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
2017
2
2017年2月18日発行
毎月1回18日発行
通巻382号
（発刊316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

》Dockerって便利？ 》Linuxファイルシステム 》採用されやすい技術者

なぜプログラマの役に立つのか

# いまはじめる Docker

コンテナ技術を身につける

Special Feature 2

ファイルがわかれば Linuxがわかる

## Linux ファイルシステム 徹底入門

Ext3/4,XFS,F2FS,Btrfs

Special Feature 3

## エンジニアが 採用できない会社と 評価されない エンジニア（後編）

特別企画

## Black Hatの トレーニングから学ぶ ペネトレーションテスト


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。

2
2017
［目次］
Software Design

contents

Special Feature 1 第1特集

なぜプログラマの
役に立つのか

# いまはじめる Docker

## コンテナ技術を身につける 017



2
2017
［目次］
Software Design

## Special Feature 2 第2特集

# Linuxファイルシステムの教科書

Ext3、Ext4、XFS、F2FS、Btrfsの特徴と進化



## Special Feature 3 第3特集

# エンジニアが採用できない会社と評価されないエンジニア（後編）

なぜ入りたい会社に入れないのか？



## Extra Feature 一般記事



## Catch up trend 特別広報



## àla carte アラカルト



# contents

# Column



# Development

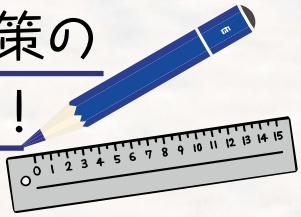


# OS/Network


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

# 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！




岡嶋裕史　著
A5判／680ページ
定価（本体2880円+税）
ISBN978-4-7741-8502-6


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）
ISBN978-4-7741-8505-7


角谷一成・イエローテールコンピュータ　著
A5判／544ページ
定価（本体1680円+税）
ISBN978-4-7741-8495-1


山本三雄　著
B5判／576ページ
定価（本体1480円+税）
ISBN978-4-7741-8496-8


岡嶋裕史　著
A5判／432ページ
定価（本体1880円+税）
ISBN978-4-7741-8501-9


庄司勝哉・吉川允樹　著
B5判／320ページ
定価（本体1480円+税）
ISBN978-4-7741-8504-0


大滝みや子・岡嶋裕史　著
A5判／744ページ
定価（本体2980円+税）
ISBN978-4-7741-8500-2


加藤昭・高見澤秀幸・矢野龍王　著
B5判／464ページ
定価（本体1780円+税）
ISBN978-4-7741-8503-3


金子則彦　著
A5判／640ページ
定価（本体3200円+税）
ISBN978-4-7741-8433-3


金子則彦　著
A5判／560ページ
定価（本体3200円+税）
ISBN978-4-7741-8387-9

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

# 技術評論社の 確定申告本

平成29年3月締切分


## 初めてでも大丈夫！ マネして書くだけ 確定申告 平成29年3月締切分

山本宏　監修／A4判／176ページ
定価（本体1,380円＋税）

ISBN978-4-7741-8442-5

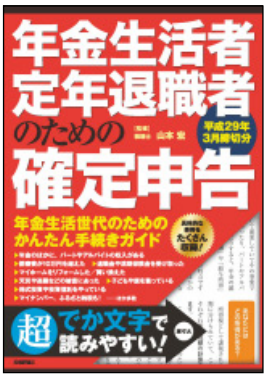
家族が働いていたり副業があるために確定申告が必要なサラリーマンや、主婦、パート、アルバイト、年金受給者はもちろん、個人事業主やフリーランサー、不動産オーナーまで、個別のケースごとに具体的な事例をたくさん掲載しています。自分の状況に近い事例を選んで、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などに、特におすすめしたい1冊です！
ふるさと納税をした方の申告方法や、確定申告の基礎知識もわかります。


## フリーランス&個人事業主 確定申告でお金を残す！元国税調査官のウラ技 第3版

大村大次郎　著
A5判／224ページ
定価（本体1,580円＋税）

ISBN978-4-7741-8441-8

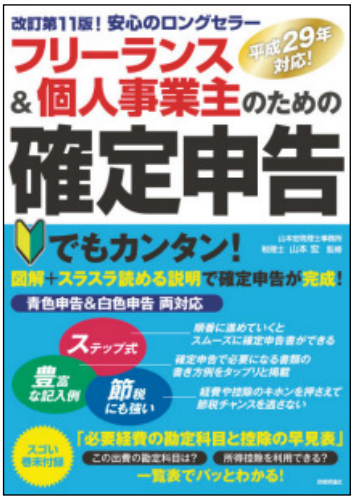
フリーランスや個人事業主の方が1年間の仕事の成果を書き入れる確定申告書。その申告書に何を書くのかによって、手元に残るお金の額は違ってきます。確定申告には、「こうしたらトクになる」というやり方がありますが、納税者に有利になる（納める税金が少なくなる）情報を税務署が教えてくれることはありません。節税にはとくに複雑な手続きは必要ないため、節税できるかどうかを分けるのは、ズバリ“知っているかどうか”です。確定申告を絶好の節税の機会にする方法を、元国税調査官の著者がお届けします！




## 年金生活者のための 定年退職者のための 確定申告 平成29年3月締切分

山本宏　監修
A4判／144ページ
定価（本体1,480円＋税）

ISBN978-4-7741-8443-2




## フリーランス&個人事業のための 確定申告 改訂第11版

山本宏　監修
A5判／256ページ
定価（本体1,480円＋税）

ISBN978-4-7741-8444-9

フリーランス&個人事業主として働く人の確定申告をサポートする定番書。
今回の第11版では、皆さんの「そうそう、それが知りたかったんだ」により応えられるようボリュームアップを行いました。ステップ式で確実にスピーディに手続きができるほか、勘定科目をさっと検索できる付録も充実しています。お手元にあることで事業を営む皆さまのお役に立てること請け合いです！

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

紙面版
A4判・16頁
オールカラー

# 電脳会議

DENNOUKAIGI

一切無料

# 新規購読会員受付中!


## 『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

## 新規送付のお申し込みは…

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

電脳会議事務局　検索

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

**一切無料!**

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

## 毎号、厳選ブックガイドも付いてくる!!


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

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


OSとネットワーク、
IT環境を支えるエンジニアの総合誌

# Software Design

毎月18日発売

**PDF電子版**
**Gihyo Digital Publishingにて**
**販売開始**

# 年間定期購読と電子版販売のご案内

**1年購読(12回)**

**14,880円**(税込み、送料無料)1冊あたり1,240円(6%割引)

PDF電子版の購入については

**Software Designホームページ**
**http://gihyo.jp/magazine/SD**

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

- インターネットから最新号、バックナンバーも1冊からお求めいただけます!
- 紙版のほかにデジタル版もご購入いただけます!
  デジタル版はPCのほかにiPad/iPhoneにも対応し、購入するとどちらでも追加料金を支払うことなく読むことができます。

※ご利用に際しては、／˜＼Fujisan.co.jp(http://www.fujisan.co.jp/)に記載の利用規約に準じます。

**Fujisan.co.jpからのお申し込み方法**

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

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

# 及川卓也の プロダクト開発の道しるべ

## 品質を高めるプロダクトマネージャーの仕事とは？

第4回 製品要求仕様書（PRD）の書き方

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

### PRDとは

前回、製品要求仕様書（PRD）のことをProduct Manager（PM）の武器と紹介しました。PRDの作成に必要なタスクをこなすことで、PMとしての役割を全うすることにつながるので、PRDを武器と評しました。今回、PRDの書き方を紹介するに際して、まずはあらためてPRDの定義を考えてみましょう。

PRDは、その名のとおり「製品への要求」を記述したものです。新製品の開発や既存製品の変更を行う場合に記述します。ここで「製品」と書きましたが、製品の粒度はまちまちです。新しい製品やサービスを新規開発する場合はその製品がPRDの対象となりますが、既存製品に対して、一部の機能を変更したり、新機能を提供する場合には、その機能をPRDとして記述します。新しい製品の場合でも、全体を1つのPRDとして記述するにはあまりにも大規模なドキュメントとなってしまうという場合には、機能ごとにPRDを用意することもあります。このように、PRDが複数のPRDに分割される形を取ることも可能ですので、運用は臨機応変に組織で決めると良いでしょう。大事なのは、書くにも読むにも適切なサイズにすることであり、それにより必要十分な情報を漏らすことなくドキュメントに含めることができるのです。

前回に述べたことの繰り返しになりますが、PRDで書くべきことは「何をするものか（What）」です。その製品の持つ目的や機能、特徴、動作などについて明確に曖昧さを排除した形で記述します。一方、「どのように実現するか（How）」は記述しません。Howについては、PRDを読むデザイナーやエンジニアが自身のスキルと経験をもとに、最適な実現方法を考えることになります。これについては、エンジニアの発想を狭めないためにも「書かなくて良い」というよりも、むしろ「書いてはいけない」としておくことをお勧めします。

PRDは開発の初期段階で完成していることが望ましいですが、現実的にはそれは不可能です。実装が進んだ段階で仕様が曖昧（あいまい）であったところが判明することもよくありますし、さまざまな理由により、当初予定していた機能を変えなければならないこともあります。PRDは製品を作るための原理原則として、常に変わらないことを期待される一方、現実にそぐわなくなった部分については、常に正されていく、そのような存在です。

### 他ドキュメントとの関係

PRDと似たドキュメントにMRD（Marketing Requirements Document）と呼ばれるものがあります。これはマーケティング担当者がマーケットの要求を記述するためのドキュメントです。MRDを元にPRDを作成するようにしている組織もありますが、筆者はこのMRDもPRDに含めて良いのではないかと考えています。グロースハックという手法が一般化したように、小さく始めたサービスを大きく育てるには、一昔前

だったらマーケティングに分類されていた活動も今はサービスの運用に組み込まれています。製品の成功に責任を負う役割のPMとしては、リリースして終わりではなく、その製品を育てていくにはマーケティング的な活動にも責任を負うことになり、必然的にマーケティングの要求をまとめたMRDにもかかわることになります。もちろん、担当者が別であり、ドキュメントのサイズが肥大化する場合などは別ドキュメントとして用意してもかまいませんが、PRDと重複する内容が多い場合などは1つのドキュメントとしてまとめてしまっても良いでしょう。

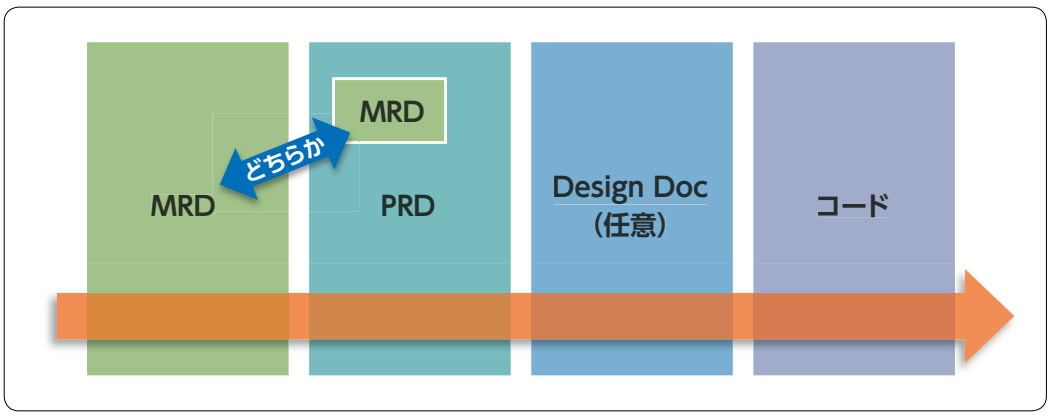
ちょっと脱線しますが、筆者はここで「サービス」と「製品」を使い分けてみました。「サービス」はWebやスマートフォンアプリケーションなど、製品そのものにマーケティング的手法を取り入れやすいものを意味し、「製品」はそれ以外にもハードウェア製品など従来製品も含むものを意味しています。多くの製品はサービス的な役割も持つようになってきていますが、製品単体としてグロースに貢献できないものや製品以外のマーケティング要素(パッケージデザインやパートナー戦略など)が多くある場合は、MRDを別に用意することをお勧めします。

PRDに含めるかどうかは別としても、MRDがPRDの前に用意されるドキュメントですが、一方、PRDの次に用意されるドキュメントがデザインドキュメントです。Design Docと略されて呼ばれることも多いこのドキュメントはPRDに書かれたWhatをもとに、Howについてエンジニアが記述するものです。FRD(Functional Requirements Document)と呼ぶこともあります。このDesign Docには、どのような技術を用いてPRDの内容を実現するかを文書化します。ただし、シンプルな実装となる場合は、Design Docを用意しないこともあります。チーム開発における実装の話になるので、ここでは詳しくは書きませんが、Design Docは効率的なエンジニアリングのために重要なドキュメントです。エンジニアはドキュメントを書くことを面倒くさがる傾向がありますが、実装を進め、いざコードレビューを他エンジニアに頼もうとしたときに、コードの内容ではなく、そもそもの実装方針について揉めることがあります。コードレビューの段階になって、ほかの実装方法にするべきであるということになったら、そのエンジニアの書いたコードは無駄になってしまいます[注1]。Design Docはこのような事態を避けるためにも、実装方針を事前に関係者と議論し、同意を得ておくためのドキュメントなのです。

◆　◆　◆

PRDの内容について、もう少し説明しようと思ったのですが、誌面が尽きてしまったようです。次回は実際のPRDの構成要素について解説します。SD

注1) もちろん、POC(Proof of Concept)として試しに実装してみることは重要です。




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

第98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。

## .NET Standard

### .NET の標準仕様「.NET Standard」

2016年9月に、Microsoftは同社が開発する.NETプラットフォームに関する新しい方針を発表しました。.NETが備えるべき共通的な機能の仕様を「.NET Standard」として定義し、個々の.NET実装はこの仕様に準拠するということを決定したのです。

今回発表されたバージョンは.NET Standard 2.0となっており、次に挙げる要素から構成されるとのことです。

- **XML**……XLinq、XML Document /XPath/XSD/XSL
- **シリアライゼーション**……バイナリフォーマッタ／データコントラクト／XML
- **ネットワーキング**……ソケット／HTTP ／メール／ Web ソケット
- **I/O**……ファイル／圧縮／ MMF (Memory Mapped File)
- **スレッド**……スレッド／スレッドプール／タスク
- **コア**……プリミティブ／コレクション／リフレクション／相互運用性／ LINQ

アプリケーション基盤として必要となる機能は網羅されているため、開発者は.NET Standardをおさえておけば基本的な.NETアプリケーションの開発が可能になるとのことです。

### .NET Standard の目的

この.NET Standard構想には、将来的な.NETの実装の分断を防ぐ目的があると発表されています。現在の.NETには、大きく分けて次の3種類の実装があります。

- .NET Framework
- .NET Core
- Xamarin

.NET FrameworkはWindows向けのライブラリを含むプロダクトで、いわゆるフル機能の.NET実装と言えるものです。

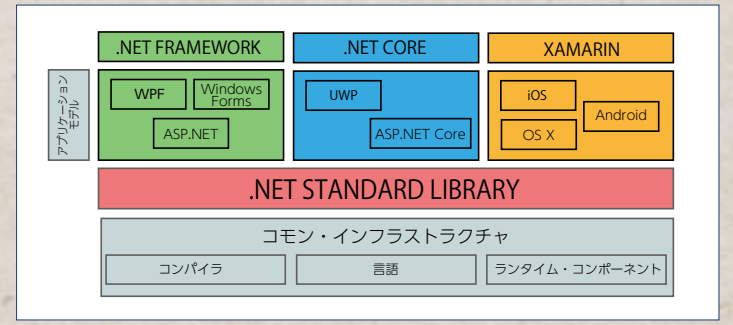
一方.NET Coreは、.NET Frameworkから核となる部分のみを抽出してパッケージ化したものです。クロスプラットフォームな設計になっている点が大きな特徴で、ユニバーサルWindowsプラットフォームアプリの開発にも対応しています。

この2つの実装に対して、2016年2月にMicrosoftがXamarin社を買収したことで、新たにXamarin（製品）が加わることになりました。Xamarinはオープンソースの.NET互換ツールであるMonoをベースとしたクロスプラットフォーム開発ツールです。OS XやiOS、Androidなどをサポートしています。したがってXamarinも.NET Coreと同様にクロスプラットフォームな.NET環境となりますが、スマートデバイスにフォーカスが当てられている点が.NET Coreとの差別化ポイントと言えます。

従来、これらの3つの.NET実装はそれぞれ異なるベースライブラリの上に構築されていました。それに対して、今後は.NET Standard仕様に準拠した「.NET Standard Library」が提供され、基本機能の一元化が図られることになります（**図1**）。.NETとしてカバーするべき要件を明確にしてコントロールすることで、無差別な分断を未然に防止しようというわけです。

.NET Standard 2.0は、次期バージョンのVisual Studioと合わせてリリースされる見込みとなっています。SD

▼図1　.NET Standard の構造（引用元：.NET Blog「Introducing .NET Standard」）




**.NET Standard**
https://github.com/dotnet/standard

# DIGITAL GADGET

vol.218

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

## ハードウェア開発を加速する

### アクセラレータというしくみ

アクセラレータというのは、生まれてまもないスタートアップのチームや個人を手助けし、その成長を加速させるための支援組織のことです（CPUや浮動小数点演算の加速装置をアクセラレータと呼ぶ場合もありますが、ここでのアクセラレータは違った意味です）。その役目は資金的な援助はもちろんですが、経験や人脈に基づき、スタートアップが失敗しないようノウハウを伝授し、良いところをより良く伸ばし、しなくてよい失敗を回避していく役目を担っています。

そのようなアクセラレータの中でも注目を浴びているHAX（https://hax.co/）は、もともとHAXLR8Rと呼ばれていた、ハードウェア、デバイスに特化したスタートアップ支援組織です。資金援助やオフィス空間の提供、メンターシップなど、必要なことがすべてまかなえる充実したアクセラレータです。現状、HAXが支援したスタートアップの90%は潰れずに残っており、かつ支援したスタートアップのほとんどがクラウドファンディングで目標額を達成するという快挙を成し遂げています。

多くのハードウェアスタートアップが、そこそこ良い感じの試作品を開発し、多くの支援金を集めたとしても、その先に待っているのは量産化と販売にまつわるさまざまな困難です。ソフトウェアで完結するプロダクトであれば、コードを書けばさまざまなことが解決し、特殊な資産を持つこともなくクラウドを活用してサービスを作り上げることができるかもしれません（もちろんソフトウェアはソフトウェアでまた違った意味での困難や競争は数多くあります）。物質的な製品が重要なハードウェアは、往々にしてそう簡単にはいきません。ハードウェアの開発は、失敗した場合、その際に失う資金や損失は計り知れません。ソフトウェアなら、随時パッチで更新が可能だったり、ソフトウェアやアプリをアップデートして不具合を回避する方法が使えますが、ハードウェアの場合は最悪の場合、製品の回収、リコールの告知、発火による火事や利用者のケガさえも起こりうるのです。

HAXでは、半年ごとに15チームのスタートアップをピックアップして支援します。いくつかの採用基準があるようです。個人での参加は認められず、必ずチームで参加することや、製品を売るべき大きな市場があること、コンセプトがしっかりしていることなどです。15チームのうちいくつかは、普通に成功しそうなプロダクトではなく、あえて突拍子もないアイデアが選ばれています。

支援を受けたチームは約3ヵ月の間、世界の工場と呼ばれる深圳にチーム全員で来て住み込み、メンターと呼ばれるその筋の指導者からのアドバイスを受けながら作業します。量産用のプロトタイプ作りから始まり、KickstarterやIndiegogoなどのクラウドファンディングで公開して賛同者や資金を集めるまで、ものすごいスピードで実行するそうです。

現在HAXが力を入れているのは、


HAXのホームページには、2012年から現在までの支援のすべてが記載されている。支援申し込みもここから行える

連載第216回（2016年12月号）で紹介したMakeBlockは、HAX卒の躍進中スタートアップ

※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。

LIFE(生活)、FAB(物作り)、INFRA(インフラ基盤)、HEALTH(ヘルスケア)、ROBOTICS(ロボット)の5分野です。その中から最近のHAX支援プロジェクトをいくつかご紹介しましょう。

## HAXの息のかかったハードウェアスタートアップ

### Kniterate:ニット編みマシン[FAB分野]

http://www.kniterate.com/

Kniterateは3Dプリンターのような感覚で編み物を自動化するマシン(**pic.1**)。単に編み物を出力するだけでなく、編み物のデータを共有し、衣服デザインも共有するという展開を想定している。単純なものであれば1枚のセーターを3時間程度で出力することができ、好みの柄や素材を自由に選択できるだけでなく、従来の衣服の流通や、人それぞれのサイズ問題といったことが一気に解消される。単なる製作ツールというだけでなく、新しいエコシステムを構築しようとしているところが特徴。

### WAZER:卓上に置けるウォータージェットカッター[FAB分野]

http://www.wazer.com/

WAZERは研磨粒子と水を高水圧で照射することで切断する、ウォータージェットカッター(**pic.2**)。通常は大規模な装置が必要であるところを、小型化したのが特徴。本体サイズは864×635×534mm。タイル、ガラス、カーボン、アルミ、ステンレススチール、チタンなどが切断できる。専用の制御ソフトウェアも用意されており、平易に使えるように配慮されている。3,599ドルから。

### DARMA:正しい姿勢を保つためのデジタルクッション[HEALTH分野]

http://darma.co/

DARMAは、座っているときの血流や血圧、心拍数、姿勢をズボン越しに測ることのできるクッション(**pic.3**)。時計型のアクティビティトラッカーは数多く発売されているが、椅子のクッションというのはまた違ったアプローチ。運動不足やエコノミー症候群を防ぐことを想定しているそう。

### dispatch:宅配ロボット[ROBOTICS分野]

http://dispatch.ai/

dispatch社のCarryというロボットは宅配業者のように道を進んで荷物を運搬する自律型ロボット(**pic.4**)。約45kgまでの荷物を運ぶことができる。車の自動運転というよりも、脇道をゆっくりと進んで移動する。現在は大学のキャンパス内での配達を実験中とのこと。ドローンよりも安い費用で、確実に配達することが特徴。

## これからのハードウェア開発の行方

近未来を描いたジョン・スコルジーのSF小説「ロックイン」では、家庭用の3Dプリンターで必要なハードウェアが出力できてしまう未来が描かれていました。現在でもそのような誤解はあり、インクジェットプリンターで電子回路がプリントできる研究などは進んでいますが、まだまだ完成された家電製品がそのまま3Dプリントできるほどの


pic.1 Kniterate


pic.2 WAZER

pic.3 DARMA


pic.4 dispatch

技術はありません。

一方、汎用的なハードウェアがさまざまな家電に組み込まれることでスマートフォンと連携し始めたり、コストが極限まで安くなったことで、使い捨てのデバイスや同じ機能を持った複数のデバイスを使い分けるということも増えてきました。最近日本でも販売が開始されたAmazon Dashは、ボタンを押すとある特定の商品をAmazonに注文するという単機能の小型デバイスで、商品ごとにボタンが用意されています。

HAXでは最近、すでに製品が出来上がっているものを、よりたくさん大きく広げるために、HAX BOOSTというしくみでの支援も行っています。流通経路を育て、小売組織とのつながりを強める方法を学び、広く世界に展開していくのです。アプリケーション販売が、箱に入ったパッケージを店頭で購入するものから、ネット上のアプリストアで購入するものに変化してきたのと同様に、ハードウェアもAmazonの倉庫で在庫を管理し、販売や運送はAmazonに任せてしまうといったアウトソーシングが可能になってきました。こういった設計や組み立て、在庫管理、販売、配送などさまざまな作業を誰かに任せることができれば、最初のアイデアに注力することができます。アイデアがアイデアのまま終わらずに実現可能な世界がやってきているわけです。

HAXの比喩によれば、ソフトウェアの開発は「ロックバンド」であり、数人のスタープレーヤーがいれば人気のバンドになれるのが特徴です。その一方、ハードウェアの開発は「オーケストラ」であるとのこと。それは、すべての人々がそれぞれの役目を全うし、一人でも調和を乱すとすべてをダメにしてしまうといった比喩です。これからも多くの人々の協力で、まだ見たことのないような、便利ですばらしい製品の出現が加速（アクセラレート）してほしいものです。SD

Gadget 1

## Nura

https://www.facebook.com/nuraphone/

### 自動調整ヘッドフォン

Nuraは装着後、ユーザの耳の感度に合わせ、各周波数ごとの音を最適化してくれるヘッドフォンです。調整にかかるのはわずか30秒。一度調整した後はユーザ設定が記憶され、次回からはどの人がヘッドフォンを装着したのかを2秒で判別します。インナー型イヤホンと耳が隠れるタイプのオーバーヘッド型の両方を併せ持った形となっており、その人の耳の特性に合わせた音質に調整されたうえで、インナー側では高音域を、オーバーイヤーで全体の周波数の音を再生します。クラウドファンディングでプリオーダー中で、一般販売の際は399ドルの予定。

Gadget 2

## PRYNT

https://www.pryntcases.com/

### スマートフォン用携帯プリンター

PRYNTは、スマートフォンのケースとして持ち運べるモバイルプリンターです。スマートフォンで撮影した写真をその場でプリントできる利点と、プリントアウトした写真をスマートフォンにかざすと、AR（拡張現実）機能によって、写真が動いているかのような動画を楽しむことができます。印画紙にはZINKフォトペーパーを用い、PRYNT本体に10枚収納可能です。1回の充電で約20枚のプリントが可能。重さは225gです。

Gadget 3

## HABITAWARE

https://www.habitaware.com/

### クセ検知デバイス

HABITAWAREは、人が知らず知らずのうちに繰り返している変なクセ、たとえばツメを噛むとか、前髪をいじるとか、指をくわえるとか、鼻をかくといった腕や手の特定の動きを検知し、スマートフォンと連携して記録し、注意をうながしてくれるウェアラブルデバイスです。クセの動作や、生じる時間帯などを記録したうえで、身につけたデバイスの震動通知によりクセを意識するきっかけを作ることができます。149ドルで販売の予定。

Gadget 4

## Trainerbot

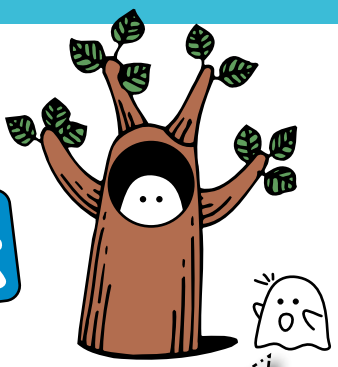
http://trainerbot.com/

### 卓球トレーニングロボット

Trainerbotはスマートフォンで細かにコントロールできる卓球練習用のいわゆる投球マシン。卓球ボールの速度、回転、落ちる場所、角度などが、コントロールでき、不得意な球筋を練習したり、左右に切り替えながら投球したりと、実戦さながらの訓練ができます。難易度を設定したり、ある難易度をクリアできるか、ライバル同士で競争しても良いそう。400ドルほどで販売の予定。

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

# 結城浩の再発見の発想法 45



## メモ化

### メモ化とは

**メモ化**(Memoization)は、プログラムを高速化する技法の1つです。メモ化では、**入力に対応する計算結果を記録しておき、以前と同じ入力が与えられたなら、記録しておいた計算結果を出力**します。計算を再度実行することなく、記録しておいた計算結果をそのまま出力するので計算時間を節約することができるというわけです。

簡単な例として、$n$番目のフィボナッチ数$F_n$を計算するプログラムを考えてみましょう。入力は$n$で、出力はフィボナッチ数$F_n$です。フィボナッチ数の漸化式は、

$$\begin{cases} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \quad (n = 2,3,4,\ldots) \end{cases}$$

**▼リスト1　フィボナッチ数を求める関数fib1（メモ化なし）**

```
def fib1(n)
  if n == 0
    0
  elsif n == 1
    1
  else
    fib1(n-1) + fib1(n-2)
  end
end
```

ですから、そのままRubyのプログラムとして書くと**リスト1**のようになります。

**リスト1**の関数fib1は正しく動作しますが、入力$n$が少し大きくなるだけで、出力を得るまでの時間は非常に長くなります。これは関数fib1の中でfib1を再帰呼び出ししており、入力$n$が大きくなると、呼び出し回数が指数関数的に増えてしまうからです。

**リスト2**に、メモ化を使って高速化した関数fib2を示します。関数fib2のほとんどの部分はfib1と同じです。違うのは、MEMOを使ったメモ化の部分だけです。一度でも計算したことがあるフィボナッチ数$F_n$はMEMO[n]に記録されま

**▼リスト2　フィボナッチ数を求める関数fib2（メモ化あり）**

```
MEMO = Hash.new(nil)

def fib2(n)
  if n == 0
    0
  elsif n == 1
    1
  else
    m = MEMO[n]
    if not m
      m = fib2(n-1) + fib2(n-2)
      MEMO[n] = m
    end
    m
  end
end
```

す。関数fib2では、再帰呼び出しで計算をする前に、過去に計算したことがあるかどうかをまず調べます。もしも計算していたらその記録を出力し、計算していなかったときに限り実際に計算を実行します。これによって、同じ入力に対する再計算という無駄を防ぎ、高速化ができるのです。

fib1で何時間もかかるような大きな$n$の場合でも、メモ化で高速化したfib2は一瞬で計算を終えます。

フィボナッチ数を求める計算は単純な例ですが、メモ化を行えば、計算がどれほど複雑であっても高速化が簡単に実現できます。メモ化では、計算のアルゴリズムを高速化するわけではなく、入力に対応する出力をただ記録しておくだけで済むからです。

しかし、どんな計算に対してもメモ化が有効なわけではありません。まず、メモ化では過去の記録を検索しますから、検索にかかる時間よりも計算にかかる時間のほうが長いものでなければ意味がありません。

また、計算のたびに結果が異なるような計算はメモ化できません。入力をキーとして記録された出力を探すわけですから当然ですね。メモ化を使うには、入力から出力が一意に定まる必要があるのです。

さらに、同じ入力が何回も発生するような計算でなければメモ化で高速にはなりません。過去の計算結果を記録しておいて再利用するわけですからこれも当然です。メモ化では、計算時間を節約するために、以前の出力を記録しておくための空間を消費します。ですから、入力のバリエーションが多過ぎたり、出力を記録するための空間を大量に消費する計算の場合には、メモ化は使えないでしょう。つまり、メモ化には時間と空間のトレードオフがあるということです。

ただし、メモ化のいいところは「実際に必要になった計算のみを行っている」という点です。「こういう入力が来るかもしれないから、それに備えて前もって計算しておこう」と考えると、もしかしたら無駄な記憶領域を消費してしまうかもしれません。その意味では、メモ化は本当に必要な計算しか行っていないと言えますね。

## 日常生活とメモ化

メモ化によるプログラムの高速化は直観的にもわかりやすく、またシンプルなしくみですから、失敗も少ないでしょう。

私たちの日常生活でも、メモ化に似た高速化はよく行われます。たとえば、多くの参考書を使って**時間がかかる調査**をしているとしましょう。事項を調べるのに手間や時間がかかるとしたら、調べた結果を自分でまとめておけば便利です。同じ事項を調べる必要が生じたときには、自分の作ったまとめを見ればいいからです。これはまさにメモ化ですね。

顧客からの問い合わせに答えるサポート業務でも、メモ化に似た活動が行われます。顧客からの問い合わせとその回答を**事例研究**として共有しておくことで、類似の問い合わせが発生したときに短時間で回答することができるからです。

調査であれ、事例研究であれ、入力と出力の対応関係についてはよく理解しておく必要があります。私たちの世界は常に変化を続けていますから、過去の記録が現在では当てはまらなくなってしまう危険性があるからです。その意味では記録のタイムスタンプが重要ですね。

また、時間と手間を省くために過去の記録を利用しようとするのはいいのですが、検索にかかる時間を考慮することは大切です。メモ化が有効になるのは、検索にかかる時間が短いときに限るからです。

◆ ◆ ◆

あなたの周りを見回して、時間や手間がかかる作業が繰り返されていることはないでしょうか。その作業結果を記録しておくことで、時間や手間を軽減することはできないでしょうか。

ぜひ、考えてみてください。SD

増井ラボノート

# コロンブス日和

## 第16回　Scrapbox（1）

**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

## Scrapboxの紹介

2017年1月号ではさまざまなファイルや書類を「フラット」に管理する方法について説明しましたが、今回はさまざまな情報をより効果的に整理し共有できる「Scrapbox」というシステムを紹介します。

## 理想の情報整理システムとは

2017年現在における理想的な情報整理法とはどのようなものでしょうか？　現在のテクノロジ状況をふまえつつ要件を考えてみます。

### ブラウザを使ってネット上で情報を管理する

ネットやブラウザを使えない環境は現在ほぼなくなりましたから、情報整理は当然これらを活用するべきでしょう。

### 一ヵ所で集中管理する

ネット上の1つの場所にあらゆるデータを保存してブラウザからアクセス可能にするのが良いでしょう。

### 気軽な入力と編集

ブラウザ上での入力／編集手法は極力単純にしておく必要があるでしょう。

### タグを活用する

先月号でも解説しましたが、個人的な雑多なデータを階層的に管理することは常人にはとても難しいので、タグを使って管理するのが良いと思われます。

### 他人との情報共有

Web上に置いた情報を簡単に共有できると便利でしょう。

現在、Web上で情報を整理したり共有するためのさまざまなシステムが利用されていますが、上記の要望をすべて満たすものはありませんでした。

## Scrapbox

私は、Web上に気軽に情報を書いて簡単に共有できるScrapboxというシステムを開発して利用しています。長年に渡って「Gyazz」という名前のWikiを作成して使っていましたが、Scrapboxはこれを大幅に改良して前述の要件をすべて満たすようにしたものです（**図1**）。

Scrapboxは次のような特長を持っています。

### ブラウザ上で情報編集／整理

Scrapboxではあらゆる情報をWeb上に置き、ブラウザで閲覧や編集を行います。このためあらゆるパソコンやスマホなどから


◀ **図1**　Scrapbox（https://scrapbox.io/）

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

同じデータにアクセスできます。

### グループで利用

Scrapboxのページにはユニークなurlがついており、個人で利用することも共有して利用することもできます。

### 強力な編集機能

Scrapboxのページはemacsのようなテキストエディタと同じようにブラウザ上でWYSIWYG（What You See Is What You Get）的に編集を行えます。編集されたテキストは自動的に保存されるので保存ボタンなどを押す必要はありません。

### リアルタイム同時編集

Scrapboxではgoogle docsと同じように、複数ユーザが同時にリアルタイムにWYSIWYG編集することができます。

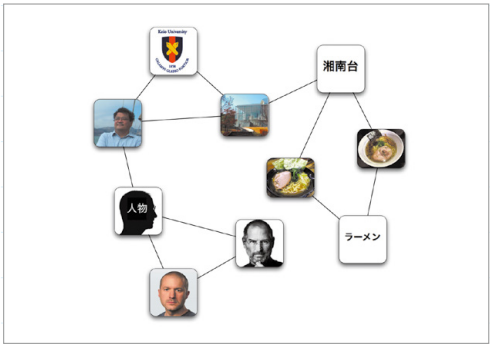
### リンクの活用

Scrapboxではタグを利用して情報管理を行います。Scrapboxはwikiなので、「増井」というページに「人物」というタグを書くとこれは「人物」というページへのリンクになりますが、このとき「人物」ページから「増井」ページへのリンクも同時に作成されるようになっています。このため「増井」のような名前を忘れたとしても「人物」ページから「増井」ページへのリンクを見つけることができます。

このような「逆リンク」を利用すると、特別なタグ検索機能を用意しなくても、タグ的なものをすべて独立したwikiページとして作成するだけでタグを用いた検索が利用できることになります。

また、Scrapboxではリンク先ページからリンクされているページ（2ホップ先のページ）までを関連ページとして表示するようになっているので、「Steve Jobs」のページも「人物」ページにリンクされている場合「増井」のページから「Steve Jobs」のページを直接参照できます（**図2**）。

▼**図2** リンクの構造




たとえば「湘南台」のページを表示したときは、湘南台のラーメン屋のページや私のページは表示されますがSteve Jobsのページは表示されませんし、Steve Jobsのページを表示したときは「人物」つながりで私のページやJonathan Iveのページは表示されますが、慶應大学や湘南台のページは表示されません。

### 代表画像

リンク先ページを表示するとき、ページに含まれる画像を表示するようになっているので、関連ページの視認性がよくなっています。

### その他各種の便利機能

各種の簡単なタグ記法、プログラムコードをきれいに表示する機能、簡単な記法による文字装飾、テキストと同じ大きさで画像を文中に埋め込むアイコン表示機能など、たくさんの便利な機能が搭載されています。

## Scrapbox利用例

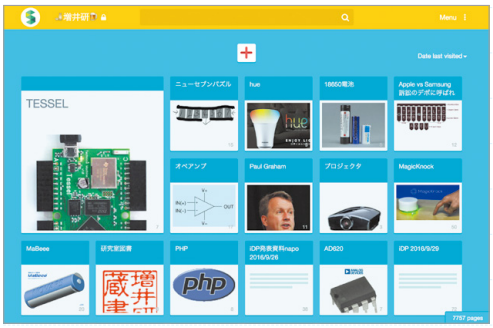
Scrapboxは Web[注2] から利用できます。Scrapboxを長年に渡って利用してきた例を紹介します。

### 研究室での利用

私の研究室では数年に渡ってScrapboxを利

注2） http://scrapbox.io/

▼図3　研究室のトップページ




用しており、2016年末現在、約8,000ページが作成されています。研究室に所属する学生の興味はいろいろであり、論文や研究トピックや開発Tipsのような情報に加え、ラーメン情報もアニメ情報も部品情報もイベント情報もすべて同じところに置いてあります。雑多な情報が何千ページもあるとたいへんなことになりそうですが、とくに分類を行わなくても関連ページやカテゴリをページとして関連付けておくだけで、これらの情報が適切に分類管理されるのが便利です。

研究室のScrapboxのトップページは**図3**のようになっています。研究関連情報からアニメ情報、ラーメン情報までかなり雑多な情報が並んでいることがわかります。

ここで「AD620」という部品に関するページを選んで表示すると、その部品の詳しい情報が表示され、似た部品など関連研究のページが表示されます(**図4**)。この部品ページには「ストレンゲージ」「オペアンプ」というページへのリンクが定義されているので、同じページへのリンクを持つ「キッチンスケール」や「LMC660」が関連ページとして表示されているというわけです。

一方、藤沢のラーメン屋のページを選んで表示すると、藤沢やラーメン屋のページがたくさん関連ページとして表示されます(**図5**)。部品情報もラーメン屋情報も同じ場所に書いてあるのですが、リンク関係がまったく違っているので別のクラスタとしてうまく管理できています。

▼図4　1つの部品ページを開くと関連する情報がすべて表示される


## UIPedia

前述の例は研究室内の人間だけが参照できるページでしたが、ユーザーインターフェースに関連する論文やシステムなどをScrapboxで公

開しています（図6）。UIPediaページは誰でも閲覧できますし、参加も可能になっています注3。

文献情報を管理するさまざまなシステムが利用されていますが、文献データベースでは入力できる情報の属性が限られているのが普通です。Scrapboxを使うと、タイトルや著者名のような一般的な書誌情報だけでなく著者の写真や弟子筋情報、配偶者情報を記述したり、普通の文献整理システムではできないことができるので味のある情報ページにできます。

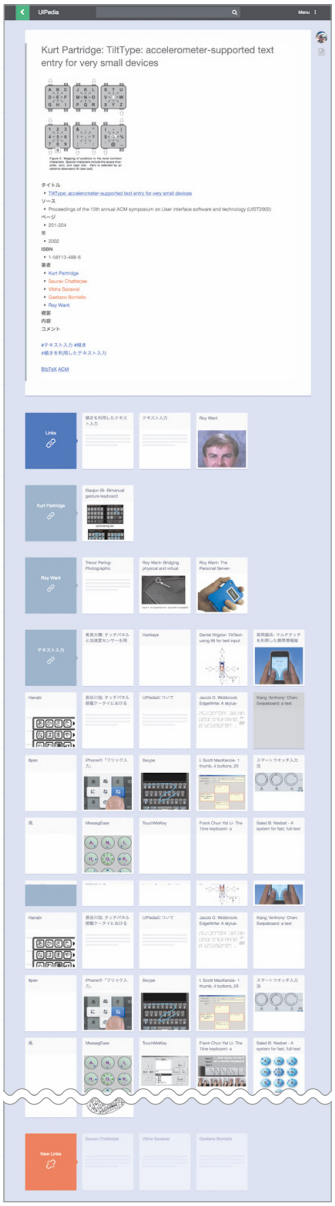
## 家族情報

私は個人的なメモや予定表、TODOなどをすべてScrapboxで管理しており、現在6,000ページほどが作成されています。また家族間で共有したい情報もScrapboxで管理しています。親戚の連絡先、銀行口座情報、各種の契約情報、予定表など、家族間で共有したい情報は意外とたくさんあるものです。こういった情報は住所録やスケジュール帳のようなアプリケーションで管理している人が多いと思いますが、家族関連情報はすべて1つの場所に置いておけば何かと便利です。私の場合、家紋の情報、家系図、引っ越し履歴など家族に関連するさまざまな情報を書いていたら簡単に100ページを越えてしまいました。

私はこの考えに基づくシステムを10年以上利用しており、Scrapboxはその最新版です。多くの機能を盛りこんだために現在のScrapboxの仕様はある程度は大きなものとなっていますが、多くのアイデアは「コロンブスの卵」的であり、仕様のシンプルさは保てています。次号では、Scrapboxのさらに詳しい利用法および実装について解説します。SD

▼図5 藤沢のラーメン屋のページを開いたところ


▼図6 入力インターフェースの研究ページ




注3） https://scrapbox.io/UIPedia


# 第12回 2016年のOSCも無事に終了

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

## 2016年もたくさん飛び回りました

早いもので、この連載も第12回。1年経ってしまいました。2016年のオープンソースカンファレンスもOSC広島で無事に終了し、全14回の開催をすべて終了しました。今年も北は北海道から南は沖縄まで、全国各地を飛び回る1年でした。

今年は延べ参加者数が約7,000名と、前年に比べて若干参加者が減ってしまいましたが、一方で各開催で学生のみなさんに、積極的に参加、交流してもらう機会を作ったことで、内容的には充実していたように感じます。とくに今回レポートする福岡では、学生の活躍が目立ちました。

## OSC福岡は10回目の開催

OSC福岡は2007年12月に初開催後、今年が10回目の開催でした。過去には何度か会場に大学や専門学校の校舎を使わせていただいたり、学校の先生が実行委員に入ってくれたりと、学生のみなさんが参加しやすい土壌です。今回も実行委員長を福岡大学の学生である前田恵里さん(**写真1中央**)が務めてくれました。

直前のOSC東京秋にもスタッフとして参加してもらい、運営に慣れてもらったおかげで、しっかりと大任を果たしてくれました。今後、そのほかの地域で開催するOSCでも、学生さんにリーダーとして頑張ってもらいたいですね。

もちろん、ほかにも多くの学生さんがスタッフや講師を務めてくれました。たとえば、私も趣味で遊んでいるRaspberry Piを使ったオーディオ再生で出展してくれた松本壮史さんは佐賀県在住の大学生ですが、展示会場にたくさんのオーディオ機器を持ち込んでデモ展示を行い(**写真2**)、セミナーにもたくさんの参加者を集めていました。ただ、セミナーが30分で終わってしまったため、私も飛び入りでラズパイオーディオのネタを話させてもらいました(ちょっと狙ってたのは秘密です)。

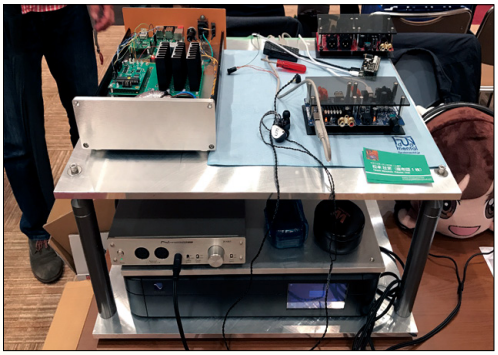
こんな若者を、来年のOSCでもたくさん発掘していきたいですね。

## サプライズで第5回OSCアワード表彰式

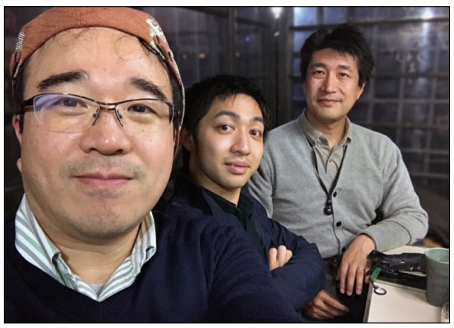
閉会式では、1回目の開催から縁の下の力持ちとして貢献していただいている坂本好夫さん(**写真1右**)にOSCアワードを贈呈しました。サプライズ受賞ということで最後まで黙っていたのですが、何も言わなくても第1回のときに使ったスタッフTシャツ(黄色)を着てきてくれていました。偶然にも、私も試作した黄色いスタッフTシャツ(現行デザイ

**▼写真1 実行委員長の前田さん、OSCアワード受賞の坂本さんと一緒に記念撮影**

**▼写真2 松本さんの持ち込んだラズパイオーディオのデモ機材。左上の銀色の筐体にRaspberry Piが組み込まれています**



▼写真3　今回も登場、日本MySQLユーザ会の坂井 恵氏(右)も屋台まで付き合ってくれました



▶写真4　懇親会後、みんなで記念撮影。美味しいお酒を飲み過ぎて、みんなおかしくなっています

ン)を着ていたので、新旧黄色Tシャツでそろい踏み。表彰盾に刻んだお名前が間違えているというアクシデントがありつつも、和やかに表彰式を執り行わせていただきました。

## 博多といえば中州の屋台ですよね

さて、恒例のお酒の話。OSC福岡では前日準備が終わったあと、前夜祭をするのが恒例ですが、さらに2次会では中州の屋台に繰り出しました。屋台はたくさん並んでいますが、今回は以前お邪魔したことがある「紀文」さんに入りました。ほかの屋台はどちらかというと観光客相手という感じですが、こちらは正当派の屋台です。美味しい焼き魚とお酒をいただき、翌日の本番に向けて鋭気を養ったのでした。透明なビニールシートで囲った屋台が多い中、格子戸で囲われた屋台ですので見つけやすいです。もし中州の屋台で行くところに迷ったら探してみてください。ただ、あまりお店にそぐわない雰囲気の客だと思われると、店主の親父さんに追い返されてしまうこともあるので注意してくださいね。親父さん曰く、お客さんが楽しめる雰囲気を大事にしているのだそうです。ガンコ親父というわけではありませんので、普通に行けば大丈夫です。

## 福岡の懇親会も持ち込みのお酒だらけ

今回のOSC福岡の懇親会は、展示会場を片付けてそのまま立食形式で開催。ビールなども用意しましたが、やはりメインは各自持ち寄ったお酒の飲み比べです。九州のお酒というとやはり焼酎ですが、鹿児島からの参加者の方が東京ではプレミアムな焼酎である「伊佐美」を持ってきてくれたり、佐賀大学農学部で作られている大学限定酒や、同じく佐賀の「鍋島」など、美味しいお酒がたくさん集まりました。

OSC広島のレポートもしようと思ったのですが、盛りだくさん過ぎたので、次回は広島、そして香川と移動して学生さんたちと交流したお話をしたいと思います。SD

Report

## 焼酎工場も見学してきました

開催翌日の日曜日に代表的な焼酎である「いいちこ」の日田蒸留所を見学してきました。偶然にも紅葉祭を開催しており、アンケートに答えて試飲用ミニボトルをゲット。さらに試飲コーナーでさまざまな焼酎を飲んだ後、自分の好みのいいちこが作れる原酒セットを購入。焼酎を蒸留するときの圧力を、常圧(とくに圧力を変えない)にしたり、減圧(圧力を下げる)したりすることで味が変わるそうです。それらの原酒をブレンドすることでいろいろな味の銘柄が作られているとか。年末年始の休みの間、じっくりと楽しもうと思います。

**いいちこ日田蒸留所で記念撮影。こんなに巨大なボトルだったら、簡単には飲み終わらなさそうです**

# ツボイのなんでもネットにつなげちまえ道場

第20回

## CとC++

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

### ET2016

「Embedded Technology」という展示会をご存じでしょうか。毎年11月くらいに、みなとみらいにあるパシフィコ横浜で開催されている組み込み技術に関する展示会です。昨年も11月16日から18日にかけてET2016が開催され、たいへん盛況でした。ここ数年間、筆者は一般来場者としてETに参加してきました。しかし今回はARMブース（**写真1**）で「ARMを触ってみよう」という体験コーナー（**写真2**）の手伝いと、同ブースでプレゼンを行う機会（**写真3**）をもらったので、出展者タグでの参加になりました。

