi (ssh)
PARANCELL ~% ssh azure
Last login: Fri Jun 24 14:28:58 2016 from dullmdaler.ongs.co.jp
FreeBSD 10.3-RELEASE-p4 (GENERIC) #0: Sat May 28 12:23:44 UTC 2016

Welcome to FreeBSD!
```

▼**図6　Microsoft Hyper-Vで動作していることを確認できる**

```
3. dmesg  /usr/home/daichi (ssh)
    root@amd64-builder.daemonology.net:/usr/obj/usr/src/sys/GENERIC amd64
FreeBSD clang version 3.4.1 (tags/RELEASE_34/dot1-final 208032) 20140512
CPU: Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz (1112.13-MHz K8-class CPU)
  Origin="GenuineIntel"  Id=0x206d7  Family=0x6  Model=0x2d  Stepping=7
  Features=0x1f83fbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR,PGE,MCA,C
MOV,PAT,PSE36,MMX,FXSR,SSE,SSE2,SS,HTT>
  Features2=0x9e982203<SSE3,PCLMULQDQ,SSSE3,CX16,SSE4.1,SSE4.2,POPCNT,AESNI,XSAV
E,OSXSAVE,AVX,HV>
  AMD Features=0x20100800<SYSCALL,NX,LM>
  AMD Features2=0x1<LAHF>
  XSAVE Features=0x1<XSAVEOPT>
Hypervisor: Origin = "Microsoft Hv"
real memory  = 15032385536 (14336 MB)
avail memory = 14436663296 (13767 MB)
Event timer "LAPIC" quality 400
ACPI APIC Table: <VRTUAL MICROSFT>
FreeBSD/SMP: Multiprocessor System Detected: 2 CPUs
FreeBSD/SMP: 1 package(s) x 2 core(s)
 cpu0 (BSP): APIC ID:  0
 cpu1 (AP): APIC ID:  1
```

38

Debian Developer　やまねひでき　henrich@debian.org

NMプロセス変更、GitLab利用ほか、Debianプロジェクト改善の動き

Debian Hot Topics

「最小限」Debianのパッケージダウンロードサイズ

「init」パッケージの優先度がrequiredからimportantに引き下げられました。それに伴い、debootstrapコマンドを利用して最小限のDebian環境(minbase)を作成した場合、ダウンロードされるパッケージのサイズが29MBとさらに少なくなっています[注1](実際にパッケージが展開されるとディスク消費量は175MBとなります)。実験用のchroot環境や昨今流行りのコンテナ環境のイメージを作る際にはサイズが小さいほうが有利ですので、それらの作業を頻繁にされる方にはうれしい話ですね。

この環境を作成するには、次のように--variant=minbaseオプションを付けてdebootstrapコマンドを実行します。

```
$ sudo debootstrap --variant=minbase sid ➔
/srv/chroot/sid http://ftp.jp.debian.➔
org/debian/
```

OpenSSL 1.1.0対応

Debian 9"Stretch"では、OpenSSLもアップデートして1.1.0を導入する予定になっていますが、こちらの導入に対して長大なFTBFS(Fails To Build From Source、ビルドエラー)のパッケージリスト[注2]が流れてきました(筆者も含められてしまって頭が痛いです)。

OpenSSL 1.1.0を入れることでビルドができなくなるソフトウェアは、パッケージの問題というよりソフトウェアそのものの対応が必要で、Upstream(ソフトウェアの開発元)側での修正が必要なものばかりです。対応のためにはDebianパッケージメンテナからUpstreamへの働きかけが重要になりますね(この原稿が終わったら、かけ声だけでなくて自分もやらないといけません……がんばります……)。

NMプロセスの変更

現在まで、Debian開発者になるには、「NMプロセス」[注3]という官僚的事務手続きのやりとりをひとつひとつ経て、開発者として認定される、という長い処理が必要でした。長い間、不満がこぼされながらもずっと手つかずだったのですが、この処理作業を管理するサイト「nm.debian.org」に対して、Enrico Zini氏が改善をしたこと[注4]によって、大幅に変更が加わりました。

今後は「応募者自身が必要な作業を実施してサイトに入力していく」という形になり、処理

注1) URL https://lists.debian.org/debian-devel-announce/2016/06/msg00002.html

注2) URL https://lists.debian.org/debian-devel/2016/06/msg00205.html

注3) 「New Memberプロセス」の略。以前は「New Maintainerプロセス」と呼ばれていました(そのため、一部ドキュメントもそのように記載していることがあります)。Debian開発者において、パッケージのアップロード権限を持たないNon-uploading developerという区分ができたこともあり、「Member」という呼び方に変わっています。

注4) URL https://lists.debian.org/debian-devel-announce/2016/06/msg00003.html

のスピードアップが期待されています。これまでとの違いは「シーケンシャルに1つずつ手続きをしていくのか(**図1**)」、「できる作業から随時、ハブ&スポーク的に進めていくのか(**図2**)」というところです。これで応募者の待ち時間に対するフラストレーションがかなり軽減されそうです。

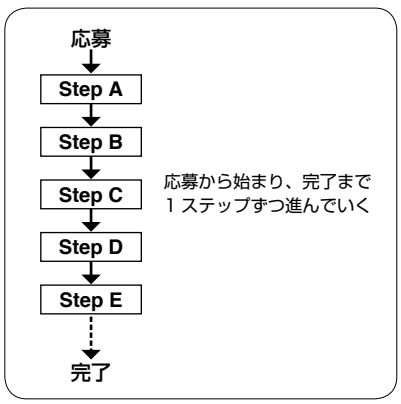
ちなみに、Debianの公式開発者になるには、次のようなステップを経る必要があります。プロセスの進捗はhttps://nm.debian.org/processから閲覧できます。

- Alioth[注5]の登録ページ[注6]にてユーザアカウントを作成する
- 登録したユーザ情報で、Debian Single Sign Onのページ[注7]にログインすると、クライアント証明書が作成されてブラウザにインストールされる
- nm.debian.orgに移動し、ログインする
- 自身のプロファイルを変更(存在しなければ作成)する
- 登録ページ[注8]で簡単な質問に答えて登録を行う。質問内容はDebian社会契約/Debianフリーソフトウェアガイドライン/Debianマシン利用規則の順守に同意するかどうか、名前と希望するアカウント名、GPGキーについてなど。さらに、自身の自己紹介とこれまでにDebianにcontributeしてきた内容やこれからDebianで実現したいことなどを記入する。これは公開メーリングリスト[注9]に送付される
- 投稿された内容の確認(confirm)のリンクが送られてくる。これは自身のGPG鍵で暗号化されており、ほかの人は読めない(これをもって本人確認としている)
- ほかのDebian公式開発者にプロファイルページから推薦(Adovocate)をしてもらう
- Application Manager(AM)から与えられたスキルチェックなど、必要なプロセスを並行してこなす
- すべてのプロセスを完了すると、AMの許可がおり、DAM(Debian Account Manager)によってアカウントが作成され、GPGで暗号化された説明メールが送られてくる(以上で完了)

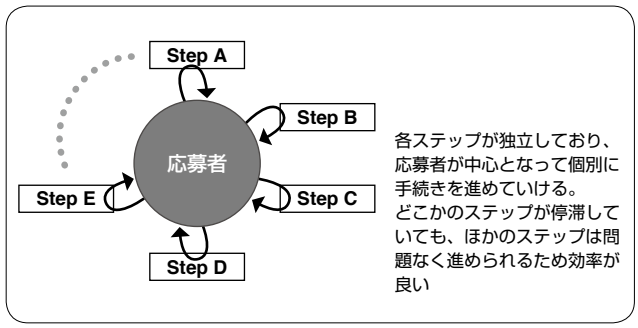
また、リニューアルされたサイトnm.debian.orgは、今回発表されたNew Memberプロセス以外の作業、たとえば「移植作業用マシンの利用申し込み[注10]」「引退からの復帰」作業など、についても対応が可能なインフラになることが期待されています。

注5) alioth.debian.orgのこと。SourceForgeと源流をともにするFusionForgeというソフトウェアを使っているDebianプロジェクト向けの開発ホスティングサービス。
注6) URL https://alioth.debian.org/account/register.php
注7) URL https://sso.debian.org/sso/login
注8) URL https://nm.debian.org/public/newnm
注9) debian-newmaint@lists.debian.org
注10) 現状の移植作業用マシンの利用申し込みについては、㈱クリアコードの林健太郎さんのブログ記事が参考になります。URL http://www.clear-code.com/blog/2016/2/24.html

▼図1　従来のDebian開発者になるまでの流れ(シーケンシャル方式)



▼図2　今後のDebian開発者になるまでの流れ(ハブ&スポーク方式)



Debianでの GitLabサービスの議論

先だってDebianでの公式gitlabパッケージについてお伝えしているとおり、gitlabパッケージメンテナの1人であるPirate Praveen氏が「GitLabをDebianでの開発サービスとして利用したい」という意思表明をdebian-develメーリングリストで行いました。すると、さまざまな視点から質問／疑問／賛成が寄せられる事態となっています。長大な議論になっていますが、争点としては次の3点が肝でしょうか。

①サービスはdebian.org配下の公式サービスとするのか、debian.net配下の試験サービスという位置づけにするのか
②DebianでのGitリポジトリサービスをAlioth から移管するのか、しないのか
③移管するとしたらGitLabが最適な代替サービスなのか。ほかのソフトウェアを使ったホスティングはどうか

①の議題については、debian.orgドメイン配下のマシン群は基本的にDebian System Administratorチーム（略してDSA）の管理下にあります[注11]。こちらで運用する場合にはDSAが要求する条件を満たさなければなりません。たとえば、security.debian.orgのミラーを日本に置いた際には、

- 最低400GBのディスクと4GB RAM
- 2、3個の静的IPv4アドレス（追加でIPv6アドレスも歓迎）
- 仮想マシンではないこと
- リモートコンソール（HP iLO、DRAC、IBM RSA、SuperMicro IPMIなど）か、リモートからアクセス可能なシリアルコンソールがあること
- ファイアウォールでの制限がないこと
- データセンターで稼働していること

というように、昨今のVPS／クラウド全盛時代には厳しい要求（とくに「仮想マシンではないこと」）が課せられました。しかし、このときはさくらインターネット㈱の協力により無事クリアでき、晴れて設置が完了しました（これ以来、アジア圏からのセキュリティアップデートで速度の不満を感じることはなくなったはずです）。

対してdebian.netドメイン[注12]の場合は、Debian公式開発者ならば自由に利用でき、サーバもとくに制限なく指定ができます（単なるDNSエイリアス設定のため）。仮に、「softwaredesign.debian.net」というドメインで筆者が何か実験をしたいとしたら、今日明日にでも利用が可能です。

Pirate氏は当初gitlab.debian.netの利用を希望していたのですが、ほかの開発者からの「gitlab.debian.orgのほうがいいのでは」という声や、DSAの一部から「foobar.debian.netからfoobar.debian.orgに将来的に移行するのは、かなりの手間だ」という声から、debian.orgでのホスティングもありか……という話になってきました。それに対して「DSAとしては、あまりメンテナンスするサーバを増やしたくない。1サービスあたり1ドメインという原則でやっているので、それなら現在のgit.debian.orgをマージするのが良いのでは」という意見が出され、②の問題に続きます。

②については、現状のAliothのメンテナであるAlexander Wirt氏から「“Open Core”[注13]なアプローチのソフトウェアだと自分たちで必要な部分をスクラッチで書く必要がある。しかも、商用版と競合する機能であれば、Upstreamにマージされない可能性が高く、ずっとアップデート

注11）「基本的に」というのは、パッケージリポジトリのftp.*.debian.orgサーバはDSA管理下ではないからです。たとえば、ftp.jp.debian.orgの場合は、複数のミラーサーバを参照するようになっています。

注12）URL https://wiki.debian.org/DebianNetDomains

注13）ベースとなるバージョンはOSSライセンスで提供するが、追加機能はプロプライエタリなライセンスで提供する、という形。今回の場合、全開発者書き込み可能なリポジトリを持つ、という機能はGitLab CEでは存在せず、商用版であるGitLab EEにしかない。

に合わせてメンテナンスをしていくコストが生じるので、あまりやりたくない(Aliothでは採用したくない)」という意見が出されます。これに対しては、「マージについて、Upstreamは耳を傾ける姿勢があると思う」とPirate氏が意見を述べました。

また、「gitlabはとても複雑なので避けたほうがいいんじゃないか？」という意見には「gitlabは巨大なRubyパッケージ群だから嫌だっていうけど、FusionForgeをメンテナンスし続けたいの？(FusionForgeも巨大なソフトウェアだよね)」という反論が寄せられています。どっちに転んでもたいへんな道には変わりないなら、もっとモダンなサービスを提供できる基盤＝GitLabなどにしたほうが良いだろう、という意見ですね(筆者も同じ考えです)。そして「GitLabの以外のalternativeなものも試してみたら？」という意見から③の問題に続きます。

③についてはhttps://wiki.debian.org/Alioth/GitNextに比較表が作られ、Gitホスティングサービスに利用できるソフトウェアが検討されています。「Gitoliteはシンプル過ぎて今回のGitLabとの比較にならない。Merge Requestが使えない」「Gogs[注14]は使っているが、現在のDebianにある20,000超のGitリポジトリは扱えないと思う」「Pagure[注15]はどうだろうか。Fedoraが作っている」「Kallithea[注16]はGitとMercurialが扱えるよ」などの情報が出されています。

3つの議題とも結論は出ていませんが、①についてはdebian.orgで始めるよりはdebian.netで始めてしまっていいのではないか、という方向のようです。GitLab社のCEOからのオファーで「GitLabがホストするVMでgitlab.debian.netを動かす」という話が出ていたので、おそらくはこちらを受け入れてgitlab.debian.net(あるいは、Pirate氏が「GitLabだとgit.debian.orgと名前が似通っていて混乱する」と言われたときに案として挙げていたshukra.debian.net[注17])でテスト運用を開始するのではないかと思います。

②については、そもそも論として今回のGitLab採用の話からちょっと飛躍させ過ぎですので、まずはGitLabを使ってみてからインフラの統合をテストするのではないでしょうか。

③については、今のところ未知数です。長い歴史から過去のしがらみ(レガシーインフラ)が多いDebianですので、機敏に新しいインフラを追加……とはいかないところがあります[注18]。とはいえ、GitLabのようなモダンなWebインターフェースを持ち、Pull Request(Merge Request)という手法に対応できる開発用インフラは今後必要性を増していきますので、いい方向に向かってほしいですね。

DebConf16開催

本誌が出るころには終了しているのですが、7月2～9日に、南アフリカのケープタウン大学にて、DebConf16が開催されます[注19]。筆者は「Microsoftが本当に『MS♥Linux』と言うなら、DebConfのスポンサーになってくれたらなー」などと自身のブログで書いていたのですが、本当にスポンサーになっていたのにはちょっと驚きました[注20]。

詳細は次号以降でお伝えできればと思いますが、公式サイト[注21]を覗いて雰囲気などを想像していただければと思います。SD

注14) URL https://gogs.io/
注15) URL https://pagure.io/
注16) URL https://kallithea-scm.org/

注17) URL https://en.wikipedia.org/wiki/Shukra ヒンドゥーの神様の名前のようです。
注18) もう1つの理由として、プロジェクトの目的が「自由なOSの作成」ですので、フリーソフトウェアではないインフラには極力頼らないという方針があるからです(そのため、昨今のクラウドも、各ベンダーのプロプライエタリなコードの上に成り立っているので第一の選択肢にはなりえません)。正直「面倒くさい」という側面があるのは否めないですが、その意義を見失ったらプロジェクトが存在する意味も消え失せますので、「こだわり」は捨てるわけにはいきません。
注19) 初のアフリカ大陸での開催です。アジア地域、とくに日本でも開催したいので、ご協力いただける組織／団体／個人の方はご連絡ください。
注20) Microsoftに勤めているDebian開発者がいるのは知っていたのですが、実際にスポンサーになることまでは想像しなかったのです。
注21) URL https://debconf16.debconf.org/

Unixコマンドライン探検隊
Shellを知ればOSを理解できる

Author 中島 雅弘(なかじま まさひろ) ㈱アーヴァイン・システムズ

コンピュータを使ううえでもっとも頻度が高い作業といえるファイル操作の基本を2回連続で紹介します。1回目は、ファイルとディレクトリの作成、複製、削除と、シェルのグロビングとbrace expansionを理解します。

第4回 ファイル操作の基本(その1)

「勇者殿、はじめての冒険なら北の村をめざすがよい。西には恐ろしいモンスターがいる。東は険しい山、南は海じゃ。南国との交易船は、後3ヵ月戻らぬ。」

はじめに、どこに行けばいいのか。モンスターとの戦い方も、サービスバトルで安全に覚えられます。RPGだったら、マニュアルなんて読まなくても冒険をはじめられます。死んでしまってもやり直しができますが……。

ファイル操作

ファイル操作の演習を開始すると間もなく、当社の新人エンジニアさんは何度もファイルシステムの中で迷子になってしまいました。GUIを使えば、俯瞰でファイルシステムを見ることができますが、CLIでは勝手が違います。初心者が文字の情報だけをたよりに、ファイルシステムの中を歩きまわるのは難しいことかもしれません。迷わないようにするにはどうしたらよいのでしょうか、実際に作業しながらコツをつかみましょう。

案ずるより産むが易し。早速システムにログインして作業開始です。はじめは、コマンドの意味がわからなくてもあまり気にせず、動作を確認しながら一歩ずつ進んでいきましょう。この小さな冒険を終えたら、きっとファイル操作の基本はばっちりです。まずは以下のようにホームディレクトリに作業環境を準備します。

旅立ちの前に

●環境の準備
```
$ cd ; mkdir Work ; cd Work↵
```

cd—Change Directory—ディレクトリの移動(シェル内部コマンド)

cdコマンドは、ワーキングディレクトリを移動します。上記「環境の準備」の最初のように引数なしなら、ホームディレクトリに移動します。cdは、ls以上に頻繁に使うコマンドでしょう。

シェルは;を改行と同じように解釈します。つまり、;はコマンドの区切りとして認識する区切り文字ですので、1行に複数のコマンドを書くことができます。コマンドは、左側のものから順番に実行され、実行されたコマンドの終了を待ってから次のコマンドを実行します。

mkdir—MaKe DIRectory—ディレクトリを作る

mkdirで、ディレクトリを作ります。

つまり上記の1行(環境の準備)で、作業用のディレクトリ(Work)をホームディレクトリに作成し、ワーキングディレクトリをWorkとして移動しました。

pwd—Print Working Directory—ワーキングディレクトリを確認する

pwdコマンドで、今いるディレクトリの位置を確認してみましょう。

```
$ pwd
/Users/masa/Work
```

OS Xの場合、通常一般ユーザのホームディレクトリはデフォルトで/Usersの下に作られるので、/Users/**ユーザ名**/Workでしょう。Ubuntuなどの Linuxでは、ホームディレクトリはデフォルトで/homeの下に作られるので、/home/**ユーザ名**/Workとなっているのではないでしょうか。

STEP UP! グロビングとbrace expansion

OSとユーザの間に入って、やり取りを仲介するのがシェルの役割です。シェルは、ターミナルでキーボードからの入力を受け、結果を画面に出力します。Linuxでも OS Xでも最近のほとんどの環境で標準的なシェルは“bash[注1]”で、古典的なUnixで標準だったsh[注2]の直系と位置づけられ機能を大幅に拡張したものです。

シェルには、ファイル名を効率よく表現するためのglob（グロブ）[注3]という機能があります。グロブは、**表1**に示すような特別な意味を持つグロブ文字を使った式で、文字列式にマッチするファイルがあれば、それを展開するしくみでとても便利ですが、理解が不十分だと予期しない動作をさせてしまうことがあります。

たとえば、カレントディレクトリに次のようなファイルがあるとします。

```
<カレントディレクトリにあるファイル>
aac  aAc  abc  Abc  bc  c
```

このディレクトリにあるファイルを対象にした、基本的なグロブによる文字列の置き換えは、

注1） バッシュと読みます。
注2） ボーンシェルとか単にシェルと呼びます。
注3） ワイルドカードとも呼ばれることがあります。

▼表1 「カレントディレクトリにあるファイル」に対してグロビングをする

グロブ文字	説明	表記	展開結果
*	任意の0文字以上の文字列とマッチ	a*	ac aAc abc
?	任意の1文字とマッチ	?b?	abc Abc
[abc]	[ ]内のa、b、cどれか1文字とマッチ	a[abc]c	aac abc
[!abc]	[ ]内のa、b、cいずれの1文字ともマッチしない1文字	a[!abc]c	aAc

表1のようになります。

bashでグロブに加えて便利でよく使うのが、文字列を生成する**{ }**（波括弧）の展開（brace expansion）と呼ばれる表記です。こちらも併せて確認しておきましょう。

echo—ECHO—文字列を表示する

brace expansionは、**,**で区切った文字列を生成します。括弧**{ }**を入れ子にすることもできます。早速文字列を表示する**echo**コマンドを使ってbrace expansionの動作を見てみましょう。

```
●brace expansionの例
$ echo {xxx,yyy,zzz}
xxx yyy zzz

$ echo *.{htm{l,},jp{e,}g}
*.html *.htm *.jpeg *.jpg
```

brace expansionでは、{***a..z***}と昇降順序がある連続する任意の文字・数字の、斜字体***a***から***z***までの範囲と指定できます。このように記述すると、***a***と***z***の間連続する文字や数字を生成します。

```
●..の例
$ echo {a..d}
a b c d

$ echo {12..8}
12 11 10 9 8
```

*、?、[、]、{、}、,などの特別な意味を持つ文字を通常の文字としてbashに認識させるには、\（バックスラッシュ）でエスケープします。

```
●\によるエスケープの例
$ echo {abc,def,d\,ef}↵
abc def d,ef

$ echo *.\{htm{l,},jp{e,}g}↵
*.{html,jpeg} *.{html,jpg} *.{htm,jpeg}
*.{htm,jpg}
```

シェルは、brace expansionやグロビング、シェル変数の展開など、さまざまな変換を**順番**に処理します。これらの処理の中でもbrace expansionは、最初に展開されます。

冒険再開

グロブとbrace expansionを理解したところで、作業を続けましょう。作業ディレクトリの下に、**図1**のようにディレクトリを作ってみましょう。

mkdirは、一度に複数のディレクトリを作ることができますので……

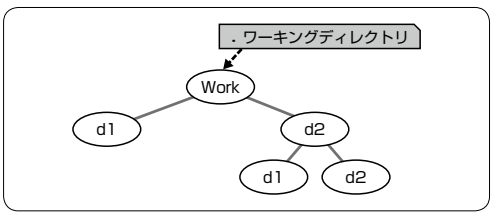
```
●失敗例
$ mkdir d1 d2/{d1,d2}↵
mkdir: d2: No such file or directory
mkdir: d2: No such file or directory

$ ls -FC↵
d1/
```

エラーが生じます。まだ存在していないd2ディレクトリの下にディレクトリを作ろうとして失敗しました。このように、まだディレクトリが存在していない階層にも作る場合は、mkdirに-pオプションを付けて実行します。

```
●成功例　:d1はまだ存在していないとして
$ mkdir -p d1 d2/{d1,d2}↵
```

うまくいきましたので、今度はファイルを作りましょう。

▼図1　ディレクトリを作る



touch—TOUCH—空ファイルを作る

touchは、空のファイルを作るときによく使います。touchコマンドは、本来はファイルの内容を変更せずに最終更新日を変更するためのコマンドで、makeやrsyncなど、対象ファイルのタイムスタンプによって動作を決定するプログラムと併せて使うツールです。存在していないファイルを指定してtouchすると、0bytesのファイルができます。

```
●touchの例
$ touch d1/f{1..3} d2/f1 d2/{d1,d2}/{f1,f2}↵

$ ls d2/d?↵
d2/d1:
f1  f2

d2/d2:
f1  f2
```

この例では、brace expansionを使って、狙いどおりうまくファイルができました。確認のためのlsで、引数にグロビングの?を使っているところにも注目してみてください。この例でのtouchをbrace expansionを使わずに書くと次のようになります。brace expansionを使うほうが短くてシンプルですね。

```
$ touch d1/f1 d1/f2 d1/f3 d2/f1 d2/d1/f1 ↵
d2/d1/f2 d2/d2/f1 d2/d2/f2
```

STEP UP! グロビングとbrace expansionを一緒に使ったときに失敗するケース

上と同じ狙いですが、次のやり方だとファイルを作る操作の一部は失敗します。これは、先ほど解説したように**brace expansionがグロビングに優先して実施され**てから、グロビングの処理がされるためです。

```
$ touch d1/f{1..3} d2/f1 d2/d?/{f1,f2}↵
touch: d2/d?/f1: No such file or directory
touch: d2/d?/f2: No such file or directory
```

touchの最後の引数部分d2/d?/{f1,f2}だけに注目してください。brace expansionが終了した状態では、d2/d?/f1、d2/d?/f2となります。これに対してグロビングを期待しても、d2/d1/f1、d2/d1/f2、d2/d2/f1、d2/d2/f2はまだ存在していませんので、グロビングを期待した?は展開されずそのままです。もちろん、d2/d1/f1……d2/d2/f2が存在していれば、グロビングされて?は置き換えられます。

ディレクトリマップを作る

CLIの操作では、視覚的にディレクトリ構造を捉えにくいために、作業対象のパスや現在作業している位置がわからなくなってしまうことで挫折してしまう人がいます。いわゆるファイルシステム内で迷子になってしまうわけです。この対策に、迷子にならないためのベストプラクティスを3つあげておきます。

【ファイルシステムで迷わないベストプラクティス】
1. 操作対象の、マップを描く
2. むやみにcdしない 歩きまわらない
3. コマンドを発行する前に、対象のパス指定(文字列)とマップ(図)を照合

GUI環境ならオートマップともいえる、視覚化されたファイルマネージャがあるので迷うことはありません。しかし、CLI(テキストによる情報のみ)の環境では自分でマッピングしなければなりません。次は、これまでに作ったディレクトリとファイルのディレクトリマップを作ってみましょう。たった今自分で作ったので、構造はすっかり頭の中に入っていると思いますが、まったく未知の環境を調査しているという想定で行きましょう。;-)

手順としては、まずpwdで自分の位置を確認します。そしてlsによって構造を調べていきます(ディレクトリ階層を俯瞰で見る方法については、別の機会に紹介します)。相対パス指定を使いましょう。むやみにcdしないのがポイントです。慣れてしまえば、マッピングをしなくても、視覚化したマップが諳(そら)んじられるようになりますので、たくさん経験を積みましょう。

では実際にやってみましょう。

```
①カレントワーキングディレクトリを確認
$ pwd
/home/masa/Work      ←Workの前のホームディレクトリ部分は、自分のディレクトリになっているはずです。

②ワーキングディレクトリの直下にあるものを確認
$ ls -FC
d1/  d2/

③d1、d2はディレクトリであるので、まとめてそれぞれの中にあるものを確認
$ ls -FC *
d1:
f1  f2  f3

d2:
d1/ d2/ f1

④d2/d1、d2/d2の中を確認
$ ls -FC d2/d?
d2/d1:
f1  f2

d2/d2:
f1  f2
```

これで、すべてのサブディレクトリの中も確認できました。図を描いてみます。**図2**のように描けましたか？　うまくいかなかったら、どこが違っているのか見なおしてください。

うまくいったら、次にファイルを消してみましょう。ディレクトリマップ上に対象と、手順を記します。**図3**を見れば明らかですが、ここではファイルは消して、ディレクトリは消さずに残していることに注意してください。

rm—ReMove—削除する

この目的を達成するためにはいろいろな表記の仕方がありますが、たとえば次のようにグロビングを使ってみましょう。