### IARシステムズ

ところで、IARシステムズ[注1]という会社をみなさんご存じでしょうか。1983年にスウェーデンで創業した会社で、「世界初の組み込みC言語コンパイラメーカー」ということです。Embedded Workbenchという開発環境を販売していて、ARM用のC/C++コンパイラのほかにも、各社のマイコン用のコンパイラを提供しています。ちなみに、mbed OS 5からmbed enabledの認定を得るためには、このIARシステムズのコンパイラにも対応することが要件になりました[注2]。

このIARシステムズもET2016にブースを出していました。スウェーデンの会社ということで、スウェーデンコーヒー（**写真4**）がブースで振る舞われていて、筆者はとても気に入って自宅用にも買おうかなと思ったくらいです。

それはさておき、このブースでちょっとおもしろいクイズがあったので紹介したいと思います。出題されたC言語のコードの出力結果を解答用紙に記すというものでした。

**リスト1**をご覧になって回答してみてください。インチキプログラマであるところの筆者は、

注1） https://www.iar.com/jp/

注2） https://www.mbed.com/en/about-mbed/mbed-enabled/mbed-enabled-program-requirements/

▼写真1　ET2016でのARMブース


▼写真2　ブース内の「ARMを触ってみよう」コーナー

▼写真3　ARMブースでプレゼンを行う筆者


▼写真4　スウェーデンコーヒー


演算子の優先順位を完全に覚えていません。C言語のコードは、コンパイラに食わせて処理するのが正しいだろうということで、手元のCコンパイラが入っているマシンとmbedのオンラインコンパイラで処理をしてみました。

まず**リスト1**を、筆者の手元のMacBookで入力してtest.cとして保存、gccでコンパイルし、実行してみました(**図1**)。C言語は親切ですね。ちゃんと演算子の優先順位を教えてくれました[注3]。

手元のマシンにCコンパイラが入っていないという人向けのソリューションが、mbedのオンラインコンパイラです。出題のプログラムでインクルードされているヘッダファイルをmbed 2.0のものに書き換え(**リスト2**)、オンラインコンパイラでコンパイルをしてみました。この内容であれば、ほぼすべてのmbed対応ボードをターゲットとして使うことができます。

オンラインコンパイラでビルドし、できあがったバイナリを手元にある適当なmbedで実行して

▼リスト1　クイズ1

Q1 printfで表示されるresultの答えはいくつでしょうか

```
#include "stdio.h"
int main(void)
{
    unsigned int a = 1;
    unsigned int b = 1;
    unsigned int c = 2;
    unsigned int d = 2;

    unsigned int result = a << b * c + d;

    printf ("result: %d\n", result);

    return 0;
}
```

▼図1　クイズ1をgccでコンパイルし、実行してみる

```
yoshi$ gcc test.c
test.c:9:38: warning: operator '<<' has lower precedence than '+'; '+' will be evaluated first
      [-Wshift-op-parentheses]
    unsigned int result = a << b * c + d;
                          ~~ ~~~~~~^~~
test.c:9:38: note: place parentheses around the '+' expression to silence this warning
    unsigned int result = a << b * c + d;
                                   ^
                               (       )
1 warning generated.
yoshi$ ./a.out
result:16
```

注3)　ちなみに演算子の優先順位は＊＞＋＞＜＜です。

▼リスト2　リスト1のヘッダをmbed 2.0用に書き換えた

```
#include "mbed.h"
Serial pc(USBTX, USBRX);
int main(void)
{
    unsigned int a = 1;
    unsigned int b = 1;
    unsigned int c = 2;
    unsigned int d = 2;

    unsigned int result = a << b * c + d;

    pc.printf ("result: %d\n", result);

    return 0;
}
```

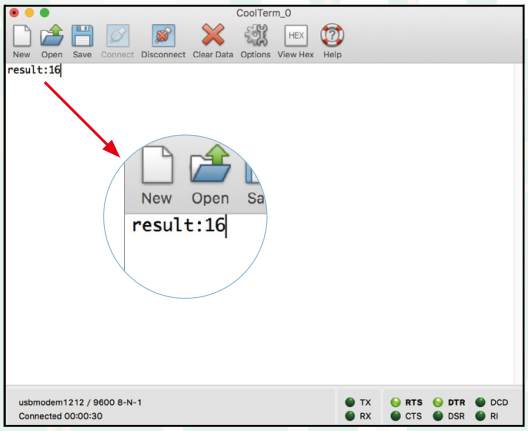
みます。mbedを接続したパソコンでシリアルターミナルを開き、mbedのリセットボタンを押すとバイナリが実行され、ターミナルに実行結果が出力されます(**図2**)。

クイズ2は**リスト3**のようなプログラムです。このプログラムもクイズ1と同様に、筆者の手元のMacBookで入力してtest2.cとして保存、コンパイルと実行をしてみました。

▼リスト3　クイズ2

```
Q2 printfで表示されるs1+s2の答えはいくつでしょうか？

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int s1,s2;
    char *p1, *p2;

    s1 = sizeof('z') * 10;
    s2 = sizeof(char) * 10;

    p1 = malloc(s1);
    p2 = malloc(s2);

    printf("s1+s2= %d\n", s1+s2);

    return 0;
}
```

▼図2　クイズ1をオンラインコンパイラで実行してみた




```
yoshi$ gcc test2.c
yoshi$ ./a.out
s1+s2=50
```

次に、先ほどのようにヘッダファイルをmbed 2.0のものに書き換え(**リスト4**)、オンラインコンパイラで実行してみましょう。

オンラインコンパイラでビルドしようとすると、**図3**のようなエラーとワーニングが出ます。

怒られたので、7行目をコメントアウトし、12行目と13行目を次のように書き換えてキャスト

▼リスト4　クイズ2のヘッダをmbed 2.0用に書き換えた

```
#include <mbed.h>
// stdlib.hはmbed.hでincludeされています。
Serial pc(USBTX, USBRX);

int main()
{
    int s1,s2;
    char *p1, *p2;  // 7行目

    s1 = sizeof('z') * 10;
    s2 = sizeof(char) * 10;

    p1 = malloc(s1);  // 12行目
    p2 = malloc(s2);  // 13行目

    pc.printf("s1+s2= %d\n", s1+s2);

    return 0;
}
```

▼図3　コンパイルエラーとワーニング

```
Error: A value of type "void *" cannot be assigned to an entity of type "char *" in "main.cpp", Line: 12, Col: 9
Error: A value of type "void *" cannot be assigned to an entity of type "char *" in "main.cpp", Line: 13, Col: 9
Warning: Variable "p1" was set but never used in "main.cpp", Line: 7, Col: 12
Warning: Variable "p2" was set but never used in "main.cpp", Line: 7, Col: 17
```

してみます(ところで、p1とp2は出力に使っていないのに、なぜ計算しているのでしょうね……)。

```
    char *p1 = (char *) malloc(s1);
    char *p2 = (char *) malloc(s2);
```

無事、コンパイルが通るようになったので、実行してみました(図4)。

あれ、Mac上でgccでコンパイルしたものと、mbedのオンラインでコンパイルしたものとで結果が違っちゃいましたね。

いくらなんでもこういうことが起きると困るので、原因を探ってみることにします。

printf()の行を書き換えて、s1とs2がそれぞれどうなっているのか表示してみます。C言語でコンパイルするとs1が40でs2が10、オンラインでコンパイルするとs1が10でs2が10でした。つまり、s1、「sizeof('z')」の計算結果が異なっているということがわかります。

ここでなぜ「sizeof('z')」の結果が異なるのかを考えてみたところ、あることに気づきました。Cでは文字定数'z'がintとして扱われるので4バイト、C++ではcharとして扱われるので1バイトです[注4]。

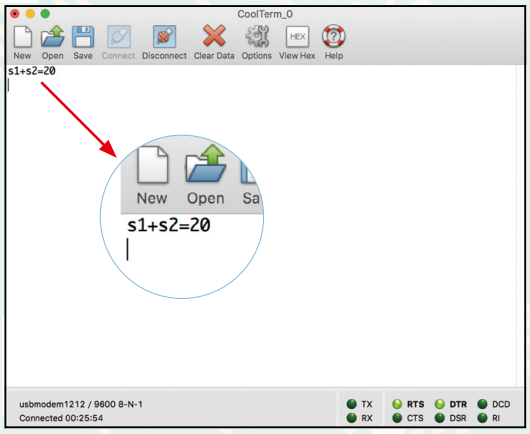
そういえば、C言語でコンパイルしたときのソースのファイル名はtest2.cで、オンラインコンパイラでコンパイルしたときのファイル名はmain.cppでした。

試しに、C言語でコンパイルするソースのファイル名(拡張子)をcppにしてみます。

```
yoshi$ mv test2.c test2.cpp
yoshi$ gcc test2.cpp
yoshi$ ./a.out
s1+s2=20
```

やはり、先ほど気づいたようにC言語とC++言語で'z'が、それぞれどういう型として扱われるかということが原因のようです。

▼図4　クイズ2をオンラインコンパイラで実行してみた




## まとめ

みなさんご存じでしょうが、C言語の仕様では、intやlongなどの整数型はサイズが定まっていません。このため、C99やC++11で用意されているuint32_tやint8_tなどの環境に依存しない型を使ったほうがよいでしょう。また、クイズ2のように、C言語とC++言語で型が異なるような設問を出すのであれば、C言語なのかC++言語なのかを明らかにしたほうがよいでしょう。

期せずして、C言語とC++言語について、いろいろ考えさせられました。なかなかおもしろいきっかけでした。SD

注4)　根拠については、http://www.bohyoh.com/CandCPP/FAQ/FAQ00004.htmlを参照してください。

# 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2017年2月16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 高速無線LANルータ「WNPR2600G」

**1名**

「新しい技術トレンドや使い方を安心の日本品質で提供し、根付かせていく」がコンセプトの新ブランド「PLANT」。その第1弾製品、IEEE802.11ac規格の高速無線LANルータです。規格値は1,733Mbps、全ポートがギガビットに対応しています。複数の子機に対して同時に電波を出せる技術「MU-MIMO」に対応し、各端末との通信速度を落とさず、安定した通信が可能になります。

提供元 アイ・オー・データ機器 http://www.iodata.jp

## 02 ディレクターズ10周年記念バッグ

**1名**

ディレクターズが2017年の創業10周年を記念して、女性向けPCバッグを作りました。本皮を使用した高級感のあるつくりで、中がふかふかの素材で作られたポケットには13インチのPCがすっぽり入ります。今月は『ベージュ』をプレゼント。

提供元 ディレクターズ http://www.directorz.ad.jp

## 03 ご近所猫の写真集 ねこ舌 ヒロジカズオ 著


ご近所で見かける猫たちの一瞬をとらえた、猫好きの猫好きによる猫好きのための写真集です。毛づくろいの途中の舌や、大あくびしているときの舌、まれにしまい忘れの舌など、見られたらラッキーな“ねこ舌”を集めてみました。

提供元 技術評論社 http://gihyo.jp

**2名**

## 04 データサイエンティストの秘密ノート

高橋 威知郎、白石 卓也、清水 景絵 著


データ分析の初心者を対象に、データ分析技法の実際を解説した1冊。ソフトバンク・テクノロジーのデータサイエンティストが、実際に手掛けた事例を失敗点とその克服法とともに紹介していきます。

提供元 SBクリエイティブ http://www.sbcr.jp

**2名**

## 05 やさしく学べるMySQL運用・管理入門

山崎 由章、梶山 隆輔 著


MySQL 5.7の運用・管理の現場で初心者が押さえておくべき内容にしぼって解説しています。「1レッスン45分」のセミナー感覚で学習しやすく、章末の演習問題を解くことで要点が理解できます。

提供元 インプレス https://www.impress.co.jp

**2名**

## 06 Pythonクローリング＆スクレイピング

加藤 耕太 著


PythonによるクローリングやAPIを活用したデータ収集、HTMLやXMLの解析から、データ分析や機械学習の処理まで解説。強力なライブラリとフレームワークを活用して、高効率に開発しましょう。

提供元 技術評論社 http://gihyo.jp

**2名**

## 07 はじめよう！プロセス設計

羽生 章洋 著


『はじめよう！要件定義』の続刊です。業務フローの見える化・しくみ化を行う「プロセス設計」の手法を学び、日々の仕事から業務改革、IT化プロジェクトの問題を解決しましょう。

提供元 技術評論社 http://gihyo.jp

**2名**

第1特集

# なぜプログラマの役に立つのか
# いまはじめるDocker
## ～コンテナ技術を身につける～


環境を特定用途ごとに隔離できるコンテナ技術を利用して、開発を効率化する現場が増えています。Dockerの登場がその流れを加速していますが、Dockerで構築するものはアプリケーションを動かす基盤側のため、自分とは縁遠い話だと感じていたプログラマの方もいらっしゃるのではないでしょうか。

しかし、プログラマ自身がDockerの恩恵を理解していなければ、自分の仕事を効率化できるチャンスをみすみす逃してしまいます。もしいままでDockerに手をつけていないなら、なにはともあれ自分の手元で動かして試してみましょう。そこがスタートラインです。コンテナ技術を適用しやすいところ／しにくいところは導入事例を参考にしてみてください。

# 第1章 プログラマのためのコンテナインフラ環境とは?

# いっきに押さえる Dockerの基礎からKubernetesまで


インターネット上のアプリケーション開発で、コンテナを使用することは決して珍しいことではなくなりました。今は、インフラエンジニアの手を離れ、アプリケーションを実際に開発し動かす段階に入っています。本章では、そんな状況を鑑み、プログラマ目線でのDockerを解説します。

Author
中井 悦司(なかい えつじ)
グーグル㈱
Twitter
@enakai00

## コンテナSIGの誕生

皆さんは、「コンテナSIG」というコミュニティをご存じでしょうか? 2016年10月に開催されたイベント「Container SIG Meet-up 2016 Fall」の開催レポートでは、「Dockerを中心とするコンテナ技術・実装・業界動向などを総合的に共有するために設立されたコミュニティ」との説明があります[注1](**図1**)。Dockerそのもののコミュニティではなく、Dockerをとりまく情報の共有が目的というわけですが、これにはいったいどのような意味があるのでしょうか?

実はここには、「Dockerの利用方法の変化」が隠されています。Dockerが登場した当初は、プログラマが自分専用の開発・テスト環境を用意するためにDockerを利用するシーンが多かったように思われます。現代的なサーバアプリケーションの実行環境には、さまざまなライブラリやフレームワーク、データベースなどが必要となりますので、事前にこれらが適切に用意された環境をセットアップすることが、プログラムの開発をスタートする前提となります。一般に、このような前提環境を用意するのは、「インフラエンジニア」の役割でした。大規模なアプリケーション開発の世界では、専任のインフラエンジニアが開発環境、あるいは、テスト環境の構築やメンテナンスを担当することになります。金融機関のシステム統合プロジェクトの現場をご存じの方であれば、どのような世界か容易に想像がつくことでしょう。

**▼図1 コンテナSIGのロゴマーク**


しかしながら、小さなチームで開発するアプリケーションの場合、専任のインフラエンジニアを確保する、あるいは、開発・テスト用のサーバ環境を永続的に保持することが難しい場合もあります。このような場合、Dockerを用いて、必要なライブラリなどがインストールされた環境をコンテナイメージに固めておけば、間違いなく便利です。

Dockerが利用できる環境であれば、手元のMacBook、机の下で動いているワークステーション、あるいは、パブリッククラウド上の仮

注1)「Container SIG Meet-up 2016 Fall」レポート(http://knowledge.sakura.ad.jp/event-seminar/6192/)

想マシンなど、あらゆる場所で同一の開発環境を即座に再現することができます(**図2**)。チームメンバでコンテナイメージを共有することで、誰もが間違いなく同じ環境で、迅速に開発作業をスタートできるというわけです。

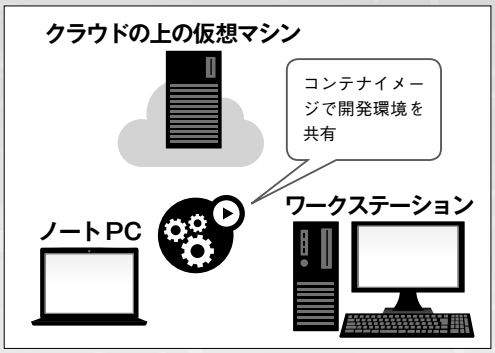
## サービス環境への利用の広がり

そのあと、開発・テスト環境を超えて、実サービス環境でのコンテナの利用を検討する企業が増えてきました。これまでは、開発環境、テスト環境、そして、実サービス環境のインフラは、それぞれ個別に構築されていました。そのため、これらの環境が違うことに起因する問題が発生することがあります。

アプリケーション開発の現場によっては、この手の問題は、日常茶飯事という話を聞くこともあります。開発者の手元にある開発用PCでは実行できるコードが、テスト環境ではなぜかテストに失敗する。テストに成功したはずのアプリケーションが、サービス環境では原因不明のエラーで停止する。この手のエラーの原因を調べるには、アプリケーションそのものだけではなく、OS、ライブラリ、データベースなど、あらゆる要因を探る必要があります。

とりわけ、アプリケーションをサービス環境にリリースしたあとに、OSやライブラリのバージョンアップが必要になった場合に、開発・テスト・サービス環境で個別のバージョンアップ

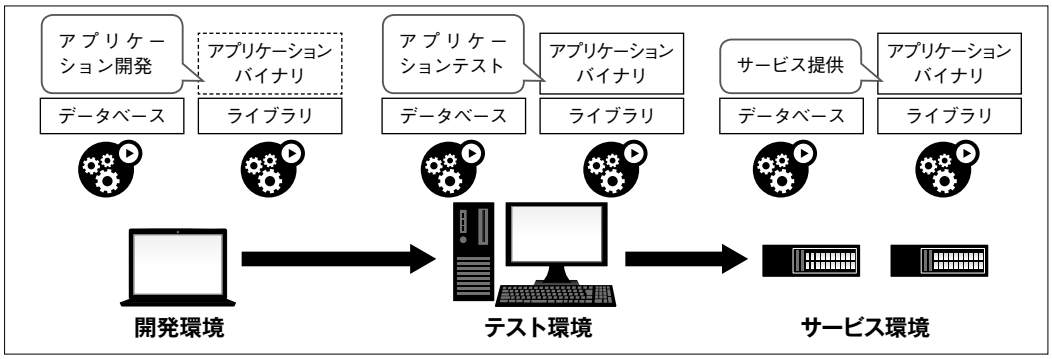
**▼図2　コンテナイメージ化した開発環境**




作業を行うと、すべての環境を同一に保つことはもちろん、それぞれの環境の差異を把握することすら困難になっていきます。そこで、開発環境だけではなく、テスト環境、サービス環境を含めた、すべての環境で同じコンテナイメージを利用して、アプリケーション実行環境の差異を完全になくしてしまおうというわけです(**図3**)。

従来、インフラエンジニアは、必要なサーバを用意してOSを導入したあと、アプリケーションの実行に必要なライブラリ、データベースなどのインストールを進めていくのが定番です。サービス環境においては、アプリケーション開発者が用意した手順書を見ながら、完成したアプリケーションを手作業でインストールすることもよくあります。このような手作業もまた、環境ごとの差異が生じる大きな要因となりえます。そこで、OS導入以降の作業をコンテナイ

**▼図3　コンテナイメージで開発・テスト・サービス環境を統一**



メージのデプロイに置き換えてしまいます。アプリケーションの実行に必要な環境が、すべてコンテナイメージとして用意されていれば、dockerコマンドでコンテナイメージをデプロイするだけで作業完了です。

アプリケーションをコンテナ化した場合、ログの収集やアプリケーションの監視方法など、従来と運用手順が変わることを心配する人もいるようですが、これは、それほど大きな問題ではありません。「1台のサーバ(もしくは、仮想マシン)に1つのアプリケーション(すなわち、1つのコンテナ)」という構成を変更しなければ、基本的には、従来と同じ運用方法を適用することができます[注2]。

## コンテナイメージは誰が作るもの?

コンテナイメージを用いたサービス環境を実現した場合、アプリケーション導入済みのコンテナイメージを作成するのは誰の役割になるのでしょうか? 理想を言うならば、アプリケーション開発者、すなわち、プログラマ自身ということになります。もともと、プログラマが自分で必要な環境を用意できるという手軽さが、Dockerが人気を集めるようになった理由です。従来のサーバ環境のように、「インストール担当者」が手順書を読みながらコンテナイメージを作成するのでは本末転倒でしょう。アプリケーション開発の世界では、継続的インテグレーション(CI:Continuous Integration)のように、アプリケーションのビルドとテストを自動化するしくみが発展していますので、CIの延長として、コンテナイメージの作成を自動化することもそれほど難しくはないはずです。

――というような話をしていると、「おっと。これはまた、インフラエンジニアは仕事がなくなる系の話か?!」と考える読者もいるかもしれません。この点については、最後に改めて振り返ることにします。

## 継続的デリバリとマイクロサービス

ここまで、「プログラマが必要な環境を自分で用意できる」「開発、テスト、サービス環境の差異を取り除く」という2つの観点で、Dockerが役に立つ理由を説明してきました。そして、近年、Dockerが注目を集めるもうひとつの理由が「継続的デリバリ(CD:Continuous Delivery)」の実現です。

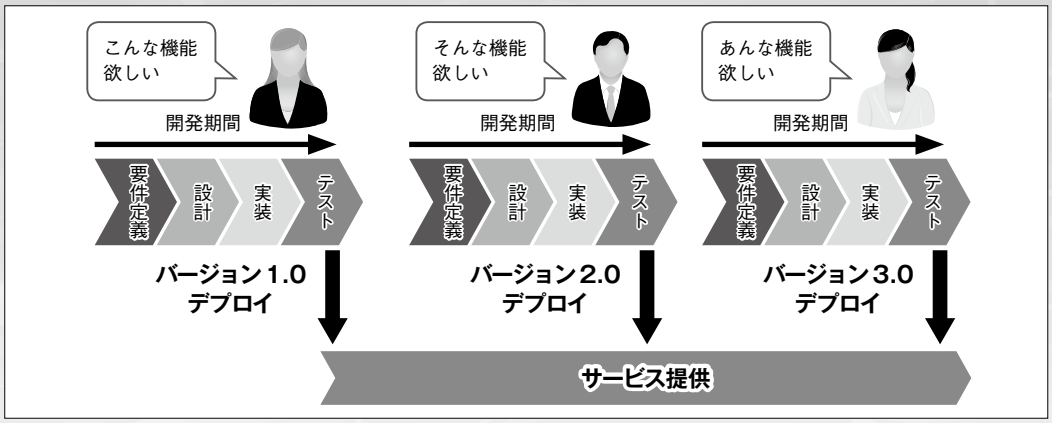
先ほど触れたCIの活用が広がる理由の1つは、開発期間の短縮です。従来のウォーターフォール型の開発では、要件定義からアプリケーションの完成までに長い期間がかかるため、アプリケーションが完成した時点では、すでにその機能は時代遅れになっているということもありえます。そこで、すべての機能を一度に実装するのではなく、自動化を利用して開発とテストのサイクルをスピードアップしながら、機能の追加・拡張を段階的に実施していきます(**図4**)。

ただし、この際、せっかく機能の追加・拡張を行っても、即座にサービス環境にデプロイできなければ意味がありません。アプリケーションの開発・テストをCIで自動化すると同時に、完成したアプリケーションのサービス環境への展開を自動化して、サービス環境における機能の追加と拡張を継続的に実施しようというのが、「継続的デリバリ」の考え方になります。

このとき、完成したアプリケーションをコンテナイメージに固めておけば、「サービス環境への展開を容易に自動化できるのでは?!」と期待が膨らみます。ただし、この点は、単純にDockerだけで解決できる問題ではありません。サービス環境のアプリケーションを頻繁に更新するには、サービスの停止時間をできるだけ短くする、あるいは、想定外の問題が発生した際にすみやかに元の状態に戻すなどのしくみづくりが必要となります。ここで登場するのが、Kubernetes

**注2)** アプリケーションをコンテナ化した際の運用設計については、次のWeb記事が参考になります。「Dockerによるコンテナ化アプリケーションの運用設計ガイド」(http://jp-redhat.com/openeye_online/column/nakai/3077/)

▼図4　継続的デリバリによるアプリケーションの機能拡張



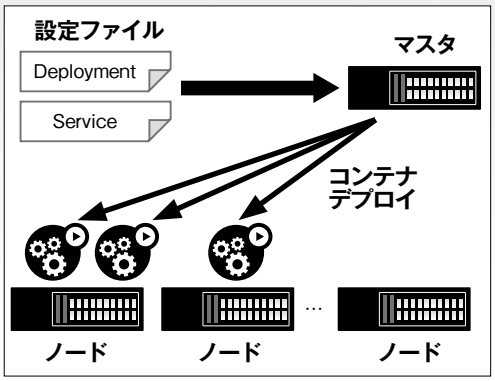

などのオーケストレーションツールが実現するコンテナインフラ環境、そして、「マイクロサービスアーキテクチャ」の考え方になります。

マイクロサービスアーキテクチャについて簡単に説明すると、アプリケーションを機能単位に分割して、それぞれを個別のコンテナで起動する手法になります。アプリケーションを機能拡張する際は、該当部分のコンテナのみを更新すれば良いので、変更の影響範囲を局所化して、事前の機能テストや問題発生時の切り戻しを容易にできます。すぐ後で説明するように、Kubernetesの機能を利用すれば、同一機能のコンテナを複数起動してスケールアウトしたり、ブルー・グリーンデプロイメントによって、サービスを停止せずに機能をアップデートすることも可能になります。

――と言っても、マイクロサービスアーキテクチャがどのようなものか、すぐにはイメージがわかないかもしれません。次節からは、具体的なアプリケーションの例を用いて、マイクロサービスの考え方を紹介したいと思います。


実は、冒頭で紹介したコンテナSIGのイベントには筆者も登壇しており、Kubernetesを利用したサンプルアプリケーションのデモンストレーションを実施しました。デモンストレーションの手順を筆者のブログで公開しているので、興味のある方は、実際に試してみると良いでしょう[注3]。ここでは、このアプリケーションの構成を理解する前提となる、Kubernetesのしくみについて解説を行います。ポイントとなるしくみは、「Deployment」と「Service」の2つです。

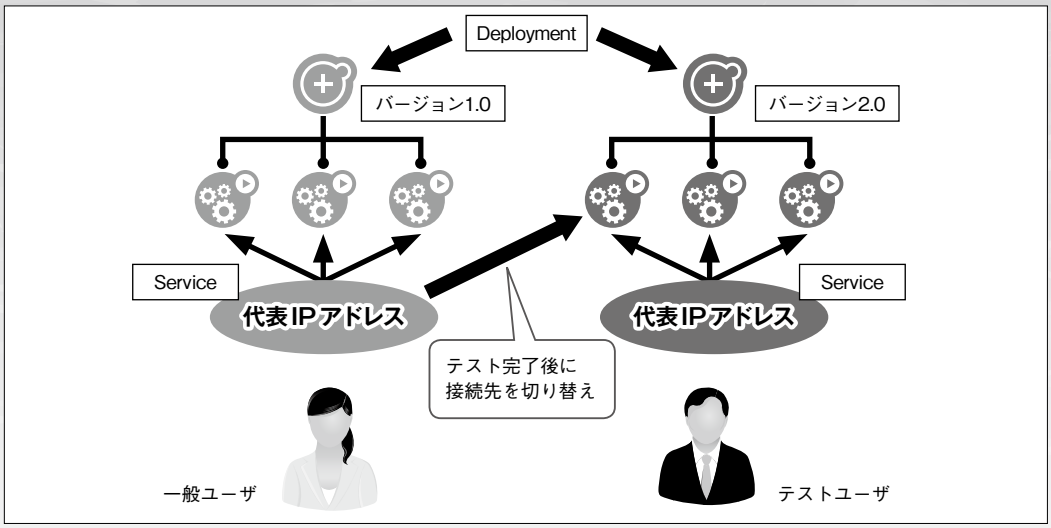
▼図5　Kubernetesによるコンテナクラスタ管理




まず、「Deployment」は、ある特定の機能を提供するコンテナをどのような構成でデプロイするかを指定するしくみです。Kubernetesでは、**図5**のように、複数のノード(コンテナホスト)を束ねて、1つのクラスタ、すなわち、巨大なリソースプールとして管理します。Kubernetes

注3)「Google Container Engineで五目並べアプリケーションのAPIサーバを作るデモ」(http://enakai00.hatenablog.com/entry/2016/08/10/152334)

▼図6　ブルー・グリーンデプロイメントのしくみ




のマスタにDeploymentの設定ファイルを与えると、リソースの空いているノードを見つけて、自動的にコンテナのデプロイを行います。この際、コンテナ数を指定することで、同一の機能を提供する複数のコンテナを起動できます。何らかの障害でコンテナ、もしくは、ノードが停止した際は、健全な状態のノード上で、自動的にコンテナを再起動してくれます。つまり、ノード障害に対する高可用性クラスタの機能が、最初から組み込まれているわけです。

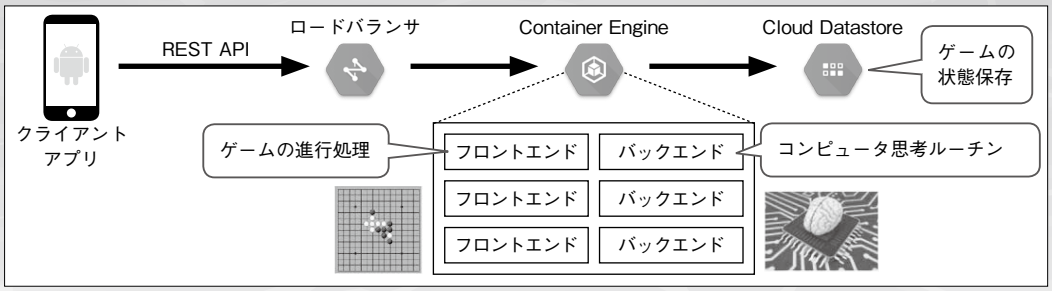
次に、「Service」は、コンテナに対するネットワークを構成するしくみです。前述のように、同じ機能を提供するコンテナが複数起動するので、代表IPアドレスを用意して、アクセスを振り分けるロードバランサの機能が必要になります。Serviceの設定ファイルでは、どのコンテナのどのポートに対して負荷分散するかを指定します。先ほどと同様に、Kubernetesのマスタに Serviceの設定ファイルを与えると、自動的にロードバランスの設定が行われます。

そして最後に、これらを組み合わせて実現するのがブルー・グリーンデプロイメントです。たとえば、Deploymentの設定を変更することで、**図6**のように、既存のコンテナと新しいバージョンのコンテナを並行して稼働できます。さらに、それぞれのコンテナに個別のServiceを適用して、外部ユーザからのアクセスは既存のコンテナに受け渡し、内部のテストユーザからのアクセスのみを新しいバージョンのコンテナで受け取ります。テストが完了したあとに、Serviceの設定を変更して、外部ユーザからのアクセスを新しいバージョンのコンテナに切り替えれば、コンテナのバージョンアップが完了するという寸法です。

## マイクロサービスによる五目並べアプリケーションの構成例

最後に、デモンストレーションで紹介したサンプルプリケーションの構成を説明しておきましょう。全体像は、**図7**のようになります。これは、五目並べアプリケーションのAPIサーバをKubernetesのクラスタ上で稼働するというもので、Google Cloud Platform(GCP)が提供するKubernetesのマネージドサービス環境(Google Container Engine)を利用しています。ゲームの進行を管理するフロントエンド機能と、コンピュータプレーヤの思考ルーチンを提供するバックエンド機能を別々のコンテナで起動しています。どちらもDeploymentのしくみを用いて、複数コンテナで負荷分散するようになっています。

▼図7　五目並べアプリケーションの全体構成




このとき、プレイ中のゲームの状態は、GCPが提供するデータストアサービス(Cloud Datastore)に保存するようにしてあります。ノードの障害停止に対する高可用性クラスタの機能があるとはいえ、障害に伴ってゲームの状態が失われては意味がありません。マイクロサービスアーキテクチャを実現する際は、失われると困るデータは外部に保存して、データを持たない部分(いわゆる「ステートレス」なサービス)をコンテナでスケールアウトするという考え方が大切です。

そして、この構成においては、フロントエンドの機能とバックエンドの機能を個別にバージョンアップすることができます。イベント当日は、エンドユーザが五目並べをプレイしている最中に、こっそりとバックエンドをバージョンアップするというデモンストレーションを行いました。ゲームを中断することなくバージョンアップを行い、ゲームのプレイ中に、突然、コンピュータの腕前が上がって強くなる様子を見せて、オーディエンスを驚かせてみました。


本章では、「Dockerの利用方法の変化」という観点で、コンテナを利用するメリットを紹介しました。アプリケーション開発に必要な環境をプログラマが自分で用意できるという点から始まり、開発・テスト・サービス環境でのアプリケーション実行環境の差異がなくなるというメリット、そして、アプリケーションをマイクロサービス化して、機能単位でのライブアップデートを実現するというところまで話が広がりました。

とくに、最後に紹介したKubernetesの環境であれば、DeploymentやServiceなどの設定ファイルを用いてコンテナのデプロイを行います。このような作業はアプリケーション開発者、すなわち、プログラマ自身で行うことも可能です。どのサービスをどのような構成でデプロイするべきかなど、最適な構成の判断ができるのは、プログラマ自身にほかなりませんので、そちらの方がより理想に近い役割分担と言えるでしょう。

ただし、サンプルアプリケーションの例からもわかるように、現実のサービス環境では、さまざまな外部サービスと連携したしくみが必要となります。ネットワークや外部のデータストアなど、すべてのしくみを1人のプログラマが理解するのは困難な場合もあるでしょう。アプリケーション開発者と、サービスインフラの構成要素を深く理解したアーキテクトが協力して、アプリケーション全体の設計を実施する必要があります。

本文中では、「インフラエンジニアの仕事がなくなるのでは？！」という話題にも触れましたが、インフラの構築だけがインフラエンジニアの仕事ではありません。インフラの構成要素を深く理解して、最適なアプリケーションの設計に貢献すること、これこそが、これらからのインフラエンジニアに求められる役割ではないでしょうか。SD

# 第2章 すぐに使える!
# マネージドサービスでDockerを活用

**第2章と第3章では、Dockerを使ってアプリを本番の実行環境で稼働させるまでの流れを体験しながら、プログラマにとってのメリットを感じてください。まずはGoogle Cloud Platformを利用して、アプリのデプロイ環境を構築します。**


**Author**
阿佐 志保(あさ しほ)　TIS㈱
**Twitter** @_Dr_ASA
**General editor**
山田 祥寛(やまだ よしひろ)
WINGSプロジェクト

ここ数年、雑誌や書籍などでDockerが大きく注目されていたため、手元の開発環境や検証環境などでDockerを動かし、その便利さを享受されたプログラマの方も少なくないのではないでしょうか。しかしながらDockerは、本番環境で稼働させることで、さまざまなメリットが生まれます。とくに、ビジネス環境の変化や技術革新のスピードに追従できる継続的デリバリを実践するための「変化に強いシステム」を実現するためのプラットフォームとして大きな可能性を秘めています。本章では、Dockerがプログラマにとって、どのようなメリットをもたらすかを具体的に見ていきます。

## プログラマにとってのDocker

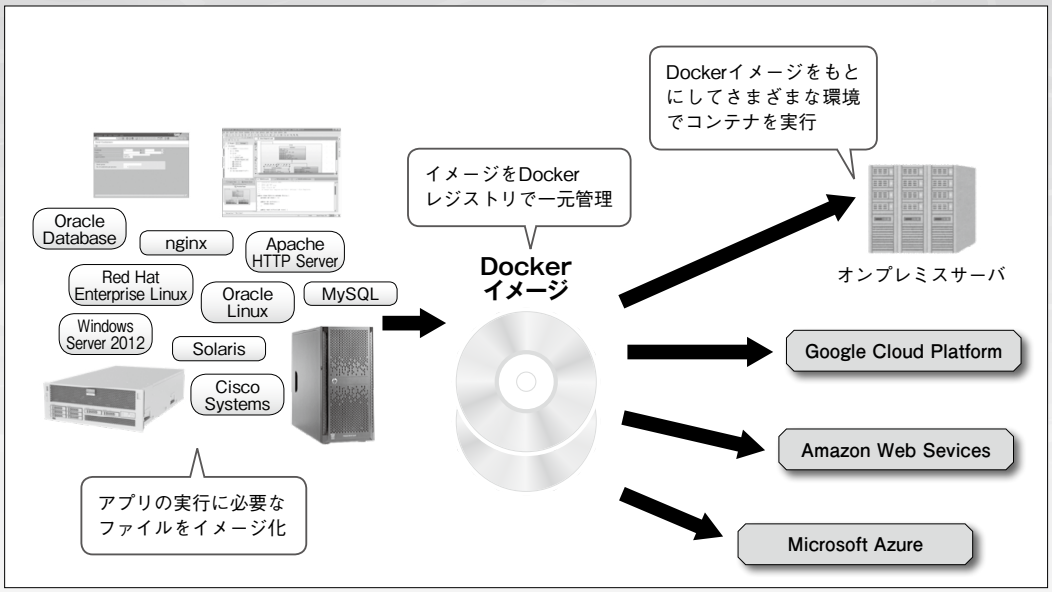
第1章での解説にあったとおり、Dockerは、アプリの実行に必要なものを1つのイメージにまとめて、任意の環境で稼働させるためのプラットフォームです。そのため、プログラマがテスト済みの開発／テスト環境で動作している**安全なアプリ**を、そのままの状態で本番環境で動かすことができます。そのため、多くの開発プロジェクトで頭を悩ます「こっちでは動くけど、あっちでは動かない」を減らすことができます。

しかしながら、実際に本番環境でDockerを使ったアプリを稼働させるという観点からみたとき、システムの可用性／拡張性をどう確保するか、永続データの管理をどうするかなどの検討が必要です。これらを検討し、最適な基盤を自前で構築／運用するためには、どうしてもインフラに関する高度な知識と経験が必要です。

一方、近年パブリッククラウドは勢いを増し、極めて早いスピードでさまざまな機能やサービスを次々に提供しています。AWS/Microsoft Azure/Google Cloud Platformといったメジャーなクラウドサービスは、いずれも国内にリージョンを持ち、先進的な企業やスモールスタートが求められるスタートアップ企業だけでなく、エンタープライズシステムへの導入も本格的に進んでいます。パブリッククラウドを利用するメリットは、コスト削減や高いスケーラビリティの確保など数えきれないほどありますが、なんといってもシステム構築にかかる時間を大幅に短縮できるため、常に変化するビジネス環境の変化や技術革新のスピードに追従しやすいという点で大きく注目されています。

現在、パブリッククラウド各社はいずれも、コンテナの実行基盤を構築するためのサービスを豊富に提供しています。そこで、先ほど挙げたDockerを使ったアプリ実行基盤をパブリッククラウド上で動作させることで、Dockerの持つ利便性と、パブリッククラウドの強みを組み合わせることができ、プログラマは次のようなメ

▼図1　Docker概要




リットを享受できます。

・テスト済みの安全なアプリを本番環境に即座にデプロイ
・障害時にも即時切り戻し
・プログラマ自身でDocker実行環境の構築／運用の実現
・機能単位での短い開発サイクルによる継続的デリバリ
・提供サービスの無停止アップデート

それでは、これらのメリットを体感するため、サンプルアプリを使ってDockerを使ったアプリを本番の実行環境で稼働させるまでの流れと、プログラマにとってのDockerのメリットを見ていきましょう。

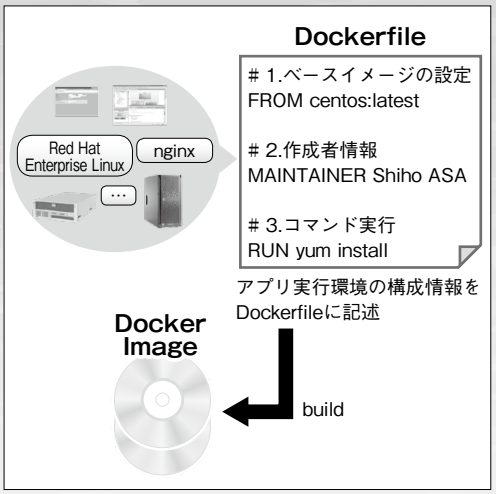
## Dockerの基本機能

まず、Dockerにはどのような機能があるかを簡単におさらいしておきます。Dockerでは、インフラ環境設定ファイルやアプリの実行モジュール／ライブラリなどをまとめてDockerイメージを作成し、イメージをもとにしてDockerコンテナをオンプレミスやクラウドなど任意の環境で実行できます。高い移植性を持ち、プログラマの手元にある開発環境や、テスト環境で動いたものが、そのまま本番環境で動作するのが特徴です。どのようなしくみでそれらを実現しているのでしょうか？　Dockerには、大きく分けて**図1**のような3つの基本機能があります。

### Dockerイメージ作成機能(Build)

Dockerは、アプリの実行に必要になるプログラム本体／ライブラリ／ミドルウェアや、OSやネットワークの設定などを1つにまとめてDockerイメージを作ります。出来上がったDockerイメージは、実行環境で動くコンテナのもとになります(**図2**)。Dockerイメージの実体は、アプリの実行に必要なファイル群が格納されたディレクトリツリーです。テキストファイル(Dockerfile)にDockerイメージの構成情報を記載して、これをもとにイメージをビルド(自動作成)することが可能です。Dockerでは1つのアプリを1つのコンテナに入れ、複数のコンテナを組み合わせてサービスを構築する、というシンプルな構成にすることが推奨されています。

▼図2　Dockerイメージの作成機能

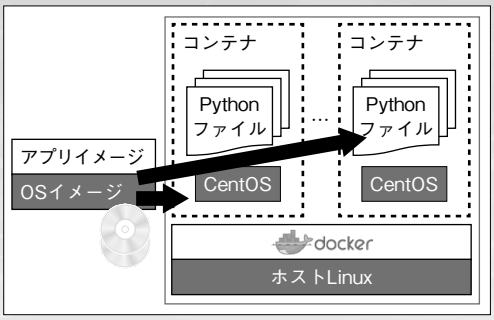



## Dockerコンテナ実行機能（Run）

Dockerは、Linux上で、コンテナ単位でサーバ機能を動かします。このコンテナのもとになるのが、Dockerイメージです（**図3**）。Dockerイメージさえあれば、どこでもコンテナを動かすことができます。1つのDockerイメージから複数のコンテナを起動できますので、高いスケーラビリティを要求されるシステムでは、リクエストの多寡（たか）に応じて必要な数のコンテナを実行できます。Dockerはオーバーヘッドが少なく、すでに動作しているOS上でプロセスを実行するのとほぼ同じ速さで起動するのが特徴です。

Dockerは1つのLinuxカーネルを複数のコンテナで共有しています。コンテナ内で動作するプロセスを1つのグループとして管理し、グループごとにそれぞれファイルシステムやホスト名／ネットワークなどを割り当てています。グループが異なればプロセスやファイルへのアクセスができません。このしくみを使って、コンテナを独立した空間として管理しています。これらを実現するために、Linuxカーネル機能（name space/cgroupsなど）の複数の技術が使われています（第6章を参照）。興味がある方は、参考書籍を一読されるとより理解が深まります。

▼図3　Dockerイメージの実行機能




## Dockerイメージ共有機能（Ship）

DockerのイメージはDockerレジストリで一元管理します。公式のDockerレジストリであるDocker Hubでは、UbuntuやCentOSなどのLinuxディストリビューションの基本機能を提供するベースイメージが配布されています。また、公式のイメージ以外にも、個人が作成したイメージをDocker Hubで公開／共有することができます。

ただし、Docker Hubはインターネット上に公開されたパブリックなレジストリであるため、機密情報が含まれる場合、セキュアな環境にDockerレジストリを別途用意する必要があります。業務系システムであれば、開発メンバーのみがアクセスできるセキュアな環境内で、Dockerレジストリを運用することが必須でしょう。

## クラウドが提供するマネージドサービスの活用

Dockerをマルチホスト環境からなるクラスタ構成で稼働させるためには、コンテナの起動／停止などの操作だけでなく、ホスト間のネットワーク接続やストレージの管理、コンテナをどのホストで稼働させるかなどのスケジューリング機能が必要になります。さらに、コンテナが正常動作しているかどうかを監視するしくみも必要でしょう。これらの機能を備え、コンテナを統合管理できるツールを**コンテナオーケストレーションツール**と呼びます。

オーケストレーションツールの利用環境を適切に設計／構築するには、システム全体の信頼性／可用性／拡張性、さらに運用方式などの知識や経験が必要なため、アプリを設計／開発することが本業であるプログラマにとっては、ハードルが高いのではないでしょうか。そこで、クラウド各社が提供するコンテナ実行環境構築のフルマネージドサービスをうまく活用すれば、手軽にコンテナの実行環境が用意できます。代表的なサービスは次のとおりです。

### Amazon EC2 Container Service

Amazon Web Servicesの仮想マシンサービスAmazon EC2のインスタンスでクラスタを構成し、Dockerの実行環境を提供するフルマネージドサービスです。AWSは数多くのサービスと豊富なドキュメント、活発なコミュニティを持ち利用者も多いため、中小規模から大規模システムまで安心して導入できるプラットフォームといえるでしょう。

### Azure Container Service

MicrosoftのクラウドサービスであるAzureのマネージドサービスです。オーケストレーションツールとしてMesosベースのDC/OSかKubernetesかDocker Swarm/Docker Composeを選べるのが特徴です。国内に2つのリージョンを持ち、これまで同社が培ってきた充実したエンタープライズ向けサポートもあるため、既存の業務系システムとの親和性は高いでしょう。

### Google Container Engine

Googleが提供するパブリッククラウドサービスの1つで、オープンソースのオーケストレーションツールであるKubernetesをベースにしたフルマネージドサービスです。Googleが提供している検索エンジン／YouTube／Google Mapsなどのサービスはすべてコンテナで稼働しています。

◆ ◆ ◆

どのサービスを選ぶべきかはシステム要件によりますが、前述したとおり、開発したアプリの移植性が高いのがDockerの大きな特徴です。特定の環境にロックインされることが少ないため、将来的にやむを得ない理由でオンプレミス上で稼働させなければならない場合でも、マイグレーションが比較的容易です。したがって、各サービスの提供機能やサポート体制などの比較検討に時間と労力をかけるより、まずはプログラマ自身が、手を動かして試してみることをお勧めします。オンプレミスの世界では、インフラ構築といえば、ネットワーク敷設や機器調達から始まり……という手戻りが難しい工程が多いため、どうしても机上での比較検討が欠かせませんが、クラウドの世界では、インスタンスを生成しては破棄してを繰り返す、ということが大前提です。

本記事では、現時点でコンテナの開発環境が充実していてDocker導入の敷居が低く、かつ本番環境で大規模システムを導入するのに最も適したフルマネージドサービスであると筆者が考えている、Google Container Engineを使って環境を構築します。


Google Cloud Platform(以降、GCP)は、Googleが自社で使っているインフラを一般でも利用できるようにしたクラウドサービスです。コンテナの実行環境を提供するGoogle Container Engine(以降、GKE)は、Kubernetesをベースにしたフルマネージドサービスです。言い換えれば、専門的な知識が必要なKubernetesの環境を自動で構築してくれるサービスだ、と理解すればよいでしょう。操作はすべてGoogle Cloud Consoleと呼ばれているWebベースのGUIから行います。

GCPを使うにあたり、お使いのGoogleアカウントでログインしてアカウントを登録します。トライアルで60日／300ドルまで無償で利用で

▼図4　プロジェクトの作成


▼図5　アクセス権の設定


きます。また、課金の設定が必要になりますので、次の公式サイトでアカウント登録を行い、クレジットカードの登録を行ってください。

なお、明示的に有償アカウントにアップグレードしない限り、課金されることはありません。

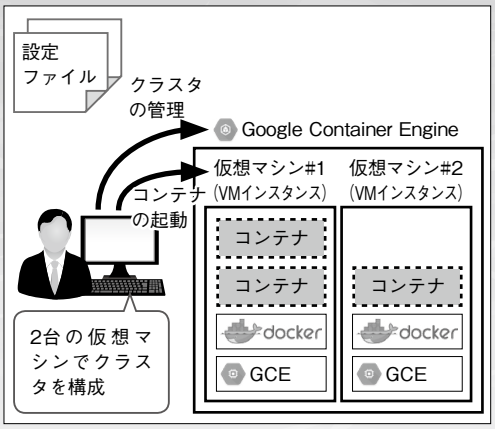
https://cloud.google.com/free-trial/

登録が完了すると、Webベースの管理コンソールにアクセスできます。これはGoogle Cloud Consoleと呼ばれていて、多くの管理作業をGUIで行うことができます。