```
●図3のStep1.
$ rm d2/d2/*
```

次はbrace expansionを使って、d1の中にあるf2とf3を削除します。

```
●図3のStep2.
$ rm d1/{f2,f3}
```

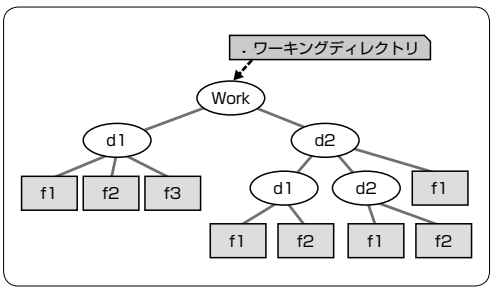
最期に、残っているd1/f1とd2/f1をまとめて……

```
●図3のStep3.
$ rm */*
rm: d2/d1: is a directory
rm: d2/d2: is a directory
```

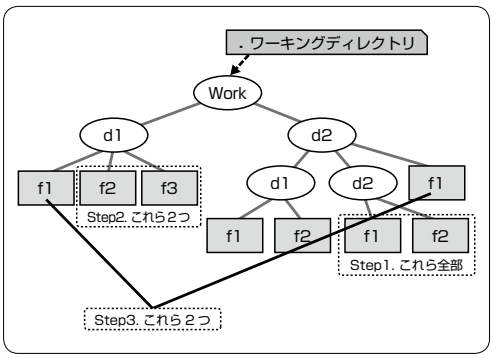
rmはデフォルトではディレクトリを削除しません。グロビングでマッチしたディレクトリd2/d1、d2/d2の削除でエラーを出力しましたが、狙いのとおりファイルはすべて消し去り、ディレクトリは残すことができました。このように、実践の現場では、エラーが生じることを利用しながら余計なコマンド発行をしないようにすることもあります。Unix系のコマンドは、エラーはもちろんメッセージが少ないのが特色ですが、コマンドが出力するメッセージには注視して狙いどおりの動作をしているか冷静に判断します。エラー即「いけないこと」と早計に判断しないようにしましょう。

さて、次はディレクトリの削除をしてみましょう。こちらも現在のディレクトリマップ**図4**に手順を示しますので、挑戦してみてください。

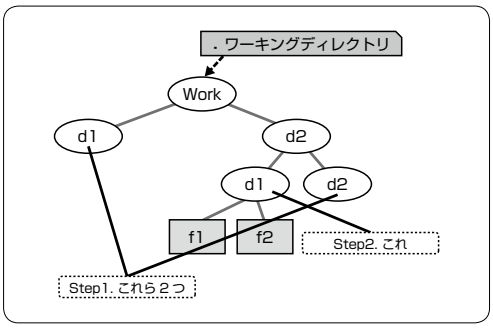
▼図2　ディレクトリマップを作った



▼図3　ファイルを消す



▼図4　ディレクトリを消す



rmdir—ReMove DIRectory—ディレクトリを削除する

rmdirコマンドはディレクトリを削除するコマンドです。ディレクトリの中が空っぽでなければディレクトリを削除することはできません。

```
●図4のStep1.
$ rmdir d1 d2/d2↵

●図4のStep2.
$ rmdir d2/d1↵
rmdir: d2/d1: Directory not empty
```

rmdirは、空でないディレクトリを削除することはできないので失敗しました。

このような場合は、中身をきちんと消してからディレクトリの削除をおこなうか、rmの-rオプションを使って指定したディレクトリ以下のすべてのファイルとディレクトリを削除します。実行すると、確認もなく取り消しが効かない操作ですので、十分慎重に実行するようにしましょう。

```
●図4のStep2. 成功例
$ rm -r d2/d1↵
```

> rm、rmdirは一度実行してしまうと取り消しが効きません。ゴミ箱に一度入るということもありませんので注意が必要です。
> rmを使うとき、一度に消してしまう自信がないときには、-iオプションを指定します。そうすれば削除対象のファイルやディレクトリを削除するかどうか、1つずつ確認のプロンプトが出ますので、消していい場合にはyを入力します。

さて、次はファイルのコピーです。今、すべてのディレクトリの中にファイルはありません。次のようにd1の中にf1、f2、f3を作っておき

ましょう。

```
$ touch d1/f{1..3}
```

図5の状態になったはずです。この状態で、**図5**の指示のようにワーキングディレクトリを./d2に移動してからコピーをおこないます。

cp—CoPy—ファイルのコピー

cpコマンドは、ファイルをターゲットとなるファイルにコピーします。ターゲットがディレクトリである場合は、複数のコピー元ファイルを、ディレクトリ内にコピーすることができます。

cpに-rオプションを指定すれば、コピー元がディレクトリである場合は、ディレクトリの中もすべて階層的にコピーします。

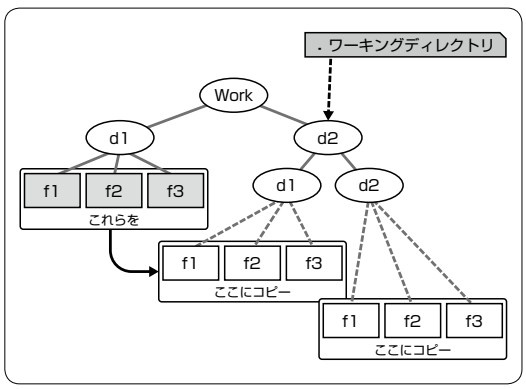
```
●複数のコピー先を指定した失敗例
$ cp ../d1/* {d1,d2}↵
cp: d1 is a directory (not copied).
$ ls *↵
d1:

d2:
f1  f2  f3
```

上のやり方では失敗します。狙いは、コピー先にd1とd2の2つのディレクトリをまとめて指定したかったのですが、d2にのみコピーされました。d1は../d1と同様、コピー元と判断され、コピー元がディレクトリであるためにエラーが表示されました。cpのコピー先は、1つのディレクトリかファイルでなければならないことに注意しましょう。

この操作は1度にはできませんので、次のように;で区切って2回のcpとして処理します。

```
$ cp ../d1/* d1 ; cp ../d1/* d2↵
```

▼図5　コピーする



今回のまとめと次回について

今回は、ファイルとディレクトリの作成、複製、削除といった基本的な操作を見てきました。そして、コマンドライン上で効率よくファイル操作をするには、シェルのグロブやbrace expansionを理解することが重要ということを解説しました。

ファイルの削除や移動[注4]は、重大なシステムへの変更を加えることになる場合があります。慎重に作業を進めたいときは、実際に変更を加えるコマンドを実行する前にechoコマンドで対象と指示が適切であるか確認しておけば、間違いを減らすことができます。

次回は、引き続きファイルの移動、リンクといった操作と権限について紹介します。

今回の確認コマンド

【manで調べるもの（括弧内はセクション番号）】
mkdir(1)、pwd(1)、echo(1)、touch(1)、rm(1)、rmdir(1)、cp(1)
【以下はbashのhelpコマンドを使って確認】
cd、echo

よく使う、rm、mkdir、rmdir、cpなどにも便利なオプションがたくさんあります。また、一般ユーザで使う場合とスーパーユーザで使う場合に振る舞いが異なるオプションなどもありますので、しっかりとmanで確認しておくとよいでしょう。echoには、コマンド/bin/echoとbashの内部コマンドのechoの両方がありますが、この手のお話はあらためて…… ;-) SD

注4）移動については次回解説します。

第53回

Linux 4.2の新機能
～CDGによるTCPの輻輳制御

Text：青田 直大　AOTA Naohiro

6月20日にLinux 4.7-rc4がリリースされています。うまく進めば、本誌発売のころにはLinux 4.7がリリースされているのではないでしょうか。

今回は、Linux 4.2で追加された新しいTCPの輻輳制御手法であるCAIADelay-Gradient(CDG)について解説します。

TCPの輻輳制御

通信では、その経路上の帯域を可能な限り多く使うのが望ましいです。しかしTCPでは経路上の帯域について何も仮定を置きませんし、仮に帯域がわかっていたとしてもほかの通信と経路上の機器を共有することから、自分だけがその帯域をフルに使うわけにはいきません。好き勝手に大量にデータを送信してしまうと、経路上で問題が発生してしまいます。

たとえば、経路上にあるルータに大量のパケットが入ってきているとします。ルータはパケットを処理し送信するためのバッファを持っていますが、大量のパケットが来て混雑してくると、やがてバッファにパケットを保持できなくなり、パケットを破棄してしまいます。こうしたパケットの混雑を輻輳と呼びます。TCPはACK(ACKnowledge)が来ない場合、一定のタイムアウト後にパケットを再送するようにできていますが、ここでまた一度に送るパケット量を変えずに送信していると、混雑状況はより悪化していきます。

そこでTCPでは、輻輳の発生を自分で検出し、通信量を調整する輻輳制御という機能が古くから実装されています。輻輳の検出には大きく分けて2つの方法があります。1つの方法ではパケットロスを用い、もう1つの方法ではパケットの遅延を用いて輻輳の検出を行う方法です。

パケットロスを用いた輻輳制御

まず、パケットロスを用いた輻輳制御アルゴリズムの1つであるTCP NewRenoについて見てみましょう。これは多くの環境で実装されている輻輳制御アルゴリズムで、FreeBSDではデフォルトとなっています。

通信開始時の輻輳ウィンドウサイズ(cwnd)は1となっています。そのあと、SlowStartというモードでウィンドウサイズを増やしていきます(**図1**)。Slow StartではACKが返ってくるごとにこのウィンドウサイズを2倍にしていきます。SlowStartには“ssthresh”という閾値が設定されており、ウィンドウサイズがssthres以上にな

るとよりゆっくりウィンドウサイズを増やしていく輻輳回避モードに切り替わります。通信開始時にはssthreshはかなり大きく設定されているため輻輳回避モードには入ることなく、ウィンドウサイズは指数的に増大し、やがて回線がいっぱいになってパケットロスが発生します。

ウィンドウに余裕があれば、送信側はロスが発生したパケットよりも後にも、ACKを待たずにパケットを送信しています。これら後続パケットが破棄されずに受信側に到着すると、受信側は最後に受け取ったパケットに対応するACKを送り返します。このACKは送信側には重複したACKとして観察されます。重複したACKを3度受け取ると、送信側はパケットロスが発生したと認識します。

後続パケットがない場合や、後続パケットも破棄されてしまった場合には、重複したACKも帰ってきません。その場合、RTO(Retransmission Time Out)で設定された時間待ってもACKが帰ってこないことをパケットロスの発生とみなします。

パケットロスが起きた場合、通信経路の転送量上限は現在のウィンドウサイズから前回のウィンドウサイズの中間のどこかにあるということができます。そこでssthreshを現在のウィンドウサイズの半分に設定します。ここから処理はパケットロスを検出した方法によって分かれます。タイムアウトが起きた場合、多くのパケットロスが発生していることが予測されます。そこでウィンドウサイズを1まで戻して、Slow Startをやりなおします。

今度はssthreshが設定されているので、前回のパケットロス発生時の1つ前のウィンドウサイズまで来ると輻輳回避モードに入ることになります。一方、重複ACKによりパケットロスを検出した場合、後続のパケットが受信できていることからそれなりに回線の容量が空いていることが予想されます。この場合、ウィンドウサイズを1まで戻してしまうと利用できる帯域を無駄にしていることになります。そこで重複ACKの場合には次のウィンドウサイズをssthreshと同じく現在のウィンドウサイズの半分に設定してSlow Startモードをスキップし、輻輳回避モードを開始します。

輻輳回避モードでは、ACKごとにウィンドウサイズを“ウィンドウサイズ分の1”(1/cwnd)ずつ増やしていきます。これによってウィンドウサイズは線形に増えていくことになります。

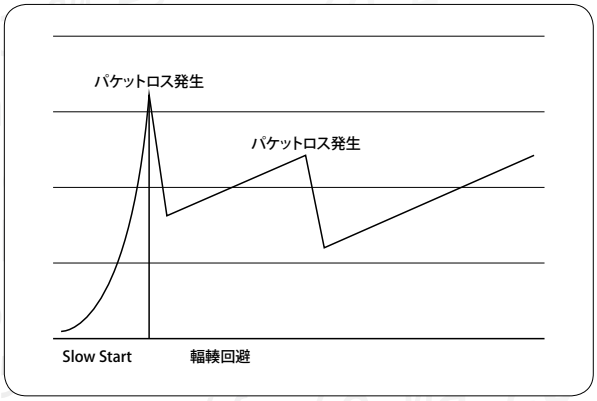
パケット遅延を用いた輻輳制御

輻輳を検出するもう1つの方法では、RTT(Round-Trip Time)の増加により輻輳の判定を行います。通信経路が混雑してくると、通信経路上のルータのバッファが埋まってくるのでパケットがバッファ内で保持されている時間が長くなり、RTTが増加します。

おもにパケット遅延を輻輳制御に用いる方法の1つであるTCP Vegasについて見ていきましょう(**図2**)。全体的な構成はNewRenoと同じで、ssthreshまではSlow Startモードで、ssthresh以上になると輻輳回避モードになります。ポイントは、

❶Slow Start時にどのようにウィンドウサイズを変化させるか
❷どのようにssthreshを設定するか
❸輻輳回避時にどのようにウィンドウサイズを変化させるか

▼図1　パケットロスを用いた輻輳制御



の3点になります。

まずSlow Start時の挙動からみていきましょう。VegasにおいてもACKごとにウィンドウサイズを倍にしていきます。ただしNewRenoのようにパケットロスが起きるまで増大させていくのではなく、RTTを監視して遅延が始まると倍増を止めてssthreshを設定し、輻輳回避モードに入るようになっています。

具体的にLinuxのコードを見てみましょう。Linuxに実装されたVegasでは、次の式でdiffを計算します。