## プロジェクト作成とアクセス設定

GCPを利用するには、まず最初に「プロジェクト」を作成する必要があります。GCPでは、このプロジェクト単位で、各サービスの利用やアクセス権／支払い情報などをまとめることができます。新しいプロジェクトを作成するには、Google Cloud Consoleの[ホーム]－[ダッシュボード]から[プロジェクト作成]ボタンをクリックします。ダイアログが表示されますので、任意のプロジェクト名を入力します(**図4**)。プロジェクト名を入力すると、プロジェクトIDが付与されます。このプロジェクトIDは、以降の作業で利用しますので、忘れないように控えておいてください。

▼図6　Dockerイメージの実行機能




アカウントを登録した直後のデフォルトの状態では、GKEを利用できません。そこでまず、GKEを利用するためのアクセス権を設定します。Google Cloud Consoleの[API Manager]－[ライブラリ]を選択し、「Google Container Engine API」を検索します。[有効にする]ボタンをクリックして、アクセスを許可します(**図5**)。

以上で、GCPを利用する準備が完了しました。

## GKEでのコンテナ実行環境構築

つづいて、Google Cloud Consoleからコンテナの実行環境を構築します。今回は**図6**の構成となるよう、2台の仮想マシン(ノード数)を使ってクラスタを構築します。

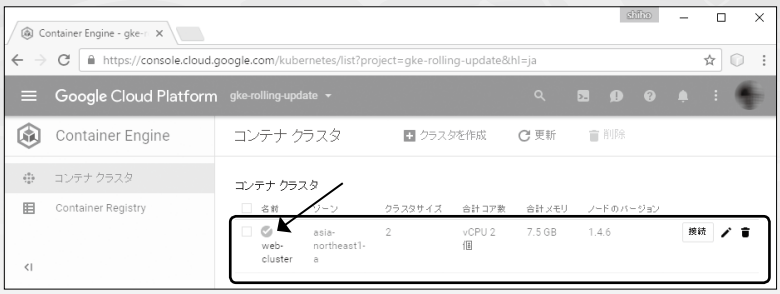
Cloud Consoleメニューの[コンピュート]－[Container Engine]を選択し、[コンテナクラスタを作成]ボタンをクリックします。今回はクラスタの仮想マシンの数(ノード数)を2とするため、**表1**の値を設定します。指定以外の個所はすべてデフォルトとします。

クラスタが構成できているかどうかは、

**▼表1　クラスタの設定**

| 項目 | 説明 | 今回の設定値 |
|---|---|---|
| 名前 | Kubernetesで管理するクラスタの名前 | web-cluster |
| ゾーン | 使用可能なコンピューティングリソースとデータの保存場所と使用場所を指定。ここでは東京リージョンを指定 | asia-northeast1-a |
| マシンタイプ | クラスタのノードに使用するマシンのスペックを選択 | vCPU × 1 |
| サイズ | クラスタ内のノード数。クラスタサイズは、使用可能なCompute Engineの割り当て量によって制限される | 2 |

**▼図7　クラスタの構築**




[Container Engine]－[コンテナクラスタ]を確認します。作成した「web-cluster」に緑色のチェックがついていれば、問題なく稼働しています(**図7**)。

驚いた方もいるかもしれませんが、わずか数分でクラスタ化されたコンテナ実行環境が構築できました。次章では、このコンテナ実行環境を使って、開発したアプリをデプロイ／アップデートを繰り返す、継続的デリバリの流れを見ていきましょう。SD

## Column Google Cloud Platform(GCP)の生い立ち

GCPは、Googleが提供するパブリッククラウドサービスです。本章でご紹介したコンテナマネージドサービスであるContainer Engine以外にも、Python/Go言語/Javaなどをサポートする PaaSサービスであるAppEngineや、大規模データセットを扱うBigQuery、翻訳や画像解析など機械学習に関するAPIなどが提供されています。

GCPの特徴は、なんといっても自社サービスのために開発したシステム基盤をもとにしているというところにあります。Googleが提供する検索エンジン／YouTube／Google Mapsなどは、地球上のすべての人々がユーザです。そのため、世界中のデータセンターが同じしくみで構成されており、ストレージやデータベースなど、アプリの実行に必要な基盤は、独自のソフトウェアで実装されています。Googleのアプリ開発者は、実装／テスト／デプロイまでを標準化されたプロセスに則って行います。まさに世界規模での稼働実績があるといっても過言ではありません。

現時点では、大規模案件の導入実績や日本語ドキュメントが少ないという観点から、どうしてもAWSが注目されることが多くなっています。しかしながら、GCPの生い立ちを紐解いてみると、全社共通基盤を作成して、その上で自社フレームワークを採用し、アプリ開発者から基盤レイヤーをできるかぎり抽象化して、品質の安定したアプリを実装するという、今日の多くのエンタープライズシステムが採用している開発手法に極めてよく似ていることがわかります。GCPがサービスとして提供する「変化に強いインフラ」を上手に活用することで、エンタープライズシステムは、より一層顧客のビジネス目的実現のためのアプリ開発に注力できるのではないかと考えています。

# 第3章 使ってみよう!
# アプリ開発のストーリーからメリットを知る


第3章では、アプリ開発のストーリーを想定し、それぞれの段階でどのようにDockerが使われていくかを見ていきます。第2章で用意したクラウド上に、実際に手を動かしながら4つのステップを追って確認してみてください。

**Author**
阿佐 志保(あさ しほ)　TIS㈱
**Twitter** @_Dr_ASA
**General editor**
山田 祥寛(やまだ よしひろ)
WINGSプロジェクト

第2章では、Dockerのもつ利便性と、パブリッククラウドが提供するコンテナのフルマネージドサービスの強みを組み合わせれば、プログラマがコンテナを使ってアプリ開発を行ううえで、さまざなメリットがあることを説明しました。第3章では、実際に、構築した基盤を使って、サンプルアプリを開発／デプロイしていく流れに沿って、Dockerがもたらすメリットを確認していきます。

## サンプルから学ぶDockerを使ったアプリ開発の流れ

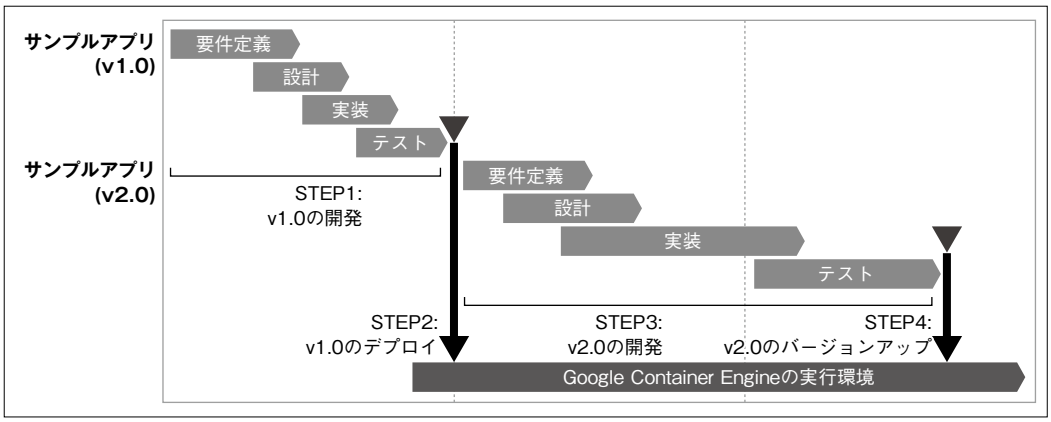
Dockerを使ったアプリ開発では、インフラの構成情報も含めたアプリケーションをパッケージングできるため、プログラマとインフラエンジニアで分担していたタスクの大部分を、プログラマだけで実施できるようになります。

アプリで使うライブラリ群やインフラの構成管理だけでなくデプロイの作業も自動化できるため、開発からリリースまでの期間を短くでき、小さな機能ごとに分割した継続的デリバリが可能になります。今回は、サンプルアプリを使って、**図1**の流れでアプリ開発と本番環境へのデプロイを行います。

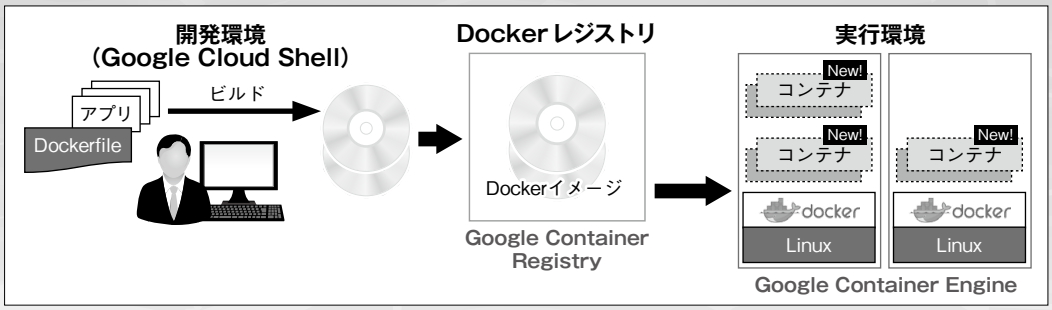
### STEP1. アプリ(v1.0)の開発

まず、サンプルアプリのv1.0を開発します。アプリの実行に必要な環境の設定とアプリの実

**▼図1　サンプルアプリ開発の流れ**



▼図2　開発環境／実行環境の構成




行モジュールをDockerfileに定義します。Dockerは、Dockerfileをビルドして Dockerイメージを作成します。開発環境でビルドしたDockerイメージを、第2章で構築したGoogle Container Engine(以降、GKE)の実行環境で利用できるよう、プライベートレジストリで共有します。このイメージ共有には、Google Cloud Platform(以降、GCP)が提供するDockerイメージ共有サービスの、Google Container Registryを使います。

### STEP2. アプリ(v1.0)のデプロイ

アプリ開発が完了したら、GKEの実行環境にデプロイします。どのイメージをどの環境にデプロイするか、などの構成情報はすべてyaml形式の定義ファイルで指定します。デプロイが完了したら、システム利用者がアプリにアクセスができるようサービスを作成します。サービスでは、アプリへアクセスするときに必要なネットワークの設定などを定義します。

### STEP3. アプリ(v2.0)の開発

変更個所を追加開発して、イメージを新しく作成します。イメージの作成手順はSTEP1と同じ流れになります。ただし、イメージにバージョンを区別できるようタグを設定します。今回は、「v1.0」のタグを「v2.0」にしたイメージを作成します。

### STEP4. アプリ(v2.0)の無停止バージョンアップ

すでにGKE上で稼働しているサービスを停止することなく、アプリをバージョンアップします。バージョンアップは、GKEのベースになっているコンテナオーケストレーションツールKubernetesのRolling Update機能を使います。この機能を使うと、クラスタ上で稼働しているコンテナのアプリを段階的にバージョンアップさせることができます。そのため、システム利用者はダウンタイムなしでアクセスできます。

▼図3　Cloud Shellの起動


◆ ◆ ◆

本章で扱う全体の構成は、**図2**のようになります。それでは具体的に、手順を追っていきましょう。


はじめに、サンプルのアプリを実行するためのDockerfileを作成します。

#### Dockerfileの作成

Dockerfileは Cloud Shellで作成します。Cloud Shellを起動するには、Webコンソール上で、右上のアイコンをクリックします(**図3**)。

ここで使用するDockerfileのサンプルをGitHubで公開しています。Cloud Shellで**図4**のコマンドを実行して、サンプルファイルをダ

▼図4　サンプルの取得

```
$ git clone https://github.com/asashiho/gke-rolling-update-sample.git
```

▼図5　FROM命令の設定

```
FROM nginx
```

▼図6　MAINTAINER命令の設定

```
MAINTAINER Your Name your_name@your_domain.com
```

ウンロードしてください。この後は、サンプルのDockerfileに含まれる命令を順番に説明していきます。

### [1]ベースイメージの指定

Dockerfileでは、Dockerコンテナをどの Dockerイメージから生成するかの情報が必要です。このもとになるイメージをベースイメージと呼び、FROM命令で指定します。Dockerfileでは、FROM命令は必須項目になります。また、タグ名を省略したときは、ベースイメージで指定したものの最新版(latest)が適用されます。ここでは、nginxが動作するベースイメージを、**図5**のように指定します。

### [2]作成者情報の設定

Dockerfileの作成者の情報を記述するときは、MAINTAINER命令を使います。ここでは、Dockerfileを作成した人のメールアドレスを記述します(**図6**)。

### [3]コマンドを実行

FROM命令で指定したベースイメージに対して、「アプリケーション／ミドルウェアをインストール／設定する」「環境構築のためコマンドを実行する」など、なんらかのコマンドを実行するときには、RUN命令を使います。ここでは、コンテナのログを転送するため、**図7**のように設定します。なお、命令を改行するときは「\」で区切ります。

▼図7　RUN命令の設定

```
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
  && ln -sf /dev/stderr /var/log/nginx/error.log
```

▼図8　EXPOSE命令の設定

```
EXPOSE 80
```

▼図9　ADD命令の設定

```
ADD html/ /usr/share/nginx/html/
```

▼図10　CMD命令の設定

```
CMD ["nginx", "-g", "daemon off;"]
```

### [4]ポートの開放

コンテナの公開ポート番号を指定するときは、EXPOSE命令を使います。ここでは、HTTPでの通信(80番ポート)を受け付けるため、**図8**のように指定します。

### [5]アプリのデプロイ

イメージにホスト上のファイルやディレクトリを追加するときは、ADD命令を使います。ここでは、Cloud Shellのhtmlディレクトリにあるファイルを、コンテナ内の/usr/share/nginx/html/にデプロイするため、**図9**のように指定します。

### [6]サーバの起動

これで環境構築は完了しましたので、最後にWebサーバを起動します。起動はCMD命令を使います。Dockerfileには、CMD命令は1つだけ指定してください。もし複数指定したときは、最後のコマンドが有効になります。**図10**は、nginxを実行するという意味になります。

▼図11　docker buildコマンドの実行例

```
$ cd gke-rolling-update-sample/v1.0
$ docker build -t sampleapl:v1.0 .
```

▼図12　docker imagesコマンドでの確認

```
$ docker images
REPOSITORY  TAG     IMAGE ID      CREATED          VIRTUAL SIZE
sampleapl   v1.0    d8f98741caec  20 seconds ago   181.6 MB
nginx       latest  ef068b58f0cd  8 days ago       181.5 MB
```

▼図13　Dockerイメージへのタグ設定コマンド

```
$ docker tag sampleapl:v1.0 gcr.io/<ご自身のPROJECT_ID>/sampleapl:v1.0
```

▼図14　Google Container Registryへのアップロード

```
$ gcloud docker -- push gcr.io/<ご自身のPROJECT_ID>/sampleapl:v1.0
```

Dockerfileの命令はここで取り上げた以外にも、環境変数を設定する／ボリュームのマウント／作業ディレクトリの指定などを行う命令があります。バージョンアップで仕様が変更になることもありますので、詳細についてはDocker公式サイト[注1]を参照してください。

## Dockerイメージのビルト

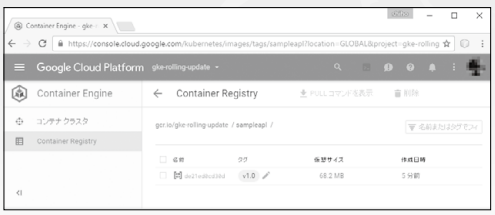
Dockerfileからイメージを作成するには、docker buildコマンドを使います。サンプルのv1.0ディレクトリ配下に格納されたDockerfileをもとにして、「sampleapl」という名前のイメージを作成するときは、**図11**のコマンドを実行します。なお、イメージのタグ名に「v1.0」というバージョンをつけておきます。

Dockerイメージは、docker imagesコマンドで確認できます。**図12**のコマンドを実行すると、作成した「sampleapl」いう名前のイメージが作成できているのがわかります。また、ベースイメージである、「nginx」のイメージも作成されています。

## Dockerイメージの公開

これで、Dockerコンテナを動かすもとになるDockerイメージが作成できましたので、次はGoogle Container Registryにイメージをアップロードします。アップロードするには、「gcr.io/<PROJECT ID>/名前:タグ」という命名規則でイメージにタグを設定します(**図13**)。プロジェクトIDは、第2章でプロジェクトを作成したときに付与される一意のIDです。プロジェクト名とは異なる場合もありますので、注意してください

▼図15　Dockerイメージのアップロード確認




次に、Dockerイメージを、Google Container Registryにアップロードします。gcloudコマンドを使用することで、Cloud Shellを起動したアカウントの権限でGoogle Container Registryへのアクセスが行われます(**図14**)。

イメージのアップロードが完了すると、Google Cloud Consoleの[Container Engine]－[Container Registry]にアップロードされたイメージが格納されていることが確認できます(**図15**)。

これで、「sampleapl」という名前のアプリのv1.0が稼働できるDockerイメージが、GKEから利用できる状態になりました。今回はGCPで動かすためDockerイメージを、Google Container Registryで共有しましたが、DockerはDocker

注1）URL https://docs.docker.com/engine/reference/builder/

▼図16　環境設定

```
$ gcloud container clusters get-credentials web-cluster --zone=asia-northeast1-a

Fetching cluster endpoint and auth data.
kubeconfig entry generated for web-cluster.
```

▼図17　コンテナのデプロイ

```
$ kubectl create -f deployment.yaml
deployment "web-container" created
```

fileさえあれば、Dockerイメージをビルドできます。たとえばDocker公式のリポジトリサービスであるDocker Hubであれば、任意のDocker実行環境で動作させることができますし、他クラウド上のリポジトリやオンプレミス環境で構築したプライベートレジストリなどでも同様に運用できます。この移植性の高さがDockerの持つ大きなメリットです。

## STEP2 アプリ（v1.0）のデプロイ

次に、第2章で作成したコンテナの実行環境にアプリケーションをデプロイします。

### GKEを操作するための設定

まず、Cloud Shell上で環境設定を行います。**図16**のコンテナを実行することで、東京リージョンである「asia-northeast1-a」に作成した、「web-cluster」という名前のGKEのクラスタにアクセスするために必要な認証情報が設定されます。

### デプロイ構成情報ファイルの作成

アプリをデプロイするために、構成情報ファイルを作成します。ファイルはサンプルのdeployment.yamlになります（**リスト1**）。起動したいコンテナの数はreplicasに指定します。ここでは3を指定していますので、第2章で構築した2台の仮想マシン上のいずれかで3つのコンテナが起動します。なお、コンテナイメージ名に含まれる「<PROJECT ID>」の部分を実際に使用するプロジェクトIDに書き換えて、ファイルを修正してください。

ファイルが作成できたら、kubectl createコマンドを使ってコンテナをデプロイします（**図17**）。デプロイ構成情報ファイル（deployment.yaml）は、コマンドのfオプションで指定します。

### サービス定義ファイルの作成

次に、サービス定義を作成します。**リスト2**は、起動したコンテナに外部からアクセスするために必要なネットワーク環境を定義するためのファイルです。

定義ファイルの準備ができたら、**図18**のkubectl createコマンドを使ってサービスを定

▼リスト1　デプロイ構成情報ファイル（deployment.yamlの抜粋）

```
apiVersion: extensions/v1beta1
kind: Deployment  ←デプロイ構成情報ファイルの指定
...略...
spec:
  replicas: 3  ←レプリカ数
  ...略...
    spec:
      containers:
      - image: gcr.io/<ご自身のPROJECT ID>/sampleapl:v1.0  ←コンテナのもとになるDockerイメージ
        name: web-container
        ports:
        - containerPort: 80  ←コンテナに外部からアクセスできるポート
```

▼図18　サービスの定義と確認

```
$ kubectl create -f service.yaml

$ kubectl get services
NAME          CLUSTER-IP     EXTERNAL-IP       PORT(S)   AGE
kubernetes    10.59.240.1    <none>            443/TCP   9m
web-service   10.59.240.35   104.198.118.141   80/TCP    1m
```

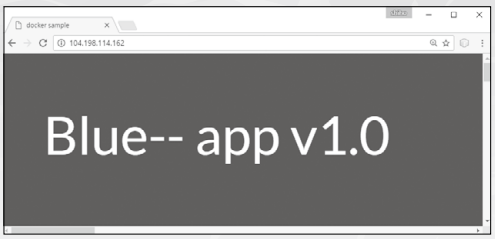
▼リスト2　サービス定義ファイル
（service.yamlの抜粋）

```
apiVersion: v1
kind: Service  ←サービス定義ファイルの指定
 ...略...
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80  ←受付ポート
  selector:
    name: web-container
  type: LoadBalancer  ←ロードバランサの設定
```

義します。このコマンドの実行で、外部からコンテナに対してアクセスが可能になります。サービスの情報を確認するには、kubectl getコマンドを実行します。コマンド実行結果の「EXTERNAL-IP」が外部からアクセスするときのIPアドレスになります。また、外部アクセス用のIPアドレスは自動的に割り当てるのではなく、事前に確保しておいたIPを明示的に指定することも可能です。

ここまでの準備ができたら、動作確認のため、ブラウザから[EXTERNAL-IP]にアクセスします。**図19**のような画面が正しく表示されたら、v1.0のアプリのデプロイに成功しています。

▼図19　サンプル（v1.0）の動作確認




▼図20　バージョン情報の設定

```
$ docker build -t sampleapl:v2.0 .
```


無事、v1.0のアプリがリリースできましたので、引き続いてv2.0の開発を行います。Dockerfileの作成およびビルド／公開はv1.0の手順と同じですので、詳細な手順は省略しますが、イメージを作成する際に、タグ名として「v2.0」を指定する点に注意してください（**図20**）。

イメージが作成できたら、Google Cloud Consoleの[Container Engine]－[Container Registry]に、アップロードされたイメージが格納されていることを確認できます。v1.0とv2.0のイメー

**Note**

GKEでは、デプロイの構成情報や外部からアクセスするときの接続情報をすべて定義ファイルで管理します。定義ファイルはソースコードとして管理できるので、アプリ開発で行われているGitなどのバージョン管理のしくみを利用すれば、容易にバージョン管理が可能です。オンプレミス環境でありがちな、運用手順書を作成して、手作業で環境を構築／運用し、構成変更があったときは差分情報を修正して……ということなく、アプリの実装コードとまったく同じように、インフラの構成情報をソースコードとして管理できることは、プログラマにとっては大きなメリットになるでしょう。なお、インフラの構成情報をソースコードで管理するという概念は、Infrastructure as a Codeと呼ばれています。

▼図21　Dockerイメージのアップロード確認

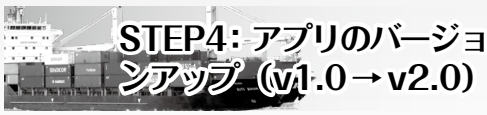

▼図22　デプロイ構成ファイルの修正

```
$ kubectl edit deployment/web-container
```

▼図23　ローリングアップデートの確認

```
$ kubectl describe deployment/web-container
```

ジが2つアップロードされているのがわかります(**図21**)。



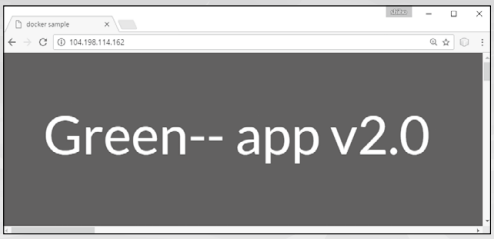
## STEP4：アプリのバージョンアップ（v1.0→v2.0）

v2.0のアプリの開発がおわり、Dockerイメージをレジストリにアップロードできたので、コンテナ実行環境で稼働しているv1.0のアプリを、Kubernetesの提供するRolling Update機能を使ってバージョンアップします。この機能を使うと、サービスを無停止の状態でバージョンアップできます。まず、**図22**のコマンドを実行して、デプロイメント設定ファイルを修正します。修正する個所は1ヵ所だけです。

デプロイメント構成ファイルがエディタで開くので、コンテナのイメージのタグ名を「v1.0」から「v2.0」に修正して保存します(**リスト3**)。

デプロイの履歴を確認するときは、kubectl describeコマンドを使います。**図23**のコマンドを実行すると、新しいイメージのPodを起動した後に、古いイメージのPodを停止するという操作が順次行われたことがわかります。

▼図24　サンプル(v2.0)の動作確認




なお、ここでは稼働環境の構成ファイルを直接に修正しましたが、事前に修正済みのファイルを用意して、それをkubectlコマンドで適用することも可能です。構成情報を設定ファイルで管理するという意味では、こちらのほうが望ましい運用方法と言えるでしょう。

以上の手順を終えたら、動作確認のため、ブラウザから再度[EXTERNAL-IP]にアクセスします。**図24**のような画面が表示されたらデプロイに成功しています。

サービスを停止させることなく、アプリをバージョンアップできるということがおわかりいただけたでしょうか？　このように、規模を小さくして開発とバージョンアップを繰り返すことで、継続的デリバリが実現できます。

## クラスタの停止と削除

動作を確認できたら、クラスタを停止／削除します(**図25**)。サービスを削除したタイミングで、

▼リスト3　デプロイ構成情報ファイル(deployment.yamlの抜粋)

```
apiVersion: extensions/v1beta1
kind: Deployment
...略...
    spec:
      containers:
      - image: gcr.io/<PROJECT ID>/sampleapl:v2.0  ←v1.0をv2.0に変更
...略...
```

▼図25　クラスタの停止と削除

```
$ kubectl delete service web-service
$ kubectl delete deployment web-container
```

Global Load Balancerの設定も破棄されます。

次に、Google Cloud Consoleで[Container Engine]－[コンテナクラスタ]で、コンテナクラスタの削除を行います。最後にDockerイメージを格納した[Container Engine]－[Container Registry]にある2つのバージョンのDockerイメージを削除します。


第2章、第3章では、プログラマにとって、Dockerは何の役に立つのか、という観点で、Google Cloud Platformを使ったコンテナ実行環境の構築を紹介しました。サンプルアプリの開発の流れを通して、Dockerのもつ移植性の高さに加えて、パブリッククラウドのもつ導入の容易さ／可用性や拡張性の高さなどがもたらす、さまざまなメリットを体験していただけたと思います。

継続的デリバリを実現した「変化に強いシステム」を導入することは、アプリのリリースまでの期間を短縮できるため、プログラマだけでなくシステムを利用するエンドユーザに大きなメリットが生まれます。本稿では、段階的にバージョンアップするRolling Updateの紹介にとどまりましたが、第1章で説明のあったv1.0とv2.0を並列に動作させておき、即座にアクセス先を切り替える、ブルー・グリーンデプロイメントも容易に実現できます。

なお、ここまで、プログラマにとってのDockerのメリットばかりを述べてきましたが、必ずしもメリットばかりではありません。残念ながら、既存システムの開発体制でアーキテクチャや開発手法がそのままの状態のものを、コンテナ基盤に「**移植**」してもうまくいかないケースがほとんどでしょう。

Dockerやクラウドに限らず、テクノロジーがもたらすメリットとデメリットを正しく知るには、「手を動かす」ことがなによりも重要です。つまり、テクノロジーの恩恵を受けるためには、エンジニア自身が環境を変え、新しいことを学び、変化し続けることが大切です。SD

## Column DockerとDevOps

DevOpsとは、プログラマ(Dev)とインフラエンジニア(Ops)がお互いに協力してシステムを開発／運用していこうという概念です。プログラマのミッションは、新しいビジネス価値の創造ですが、一方のインフラエンジニアのミッションは、システムの安定稼働です。相反する両者では、お互いの価値観が異なるため、時には意見が衝突することもあるでしょう。Dockerは、プログラマとインフラエンジニアの境界をあいまいにします。そのため、お互いの専門性を生かし、お互いを尊重／信頼しあって、より良いシステムにしていくことが望まれます。

なお、大規模な開発の場合、プログラマとインフラエンジニアの部門が異なることが多くなります。所属する組織が異なれば、どうしても意思決定に時間がかかるため、せっかく変化に強いシステムが完成したとしても、そのメリットを十分に生かせない恐れがあります。Dockerをベースにしたコンテナ実行環境を導入するには、技術的な側面だけでなく、組織改革や人材評価のしくみなどとあわせて、システム開発全体のしくみを熟考する必要があります。

第4章 用途ごとに分類

# Dockerイメージとコンテナを活用するコマンドの理解

Dockerの操作はdockerコマンドライン・ツールを使うのが一般的です。コマンドは多岐に渡りますが、本章ではイメージの管理、コンテナの実行、レジストリ操作、システム管理と用途で分類し、開発担当者が知っておくべき最低限のコマンドとオプションを紹介します。


Author
前佛 雅人(ぜんぶつ まさひと)
さくらインターネット㈱
Twitter
@zembutsu

## Dockerコマンドとライフサイクル

Dockerを使ったコンテナの実行、イメージの構築、デーモンの確認など、Dockerに関するさまざまな操作は、dockerコマンドを通して次のような書式で行います。

```
docker <コマンド> [オプション1] [オプション2] …
```

docker <**コマンド名**>と続くコマンドは、現在(Docker 1.12)約50個近く存在しています。コマンドごとにさまざまな機能(**表1**)がありますが、すべてを覚える必要はありません。それよりも、Dockerの基本である「イメージを使ってコンテナを実行し、使い終わったらコンテナを削除する」というライフサイクルの理解が大切です。とくにDockerイメージを構成している「イメージ・レイヤ(層)」に対する操作という概念が、コマンドを理解する手助けになります。

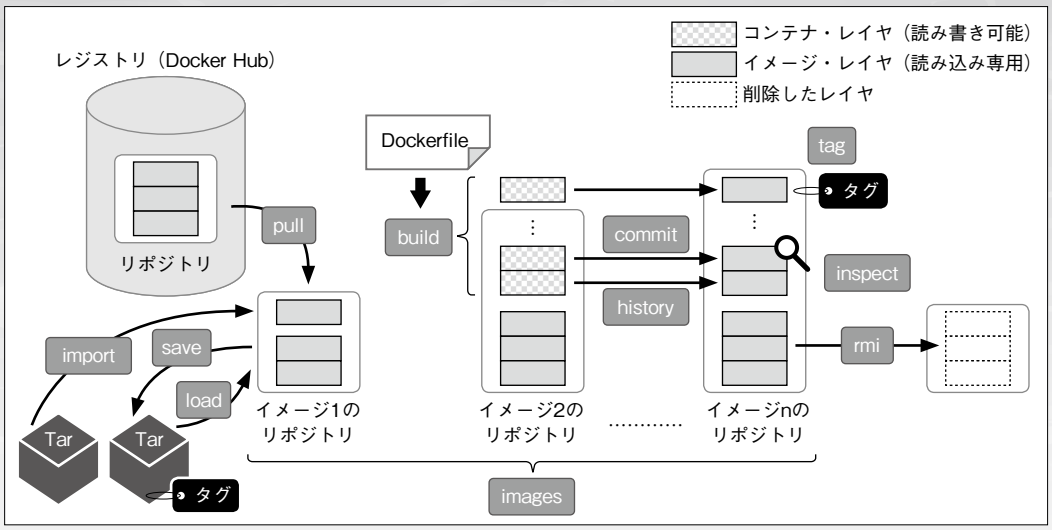
以降では、作業時のフローを通して、コマンドに対する理解を深めていきましょう。各コマンドには次のようにそれぞれ頻出度を振り、よく使われるものを抜粋して紹介していきます(書式やオプションの説明は、特筆するものがないコマンドについては省いています)。

| | |
|---|---|
| ★★★ | 頻繁に使うので、覚えておくべきコマンド |
| ★★ | たまに使うので、存在を知っておいたほうが良いコマンド |
| ★ | 用途が限られるので、必要なときに参照 |

**▼表1　Dockerコマンドの分類**

| 分類名 | コマンド | 説明 |
|---|---|---|
| イメージ | build、comit、history、images、impoart、inspect、load、rmi、save、tag | イメージの構築や管理など、イメージ操作用 |
| コンテナ | attach、cp、create、diff、events、exec、export、kill、logs、pause、port、ps、rename、restart、rm、run、start、stats、stop、top、unpause、update、wait | コンテナの実行や停止など、コンテナ操作用 |
| レジストリ | login、logout、pull、path、search | Docker Hubなどのレジストリに対する操作や検索 |
| ネットワーク | network connect、network create、network disconnect、network inspect、network ls、network rm | Dockerコンテナ間で使うネットワークの操作用 |
| ボリューム | volume create、volume inspect、volume ls、volume rm | ボリュームの作成や削除など、管理用コマンド |
| システム | dockerd、info、inspect、version | dockerデーモンの操作や、イメージとコンテナの調査用 |

▼図1　イメージ操作コマンドの関係




またコマンドの一覧は、Dockerのコマンドライン・リファレンス[注1]からもご確認いただけます。


Dockerコンテナを実行するためには、Dockerイメージが必要です。イメージとは、イメージ・レイヤの親子関係を積み重ねたものです。そのため、イメージの操作コマンドは、イメージ・レイヤに対する操作を意味します(**表2**、**図1**)。


**○書式とおもなオプション**

```
docker pull [オプション] リポジトリ[:タグ]
```

・--all-tags、-a……すべてのタグを一括ダウンロード

Docker Hubなどのレジストリ上にあるリポジトリから、イメージを取得(pull)します。通常は、何も指定しなければDocker Hubからイメージをダウンロードします。リポジトリ名とタグを指定すると、対象のイメージをダウンロードします。なお、書式にある[]内のオプションとタグは必須ではありません。また、リポジトリにIPアドレスやホスト名を指定すれば、Docker Hub以外のリポジトリからもイメージを取得できます。

タグを指定しなければ、自動的にlatestタグが付与されます。次のcentos:latestとcentos:7のタグは、現時点では同じです。latestは自動で最新版を取得できるので、とりあえず

▼表2　イメージ操作コマンド

| コマンド名 | 説明 | 頻出度 |
|---|---|---|
| build | Dockerfileからイメージを構築 | ★★★ |
| commit | コンテナのファイル変更内容をイメージ化 | ★★★ |
| history | イメージの構築履歴を表示 | ★★ |
| images | ローカルのイメージを一覧 | ★★★ |
| import | tarアーカイブからイメージを作成 | ★ |
| inspect | イメージの詳細情報を表示 | ★★ |
| load | tarアーカイブまたは標準入力からイメージ作成(タグ付き) | ★ |
| pull | Dockerレジストリからイメージやリポジトリをダウンロード | ★★★ |
| rmi | イメージを削除 | ★★★ |
| save | イメージをtarアーカイブとして保存(タグ付き) | ★ |
| tag | イメージにタグを付ける | ★★★ |

注1）Dockerコマンドライン・リファレンス　URL https://docs.docker.com/engine/reference/commandline/

▼図2　docker historyの実行例

```
$ docker history 8315978ceaaa
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
8315978ceaaa   4 weeks ago    /bin/sh -c #(nop)  CMD ["/bin/bash"]            0 B
<missing>      4 weeks ago    /bin/sh -c #(nop)  LABEL name=CentOS Base Ima   0 B
<missing>      4 weeks ago    /bin/sh -c #(nop) ADD file:e5428f255dd7260252   194.6 MB
<missing>      3 months ago   /bin/sh -c #(nop)  MAINTAINER https://github.   0 B
```

使う場合には役立ちます。

・centosのlatestタグを持つイメージを指定

```
$ docker pull centos:latest
```

・centosの7タグを持つイメージを指定

```
$ docker pull centos:7
```

ですが、イメージによっては頻繁なバージョンアップや中身の変更により、いざ使いたいときに動かなくなってしまうという問題が起こり得ます。そのため、検証用途以外であれば、タグを指定するよう習慣付けするほうが望ましいでしょう。

Docker Hub以外のレジストリを指定する場合は、リポジトリ名の前にホスト名とポート番号を付けます。たとえば、ホスト192.168.0.100のポート5000で動作しているレジストリにあるmyimage:latestを取得する場合は、次のようになります。

```
$ docker pull 192.168.0.100:5000/myimage
:latest
```

## docker images

○書式

```
docker images [オプション] [リポジトリ[:タグ]]
```

ホスト上のイメージを表示するのがdocker imagesです。イメージはリポジトリという単位で管理されており、各イメージはバージョン番号を表すタグを持っています。オプションを付けずに実行すると、すべてのイメージが含まれるリポジトリとタグを一覧表示します。扱うリポジトリが多くなった場合、オプションでリポジトリ名を指定すると便利です。

## docker inspect

○書式

```
docker inspect リポジトリ[:タグ]
```

イメージに対する詳細な情報を表示するコマンドです。そのイメージがいつ作成されたか、64文字のイメージIDなどの情報を返します。とくに便利なのは、コンテナ実行時のデフォルトのコマンドとパラメータの表示です。コンテナとして実行しなくても、どのような役割を持つイメージなのかを確認できるため、デバッグやメンテナンスに役立ちます。

## docker history

○書式

```
docker history [オプション] リポジトリ[:タグ]
```

イメージがどのようなイメージ・レイヤの積み重ねなのかを確認するコマンドです。自分の使おうとしているイメージが、どのような過程を経て作られたのかという情報と、各レイヤの容量がわかります(**図2**)。

## docker commit

○書式とおもなオプション

```
docker commit [オプション] コンテナ名またはID
[リポジトリ[:タグ]]
```

・-a……作者(author)の記録
・-m……コミット時のメッセージ

正確に言うと、Dockerコンテナ用のイメージ・レイヤをもとに、新しいイメージ・レイヤ

▼図3　GitHubをパスに指定したdocker buildの実行例

```
$ docker build https://github.com/<リポジトリ>/<ファイル>.git
```

を作成するコマンドです。コミット時はコンテナが一時停止(paused)の状態になりますが、オプション(-p)を使うと、停止させないということもできます。

## docker build

○書式とおもなオプション

```
docker build [オプション] パス
```

・-f……Dockerfileのパス(デフォルトは「パス/Dockerfile」)
・-t……タグの指定
・--no-cache……構築時にキャッシュを使わない(デフォルトは有効)
・--pull……常にレジストリから新しいイメージをダウンロード

イメージを自動構築するコマンドです。Dockerfileの中にある命令を読み込みます。命令1つごとに中間コンテナをバックグラウンドで起動し、命令を処理し、イメージをコミットします。このコマンドはキャッシュ機能が使えるため、Dockerfileの命令に変更がなければ、スムーズにイメージを構築できるコマンドです。

```
$ docker build -t myapp:latest .
```

コマンド実行時、タグの指定-tは、ほぼ必須のオプションです。またパスの指定では、通常はDockerfileのあるカレント・ディレクトリを指定します。あるいは、GitHubなどGitリポジトリやTarアーカイブの指定もできます(**図3**)。

また、バッチ処理などでbuildの成功／失敗を判断するときには、戻り値の確認が役立ちます。正常であればecho $?の戻り値は「0」です。それ以外の場合は数値のエラーを返します。

## docker tag

○書式

```
docker tag イメージ[:タグ] 新しいリポジトリ[:タグ]
```

既存イメージに対して新しいタグを追加します。1つのイメージIDに対して複数のタグを付けられるため、ホスト上でのディスク容量は増えません。イメージの部分はイメージ名か、イメージ用の「リポジトリ名:タグ」、またはイメージIDのいずれかを指定できます。

```
イメージ名指定
$ docker tag nginx mynginx:1.0
リポジトリ名:タグ指定
$ docker tag nginx:latest mynginx:1.0
イメージID指定
$ docker tag abf312888d13 mynginx:1.0
```

## docker rmi

○書式とおもなオプション

```
docker rmi [オプション] リポジトリ
```

・-f……依存関係があっても強制的に削除

イメージはホスト側のディスク容量を消費しますので、このコマンドで使わなくなったイメージは削除しておきます。あるいは、不要なタグを削除する場合にも使えます。

docker rmi実行時、コンテナ用のイメージを使っている場合に警告が出ます。ほかのイメージから参照されている場合も、警告が表示されます。そのため、-fオプションで強制削除する前に、エラー内容の確認が必要です。

またdocker imagesコマンドと組み合わせると、イメージを強制一括削除することもできます。docker images -aqはすべてのイメージIDだけを画面に表示するという動作をしま

▼表3　コンテナ操作コマンド

| コマンド名 | 説明 | 頻出度 |
|---|---|---|
| attach | 実行中のコンテナにアタッチ | ★ |
| cp | コンテナ内のファイルをホスト側にコピー | ★★ |
| create | 新しいコンテナの作成 | ★ |
| diff | コンテナ内のファイルに対する変更を調査 | ★★ |
| events | すべてのコンテナの操作状況(attach、commitなど)を一覧表示 | ★ |
| exec | 実行中コンテナ内でコマンド実行 | ★★ |
| export | コンテナのファイル階層をtarアーカイブに出力 | ★ |
| kill | 実行中のコンテナをkillシグナルで停止 | ★★ |
| logs | コンテナの標準出力を表示 | ★★★ |
| pause | コンテナのプロセスを一時停止 | ★ |
| port | 特定またはすべてのコンテナに対するコンテナの表示 | ★ |
| ps | コンテナを一覧表示 | ★★★ |
| rename | コンテナ名の変更 | ★★★ |
| restart | 実行中のコンテナを再起動 | ★★★ |
| rm | コンテナを削除 | ★★★ |
| run | 新しいコンテナを実行 | ★★★ |
| start | 停止中のコンテナを起動 | ★★ |
| stats | リソース(CPU、メモリ、ネットワーク、ブロックI/O)を表示 | ★★ |
| stop | 実行中のコンテナを停止 | ★★★ |
| top | コンテナで実行しているプロセスを表示 | ★★★ |
| unpause | 一時停止したコンテナの再開 | ★ |
| update | コンテナの設定を更新 | ★ |
| wait | コンテナ終了まで待機し、終了コードを表示 | ★★ |

す。そのため、次のように実行すると、イメージを逐次削除します。

```
$ docker rmi -f $(docker images -aq)
```

docker rmiの次に出てくる$(<コマンド>)の部分はシェルスクリプトの機能を利用して括弧内のコマンドを実行した結果を変数展開します。


Dockerコンテナの実行とは、Dockerイメージの中に含まれているファイルをコンテナ状態で実行することです。

Linuxカーネルの名前空間との分離(isolate)により、ホスト上に存在する個々のプロセスやファイルシステムは、それぞれのコンテナで分かれています。また、cgroup(control group)機能によって、各プロセスに対してCPUやメモリなどのリソースが割り当てられます。

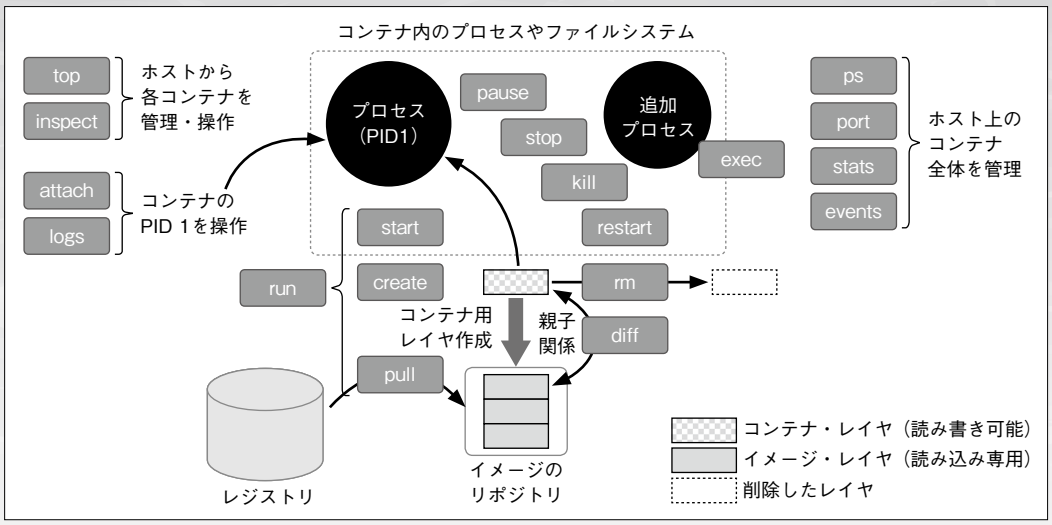
このコンテナの実行や停止が、Dockerを使ううえでの中心的な操作となります。順を追って理解していきましょう(**表3**、**図4**)。


**○書式とおもなオプション**

```
docker run [オプション] イメージ [コマンド] [引数]
```

- -i……標準入力の有効化
- -t……疑似ターミナルの有効化
- -d……バックグラウンド(デタッチド・モード)で実行
- -P……コンテナのポートをホスト側に自動で割り当て(マッピング)
- -p <ホスト側ポート>:<コンテナ側ポート>

▼図4　コンテナ操作用コマンドの関係




……ポートを手動で割り当て

・-v［ホスト側パス:］<ボリューム名>
……ボリュームをコンテナに割り当て

・--rm……コンテナ実行終了時、自動的にコンテナを削除

新しいDockerコンテナを作成し、起動するコマンドです。イメージ(リポジトリ名とタグ)を指定すると、イメージ内のファイル階層に含まれているファイルを、Dockerが動作するホスト上のプロセスとして起動します。

なお、コンテナを実行するdocker runは複数のコマンドの集合です。これは、次の3つのコマンド実行時と同じ結果になります。

・docker pull……ローカルにイメージがなければダウンロード
・docker create……新しいコンテナ(用のイメージ・レイヤ)を作成
・docker start……イメージ・レイヤに含まれるファイルをコンテナのプロセスとして実行

イメージをダウンロードして、そのまま実行するにはdocker runは便利なコマンドです。実行する前にファイルをコンテナにコピーしたい場合や、イメージを取得する場合、起動時のパラメータを実行前に設定しておきたい場合は、各コマンドを使います。

## docker ps

○書式

```
docker ps［オプション］
```

実行中のコンテナ一覧を表示するコマンドです。デフォルトでは実行中のコンテナ情報しか表示しません。停止中のコンテナを含めて表示したい場合はdocker ps -aと実行します。

便利なオプションは-lです。これは直近で操作したコンテナの情報を表示します。コンテナが停止／実行中にかかわらず情報を表示しますので、作業時に役立ちます。

## docker attachとdocker exec

どちらも実行中のコンテナで操作するときに使うコマンドです。しかし、操作対象のプロセスが異なります。

docker attachはコンテナ内の「PID 1」のプロセスに対して接続します。コンテナ実行時に-iと-tオプションを指定しておけば、ログの出力やキーボードからの入力を受け付けられます。このとき、Ctrl+Cなどを入力するとシグ

ナルがPID 1に直接送信され、プロセスが停止することがあるため注意が必要です。

docker execはコンテナ内に新しいプロセスを追加して実行します。docker attachとは違って別のプロセスを追加しますので、デバッグ時には重宝するコマンドです。次は、コンテナ内でbashを実行する例です。

```
$ docker exec -it <コンテナ> bash
```

## docker logs

**○書式とおもなオプション**

```
docker logs [オプション] コンテナ
```

- -f、--follow……表示し続ける
- --tail <行数>……末尾から指定した行を表示
- -t……タイムスタンプ表示

コンテナの標準出力を確認するコマンドです。docker attachを実行しなくても、プロセスの出力やログを確認できます。何もオプションを付けなければ、コンテナ起動時から今に至るすべての情報を表示します。ログが多い場合は-fと--tailの組み合わせが便利です。次は、直近の10行目以降の情報を表示し続ける例です。

```
$ docker logs -f --tail 10 コンテナ
```

## docker rm

**○書式とおもなオプション**

```
docker rm [オプション] コンテナ
```

- -f……強制削除
- -v……ボリュームの削除

コンテナを削除するコマンドです。コンテナ作成時にコンテナ用のイメージ・レイヤを作成すると、このレイヤを削除するまで、コンテナの情報は残り続けます。

オプション-fはコンテナが実行中かどうかにかかわらず、強制的にコンテナを削除します。また-vは、コンテナ起動時に作成したボリュームも一緒に削除するコマンドです。ボリュームはコンテナの作成、実行、削除から独立しているため、このように明示しないと削除できません。コンテナ削除時に残ったままのボリュームはdocker volume lsで対象イメージを確認したあと、docker volume rmコマンドで削除できます。

また、一括してコンテナを削除するには、次のように複数のコマンドをまとめられます。

```
$ docker rm $(docker ps -aq)
```

括弧内のコマンドdocker psには2つのオプションが付いています。-a(--all)は実行中、停止中を含む、すべてのコンテナを表示するためのオプションです。そして-q(--quite)は、コンテナIDのみを表示します。

つまり、docker ps -aqで実行中／停止中を含めたすべてのコンテナIDを表示し、その出力結果を変数としてdocker rm <**コンテナ**ID>を実行するのと同じ状態を実現しています。

なお、シェル上のエイリアス機能を使えば、これらのコマンドを忘れても大丈夫です。**図5**のようにコマンドを登録しておけば便利でしょう。

## レジストリ操作関連コマンド

Dockerイメージを別のホストへ移動するにはレジストリ(registry)を通します。レジストリとはイメージを保管する場所で、一番有名なのは、Docker Hubです。これはパブリックで、誰でも利用可能なレジストリという位置付けであり、各種の公式イメージが配布されています。ほかにも、ローカルで使うレジストリやDocker Trusted Registry(トラステッド・レジストリ)もあります。そのレジストリのイメージ送受信

**▼図5　エイリアス機能でdocker rmのコマンドを登録**

```
$ alias drm='docker rm $(docker ps -aq)'
$ alias drmi='docker rmi -f $(docker images -aq)'
```

▼表4　レジストリ操作コマンド

| コマンド名 | 説明 | 頻出度 |
|---|---|---|
| login | Dockerレジストリにログイン | ★★★ |
| logout | Dockerレジストリからログアウト | ★ |
| pull | Dockerレジストリからイメージやリポジトリをダウンロード | ★★★ |
| push | Dockerレジストリからイメージやリポジトリをアップロード | ★★★ |
| search | Docker Hub上のイメージを検索 | ★ |

や、認証に関するコマンドを見ていきましょう(**表4**、**図6**)。レジストリ操作関連コマンドのポイントは次の2点です。

・docker pullでイメージを受信
・docker pushでイメージを送信。ただし、Docker Hubに送信する場合は事前にdocker loginで認証が必要

## docker loginとdocker logout

○書式

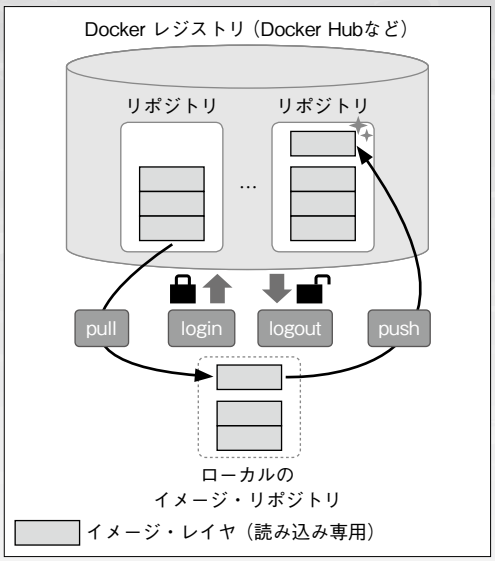
```
docker login [オプション] [サーバ]
```

認証機能付きのレジストリに接続するコマンドです。デフォルトではDocker Hubへのログインを試みますが、任意のレジストリを指定できます。

ログインに成功すると(**図7**)、Docker Hubではdocker pushコマンドが利用可能になります。また、認証時の情報は~/.docker/config.jsonに格納されています。

docker logoutコマンドを実行するか、ファイルを削除するまで認証が有効なため、注意が必要です。docker logoutはDockerレジストリからログアウトするコマンドで、書式は次のとおりです。

▼図6　レジストリ操作コマンドの関係




○書式

```
docker logout [サーバ]
```

## docker push

○書式

```
docker push [オプション] リポジトリ[:タグ]
```

レジストリにイメージを送信(push)します。Docker Hubに送信する場合は、「ユーザ名/リポジトリ」を指定します。そのため、送信前にdocker tagコマンドでユーザ名を含むリポジトリ名に変更します(**図8**)。

▼図7　docker loginの実行例

```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a
Docker ID, head over to https://hub.docker.com to create one.
Username: <Docker Hubのユーザ名を入力>
Password: <パスワードを入力>
Login Succeeded
```

▼図8　docker push前には、docker tagで名前を変更する

```
$ docker commit mynginx  mynginxイメージ作成
$ docker tag mynginx zembutsu/mynginx  <ユーザ名>/mynginxタグを付ける
$ docker push zembutsu/nginx  送信
```

## docker search

○書式

```
docker search［オプション］検索語
```

コマンドライン上でDocker Hub上のイメージを検索します。表示する情報は、「名前」「公式なイメージであるかどうか」「スター数」のみです。詳細なドキュメントやタグ一覧の確認は、ブラウザでDocker Hubの各リポジトリをご覧ください。

## システム管理用コマンド

最後に、デバッグやシステム情報確認のための管理コマンドを紹介します(**表5**)。その中でも重要なコマンドを2つ見ていきます。

## docker version

おもな用途は、Dockerクライアントとサーバのバージョン情報を確認するコマンドです。Dockerはバージョンアップが頻繁ですので、使っているクライアントとサーバのバージョンに相違があれば、各種のコマンドやオプションに相違が出てしまうこともあります。そのため、利用前にはバージョン確認をお勧めします。

▼表5　システム管理用コマンド

| コマンド名 | 説明 | 頻出度 |
|---|---|---|
| dockerd | Docker デーモン(Docker Engine)の起動や管理用 | ★ |
| info | さまざまなシステム情報を表示 | ★★★ |
| inspect | コンテナやイメージに関する情報を表示 | ★★★ |
| version | Dockerクライアントとサーバ両方のバージョン情報を表示 | ★★★ |

また、クライアントがサーバに接続できるかどうかの確認にも使えます。コマンド実行後に、サーバのバージョン番号ではなく、「Cannot connect to the Docker daemon.(デーモンに接続できない)」とエラーが出る場合は、デーモンが起動しているかどうかや、/var/run/docker.sockへのアクセス権限があるかどうか確認しましょう。

## docker info

Dockerサーバの情報を一覧表示します。実行中や停止中のコンテナ数だけでなく、サーバ上のメモリやディスク容量のほか、ストレージドライバなどのシステム情報を表示します。

docker infoコマンドをよく使うのは、サーバにセットアップした直後や、初めてログインする環境です。Dockerのデフォルト起動時のオプションはディストリビューションによって異なります。また、サーバの設定がパフォーマンスに影響を与える場合もありますので、役立つことも多いでしょう。

◆ ◆ ◆

Dockerのコマンドは多岐に渡りますが、それぞれイメージの構築や実行のライフサイクルと関係があります。重要なコマンド以外、すべてのコマンドを覚える必要はありません。あとは利用形態に合わせて、コマンドの詳細オプションの理解を深めれば、Dockerを自分の手足のように、効率的に使えるようになるでしょう。SD

**参考文献**


・Dockerコマンド・リファレンス(公式)
https://docs.docker.com/engine/reference/commandline/
・日本語訳
http://docs.docker.jp/engine/reference/commandline/

第5章 現場は何に悩み、何を解決したのか

# 導入事例で見えてくるDockerの使いどころ


この章では筆者が日々の開発において直面した開発サーバ上の検証環境の不足問題を、Dockerコンテナの導入、そしてmirageというDockerコンテナの管理を便利にするミドルウェアの開発によって解決した話を紹介します。

Author 川添 昌俊(かわぞえ まさとし) Twitter @acidlemon
面白法人カヤック

Author 矢吹 遼介(やぶき りょうすけ) Twitter @Konboi
面白法人カヤック

## 慢性化した検証環境不足

筆者が担当しているチームではスマートフォン向けゲームの開発／運用を行っています。チーム規模は全体で20～30人程度です。

チーム内にはいくつかのサブチームがあり、複数の開発ラインが並行で開発をしています。検証環境が複数必要となるため、1つのサーバにdev01～10まで番号を振ったサブドメインの検証環境を用意し、エンジニアがメンバーの依頼を受けてブランチを切り替え、使用する運用を行っていました。

しかし、メンバーが増えるにつれ検証環境が枯渇する時間と、各検証環境のgitブランチを切り替える回数が増えました。この作業はデータベースのマイグレーションやデプロイなど、サーバサイドエンジニアの作業が必要不可欠です。環境を切り替える回数が増大した結果、サーバサイドエンジニアのリソースが逼迫(ひっぱく)して開発に遅延が発生するようになりました。

この問題を打開するため、Dockerコンテナに注目しました。Dockerコンテナに検証環境を入れることができれば、環境を使い捨てることができます。切り替え作業が単純になり、サーバサイドエンジニアの手間を大幅に軽減できると考えました。

## Dockerの導入どころを検討する

Dockerは便利ですが、いきなりすべてをDockerにしていくのはなかなか馬力を必要とする作業が待っています。ある程度以上の規模のプロダクトを開発するようなチームにDockerを投入するためには、チームのエンジニア全員がDockerをある程度理解することが必要となります。さらに、本番環境のデプロイにDockerを利用する場合はインフラエンジニアもDockerに精通していることが求められます。

### 開発環境の構築に利用する

Dockerの利用を検討するような現場では、開発マシンとしてはOS X(macOS)もしくはLinuxを利用しているはずです。Windowsを利用していても、おそらく仮想マシンで動かしたLinux環境はあることでしょう。Dockerを利用しない場合の開発環境構築は、おもに次のような手段が一般的です。

- POSIX系OSの開発マシン上に直接環境構築する
- 開発マシン上で起動した仮想マシン上のLinuxに環境構築する

前者の方法は構築が簡単な半面、開発マシン

を長く使っていると、開発環境のライブラリバージョンなどが異なる複数のプロジェクトを開発するのに苦労することが多くなってきます。

後者は前者の問題を回避するため、プロジェクトごとに仮想マシン（VM）を用意して、そのVMごとにそれぞれ環境を構築する方法です。この方法はVM単位で環境を完全に分け、ライブラリバージョンを独立に管理することができます。この方法は、Vagrantを使ってプロビジョニングまでを自動化するやり方で自動化が可能です。

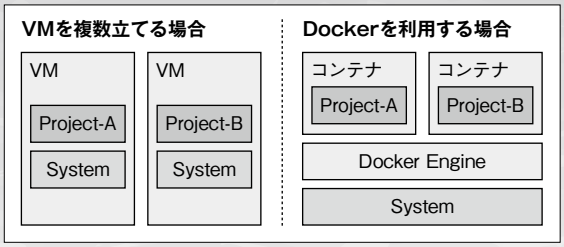
VM別に環境を分けると、環境の削除はVMを削除するだけで完了するため、他のプロジェクト用の環境に傷をつけることがありません。ただし、各VMごとにLinuxカーネルなどのシステム部分を仮想ディスク上に持つ必要があるため、ディスク効率が悪くなるというのがデメリットです。

Dockerを開発環境の構築に利用する際のイメージは、仮想マシンを利用した環境構築に近いです。仮想マシンを利用する場合、Vagrantを使って、Linux VMの起動と必要な開発ツールやライブラリをインストールするプロビジョニング作業をセットにすると、効率的に環境を構築できます。Dockerを利用する場合は、Dockerfileでベースのイメージを指定し、必要な開発ツールやライブラリをインストールして開発環境用のコンテナを構築します（**図1**）。

どちらも決められた手順により再現性のある環境構築ができるというメリットは同じです。再現性のある環境構築により、チームメンバー間の環境差分が出ないようにできます。

開発環境構築にDockerを利用するメリットは、コンテナの起動速度と環境の変更の柔軟性にあると筆者は考えます。開発環境のバージョン変更などを行いたいときの実験的な操作は、VMの再プロビジョニングが必要な仮想マシンよりも、ダメだったときにコンテナごと捨ててしまい、コンテナを起動しなおせばよいDockerのほうが気軽に試すことが可能です。また、開発マシンがLinuxであれば、Dockerの動作環境を満たしていれば新たにVMを起動する必要もなくなり、オーバーヘッドが小さくなることも挙げられます。

**▼図1　環境構築における仮想マシンとDockerの比較**




## 本番環境の構築に利用する

開発環境をDocker化したら、次は本番環境です。しかしながら、筆者が所属する会社ではほぼ本番環境でDockerを採用していません。現時点では一部の社内サービスに採用するに留まっているという状態です。

なかなか採用が進んでいない理由としては、リリース済みのモノリシックな巨大Webアプリケーションを Docker化する労力が、得られるメリットに見合わないという点があります。筆者のかかわるところで一番開発規模が大きいスマートフォン向けゲームのプロジェクトだと、22万行程度のPerlコードに加え、依存パッケージが400個程度あります。この規模のプロジェクトだと、Dockerイメージを高頻度で作り直すのが難しくなっています。

元々デプロイにDockerやImmutable Infrastructureを使うことを前提としていないプロジェクトをDocker化するのは、インフラエンジニアも巻き込んで進める必要があります。たとえば、ゼロダウンタイムデプロイをどうするか、迅速なロールバックをどうやるか、などなどDockerを本番環境へ導入するにあたっての課題はたくさんありますので、よく検討のうえで進める必要があります。

本番環境への導入はインフラ構築のトピック

が中心となるため、本章では開発環境の構築について深掘りしていきます。

## アプリケーションコンテナとシステムコンテナ

Dockerでコンテナを作る場合、どのような役割のコンテナを作るかということを意識する必要があります。コンテナの役割は、大きく分けて2種類あります。

・単一のアプリケーションを動かすコンテナ
・複数のアプリケーションを動かすコンテナ

Dockerはコンテナ起動時にDockerfileのCMD行で指定した、もしくはdocker runで指定したコマンドを1つだけ実行し、そのプロセスが終了するとコンテナを停止します。コマンドを1つしか実行できないため、基本的には前者の単一のアプリケーションを動かすコンテナとして構成するのが通常の使い方です。これをアプリケーションコンテナと呼びます(**図2左**)。

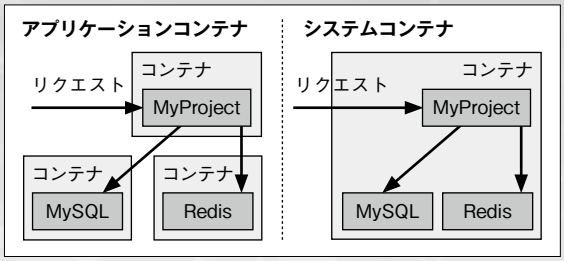
コンテナ起動時に実行するコマンドを工夫すると、1つのコンテナの中で複数のアプリケーションを動かすことも可能です。たとえばPerlのProcletやRubyのForemanを利用し、Webアプリ本体のほかにMySQLやRedisなどのミドルウェアを起動する、といったことができます。このようなコンテナをシステムコンテナと呼びます(**図2右**)。

アプリケーションコンテナとシステムコンテナにはそれぞれメリット／デメリットがあります。

### アプリケーションコンテナのメリット／デメリット

アプリケーションコンテナのメリットは、アプリケーションから参照する永続化したデータ(RDBやNoSQLに入ったデータ)をコンテナの外に置くことで、コンテナの起動終了でデータが失われないことです。複数台のサーバを使うWebサービスは基本的にデータベースサーバとWebアプリケーションサーバは分けて配置しますので、そのような環境に向いています。

**▼図2　アプリケーションコンテナとシステムコンテナの違い**




また、コンテナイメージにはアプリケーションの動作に必要なファイルだけを格納すればよいため、システムコンテナと比較してコンテナイメージが小さくなります。コンテナが小さいということは、ネットワーク経由でコンテナイメージを配布するのにかかる時間が短くなるということですので、デプロイにかかる時間が短くなります。

近年はマイクロサービスという考え方でアプリケーションを細かく分割するアーキテクチャを取り入れる会社もあり、そのような環境のデプロイにDockerを利用するというケースが多いです。

逆にデメリットとしては、アプリケーションコンテナだけではサービスを動かすことができないため、開発マシンで動かすにはデータベースなどを別途動かす必要があるという点です。ストレージもDockerコンテナ化してdocker-composerを利用して開発マシン上で動かす、もしくはクラウド上のマネージドサービスに開発マシンから直接接続するなどの方法で、ある程度デメリットを軽減することができます。

### システムコンテナのメリット／デメリット

システムコンテナのメリットは、コンテナイメージが1つあれば誰の手元でも動かすことができる点です。たとえば(Dockerに詳しくない)デザイナーやプランナーの手元でも、コマンドさえ実行すれば動かすことができます。現実的にはなかなか難しいかもしれませんが、しくみを作ることで十分可能となるポテンシャルを秘めています。また、コンテナを破棄するとすべ

て消えますので、データを残す必要がない場面で有効です。手元でちょっと動かして、動作確認したらすぐ捨てる、といった用途にはうってつけです。

デメリットとしては、アプリケーションコンテナよりもイメージサイズが大きくなるという点があります。いろいろなミドルウェアが同居したコンテナとなるため、この点は許容するしかありません。

## どちらを選択するか

アプリケーションコンテナとシステムコンテナ、どちらを選ぶべきかは以上のようなメリット／デメリットを勘案して決める必要があります。どちらが優れている、劣っているというものではなく、用途に合わせて適切に選択するのが大事です。

本番環境ではスケールを考慮するとアプリケーションコンテナ一択となりますが、開発環境を整備するという場面では、システムコンテナにすることで利便性を得られる場面もあります。コンテナをどのぐらい長く使うかというライフサイクルの観点と、ライフサイクルが単一なのか、長いものと短いものがあるのかによってアプローチが変わります。

たとえば、CI(継続的インテグレーション)によるテスト実行のように、コンテナ上でアプリケーションおよびテスト用データベースを起動し、テスト完了時に破棄するような用途であればシステムコンテナを作るのが向いています。開発用サーバで起動しっぱなしにしておくコンテナであれば、データベースのデータは引き継ぎたいが、Webアプリは入れ替えたいというニーズに対するアプローチとしてアプリケーションコンテナを複数立ち上げてライフサイクルを分けるという方法が適切です。

# コンテナを用いた開発環境整備

ここまでで、アプリケーションコンテナとシステムコンテナについて説明しました。ここからは実際にDockerイメージを作って環境構築する例を紹介します。

## 最初の一歩

**リスト1**に、Perlの開発環境用Dockerfileの例を示します。

このDockerfileは、開発マシン上にあるソースコードをDockerコンテナの/myappディレクトリにマウントして、コンテナ内で動かすためのコンテナイメージを作るものです。依存パッケージの管理のため、パッケージマネージャをインストールしています。

パッケージマネージャでインストールするライブラリが混ざらないようにするため、Cartonを利用します。一番原始的な開発環境としてはこれでよいでしょう。

Dockerfileの最後で、パッケージマネージャ経由でコマンドが実行されるようENTRYPOINTを設定します。これを指定すると、docker runするときに実行するコマンドはこのENTRYPOINT経由の起動となります。

このコンテナイメージを、プロジェクト名をタグに指定してビルドします。次の例では、プロジェクトディレクトリの下にdockerディレクトリを作成し、そこへDockerfileを配置しています。プロジェクトディレクトリで次のコマンドを実行します。

```
$ docker build -t myapp -f docker/Dockerfile docker
```

Dockerイメージのビルドが完了したら次のコマンドを実行します。

```
$ docker run -v $(pwd):/myapp -it myapp /bin/bash
```

これで、コンテナ上でソースコードのあるディレクトリを/myappにマウントした状態でbashを起動できました。carton installで、まず依存ライブラリをインストールします。依存ライブラリはマウントしたディレクトリにインストールされるため、プロジェクトディレクトリ

▼リスト1　Perl向けDockerfile

```
FROM perl:latest

RUN mkdir /myapp
VOLUME /myapp

RUN cpanm Carton
RUN useradd -m myuser

WORKDIR /myapp
USER myuser

ENTRYPOINT [ "carton", "exec" ]
```

▼リスト2　docker_entry.sh

```
#!/bin/sh

sudo service mysql start
sudo service redis-server start

exec carton exec $@
```

▼リスト3　Dockerfile最終版

```
FROM perl:latest

RUN mkdir /myapp
VOLUME /myapp

RUN cpanm Carton

ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update && apt-get install -y ➥
mysql-server redis-server sudo
RUN echo "myuser ALL=NOPASSWD: ALL" >> /etc/➥
sudoers

COPY docker_entry.sh /usr/local/bin/docker_➥
entry.sh

RUN useradd -m myuser
WORKDIR /myapp
USER myuser

ENTRYPOINT [ "/usr/local/bin/docker_entry.sh" ]
```

のある開発マシン側にインストールされます。

あとは、ソース編集以外の作業はこのコンテナ上で行いましょう。たとえば、Webアプリの起動や、テストの実行などがそれに当たります。

この方式で行うと、実行する環境はDocker上で一意に決まるため、多人数開発にありがちな「Macで開発する人とLinuxで開発する人がいるためsnapshotファイルがOSにより異なる内容になり、Gitで管理できない」という問題を解決することができます。本番環境にもsnapshotファイルを使って環境を作ることができるため、当たり前のことですがとても心強いです。

## システムコンテナを作る

前の節で作ったコンテナに、MySQLとRedisをインストールしてCIができるようにしてみましょう。

DockerfileにMySQLとRedisのインストールを追加します。myuserユーザでサービスを起動するため、sudoもインストールします。

コンテナ起動時にmysqldとredis-serverを起動する方法はいくつかあります。ここでは一番簡単な、シェルスクリプトで起動する方法を紹介します。

**リスト2**のシェルスクリプトをdocker_entry.shという名前で作成してdockerイメージに組み込み、ENTRYPOINTに指定します。このスクリプトをdockerディレクトリに配置して、実行パーミッションを付与します。

最終的にできたDockerfileは**リスト3**です。

これを先ほどと同じ手順で`docker build`し、`docker run`します。コンテナ起動時のコマンドに`mysql -u root`を指定すると、コンテナ内のMySQLサーバに接続できます。また、`redis-cli`を指定するとコンテナ内のRedisサーバに接続できます。

コンテナ内がどうなっているか調べるため、bashを起動して`pstree`を実行してみた結果が**図3**です。`exec`によりpid=1となったbashの下に、MySQLサーバとRedisサーバが子プロセスとしてぶら下がっています。

これでアプリとともにMySQLサーバとRedisサーバが稼働するシステムコンテナが出来上がりました。開発マシンでCIをする、もしくはWebアプリを立ち上げてみて動作確認してみるなどの用途にはこれで十分なことも多いです。

システムコンテナは全部入りですので、開発環境として立ち上げが簡単です。しかし、中に

入っているWebアプリケーションとストレージサービスでライフサイクルを分けたいというニーズが出てきたときには、システムコンテナではそのニーズを満たすことができません。その場合、Webアプリケーション、データベース、セッションストレージなどとアプリケーションコンテナをいくつも立てる必要が出てきます。次は、開発環境構築で複数コンテナを手際よく立てる方法を紹介します。

## docker-composeを使って開発環境を構築する

新しいプロジェクトに配属されたときやマシンを新調したときなど、開発環境の構築のために必要なミドルウェア／ライブラリのインストールを行う必要がある場面があります。その際、バージョンを固定しているなどの理由で古いバージョンを指定してインストールする必要がある場合、手間となることがあります。

また、複数のプロジェクトの開発を1つの開発マシンで行う場合、ミドルウェアのバージョンが異なるとプロジェクトごとに切り替える必要があります。このような面倒ごとをコンテナを使用することで解決できます。

### docker-composeの利用

docker-composeは、Dockerコンテナを複数利用するアプリケーションの構成をYAMLで定義し、まとめて起動するアプリケーションです。自身で作成したコンテナイメージと、Docker Hub[注1]で提供されているコンテナイメージを一度に起動し、連携させることが可能です。

### docker-composeを使った開発環境の例

**リスト4**は筆者が使っている開発環境のdocker-compose.ymlを一部修正したものです。

注1） URL https://hub.docker.com/

**▼図3　pstreeの実行結果**

```
$ docker run -v $(pwd):/myapp -it myapp:latest /bin/bash
[ ok ] Starting MySQL database server: mysqld ..
[info] Checking for tables which need an upgrade, are corrupt or were
not closed cleanly..
Starting redis-server: redis-server.
myuser@af8b1e78dd42:/myapp$ pstree
bash-+-mysqld_safe---mysqld---16*[{mysqld}]
     |-pstree
     `-redis-server---2*[{redis-server}]
```

**▼リスト4　docker-compose.ymlの例**

```
version: '2'
volumes:
    mysql:
    redis:
services:
  mysql:
    image: mysql:5.6
    environment:
      MYSQL_ALLOW_EMPTY_PASSWORD: "yes"
    ports:
      - "3306:3306"
    volumes:
      - mysql:/var/lib/mysql
  redis:
    image: redis:2.8
    ports:
      - "6379:6379"
    volumes:
      - redis:/var/lib/redis
  perl-app:
    build: "./"
    working_dir: "/myapp"
    ports:
      - "5000:5000"
    volumes:
      - "/:/myapp"
    links:
      - "mysql:docker-mysql"
      - "redis:docker-redis"
    command: "carton exec plackup -r app.psgi"
```

Redis、MySQLを起動し、ホストマシンへのアクセスをコンテナ内のポートへマッピングしています。

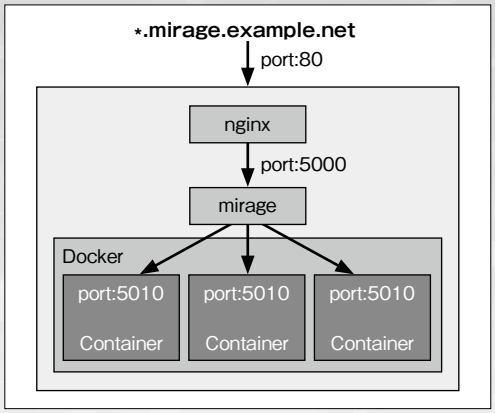
docker-composeを使ったコンテナの起動と終了は、サブコマンドupとstopを使用します。

この設定ファイルでは、コンテナ群の再起動を行ってもデータを保持できるため便利です。

## 開発サーバ上向けのDockerコンテナ作り

開発サーバ上で利用するコンテナは「開発環境

▼図4　mirageを利用した開発サーバ環境の構成図




のスナップショット」として作成するのがオススメです。具体的には、Dockerfileで次のような作業をしてコンテナイメージをビルドします。

・必要なミドルウェア、ライブラリのインストール
・プロジェクトのgitリポジトリのclone
・プロジェクトで利用する言語の依存パッケージのインストール
・データベースへのスキーマ適用

Dockerコンテナを起動するときには環境変数でブランチ名を渡します。起動シェルスクリプトで指定のブランチをチェックアウトし、依存モジュールのインストールとスキーマ適用を行ってからWebアプリケーションを起動します。

このようにするとコンテナ起動時の処理はコンテナイメージへの差分アップデートという形になるため、短時間で更新が終わります。差分の量にもよりますが、1分程度で指定ブランチの内容で最新化したコンテナを起動できます。これで、煩雑だった検証環境の切り替えがコンテナ起動だけで済むようになりました。


ここまでで、Dockerコンテナを利用することで、開発サーバの検証環境切り替えが煩雑になっていた問題を解決することができました。

しかし、依然としてdev01～10までで枠が決まっていて慢性的に枠不足となる問題を解決できていません。そこで、mirageを使うことで各メンバーが必要な環境をDockerコンテナを使い起動し任意のサブドメインを割り当て利用することができるようになりました。

以降でmirageの詳細と設定とmirageの運用におけるTipsについて紹介します。任意のサブドメインを特定のコンテナと結びつけることができるmirageというミドルウェアを開発しました。

## mirageとは

mirageはDockerの利用に特化したリバースプロキシサーバです。APIおよびWebインターフェースでDockerコンテナを起動／停止し、指定のサブドメインへのトラフィックをDockerコンテナにルーティングする機能を持っています。コンテナの起動時には、サブドメインのほかコンテナに渡す任意の環境変数を指定することができます。

カヤック社内ではChatOpsを活用しており、mirageをSlack上のBotからAPI経由で呼び出して利用しています。

## mirageの導入

nginxからmirageを通してコンテナへリバースプロキシする手順を説明します。**図4**の構成を構築します。mirageのインストールはGitHub上の公式サイト[注2]を参照してください。

mirageはデフォルトでは8080ポートを開いて起動します。**<起動しているマシンのIP>:8080**にブラウザでアクセスすると、mirageのダッシュボードを表示します。問題なく起動していれば、ダッシュボードはコンテナ起動ボタンおよびコンテナリストが確認できます。

また、APIを利用できます。**図5**はmirageで管理しているコンテナリストを取得するAPIの

注2）URL https://github.com/acidlemon/mirage

**▼図5　APIでmirageにアクセスしたところ**

```
$ curl <起動しているマシンのIP>:8080/api/list
{"result":[]}
```

**▼リスト5　mirageのconfig.yml**

```
host:
  webapi: web.mirage.example.net
  reverse_proxy_suffix: .mirage.example.net

listen:
  foreign_address: 127.0.0.1
  http:
    - listen: 5000
      target: 5010

docker:
  endpoint: unix:///var/run/docker.sock
  default_image: myapp:latest

parameters:
  - name: branch
    env: GIT_BRANCH
    required: true
    rule:
```

実行結果です。最初は起動しているコンテナがないため、空の値のJSONが返ります。

## mirageの設置

mirageの設定ファイルを編集し、nginxのバックエンドに設置します。例として設定するホストをmirage.example.netとします。適宜読み替えてください。

- APIおよびWebインターフェースのエンドポイントをweb.mirage.example.netに設定
- *.mirage.example.netへのアクセスがmirageにきたら該当のコンテナにリバースプロキシ
- 5000番ポートへのアクセスをコンテナの5010番ポートへプロキシする
- 起動時に使用されるコンテナのデフォルトイメージはmyapp:latestを使用する

このような環境になる設定ファイルは、**リスト5**のとおりです。

Docker APIの呼び出し方法がUnix Socket以外の場合は、endpointの部分を変更する必要があります。

mirage自身はサービス化の機能を持たないため、ユーザが何らかの方法でサービス化する必要があります。参考までに、筆者の環境ではdaemontoolsを利用してサービス化を行っています。

次に、nginxからmirageへリバースプロキシする設定をします。

mirageがサブドメインに対応するコンテナへリクエストを転送するので、nginxはサブドメインをワイルドカードにしてリクエストを受け付けて、mirageに転送します。

## mirageの高度な使い方

mirageは開発当初シンプルなリバースプロキシでしたが、現場で使い込むにつれて機能が増えていきました。現在では、次のような使い方ができます。

- 複数ポートをListenしてコンテナの複数のポートにマッピングする

これはWebアプリ本体と管理画面を1つのコンテナに別ポートで建てるときに便利です。

- コンテナ内から外部DBにサブドメインを含むDB名で接続する

mirageはコンテナ起動時に環境変数SUB DOMAINを設定します。コンテナ内でこれを利用してDB名を生成することで、サブドメインごとに異なる外部DBへ接続できます。

## さいごに

本章ではDockerを使った開発環境の構築について紹介しました。一人で複数のプロジェクトを受け持つ人、多人数で1つのプロジェクトを開発する人、それぞれに採用するメリットがあります。開発環境であれば導入のハードルが低いので、ぜひお試しください。SD

第6章 しっかりと基礎を固める

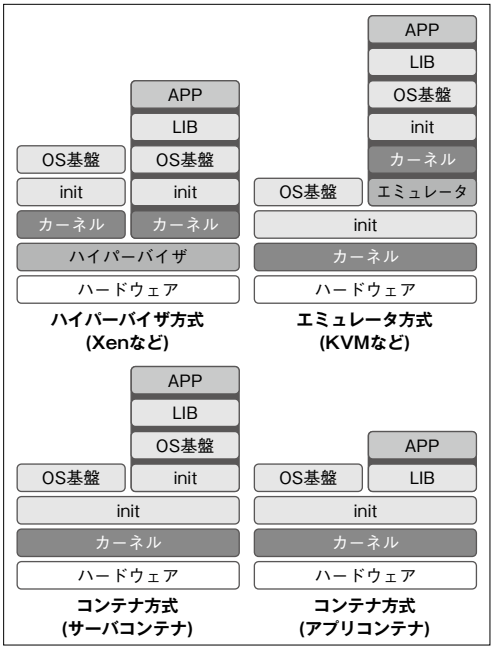
# Linux Containerの歴史としくみ

これまでDockerの使い方をひととおり見てきましたが、コンテナ型仮想化技術と言われるところの、「コンテナ」というしくみについても本章で確認しておきましょう。コンテナ技術そのものは、Unix、そしてLinuxそのものの発展とともに考えられてきた便利なしくみです。


Author
花高 信哉(はなたか しんや)
㈱インターネットイニシアティブ

Dockerは優れたツールでコンテナのしくみについて詳しく知らなくても利用できるのですが、技術者としては内部のしくみがどのようになっているのか気になるところです。しくみを知っていればトラブルに遭遇したときにも原因を追求したり、問題を回避したりできるかもしれません。ということでLinuxのContainer機能について、歴史としくみを簡単に紹介します。

▼図1　仮想サーバとコンテナ型仮想サーバの違い




## コンテナと通常の仮想サーバの違い

XenやKVMのような通常の仮想サーバとコンテナ型の仮想サーバの違いについてもう一度確認してみましょう。**図1**において上の2つが通常の仮想サーバ、下の2つがコンテナの例になっています。

通常の仮想サーバではエミュレータとかハイパーバイザなどと呼ばれる仮想ハードウェアを提供するためのソフトウェアを使い、その上で仮想サーバが実行されています。それに対してコンテナは、1つのカーネルの上に作られた隔離環境として実現されています。同じカーネル内ですべてが動いているため、余分なリソースを必要とせず最適なリソース配分が可能なことや、オーバーヘッドがないおかげで軽くて速いなどの特徴があります。

## 歴史からひもとくコンテナ

### chrootから始まった

まずは少しさかのぼってコンテナのしくみの歴史について見てみましょう(**表1**)。最初のプロセスを隔離するしくみは1979年にUnix System 7に導入された「chroot」です。

▼表1　コンテナにまつわる主な技術の年表

| 年 | 仮想化機能の登場 |
|---|---|
| 1979 | chroot［Unix System 7］ |
| 1998 | jail［FreeBSD］ |
| 2001 | Linux-VServer（パッチ） |
| 2001 | Virtuozzo（商用） |
| 2002 | **Mount Namespace** |
| 2003 | （Xen 1.0パッチ） |
| 2005 | OpenVZ（パッチ） |
| 2005 | Solaris Container［Solaris 10］ |
| 2006 | **UTS Namespace** |
| 2007 | （KVM統合） |
| 2008 | **CGROUP（cpu、memory、device）** |
| 2008 | **User Namespace** |
| 2008 | **PID Namespace** |
| 2008 | **Network Namespace** |
| 2008 | （LXC） |
| 2009 | **CGROUP（freezer、net_cls）** |
| 2010 | （Xen/pvops統合） |
| 2011 | **CGROUP（blkio、perf_event）** |
| 2012 | **CGROUP（net_prio、hugetlb）** |
| 2013 | **User Namespaceの改修** |
| 2013 | （Docker、systemd-nspawn） |

これはプロセスごとにルートディレクトリの位置を変更し、それより上のディレクトリやファイルにアクセスできなくする機能です。アクセスできるファイルを制限してセキュリティを確保したり、標準とは違うライブラリを置いて独自の開発環境や運用空間を作るなどいろいろな面で活躍しました。

ただchrootは、root権限があれば簡単に抜け出せるため一般ユーザしか閉じ込めることができず、ファイルシステム以外のリソース（たとえばプロセスとかネットワークとか）には自由にアクセスできるため、隔離としては不十分でした。

## FreeBSDのjail

大きな進歩は1998年ごろからFreeBSDに実装が開始されたjailシステムから始まりました。これはchrootの考え方を大幅に進めて、ファイルシステムだけでなくプロセス空間や管理者（root）権限の一部を分離する機能です。ユーザは隔離空間の中でroot権限を持つように振る舞うことができ、IPアドレスを隔離環境に渡すことで内部でWebサーバなどを自由に起動できるようになりました。

## Linuxでの取り組み

それから少し遅れて2001年ごろに、Linuxでもjailのような軽量サーバ機能を提供したいという目的で開発されたのが、Linux-VServerです。これはLinuxカーネルへのパッチとして開発されており、FreeBSDのjailとほぼ同様の機能を持っていました。

同じころに商用のVirtuozzoが開発・販売されています。これは商用だけあって高機能で、ユーザ空間、ネットワーク空間、プロセス空間などの必要なすべての機能を分離し、リソース制限などの機能も備わったコンテナ型の仮想化として十分なものでした。

2005年にVirtuozzoのオープンソース版として公開されたのがOpenVZです。公開後はjailやLinux-VServerなどに代わって、VPS（Virtual Private Server）などの提供に広く使用されるようになりました。

## 標準カーネルへの統合へ

Linux-VServerもOpenVZも標準カーネルへの改造パッチとして提供されていたため、バージョンアップ作業などには余計な手間がかかりました。そのため広く使われるようになると、標準カーネルにこれらの機能を取り込みたいと要望されるようになりました。

しかし、OpenVZのパッチはあまりにも巨大でそのまま統合するには無理がありましたし、コードも自由度のない専用のものでLinuxカーネルの開発者たちを満足させるものではありませんでした。それでカーネル開発者たちはコンテナ機能を小さな機能ごとに分解整理して、もっと汎用性のある形で実装しなおす方針を採用しました。

こうして「Namespace」と「CGROUP」というしくみが実装されていきました。**表1**の年表を

見ていただくと、KVMやXenなどの仮想サーバ機能の実装と並行して、NamespaceとCGROUPが順に実装されていったのがわかると思います。

そして2013年のUser Namespaceの改修により、Linuxカーネルのコンテナ機能が一通りそろいました。それを受けて2013年からDockerを始めとするさまざまなコンテナツールが次々と登場してきたため、この年がコンテナ元年などと呼ばれたりします。

## 名前空間の分離機能：Namespace

それでは具体的にLinuxのNamespace機能について見てみましょう。これは文字どおりプロセスごとの名前空間を分離する機能で、**表2**のように複数の名前空間から構成されています。

### mount namespace

まず最初のmount namespaceですが、ファイルシステムのマウントをプロセスごとに別にして見せる機能です。本来OS上のすべてのプロセスは同じファイルシステムを参照していますが、マウント名前空間を別にすると、あるプロセスからはマウントしているように見えるのに、別のプロセスからはマウントしていないように見せることができます。この機能は2002年に実装されたもので、コンテナシステムを直接目的としたものではなく、ファイルシステムの柔軟な利用を意図したものでした。

### uts namespace

Linuxカーネルの開発者たちはコンテナシステムの実装を計画したときに、このmount namespaceのやり方を採用することにしました。その方針が決まっていち早く実装されたのがuts namespaceで、プロセスごとにホスト名を別にすることができます。utsというのはUnix Time-sharing Systemの略で、伝統的にUnixのホスト名を格納する構造体がutsnameと呼ばれていたため、この名前になっています。

### ipc namespace

ipc namespaceは共有メモリ、セマフォ、メッセージキューなどのプロセス間通信(Inter-Process Communication)を分離する機能です。同じ名前空間に所属するプロセスどうしは互いに通信できますが、別の名前空間に所属するプロセスとは通信できなくなります。

### user namespace

少し遅れてユーザID(UID)とグループID(GID)の名前空間を分離するuser namespaceが実装されました。これにより隔離空間内で外部とは別のUIDとGIDを持つことができるようになります。また外部と内部でマッピングを指定することができます。

このユーザ名前空間の特徴は管理者権限の移譲にあります。Unix系のOSではUID 0はスーパーユーザ(root)と呼ばれ全特権を持っていますが、名前空間の内部でUID 0を割り当てられると管理者特権を行使できるようになります。この権限はあくまでも隔離空間の中でのみ有効で、何らかの手段で外部に脱出したとしてもそこでは特権は利用できません。

### pid namespace

続けてプロセスID空間を分離するpid namespaceが実装されました。この名前空間には親子関係があり、親の名前空間からは子供のプロセスを見ることができますが、子供から親のプロセスは見えません。そして子供の名前空間の

**▼表2　Namespaceの名前空間一覧**

| 名前 | カーネル | 分離する対象 |
|---|---|---|
| mount | 2.4.19 | ファイルシステムのマウント |
| uts | 2.6.19 | ホスト名、ドメイン名 |
| ipc | 2.6.19 | System V IPC、POSIX message queue |
| user | 2.6.23 | ユーザID、グループID |
| pid | 2.6.24 | プロセスID |
| net | 2.6.24 | ネットワークデバイス、プロトコルスタック |
| cgroup | 4.6 | CGROUP |

中ではプロセスID（PID）が1から順に振られ始めます。

Unixではプロセス1のプロセスはinitプロセスと呼ばれ、親が死亡して孤児になったプロセスを引き取って親になります。そしてinitが終了することでOS自体が停止します。プロセスID名前空間の中で最初に起動して内部でPID 1になるプロセスも、この孤児を引きとる役割が割り当てられています。そのプロセスが終了すると名前空間内にいる全プロセスが終了します。

**▼図2　実行中のシェルの名前空間の確認**

```
$ ls -l /proc/$$/ns
lrwxrwxrwx 1 root root 0 Nov 15 21:11 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 Nov 15 21:07 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 root root 0 Nov 15 21:07 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 root root 0 Nov 15 21:07 net -> net:[4026531969]
lrwxrwxrwx 1 root root 0 Nov 15 21:07 pid -> pid:[4026531836]
lrwxrwxrwx 1 root root 0 Nov 15 21:07 user -> user:[4026531837]
lrwxrwxrwx 1 root root 0 Nov 15 21:07 uts -> uts:[4026531838]
```

#### net namespace

net namespaceはネットワークデバイスを隔離します。個々のネットワークデバイスは1つの名前空間にしか所属できないため、子供に渡したデバイスは親からはアクセスできなくなります。ネットワーク名前空間では、IPv4やIPv6などのプロトコルスタック（IPアドレス、ポート番号、ルーティングテーブル、フィルタ情報など）も分離します。これにより同じ80番ポートをlistenしたり、コンテナごとに別のルーティングをしたりできます。

#### cgroup namespace

最後のcgroup namespaceは最近実装されたもので、後で説明するCGROUP機能を分離します。コンテナ内でさらにコンテナを実行したいような場合に使用します。

## namespaceの特徴と実装

Linuxの名前空間はプロセス単位で個別に指定することができます。全部の名前空間を分離してコンテナ型仮想サーバを作ることもできますし、一部だけを別の名前空間にすることもできます。

名前空間の操作には専用システムコールunshare()、setns()、およびclone()を使用します。新しく作られたプロセスは親プロセスと同じ名前空間に所属していますが、引き数で名前空間の種類を指定してunshare()システムコールを呼び出すことにより、新しい名前空間を作成してそこに移動できます。名前空間の中にいるプロセスがすべて終了すると、その名前空間は自動的に削除されます。

各プロセスが現在所属している名前空間は/proc/<プロセスID>/nsディレクトリ以下にシンボリックリンク形式で読み取ることができます。たとえば実行中のシェルの名前空間は、**図2**のようにして確認できます。

名前空間を使っていなければすべて数字がinit（/proc/1/ns）のものと一致しますが、名前空間が別になっていれば違う数字になります。psコマンドのオプションでも名前空間を確認できます。たとえば、uts namespaceならば次のようになります。

```
$ ps -o pid,utsns,args
  PID      UTSNS COMMAND
 5047 4026531838 bash
 5071 4026531838 ps -o pid,utsns,args
```

新しい名前空間ではなく既存の名前空間へ移動するにはsetns()システムコールを使用します。移動先は上記の/proc/<プロセスID>/ns/配下を指すファイルディスクリプタで指定します。

clone()はLinux版の多機能fork()で、スレッドを作成するなどいろいろな機能がありますが、新しい名前空間を作成してその中でプロセスやスレッドを作成することもできます。

以上のように名前空間はプロセスの属性とし

て実装されているため、カーネルから見れば個々のプロセスがinitなどほかのプロセスと同じ名前空間で実行されているか、別の名前空間で実行されているかの違いに過ぎません。そのため実行にあたってオーバーヘッドがほぼ存在しないのが大きな利点です。

## namespaceを試してみる

実際にnamespaceを試してみましょう。名前空間を作成するツールにはLXCやsystemd-nspawnやrunCなどいろいろとあるのですが、今回は最も単純なunshareコマンドを使ってみることにします。

unshareコマンドは先に説明したunshare()システムコールを叩く単純なツールで、dmesgやmountなどのLinux標準ツールを収容しているutil-linuxパッケージに含まれているため、ほとんどのLinuxディストリビューションで最初からインストールされています[注1]。

まずは単純なuts namespaceを試してみましょう。

```
# hostname
base
# unshare --uts
  ↑uts namespaceを作成してその中でシェルを実行
# hostname
base  ←最初は外部のホスト名を引き継いでいる
# hostname in-container
  ↑名前空間の中でホスト名の変更
# hostname
in-container  ←ホスト名が変わっているけど
# exit  ←名前空間を抜けると
# hostname
base  ←変わっていない
#
```

lsnsコマンドが使えれば、次のように名前空間の一覧が確認できます[注2]。lsnsが使えない場合は前述のpsのオプションなどで対応してください。

```
# unshare --uts
# echo $$
24540
# lsns
        NS TYPE  NPROCS   PID USER     COMMAND
4026531836 pid      301     1 root     init [3]
4026531837 user     300     1 root     init [3]
4026531838 uts      302     1 root     init [3]
4026531839 ipc      305     1 root     init [3]
4026531840 mnt      304     1 root     init [3]
4026531857 mnt        1    47 root     kdevtmpfs
4026531957 net      300     1 root     init [3]
4026532478 uts        2 24540 root     /bin/bash
          ↑新しくできた
# exit
#
```

加えてuser namespaceも分離してみましょう。権限がわかりやすいよう今回は一般ユーザから実行し、--map-root-userをオプションにつけて起動したユーザを、コンテナ内でrootにマッピングしてみます。

```
$ hostname
base
$ id
uid=1001(guest) gid=1000(guest)
$ unshare --uts --user --map-root-user
  ↑名前空間を作成してUID 0にマップ
# id  ←プロンプトが # に変わっている
uid=0(root) gid=0(root)  ←rootになっている
# cat /etc/shadow  ←でも外部ではrootではないので
cat: /etc/shadow: Permission denied
  ↑権限のないファイルは読むことはできない
# hostname in-container  ←ホスト名の変更はできる
# hostname
in-container
# exit
$
```

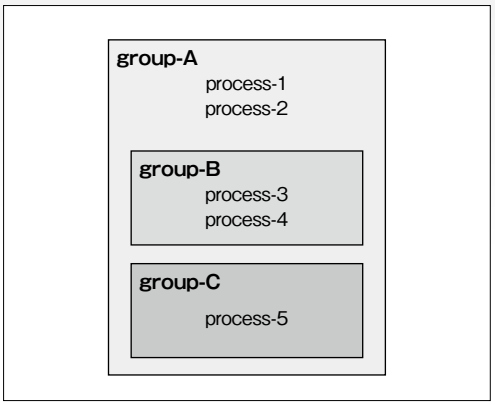
uts namespaceが別になっているため、変更されるのはコンテナ内部のホスト名のみなので内部のroot権限で変更できるというわけです。--map-root-userオプションを使用する代わりに、親の名前空間から/proc/<プロセスID>/uid_mapや/proc/<プロセスID>/gid_mapに「内部のID　外部ID　個数」という形式で書き込み、直接マッピング情報を指定することもできます。

注1）コンテナまわりの機能やツールは最近になってどんどん追加／拡張されたものなので、使用しているディストリビューションによっては採用しているカーネルやutil-linuxのバージョンが古く、すべての機能が使えない可能性もあります。
注2）新しいutil-linuxが導入されているUbuntu 16.10以降、Fedora 24以降などが必要です。

▼表3　CGROUPのコントローラー覧

| 名前 | カーネル | 機能 |
|---|---|---|
| cpuacct | 2.6.24 | CPUの利用時間をレポートする |
| cpu | 2.6.24 | CPUの利用時間を重み付きで配分したり上限設定する |
| cpuset | 2.6.24 | 使用するCPUコアとNUMAメモリを指定する |
| memory | 2.6.25 | メモリやスワップ使用量を制限したりOOM Killerの動作を制御する |
| device | 2.6.26 | デバイスファイルの書き込み、読み込み、作成などの操作を禁止する |
| freezer | 2.6.28 | プロセスをまとめて一時停止したり再開する |
| net_cls | 2.6.29 | ネットワークパケットにラベルを付けて、パケットフィルタをしたり、トラフィック制御(tc)で帯域制御したりする |
| blkio | 2.6.33 | ブロックデバイス(ディスク)に対するI/Oを重み付きで配分したり上限設定する |
| perf_event | 2.6.39 | perfコマンドを使用してコンテナのパフォーマンスモニタリングする |
| net_prio | 3.3 | ネットワークパケットに優先度を付加して、トラフィック制御(tc)で送出順指定や帯域制御する |
| hugetlb | 3.5 | 連続した巨大メモリ領域(Huge Page)の確保量を制限する |
| pids | 4.3 | プロセス数の上限を設定する。最近実装された機能で新しいカーネルでのみ利用可能 |

▼図3　cgroupによるグループの階層化例




## リソース管理機能：CGROUP

次はコンテナ機能のもう1つの重要な要素CGROUP(control group)について見てみましょう。Namespaceがプロセスを隔離する機能だったのに対して、CGROUPはプロセスのリソースを管理するための機能です。