```
diff = cwnd * (rtt-baseRTT) / baseRTT;
```

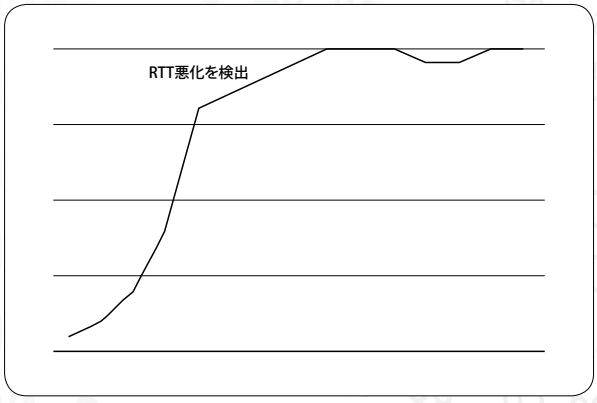
cwndは現在のウィンドウサイズ、rttは前回ウィンドウサイズを計算した時点以後に受信したACKのRTTの最小値になります。baseRTTは、通信期間全体でのRTTの最小値です。baseRTTが通信経路が一番空いていたときのRTTを示し、rttは現在のウィンドウで通信経路が一番空いているときのRTTを示すことになります。したがって、diffは最速の状態で保持できるパケット数と比較して、どれだけ多くのパケットがネットワーク上に乗っているのかを示すことになります。

このdiffがパラメータgamma(Linuxで、デフォルトは1)より多くなると、ネットワークが混雑してきたと判断し、ssthreshを設定して輻輳回避に入ります。このとき、次のウィンドウサイズは次の式で計算され、新しいウィンドウサイズでのRTTがbaseRTTと同じ程度になるように調整されます。また、ssthreshは新しいウィンドウサイズの1つ下に設定されます。