CGROUPにはLinux 2.6.24から実装が開始されたバージョン1と、Linux 3.10から実装が開始されたバージョン2がありますが、今のところほとんどのディストリビューションではバージョン1を使用していますので、本稿ではそちらで説明します。

cgroupの機能はコントローラとかサブグループと呼ばれる複数のリソース管理機能からなっています。コントローラには**表3**のようなものが存在しています。

### cgroupの特徴と実装

cgroupはプロセスをグループ化する機能と、そのグループごとにリソースを割り当てたり、利用状況をレポートしたりする機能からなっています。

cgroupを使うと自由にプロセスを集めてグループ化することができ、任意のタイミングでグループにプロセスを追加したり削除したりできます。さらにグループの中にグループを作って、グループを階層化することができます(**図3**)。内のグループに所属しているプロセスは、同時に外のグループにも所属しているとみなされます。

このような階層化された構造を扱うのにぴったりのしくみがUnixにはあります。ファイルシステムです。cgroupのインターフェースはシステムコールではなく/procや/sysなどと同じような仮想ファイルシステムとして実装されており、次のファイルシステムへのアクセス手段で操作することができます。

▼図4　/proc/cgroupsの表示例

```
$ cat /proc/cgroups
#subsys_name    hierarchy       num_cgroups     enabled
cpuset          6               1               1
cpu             5               86              1
cpuacct         5               86              1
blkio           11              86              1
memory          9               202             1
devices         10              86              1
freezer         3               1               1
net_cls         4               1               1
perf_event      2               1               1
net_prio        4               1               1
hugetlb         8               1               1
pids            7               87              1
$
```

▼図5　/sys/fs/cgroups以下へのmount例

```
# mount -t tmpfs tmpfs /sys/fs/cgroup
#
# mkdir /sys/fs/cgroup/cpu
# mount -t cgroup -o cpu cgroup /sys/fs/cgroup/cpu
#
# mkdir /sys/fs/cgroup/cpuset
# mount -t cgroup -o cpuset cgroup /sys/fs/➥
cgroup/cpuset
...略...
```

・グループの作成 ⇒ ディレクトリを作成
・グループの削除 ⇒ ディレクトリを削除
・プロセスの移動 ⇒ 特殊ファイルへの書き込み
・設定の書き込み ⇒ 特殊ファイルへの書き込み
・情報の読み出し ⇒ 特殊ファイルの読み出し

cgroupはコンテナ以外でも任意のプロセスの集まりに対してリソースの管理ができるようになっており、デーモンやKVMの仮想サーバのリソースを制限するなど多目的に使えます。

## cgroupを試してみる

それではcgroupを実際に使ってみましょう。cgroupの使用には特別なツールは必要ありません。対応しているLinuxカーネルがあれば利用できます。カーネルがどのcgroupをサポートしているかは/proc/cgroupsを見れば確認できます(**図4**)。

cgroup操作用の仮想ファイルシステムは/sys/fs/cgroups以下にマウントする決まりになっています。最近のディストリビューションならば最初からマウントされているはずですが、なければ**図5**のような感じでマウントしてください。通常はこのように、コントローラごとに個別のグループを管理できるよう別々にマウントします。

▼図6　個々のプロセスがどのcgroupに所属しているかの確認例

```
$ cat /proc/$$/cgroup
11:blkio:/user.slice
10:devices:/user.slice
9:memory:/user.slice
8:hugetlb:/
7:pids:/user.slice/user-1000.slice
6:cpuset:/
5:cpu,cpuacct:/user.slice
4:net_cls,net_prio:/
3:freezer:/
2:perf_event:/
1:name=systemd:/user.slice/user-1000.➥
slice/session-c2.scope
$
```

/proc/<プロセスID>/cgroupを見れば個々のプロセスが、現在どのcgroupに所属しているか確認できます(**図6**)。

比較的単純なcpusetを実際に使ってみましょう。これは複数のCPUコアがある機材でプロセスの使用可能なCPUコアを制限できます。

次のようにすると、このシェルおよび子孫のプロセスはcpu 0、1、2、3の4個のみを使用するようになります。

```
# cd /sys/fs/cgroup/cpuset
# mkdir test
  ↑グループtestをcpusetに作成
# echo 0-3 > test/cpuset.cpus
  ↑CPUコア0-3の4個に設定
# echo 0 > test/cpuset.mems
  ↑メモリをNUMA 0に設定
# grep cpuset /proc/$$/cgroup
6:cpuset:/              ←今シェルは / にいる
# echo $$ > test/cgroup.procs
  ↑このシェルをtestに移動
# grep cpuset /proc/$$/cgroup
6:cpuset:/test          ←確かにtestに移動している
# cat test/cgroup.procs
15976                   ←シェルとプロセスID
16177                   ←catのプロセスID
#
```

OpenSSLを使って暗号化速度のベンチマークをしてみましょう。

```
# openssl speed rsa -multi 4 2>&1 | tail -5
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000012s 0.000001s  86956.5 1083333.3
rsa 1024 bits 0.000034s 0.000002s  29795.5  422222.2
rsa 2048 bits 0.000231s 0.000007s   4324.3  146825.4
rsa 4096 bits 0.001622s 0.000025s    616.5   39506.9
```

ここで、使用可能なCPUをcpu 0の1個に減らして同じ計測を実施してみます。

```
# echo 0 > test/cpuset.cpus  ←CPUコア0の1個だけに設定
# openssl speed rsa -multi 4 2>&1 | tail -5
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000046s 0.000003s  21918.0 285714.3
rsa 1024 bits 0.000133s 0.000009s   7543.6 112166.3
rsa 2048 bits 0.000912s 0.000027s   1096.4  36867.1
rsa 4096 bits 0.006431s 0.000100s    155.5  10031.5
```

上下の結果を比較してみれば、同じ4並列でもCPUが1個しか使えないため、速度が4分の1になっているのがわかります。

グループの削除はディレクトリを消すだけですが、その前にプロセスを空にする必要があります(**図7**)。

## Dockerの登場

コンテナはNamespaceとCGROUPというLinuxカーネルの機能であり、新しいカーネルさえあれば使えることがわかりました。それでは評判のDockerとは何でしょうか？

一言で答えると「Dockerはコンテナイメージの管理ツール」です。yumとかaptのようなパッケージ管理ツールの親戚と思えばわかりやすいかもしれません。Docker自体は直接コンテナを操作せず、昔のDockerでは内部でLXCを呼んでいました。最近のバージョンではコンテナ操作にrunCを使用しています。

コンテナの世界にDockerのもたらしたものは次のようになります。

### 1. コンテナのイメージ形式

コンテナには速いだけでなく、イメージのサイズが小さくて済むという利点があります。Dockerが標準のイメージ形式を決めたことにより、別のサーバへコンテナを持ち運んだり、複製、保管、共有するなどポータビリティの高いシステムが実現できます。それまではコンテナ技術に詳しくないと使えませんでしたが、Dockerによりほかの人の作成したイメージを使って誰でも簡単にコンテナが使えるようになりました。

### 2. イメージの作成ツール

Dockerはイメージの作成も簡単にしました。既存のイメージを入手してきて改変してセーブすれば、誰でも簡単に新しいコンテナイメージを作成できます。Dockerイメージは差分形式で保存できるため、イメージサイズも小さくできます。

### 3. Docker Hub

そしてこれらのコンテナイメージをネット上で交換したり共有したりできるように、Docker Hubというイメージ管理サービスを提供しています。このDocker Hubを中心としたコンテナイメージ管理のエコシステムを持っていることが、Dockerの最大の特徴です。

◆ ◆ ◆

このようにLinuxカーネルによって柔軟なコンテナ環境が使えるようになったところへDockerが登場したことで、コンテナ利用の障壁が下がり、コンテナの普及が爆発的に広まるきっかけとなりました。SD

**▼図7　グループの削除**

```
# cat test/cgroup.procs
3420 3599                    ←中にシェルとcatがいる
# rmdir test
rmdir: failed to remove 'test': Device or resource busy
  ↑消せない
# echo $$ > cgroup.procs     ←シェルを親グループへ移動
# cat test/cgroup.procs      ←testが空なのを確認
# rmdir test                 ←今度は消せる
#
```

第7章 runC、Swarmモード、Dockerストア……

# Dockerの最新動向を知る

Dockerは2011年3月に公開されて以来、バージョンアップを重ねるごとに機能の追加や拡張が行われています。本章では、ここ1年でのアップデート情報と、覚えておくべき重要なトピックをまとめました。


**Author**
前佛 雅人(ぜんぶつ まさひと)
さくらインターネット㈱
**Twitter**
@zembutsu

## ネットワーク機能の正式化と機能追加

コンテナのリンク機能(コンテナ実行時に--linkオプションを使用)は、コンテナのポートを開くことなく、コンテナ間を1対1で名前解決し、接続できます。しかし、複数のコンテナを接続したい場合や、動的にコンテナ数やコンテナ名が変わる場合には有用な機能ではありませんでした。

そこで、v1.9からはdocker networkコマンドが正式導入され、コンテナのネットワークが強化されました。Dockerホスト上で複数のブリッジ・ネットワークが扱えるようになり、コンテナは複数の仮想インターフェースを扱えるようになりました。つまり、コンテナが複数のDocker内部ネットワークを利用できるようになったのです。また、ホスト側とネットワーク・インターフェースを共有するhostネットワーク、インターフェースを持たないnoneネットワーク、複数のホスト間をつなげるoverlayネットワークの概念も導入されています。

しかし、従来の名前解決はコンテナ起動時に/etc/hostsを変更していたため、動的な環境には適しませんでした。そこでv1.10ではネットワーク機能が強化され、Docker Engine(Dockerデーモン)に、内部で名前解決するためのDNSサーバが内蔵されました。そして、ネットワークにコンテナを追加・削除しても動的な名前解決ができるようになりました。また、任意のCIDRを持つネットワークの作成や、コンテナに対するIPアドレスの割り当てもできるようになり、複数のコンテナを扱ううえでの利便性が高まりました。またv1.11からは、内部DNSのラウンドロビンにも対応しています。

## ランタイムrunCの導入

外見上大きく変わったのが、Docker 1.11からのrunC導入です。runCはコンテナ標準化団体のOCI[注1]の規格に則ったライブラリであり、Linuxカーネルのコンテナに関する各種操作を行います。このライブラリを扱うために、v1.11からはcontainerdデーモンが導入されています。

v1.11までは、docker daemonコマンドを指定すると、dockerバイナリがデーモンとして稼働していました。v1.12からはcontainerdの機能をdockerデーモンに統合した、新しいdockerdデーモンが導入されました(**表1**)。Dockerエンジンを起動・停止する手順が変更になっただけでなく、監視対象も変更になったため、運用時の手順見直しや、監視対象のデーモンの変更が

注1) Open Container Initiative URL https://www.opencontainers.org/

必要になっています。なお、runCはDocker社のプロダクトでしたが、2015年にOCI(Open Container Initiative)へ寄贈されました。そしてcontainerdも、2016年12月に独立したプロジェクトへの移行が発表されています。

## クラスタを管理する新しいSwarmモード

### 既存機能「Docker Swarm」との違い

英語のswarm(スウォーム)とは「群れ」の意味です。Dockerにおいては、Docker Engineが動作するホストが相互に通信可能なクラスタを形成している状態を「swarm」と呼びます。これまでのDockerは、Docker Engineをリモート管理するしくみとしてDocker Swarmを開発、提供していました。これは複数のDocker Engineが動作するホストをクラスタに束ね、1つのリソース・プールと見立ててコンテナを実行・管理します。Swarmはストラテジとフィルタによって、コンテナをどのように実行するかルールを決められます。そのため、ホストを意識しなくてもコンテナを実行できるのが利点です。

Docker v1.12からは、新しいSwarmモードが追加されました。この「Swarmモード」と従来の「Docker Swarm」は、どちらも「Swarm」の名前を冠しています。しかし、機能(**表2**)もクラスタの構成のしかた(**図1**)も異なります。クラスタ形成に必要なのはDocker Engineが動くサーバだけであり、Docker Swarmに必要なマネージャやディスカバリ・バックエンドは不要です。

### Swarmモードの開発の背景

Dockerはそもそも、アプリケーションを簡単に開発、移動、実行するためのプラットフォームとして開発がスタートしました。ですが、クラスタ環境でDockerを動かそうとすると、複雑なDocker Swarmのセットアップや、キーバリュー・ストアなどの外部ツールの導入や管理が必要となります。また可用性を考慮すると、サーバ台数も増え、複雑な構成になってしまいます。

このような状況はDockerの設計思想とは相容れないため、Swarmモードの開発に至りました。

### クラスタの手軽な管理と新機能を提供

Swarmモードでクラスタを作成するには、外部のツールを必要としません。新しく導入された`docker swarm`コマンドでクラスタを形成します。すると標準のDocker Engineだけで、複数のノードでクラスタ管理・コンテナ実行をする機能や、複数のSwarmマネージャでデータを複製する機能を実現できます。さらにDocker Engine間の通信は、デフォルトでTLS暗号化によって保護されるため、セキュリティも高まっています。

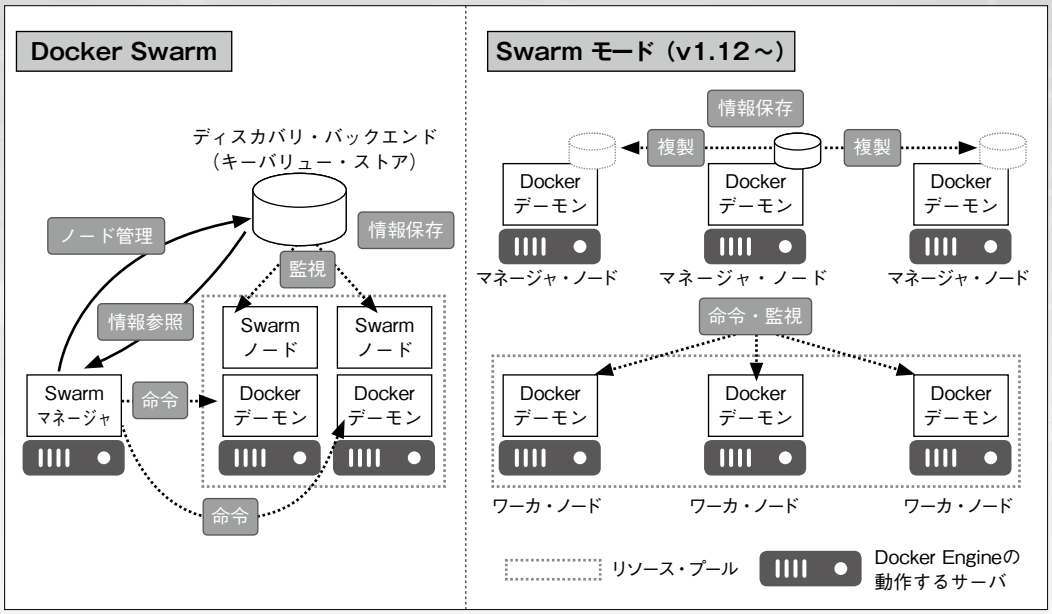
**▼表1　Dockerデーモンとランタイムの変遷**

| Dockerバージョン | ～0.8 | 0.9～1.10 | 1.11 | 1.12 |
|---|---|---|---|---|
| コンテナ用ランタイム(ランタイム用デーモン) | LXC | libcontainer | runC(containerd) | |
| Dockerデーモン | docker daemon | | | dockerd |

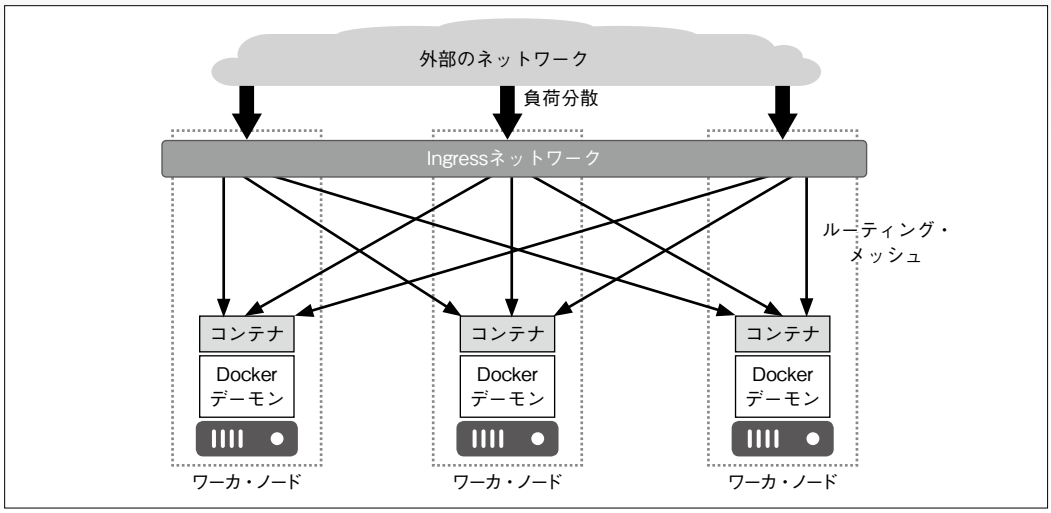
**▼表2　イメージ操作コマンド**

| | Swarmモード | Docker Swarm |
|---|---|---|
| クラスタの管理 | Dockerエンジン内蔵コマンド(docker node)で管理 | Swarmマネージャが Swarmノードを管理 |
| クラスタ上のコンテナ操作 | docker serviceコマンド | SwarmマネージャのAPIにアクセス |
| Swarmマネージャの追加 | 不要 | 必要 |
| KVSのセットアップ | 不要 | 必要 |
| クラスタ間通信の暗号化 | デフォルトで有効 | 別途設定が必要 |

▼図1　Docker SwarmとSwarmモードのクラスタ比較




▼図2　ルーティング・メッシュとIngressロードバランシング




コンテナを管理する新しい概念として、サービスとタスクが導入されました。コンテナを実行するにはサービスを定義します。たとえばnginxコンテナを3つ実行し、ポート80番を公開するには次のように実行します。

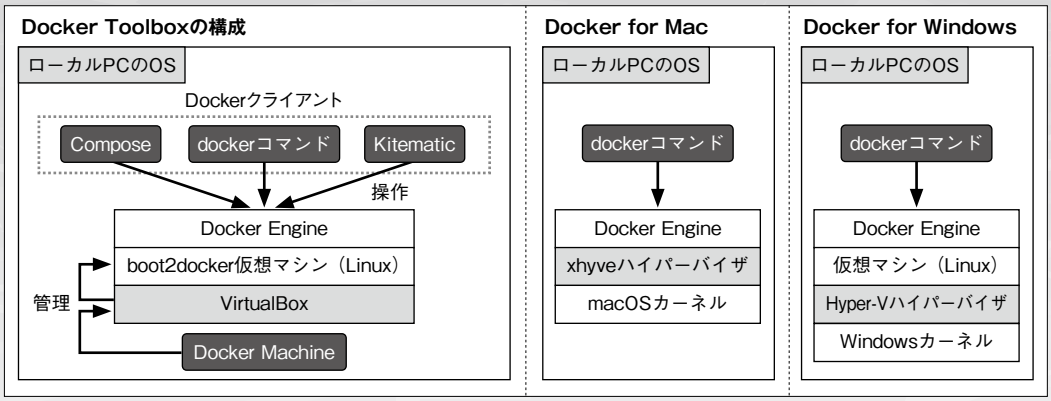
```
$ docker service create --replicas 3 -p
80:80 nginx:latest
```

これはnginxコンテナを3つ使うサービスを作成(create)する命令であり、期待する状態を定義しています。そして、これを達成するため、クラスタ上のホストのいずれかで、コンテナを実行するタスク(としてのコンテナ)が3つ実行されます。

またSwarmモードは、Docker Swarmよりもクラスタが簡単に組めるというだけではありません。新しい機能も追加されています(**図2**)。

Swarmクラスタを形成すると、自動的にクラ

▼図3




スタ内部で通信可能なIngress（イングレス）という名前のoverlayネットワークを作成します。そして、Swarmホストに外部のネットワークから通信があるとき、ホスト上に対象のコンテナ（のサービス）が動作していなければ、適切なホストにIngressネットワークを通してルーティングします（ルーティング・メッシュ機能）。

また、ルーティング先に複数のコンテナが動作している場合は、負荷分散機能が有効になります（Ingressロードバランシング）。そのため、どのホスト上でコンテナを実行しているか意識することなく、スケールアウトやスケールインが容易に行えるようになります。

Swarmモードの詳細は、公式ドキュメント[注2]や筆者の日本語訳[注3]をご覧ください。

## 今後のDocker Swarm

便利なSwarmモードが登場したからといって、これまでのDocker Swarmが急になくなるわけではありません。商用サポート版のDocker Engine 1.12未満では、Docker Swarmのサポートが今も続いています。また、GitHub上のリポジトリでも開発は継続していますが、積極的なバージョンアップは2016年夏以降行われていません。機能がSwarmモードと競合していることもあるため、今後の大きな機能改善は望めないものと思われます。

注2）URL https://docs.docker.com/engine/swarm/
注3）URL http://docs.docker.jp/engine/swarm/

## その他のトピック

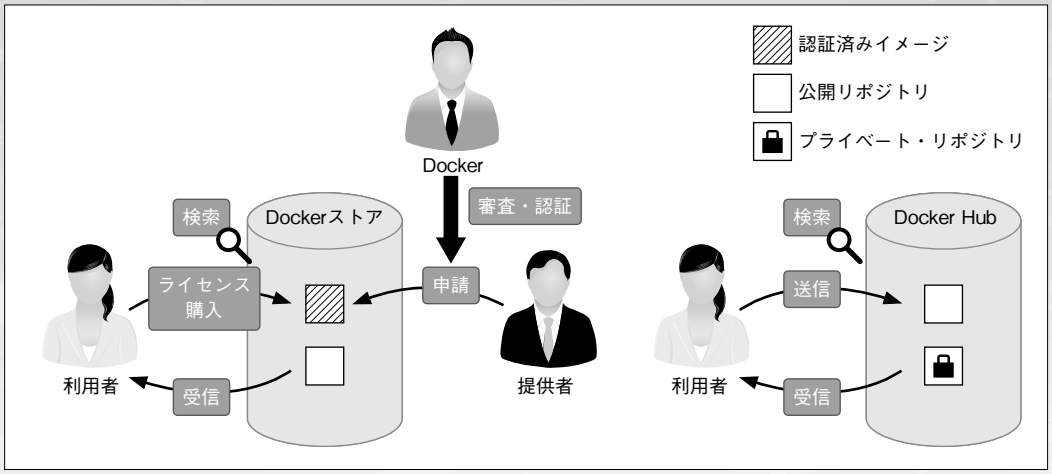
### Docker for Mac/Windows

Docker for MacとDocker for Windowsは、ローカル開発環境向けの環境を提供します。これまで提供されていたDocker Toolboxと同じような機能ですが、Docker for Mac/Windowsともにローカルでしか利用できません。用途に応じた使い分けが求められます（**図3**）。

まずDocker Toolboxの説明ですが、これはDocker Machine、Docker Compose、dockerクライアント、VirtualBoxをひとまとめにして提供しています。Docker MachineはVirtualBox上にboot2dockerという名前のLinuxディストリビューションをセットアップし、その上で動作するDockerデーモンをリモートから操作できるようにします。Docker Machineの管理できる対象はローカルPCの仮想化環境上だけではありません。クラウド上のAPIを使うことで、リモート環境上で仮想サーバの起動とDockerのセットアップを、`docker-machine`コマンドを通して管理できます。

Docker for Macは、macOS上のネイティブ・アプリケーションです。そして、`docker`コマンドや`docker-compose`コマンドも利用できます。

▼図4　DockerストアとDocker Hubの比較




一見すると同じように見えますが、Dockerを動かすためにVirtualBoxは不要です。通常のDockerはLinuxカーネルの機能を使うために、何らかのLinuxディストリビューションを必要とします。ですが、Docker for MacではmacOS 10.10 Yosemite以降で利用可能となった、Hypervisor.framework(xhyve)を使います。ローカルでの開発に限った用途であれば、Docker Machineよりも使い勝手が良いでしょう。

そして、Docker for Windowsの提供も始まりました。Hyper-V上にLinux仮想マシンを作成し、そこでDocker Engineを動かします。Docker Toolboxとは異なり、VirtualBoxで仮想マシンの作成や削除が不要です。コマンドラインやパワーシェルで直接dockerを操作できるようになります。利用するにはWindows 10以上で、かつ、Hyper-Vが動作するライセンス(Pro、Enterprise、Education)が必要です。注意点としては、Hyper-Vを有効化するとVirtualBoxが使えなくなることです。Docker MachineやVagrantなど、VirtualBoxご利用時にはご注意ください。

## Dockerストアの提供開始

Dockerストア[注4]とは、Dockerイメージの「マーケットプレイス」です。Dockerストアを通して各種の公式イメージや、商用サポート版のイメージが配布されています。2016年6月にプライベート・ベータ版の提供が始まり、現在は誰でも利用できます。

Dockerストアでイメージを検索すると、Dockerの確認済みイメージだけでなく、Docker Hub上のイメージも検索できます。Dockerストアで配布されているイメージは2種類です。

- **Docker Verified images**
  Docker社が確認済みのイメージであり、セキュリティやノウハウを適用済み。これまでの公式イメージの位置付け
- **Community/Hub images**
  Docker Hub上のイメージであり、Docker社は未確認

従来のDocker Hubで配布されていた「公式イメージ」は、Dockerストアを通してでも確認できるようになりました。それだけでなく、商用ライセンス(サブスクリプション)も購入できます。そのため、利用者はDockerストアでライセンスを購入するだけで、すぐに商用アプリケーションを利用できます。また、イメージの提供者は、自分で決済システムを持たなくとも手軽にイメージを配布できるようになりました(**図4**)。

さらに、ストアに登録されている確認済みイ

注4) URL https://store.docker.com/

▼表3　CS版Docker Engineの対応表

| Dockerバージョン | | 1.10 | 1.11 | 1.20 |
|---|---|---|---|---|
| ディストリビューション | Red Hat Enterprise Linux | 7.0, 7.1 | 7.0, 7.1, 7.2 | |
| | CentOS | 7.1-1503 | 7.1-1503, 7.2-1511 | |
| | Ubuntu | 14.04 LTS | | |
| | SUSE Linux Enterprise Server | 12 | | |
| Dockerストレージ・ドライバ | Red Hat Enterprise Linux | devicemapper | | |
| | CentOS | devicemapper | | |
| | Ubuntu | aufs3 | | |
| | SUSE Linux Enterprise Server | btrfs | | |

メージは検索機能も充実しています。たとえば、Linux用かWindows用か、アプリケーションやデータベースなどのカテゴリでも検索できます。

## 商用Dockerエンジンのバージョンごとの差違

Docker 1.9以降、ネットワークやSwarmモードのサポートなど、大きな変更を伴うバージョンアップが続いています。オープンソース版のDockerは多くのディストリビューションで動作します。一方で、CS(商用サポート)版を使う場合は、バージョンごとにサポート対象のディストリビューションが利用できるストレージドライバが異なります(**表3**)。

今後もDockerのバージョンが変わる場合は、都度、Docker社のサイト[注5]で対応情報の確認が必要です。

## Docker Datacenter

Dockerが提供を始めたツール群の総称で、DTR(Dockerトラステッド・レジストリ)とUCP(ユニバーサル・コントロール・プレーン)がセットになり、ローカル環境もしくはプライベートな環境でDocker Hubと同等のリポジトリ機能、ユーザ認証機能を提供します。従来はイメージを保管するリポジトリとしてのDTRのみの提供でした。UCPを使えば、任意のインフラ上でブラウザのGUIを通したイメージ管理が可能になります。

また、イメージやコンテナなどのリソースのアクセス権限を、ユーザとチームごとのロール(役割)設定に応じて設定できます。コンテナに対するアクセス権限を開発チームと運用チームで分けることも可能であり、誤操作の防止やセキュリティ向上という利点があります。

## Windows ServerのDocker対応

Windows Server 2016から、Windowsに対応したDockerデーモンでDocker Windowsコンテナが実行可能になりました。これはLinuxコンテナと同じように、Windowsのプロセスに対して名前空間やファイルを分離する機能です。ただし、あくまでもWindowsアプリケーションをコンテナとして実行できる機能であり、Linux用のイメージは実行できません。

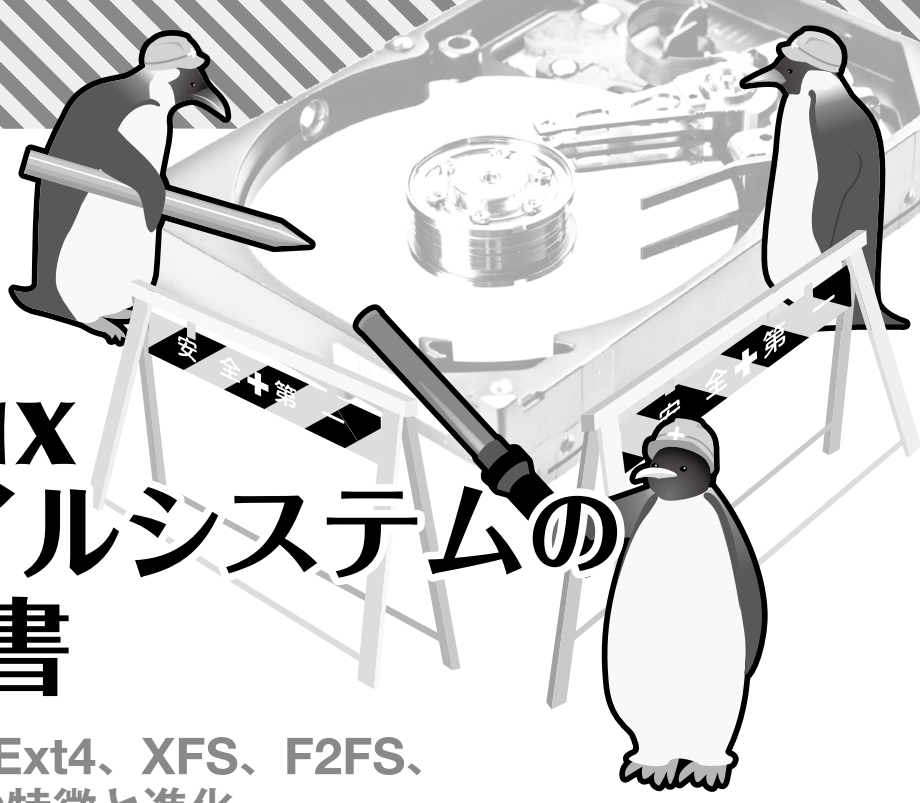
## まとめ

直近の1年を振り返るだけでも、多くの機能やツールの発表がありました。より新しい情報を参照されたい場合には、Dockerのブログ[注6]とDockerのTwitterアカウント(@docker)が参考になります。また、新機能に対するドキュメント[注7]も充実していますので、併せてご覧ください。SD

注5) Compatibility Matrix URL https://success.docker.com/Policies/Compatibility_Matrix

注6) URL https://blog.docker.com/
注7) URL https://docs.docker.com/

第2特集



# Linux ファイルシステムの教科書

## Ext3、Ext4、XFS、F2FS、Btrfsの特徴と進化

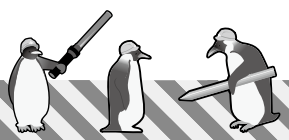
パソコンはよく「ソフトがなければただの箱」と言われます。それらのソフトは、メモリに読み込まれCPUによって実行されますが、そのソフトやソフトが扱うデータが保存されているのはHDDやSSDなどの記憶装置です。これらの記憶装置を使いやすく抽象化するのがファイルシステムの役割です。

ファイルシステムによって、データに名前を付けて、ディレクトリ（Windowsではフォルダ）に保存し、あとから読み出すことができます。こういったファイルシステムはいったいどのように動いているのでしょうか。

本特集では、Linuxのファイルシステム（Ext3、Ext4、XFS、F2FS、Btrfs）の特徴としくみや、トランザクション単位で問題解決するジャーナリングファイルシステムについて詳しく解説します。

Author 青田 直大（あおた なおひろ）






# いろいろなファイルシステムの特徴とファイルの整合性を保護する機能

本章では、Linuxのさまざまなファイルシステムがどのようにデータをディスクに保存し、そしてデータが壊れないように保護しているのかを見ていきます。

Author 青田 直大(あおた なおひろ)

## Linuxのファイルシステム

Linuxにはさまざまなファイルシステムが実装されています。Ext4やXFSなど有名なもの、BtrfsやF2FSなど比較的近年に追加されたもの、UBIFSといったちょっと珍しいものなど総勢60種類ものファイルシステムが実装されています。

60種のファイルシステムは、

①ローカルファイルシステム
②ネットワークファイルシステム
③特殊ファイルシステム

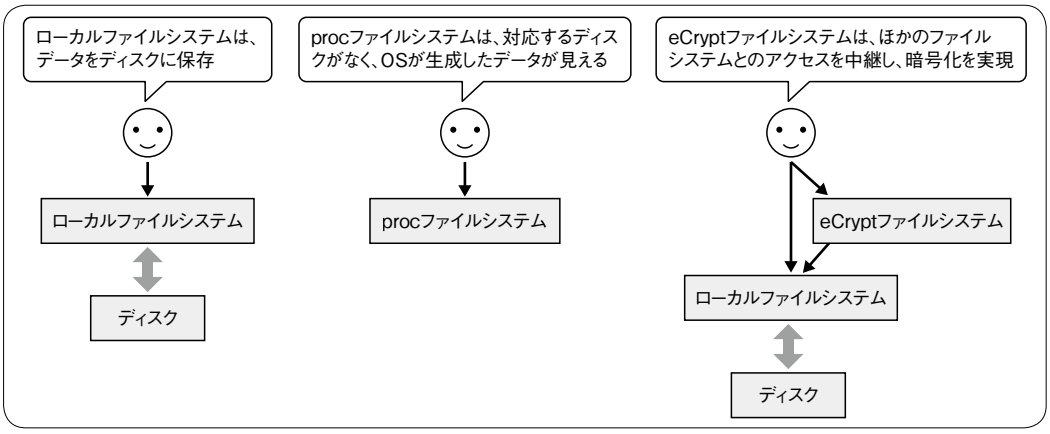
と大きく3つのカテゴリに分けることができます。①のローカルファイルシステムは、いわゆる普通のファイルシステムで、HDD(やSSD)にデータを保存するために使われるもののことです。よく使われるものとして、Ext4やXFSがあります。

②のネットワークファイルシステムは、Windowsの共有フォルダのようにネットワーク上で、あるマシンから別のマシンのファイルへのアクセスを提供するものです。たとえばNFSや、少し構成は異なりますがクラスタファイルシステムのCeph[注1]というものが、Linux上のネットワークファイルシステムとして知られています。

③の特殊なファイルシステムには、カーネルとのやりとりに使われるものや、ほかのファイルシステムに付加的な機能をつけるものがあります(**図1**)。前者には、たとえばプロセスの情

注1) http://ceph.com

**▼図1 特殊なファイルシステム**



報を取得し/procにマウントされるprocファイルシステムがあります。後者にはたとえば、ユーザとほかのファイルシステムとのデータを中継して、暗号化・復号を行うeCryptファイルシステム[注2]があります。

60種のうち、Linuxのローカルファイルシステムは、特殊なデバイス用のものや書き込み機能が実装されていないものを除いても18種類あります。その中でもExt2、Ext4、Reiserfs、JFS、XFS、Btrfs、NILFS2、F2FSの8個がとくにLinuxネイティブのファイルシステムとして知られています。

ほかのOS、たとえばWindowsでは公式にサポートするローカルファイルシステムはNTFSとFATの2つです。Linuxと同じUNIX系のFreeBSDにしても、UFS2とZFSの2つがネイティブサポートとされています。このようにLinuxはほかのOSと比べて、ネイティブサポートのファイルシステムが多く、さまざまな特徴を持ったファイルシステムを比べていくことができます。

## ファイルシステムの特徴とは？

本特集ではLinuxのファイルシステムから、Ext3、Ext4、XFS、F2FS、Btrfsを紹介します。さまざまなファイルシステムを比較するにあたって、どこに注目するとよいでしょうか。多くのファイルシステムの比較では、扱えるファイルのサイズや、ファイルシステム全体の最大サイズ、あるいは透過的圧縮機能など機能の違いを並べていることが多いように思います。しかし、これらの機能的差異の多くは、その実装の違いによって生まれてくるものです。本特集では、ファイルシステムの作りを解説し、そこから機能の違いを見ていくボトムアップのアプローチをとります。

ボトムアップでファイルシステムを見ていくにあたって、まずファイルシステムの構成要件を考えましょう。冒頭にあったようにファイルシステムとは「データに名前をつけて、アクセスできるようにするもの」です。ここで、より正確に、Linuxにおいてローカルファイルシステムを実現するための必須要素とは何かを見ていきます。

### データの保存

ファイルシステムの最も基本的要件、それは「データを保存すること」でしょう。ファイルシステムはユーザが指定したデータを、ディスクのどこに保存するかを決める必要があります。当然ですが、このときほかのファイルに割り当てられている領域を使うわけにはいかないので、ディスクのどこが空いているのかを管理する情報を保持する必要があります。ファイルシステムは、ディスクを一定のサイズ(一般に4KB)の領域(**ブロック**)に区切り、その単位で領域を管理しています。

### ファイルの管理

データを保存しただけでは、あとから読み出すことができません。そのファイルのデータがどこにあるのかを記録していなければいけません。また、ほかにもファイルのサイズ・所有者・アクセス権限・タイムスタンプなどが管理情報として記録されます。また、**ファイルタイプ**という情報も記録されています。これはあるファイルが、データを保存する**通常ファイル**なのか、ディレクトリなのか、はたまたそのほかの特殊なファイルなのかを表現するものです。これらの管理情報のことを、ファイルの**inode**と呼びます。inodeは、**inode番号**というIDで識別されます(以降、inode番号がXXXである、inodeのことをinode #XXXと表記します)。

### ファイルデータのマッピング

inodeの情報のうち、ほとんどはデータサイズが小さいので、まとめて保存できます。しかし、

注2) http://ecryptfs.org

データの保存アドレスの表現には工夫が必要です。理想的にはディスク上のアドレスと長さで1つの領域を指定できればよいですが、現実的にはさまざまなファイルが同時に更新され、追記されていく中で、たとえば100GBの連続領域をいつも確保できるとは限りません。したがって「ファイル上のこの領域は、ディスクのこの領域に対応する」という情報(マッピング)を保存することになります。さまざまなサイズのファイルについて、マッピングを効率的に保存する方法が必要となります。

## ファイルの階層と名前

Linuxのファイルシステムは階層構造を持ちます。すなわちディレクトリは、その中にどのファイルがどのような名前で入っているかのデータを管理しています。当たり前のようですが、ディレクトリに対して、その中のファイルの一覧を取得するAPIがあります。大量のファイルを持つディレクトリの場合、一度のAPIですべてのファイル一覧を取得することはできず、どこまで読んだかを覚えておき、次のAPIで続きを取得していきます。

◆ ◆ ◆

まとめるとLinuxのローカルファイルシステムを実装しようと思うと、次の要素をどのようにディスク上で表現するかを決める必要があります(**図2**)。

・データ領域の使用状況の管理情報
・inode番号の使用状況の管理情報
・inodeの保存方法
・ファイル内アドレスと、物理アドレスとの変換情報(マッピング)
・ディレクトリ内のファイルの表現

## 整合性の保護

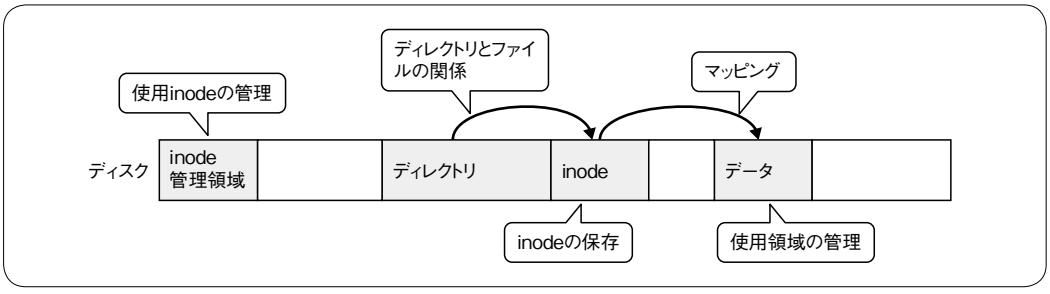
ときにファイルシステムは壊れます。コンピュータを起動するときに、"fsck"というものが実行されているのを見たことが、一度はあるのではないでしょうか。これはファイルシステムが「壊れて」いないかどうかをチェックし、場合によっては修復するツールです。そもそもファイルシステムとは、どのようにして壊れるのでしょうか。

前述したようにファイルシステムは、さまざまな管理情報をディスクに保存します。それらのデータはディスク上の連続しない離れた場所に保存されることがあります。その場合、OSのクラッシュやコンピュータの電源喪失などにより、更新の一部だけがディスクに書き込まれ、ファイルシステムの構造が壊れてしまうことがあります。

具体的に、新しくファイルXをディレクトリDの下に作り、そこに"ABC"と書いたときのファイルシステムの動きを見てみましょう。前項で見たように、

①"ABC"と書くブロックを決定し、確保
②使用するinodeの確保
③inodeの管理情報を設定

**▼図2　ファイルシステムの構成要件**



④inodeから“ABC”へのマッピングを追加
⑤ディレクトリDに、名前Xで、確保したinodeを指すエントリを追加

と5つの操作が行われます。

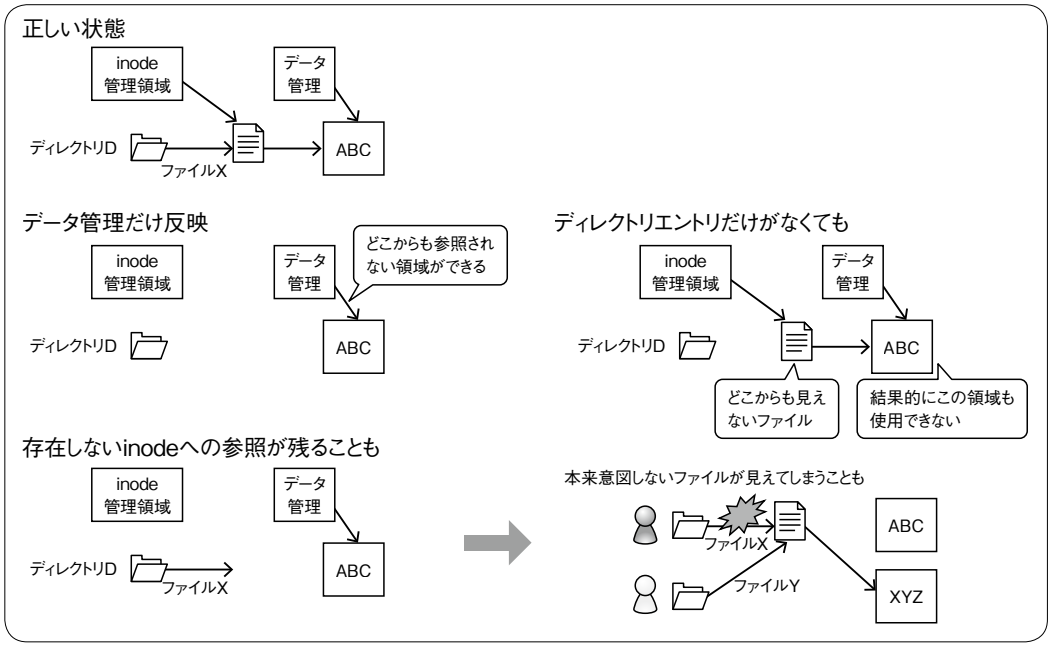
これらの操作を行う途中でコンピュータが停止した場合、さまざまな形でファイルシステムに不整合が発生します(**図3**)。たとえば、①の操作だけが反映されると、どのファイルからも使われていないデータ領域が生まれてしまいます。②～④までの場合には、どこからも参照されていないinodeも発生します。これらの領域やinodeは、ファイルシステム全体をスキャンして、「参照されていない」ということを確認しなければ解放できません。

さらには、OSやディスクの最適化により、これら操作の順番が入れ替えられることもあります。その場合、たとえばDからXへのエントリはありますが、対応するinodeは作られていないということも起こりえます。こちらは前のケースよりも状況が悪くなるかもしれません。

たとえば、ユーザAさんが新規ファイルへの書き込み中にコンピュータが停止して、「ディレクトリD内に、inode #42のファイルXがある」ことだけが記録されたとします。再起動後に、(fsckが行われず)別のユーザBさんが新しいファイルYを作ります。ファイルシステムはinode #42は空いていると思って、Bさんのファイルにinode #42を割り当てます。すると、本来は別のファイルであったはずのAさんのXと、BさんのYとが同じファイルを指してしまいます。すなわち場合によっては、ほかの人のファイルが覗(のぞ)けるわけで、これはセキュリティ上の問題となってしまいます。

fsckはファイルシステムをスキャンして、こういった誰からも参照されていない領域や、不完全なエントリを見つけ出すツールです。ファイルシステムを使い始めるとき(マウント時)に、OSはファイルシステムにdirty bitというものを立てて、ファイルシステムが使用中であることを示し、使い終わったとき(アンマウント時)にdirty bitを落とします。システムが正常終了しなかった場合、dirty bitが立ったままになります。dirty bitが立っていなければ、正常終了

**▼図3　ファイルシステムの不整合**



ということなので、ファイルシステムを調べる必要はありません。逆にdirty bitが立っていれば、ファイルシステムのどこかが上記のように不整合を起こしている可能性があるので、fsckはファイルシステムを調べにいきます。

このツールは一般に、ファイルシステム全体をスキャンするため、ファイルシステムが大きくなるほどに膨大な時間が必要になっていきます。さらには、fsckは一般的にシステムが起動する前に、ほかのプログラムが走らない状況で動作するので、システムの起動を遅くしてしまいます。

ここまで説明したケースでは、最悪fsckを行うことでファイルシステムを復旧できました。しかし、場合によってはデータが消えて元には戻せないこともあります。**図4**のようにAというファイル(inode #10)と、Bというファイル(inode #20)とがあるときに、"rename A B"とすることを考えます。

この場合に行われる操作は、

①Bがinode #10のファイルを指すように書き換える
②Aからinode #10を指すエントリの削除
③inode #20の解放
④inode #20のデータ領域の解放

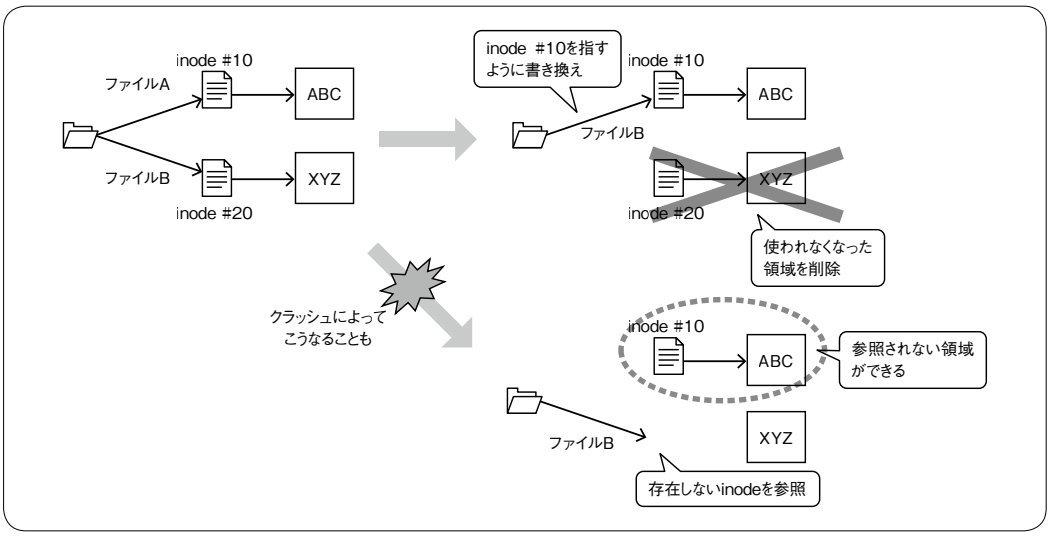
の4つです。前述のように、システムクラッシュ時には、これらの操作の一部だけが適用される場合があります。たとえば、②Aのエントリ削除と、③inode #20削除だけが行われるとどうなるでしょうか。Aの指していたinode #10のファイルは誰からも参照されず、Bは存在しないinode #20を指しています。さらに、Bのデータ領域は誰からも参照されずに残っています。元の状態に戻そうにも、inode #10を誰がどのような名前で参照していたのかはわかりませんし、Bのデータ領域がどのファイルのものであったのかもわかりません。できるのはせいぜい、inode #10に別の名前をつけて参照できるようにして[注3]、Bのエントリを削除、Bのデータ領域を解放して無駄な領域を片付ける程度です。