```
cwnd = min(cwnd, cwnd * baseRTT / rtt);
ssthres = cwnd - 1;
```

次に輻輳回避時の動きを見ていきましょう。輻輳回避時も先ほどと同様に、最速状態に比べてどれだけ多くのパケットがネットワーク上に乗っているのかを示すdiffを計算します。この値がパラメータalphaとbeta(Linuxではデフォルトで2と4)の間にあれば現在のウィンドウサイズを保持します。diffがalphaよりも少なければ、ウィンドウサイズを1増やしてネットワークにより多くのパケットを乗せるようにします。逆にdiffがbetaよりも多くなっていれば、ウィンドウサイズを1減らしてネットワーク上にあるパケット数を減らします。ウィンドウサイズを減らした結果、ssthreshと新しいウィンドウサイズが一致した場合にはssthreshの方も1つ減らします。

▼図2　TCP Vegasにおける輻輳回避のイメージ



パケットロス方式とパケット遅延方式の比較

ここでパケットロスを用いるNewRenoとパケット遅延を用いるVegasについて比較してみましょう。NewRenoはパケットロスを輻輳の判定に使うために、破棄されたパケットとその再送の分だけスループットが悪化することになります。一方でVegasではRTTを観察することでロスなしに輻輳を検出するので、NewRenoよりもよいスループットを実現すると言われています。

一方でVegasはNewRenoと同じ環境で動かした場合にNewReno側に帯域を食われてしまう問題があります。この2つが同じルータ上で競合し混雑してきた場合、VegasはRTTの遅延を検出し、ウィンドウサイズを減少させますが、NewRenoで

はパケット破棄が起こるまではウィンドウサイズを増加させるのでVegasが減らした分を奪いとるように振る舞います。

また、パケットロス方式は無線環境下において問題を起こし得ることが指摘されています。無線LANなどの環境では、エラーレートが低い有線環境とは異なり輻輳以外での原因でパケットが破棄されることがあります。するとパケットロス方式ではこれを輻輳発生とみなしてウィンドウサイズを縮小し、スループットを低下させてしまいます。さらに、パケット破棄を前提とした方式ではVoice over IPのようなパケット破棄に対して敏感なアプリケーションに悪影響をおよぼします。

CAIA Delay-Gradient(CDG)

Linux 4.2で追加されたCAIA Delay-Gradient(CDG)は、その名のとおりパケット遅延をベースにした輻輳制御です。CDGの特徴は、

❶パケット遅延の変化の勾配を用いて輻輳を検出する
❷確率ベースで平均的にはRTTと独立して、ウィンドウの縮小を行う方法を採用
❸パケットロス方式のフローと共存できる
❹輻輳以外の原因によるパケットロスへの耐性

の4つが挙げられます。

「❶パケット遅延の変化の勾配を用いる特徴」について見ていきましょう。VegasではRTTの値をそのまま使って輻輳の判定を行っていました。一方でCDGではRTTの変化を使って輻輳を検出しています。すなわち、一回のRTT期間ごとに、その期間中のRTTの最大値・最小値を記録し、その差分が大きくなれば輻輳が発生しているとみなします。さらに一回分の変化だけを見ると急な変動に弱くなってしまうので、過去数期間分(Linuxのデフォルトでは8期間)の移動平均を用います。

CDGをVegasと比較してみましょう。Vegasではbase RTTとalpha、betaの値がその動きを制御します。TCPの特性上、真のbase RTTを知ることはできないので、通信中のRTTの最小値で推定し得るという弱点があります。同様にどれだけ多くのパケットをネットワーク上に乗せてよいかを示すalphaとbetaの値も適切な設定はその通信経路によって変わってくるはずです。こうした弱点からRTTを直接使う手法はインターネット規模でのデプロイは難しいと指摘されています。一方で遅延の勾配であれば経路上のbase RTTに依存せず値を取得できますし、経路上の状況の変化に強くなります。

次に「❷確率ベースのウィンドウ縮小」について見ていきましょう。CDGでは次の確率Pでウィンドウの縮小を決定します。

$$P(g) = 1 - e^{-(g/G)}$$

※eは自然対数の底

ここでgは、❶で計算されるRTTの変動分の移動平均で、Gは変動分に対する確率を調整するパラメータになります。この確率Pが変動分gに対して指数的に変化することから、小さいRTTの経路(変動も小さいことが予想される)でも、大きなRTTの経路(変動も大きいことが予想される)でも、平均的にはPが同等になるとCDGの論文中では主張しています。

たとえばg/Gの変動がaからbの間で起きる場合、縮小が起きる確率の平均P_{avg}は次の式で与えられます。

$$P_{avg}(a,b) = \frac{1}{b-a}\int_a^b \left(1 - e^{-x}\right) dx$$

変動の中心をxとしてa=x/2、b=3x/2として$P_{avg}(x/2, 3x/2)$をx=0.001からx=1の範囲で計算すると、その確率は0から0.6まで変動し、平均的に確率が一致しているようには見えません。

おそらくポイントはこの確率評価がRTTごとに起きることにあるかと思われます。ある一定の期間T内にウィンドウ縮小が少なくとも1回起きる確率P_tを考えるとRTTが短いものは先ほどの平均確率が低い一方で、確率評価される

回数が増えて結果としてP_tは大きくなります。P_tは次の式で与えられます。

$$P_t(T,a,b) = 1 - (1 - P_{avg}(a,b))^{(T/b)}$$

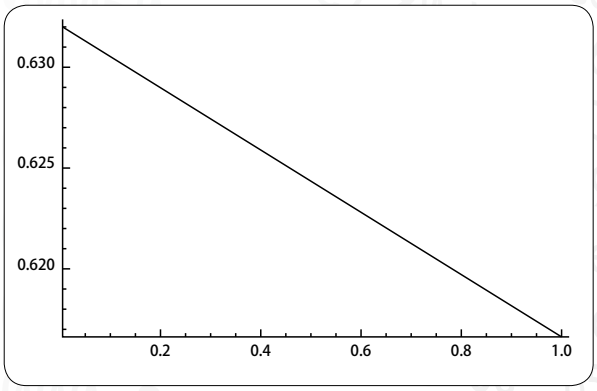
期間T=1として、先ほどと同様にしてx=0.001からx=1までプロットすると**図3**のようになり、RTTの変動が1,000倍違っていても確率は1%ほどしか変動していないことになります。直感的に解釈するとRTT変動が10倍になることで確率評価回数が1/10になり、それと指数関数が打ち消しあって一定期間中の平均確率が一定に近付くということになります。

経路のRTTに依存して、挙動が変わらないというのも重要な性質です。たとえばNewRenoではSlow Start時でも輻輳回避時でもRTTごとにウィンドウが2倍または1増えていきます。RTTが大きく違うフローが共存していると、まったく同じタイミングでパケットロスが発生したとしてもRTTが短いフローの方が先にウィンドウを拡大し帯域を奪っていくことになります。

「❸パケットロスとの共存」について見ていきましょう。Vegasではパケットロスベースで動く輻輳制御に対して帯域を食われてしまう問題がありました。CDGでは2つの手法でこの問題を回避しています。

1つはIneffectual backoff detectionという方法です。先ほどのルールにしたがってウィンドウを縮小すると通常はその分帯域の使用量が削減され、RTTが減少することが予想されます。

▼図3　ウィンドウ縮小が少なくとも1回起きる確率



その予想に反してRTTが増加している場合、ほかのフローによって自分が減らした分の帯域が食われてしまったと予測できます。そのような状況を観測したときは、ウィンドウの縮小を行わず、ロスベースの方式に対抗します。

もう1つはShadow windowと呼ばれる方法です。これはRTT変動によるウィンドウ縮小を行ったあとに、もし自分がNewRenoであればどのようにウィンドウサイズが変化していたのかをshadow windowとしてトラッキングし、パケットロス発生時には実際のウィンドウサイズではなく、shadow windowの半分を新しいウィンドウサイズとして設定する方法です。

CDGとNewRenoのフローが同じタイミングで同じウィンドウサイズになったとして、そのあとのウィンドウサイズの変化を見ていきましょう(**図4**)。

①初期はまだネットワークが混雑しておらず、CDGもNewRenoも同様にウィンドウサイズを拡大していきます。
②その後、どこかでRTTが悪化しCDGはウィンドウサイズを縮小します。一方でNewRenoは、その後もウィンドウを拡大していきます。このときShadow WindowはNewRenoと同様に増えていきます。
③CDGは、NewRenoに食われた分さらにウィンドウを縮小する一方で、NewRenoはまだウィンドウを拡大します。
④やがてパケットロスが発生します。

ここで、もしCDGがウィンドウを実際のサイズの半分にしてしまうと、ずっとNewRenoに敗け続けてしまいます。代わりにShadow Windowを使えば、NewRenoに追い付くことができます。ある種、RTT悪化によるウィンドウ縮小で失った分をパケットロスで取り返しているとみなすことができます。

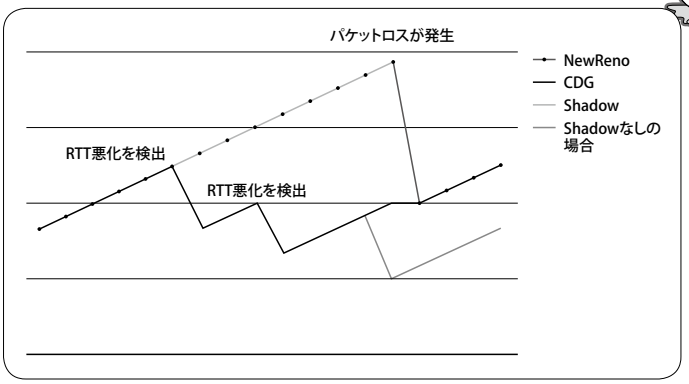
最後に、「❹輻輳以外の原因によるパケットロスへの耐性」について見ていきましょう。先ほど述べたようにCDGはパケット遅延だけでなく

パケットロスでもウィンドウサイズを調整します。しかし、すべてのパケットロスに反応するわけではなく、輻輳に対応するパケットロスにだけ反応するようになっています。

では、どうやって輻輳によるパケットロスを見分けているのでしょうか。その鍵はRTTの最大値・最小値の変動、g_{max}、g_{min}を監視していることにあります。パケットロスが起きるルータに注目して考えてみましょう。ほかの部分をひとまず無視すると、RTTはこのルータのバッファ量に依存して増減します。するとRTTの最大値の上限は、このルータのバッファがいっぱいになっているときのRTTになることがわかります。すなわちルータのバッファがいっぱいになりかけている状況ではRTTの最大値はバッファがいっぱいのときの上限は貼りつく一方で、RTTの最小値はまだ増加する余地があるということになります。輻輳によるパケットロスが起きるときには、ルータのバッファがほとんどいっぱいになっているはずですので、RTTの最大値の変化が止まっているのに、最小値はまだ変化しているかどうかであるパケットロスが輻輳によるものかどうか判別できるわけです。

特徴を一通り見たので、全体的な動きについてまとめます。通信開始後はSlowStartでウィンドウを拡大しつつ、RTTの変動を監視します。変動が大きくなってくれば、変動量に応じた確率でウィンドウを縮小します。縮小量は実装によりますが、新しいウィンドウをおおよそ元の7割に設定します。このとき、縮小後のサイズがssthreshに設定されます。ssthresh以上の領域ではNewRenoと同様にウィンドウサイズをACKごとに1つ増やしていきます。

▼図4　ウィンドウサイズの変化



Linuxにおける CDGの実装

最後に簡単にLinuxにおけるCDGの実装についてまとめます。Linuxの実装ではIneffectual backoff detection、Shadow Window、および輻輳以外のパケットロスの検出機能がパラメータでオン／オフできるようになっています。デフォルトでは、Ineffectual backoff detectionは5回分のRTTの増加を無視する、Shadow Windowは有効、輻輳以外のパケットロスの検出は無効に設定されています。

また、Slow Startの部分にはHyStartというアルゴリズムを使用しています。これはLinuxのデフォルトの輻輳制御であるCUBIC TCPのSlow Startに実装されているものと同じアルゴリズムで、高帯域の環境において従来のSlow Startが起こす問題を回避するものです。

従来のSlow StartではACKごとにウィンドウサイズを2倍にしてきました。SlowStartはパケットロスによって初めて停止するので、ウィンドウサイズは最大で帯域の2倍まで成長し得ることになります。高帯域になるほど最初のパケットロス時に発生するロスの量は指数的に増大していくことになります。また、この時送受信両方で大きなCPU usageが発生します。HyStartはACK Trainという方法で回線の帯域幅を推定し、またRTTを観測し最小のRTTよりある程度遅くなってきたところでSlow Startを早めに離脱するという方法をとります。後者はVegasと似たところがありますね。SD

August 2016

NO.58

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
榎 真治　ENOKI Shinji　enoki-s@imail.plala.or.jp

IT勉強会共通の悩み？　集客の工夫／長く続ける秘訣

今回は5月に名古屋で行った研究会の模様をお伝えします。

jus研究会 名古屋大会

■ITコミュニティの運営を考える
【講師】You&I（名古屋アジャイル勉強会）
マツモトサトシ（名古屋ギークバー）
榎 真治（日本UNIXユーザ会／LibreOffice日本語チーム）
法林 浩之（日本UNIXユーザ会）
【日時】2016年5月28日（土）14:00～14:45
【会場】名古屋中小企業振興会館 吹上ホール 4F

jusではこの1年間、「ITコミュニティの運営」をテーマに全国各地で研究会を行っています。講師全員がコミュニティ運営に関する課題や疑問を1つずつ挙げて、それに対して全員が回答してディスカッションします。今回は名古屋のITコミュニティ運営で活躍されているお二人を招いて開催しました。オープンソースカンファレンス2016 Nagoya（以下OSC）の中での開催で、参加者は30名でした。

■勉強会の存在をいかに伝えるか

1つ目のお題はマツモトさんからで「勉強会の存在をいかに伝えるか？」でした。DoorkeeperやTwitterでしか告知できていないこともあるそうですが、OSCでブース出展し、さまざまなイベントを開催していても、「名古屋で勉強会がないですね」と言われるそうです。そこで、「参加者はどこでイベントを知るのか？」について議論になりました。

You&Iさんは申込サイトのconnpassでアンケートをとっており、イベントを知った先はFacebookが多く、最近ではdots.などもあるそうです。アンケート結果は回によって違うようで告知方法の決め手はなさそうでした。dots.のようにさまざまな申込サイトからイベント情報を収集して掲載するサイトが増えてきたので、申込方法を以前のメールから外部サイトのconnpassに切り替えたそうです。今回のOSCはどこで知ったかを会場に挙手で調査したところ、いつも参加している方が多かったのですが、ほかの勉強会で知った、法林さんのツイートで知った、という方もいました。

You&Iさんは、わんくま同盟の名古屋ディレクターでもありますが、昨年10周年と歴史のある勉強会であるせいか、告知はTwitter程度とサボっていても名古屋での集客は20名程度だったそうです。一方、名古屋アジャイル勉強会は告知をがんばってみても10名以下であり、告知への力の入れ具合と集客数とは必ずしも比例していません。最近参加者が減っている理由はアジャイルが普及したせいかもしれませんが原因はつかみきれていないそうです。

筆者（榎）もLibreOfficeの知名度UPや、そのイベントの告知に苦戦しており、最近では告知目的でFacebookイベントを立てて興味がありそうな方を選んで招待していることを紹介しました。

会場からは「何ができるのかが明確なほうが行きやすいのではないか」というコメントがありました。法林さんからは「行く側は何かが得られるのかなという意識かもしれないが、運営側はそこを気にして

いない場合もあり、ギャップがあるかもしれません」とコメントがありました。

■勉強会で勉強しているか

次のお題は、You&Iさんの「勉強会で勉強していますか？」でした。名古屋アジャイルでは、実際にやらないと身につかないので、講師を呼ばずにワークショップスタイルにこだわっているそうです。運営側としてどう意識しているかという問いかけでした。

マツモトさんの場合、イベントによって違うとのことで、次のような話がありました。

- ギークバーは週のあと４日間を生き延びるために月曜に集まり楽しくやれるもの、Ruby東海もミートアップでありわいわい楽しくするもの、どちらも身にならないこともあった
- スタートアップウィークエンドは３日間で起業家を育てるプログラムなので、しっかり作られていた
- CSNagoyaという読書会は心が折れないためのもので勉強になっていたが、効率は悪いものだった

筆者からは、関西LibreOffice勉強会は集まりであって、勉強が目的ではないことを話しました。役立つ話も多いですが、運営側も聞いてみるまで勉強になるかどうかはわかりません。スピーカーは立候補で募るスタイルであり、運営側は場の提供に徹してコンテンツをコントロールしていないからです。

マツモトさんからは、確実に勉強になるにはスピーカーをやることというコメントがありました。スピーカーは調べる必要があり勉強になります。これにはほかのパネリストからも賛成の声がありました。

法林さんからは運営側は話を聞けないこともあり、それが割り切れるかどうかが運営に関われるかどうかかもしれないとのことでした。

また、会場からは勉強になっても活用する機会がないこともある、というコメントもありました。

■長く続ける秘訣

3つ目は、筆者から「長く続けられている秘訣は？」という質問をしました。

You&Iさんは、スタッフとして参加するコミュニティを3つに決めているそうです。モチベーションとしては、発表者をすることで勉強になる、その場を自分で作っているというものだそうです。自分1人でできるようになるとスタッフが減ってしまったので、やり過ぎは良くないという話もありました。

マツモトさんは、毎週やっているのでリズムとして楽であること、準備がとくに不要なことが続いている理由だそうです。毎週でないイベントのほうが逆に続けられていないそうです。

法林さんは、学生のころにライブをやっていたこともあり、人に見てもらう、集まってもらうのが好きというのが続けられている理由、とのことでした。

筆者は、LibreOfficeへの関わりはライフワーク的なもので、イベントに対してモチベーションを高める意識はありません。続けられている理由は、会場担当の方と最低2名でできていることが大きいです。2名以上いると心が折れることがないと話しました。

名古屋アジャイル勉強会ではスタッフもアジャイルで回しており、内容の質を上げるため、あるいは発表者が参加できないというトラブルを防止するために、スタッフレビューをしているそうです。

■運営に向いている人／向いていない人

最後は、法林さんの「運営に向いている人と向いていない人がいると思うか？」というお題でした。

マツモトさんは、自分自身は向いてないと思うが、やったら続けられているそうです。You&Iさんによると、やるかやらないかの覚悟の問題が大きいそうです。筆者は、やりたいかやりたくないかのほうが大きいのではないとコメントしました。

会場からは、スキルセットのうち、モチベーションが一番大事ではないかとのコメントがありました。

◆　◆　◆

本研究会の動画が以下に公開されています、詳しい内容に興味を持たれた方はご覧ください。SD

URL https://www.youtube.com/watch?v=GsfW8Nv_MoY

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
及川 卓也　OIKAWA Takuya
Twitter @takoratta

第56回

情報支援レスキュー隊の熊本地震対応

4月14日の夜および16日深夜に発生した地震を代表とする一連の熊本地震は、熊本県と大分県を中心に九州各地に甚大な被害をもたらしました。本原稿執筆時もまだ予断を許さない状況であり、復旧もまだ始まったばかりですが、この地震に対し、ITがどのように活用されたかを筆者がスタッフを務める一般社団法人情報支援レスキュー隊の活動を通じてご紹介します。

情報支援レスキュー隊

一般社団法人情報支援レスキュー隊（略称：IT DART）については、この連載でも何度か紹介させていただいています。災害発生直後より情報支援の立場からITで被災地を支援する活動を目指し、検討や訓練を重ねてまいりましたが、2015年8月に一般社団法人として正式に発足いたしました。

その後、昨年9月に発生した関東・東北豪雨災害において、IT DARTは茨城県守谷市のサポートや、栃木県小山市および茨城県常総市のニーズ調査を行いました。

熊本地震への初動

2016年4月14日の夜、後に前震とされた地震が発生した後、IT DARTでは支援を志す人たちのためのFacebookグループを設置しました。IT DARTに関係する方々だけではなく、広く支援を考える人たちの情報共有の場となり、メンバーは4,000人を超えるまでになっています。

4月16日には4名からなる先遣隊を派遣しました。

支援体制

今回の支援は、現地入りするメンバーとそれを支える後方支援チームという体制で行われました。4月16日と17日の先遣隊派遣の後も、IT DARTのメンバーが代わる代わる現地入りしましたが、そこで汲み上げられたニーズを元に、後方支援チームとともに調査や開発を行いました。

昨年秋の関東・東北豪雨災害までは、活動はIT DARTの理事と運営委員だけで行っていましたが、今回は隊員も後方支援活動に参加しました。

活動を支えるためにおもに用いたツールがSlackでした。現地入りしたメンバーと後方支援チームが連携し、多様なタスクを迅速にこなす必要がありましたが、情報量が多く、内容も多岐にわたるため、メーリングリストや1つしかカテゴリを持たないようなチャットやグループサービスだと恐らく情報が埋もれてしまっていたでしょう。Slackの持つ複数チャンネル機能、投稿ごとにユニークなURLが付与される機能、高度な検索機能などを活用しました。

Slack上では、プロジェクトやタスクごとに15のチャンネルが作られ、その状況を運営委員2名が毎朝まとめ、熊本地震のメインチャンネルに投稿するようにしました。

支援活動

IT DARTでは次の3種類の支援活動を行いました。

- 情報流通の支援
- 情報システムの開発
- 通信環境整備の支援

情報流通の支援

情報流通の支援は、災害対応を行う組織や個人に対して、情報の収集と利用そして発信という運用を支援する活動です。具体的には次のようなプロジェクトが行われました。

●災害ボランティアセンター募集状況一覧作成

IT DARTでは、熊本県や大分県のNPOや災害ボランティアセンターによるボランティアの募集状況を収集し、一覧を作成しました。最新の情報は毎日IT DARTのTwitterアカウントから発信しています。

2011年の東日本大震災では、タイムリーに正しい情報が発信されないことが原因で、ボランティアが集まり過ぎたり、逆にまったく集まらなかったりする地域が出てきてしまいました。

そのときの経験を教訓とし、災害ボランティアセンターやNPOなど、これまでに計35ヵ所の募集情報を一覧表にして毎日更新するとともに、@it_dartから毎日約30件をツイートしています。

●自治体HPレスキュープロジェクト[注1]

これは、被災者および支援に必要な自治体Webサイトからの情報発信をバックアップするためのプロジェクトです。

災害発生後、自治体のWebサイトがダウンしたり、つながらなくなることが多くあります。今回の熊本地震においても、複数の自治体や公共団体のサイトがアクセス不能に陥りました。理由は停電などの電源の問題からアクセス過多までいろいろありますが、必要なときに必要な情報が得られないことは、ともすれば生死にかかわる問題に発展しかねません。

そこでIT DARTでは、熊本県と大分県の自治体Webサイトを定期的にモニターし、サイトがダウンしていないか、また更新されているかをチェックするしくみを作り上げました(**図1**)。更新されているかどうかは、HTTPヘッダの最終更新日(Last-modifiedフィールド)を用いましたが、このフィールドをサポートしていないサイトやサポートしていても実態に即していないサイトが多かったため、これに加えて、実際にトップページの差分を取るというアプローチもあわせて行うようにしました。また、レスポンスタイムも取得してサイトの状態を類推しようとしましたが、こちらはあまり参考にすることはありませんでした。

現在のところ幸いにして出番はありませんが、もし自治体サイトがダウンしていた場合にはJimdoなどの簡易サイト制作サービスを用い、最低限の情報を掲載した暫定サイトを立ち上げる準備もしてあります[注2]。

なお、このプロジェクトはさくらインターネット㈱からサーバを支援いただきました。

●「詐欺に注意」チラシの作成と配布

残念な話ですが、被災地では詐欺が横行します。不届きな輩(やから)が善意のふりをして、被災地の方々を騙(だま)そうとします。IT DARTではこういった詐欺への注意を喚起するチラシを準備しました(**図2**)。FacebookページやTwitterアカウントから拡散するとともに、現地のコンビニエンスストアでも直接印刷できるよう、ネットプリント(セブン-イレブン)とネットワークプリント(ローソン、ファミリーマート、サークルKサンクス、セイコーマート)にもファイルを登録し、それぞれを用いた印刷の仕方とともに公開しました。

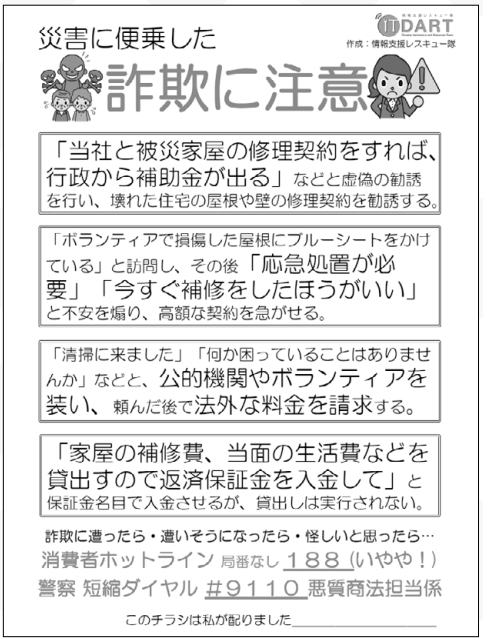
このチラシの情報はFacebookで5万人を超える方にリーチはしたのですが、一方で実際にコンビニエンスストアで印刷された件数は大変少なく、また被災地でもこのチラシを見かけることはほとんどありませんでした。ソーシャルメディアなどを通じての情報共有と実際の支援を結びつけることが次の課題と感じています。

注1　HPはホームページの略です。本来ならば、Webサイトとすべきところですが、ITに詳しくない人たちの間では、HP＝ホームページのほうが理解されやすいため、対外的な呼称としてはHPを用いることにしています。

注2　そのためには、事前に自治体と約束を取り交わします。

◆図1　自治体ホームページレスキュープロジェクトのWebサイト(http://klgmonitor.itdart.org/)

◆図2 「詐欺に注意」チラシ(https://drive.google.com/file/d/0B_0uoQjpPS6aUmFFZ1lGc19nZ2c/view)



情報システムの開発

情報システムの開発は、災害対応を行う組織や個人が情報の収集や利用に活用するためのツールを提供する活動です。具体的には次のようなプロジェクトが行われました。

●ExcelGeo

災害時にはさまざまな情報を地理空間情報として整理することがよくあります。避難場所や避難所、炊き出しの場所などを地図上に配置するのがわかりやすい例でしょう。その際に意外に面倒なのが、場所の緯度・経度を指定することです。

住所から緯度・経度を調べることをジオコーディングと呼びます。このジオコーディングはMaps系のAPIにはたいがい備わっています。GoogleにもGeocoding APIとして用意されているのですが、残念ながらGoogle Mapsと組み合わせて利用する場合のみ使用が認められています。つまり、住所から緯度・経度を調べたいという目的のためだけには使えないのです。他社のジオコーディングAPIも利用規約が不明だったり、品質が期待値に達していなかったりで使えず、規約的にも問題なく高品質なジオコーディングは、地理空間情報に携わる人にとっては以前より強く要望されているものでした。

また、大量の住所の緯度・経度を一括処理で取得することもそうたやすいことではありません。ITに明るくない人たちにとっても使いやすいインターフェースを持つサービスは待ち望まれていました。

今回、IT DARTでは、ExcelGeoという形でそれを実現しました(**図3**)。これはExcelファイル(xlsxおよびxls)をドラッグ&ドロップするだけで、ファイルに書かれている地名や住所の緯度経度を一括で取得します。結果は、元のExcelファイルにマージするか、JSON形式で返されます。

使っているのは、GeoNLPという大学共同利用機関法人情報・システム機構と国立情報学研究所が共同で開発したジオコーディングサービスです。これには、Google Geocoding APIのような制約もありません。

このプロジェクトもさくらインターネット㈱からサーバを支援いただきました。

●マークシートを用いた災害ボランティア登録システムの開発

災害ボランティアとして活動する人はボランティ

◆図3 ExcelGeo(http://excelgeo.itdart.org/)

ア活動保険に入っている必要があるのですが、その加入の有無やセンターでの加入の要否などを受付時に確認する必要があります。また、後日の集計や検索も必要となりますが、多い日には数百人を超えるボランティアが訪れることがあるため、2015年の関東・東北豪雨災害の際には災害ボランティアセンターに大きな入力の負荷がかかりました。

この業務をマークシートを用いて軽減するシステムを開発しました。当初は電話番号や生年月日などはすべて手書きで入力してもらい、それを手書き文字認識OCRで処理する予定だったのですが、オープンソースのOCRエンジン[注3]ではどうしても精度が低く、その部分も数字をマークしてもらう形式としました。

●kintoneを使用した物資管理帳システム

支援団体からの依頼を受け、支援物資管理システムの構築および運用を行いました。このシステムは、現地チームで物資管理票を撮影した画像をアップロードし、後方支援チームが画像の情報をデータベースに入力することで物資管理を行います。現地での作業を“撮影”という簡単な作業に限定することで、多様なメンバーが関与する現場でもITシステムを活用できる体制としました。

このシステムは、サイボウズ㈱のkintoneを用いることで、迅速な開発とイテレーションを数度繰り返しながら完成度を高めることが可能となりました。サイボウズからはこのシステムだけではなく、り災証明書発行システムの試作においても、kintoneを無償提供いただきました。開発はkintone Café[注4]の方々にご協力いただいています。

通信環境整備の支援

災害対応を行う組織に対して、必要なIT環境の提供および設定支援も行いました。

●PCやWi-Fiルータなどの貸与・支援

13の支援団体に、レノボ・ジャパン㈱（NECパーソナルコンピュータ㈱からの紹介）とITで日本を元気に！[注5]から提供されたPCやWi-Fiルータ（KDDI㈱からの提供）、タブレット（㈱NTTドコモからの提供）の環境を整え、各団体に配布しました。

提供されたPCが中古PCの場合は、OSやOfficeを再インストールする必要があります。IT DARTでは、効率よくセットアップするため、災害復旧・復興支援用中古PC設定手順書を作成しました。この手順書作成および講習会、中古PCにインストールするOfficeのライセンスについては、日本マイクロソフト㈱から支援いただいています。また、PCのセットアップは福岡県にあるシステムラボラトリー㈱[注6]にも協力いただきました。

●現地での環境構築サポート

地元NPOと県外支援組織との連携調整会議である「熊本地震支援団体火の国会議」事務局をはじめとするいくつかの支援団体の拠点で、IT環境の構築をサポートしました。モバイルルータを使ってのネットワーク環境の構築やプリンタの共有設定など、限られたリソースを活用してのIT環境構築は、なかなか骨の折れる作業です。モバイルルータに接続できるPCの台数に制限があったり、型式の古いプリンタをネットワーク内で共有するための設定が複雑だったり、さまざまな制約がある中での作業となりました。

今後に向けて

熊本や大分の被災地の復旧や復興はまだこれからが本番です。IT DARTとしては、継続して熊本や大分の支援をしつつ将来発生する次の災害に備え、今回の経験をもとに、さらにITで情報支援を行うべく体制を整えていきたいと考えています。SD

注3　ImageMagicで画像を切り出し、tesseractというGoogleが公開しているオープンソースOCRエンジンを使ったのですが、どうしても認識率を期待値まであげることができませんでした。
注4　http://kintonecafe.com/
注5　http://revival-tohoku.jp/
注6　http://www.syslabo.co.jp/

温故知新
IT むかしばなし

第57回

電子ブロックと4bit CPU

速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami

はじめに

ハンダ付けを不要とする電子工作の方法として、アポロ宇宙船のコンピュータ回路に用いられたワイヤラッピングやブレッドボードなどがあります。最近では、ブレッドボードの電子回路にArduinoなどのマイコンボードを接続してさまざまな電子回路実験も行われています。1970年代、ハンダ付けを行わない簡易な電子回路組み立て方法として電子ブロックが広く使われており、80年代になると電子ブロックとマイコンを組み合わせた製品もでてきました。今回は、電子ブロックと、そこに使われた4bitマイコンのお話をしましょう。

電子ブロックとは

電子ブロックは、縦横17mm高さ23mmの半透明なプラスチック製のブロック内に電子部品がハンダ付けされて入っており、四角の4辺の一辺ずつに端子があり、隣同士のブロックの端子が接触することで電気的な接続が行われるしくみになっています(**写真1**)。電子ブロックの原型となる製品は1960年代からありましたが、70年代になり容易にブロックの抜き挿しのみで配線でき、当時広く購読されていた「子供の科学」の発行元の学習研究社(現・学研ホールディングス。以下、学研)から発売されてブームになりました。

ブロック内の電子部品は、抵抗、コンデンサー、コイル、ダイオード、トランジスタを基本として、配線用のブロックやスイッチが用意されていました。これらを組み合わせることになるのですが**写真1**のように容量や抵抗値の違いだけでなく異なる配線のブロックもあるため、置き方に工夫が必要でした。したがって、付属している詳細なマニュアルに説明されている電子回路をそのとおりになぞって工作することが主な使い方だったと思います。

復刻版が発売された当時の最上位機種のEX-150では、横8×縦6=48のブロックが配置できる筐体になっており、ブロックを取り囲む周辺には端子が複数あります。端子には、電池4本から供給される6Vとグラウンド、アンプに接続してスピーカーから音を出すサウンド入力、光センサー端子、ラジオのアンテナ回路との接続端子があり、ブロックの4つの端子とうまく接続すれば多用途な電子回路が実現できたのです。

筐体の本体には、この端子から接続された電池ボックス、光センサー、ボリュームアンプ付きのスピーカー、アンテナ回路が備わっていました。

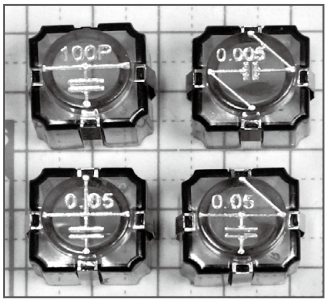
電子ブロックの限界

容易に電子回路が組める大きなメリットを有した電子ブロックですが、問題となる限界がありました。