## さまざまな更新保護方法

こうして見たように、ファイルシステムの発行するI/O操作がOSやディスクによって最適化されるので、システム停止後に思わぬ結果が

注3) Ext4では、この復旧を行うための"lost+found"ディレクトリがあります。

▼図4 リネーム時の障害



起きることがあります。このような現象を防ぐために、ファイルシステムはさまざまな方法を用いて整合性を保護します。ここでは、

①同期書き込み
②ジャーナリング
③ログ構造ファイルシステム
④Copy-on-Write

の4つの方法について簡単に紹介します。

## ①同期書き込み

同期書き込みの発想は単純です。I/O操作が入れ替えられることで問題が発生するなら、順番が入れ替えられないようにすればよいわけです。書き込み操作後に、ディスクに今すぐI/Oを実行するように指示し、その完了を待つことでI/O操作の順番を守ることができます。順番が守られることで、ある程度のファイルシステム不整合の問題を防ぐことができます。

最初のファイル作成の例でも、リネームの例でも、順番が守られれば、存在しないinodeをディレクトリエントリが指したり、解放済のデータ領域をinodeが参照するといったことがなくなります。参照されていないのに確保されている領域やinodeが残る問題はありますが、これはいつかfsckすれば発見されて、ほかにトラブルを起こさないので**比較的**容易な問題です。

さて、同期書き込みはシンプルですが、その分、大きな問題点を抱えています。ひとつひとつのI/O操作でディスクの完了を待つので、ファイルシステムの速度はディスクの速度がボトルネックとなり、非常に遅くなってしまいます。とくにこの手法では、ディスクの各所にちらばった管理情報をいちいち少しだけ書き換え、完了を待つので大きくパフォーマンスが下がってしまいます。

## ②ジャーナリング

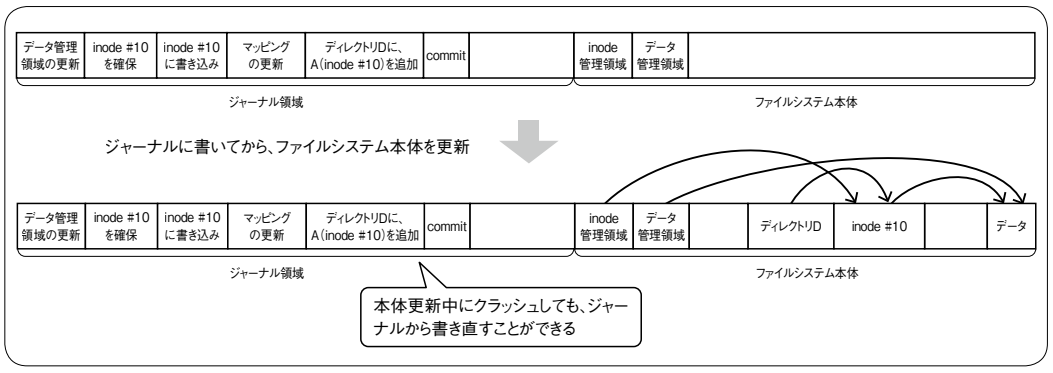
ジャーナリングは、1つのファイル更新に必要なI/O操作のログを、ファイルシステム内の「ジャーナル領域」という特別な領域に記録してから、I/O操作を実行する方法です(**図5**)。I/O操作の途中でシステムが停止しても、ジャーナル領域の操作ログから、元のI/O操作をやりなおすことでファイルシステムの整合性を守ることができます。Linuxでは、Ext3、Ext4、XFS、JFS、ReiserFSで使われています。

具体的にジャーナリングにおける、新規ファイルへの書き込みの様子を見てみましょう。まず、ジャーナル領域に、

・inode #10用のデータ領域を確保
・inode #10を確保
・inode #10へ管理情報を書く
・inode #10にマッピングを書く
・ディレクトリDに、ファイル名Aとしてinode #10を指すエントリを追加

と5つの操作ログを書きます。操作ログのI/O

**▼図5　ジャーナリングの概念**



が完了したうえで、操作ログの終わりを示すコミットログというものを書き、そのI/Oの完了を待ちます。ジャーナルへの記録中にシステムが停止した場合、コミットログが存在しないので、そのジャーナルは破棄されます。このとき、ファイルシステムの本体部分はまったく変更されていないので、整合性は保たれています。コミットログが記録されれば、ファイルシステム本体にログのとおりに変更を加えていきます。前述したように、この途中でシステムが停止しても、ログからすべてのI/O操作をやりなおせば整合性が保たれます。

同期書き込みでは、各操作ごとにディスク待ちが必要でした。一方、ジャーナリングではコミットログの前後と、操作ログの適用後の3回だけディスク待ちをすればよいので、その分性能が向上します。逆にジャーナリングでは、ログを書いてから実際の操作を行う分、ディスクに書く量が大きくなってしまう弱点もあります。

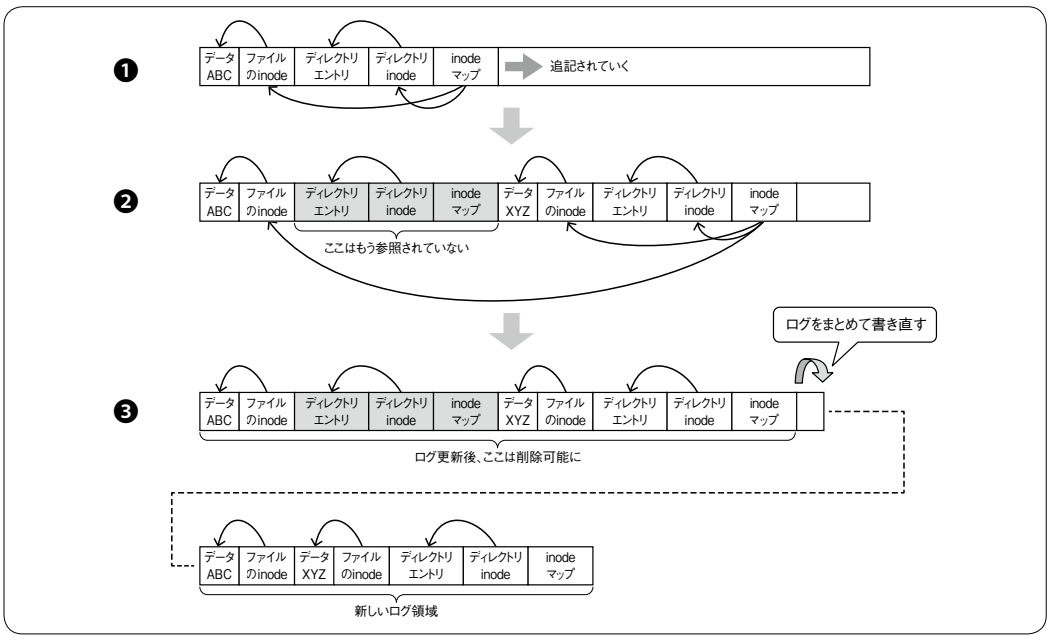
## ③ログ構造化ファイルシステム

ジャーナリングの考え方をさらに発展させて、ファイルシステム全体をログ領域にしてしまうのが**ログ構造化ファイルシステム**(以後、ログFS)です。Linuxでは、NILFS2やF2FSで使われています。ログFSは、すべてがいわばジャーナル領域で、ほとんどすべての書き込みがシーケンシャルに行われます。

具体的にログFSにおける、新規ファイルへの書き込みの様子を見てみましょう(**図6❶**)。まず、ファイルデータ"ABC"がログの中に書かれます。その後ろに、ファイルのinodeがデータを参照する形で書き込まれます。ディレクトリエントリが、新しいファイルのinodeを指すように書き込まれ、ディレクトリのinodeが書かれます。

このようにログFSでは、inodeがログ中の任意の位置に記録されます。そこでinodeの位置を記録する**inodeマップ**というデータもログに書きます。ファイルシステムの更新にしたがって、ログはどんどん追記されていき、inodeマップもinodeの更新ごとに書き込まれることになります。原理的にはログを読んでいけば、最新のinodeマップを知ることができますが、それ

**▼図6　ログ構造化ファイルシステム**



ではマウントに時間がかかってしまうので、定期的に最新のinodeマップがどこにあるのかを固定位置のチェックポイントブロックに記録します。マウント時は最新のチェックポイント以後のログを読んで、ファイルシステムの状態を復元します。

さてここで、同じディレクトリ中にもう1つファイルを追加したらどうなるでしょうか？(**図6❷**)

まずは、先ほどと同様に新しいファイルのデータとinodeが書かれます。次にディレクトリに新しいファイルが入ったので、それを反映したディレクトリエントリが記録されます。すると、ディレクトリエントリの位置が変わってしまうので、ディレクトリのinodeが書き直されます。すると、inodeの位置が変わったわけですので、inodeマップが書かれます。この更新によって、元のディレクトリのinodeとそのデータ部分への参照は消滅します。このようにログFSではファイルシステムの更新にしたがって、どんどん古いログが発生して、有効なログが断片化されます。これを放置して、ひたすらログを後ろに書いてくだけでは、いつかはディスクの容量を使いきってしまいます。

そのためログFSは、なんらかのタイミングでこれまでのログを整理して、新しいログのための連続領域を確保します。先ほどのログの状態で、このログの整理作業(クリーニング)が始まるとどうなるでしょうか。クリーニングは既存の、2つのファイルの作成と、2つのディレクトリの更新のログを読み込みます。このうち、ディレクトリ更新の片方は古いログなのです。そこで**図6❸**のように、2つのファイルの作成ログと、1つのディレクトリの更新ログを書きます。すると、クリーニングの前までのログはすべて古いログとして捨てることができるようになります。

ログFSでは全体をログにすることで、ほとんどの書き込みがシーケンシャルになり高いパフォーマンスを実現できます。その一方で、ログの整理が必要であるという弱点があります。また、ここで見たようなシンプルなログFSにおいては、ディレクトリの更新時のように、ちょっとしたデータの更新でinodeレベルまですべて更新され、書き込みが大きくなるという弱点もあります。

## ④Copy-on-Write

Copy-on-Writeは、既存のデータを上書きせず更新を別の場所に書いていくことで、既存のファイルシステム構造を壊さずに更新を行う方法です。LinuxではBtrfsで使われており、ZFSでも採用されています。

Copy-on-Writeを使うファイルシステムで、ファイルを書き換える様子を見てみましょう。ディレクトリDに2つのファイルXとYがあり、Xのデータを書き換えるとします。初めは**図7**のようにファイルX、Yのデータを XとYのinodeがそれぞれ参照し、ディレクトリエントリがXとYを参照、そしてディレクトリのinodeをroot blockが指しているという状態になっています。また、ブロック管理領域が使用中の領域を記録しており、これもroot blockから参照されています。

ここでXのデータを書き換えます。Copy-on-Writeですので、新しい場所にデータが書き込まれます。するとそこを参照するXのinodeが書き換えられ、さらにinodeを参照するディレクトリエントリが書き換えられ……と、root blockまで書き換えが連鎖していきます。書き換えにより使用中のブロックも変わるのでその情報も更新します。root blockを書き換えると、最新のroot blockを指すポインタを、新しいものに切り替えます。このように、上書きがないことで最新のroot blockからたどったファイルシステム構造が常に壊れていないことを保証できます。

ディスク上の動きとして、ログFSとCopy-on-Writeは似ています。実際、ログFSも既存のデータの上に上書きしないというCopy-on-

Write同様なポリシーを持っています。この区別は専門家でも議論が分かれるところですが、2つの違いとしてログFSでは最新のチェックポイントからログを復元していく部分が挙げられます。これはログFSが(基本的には)常にシーケンシャルに書いていくことによって実現されています。一方で多くのCopy-on-Writeのファイルシステムは使い終わった領域を可能であればすぐに再利用します。これはCopy-on-Writeでは、どの領域を使っているのかを管理していることから可能になっています。

## まとめ

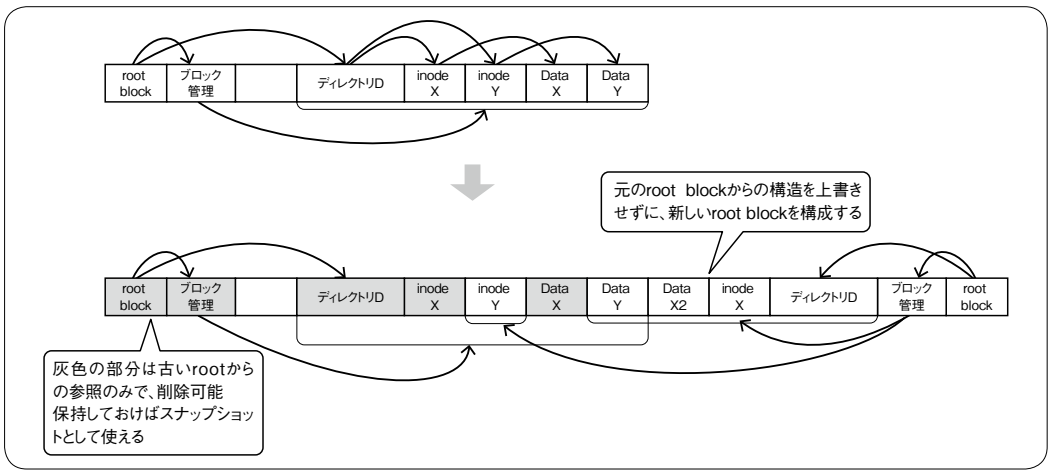
ローカルファイルシステムは、ディレクトリやファイルからなる抽象的なツリーを、ディスク上にデータとして表現する必要があります。具体的には次の5つについて、どのように管理するかが問題となり、ファイルシステム間の違いを生み出します。

- データ領域の使用状況の管理情報
- inode番号の使用状況の管理情報
- inodeの保存方法
- ファイル内アドレスと、物理アドレスとの変換情報(マッピング)
- ディレクトリ内のファイルの表現

また、OSやディスクの最適化により整合性が壊れないように、ファイルシステムはそれぞれの方法で整合性を保護しています。ここでは同期書き込み、ジャーナリング、ログ構造化、Copy-on-Writeの4つの手法を簡単に紹介しました。これらの手法のメリット・デメリットを簡単に**表1**にまとめました。

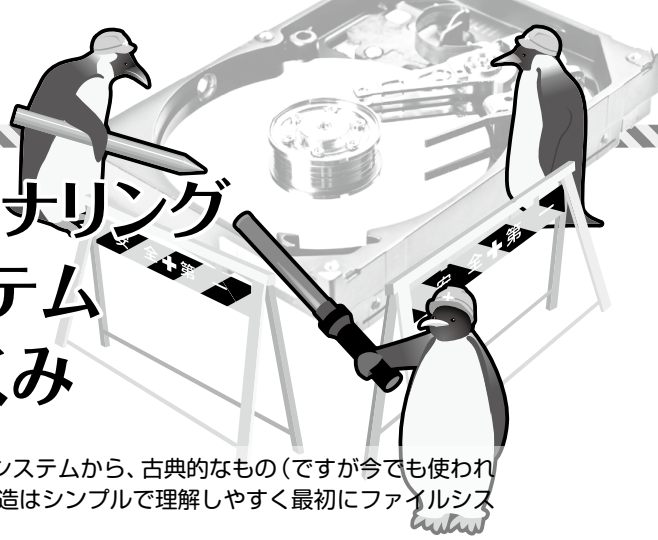
第2章以降では、具体的にLinuxのファイルシステムがどのようにこれらディスク上の表現と、整合性保護を実現しているのかを見ていきます。SD

**▼図7　Copy-on-Write**




**▼表1　整合性によるメリット・デメリット**

| 整合性保護 | Linuxでの実装 | メリット | デメリット |
|---|---|---|---|
| 同期書き込み | なし | 実装が容易 | ディスクに律速され遅い<br>リソースリークは起きる |
| ジャーナリング | Ext4、XFS | 実装が容易 | 書き込み量の増加 |
| ログ構造化 | F2FS、NILFS2 | 書き込みがシーケンシャルになる | ログのクリーニングの実装が困難 |
| Copy-on-Write | Btrfs | 上書きがないので、FSの構造が壊れない | 参照するブロックが再帰的に更新され、書き込みが増大する |



# 第2章 古典的ジャーナリングファイルシステム「Ext3」のしくみ

本章ではLinuxでジャーナリングを使うファイルシステムから、古典的なもの（ですが今でも使われているもの）として、Ext3を紹介します。Ext3の構造はシンプルで理解しやすく最初にファイルシステム構造を見るにはうってつけです。

Author 青田 直大（あおた なおひろ）

## Ext3の歴史

Ext3はExt2の流れを継ぎ、長年Linuxで多く使われていたファイルシステムです。名前からわかるとおり、Ext2の前に、Extended File System（ExtFS）というファイルシステムが、1992年4月にLinux 0.96cに実装されていました。これはMinixのファイルシステムから、最大ファイルサイズが2GBという制限と、最長ファイル名が255バイトであるという制限を取り払っただけのものでした。そのため、ExtFSには根本的に構造上の問題が残っていました。そこで1993年1月、ExtFSのコードをベースにさまざまな構造を見直して作られたものが、Ext2になります。

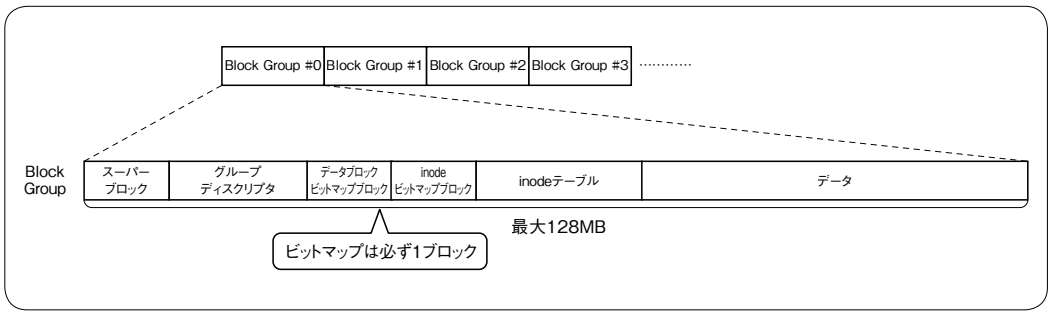
Ext3はExt2の構造をそのままに、おもにジャーナリング機能を追加したものです。そのため高い後方互換性を持ち、Ext2のファイルシステムをExt3のコードでマウントすることもできますし、Ext2のファイルシステムをマウントしたままExt3に変換することもできます。

## Ext3の全体的な構造

Ext3の全体構造を見ていきましょう。Ext3のディスクは、ファイルシステム作成時に、静的にBlock Groupという領域に分割されています（**図1**）。このBlock Groupごとに、空きブロック数や、空きinode数が管理されています。各Block Groupは次のように、さらに分割されています

- スーパーブロック（オプション）
- グループディスクリプタ領域（オプション）
- データブロックビットマップブロック
- inodeビットマップブロック
- inodeテーブル領域
- データ領域

**▼図1　特殊なファイルシステム**



### スーパーブロック

ファイルシステム全体の設定を記録しているブロックです。ブロックのサイズや、Block Groupのサイズ、ファイルシステム全体での空きブロック数や空きinode数を記録しています。これらのパラメータがないと、ファイルシステムの構造がわからないので、スーパーブロックは大切なブロックです。このブロックは最初のBlock Groupと、バックアップ用にその後ろのBlock Groupのいくつかに書き込まれます。マウント時には、基本的に最初のスーパーブロックが使われますが、ディスクの破損時には後ろのスーパーブロックを読ませることができます。

### グループディスクリプタ領域

各Block Groupについての情報を記録する領域です。それぞれの領域に、全Block Groupの情報がまとめて書かれています。ここにはデータブロックビットマップやinodeビットマップの位置や、空きブロック、空きinodeの数や、Block Groupのフラグが記録されています。ここにビットマップの位置が記録されていることで、後述するようにビットマップを動かすことが可能になっています。この領域も、スーパーブロックと同様に、最初のBlock Groupと、残りのBlock Groupのうちいくつかにだけ記録されます。

### データブロックビットマップブロック、inodeビットマップブロック

Block Groupの中で、どのブロックが使用中かを管理するのがデータブロックビットマップブロックです。同様にinodeビットマップブロックは、使用中のinodeを管理しています。これらのビットマップのサイズは、Block Groupごとに1ブロック(=ほとんどの場合4KB)と決まっています。したがって、1つのBlock Group中のブロック数は、8×4,096までとなり、すなわちBlock Groupのサイズ上限が、4KB×8×4,096=128MBとなります。

### inodeテーブル領域、データ領域

inodeを書くために確保されている領域です。Block Group 0のinodeテーブルが、inode #1からinode #4096、Block Group 1のinodeテーブルがinode #4097からinode #8192というように、inode番号順にinodeの位置が予約されています。

データ領域は、Block Groupの残りを占める、ファイルシステムのデータを記録する部分です。

## ブロックとinodeの管理

それでは、Ext3におけるファイルシステムのディスク上での表現方法について見ていきましょう。まずは、データ、inodeの使用状況と、inodeの保存方法について見てみます。

全体構成で見たように、Ext3ではブロックの使用状況と、inodeの使用状況、そしてinodeはBlock Groupごとに管理されています。ブロックの使用状況はビットマップで管理され、Block Groupの最初のブロックが使用されていれば、ビットマップの0ビット目が立っているというようになっています。Block Groupの先頭には、ビットマップやinodeテーブルがあるので始めのほうのビットは立っている、ということになります。同様にinodeビットマップもinodeテーブルの空き状況を、順番に表現しています。

inodeテーブルはinodeのデータを書くために、ファイルシステム作成時に、静的に確保される領域です。

これらの部分はExt2のころからまったく変わらず、Ext4でも制限の原因となっています。たとえば、inodeテーブルが静的に確保されるため、小さいファイルを大量に作るとデータ領域が余っていてもinodeが枯渇して、ファイルシステムがいっぱいでファイルを作れないというエラーが発生します。逆に巨大なファイルをいくつか作るというケースでは、inode用に確保されている領域をファイルデータ用に使うことができないという無駄が発生します。

実際にinodeテーブルには、どれだけのサイズが割り当てられているのでしょうか？ テーブルのサイズは、inodeのサイズと、何バイトに1つinodeを作るかのbytes-per-inodeの設定によって決まります。inodeのサイズはデフォルトで256バイトです。一方、bytes-per-inodeはファイルサイズの大きさによって変わります。

デフォルトでは、512MBから4TBまでのサイズのファイルシステムでは、16KBに1つのinode、4TBから16TBまでなら32KBに1つ、16TB以上なら64KBに1つのinodeが作られます。

すなわち、1TBのファイルシステムでは、128MBのBlock Groupにつき、128MB ÷ 16KB = 8,192個のinodeが作られます。すると、inodeテーブルのサイズは、8,192 × 256 = 2,048KBが予約されます。ファイルシステム全体では、1%、16GBがinode領域に割かれていることになります。

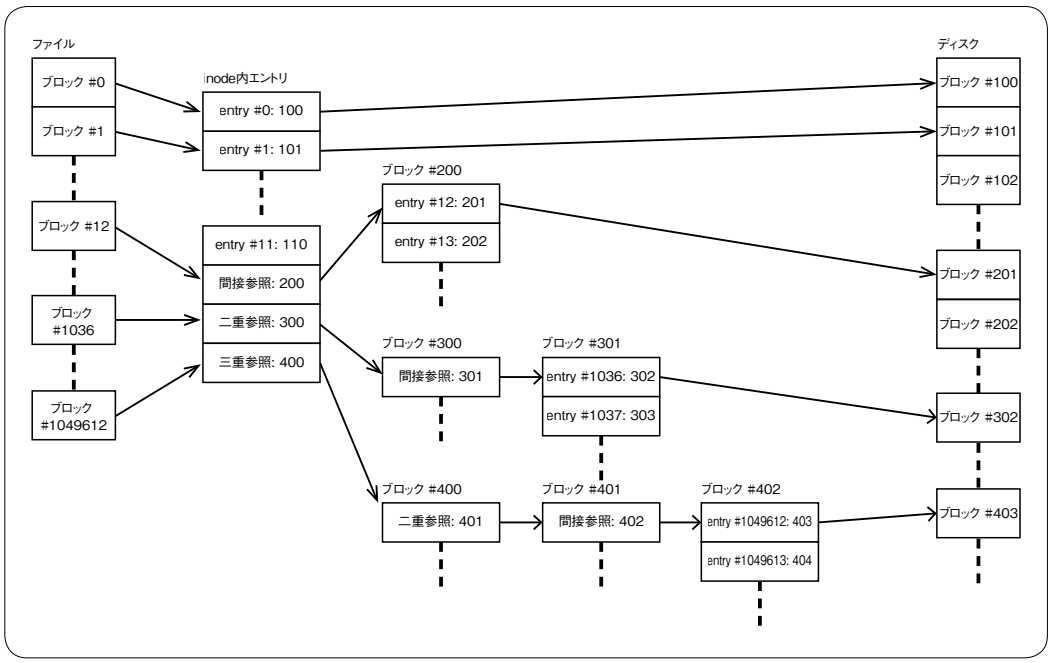
## Ext3のマッピング：block map

Ext3における、ファイル内のアドレスとディスク上のアドレスとの対応について見ていきましょう。

Ext3のinodeはi_blocksという配列を持ち、ここに1対1対応の形式でブロックアドレスが記録されます。**図2**のように、配列の最初のエントリがファイルの先頭のブロック（0から4,095バイトまで）に対応するディスクブロックを示し、その次のエントリが次のブロックを示す……というように並んでいます。この配列には15のエントリがあります。この配列に直接アドレスを書くだけでは、4KB × 15 = 60KBのファイルまでしか表現できません。

そこで末尾の3つのエントリがそれぞれ、間接参照ブロック、二重参照ブロック、三重参照ブロックを指すエントリとなっています。間接参照ブロックは、i_blocksの配列のようにブロックアドレスを記録する配列だけが入ったブロックです。1エントリが4バイトなので、1つ

**▼図2　block map**



のブロックには4KB ÷ 4 = 1,024個のエントリを持ちます。したがって、i_blocksの先頭の12エントリがファイルの4KB × 12 = 48KBの領域を示し、参照ブロック内のエントリが次の4KB × 1,024 = 4MBの領域を表現します。

同様に二重参照ブロックは、参照ブロックを指すエントリが並んだブロックであり、三重参照ブロックは、二重参照ブロックを指すエントリが並んだブロックになっています。したがって、ブロックマップのエントリ数は、(i_blocks内の)12 +(参照ブロックの)1,024 +(二重参照ブロックからの)$1{,}024^2$ +(三重参照ブロックからの)$1{,}024^3$ = 1,074,791,436個になります。これに4KBを掛けた約4TBが、block mapで表現可能な最大ファイルサイズです。

## Ext3のディレクトリ表現：リニアディレクトリ

Ext3のディレクトリは、通常のファイルのように割り当てられたブロックに、**ディレクトリエントリ**が並ぶ形式をとります。ディレクトリエントリには以下の要素が入っています。

- inode番号
- ディレクトリエントリの長さ
- ファイル名の長さ
- ファイルタイプ(通常ファイルか、ディレクトリかなどを示す)
- ファイル名

ここにinodeにも入っている情報であるファイルタイプが入っているのは、ファイルタイプを返すようになっているLinuxのシステムコールgetdents()をinodeを読むことなく効率的に実装するためです。

ファイルが追加されると、新しいディレクトリエントリを保持するのに十分な領域を探索し、そこにディレクトリエントリを保存します。

## Ext3のジャーナリング

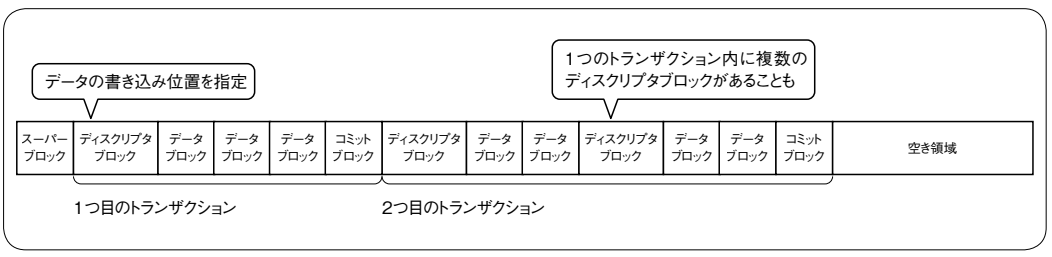
Ext3ではジャーナリングを用いて、ファイルシステムの整合性を保護します。Ext3のジャーナリング機能は、JBDというシステムに分離され、Ext3以外でも使うことができます。JBDは“Journaling Block Device”の意味で、ブロックデバイスへの書き込みにジャーナリング機能を提供する汎用的なシステムとなっています(**図3**)。

Ext3では、ジャーナル領域はinode #8のファイルのデータ領域として静的に確保されています。ジャーナル領域の先頭には、ジャーナル用のスーパーブロックがあり、残りの部分をリングバッファとしてブロックの更新情報を記録していきます。

ジャーナルへの記録は、後続するブロックがファイルシステムのどこに書き込まれるかを記録するディスクリプタブロック、実際のデータの入ったブロック、データブロックを書き終わったことを示しトランザクションを完了するコミットブロックの主要な3種類のブロック、およびリボークブロックで構成されます。

リボークブロックは、一度ジャーナルに書いたアドレスへの記録をキャンセルするためのブロックです。たとえば、ディレクトリを新しく

**▼図3　JBDの構造**



作ると、そのディレクトリエントリ用のブロックが確保され、そのブロックへの書き込みはジャーナルに記録されます。これがファイルシステム本体にまだ反映されていないタイミングで、作ったディレクトリを削除したとしましょう。このとき、ジャーナル内に記録されている、ディレクトリエントリブロックの更新はもう無意味なものになっています。そこでリボークブロックを使って、「このブロックへの書き込みはキャンセルする」ということを記録し、ジャーナルの反映時の無駄を削減します。

## 3つのジャーナルモード

Ext3ではデータとジャーナルに関して3つのモードが実装されています。デフォルトの“data=ordered”では、データが確実にディスクに書かれてから、そのデータを参照するメタデータ(マッピングやinodeなど)をジャーナルに書きます。

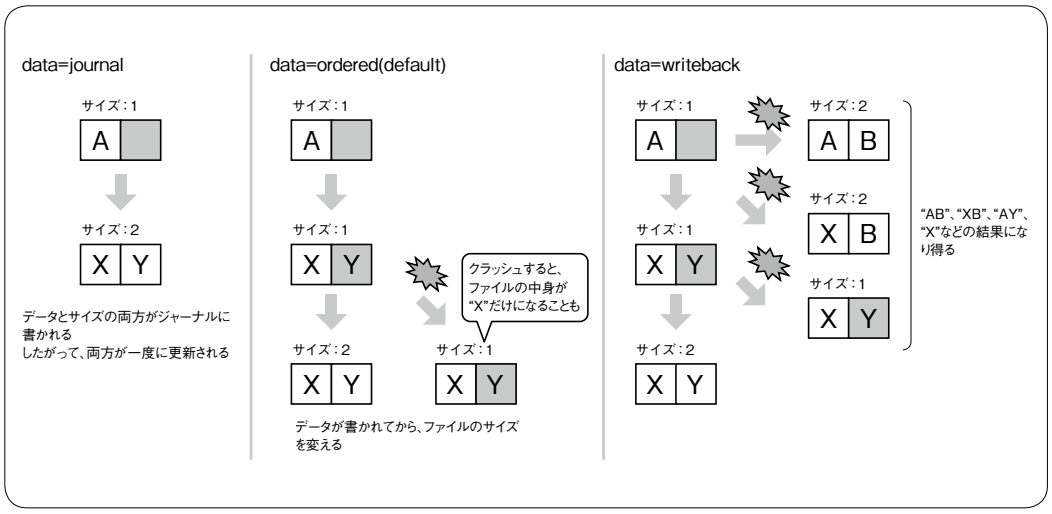
一方、“data=writeback”では、データの書き込み完了を待たずにメタデータの書き込みを開始します。“writeback”の方が書き込み完了を待たないためパフォーマンスは向上しますが、メタデータだけが書き込まれ、データが書かれていないという状況が起こりえます。その場合、「ファイルが存在するものの、書いた覚えのないデータが見える」といったことになります。

“data=journal”は、ほかの2つとは異なり、データもジャーナル領域に書くモードです(**図4**)。このモードがデータとメタデータの整合性を最もよく保護します。しかし、メタデータだけでなくデータも2回書くことになるのでパフォーマンスは下がってしまいます。

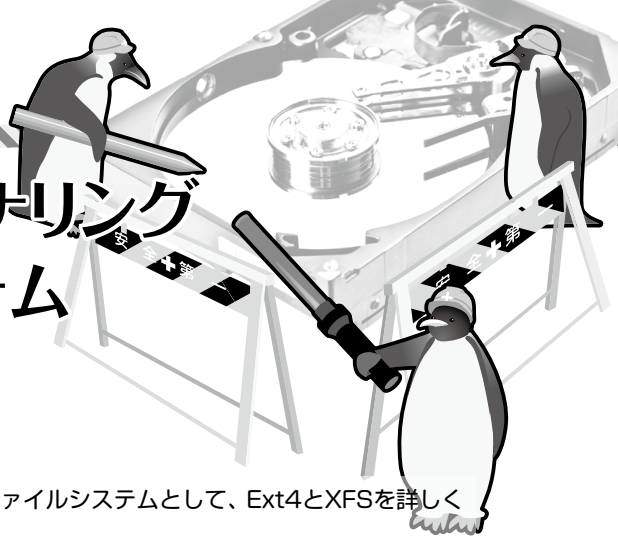
## まとめ

本章では、ジャーナリングを使うファイルシステムから古典的なものとして、Ext3の解説を行いました。Ext3の構造は、基本的な設計が昔からのファイルシステムとあまり変わらず、今ではさまざまな制限の原因となっています。次の章ではよりモダンなファイルシステムとして、Ext4とXFSを紹介します。Ext4がどうやってExt3との互換性を保ちつつ現代的なデータ構造を取り入れているか、そしてXFSがどのように作られているかを見ていきます。SD

**▼図4　データの書き込みモード。“A”を“XY”に書き換える**




# 現代のジャーナリングファイルシステム ～Ext4とXFS



本章では、Linuxでジャーナリングを使う現代的なファイルシステムとして、Ext4とXFSを詳しく紹介していきます。

Author 青田 直大（あおた なおひろ）

## Ext4の全体的な構造

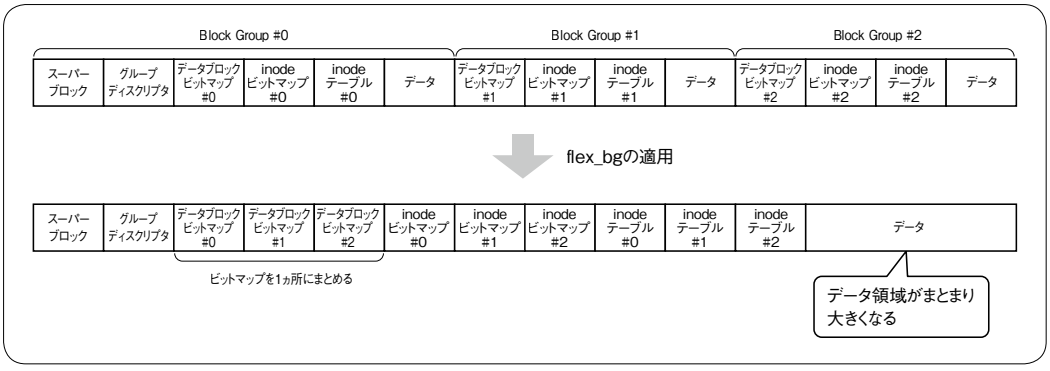
Ext3がExt2との互換性を保ったのと同様に、Ext4もExt3と後方互換性を持っています。Ext2とExt3の間では、全体的なディスク構成を変更しなかったのに対して、Ext4ではさまざまな点でディスク上の構造を変え、Ext2・Ext3にあった制限を回避しています。それでいて、プログラム上のデータ構造は多く共通しているため、Ext4のコードでExt3のファイルシステムをマウントできるようになっています。実際に、Linuxカーネルにおいて、以前はExt3とExt4のコードは分離されていましたが、今ではExt3のコードが削除され、Ext3のファイルシステムは完全にExt4のコードで取り扱われるようになっています。

Ext4のディスク構造は、Ext3のものをベースに現在のファイルシステムへの要求、すなわち巨大なファイル、巨大なファイルシステム、大量のファイルを持つディレクトリへの対応などを実現するように、うまく調整したものです。

前章で解説しましたが、Ext3のBlock Groupには128MBまでの制限があります。各データ領域の間には、ビットマップやinodeテーブルなどに予約された領域があり、データ領域は最大でも128MBで、バラバラに分割されていることになります。そうすると1GB程度のファイルであっても、ディスク上で9ヵ所以上に分割されることになります。こうした断片化はExt3において、巨大なファイルに対するパフォーマンス上の問題となります。

そこで、Ext4ではグループディスクリプタ領域に、ビットマップやinodeテーブルの位置が書かれていることを利用して、これらの位置を変更しています。Ext4では、複数のBlock Group

▼図1 Ext4における全体構造の拡張



のビットマップ、inodeテーブルをまとめて配置し、大きなデータ領域をとるflex_bgという拡張が使われるようになっています(図1)。

## Ext4のマッピング：extent tree

Ext3で使われていたblock mapは、1つのブロック(4KB)を表現するのに、4バイトを消費します。さらに、より大きなファイルであれば、参照ブロックなども消費することになります。たとえば、4GBのファイルには1,048,576個のエントリが割り当てられ、block mapだけで4MBの書き込みが必要になります。

このようにblock mapでは、大きなファイルを表現するにはスケールしません。そこでExt4での拡張では、extent treeという形式でファイルを表現するようになりました。

extentは図2のようにマッピングを、ファイル内のオフセット、長さ、ディスク上のオフセットの3つの組で表現する方法です。Ext4では1つのextentで32,768ブロック(= 128MB)まで表現できます。先ほどの4GBのファイルは32個のextentで表現されます。1つのextentのサイズは12バイトですので、384バイトで4GBのファイルをマップできるようになります。

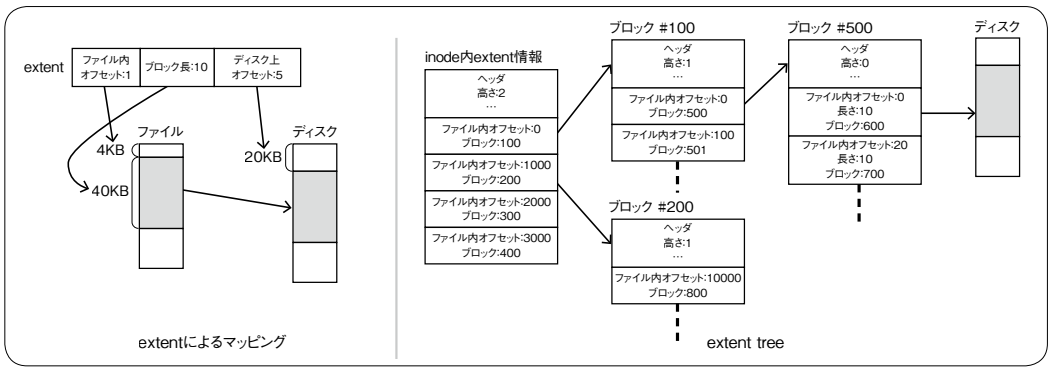
extentはblock mapの入っていた領域に保存されます。block mapの領域が60バイトで、extentのサイズが12バイトですので工夫しなければextentを5つしか保存できません。Ext4では、extent treeと言うツリー構造を使います。inodeの中には4つまでのエントリ(extent情報、またはほかのブロックへの参照情報)を持ちます。4つ以上のextentが必要な場合は、extent情報の代わりにブロックの参照情報を書いて対処します。参照されるブロックにもinode内と同様にエントリが並びます。

各エントリ保持領域は、ヘッダとエントリの並ぶ部分に分かれています。ヘッダにはマジックナンバー、この領域内で有効なエントリの数、この領域に入る最大エントリ数、ノードの高さが書かれています。高さはextent情報の並ぶブロックの高さが0で、そこを参照するブロック(またはinode)が高さ1、さらにそこを参照する部分の高さが2などとなります。参照情報にもファイルのどのオフセットを担当しているかが記録されており、ツリーをたどる際に使われます。

## Ext4のディレクトリ表現：Hash Tree

Ext3のディレクトリ表現であるリニアディレクトリでは、ある名前のファイルをディレクトリの下から探すときに、1つずつディレクトリエントリを読んでいくしかありません。ファイル数が少ないうちは、これでもよいのですが、ファイル数が増えるごとに線形に探索時間が長くなります。これもExt3におけるパフォーマンスの限界となっていました。

▼図2　extent tree

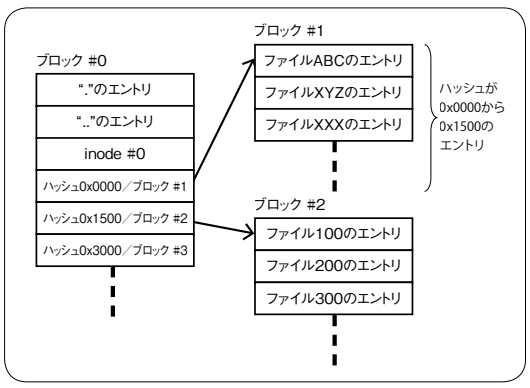


そこでExt4ではHash Treeという構造により、名前のハッシュを使って効率的にディレクトリエントリを見つける機能を追加しています。ディレクトリ内のファイルが増え、1つのブロックの内にすべてのディレクトリエントリを保持できなくなると、Hash Tree化が行われます。

Hash Tree化されたディレクトリでは最初のブロックが、Hash Treeのルートブロックとなります。このブロックには、ハッシュ値と(ディレクトリ内での)ブロックアドレスを書いたエントリが、ハッシュ値でソートされて並びます。Hash Treeをたどった末端には、Ext3の形式でディレクトリエントリが記録されたブロックが配置されます。したがって、**図3**のようにファイル名"XXX"のハッシュ値が0x2345(実際のハッシュ値は32ビット)で、ルートブロック中のエントリのハッシュ値がそれぞれ、0x0000、0x1500、0x3000であれば、0x1500≦0x2345＜0x3000なので、2番めのエントリをたどった先にファイル名"XXX"のディレクトリエントリが存在します。

Hash Tree内のブロックはうまく工夫されて、Ext2とも互換性があるようになっています。ルートブロックには、先頭に"."と".."に対応するディレクトリエントリが配置され、そのあとに4バイト分の0埋め領域が配置されています。Ext3のディレクトリ形式では、この4バイトにinode番号が入っています。Ext2のコードは、inode番号が0のエントリを終端と認識するので、こうやってハッシュツリー構造が入っている残りの部分をExt2のコードに解釈させないようにしています。同様にHash Tree内のほかのノードも先頭4バイトに0が入っており、Ext2のコードにはディレクトリエントリがまったく入っていないブロックのように見えます。一方で、末端のブロックはExt2時代のブロックと変わっていないので(探索に時間はかかりますが)、変わらず読み込むことができるようになっています。

**▼図3　Hash Tree**




## Ext4のジャーナリング

Ext4がExt3を拡張したように、ジャーナリングを担当するJBDも、JBD2として機能が拡張されています。代表的な追加機能としてチェックサムの追加と、それによるディスク同期の削減が挙げられます。

まず、JBDにおけるdata=orderedでのジャーナルへの書き込みを詳しく見てみましょう。このモードでは、まずデータを書き、ディスクへの反映を待って(ディスク同期をして)からメタデータをジャーナルに書き始めます。ジャーナルへの書き込みは、まずデスクリプタブロックとデータブロックの書き込みで始まります。

一通り書き込みリクエストを送ると、ディスク同期をしてから、コミットブロックを書き込みます。ここでディスク同期を行わなければ、コミットブロックがデータブロックよりも先に書き込まれる可能性があり、その場合不正なデータがファイルシステム本体に再現されてしまいます。したがって、JBDのdata=orderedモードでは、コミットブロック書き込み後のものを入れて、3回のディスク同期を行っています。

ディスク同期は、その時点までにディスクに送った書き込みリクエストをすぐにディスクに処理させ、完了を待つという方法で実現されています。これはディスクにおける、I/O処理のスケジューリングを阻害し、すべての書き込み

を待つという非常に「重い」処理となっています。

そこでJBD2ではチェックサム機能を導入して、このディスク同期の数を減らしています。チェックサム機能が有効になると、コミットブロックに、データブロックのチェックサムが記録されます。これによりディスクエラーでジャーナルが壊れても、ジャーナルを再生せず、ファイルシステム本体への影響を避けることができます。

さらにこのとき、**journal_async_commit機能**を有効にすると、データブロック書き込みとコミットの間のディスク同期を省略できます。もしもコミットブロックだけが先に書かれたとしても、そのときはチェックサムが不整合となるので、ジャーナルが破棄され、ファイルシステムが壊れる心配がなくなるためです。これによりジャーナル書き込み時のパフォーマンスが改善されます。

## XFSの全体構造

ここからはもう1つのジャーナリングファイルシステムであるXFSを紹介します。XFSはExt2とほぼ同時期に登場しています(Ext2が1993年、XFSが1994年)が、当初から64ビットのアドレスをサポートしたり、ツリー構造を使っているなど、高パフォーマンス・スケーラビリティを意識した作りになっています。

まずはXFSの全体構造から見ていきましょう。Ext4でディスクをBlock Groupに分割していたように、XFSもディスクをAllocation Groupという部分に分割し、Allocation Group単位で空き領域や、inodeの管理を行っています(**図4**)。Ext4のBlock Groupが最大128MBであったのに対して、Allocation Groupは最大1TBにもなります。

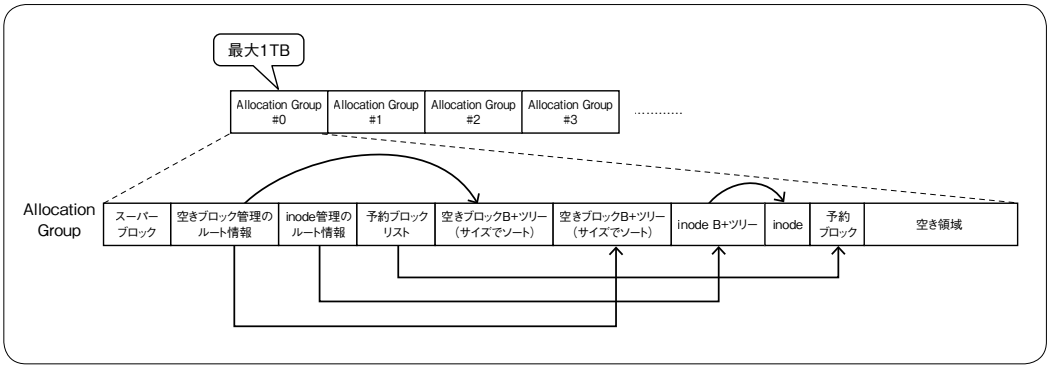
各Allocation Groupには、スーパーブロック、空きブロック管理情報のルート、使用中inode管理情報のルート、メタデータ用に予約済のブロックのリストが入っています。「スーパーブロック」とは言っていますが、これら4つ合わせて4KB以下のサイズです。このほかに空きブロック管理に6ブロック予約されていますが、それでも28KBとXFSでは静的に予約された領域は、Ext4に比べて小さくなっています。

## XFSのブロック管理

XFSは、B+ツリー構造で空きブロックを管理します。2つのツリーはどちらも連続した空き領域(Allocation Groupの先頭からの相対アドレスと、空き領域のサイズの組で表現)をアイテムとして持ちます。1つのツリーはアドレスをキーとしてソートされ、もう1つのツリーは空き領域のサイズをキーとしてソートされています。

たとえば、「アプリケーションが既存のファイルに追記して、データを近くに置きたい」という

**▼図4　XFSの構造**



ときはアドレスでソートされたツリーを使います。ほかのケース、たとえば、新規ファイルへの書き込みであれば、書き込みが入るサイズの領域を見つけなければいけないので、サイズでソートされたツリーを使います。

## XFSのinode管理

Ext4ではinode領域がファイルシステム作成時に予約されていましたが、XFSでは必要になったタイミングでinode領域が動的に確保されます。ファイル作成時に空きinodeが存在しない場合、新しく64個分のinode領域が確保されます。こうして確保されたinode領域(inodeチャンク)は、B+ツリーで管理されます。このB+ツリーには、inodeチャンクの最初のinode番号をキーとして、その領域中の空きinode数と、どのinodeが空いているのかを示すビットマップが入っています。

空いているinodeを見つけるには、inodeのB+ツリーをたどって空きinode数を見ていく必要があります。しかし、ファイル数が増えるごとに、空きinodeが0であるinodeチャンクも増えて、この探索にかかる時間もそれだけ長くなります。そこで近年のXFSでは、空きinodeがあるinodeチャンクのみを登録するB+ツリーも別個に保存しています。