電子ブロックの部品種類が一定のものしかない

それぞれのシリーズで用意されている部品しか使えないため、マニュアルに載っていない回路を作成しようとする際、部品の使い回しの工夫が必要で、独自の

▼写真1



回路を簡単には実装できません。

電子部品の種類だけでなく、ブロック内部の配線に種類の違いがあるため適切な電子ブロックを使い、限られた箱庭のような筐体にうまくはめ込むためには独特のテクニックが必要とされました。

電子ブロック間の接続が不安定

ブロックの4つの端子は板バネ状になっていますが、接触不良が多く発生します。最近のハンダ付け不要な教育用電子機器であるLittleBits[注1]は、見た目の接続と実際の接続が一致して確実な接続ができるのに比べて、大きく信頼性の差があります。

ちょうど、1960年代のデンマーク製のレゴと日本製のダイヤブロックの工作精度の差と同じような感があります。

アナログ回路の動作が安定しない

電子ブロックで作成できる回路の多くがアナログ回路であり、接触の安定性と相まって、確実に動作する回路を作成できないこともありました。

4bitCPUユニット

NEC PC-8801や富士通FM-8などのマイコンが登場した1981年、電子ブロックにもマイコンユニットが付属したFX-マイコンR-165(以下FX-R165)が登場します。165の数字が示すとおり65種類の電気回路と100種類のマイコン実験ができます。

マイコンユニット(**写真2**)は4bit CPUを搭載し16進キーボード(クリック音つき)、7個の赤色LEDそして16進数を表示する7セグメントLEDが付いており、ユニットだけでプログラミングや動作が行われるTK-80などと同じような(表示は少なくプログラムの保存もできない)マイコンボードとなっています。このユニットと電子ブロックの組み合わせは、1＋1が2以上になる絶好のものだと思われたのですが……。

写真2のようにマイコンユニットは筐体の縦6×横11のうち縦5×横8を占有します。接続端子は9V電源とグランドおよびスピーカー出力を接続することになります。

ほかにも出力1bit、入力2bitがありますが光センサーのデータを受け取るアナログ入力はありません。マニュアルのプログラム集にはマイコンユニットのみを使用するプログラム例が100種類あり、電子ブロックとの接続例はおまけの2例しかありません。内容的には電子ブロックとのつながりは、電源供給とスピーカー出力だけだったのです。

マイコンユニットのCPUは、テキサス・インスツルメンツ社のTMS-1100が元になっています。この4bitのCPUは電卓用に特化しており、10キーボード、7セグメントLEDの表示機能は優れていますが、汎用的に使うには相応しくないものでした。それを、マイコンボードとして使えるように、独自の4bitCPUをエミュレートするコードを作成して、CPU内ROMに書き込んでカスタム化したようです。

マイコン時代に期待を込めて発表されたFX-R165ですが、その後5年間新しい機種は出ず1986年に学研の電子ブロック全シリーズは生産中止になります。

2009年に学研から「大人の科学マガジン」Vol.24 4bitマイコンが発売されました。このマイコンGMC-4は、8bit CPUでエミュレートしたFX-R165のマイコンユニットとまったく同じ命令コードだったのです。

終わりに

ハンダ付けなしの電子回路とマイコンの組み合わせのアイデアは、Arduinoとブレッドボード、LittleBits、mCookie[注2]のように現在では教育現場で広まっています。1980年代初めにこのようなアイデアを製品化したことは過去のすばらしい実績だった思います。しかし、そのあとの進展は復刻版に止まるレベルで満足できるものではありません。現在の高い技術で作られたデジタル電子ブロックの出現を期待したいと思います[注3]。SD

注1) 磁石でさまざまな電子回路をつないで電子工作を行うキット。http://jp.littlebits.com/

注2) レゴのように重ねて電子回路を作成するキット。https://www.microduino.cc/store

注3) 電子ブロック機器製造株式会社では、新しい電子ブロックの開発、販売も行われているようです。http://www.denshiblock.co.jp/

▼写真2

うまくいく チーム開発のツール戦略

第3回 備えあれば憂いなし! JIRAとBitbucket Serverに記録を残そう

Author リックソフト㈱ 廣田 隆之(ひろた たかゆき)

前回はGitのメリットを紹介しつつ、アトラシアン社の開発ツールであるBitbucket ServerやSourceTreeの使い方を説明しました。中でも、ブランチを使った開発手法や、プルリクエストによるコードレビューやマージはチーム開発を支える強力な機能です。まだ記事をご覧になっていない読者の皆さんは、ぜひバックナンバーをご一読ください。

今回は、ソフトウェア開発で避けることのできない不具合対応やトラブル対応において、JIRAやBitbucket Serverをどのように活用していくかについて説明します。

トラブル発生!

不具合の報告は、いつも突然やってきます。メール、電話、最近ではチャットやSNSかもしれません。障害の報告というのはけっして気持ちの良いものではありませんが、ソフトウェアの品質を上げる、そして何より開発チームの手腕が問われるチャンスでもあります。筆者自身、長年にわたってソフトウェア開発に関わっていると、何度もピンチに見舞われることがありましたが、そのときの対応次第でソフトウェアの良い改善機会になったこともありますし、結果的にお客様から対応内容についてお褒めの言葉をいただき、バグを作りこんでしまったにも関わらずたいへん恐縮したこともあります。

不具合の連絡を受けたときは、できるだけ速やかにチームの課題管理システムに登録しましょう。どんな形であれ、記録を残すのは重要です。幸い、JIRAは組織やプロジェクトに合わせて柔軟にカスタマイズでき、Webブラウザから手軽に課題を入力できます。また、JIRA Service Deskというアプリケーションを導入すれば、より簡単にバグチケットを起票できるインターフェースが提供されます(**図1**)。この連載の第1回ではメールによるチケット起票についても書いているので、興味のある方はそちらをご覧ください。JIRAは元来バグトラッキングシステム(BTS)として生まれたということもあり、ソフトウェアのバグを記録するためのシステムと思われがちですが、けっしてそんなことはありません。ちょっとした改善項目や問い合わせなど、何でも記録していくことがうまく使いこなす第一歩だと思います。

さて、トラブル対応時はトラブルシューティングに必要な情報を漏れなく速やかに集めることが重要です。JIRAはそのための窓口でもあり、データベースとして機能します。トラブルシューティングに必要な情報は、扱っているソフトウェアやシステム、また組織によってさまざまです。たとえば、モバイル関連のサービスであれば、端末のオペレーティングシステムの情報や解像度などの環境情報が重要になることもあるでしょう。

JIRAには標準で課題の要約、優先度、担当者、期限などのフィールドが用意されていますが、カスタムフィールドを自由に追加することもできます。このJIRAの柔軟性を活かして、トラブルシューティングに必要な情報を効果的に収集できるように、業務に応じてフィールドを自由にカスタマイズしてください。

▼図1 JIRA Service Deskのインターフェース

情報を入力するときは、その内容も大切です。「○○が動きません」といった情報だけでは調査は困難なものになります。「再現手順」「期待する結果」「実際の結果」「再現率」といった情報のほか、スクリーンショットやログがあれば問題解決に役立つでしょう。組織で簡単なルールを作ったり、チケットの起票の負担を軽減するために必須フィールドを減らしたりといった工夫も必要かもしれません。

調査開始

調査に必要な情報がそろったら、問題解決に向かっていよいよ作業開始です。報告を受けた時点ですぐに原因がわかれば良いのですが、原因が不明な場合は、問題解決への第一歩は再現テストです。現象を再現できたら、原因を追及していきます。環境の問題かもしれませんし、単にソフトウェアの使い方の問題かもしれません。プログラムの問題が疑われるならば、ソースコードを解析していくことになります。そのとき助けになるのが、バージョン管理システムに蓄積してきたソースコードであり、コミットログの歴史です。目的のソースコードをすぐに見つけられるように、日ごろからBitbucket Serverのように信頼のおけるツールを活用し、ソースコードの管理はしっかりやっておきたいところです。

報告のあった不具合内容からソースコードの場所をピンポイントで特定できると良いのですが、原因がすぐには判明しないこともしばしばあります。たとえば、利用しているフレームワークやライブラリに原因があるときなどです。ソフトウェアの、あるバージョンを境に不具合が起きるようになった場合は、バージョン間の差分を調べる必要があります。Bitbucket Serverにはブランチ間の差分をグラフィカルに表示する機能が備わっており、Webブラウザ上でソースコードのdiffを見ることができます(**図2**)。また、コミットログを追うことで、「いつ」「誰が」変更を行ったかもひと目でわかります。Bitbucket Server 4.6ではBlame表示という機能が追加され、ソースコードの1行1行について変更を行った開発者を表示できるようになりました。チーム開発では1つのファイルを複数の開発者が修正することも多いので、このような機能は非常に役立つでしょう。

SourceTreeやBitbucket Serverを日常的に使っていれば、「いつ」「誰が」という情報は自動的に記録されていきます。しかし、ソースコードの変更履歴で最も重要な情報は「なぜ？」、つまり変更した理由です。1人の開発者だけで開発している場合でも、これは重要です。1年前に自分自身がコードを修正した理由をすぐに言える人は少ないのではないかと思います。優れたツールを使っていても、せっかくたどりついたコミットログに「変数名を変更した」とか「APIをfoo( )からbar( )へ変更」といったコメントしか残されていなかったらどうでしょうか。変更内

▼図2　ブランチ間の差分の比較

容はソースコードを見ればわかります。大切なのは「変更した理由」です。たとえば、「グローバルスコープと名前が衝突する可能性があるため、変数名を変更した」となっていれば、衝突の可能性を回避するための改善なのだな、ということがわかります。同様に、「パフォーマンス向上のため、APIをfoo( )からbar( )へ変更」とあれば、性能向上のために利用するAPIを変えたことがわかります。このように、コミットログはソースコードのコメントと同様に、開発者にとって貴重な情報源であることを覚えておきましょう。

とはいえ、コードのコメントやコミットログだけではすべてを書き残すことは難しいです。JIRAを使っているのであれば、コミットログにJIRAの課題キーを記載することを忘れずにしたいものです。こうしておけばBitbucket Serverのコミットと JIRAのチケットが紐付(ひも)けられ、より詳しい変更理由を知ることができるようになります。追跡の際、開発者はソースコードのコメント→コミットログ→チケットの順に見ていくはずです。ソースコードをメンテナンスするときも、この順番で記録していくのが良いと思います。

より良いコミットログを残すための工夫

良いコミットログを書くには、日ごろからの訓練が大切です。BitbucketやGitHubでホストされているオープンソースソフトウェアや、うまくいっているチームのコミットログを見て真似をしてみましょう。長文のコミットログを目にすることもあるはずです。日ごろのプルリクエストのコードレビューでも、コミットログにも注意を払うように心がけてください。さらに、適切なJIRAの課題キーが書かれているかどうかをチェックするしくみの導入も検討しましょう。Gitではコミットフックをカスタマイズし、コミットログが指定した書式のとおりに書かれているかどうかをチェックしてコミットを拒否することもできます。興味のある方は**https://www.atlassian.com/git/tutorials/git-hooks/**を参考にしてみてください。

集中型リポジトリであるSubversionの時代であれば、コミットのフックはサーバ1個所で実行すればよく、導入も比較的容易だったのです

が、分散型リポジトリのGitでは開発者自らがローカルリポジトリを持つため、コミット時のチェックを導入するのはハードルが高いというジレンマもあります。また、自分でスクリプトを書くのがたいへんという方もいらっしゃるでしょう。こういった場合は、Bitbucket Serverにアドオンを導入して、プッシュやマージでチェックを実行するというしくみを検討してはいかがでしょうか。Atlassian Marketplaceで公開されている「Jira Hooks for Bitbucket」[注1]というアドオンを使えば、プッシュやマージのタイミングでのJIRAのチケットのチェックが可能です。ステータスがルールどおりでないときは、プッシュやマージを拒否することもできます。ただし、このアドオンは無償で提供されているため、ベンダからのサポートは受けられないことにご注意ください。

コード修正とリリース

ソースコードの修正が必要になった場合は、前回の記事で紹介したBitbucket ServerとSourceTreeの使い方を参考にしながらブランチを作成し、コードをコミット、プッシュして、プルリクエストでコードレビューを実施し、マージします。

不具合修正の場合は、bugfixブランチを作成して作業することになるでしょう。Bitbucket Serverはリポジトリごとにブランチモデルをカスタマイズでき、さまざまなブランチモデルに柔軟に対応します。git-flowブランチモデルなど、チームやプロジェクトに最適なブランチモデルを適用してください。ちなみに筆者らは自社製品のJIRAアドオンを開発してAtlassian Marketplaceに公開していますが、masterを開発用ブランチとして使うシンプルなブランチモデルを採用しています。

コードの修正後は、ビルドとテスト、リリースを行います。アトラシアン社のCIサーバであるBamboo[注2]（**図3**）を使えば、ビルドの自動化から配備の自動化まで、ソフトウェア開発に関わるさまざまなタスクを自動化できます。BambooはBitbucket Serverでブランチが作られたことを自動的に検知してビルドを走らせるため、プルリクエストでマージされる前の成果物をレビュー承認前にテストしたり検証環境へ配備して動作を確認したりといったことができます。開発のインフラをアトラシアン社のツールチェインでそろえることにより、課題管理、バージョン管理、ビルド、配備といったシステム連携がスムーズになります。

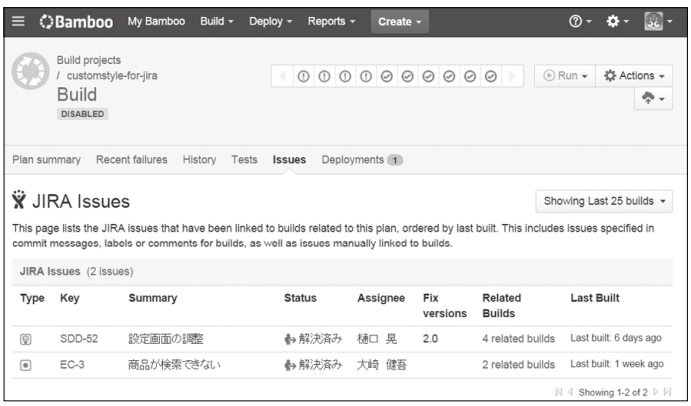
終わりに

サービスやプロダクトが成長していく中で、ソースコードの変更は避けられません。次回は、品質を保ちながらコードを改善していくコツや技術的負債の賢い返済方法について説明する予定です。SD

注1）https://marketplace.atlassian.com/plugins/com.lb.software.stash.jira.connector.lb-software-stash-jira-connector/server/overview

注2）https://www.ricksoft.jp/atlassian/bamboo/

▼図3　Bamboo



Software

Tableau Japan、データ分析ツールの新バージョン「Tableau 10」を発表

Tableau Japan㈱は6月16日、データ分析ツール「Tableau」の最新バージョン「10」を発表した。

Tableauは、ドラッグ&ドロップでデータの集計、グラフ化が簡単に行えるBIツール。製品のラインナップとしては、デスクトップアプリの「Tableau Desktop」、Tableau Desktopで作成したダッシュボードをWebアプリ化する「Tableau Server」、クラウドでTableau Serverを利用する「Tableau Online」などがある。

新バージョン「10」のおもな新機能は次のとおり。

- Googleスプレッドシート、Quickbooks Online、Kognito、memSQL、OData、Webデータ用WDC2.0に接続可能に
- 異なるデータソースのデータを結合、統合可能に
- ドラッグ&ドロップでのクラスタ分析が可能に

▲Tableauのダッシュボード

CONTACT
Tableau Japan ㈱ URL **http://www.tableau.com/ja-jp**

Service

アイレット、端末動作テストサービス「devicepack」を提供開始

アイレット㈱は6月23日、各種スマートデバイスやOSのバージョンごとにおけるアプリの動作テストの自動化を支援するサービス「devicepack」を8月1日から提供開始することを発表した。