## XFSのファイル表現

XFSもExt4と同様にextentを使ってファイルとディスクのマッピングを保持します。Ext4においては、1つのextentで128MBまで表現可能であったのに対して、XFSでは$2^{21}$ブロック(= 8GB)まで表現可能になっています。

Ext4の場合と同様に、extentは、その数が少ないうちはinode内に並べて配置されます。extentの数が増えると、ファイル内のオフセットをキーとしたB+ツリーにextentが保存されるようになります。

## XFSのディレクトリ表現

XFSでもExt4と同様に、ディレクトリエントリ(inode番号、ファイル名、ファイル名の長さ)が並べて記録されます。Ext4でファイル数が増えると、リニアディレクトリからHash Treeへと変換されたように、XFSでもファイル数に応じてディレクトリの形式が変化します。XFSでは、

①Short Form
②Block
③Leaf
④Node
⑤B+ツリー

と5つの形式を持ちます(**図5**)。Short Formディレクトリは、ファイル数が最も少ないときの形態です。このとき、ディレクトリエントリはinodeの中に記録されます。

ファイル数が増えていくと、inodeの中にはディレクトリエントリが入りきらなくなるのでBlock形式に変換されます。Block形式では、ディレクトリエントリを保持する専用のブロックが1つ割り当てられ、inode内にはそのブロックのアドレスが記録されます。このブロックには、ディレクトリエントリのほかに、エントリの空き領域情報(大きい順に3つ)、ファイル名のハッシュとエントリのアドレスの組が記録されています。これらの情報を使って、新規ファイルの作成時のエントリ位置の決定や、ファイルの探索を高速化しています。

さらにファイル数が増え、ディレクトリエントリとハッシュとでブロックが埋まると、ディレクトリはLeaf形式に変換されます。Leaf形式では、1つ以上の「データブロック」と1つの「リーフブロック」で構成されます。ディレクトリエントリと空き領域情報は各データブロックが持ち、リーフブロックにはハッシュ値の情報と各データブロックで一番空いている領域のサ

イズが保存されます。

よりファイル数が増え、リーフブロックがハッシュ値で埋まるとディレクトリはNode形式に変換されます。この形式ではハッシュ値を保持するリーフブロックが複数に分割され、2種類のブロックが追加されます。1つはノードブロックで、これはリーフブロック中のハッシュ値のインデックスを構成します。もう1つはfree indexブロックで、これはリーフブロックの「各データブロックで一番空いている領域のサイズ」を集めて保持するブロックです。

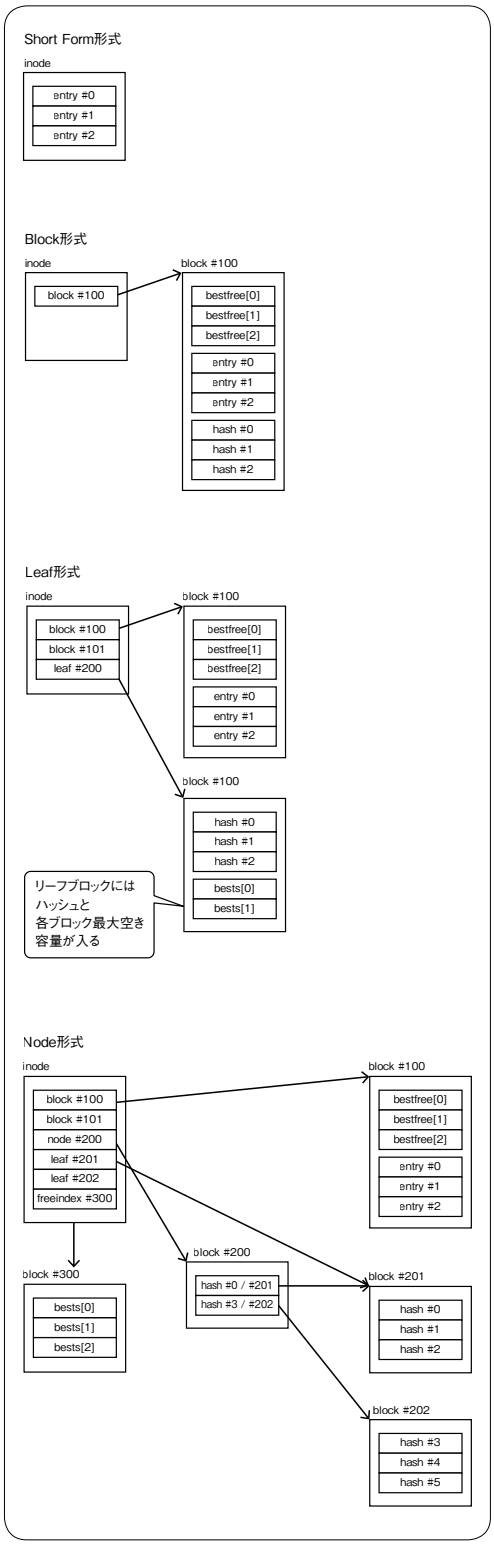
さらにファイル数が増えると、inode内にextentのリストを保持できなくなります。その場合、ファイルの場合と同じようにextentのリストがB+ツリーに拡張されます。構造的にはNode形式と変わりませんが、この状態のディレクトリをB+ツリー形式のディレクトリと呼んでいます。

## XFSのジャーナリング

XFSもExt4と同様にジャーナリングを用いて整合性を保護します。Ext4のジャーナリング機能は、汎用的に使うことができるようにブロック単位でジャーナルを行っていましたが、XFSでは「inodeをこのように編集する」「このブロックを確保する」といったように、論理的な操作のログをジャーナルに記録しています。そのため、XFSの方が同一サイズの領域に効率的に、多くの操作を記録できます。一方で実装の複雑さはXFSの方が高くなってしまいます。

簡単にXFSのジャーナルの構造を紹介します。XFSのジャーナルは、LogRecordという単位で記録されます。各LogRecordは、Log Sequence Number（LSN）という64ビットのIDで識別されます。LSNの上位32ビットは“cycle number”と呼ばれ、ジャーナル領域に一周書き込むたびにインクリメントされます。下位32ビットはLogRecordのジャーナルの先頭からのオフセット÷512で決定されます。

**▼図5　XFSのディレクトリ表現**



LogRecordは**図6**のように、先頭512バイトを占めるヘッダと、「ログアイテムバッファ」で構成されます。LogRecordのヘッダにはログアイテム(操作ログ)の数や、LSNが記録されています。ログアイテムバッファはその名のとおり、ログアイテムが並ぶバッファです。ログアイテムは所属するトランザクションのIDや、ログアイテムの種類を記録するヘッダと、種類ごとのデータを記録する領域からなります。

1つのトランザクションは、トランザクションの開始フラグの付いたログアイテムで始まります。その直後にはトランザクションの説明を行うログアイテムがつき、ここにトランザクション内のログアイテム数が記録されます。そのあとに、各種ログアイテムが続き、最後にコミットを示すログアイテムが記録されます。コミットのログアイテムがなければ、そのトランザクションは不完全とみなされ、その部分のログは再生されません。

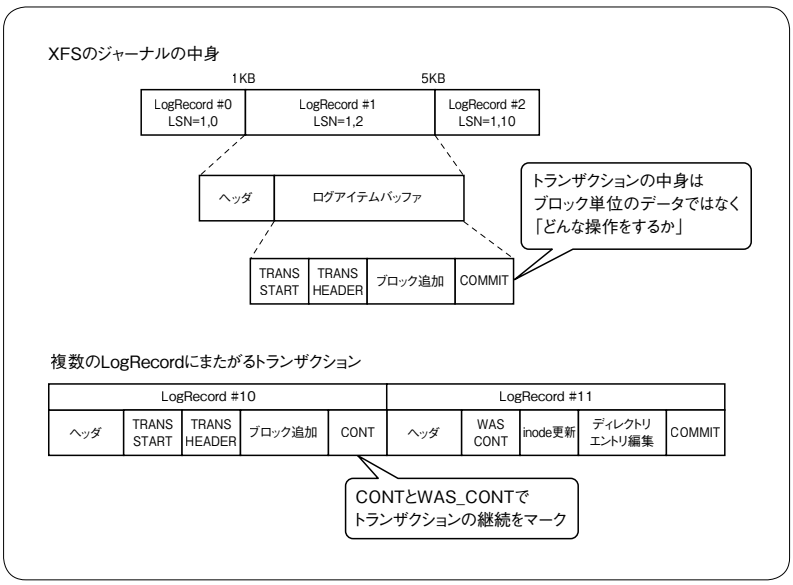
1つのトランザクションが、複数のLogRecordにまたがる場合もあります。その場合、またがる前のLogRecordの最後のログアイテムにCONTINUEフラグが、またがったあとのLogRecordの最初のログアイテムにはWAS_CONTINUEフラグがそれぞれついて、トランザクションの継続を示しています。

## まとめ

本章ではLinuxのジャーナリングファイルシステムExt4とXFSとを紹介しました。Ext4はExt2時代からの互換性をなるべく保ちながら、新しいデータ構造を導入しています。古いデータ構造と新しいデータ構造が同居しているので、Ext4を見ることで古い構造のシンプルさと問題点、新しい構造の効率のよさがわかりますね。

XFSは設計当初からExt4では拡張にあたるツリー構造や、extent、64ビットのアドレッシングをサポートしていました。したがってExt4よりも自然に、かつ高パフォーマンス・スケーラビリティが向上するように作られています。最近ではCopy-on-Writeの機能が追加され、ファイル単位でスナップショットをとることができるようになるなど、今でも活発に開発されています。SD

**▼図6　XFSのジャーナリング**



# 第4章 フラッシュデバイス用ファイルシステムと、Copy-on-Writeが特徴のBtrfs

本章では、Linuxのログ構造化ファイルシステムからF2FSと、Copy-on-Writeを用いるファイルシステムであるBtrfsとを詳しく紹介します。

Author 青田 直大(あおた なおひろ)

## Flashデバイスの特性

F2FSは、Flash Friendly File Systemの略で、その名のとおりFlashデバイスに最適化したさまざまな機能を持っています。F2FSはさまざまな点でFlashの特性を意識した設計がなされています。そこで、F2FSの解説に入る前に、まずSSDのようなFlashデバイスの特性について解説します。

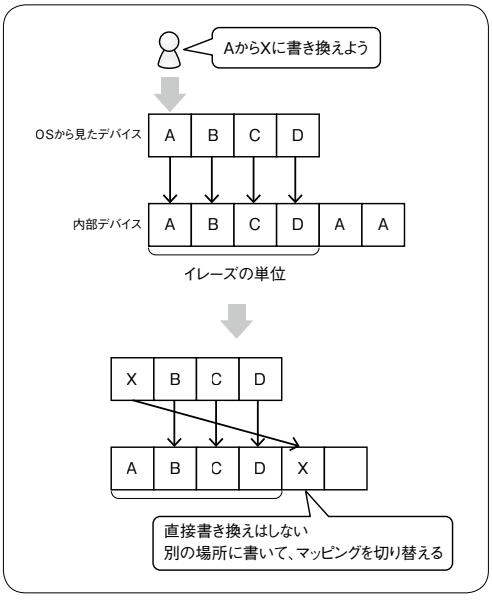
Flashデバイスはハードディスクと比べて、高いパラレル書き込み性能を持ちます。ハードディスクでは、離れたアドレスに連続して書き込みを行うと、ディスクを回転するシーク時間が必要になります。一方で、Flashデバイスではそのようなシーク時間は必要ありません。こうしたことから、Flashデバイスは、複数の離れた場所へのI/Oを効率的に処理できます。

また、Flashデバイスはハードディスクとは違った書き換え特性を持ちます。ハードディスクでは、同じブロックを好きなように書き換えることができますが、Flashデバイスでは一度書いたブロックは、リセット処理(イレーズ)を行うまでは書き換えることができません。

イレーズは遅く、かつリセットの単位が大きいため厄介な問題となります。単純に実装すると、たとえばイレーズの単位が16KBのときに、1ブロック(=4KB)を書き換えるために、周辺16KBを読み込んで、1ブロックをメモリ上で書き換え、イレーズ処理の完了を待ち、16KBを書き直す、という処理が必要となります。

実際には、Flashデバイスは Flash Translation Layer(FTL)というものを導入して、この問題を回避します。FTLはOSに見せるアドレスと、実際のデバイス上のアドレスを動的に変更する機能を持ちます。今、**図1**の上のようにブロックが対応しているとしましょう。ここに書き込みがあると、デバイスは元のブロックではなく、未書き込みのブロックに書き込みを反映します。そのうえでマッピングを今書き込まれたブロックに切り替えます。こうすることで、OSから

**▼図1 Flashデバイスの書き換え**



はすぐにデータが書き換わったように見えます。

FTLを使う上での問題は書き換えられてマッピングが外れた領域の回収(= Garbage Collection(GC))です。イレーズ領域内のすべてが書き換えられれば、イレーズして終わりですが、ユースケースによっては、領域のうちの一部がずっと書き換えられずに残る場合もあります。その場合は、その部分をほかに移してイレーズする、といった作業が必要となります。

## F2FSの特徴的機能

F2FSは、サムスン社が開発していることから、「Android用では?」とも思えるような変わった最適化をほどこしています。特徴的なのはマルチヘッドロギングという機能です。一般的なログFSでは、ハードディスクがうまく動くように、ログの書き込みは連続した1つの領域にシーケンシャルに行います。一方でF2FSでは、最大で6個所に同時にログの書き込みを行います。これは前述した、Flashの高いパラレル書き込み性能を活かした機能です。

6個所のログ領域には、それぞれ特定の種類の書き込みが割り当てられます。割り当ては、データとメタデータのそれぞれについて、書き換えが多いと予測される種類の順番に3つのカテゴリHot/Warm/Coldに分類して行われています。こうすることで、よく書き換えられるデータ同士が1ヵ所にまとまり、FTLによるGCが効率的に動くと期待されます。

ここでデータの書き換え頻度の分類にサムスンらしいと思える特徴があります。F2FSはColdなデータ(すなわち、あまり書き換えられないデータ)として、マルチメディアファイルを認識します。あるファイルがマルチメディアファイルかどうかは、ファイル名の拡張子で判別されます。この拡張子のリストはデフォルトでは、jpgやpngの画像ファイル、avi、mp3や3gpなど音声・動画ファイルのほかに、apkとAndroidのパッケージファイルまでもが入っています。このあたりはやはりAndroid端末を出しているサムスンらしいですね。

## F2FSの全体構造

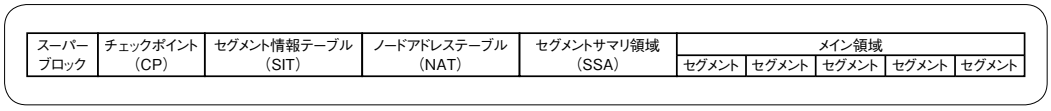
F2FSのディスクは**図2**のようにスーパーブロックを含めて6つの領域に分けられています。F2FSのメイン領域は「セグメント」という単位で管理されています。デフォルトでセグメントのサイズは2MBとなっています。

スーパーブロックには、いつものようにファイルシステム作成時に決定されて変わることのないパラメータが記録されています。チェックポイント領域は、第1章で示したように有効なログエントリを参照するチェックポイントデータが2つ入る領域です。1つは前回完了したログを参照し、1つは現在編集中のログを参照します。すなわち、どちらかのチェックポイントを使うことで、整合性のとれたファイルシステムをたどることができるということになります。

セグメント情報テーブル(SIT)は、各セグメント内で有効なブロックの数と、どこのブロックが有効かを示すビットマップが入っています。ノードアドレステーブル(NAT)は、メイン領域のメタデータブロックのアドレスを変換するためのテーブルです。詳しくは後述します。セグメントサマリ領域は、メイン領域の各ブロックについて、そのブロックを参照している「オーナー」の情報を持つテーブルです。この情報があることで、まだ参照されているブロックを、効率的にほかの場所へ動かすことが可能となります。

チェックポイントと同様に、SITとNATも2

**▼図2　F2FSの構造**



つに分かれており、それぞれ2つのチェックポイントデータに対応しています。

## Wandering Tree Problemへの対処

F2FSではExt3と同じように、ブロックマップでファイルとディスクのマッピングを表現します。古典的なログFSにおいて、このようなファイルで三重参照に入るファイル領域のデータを書き換えると、どうなるか考えてみましょう。

第1章で触れたような古典的なログFSでは、**図3左**のように、データブロックを参照エントリが並んだブロックが参照し、そこを二重参照エントリのブロックが参照し、さらにそれをinodeが参照する、という形になっています。ここでデータが書き換わると、データブロックの位置が変わるので参照ブロックが書き換えられ、参照ブロックの位置が書き換わったので二重参照ブロックが書き換えられ、結局inodeまですべて書き換えられる、というように1つのブロックの書き換えで4つのブロックの書き換えが発生します。このように書き換えの連鎖が発生することを、"Wandering Tree Problem"といいます。

F2FSでは、ノードアドレステーブル(NAT)を導入することでこの問題を回避しています。第1章で解説した古典的なログFSでは、inodeマップによって、inodeの場所が変わっても、inode番号からそのinodeの位置を取得できました。これにより、inodeが動いてもそこを参照するディレクトリエントリは書き換えずに済んでいました。これをinodeだけでなく参照ブロックなどにも拡張するのがNATのアイデアです。

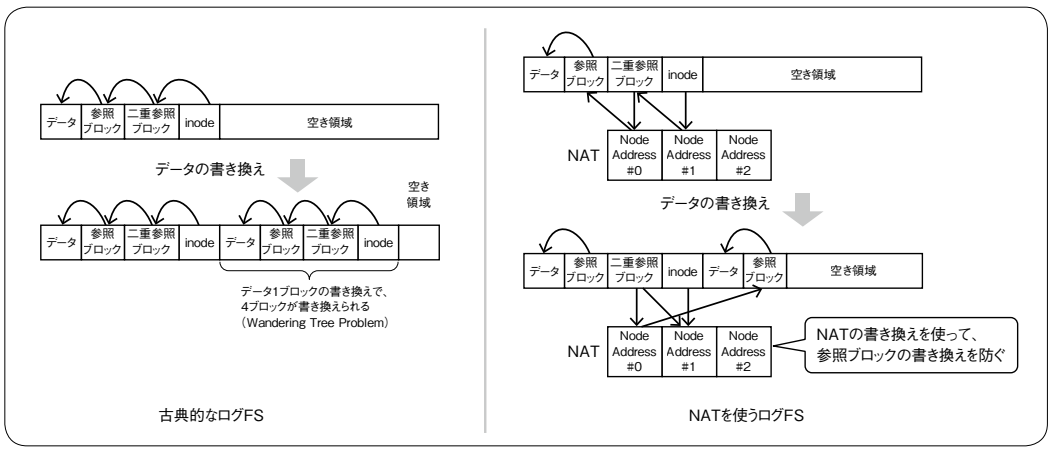
先ほどと同じ状況でNATを使うF2FSでのデータがどうなるかを見ていきましょう(**図3右**)。NATを使っているので、inodeから二重参照ブロックへのアドレスと、二重参照ブロックから参照ブロックへのアドレスには物理ディスクではなく、NAT上のアドレスが書かれています。

ここでデータが書き換わると、まず新しいデータブロックが書き込まれます。すると、そこを参照する参照ブロックも新たなアドレスへと書き換えられます。古典的なログFSでは、ここから二重参照ブロックとinodeも書き換えられていましたが、NATを使うF2FSでは参照ブロックに対応するNATのエントリを書き換えることで、ブロックの書き換えを止めることができます。

## ディスク上のファイル表現

ここまで見たように、F2FSではブロックの使用状況はSITの中でセグメントごとにビットマップで管理されています。また、inodeの使用状況はノードアドレスの使用状況として管理さ

**▼図3　Wandering Tree Problemへの対処**



れ、inodeはノードとして動的に領域が確保され保存されます。また、ファイルはExt3と同様にブロックマップとして表現されています。

F2FSでは、変わったディレクトリ表現を使っています。ディレクトリエントリ(ハッシュ、inode番号、名前長、ファイルタイプ)を持つブロックには、**図4左**のようにエントリの有効を示すビットマップとファイル名を保持するエントリが入っています。ハッシュは入っていますが、ほかのファイルシステムのようにハッシュ値でソートされて**いません**。F2FSは、1つのブロックの中では、エントリを頭から最後まで順番に探索します。

もちろん、これではファイル数が増えたときにスケールしません。そこでF2FSは階層型ハッシュテーブルを使います。ディレクトリエントリブロックは、**図4右**のように階層化されたバケツのいずれかに所属します。level #0にはバケツが1つ、level #1にはバケツが2つ、level #2には4つというようにlevelが上がるほどに、バケツが増えていきます。各バケツには、そこに属するブロックに入り得るディレクトリエントリの条件が設定されています。

たとえば、level #0のバケツにはどんなハッシュのディレクトリエントリでも入り、level #1の0番目のバケツにはハッシュが2の倍数であるディレクトリエントリが入ります。

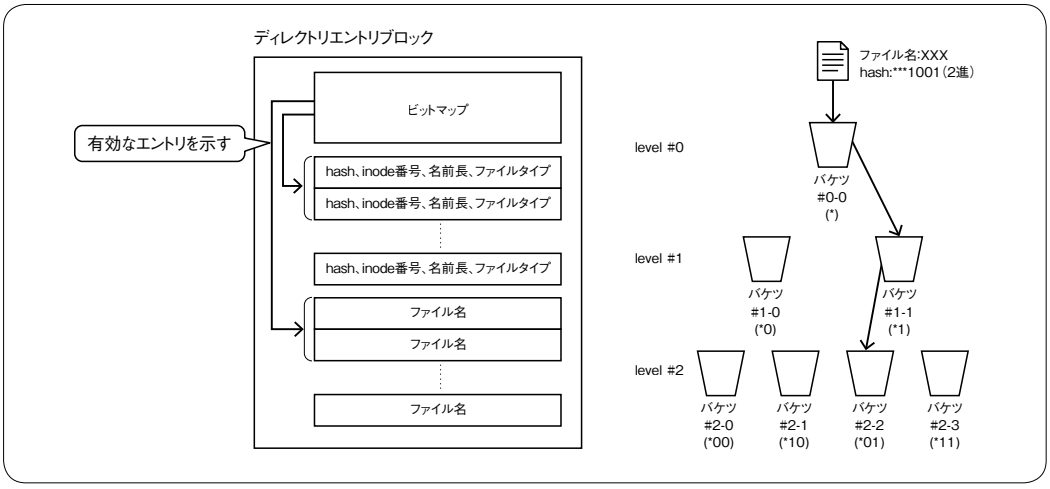
このように、level #nのk番目のバケツには、ハッシュをそのlevelの中のバケツ数で剰余をとったときにkとなるようなディレクトリエントリが入ります。この性質を使って、F2FSはlevel #0のバケツから探索を始め、見つからなければ次のレベルのバケツを見る……というように探索を行います。

## Btrfsの特徴

ここからはBtrfsについて解説します。Btrfsは複数デバイスを透過的に1つのファイルシステムにまとめ、RAID機能をファイルシステムレベルで持つなどZFSと似た機能を備えた先進的なファイルシステムです。ほかにもデータの透過的圧縮や、効率的なバックアップなどさまざまな機能を備えています。

Btrfsは、Btree File Systemの意味で、その名のとおりすべてのデータ構造がさまざまなBツリーの中で管理されています。Bツリーは、すべて"(objectid, type, offset)"の3つ組をキーとしています。typeがキーの種類を指定し、objectidとoffsetはtypeによって意味が変わります。たとえば、(42, INODE_ITEM, 0)というキーのアイテムは、objectid=42がinode番号で、

**▼図4　F2FSのディレクトリ**



offsetは常に0となり、inode情報を保持するために使われます。

Btrfsのツリーには、次のものがあります。

・ファイルシステムツリー
　inodeやextent情報を保持する
・extentツリー
　extentの被参照情報を保持する
・checksumツリー
　checksumを保持
・chunkツリー
　Block Groupのマッピングやファイルシステムを構成するデバイスを管理
・デバイスツリー
　デバイス上で確保している領域を管理
・rootツリー
　そのほかのツリーのrootのアドレスを保持する

## Btrfsのアドレス：extent空間

多くのファイルシステムでは、ほかのブロックやデータを参照するのにディスク上のアドレスを使っています。Btrfsでは、複数のディスク上にファイルシステムが構築できるため、このようなアドレス指定は使うことができません。

そこでBtrfsでは、extent空間という論理的なアドレス空間を導入しています(**図5**)。Btrfs上のほぼすべてのアドレスが、このextent空間のアドレスを使用します。ちょうどF2FSのノードアドレスが全ブロックに適用されるようなものです。

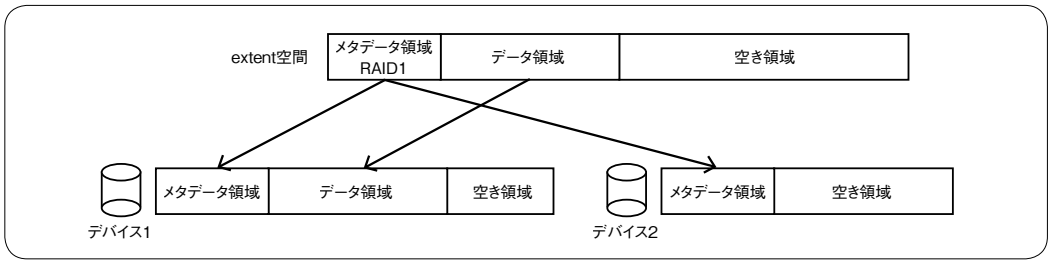
extent空間はBlock Groupという単位で分割され、ディスクへとマッピングされています。このBlock Groupで、RAIDが行われます。Block Groupのサイズは、ファイルシステムのサイズによっても変わりますが、データ用には1GB、メタデータ用には256MB(50GBを超えるファイルシステムでは1GB)で確保されます。

## Btrfsのファイル表現

Btrfs上でinodeは(inode番号, INODE_ITEM, 0)というキーで「ファイルシステムツリー」というBツリーに保持されます。Btrfsはデフォルトでは、どこのinode番号が空いているかは管理していません。単純に、これまで使った最大のinode番号を覚えておき、その1つ先のinodeを新しいファイルに割り当てるといった戦略をとります。inodeは64ビットで表現されるので、相当数のファイルを作らなければinodeが枯渇するということはありません。しかし、もしも枯渇した場合には、inode_cacheという機能を使い、inodeのどこが空いているかのビットマップを作り、そこからinodeを割り当てることができます。

XFSやExt4と同様にBtrfsのファイルもextentにより、extent空間へとマップされます(**図6**)。Btrfsにおいては、データの圧縮が行われるため、「ファイル上での長さ」と「extent空間上での長さ」とが記録されます。extent空間上の領域は、さらに前述したBlock Groupの情報を使って実ディスクへとマップされます。extentの情報は、ファイルシステムツリーの中に、

**▼図5　Btrfsのextent空間**



(inode番号，EXTENT_DATA，ファイルオフセット)をキーとして保持されます。

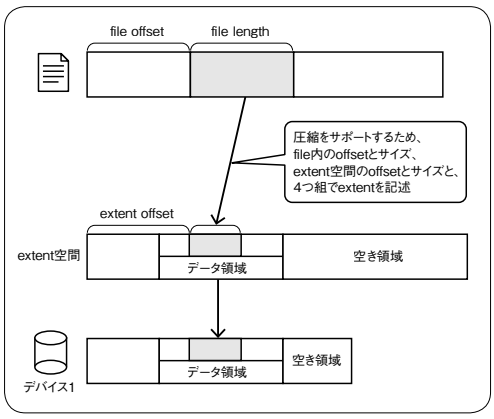
ディレクトリ中のファイルもまた、ファイルシステムツリー内に記録されます。ディレクトリとファイルの関係の表現には、3つのキーが必要となります。1つ目は(ディレクトリのinode番号，DIR_ITEM，ファイル名のハッシュ)というキーで、ほかのファイルシステムと同様にファイル名を使って効率的にファイルを見つけだすために使われています。2つ目のキーは(ディレクトリのinode番号，DIR_INDEX，index番号)というキーで、readdir()でディレクトリエントリを返す順序を決定するために使われます。3つ目のキーは、(inode番号，INODE_REF，ディレクトリのinode番号)というキーです。これは、あるファイルがどのディレクトリからどのような名前で参照されているのかを記録しています。

## スナップショットのしくみ

Btrfsはスナップショット機能で有名なファイルシステムです。スナップショットというのは、ある時点でのファイルシステムツリーを高速かつ軽量に保存する機能です。スナップショットは、実は簡単に実現できます。

前述したとおり、Btrfsのファイルシステムツリーは**図7**の左上のように、Bツリーで表現されています。スナップショットは、単純にrootをコピーすることで実現されます。これで、このコピーからも、その時点でのファイルシステムツリーをたどることができるようになりました(**図7右上**)。ここでファイルの書き換えがあるとどうなるでしょうか。ファイルの更新はBツリーの書き換えに対応します。Copy-on-Writeの原則どおり、ツリーのノードをrootまでコピーして書き換えを行います。すると、「FSツリーのroot」からは書き換えられたファイルシステムツリーが見えます。その一方で、「snapshotのroot」からたどれば、元のツリー構造が保存されています。このようにCopy-on-Writeの性質を使えば、スナップショットは自然に実装できます。

**▼図6　Btrfsのマッピング**




## Btrfsのブートストラップ

前述したように、Btrfsはほぼすべてのアドレスがextent空間のアドレスで記録されています。これはツリーのルートのアドレスすら例外ではありません。したがって、ツリーを読み込むために、どこか固定位置にextent空間から物理アドレスへのマッピングが書かれている必要があります。

この情報はスーパーブロックの中に保持されます。Btrfsを構成するすべてのパーティションには、その中の決まった位置(先頭から64KB、64MB、256GB、1TB、1PB)にスーパーブロックが配置されています。この中にsys_chunk_arrayという2KBの領域があり、ここにchunkツリーのコピー、すなわちBlock Groupから物理デバイスへのマッピングや、ファイルシステムを構成するデバイスの情報がコピーされています。スーパーブロックには、ほかにもファイルシステムのUUIDや、ファイルシステム ツリーのルートのextentアドレスが記述されています。Btrfsのマウント時には、まずここを読んでファイルシステムを構成するデバイスを認識し、ツリーが実際にディスクのどこにあるかを認識するということになります。

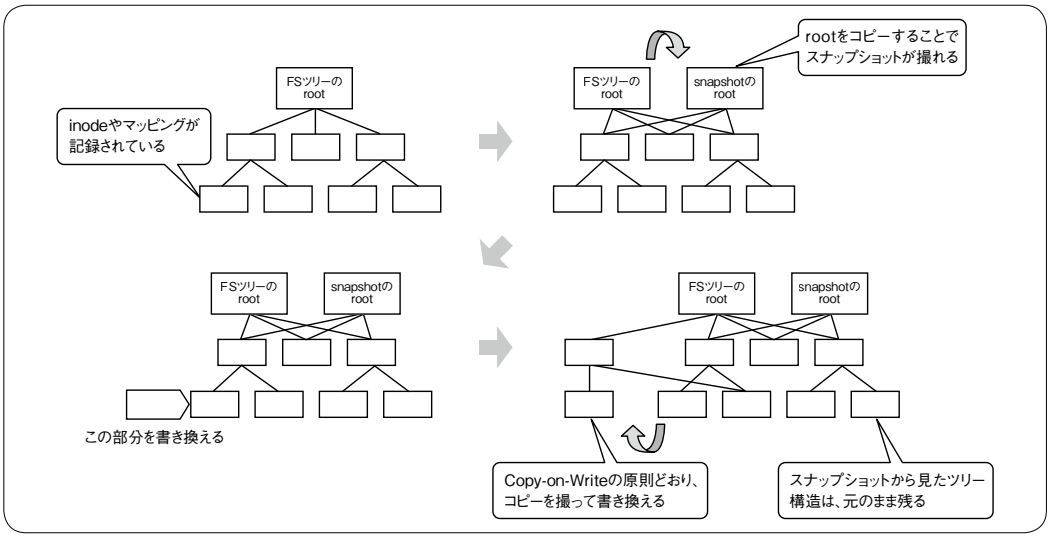
## さまざまなファイルシステム

本章ではF2FSとBtrfsを紹介しました。F2FSはデータ構造としては、ビットマップやブロックマップなどExt3と同様のものも使っています。その一方で、Flashデバイス・Androidに特化した最適化をほどこし、その特徴を彩っています。Btrfsは、すべてをBツリーの上に乗せてCopy-on-Writeを採用して、スナップショットなどさまざまな機能を実装しています。

本特集では、Ext3、Ext4、XFS、F2FS、Btrfsと、昔からあるものから、近年のものまで、ファイルシステムがどのようにファイルを表現しているのかを見てきました(**表1**)。ファイルシステムは、Ext3からExt4・XFSの変化のように、データ構造が効率化されるだけでは終わりませんでした。たとえば、F2FSのようにターゲットをFlashに絞って特化したり、あるいはBtrfsのようにRAIDやスナップショットをサポートするなど機能を拡張し続けています。研究段階でも、書き込み速度に特化したBetrFS[注1]や、ディスクの操作やクラッシュからの復帰が仕様どおりに動くと証明し、安全性を向上させたファイルシステムであるFSCQ[注2]など機能特化や機能拡充に向けて今でも研究・開発が進んでいます。本特集をきっかけにファイルシステムの内側に興味を持っていただければ幸いです。SD

注1) http://betrfs.org/
注2) https://github.com/mit-pdos/fscq

**▼図7　スナップショットのしくみ**




**▼表1　ファイルシステムの構造まとめ**

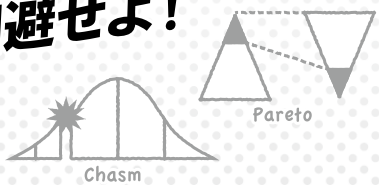
| ファイルシステム | Ext3 | Ext4 | XFS | F2FS | Btrfs |
|---|---|---|---|---|---|
| ディスク管理 | Block Group | Block Group | Allocation Group | Segment | Block Group |
| | 最大128M | 最大128M | 最大1TB | 2MB | 1GBなど |
| ブロック管理 | ビットマップ | ビットマップ | B+ツリー | ビットマップ | Bツリー |
| inode番号の管理 | ビットマップ | ビットマップ | B+ツリー | ノードとしてビットマップで管理 | Bツリー |
| inode領域 | 静的 | 静的 | 動的 | 動的 | Bツリー内 |
| マッピング | ブロックマップ | extent | extent | ブロックマップ | extent(圧縮対応) |
| ディレクトリ表現 | リニア | Hash Tree | 5つの形式 | 階層型ハッシュテーブル | Bツリー |
| 整合性の保護 | ジャーナル(ブロック単位) | ジャーナル(ブロック単位) | ジャーナル(操作ログ) | ログ構造化 | Copy-on-Write |

なぜ入りたい会社に入れないのか？

# エンジニアが採用できない会社と評価されないエンジニア（後編）

イラスト：高野涼香

## 求人の罠（キャズム）を回避せよ！




Author 伊勢 幸一（いせ こういち） Twitter @ibucho

◆ 登場人物 ◆

**藤堂**
人材コンサルタント。求める人材は百発百中という気鋭のヘッドハンター

**田草**
人材コンサルティング会社の新米エージェント。元エンジニア

**堀内**
エス社代表取締役社長CEO。ガラケーゲームで一財を成したが……

**梶**
無責任な社風に嫌気がさし、自分の将来に悩みを抱えている

**矢部**
田草の同僚。転職を希望する梶と出会う

**前編のあらすじ**

ガラケーゲームで業界トップに躍り出たエス社。しかし、スマホゲームへの対応と事業方針の転換を誤り、かつての栄光は過去のモノになってしまった。そんな状況に愛想を尽かしやめていくエンジニア達……。人材コンサルタントの田草は社長の堀内から相談を受ける。「いいエンジニアはいないのか？」。田草には一人の男の姿が脳裏に浮かぶ。人材ハンター「藤堂」。彼を呼び出し、堀内と引き合わせた結果、想像もし得ない展開がそこに待っていた。――エンジニアを採用するためにはエンジニアを理解できる人間を役員に据えるべき……。エス社を根底から揺るがす採用についての新しい価値観は、エス社自らの変化を強いるものだった。果たして、いいエンジニアは来るのか？

## プロローグ

――定時を過ぎて数時間が経つ。周囲にはまだ同僚が数人残って残業をしている。私は上司が役員会で報告するための資料作りと、終わりの見えないデバッグ作業に挟まれ、ここ数ヵ月終電前にオフィスを出たことがない……。

「梶君、この計画書の資料作成を週末までにやってくれないか。週明けの役員会で新事業として提案したいのだ」

「それは、どういう事業なのですか？」

「新たにセキュリティ診断サービスを開始したいと思っている」

「セキュリティって、当社にはその経験も実績もないと思います。ましてセキュリティに通じた技術者もいないのではないですか？」

「必要な人員の確保はこの提案書が承認されてからでいい。とにかく今セキュリティが儲かるのだ！」

「セキュリティはそれ単独で成立する分野ではなく、システムのコンポーネントすべてに関する専門性と、そこに潜在する不適正や脆弱性などを熟知している必要があるので、適切に診断ができるエンジニアをそう簡単に確保できるとは思えませんが」

「そこはいい。頭数さえそろえればポートスキャンやXSS（Cross-Site Scripting：クロスサイト

スクリプティング)、SQLインジェクションをテストするツールはオープンソースにあるし、それらを実行してレポートを作成する程度なら多少のIT知識があれば十分だろ」

「それだとセキュリティエンジニアは必要ないのではないでしょうか？　Webフォームから対象サーバを入力してもらって、プログラムで診断ツールを実行し、その結果をJSONかXMLで返すプログラムがあれば十分だと思います」

「どっちでもいい。ただ、プログラムでの自動応答だけだと金にならない。人が診断作業をすることで工数が生まれ、費用を請求できるのだ。プログラムではなく、人が介在していることが重要なのだ。とにかく是非は考えなくていい。ここにある概要書を役員会で配布できるレベルのクオリティに仕上げてくれ。週末までに」

――いつも思い付きで企画を発案し、すぐ気が変わって途中で投げ出す上司。その彼から指示された不毛な資料づくりに時間を取られている。

どうせこの資料はムダになるに決まっている。企画立案を任され自分の調査と判断で作成するならまだましだが、これは上司が頭の中で考えているアイデアを私が変わりに具現化するだけのつまらない仕事だ。

……そもそも今の自分にはこんな資料作成に時間を割いている余裕などない。来週納品予定のシステムに致命的な問題が発覚し、早急にデバッグして原因を特定しコードを修正しなければならない。そのコードを書いたプログラマはすでに先月退職してしまっている。このデバッグを任せられるような同僚もすでに辞めてしまった。ほかの同僚のほとんどは仕様書を忠実にコーディングするだけで、自分で工夫してデバッグやコード解析、テストする気など微塵(みじん)もない。昨年思いがけず大規模な開発案件が舞い込んだとき、慌てて頭数だけ増やすために採用しまくったときの社員ばかりが残っている。

おかげで私の業務範囲は広く多くなり、その責任だけは増えているが、私への評価も報酬にも反映されてはいない。実装やテストについて相談したり、ともに技術を高めたり、新しい技術を一緒に検証してくれる同僚など1人もいない。

オフィスの明かりが突然消えた。振り返るとさっきまで残業していたほかの同僚の姿が居なくなっていた。部屋の電気を消したのか。誰一人、一言も私に声を掛けずに……。

なぜ入りたい会社に入れないのか？

# エンジニアが採用できない会社と評価されないエンジニア

## 求人の罠（キャズム）を回避せよ！

## 転職

「矢部さん、お客様がいらっしゃいました」
「はーい。ありがとうございまーす」

約束の時間より5分ほど早く顧客が来たようだ。どうやら彼は誠実な男のようだ。

私（矢部）は人材コンサルタント会社3年目のエージェント。この会社ではおもに転職を希望するITエンジニアのキャリアアドバイスを担当している。先輩の田草とコンビを組んでいるのだが、田草はクライアントへ営業で外出しているようだ。

今回の顧客はエンタープライズ系システムの受託開発と運用代行をしているSI会社に新卒で就職して4年目のエンジニアと聞いている。履歴書によると大学ではニューラルネットワークを卒論のテーマとし、その研究中にシミュレータを自作してプログラミングに興味を持ち、AIのプログラマを目指すことになったそうだ。

――新卒で入社して3年、キャリアアップを目指した転職を希望。比較的技術力向上に積極的でさまざまな文献を自費で購入して勉強しているらしい。

来客が通された応接室へ向かう前に、その通路の途中にある個室のドアをノックした。中から返事はなかったが、擦りガラス越しに黒いスーツの人影が見える。私はドアを静かに開け中の人影に声を掛けた。

「ミスター藤堂、梶さんがお見えになりました。ご同席をお願いします」

ミスター藤堂は当社のエージェントのオブザーバとして契約を交わし、定期的に私達エージェントに対してアドバイスや指導をしてくれている。今回は偶然にもエージェント研修があり、彼が講師として来社していたので、梶氏との面談に同席をお願いした。ミスター藤堂を梶氏の待つ応接室へ案内し、ドアをノックして扉を開けた。

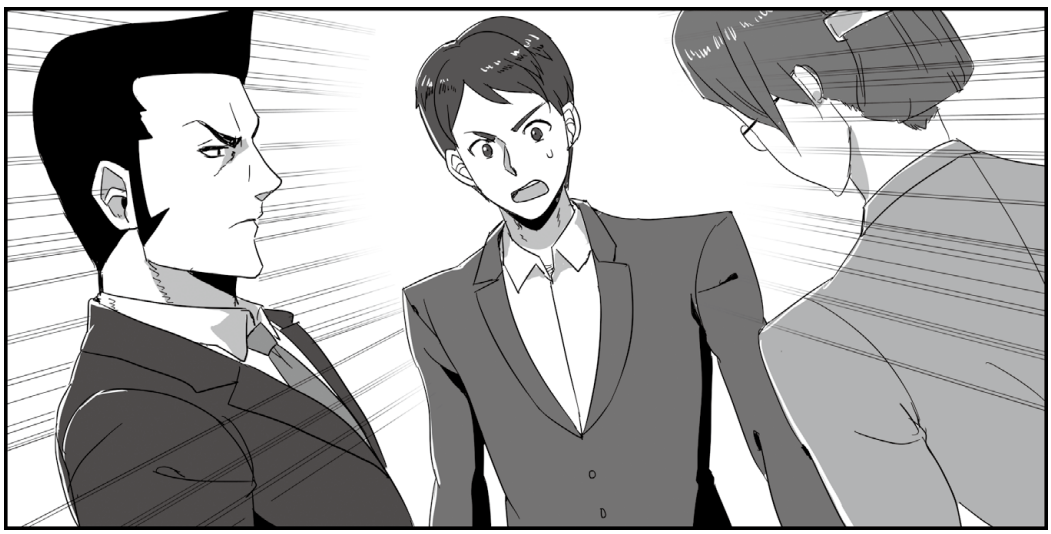
「梶さん、お待たせしてしまいました。本日はご足労いただきありがとうございます」

私達2人が室内に足を踏み入れると、2人がけのソファーに座っていた青年がすっと立ち上がった。襟元のボタンだけを外した白いワイシャツに紺のジャケット、濃いグレーのスラックス、足元はテーブルの影に隠れて見えないが、おそらくダークブラウンの革靴を履いているだろう。髪は短めで前髪が自然に左右へ流れるようにとかされ清潔感の観点では申し分ない。そして彼はそこで深々とお辞儀をした。

「梶さん、こちらがミスター藤堂です。ミスター藤堂、こちらが先ほどお話させていただいた梶さんです」
「ミ、ミスター藤堂？」

ミスター藤堂は無言で梶氏をソファーに座るように促し、そして彼の正面の椅子に自分も着座した。私も続いてミスター藤堂の隣の椅子に腰掛けた。

「なぜ転職を考えた？」



珍しくミスター藤堂のほうから口火を切った。梶氏は突然の問いかけに、うつむき加減だった顔を上げ、ちょっと怯えたような目を見開いて、ミスター藤堂を見た。そして、あらためて背筋を伸ばし、短く息を吐き、そして目線を変えず、訴えかけるような姿勢で話をし始めた。

「私は新卒で今の会社に入社し、丸３年間、システム開発と運用の現場で働いてきました。昨年ぐらいからチームを率いるようになり、数人のスタッフを取りまとめています。入社してから今までの仕事はほとんどが受託開発業務で、クライアントから要求されるざくっとした仕様やイメージからシステムの要件定義、概要設計、モジュールの詳細設計、運用設計まで行い、それらをスタッフと一緒に開発し納品し、場合によっては運用も行っています」
「普通だな」

「はい、ごく普通のSIerの業務だと思います。ただ、最近その業務に達成感を感じなくなってきました。受託案件で利用する言語やフレームワークは基本的に会社が今まで使ってきた環境をそのまま継承しています。時に別の言語やフレームワークで依頼されることもあるのですが、新しい開発環境で受託するとその学習コストが工数となって原価を上げてしまうため、営業側が顧客を説得して従来通りの言語とフレームワークで受託してしまうのです。新しい技術や言語、フレームワークで開発してみたいと個人的に学習しても業務とは認められません。私は新しいフレームワークや言語などを試したり検証したりはしていますが、実際の業務で活用してみないと適不適や性能、バグ、実用性など真の技術力として身につかないものです。私はもっといろいろな技術や実装、言語フレームワークを学び習得したいのです。いろいろな参考書を買い、自宅でさまざまな言語や実装を試したり、レンタルサーバ上で実行したりしますが、どうもそれだけでは素人の趣味の範囲を超えられないと感じるのです。業務として、仕事として責任を持ってそれらさまざまな新しい技術を習得しつつ活用してサービスやアプリケーションを作り、そして運用しつつチューニングしていくような仕事につきたいと思い、今回ご相談に伺いました」

滑舌（かつぜつ）良く、はっきりと落ち着いた話具合から察する限り、彼の言うことに嘘はないだろう。エンジニアの転職の中でスキルアップ、キャリアアップを図りたいという理由がもっとも多い

なぜ入りたい会社に入れないのか？

# エンジニアが採用できない会社と評価されないエンジニア

## 求人の罠（キャズム）を回避せよ！

（前編を参照）。彼もまた同じように自身の技術力を高め、新たな技術に挑戦したいという思いが転職を考えさせたのだろう。

「それだけか？　隠しごとはやめてもらおう」

ミスター藤堂には彼の話以外にも理由があるのではないかと感じたようだ。確かに梶氏の表情には単により高い技術者を目指しているだけのエンジニアとしては、目の表情に若干の焦りと憤りの色が見て取れる。純粋な向上心だけを願う目にはもっとハッキリと瞼（まぶた）の奥から眼光を示すものだ。

「転職理由はそれだけではありません」

### 不信感

梶氏はふたたび語り始めた。梶氏によると会社のサービス企画や受注方法などの事業姿勢に不誠実を感じるとのことだ。顧客の立場で最適なソリューションを提供するとは言っているが、実際には不適切に過剰な設計で工数を水増し、同時に開発コストや人件費を極限まで削って利益を確保しようとしている。また提案も顧客にとって最適であるとは限らず、どちらかというと受注する会社側の都合や利益を優先した内容になる場合が多い。

さらに実績も経験もないサービスを新規事業としてでっちあげ、真摯に計画を検討せず準備にもあまり時間をかけず、適当なクオリティで顧客に提供しようとしている。法的に問題があるわけではないが自身の職務に社会的正当性を見いだせなくなったそうだ。

また新事業や案件があるたび慌てて人の募集をし、必要と思われる頭数だけ集め、不要になったら能力不足を理由に減俸するなどの評価を下し、自ら退職していくよう仕向ける気配がある。いつかは自分の番が来るという恐怖観念に悩まされながら仕事をしていくことが大きなストレスとなっているようだ。会社では優秀な友人を紹介し、採用に至ったならば50万円の報奨金を出すというキャンペーンも行っているが、当然能力の高い友人をこの会社へ誘いたくないし、金で友人を売る気にもなれない。

「事業ありきの人事体制か」

会社の事業と人事に関する体制は大きく2つに分けられる。まず事業があり、そして必要な雇用を行う事業優先型と、まず人材を集めそこから生まれるものを事業化していく人材優先型である。梶氏の会社は典型的な事業優先型であり、この類の企業では社員、つまり人やエンジニアをモノ扱いする傾向にある。端的に言うと社員は給料というお金で買うモノであり、お金の流出を止めたければ解雇すれば良いという認識だ。

当然ながらエンジニアを含め、社員をモノ、つまり事業にとってのコストと考えている企業は社員の能力や可能性に対する評価などできるはずもなく、評価軸は工数コストにヒットする人件費だけである。