devicepackでは、iOS/Android/Kindleを含む285種類もの膨大なスマートデバイスへの対応テストを一度に行えるサービス「AWS Device Farm」を利用し、同社のcloudpackチームがテストシナリオの作成や環境構築およびテスト自動化の支援を行う。プラットフォームの利用料は月額50,000円からで、テストの実施回数などにより従量課金が発生する。

そのほかの特徴は次のとおり。

- 多様な端末の言語設定に対応可能
- テスト結果を確認できる独自のダッシュボードをブラウザ上で確認可能
- 実行中のテスト状況を、スクリーンキャプチャで確認可能
- テスト実行中のCPU使用率、描画性能（FPS）、メモリ利用料、スレッド数などパフォーマンス計測の結果を閲覧可能

CONTACT
アイレット㈱ URL **http://www.iret.co.jp**

Service

ヒューレットパッカードエンタープライズ、「HPE Universal IoT Platform」を発表

ヒューレットパッカードエンタープライズは5月10日、「HPE Universal IoT Platform」の提供開始を発表した。

HPE Universal IoT Platformは、デバイスとアプリの相互運用と管理を効率化するプラットフォームで、自社のオンプレミスもしくはプライベートクラウド環境へ導入し、ユーザはas-a-Serviceモデルとして利用する。

本プラットフォームは業界標準のoneM2Mに準拠しており、個別の業界やベンダには依存しない設計となっている。これによってIoT事業者は、異種混在のセンサーの組み合わせを管理し、マシンツーマシン（M2M）デバイス上で業種別アプリケーションを運用すると同時に、単一のセキュアなクラウドプラットフォーム内で収集されたデータを処理、分析、収益化できる。また本プラットフォームは、LoRaおよびSIGFOXのほか、モバイルネットワーク、そのほかの無線、Wi-Fi、Bluetoothといった接続プロトコルもサポート可能とのこと。センシングデータに対しては「HPE Vertica」「HPE Haven OnDemand」を活用して意味のあるパターンを発見、さらに外部からのデータを組み合わせることで「コンテキストデータ」へと価値を高めることができる。

CONTACT
ヒューレットパッカードエンタープライズ
URL **https://www.hpe.com/jp/ja**

Report

技術書オンリーイベント「技術書典」、イベントレポート＆主催者インタビュー

6月25日、秋葉原通運会館（東京都千代田区）にて技術書限定の同人誌即売会「技術書典」が開催された。主催は、同人誌サークル「TechBooster」と技術系電子書籍の制作・販売を行う「達人出版会」。当日の一般参加者数は1,300名。関係者やサークル入場の100人を含めると総計1,400人が来場した。参加サークル数は個人が48、企業が9、委託販売が2。

▲会場入り口の立て看板

イベントレポート

同人誌のテーマとしてはスマホアプリやWeb開発の同人誌が多く、言語系ではCrystalやElixirなど、比較的新しいものが取り上げられていた。変わり種では、量子コンピュータを扱う同人誌も頒布されていた。そのほか、R:VIEWやIndesign、XMLやLaTexで組版を行うといったDTP系の同人誌も目立った。

また会場では、リクルートテクノロジーズの伊藤敬彦氏による「RedPen」についてのセッション、技術書や出版社のグッズが当たる抽選会が行われた。


▲DevLOVE Pub
技術雑誌「Far East Developer Review」の総集編を頒布


▲㈱翔泳社
CodeZineの連載記事を書籍化したものを中心に、自社の技術書を販売

主催者インタビュー

技術書典の主催の1人、達人出版会の高橋征義氏にお話を伺った。

――技術書に限った同人誌即売会という珍しいイベントでしたが、開催の動機やきっかけはどのようなものだったのでしょうか？

これは2015年9月ごろ、TechBoosterの日高さんから「こういうイベントをやりたい」という話をもらって、それから協力することになりました。個人的には小説系の同人誌を作ったりコミケで売り子をしたりしたこともあったのですが、同人誌は商業書籍やネット上のコンテンツとは違う文化があるので、それをうまく伝えられたら……という気持ちがありました。

――会場の確保や参加サークルの調整といった事前の準備で、苦労されたことや工夫されたことなどあればお聞かせください

私のほうはスポンサー対応や、ほかの人からこぼれているところを拾ったりしていた程度で、おもなところは日高さんはじめTechBoosterのみなさんが尽力されていたので、私自身が苦労したところは正直あまりありません。イベント自体の一番の課題はやはり会場の確保で、会議室としては使えても即売会には使わせていただけないところもあって、サークル参加者や一般参加者の方には手狭になってしまったのは申しわけなかったです。

――当日の会場の雰囲気、来場者数など、振り返ってみてのイベントの手応えはいかがでしょうか？

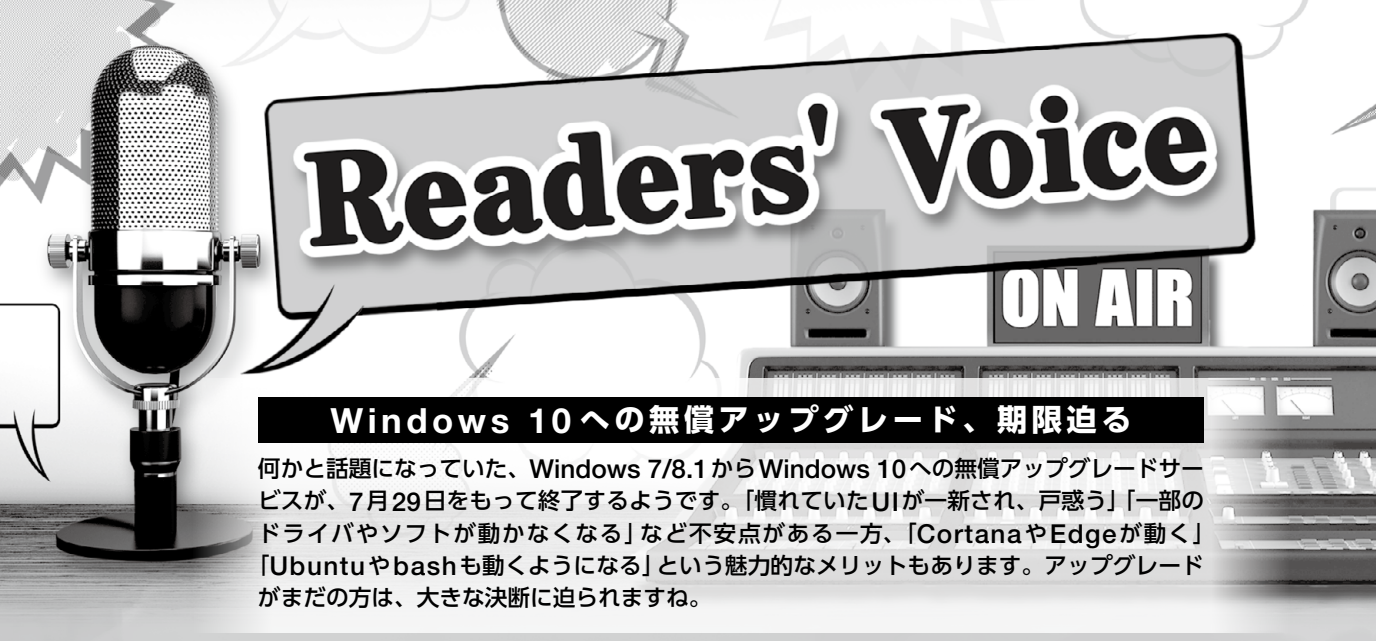
参加者としては1,000人という目標を掲げていたものの、実際にどれだけ来るかまったく予想できていなかったのですが、最終的にはサークル参加者含めて1,400名ほどの方に来場いただけて、あらためて技術書の力を感じました。来てくださった方々、また応援していただいたみなさんには感謝しています。一方で、一時期入場がスムーズにいかなかったときがあったようで、ご迷惑をおかけした参加者の方にはこの場を借りてお詫びいたします。

――本イベント、来年以降の開催の予定はございますでしょうか？　もしあるようでしたら、新たに挑戦したいこと、改善したいことなどあればお聞かせください

今後の具体的な予定はまだ何もありませんが、今回のイベントで技術書を手に取られた方が、今度は書くほうにチャレンジする機会を提供できると良いなと思っています。


▲抽選会で参加者とジャンケンを行う達人出版会の高橋征義氏

CONTACT
技術書典　URL **https://techbookfest.org**



Windows 10への無償アップグレード、期限迫る

何かと話題になっていた、Windows 7/8.1からWindows 10への無償アップグレードサービスが、7月29日をもって終了するようです。「慣れていたUIが一新され、戸惑う」「一部のドライバやソフトが動かなくなる」など不安点がある一方、「CortanaやEdgeが動く」「Ubuntuやbashも動くようになる」という魅力的なメリットもあります。アップグレードがまだの方は、大きな決断に迫られますね。

2016年6月号について、たくさんの声が届きました。

第1特集 bash再入門

bashについて、コマンドの使い方、シェルスクリプトの組み方をとことん解説しました。さらに、仕事でシェルスクリプトを使うときの注意事項、Bash on Windowsのしくみ、新しいシェル「fish」の紹介まで、盛りだくさんの特集でした。

(ページデのデザインは)バスケットシューズのダジャレかな？
ahsgrimm192さん／大阪府

シェル芸を勉強したくなりました。
tack41さん／愛知県

第2章にbashの知らなかった機能がたくさん載っており、勉強になりました。
橿山さん／埼玉県

組込みLinux機器でbashを使うことになるので、非常に勉強になります。
エゾモモンガさん／滋賀県

bashを使っていましたが中身は全然わかってなかったので、今回の特集はとても参考になりました。
WATさん／石川県

ちょうどWindowsでもbashが動くということで、良い特集だと思いました。
n0tsさん／東京都

cat [原稿ファイル] | grep -o 'シェル芸' | grep -c '' # を実行してみたいです。
ginさん／愛知県

bashはエンジニアにとって必須の知識なので、とてもいい特集だった。
りょうじさん／東京都

fishの紹介が良かったです。
近藤さん／静岡県

bashは多くのディストリビューションで採用される標準シェルということで、復習・再発見に活用された読者の方が多いようです。「Bash on Windows」の発表もあったということで、あらためて勉強しなおしても良いのではないでしょうか。

第2特集　MySQLを武器にSQLをはじめよう

オープンソースのRDBMSであるMySQLを使って、SQLの基礎を学ぶ特集。MySQLの歴史、Mac、Windows、Linuxの環境ごとのインストール方法を押さえたあとは、実際にデータベース操作を実践してみました。

文字コードのコラムがおもしろかった。日本語こわい。
藤田さん／東京都

以前から興味があったので、この機会に手を出してみようかと思った。
ナタリーさん／福岡県

社内でMySQLが稼働しているので、新人などに読ませた。
松崎さん／千葉県

SQLを理解することの重要性を認識し、勉強しようという気持ちになった。
小山さん／福岡県

MySQLを使用したちょっとした社内システムを開発する予定なので、本特集が役に立ちそうです。
山添さん／東京都

6月号ということで初心者向けの記事かと思いましたが、読んでみると意外とわかっていない自分がいました。
齋藤さん／神奈川県

開発においてDBMSとSQLは避けて通れないものですが、勉強する範囲も広く、また奥も深いので、敷居が高いですよね。今回はオープンソースのMySQLを使って「とりあえず動かし

6月号のプレゼント当選者は、次の皆さまです

①無線LANルータ「WN-AX1167GR」
井上敬利様（福岡県）

②Nimbelストレージモバイルバッテリー
宮田哲雄様（東京都）

③ESETファミリーセキュリティ1年版
キャボさん様（埼玉県）、伊集院伸幸様（東京都）、中村昭博様（石川県）

④『ブラウザハック』
jo7oem様（山形県）、lin様（新潟県）

⑤『Pythonチュートリアル第3版』
中村財蔵様（神奈川県）、白井太郎様（東京都）

⑥『基礎Visual Basic 2015』
眞鍋裕二様（香川県）、田中良明様（滋賀県）

⑦『ドキュメント作成システム構築ガイド』
斉藤和芳様（北海道）、水谷典子様（愛知県）

※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。

てみる」がテーマの特集でしたので、次の学習の入り口になれば幸いです。

一般記事　Android Wearアプリ開発入門［特別編］

実機が続々と登場し、またベースとなるAndroidがバージョンアップしたことで「Android Wear」が盛り上がりを見せています。今回は、Wearに新しく追加された「バーミッション」「サウンド出力」「マルチスクリーン対応」の3つの機能について、アプリの開発者が注意しておくべきことを解説しました。

異なる見え方になる点を再確認できてよかった。　**とーふやさん／神奈川県**

デバイスを持っていないが、興味深い。
人儚 由夢さん／福岡県

スマートウォッチアプリの開発は楽しそうですが、肝心のスマートウォッチを持っておりません……。ぜひとも読者プレゼントで……。
NGC2068さん／愛知県

現在、Android端末用にアプリ開発を行っているので非常に興味を持ちました。私の身の周りではAndroid Wearを所有している方はいないのですが、一度手にして、アプリ開発を行ってみたくなりました。
ぶううううんさん／山形県

もうただのアンドロイドアプリの時代は終わったのかもしれませんねぇ。
lipgtxさん／東京都

ぜひ実機を手に入れて実践してみたい、とは思います。実機さえあれば……。
村橋さん／北海道

注目度は非常に高いようですが、実機を持っていないという方がまだまだ多いようです。デバイスが超進化したり、キラーアプリが登場したりと、何か普及するきっかけが出てきてほしいですね。

一般記事　フリーで始めるサーバのセキュリティチェック【後編】

セキュリティベンダなどに頼らず、手元で行おうとすると敷居が高い「脆弱性診断」。これをフリーのソフトを使って比較的手軽に行おう、というのが本短期連載。後編では「OpenVAS」を使ってGUIとCLI両方から脆弱性スキャンを行う方法を解説しました。

フリーでできるんだって印象。
tekitoizmさん／東京都

診断ツールの使用では誤検知よりも検知漏れが怖いので、複数の診断ツールを組み合わせて使うことが大事だと思いました。　**永作さん／東京都**

まだまだセキュリティに関しては勉強が足らないので、良い勉強になります。
ArrayMonsterさん／東京都

セキュリティについては、積極的に理解しておかないと、忘れてしまったり、何か事故を起こしたりするので、しっかりこういう記事を読んでおきたい。
ももんがさん／静岡県

セキュリティ対策といっても、どんなツールを使ってどこから対策していけばいいのか迷いますよね。今回は「脆弱性診断」ということで、ふだん使っているサーバに弱点がないかをまず調べることで、具体的な対策を考える前準備が整えられました。

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
September 2016

2016年9月号
定価（本体1,220円+税）
192ページ

8月18日発売

［第1特集］ログ出力のベストプラクティス
役立つログを残していますか?
システムログからアプリのログまで徹底検証

［第2特集］
「良いPHP、悪いPHP」

※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

「Mackerelではじめるサーバ管理」（第18回）は都合によりお休みさせていただきます。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2016年7月号
●P.46　第1特集 第3章
Software Design 2016年7月号 第1特集 第3章の記事およびサンプルプログラムについて、輻輳制御をしないままUDPパケットをインターネットに流すことに問題があるのではないか、というご指摘を頂戴しました。
本稿ではTCPとの違いをあえて明確にすることや、スクリプト言語の特性を活かしたプログラムをできるだけシンプルにして、構造の理解しやすさや機敏な開発のプロセスを伝えるのが目的でした。
しかしご指摘のようにUDPの例は、インターネットに対して実施するには配慮にかけています。とくに初心者であっても、配布したソースコードをもとに、IPレベルでのDOS攻撃とかわらないことができ、インターネットを混乱させてしまうことになるかもしれません。
実際にインターネットでUDPの通信をする場合には、インターネット上の経路を飽和させてしまうことがないようにしなければならないむね、記事中でもあくまで誌上での実験であることと、実際に通信する場合に配慮・注意をするように記述するべきだったと思います。
ご指摘くださった方に感謝を申し上げるとともに、ご指摘を重く受け止め、この場を借りて読者の皆様へのお詫びと、サンプルプログラムの実施おけるご注意をお願い申し上げます。

【サンプルプログラムを利用する際のご注意】
インターネットに対してUDPで全力の送受信をすると、各経路において帯域を独占してしまうことがあります。サンプルプログラムには、輻輳制御など、帯域を飽和させないためのしくみが組み込まれていません。同プログラムをインターネットで用いることは、帯域の独占をしてしまうことがあります。動作の確認は、できるだけ閉じたLAN環境内でのみおこなうようご注意願います。

●P.40　第1特集「第2章　実践ネットワークプログラミング～C言語編～」リスト3 右段上から11行目
［誤］inet_ntop(serv.sin_family, &serv.sin_serv_name, sizeof(serv_name));
［正］inet_ntop(serv.sin_family, &serv.sin_addr, serv_name, sizeof(serv_name));
●P.58　第1特集「第3章　実践ネットワークプログラミング～スクリプト言語編～」左段上から8行目
［誤］udp_receive_server.rbでは、selectを使ってマルチプレクス処理をおこなっています。
［正］udp_receive_server.rbでは、マルチスレッドを使った多重化（マルチプレクス）処理をおこなっています。
●P.103　「使って考える仮想化技術」注1）のURL（旧バージョンのリンク変更に伴う修正）
［誤］例）http://ftp.riken.jp/Linux/centos/6.7/isos/x86_64/
［正］http://archive.kernel.org/centos-vault/6.7/isos/x86_64/

■2016年1月号
●P.79　第2特集「第2章　InventoryとPlaybook、2つのファイルを理解する」リスト31

［誤］
```
tasks:
 - name: Ubuntuの場合 main_ubuntu.yml を読み込む
   include: main_ubuntu.yml
 - name: CentOSの場合、main_centos.yml を読み込む
   include: main_centos.yml
```

［正］
```
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"
```

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2016年8月号

発行日
2016年8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。