### 人材

「おまえは社員になりたいのか、人材でありたいのか？」

ミスター藤堂はいつもと同じく、瞑想しているかのように細めた目からじっと梶氏を見つめ、そして呟くように問いかけた。

「雇用されるのが社員、投資されるのが人材だ」

この概念は、ミスター藤堂が人材を求めている企業に対して採用の概念を認識させるための決まり文句だ。

「私は、指示された業務に応じて結果を出しているつもりです。与えられた業務は滞りなく遂行しています」

与えられた業務を遅延なく遂行するだけでは、評価が下がらないまでも上がることは決してない。なぜなら業務を与えた側、つまり上司や会社にとっては業務が完了することを前提として指示しているのであって、それを遂行することは当然であり、期待以上の結果が得られたわけではない。

「今のおまえは会社のコストでしかない」

多くの場合、会社の評価はエンジニアの労働時間やエンジニアの業務によって生み出される売上と利益から査定される。会社や上司が要求した業務を社員が滞りなく行ったとしてもその人件費は成果物に対するコストであり、事業面から考えるとコストは安いほうが利益率が高い。高度な技術力を持っていたり、新しい技術を利用したり、チームの技術力と士気を上げるといった、コストと利益に反映しない要素は一般的に考慮されない。

「私は定時後に技術の勉強会に出席したり休日でもエンジニアセミナーに参加したり自宅でコーディングやサーバの実装なども自主的に行ってます」

ミスター藤堂はテーブルの上に置いてある彼のラップトップを手元に引き寄せて開き、何かをタイプした。そしてじっと画面を見ながらキーを押したりボタンを押しながら糸のような細い瞼（まぶた）の中で漆黒の瞳を振り子のように左右させた。すると彼は肩でゆっくりと息を吸い、そして静かにその息を吐くと目線だけを上げて梶氏を見据えた。

「1行もない」

彼は梶氏の氏名と彼のポートフォリオにある所属企業や出身校、得意分野などいくつかの単語を組み合わせたキーワードでネット上を検索したのだ。しかし、彼の氏名を含むサイトやページが1つもなかったのである。

「私は検索結果に出てくるような有名人ではありません。書籍や雑誌記事の執筆経験もありませんし、セミナーに参加することがあっても講演者として登壇するような立場ではありません」
「誰しも最初から著名なわけではない」

現在、さまざまなITエンジニア向けのセミナーや勉強会、ハッカソン、ハンズオンなどの集いがあり、そのイベントを運営したり、講師として招聘（しょうへい）される人は大勢いるが、彼らのすべてが最初から業界内で注目される人物だったわ

けではない。最初は皆、一参加者であったはずだが、時を経るに従い参加者から運営サイドに回り、場合によっては講師として登壇し、またほかのコミュニティやセミナーから講演を依頼されたりするようになったのだ。

「コミットメントとアウトプットだ」

一般参加者がイベントやコミュニティの運営サイドにマイグレーションしていくにはパターンがある。先輩社員や友人がすでにコミュニティの運営だった場合、その先輩や友人に紹介されて運営に参加する場合。とくに紹介者が居なくとも毎回同じカテゴリのイベントに参加し続け、会場内や懇親会などで自分の作業や作品、サービス、興味のある分野などを積極的に伝えたりディスカッションしている間に運営への誘いを受ける場合もある。

ただし、イベントやコミュニティに参加することによってエンジニアとして成長したいのであれば、それ相応のコミットメントが必要である。通常それらのイベントやコミュニティの運営に携わることは基本的に無償奉仕である場合が多く、定時後や休日などのプライベートな時間を割いたり、イベント会場までの交通費や宿泊費も自己負担したりと、ある程度個人でコミットメントをする必要がある。

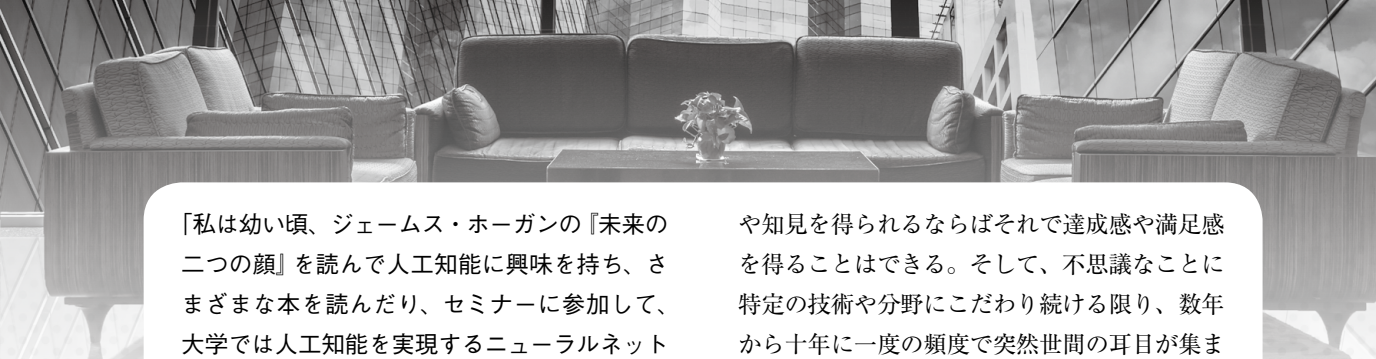
また、単にイベントやセミナーでの交流だけでコミュニティ内の評価を得られるわけではない。エンジニアが携わっている技術や業務に関してどれほど造詣があるかが重要であり、そのエンジニアが公開しているオープンソースや執筆した書籍、投稿記事、発表資料、もしくは個人のブログや会社の技術情報サイトでの寄稿など、具体的な技術スキルや実績が測れるマテリアルが求められる。

つまり、情報共有するに足る経験、知識、知見を持ち合わせているかどうかを判断するエンジニアのアウトプットが必要なのだ。書籍の執筆や講師の依頼を受けるにはある程度実績のあるエンジニアである必要がある。しかし、イベントのライトニングトークや個人ブログに技術情報を載せる行為はとくに資格や過去の実績など必要とせず、本人が情報発信するつもりがある限りいくらでも実行できることだ。そういったカジュアルな情報発信がやがてほかのエンジニアから注目されたり共感されることで、次第にエンジニアコミュニティの中で発言、講演、執筆する機会が巡ってくるものなのである。アウトプットなきエンジニアにインプットもチャンスも与えられることはない。

「しかし、私の現在の業務で使っている技術や環境は特定メーカーのブラウザ使用を前提とした業務管理システムで、そのメーカーの開発環境とフレームワークと機能に依存し、バックエンドにも同社のDBMSを利用した開発ばかりです。今どきのオープン系のエンジニアが興味を持つような要素などありません」
「ツールが何かの問題ではない、要はお前にこだわりがあるかどうかだ」

## 拘泥

エンジニア間でよく話題に上る技術の情報だけに価値があるわけではない。非常にレガシーかつビジネスライクな実装や製品であっても利用者がいる限り、新たな視点での使用方法や便利なプラグイン、新しいバージョンに関する検証結果などは価値のある情報として注目される。問題はそういった知見が得られるまでそれらの環境や実装に当のエンジニアがこだわっているかどうかである。こだわりがあればさまざまな新しい発見や利用法、隠された機能などを見つけ出し、その情報を発信できるだろう。逆にこだわりがなければ必要以上に対象を知ろうとするわけがなく、ほかのエンジニアが気づかなかったテクニックや秘密などを見つけ出すことはまずない。



「私は幼い頃、ジェームス・ホーガンの『未来の二つの顔』を読んで人工知能に興味を持ち、さまざまな本を読んだり、セミナーに参加して、大学では人工知能を実現するニューラルネットワークを研究テーマにしていました」
「……興味を失ったか？」
「今のベンダー依存の実装と開発環境で業務系を開発する仕事にAIを必要とすることなどありません。過去にエキスパートシステムが提唱されたとき一時的にAIが注目されもしましたが、実際に役に立つレベルではなく、すぐにビジネス界から消え去りました。AIとは言ってもせいぜい将棋ゲームかスパムメール駆除に応用されている程度で、AIが人の代わりになるなどSFの世界だけの話です。今の社会にAIなど必要とされてはいません」
「必要かではなく、興味があるのかと尋ねている」
「も、もちろん興味は今でもあります。時々文献を読んだり、ネット上でAI関連の話題や記事を見つけると楽しくなります」
「……ならば突き詰めるがいい」

その時々で注目される技術や流行っている技術を後追いで身に付けたとしても、その分野のイノベーターとして1番になれるわけはない。どんなに努力しても2番以下、つまりアーリーアダプターのポジション以下しか得られない。なぜなら、華やかな技術に気がついて後追いし始めるということはすでにイノベーターグループが形成された後だからだ。業界や世間から注目されていなくとも、エンジニア本人が興味を持ち、こだわりを持って追求し続けるエンジニアだけが、その分野のイノベーターとなれる。

特定の分野でイノベーターであったとしても、その分野が必ずしも業界や世間から注目され、イノベーターが社会や企業から評価されるとは限らない。しかし、興味のない分野を後追いで追っかけ続けた挙句、これといった注目も得ず、評価もされない結果となるより、評価されなくとも興味のある分野を追求し続け、新たな発見や知見を得られるならばそれで達成感や満足感を得ることはできる。そして、不思議なことに特定の技術や分野にこだわり続ける限り、数年から十年に一度の頻度で突然世間の耳目が集まり、その分野のイノベーターが強烈な勢いで注目評価されるものである。肝心なのは自身の興味に従い、こだわり続けることであり、決して現段階での流行や注目を不本意にもかかわらず追っかけまわすことではないであろう。

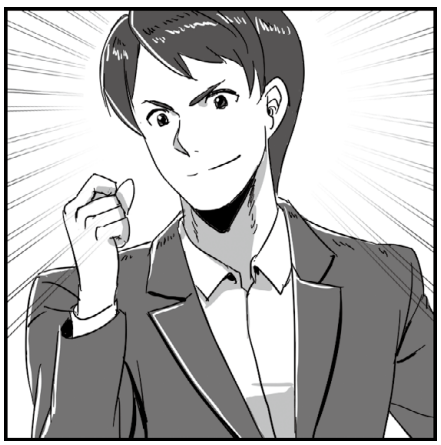
「なるほど、そうかもしれません。興味のない技術を義務的に努力習得してある程度の評価を受けるより、評価されなくとも本当に興味のある分野を突き詰めていったほうがいいですね。自分自身を納得させることができる。わかりました」
「ただし、実務を疎かにするな」

コミュニティや個人の興味のための調査検証などの活動によってエンジニアとして成長するには、あくまで本来の業務における責任を果たしていることが前提となる。本来会社から課せられた業務や義務、責任を疎かにした状態でコミュニティや個人的興味の活動に注力することは逆に社内からの信頼を失い評価を下げることになる。社外活動を通じて技術力が評価され社内で何らかのプロジェクトを任されるまでになったとしても、信頼を失ったエンジニアに協力しようとする同僚はいない。業務というのはたった1人で実施できるものではない。さまざまなステークホルダの理解と協力によって成立するものである。社内的とは言っても一度信頼を失うと、それ以上の実績を上げることなど不可能なのだ。当然、社内での評価や信頼が下がると社外や個人的活動にも大なり小なり影響するものであり、結果的に社内での評価も社外へのコミットメントも下がり、エンジニアキャリアにとって取り返しのつかないダメージになってしまう。

「わかりました。現状の責務を十分に果たしたう

# エンジニアが採用できない会社と評価されないエンジニア

## 求人の罠（キャズム）を回避せよ！



えで、もういちどAI技術に注力してみます。そして、その結果やレベルの良しあしを恐れず、すべてを自分の今の知識知見として広く情報発信していくことにします」

――梶氏は憑きものが取れたかのように生気を発し、目つきは凛とし、口元は引き締まってはいるが、わずかな笑みが見て取れる。

## エピローグ

イベントが開催されている大ホールの入口に設けられた受付で参加者用のストラップとガイドブックを受け取り、同僚の田草と一緒に会場へと足を進めた。冬だというのに会場内には熱気がこもり、通路でもエンジニアがそこかしこに団子を作って議論を交わしている。入口の右側の列には協賛会社の展示ブースがあり、それぞれサービスや製品のカタログを案内したり、社員が調理したという屋台物を振るまったり、簡単なゲームを催し賞品としてレアなノベルティを提供したりしている。

ホールの中央にあるメインステージはオープンになっており、ホールのどこからでも観覧することができ、高さ1メートルくらいのステージ上ではパネルディスカッションが行われている。メインステージの左右に設営されたセッションスペースではさまざまなテーマについてセミナーやカンファレンスが開催されている。このイベントはあらゆるカテゴリに携わるITエンジニアが一同に会し、それぞれエンジニアが興味を持つテーマで自由に議論し情報を共有するITエンジニアの祭典、MINGLE（Mixture of New Generation, Link Engineer）の会場である。

メインステージ前に準備された座席の一番後ろで田草と一緒にステージのセッションを眺めていると、右後ろから快活な足音で近づいてくる人の気配を感じた。

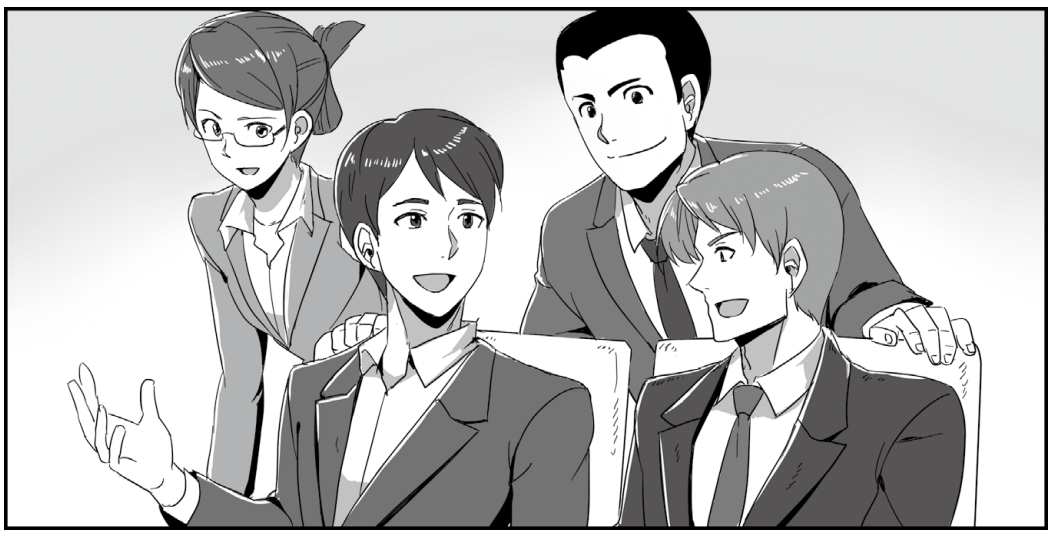
「矢部さんですか？　お久しぶりです」

――振り返ってみると数年前に私が転職コンサルティングを担当した梶氏が明るい笑みを携えて話しかけてきた。

「あら、お久しぶりですね。お名前は最近よく伺ってはいたのですが、お会いするのは本当に久しぶり。今日はご参加？　ご登壇？」
「はい、今日はニューラルネットワークとAIに関するセッションでお話しさせていただくことになっています」
「それはぜひ拝聴させていたただきます。ところで今お仕事は？」
「ええ、幸い昨年転職しまして、今はその会社でAIの開発をしています」
「それはよかったですね。そうそう、紹介が遅れました。こちら私の同僚の田草です」
「田草です。梶さん、よろしくお願いします」
「梶です。こちらこそよろしくお願いします」

3人で取り留めもない世間話をしていたところ、梶氏の後ろ側から控えめに背筋をピンと伸ばしつつ、人懐っこい表情を携えノーネクタイにダークスーツを着込む紳士が近づいてきた。

「おお、田草さん、ご無沙汰です」
「これは、堀内社長、こちらこそご無沙汰しておりました」



「あれ？　社長、こちらの田草さんをご存じなんですか？」
「おや、梶くん、君こそ田草さんと面識があったのか？」
「いえ、私は初対面です。こちらの矢部さんには以前エンジニアキャリアについてアドバイスをいただき、お世話になったことがあるのです」

梶氏は藤堂との面談後、AIや機械学習、ニューラルネットワークに関してふたたび学び直し、細々ではあるが行動してみたのだ。さまざまな機械学習ツールやニューラルネットワークのフレームワークがオープンソースとして公開されていることを知り、それを片っ端から調査した。それらを検証し、数学的根拠による考察などを加えたレポートを個人のブログ上で公開してきた。

そうして2年ほど経過したあるとき、梶のブログがエス社の堀内社長に目にとまった。堀内は興味を持ち、梶が登壇するエンジニアセミナーに足を運び、AIに興味があるということで直接梶氏を口説いたのだ。もちろんその当時のエス社にAIを必要とするようなプロジェクトや事業などなかった。しかし、堀内氏は梶氏にAIの研究に注力するよう依頼し、梶氏もその依頼に答え、やがてWebシステム開発業界でもっとも機械学習とAIに関して知見の深いエンジニアとして認知されるようになってきた。

その後AIによる画像認識の世界コンテストで海外の企業の研究チームが画期的な計算方法によって絶大な高速化を実現し、世界を驚愕させた。その事件をきっかけとして世界中に機械学習とAIのブームが押し寄せ、過去のブームと異なり、現在のコンピューティング性能の向上と相まってAIシステムが俄然現実味を帯びてきた。同時に梶氏の業界内での知名度も上がり、梶氏を慕って優秀なAIエンジニアが次々とエス社に入ってきたのである。

「1年ほど前に、梶君がオンランゲームのGM、つまりゲームマスターをAIを使ってロボット化するプログラムを作ったので自社サービスへ導入してみたのだ。このロボットはプレイヤーからの問い合わせに対し、機械学習の結果から適切な回答を返すだけではなく、そのあとのプレイヤーの行動を分析して返した回答の精度を求め、それをさらにリアルタイムで入力シナプスへのウェイト量の調整にフィードバックするというものだ。長くプレイすればするほど適切な対応をしてくれようになるとプレイヤーからの評判

も上がった。そして、他社からも問い合わせがあり、他社のゲーム向けにAI-GMを提供するという事業が生まれたのだよ。今では携帯ゲームでの直接収益より、AI-GMロボットの販売と運用事業の収益が大きくなったぐらいだ。BtoC事業だけが収益源だった当社にBtoB事業が生まれ、さらにその収益が成長し続けているという状況だ。まったく、梶君を採用したときにこうなるとはまったく期待していなかった、思いもよらないことだよ」

「社長、まったく期待してなかったんですか？ ひどいな」
「いや、それはまあ、言葉のあやであって……」

返答に困る堀内氏を囲み私たち3人は明るく笑った。

「梶くん、そろそろセッションの時間ではないのかね？」
「そうでした。矢部さん、田草さん、お会いできてうれしかったですよ。残念ですが、ここで失礼します。あ、そうそう、来月初めて私の本が出版されます。私の開発したGMエンジンの実装方法をソースレベルで解説した本です。今手持ちが1冊しかないのですがよろしかったらどうぞ」
「ありがとうございます。よろこんで拝読させていただきますよ。梶さん」

――梶氏は笑顔でゆっくりと振り返り、そしてメインステージの方へと歩みを進めた。私はいただいた本を両手で持ち上げ、そして左手で表紙をめくってってみると、扉ページの裏に短い文が書いてあることに気がついた。

「矢部さん、セッション始まるよ」

田草に急かされ矢部は慌てて本を閉じて左手に持ち替え、バッグを拾い上げてメインステージのほうへと向かった。

田草に急かされなければ、矢部は本の扉裏にあるこの言葉を読むはずであった。

「世界最高のエンジニアハンター、親愛なるミスター藤堂へ捧ぐ」

SD


## データサイエンティストの秘密ノート

高橋 威知郎、白石 卓也、清水 景絵 著
四六判／216ページ
1,800円＋税
SBクリエイティブ
ISBN＝978-4-7973-8962-3

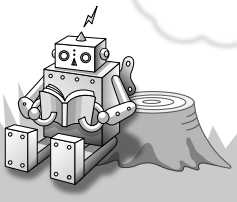
ソフトバンク・テクノロジーが行っている「データアナリティクス」事業。それにかかわるデータサイエンティストが、プロジェクトの中での失敗事例を紹介している。たとえば、「分析目的から逸れた興味本位な分析をしてしまった」「過学習が生じてしまった」「報告資料のコメントがわかりにくいといわれてしまった」などである。紹介する35の事例はすべて「失敗」したものであるが、それぞれの事例では、どのように失敗をとらえ、それを繰り返さないようにしたかという「症状と克服法」が語られている。依頼内容や依頼企業、対象データの詳細などはさすがに明かされていないが、すべて著者らが実際に経験した失敗とのことで、これからデータ分析を行おうとする読者の方、今まさにその途中だという方には大いに参考になるだろう。


## やさしく学べるMySQL運用・管理入門

梶山 隆輔、山崎 由章 著
B5変形判／296ページ
2,600円＋税
インプレス
ISBN＝978-4-2950-0019-8

データベース運用・管理者向けの、MySQLの入門書。RDBMSの基礎から解説が始まり、DBMS初心者にも優しい作りとなっている。また各章の最後では演習問題が設けられ、習得知識の確認ができる。本の内容としては、SQL文の説明やプログラムとの接続といった開発面の話題はほとんどなく、バックアップとリカバリ、レプリケーション、セキュリティ、さらにはパフォーマンスチューニングと、運用・管理に特化したものとなっている。なお本書は、InnoDBの強化、マルチソースレプリケーション、JSONデータ型、パスワードの有効期限設定など多くの変更があったMySQL 5.7に対応している。JSONデータ型については章を設け、そのメリットやJSON関数・JSON演算子の使い方を解説している。

# SD BOOK REVIEW




## Python クローリング&スクレイピング

加藤 耕太著
B5変形判／400ページ
3,200円＋税
技術評論社
ISBN＝978-4-7741-8367-1

膨大なWebのデータを収集し、必要なものだけを抽出するクローリングとスクレイピング。オープンデータの集積と可視化、SNSからの語彙分析、検索エンジンの自作、Webページの自動操作など、この技術の用途は多岐に渡る。Pythonは言語自体の特性や強力なライブラリの存在、収集後の工程との親和性の高さから、この分野で最も広く使われている言語だ。本書では、Pythonによるクローリング・スクレイピングの基礎から実践までを余すことなく解説し、その後のデータの利用方法まで紹介する。強力無比なScrapyフレームワークの活用などの即使える実践的なテクニックはもちろん、キューイングや非同期I/Oなどの汎用的な技術も扱っており、実践的なデータ収集を始めたい人には必携の1冊と言える。


## はじめよう！プロセス設計

羽生 章洋 著
四六判／240ページ
1,980円＋税
技術評論社
ISBN＝978-4-7741-8592-7

本書でいう「プロセス」とは「仕事の組み合わせ」。良い流れでプロセスが組まれていれば、仕事はスムーズに流れ、協業も適切に行われるが、悪い流れなら、その仕事にかかわる当事者がそこかしこでつまずくことになる。またプロセスには、「IT」を利用すると、「イノベーション」を引き起こすことが可能という側面もある。ITシステムはつまり、良い「プロセス」の流れを設計できれば、ユーザにとって魔法のようなパワーをもたらす存在になる。逆に、誰もプロセスに真剣に向き合わないで実現されると、ユーザ不在の末路を迎えることにもなる。本書は、エンジニアとして「プロセス」をどう考え、向き合えば良いか豊富なイラストとともに丁寧に教えてくれる。読む中でビジネスパーソンとして、仕事の本質と向き合うことができるのも魅力。

セキュリティ情報の最前線「Black Hat USA 2016」でトレーニング!

# ペネトレーションテストで学ぶ侵入攻撃の手法と対策


システムを侵入者の手から守るためには、その手口を知ることが重要です。本稿で解説するペネトレーション(侵入)テストによって、システムを守るために必要だと言われている事柄の理由を、より具体的に感じてください。

Author 國信 真吾(くにのぶ しんご) ㈱アーヴァイン・システムズ

## はじめに

2016年7月30日から8月2日の4日間にかけて、アメリカ合衆国のラスベガスでBlack Hat(以下、BHUSA)と呼ばれるセキュリティに関する世界的なカンファレンスが開催されました。

BHUSAでは、企業による展示のほかに講師によるさまざまなトレーニングプログラムを受講できます。企業展示では、セキュリティ脅威から身を守るための製品やサービスを見て回ることができますが、セキュリティ脅威そのものに対する詳細な解説は多くありませんでした。そこでセキュリティ脅威に関する具体的な手法や実例を学ぶため、サーバーインフラストラクチャへのペネトレーションテストのトレーニングを受講してきました。

本稿では、具体的なツールや実行例を元にペネトレーションテストの大まかな流れを紹介することで、攻撃者の手法の一部を知ってもらえればと思います。

## ペネトレーションテスト(PenTest)とは

ペネトレーションテストとは、ネットワークに接続されているシステムに対して、侵入を試み、システムに脆弱性が存在しないかを検証するテストです。ネットワークに接続されたシステムは「格納されている重要なデータを盗み出したい」「システムを破壊してサービスを妨害したい」などさまざまな目的のため、常に外部から悪意あるユーザの攻撃にさらされることになります。ペネトレーションテストの対象はさまざまで、アプリケーションに脆弱性がないかを確認するテストや、今回のようにアプリケーションが稼働するサーバそのものに脆弱性がないかを確認するものもあります。

### WARNINGWARNINGWARNING

### 本稿を読むにあたってのご注意

ペネトレーションテストの実施において、守っていただきたいことがあります。

**ペネトレーションテストを実施する場合は、自身が管理するサーバ・ネットワークのみを対象としてください。**ペネトレーションテストでは、実際の攻撃者が使用するツールを用い、サーバに対して攻撃をしかけます。今回紹介する内容の中には、サービス妨害行為(DoS)に該当する可能性があるものが含まれます。テストを行った結果、大量のアクセスや侵入を試みようとした記録が残ったり、場合によってはサーバに対して重大な障害を引き起こす可能性があります。また、クラウド環境やレンタルサーバ環境では、ペネトレーションテストの実行を禁止しているところも多くあります。ペネトレーションテストの実行を攻撃とみなされ、非常に厳しい処罰を受けることがあります。テストを行う際は、閉じたネットワークなどの安全な環境で試してください。

## トレーニングの概要

トレーニングでは、受講者ごとにKali Linuxがインストールされた攻撃用のサーバが用意されます。受講者は、Kali Linuxサーバを操り、ネットワークに存在する攻撃対象となるサーバを特定し、脆弱性を突いた攻撃を行い、サーバの管理者権限を取得することを目標とします(**図1**)。ネットワーク上に何台、どのようなサーバが存在するかは明らかにされておらず、サーバの特定もトレーニングの最初の課題でした(**図2**)。

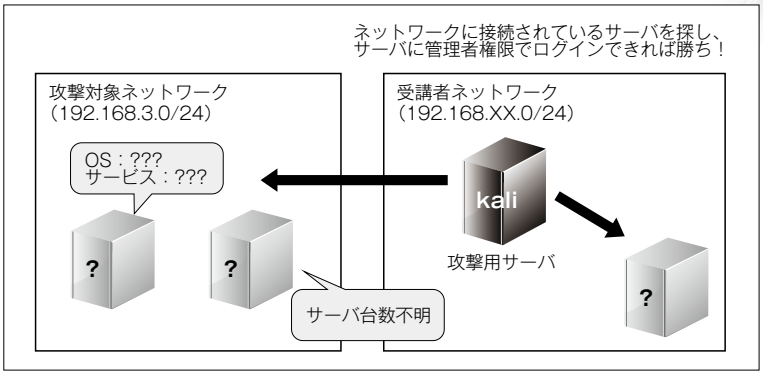
トレーニングの流れは、次のとおりです。

1. Enumeration……ネットワークに存在するサーバの特定、サーバの情報収集(OS、サービスポート、ユーザなど)
2. Strategy……Enumerationで取得した情報を元に、どのような攻撃を仕掛けるか方針を決定
3. PenTest……Strategyに沿って実際の攻撃を実施

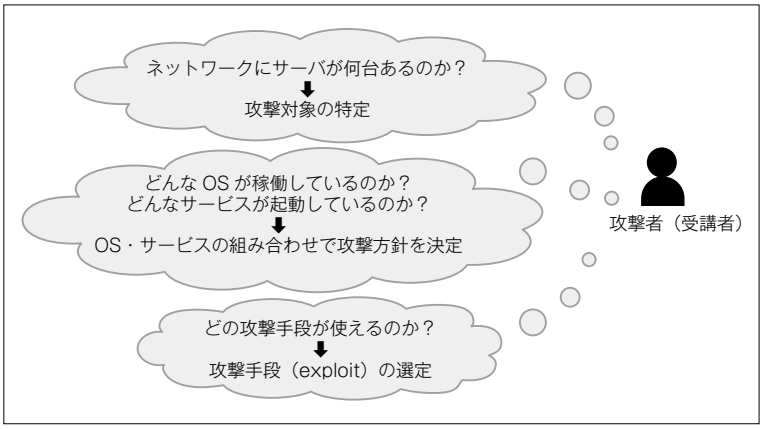
トレーニングでは、PenTestの結果、管理者権限を取得した段階で完了となりましたが、ペネトレーションテストの実際では、発見した脆弱性を報告するReportのフェーズがあります。

以降は、トレーニングの内容を元にしたペネトレーション(ハッキング)テストの簡単な流れについて紹介していきます。

▼図1　トレーニング概要




▼図2　受講者の課題




## Enumeration ——検出——

### ネットワークのスキャン

攻撃用サーバから見たとき、ネットワークは大雑把に次の2つのパターンに分類されます。1つは、自身(攻撃を実行するサーバ)が接続しているネットワーク、もう1つがそれ以外の外部のネットワークです。ネットワークのスキャンでは、対象となるネットワークによって用いるツールが異なります。

#### ■arp-scanによる内部ネットワークのスキャン

攻撃用サーバが接続しているネットワーク上に、ほかにどのようなサーバが存在するかを特定するには、arp-scanと呼ばれるツールを使用します。arp-scanは、ARP(Address Resolution Protocol)と呼ばれるIPアドレスからMACアドレスに変換するために用いられるプロトコルを利用し、ネットワークに存在するサーバの検出を行います。具体的には、arp-scanは、自身が接続されているネットワークのブロードキャストアドレスに対してARPパケットを送信し、応答があった場合サーバの情報を出力し

ます。

**図3**に挙げているのが、arp-scanの実行例です。localnetオプションを指定すると、サーバのネットワーク設定に応じて自動でスキャンするネットワークアドレスやブロードキャストアドレスを設定してくれます。

### ■nmapを使った外部ネットワークのスキャン

arp-scanコマンドは、自身が接続されている直近のネットワークに存在する機器の検出が可能です。しかし、攻撃用のサーバが攻撃対象のサーバと同一ネットワーク上に存在することはまれです。(Free Wi-Fiなど誰でもアクセスできるネットワーク上のサーバを攻撃対象とした場合は別ですが)多くの場合、攻撃者はインターネット網を経由して攻撃対象となるサーバを検出し、侵入を試みます。arp-scanでは、外部のネットワークに対して検出を行うことができないため、別のツールを使う必要があります。

外部のネットワークに対してスキャンをする場合は、nmapというツールを用います。nmapは、PINGによる検出のほかに、後述するサービスポートのスキャンを行うことができます。また、NSE[注1](Nmap Scripting Engine)と呼ばれる機能を用いることで、攻撃(Exploit)を実行することも可能です。

**図4**の出力は、nmapを用いたPINGによるネットワークのスキャン例です。nmapには非常に多くのオプションがあるので、気になった方はmanコマンドで確認してみてください。

注1 https://nmap.org/book/nse.html

## 攻撃対象リストの作成

arp-scanやnmapを用いることでネットワークに存在するサーバのIPアドレスが取得できました。しかし、実行結果にはヘッダ情報や不要な情報が含まれており、そのままでは活用することができません。出力結果からIPアドレスだけを抽出し攻撃対象のリストを作成するには、grepやawkなどのコマンドを使用します。**図5**コマンドは、nmapの実行結果からIPアドレスリストを作成する例です(**図6**)。

## サービスの検出

arp-scanやnmapのPINGによるスキャンで収集

▼図4 nmapを用いたPINGによるネットワークのスキャン例

```
root@kali-kuninobu:~# nmap -n -sn -PE ↵
192.168.2.0/24 | tee /tmp/nmap_ping_result.txt↵

Starting Nmap 7.12 ( https://nmap.org ) at ↵
2016-06-16 22:27 JST
Nmap scan report for 192.168.2.1
Host is up (0.0076s latency).
Nmap scan report for 192.168.2.10
Host is up (0.0089s latency).
Nmap scan report for 192.168.2.15
Host is up (0.21s latency).
Nmap scan report for 192.168.2.40
Host is up (0.21s latency).
Nmap scan report for 192.168.2.46
Host is up (0.20s latency).
Nmap scan report for 192.168.2.47
Host is up (0.0058s latency).
 ... 略 ...
```

▼図3 arp-scanの実行例

```
root@kali-kuninobu:~# arp-scan --localnet | tee /tmp/arp-scan_result.txt↵
Interface: eth0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 256 hosts (http://www.nta-monitor.com/tools/arp-scan/)
192.168.223.1   00:50:56:c0:00:08       VMware, Inc.
192.168.223.2   00:50:56:ec:3d:7b       VMware, Inc.
192.168.223.254 00:50:56:f3:9c:45       VMware, Inc.

3 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9: 256 hosts scanned in 2.901 seconds (88.25 hosts/sec). 3 responded
```

▼図5 nmapの実行結果からIPアドレスリストを作成するコマンド

```
root@kali-kuninobu:~# grep '^Nmap scan' /tmp/nmap_ping_result.txt | awk '{print $5}' > /tmp/server_↵
list.txt
```

したIPアドレスリストを元に、各サーバで稼働しているサービスを検出します。ここでいうサービスとは、HTTPやSMTPなどTCP/UDPポートを使用し、外部から接続を受け付けているサーバプログラムを指します。サーバへ侵入するためには、外部からの接続を受け付けているサービスを利用する必要があるためです。

### ■nmapによるポートスキャンの実行

サービスの検出には、サーバの特定でも使用したnmapがよく使われます。nmapはPINGによるサーバスキャンのほかに、特定のサーバに対してTCPやUDPポートが開放されているかをスキャンすることができます。**図7、8、9**は、nmapによるポートスキャンの実行例です。

nmapを用いると、開放されているポート番号だけでなく、稼働しているサービスやOSの情報なども取得することができます。OSやサービスのバージョン情報がわかると、そのバージョンでは対応されていない脆弱性を突くことができるということを攻撃者に教えることになります。

nmapによるスキャン時間は、スキャンするポート番号の範囲に応じて長くなります。スキャン時間が長くなると、攻撃先に気付かれる可能性が高いため、HTTP(80番)やSSH(22番)など、メジャーなサービスが含まれるウェルノウンポート番号(0～1023)のみを対象にすることが多いです。すべてのTCPポート番号(0～65535)をスキャンする場合は、膨大な時間がかかります。

▼図6　図5で作成したアドレスリストの表示

```
root@kali-kuninobu:~# head /tmp/server_list.txt↵
192.168.2.1
192.168.2.10
192.168.2.15
192.168.2.40
192.168.2.46
... 略 ...
```

▼図7　nmapによるTCPポートのスキャン例

```
root@kali-kuninobu:~# nmap -sS 192.168.2.68↵

Starting Nmap 7.12 ( https://nmap.org ) at 2016-06-17 03:56 JST
Nmap scan report for web01.irvine.lo.2.168.192.in-addr.arpa (192.168.2.68)
Host is up (0.0016s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
3000/tcp open  ppp
3306/tcp open  mysql

Nmap done: 1 IP address (1 host up) scanned in 51.22 seconds
```

▼図8　nmapによるUDPポートのスキャン例

```
root@kali-kuninobu:~# nmap -sU 192.168.2.10↵

Starting Nmap 7.12 ( https://nmap.org ) at 2016-06-17 04:02 JST
Nmap scan report for dns.irvine.lo (192.168.2.10)
Host is up (0.00083s latency).
Not shown: 999 open|filtered ports
PORT   STATE SERVICE
53/udp open  domain

Nmap done: 1 IP address (1 host up) scanned in 4.40 seconds
```

▼図9　サーバのIPアドレスリストを元にスキャンを行う例

```
root@kali-kuninobu:~# nmap -sS -iL /tmp/server_list.txt
```

### ■SNMPによるサービスの検出

サービスを検出するもう1つの方法として、SNMPが利用されることがあります。SNMPは、Simple Network Management Protocolの略で、ネットワーク監視や管理をするためのプロトコルです。外部からCPUの使用率やハードディスクの使用量などサーバの状態を監視するために利用されます。運用監視に役立つサービスですが、攻撃者にとっても非常にジューシーな情報源となり得ます。SNMPv1やSNMPv2cなどの古いバージョンでは、コミュニティ名と呼ばれる1つのパラメータのみで認証を行うため、コミュニティ名を推測できれば、情報を取得できてしまうことがあります。攻撃者はこの認証を突破して情報を盗み出そうとします。コミュニティ名のデフォルト値である、publicは設定しないように注意しましょう。

なお、SNMPの脆弱性のチェックには、onesixtyone[注2]と呼ばれるツールを使用します。これは、SNMPのデフォルトポート番号である161をもじったものです。辞書ファイルと呼ばれる、よく使われるであろうコミュニティ名のリストを元に高速でSNMPへの認証を行うツールです。用意されている辞書ファイルによるチェックをすることで、最低限の脆弱性チェックをすることができます。当然、外部のサーバに対しても実行できるので、使用には十分注意してください。

## Strategy ——戦略を立てる——

ここまでの作業で、ネットワーク上に存在するサーバの検出とサーバで稼働しているサービスの検出をしました。PenTesterは、収集した情報を元に攻撃者が行ってくるであろうサーバへの侵入シナリオを作成します。

侵入シナリオは、検出されたサービスごとに作成していきます。サービスごとに作成する理由としては、攻撃者が使用するツールがサービスごとに特化したものが使用されることや、攻撃に用いる脆弱性がサービスごとに報告・管理されているためです。脆弱性は、CVE[注3](Common Vulnerabilities and Exposures)IDと呼ばれる番号で管理され、データベース化されています。日本では、JPCERT/CC[注4]が注意喚起を行っています。CVEには、

- どのバージョンの
- どのソフトウェアが
- どういった脆弱性を持っていて
- どのくらい危険なのか

といった情報が含まれています。攻撃ツールやセキュリティ診断ツールも、どのCVE IDに対応する脆弱性かを基準にしているものがほとんどです。PenTesterも攻撃者が使用するツールを用意し、脆弱性単位でチェックをすることで、効率よくテストの実施が可能となります。

## PenTest ——実施——

### ▶総当たりによるパスワード認証

脆弱性をついた攻撃例を紹介する前に、もっと原始的な攻撃方法を紹介します。多くのサービスでは、第三者からの侵入を防ぐためにアクセス認証を設けるのが一般的です。SSHでは公開鍵認証によるユーザ認証が一般的になってきましたが、ユーザIDとパスワードがわからない場合は、考えうるユーザIDとパスワードの組み合わせを総当たりでチェックすることで認証をパスすることが可能です。この総当たりで攻撃する方法をブルートフォースアタック(力づくの攻撃)と呼びます。

パスワード認証の脆弱性検証ツールとしては、hydraやmedusaがよく使用されます。hydraやmedusaでは、ものすごい速度でパスワード認証を検証することができます。サーバスペックに影響しますが、秒間数千以上のパスワード認証を試みることができます。また、hydraやmedusaでは、SNMPのスキャンで紹介したonesixtyoneと同様に辞書

注2 https://github.com/trailofbits/onesixtyone

注3 https://cve.mitre.org/
注4 https://www.jpcert.or.jp/

ファイルを用いてパスワード認証を行います。

**{ユーザID, パスワード}**の組み合わせを{a, a}, {a, b}, ..., {abc, a}, {abc, b}, ..., {ZZZZ, ZZZZ}というようにパスワードとして利用できる文字のすべての組み合わせを試していった場合、ユーザIDやパスワードの文字の長さもわからないので、膨大な時間がかかることになります。また、総当たりによる攻撃は大量のエラーメッセージがログに記録されるため、すぐにサーバ管理者に気付かれてしまいます。攻撃者はより少ない試行回数で認証をパスできないかを考えているのです。

hydraは、SSHだけでなく、MySQLやPostgreSQLなど著名なミドルウェアのプロトコルに対応しているため、複数のサービスに対するパスワード認証のチェックに利用できます。

**図10**は、hydraを用いたMySQLサーバへの攻撃例です。パスワード認証に成功した場合は、ユーザIDとパスワードを教えてくれます。hydraは、Kali Linuxではデフォルトでインストールされています。

攻撃に用いられるユーザIDやパスワードの辞書ファイルは、インターネット上に公開されていたり販売されていたりします。有名なものとしては、John the Ripperと呼ばれるパスワードクラック用ツールの作者が作成したものが挙げられます。筆者もいくつかの辞書ファイルをみたことがありますが、英語圏向けのものが多く、案外日本語をもじったパスワードは安全かもしれないという印象を受けました。

## データベースから取得できる情報

ユーザIDとパスワードがわかってしまえば、普通にMySQLサーバに接続します。

```
$ mysql -u foo -pbar -h 192.168.2.181
```

**▼図10　hydraを用いたMySQLサーバへの攻撃例**

```
root@kali-kuninobu:~/dic/en# hydra -l foo -P pass mysql://192.168.2.181:3306↵
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, ↵
or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-06-17 00:46:13
[INFO] Reduced number of tasks to 4 (mysql does not like many parallel connections)
[DATA] max 4 tasks per 1 server, overall 64 tasks, 6 login tries (l:1/p:6), ~0 tries per task
[DATA] attacking service mysql on port 3306
[3306][mysql] host: 192.168.2.181   login: foo   password: bar
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2016-06-17 00:46:14
```

**▼図11　侵入したMySQLでの実行例**

```
mysql> SHOW GRANTS;
+-----------------------------------------------------------------------+
| Grants for foo@%                                                      |
+-----------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'foo'@'%' IDENTIFIED BY PASSWORD 'XXX' |
+-----------------------------------------------------------------------+

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| hoge               |
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

データベースサーバに接続できてしまえば、当然データベースサーバに保存されたデータを取得することができます。接続したユーザIDに与えられた権限や、参照可能なデータベースを確認します。あとは、ダンプコマンドでデータを抽出したり、データを書き換えたりと自由にできます(**図11**)。

## サーバプロセスの実行権限の悪用

データベースに接続することができましたが、ほかに取得できる情報はないのでしょうか。恥ずかしながら、筆者は今回のトレーニングを受けるまでは、データベースに接続されても盗まれるのはデータベースに格納されているデータだけだと思っていました。しかし、接続されたユーザに与えられた権限によっては、データベースサーバが稼働するサーバのファイル内容も取得できることがあるのです。

たとえば、データベースに接続しているユーザに対して、ファイルからデータをテーブルに読み込むための権限が与えられていた場合、データベースサーバを起動しているユーザの権限でファイルの内容を見ることができてしまいます。

たとえば/etc/passwdファイルには、リモートサーバで登録されているユーザの情報が格納されています(**図12**)。つまり、SSHのユーザ認証突破に必要な2つの情報のうち、1つが取得できてしまいます。併せて、SSHの設定ファイルも確認すれば、どの認証方法を使用しているかも確認できてしまいます(**図13**)。

もし、データベースサーバを特権ユーザ(root)で稼働していた場合は、サーバにSSH接続せずにあらゆる情報を読み取ることが可能になります。

このように、サーバプロセスを起動しているユーザの権限を利用した方法を用いると、ファイルの読み出しだけでなく、外部コマンドを実行し、侵入用のユーザを作成することも可能だったりするのです。

◆ ◆ ◆

2日間にかけて行われたトレーニングでは、今回紹介したもの以外にもWindowsサーバへの攻撃、Webサーバへの攻撃などさまざまな方法でサーバの管理者権限を奪取する方法を学びました。攻撃方法を学ぶことで、セキュリティ脅威からシステムを

**▼図12　/etc/passwdファイルにアクセスできた例**

```
mysql> SELECT load_file('/etc/passwd') \G
*************************** 1. row ***************************
load_file('/etc/passwd'): root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
... 略 ...
```

**▼図13　SSHの設定ファイルにアクセスできた例**

```
mysql> SELECT load_file('/etc/ssh/ssh_config') \G
*************************** 1. row ***************************
load_file('/etc/ssh/ssh_config'): #     $OpenBSD: ssh_config,v 1.25 2009/02/17 01:28:32 djm Exp $

# This is the ssh client system-wide configuration file.  See
# ssh_config(5) for more information.  This file provides defaults for
# users, and the values can be changed in per-user configuration files
# or on the command line.

# Configuration data is parsed as follows:
#  1. command line options
#  2. user-specific file
#  3. system-wide file
```

守る立場になったときに、どのような点に気を付ければ良いのか、どう振る舞うべきかについても学ぶことができたと思います。

## セキュリティを高めるために気を付けること

### ❶デフォルト設定のまま運用しない

多くのソフトウェアは、デフォルト設定の状態でも問題なく動作するよう親切な設定がされているものが多いです。しかしここまでで学んできたように、攻撃者がまず狙うのはデフォルト設定やよく使われる設定です。当然デフォルト設定で使っているのだから、きっとセキュリティ対して気を使っていないと悟られてしまうことにもつながります。

### ❷設定内容を正しく理解しよう

はじめてサーバを構築する場合、インターネットで調べて出てきたサイトにある設定をコピペして使ったりしてはいないでしょうか。また、設定内容はよくわからないけど、とりあえず動いてるからいいや、となってはいないでしょうか。攻撃者は、攻撃対象とするサーバについてどのような脆弱性があるのか、場合によってはどのようなプロトコルで通信をしているのか、ソースコードが入手可能な場合は、ソースコードから脆弱性を見つけようとします。自分が設定しようとしている内容と実際の設定内容の対応付けができるようになるべきです。

### ❸OSや権限、プロトコルなど低いレイヤのしくみも理解しよう

攻撃者は、はるかに深い知識を持っている人であるケースも少なくありません。攻撃対象としているサーバに対してどのようにすれば、管理者権限を取得できるのかを常に模索しています。また、本来であれば正しい順序で行われるべきはずであるプロトコルに対しても、もしこのタイミングで異なる手順を踏むとどうなるのか、といった異常ケースによる試行錯誤も行われています。そういった攻撃から身を守るためにも、OSやプロトコルといった低いレイヤの知識も身につけていくべきです。

### ❹情報（ログ）を素早く読めるようになろう

完璧な攻撃でないかぎり、必ず攻撃の痕跡が残ります。それは、ログファイルのように記録されるものであったり、netstatコマンドで確認できるセッションの数だったりします。それらの情報から異変がおこっていないか判断できるようになりましょう。もちろん、すべての情報を目視で確認していくのは不可能に近いので、技術者らしくソフトウェアに頼るべきです。

### ❺英語に慣れよう

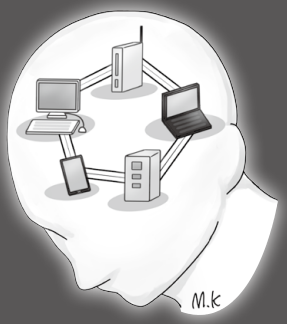
セキュリティに限らず、最新の情報は英語で公開されていることが非常に多いです。また、設定方法を確認しようとしたところ、英語だったということも多いでしょう。このとき気を付けるべきは、翻訳されたドキュメントの場合、情報の欠落や誤りが混じっている可能性があるということです。素早く、正しい情報にアクセスできるようになるため、英語に慣れておくと役立ちます。

## PenTestで確認すべき項目

最後に、今回紹介したツールや手法を元に、皆さんが管理しているサーバに対してぜひ確認いただきたい項目を紹介します。

1. 不要なサービスやポートが稼働していないかを確認しよう（nmap）
2. 脆弱なパスワードを設定していないかを確認しよう（hydra）
3. 脆弱性が報告されている古いソフトウェアを使用していないか確認しよう
4. 管理者権限で外部からの接続を受けるサービスを起動していないか確認しよう
5. 適切なアクセス制限（接続元IPを制限など）が設定されているか確認しよう

本稿が安全なサーバ運用の一助になれば幸いです。SD




## 仮想化の知識、再点検しませんか?

# 使って考える仮想化技術

本連載は「仮想化を使う中で問題の解決を行いつつ、残される課題を整理する」ことをテーマに、小さな仮想化環境の構築・運用からはじめてそのしくみを学び、現実的なネットワーク環境への実践、そして問題点・課題を考えます。仮想化環境を扱うエンジニアに必要な知識を身につけてください。

Author
笠野 英松(Mat Kasano)
有限会社 ネットワーク・メンター
URL http://www.network-mentor.com/indexj.html

## 第9回 仮想マシンの作成

連載後半「仮想ネットワーク環境で使ってみよう〜現実的な使い方」の3回目です。

今回および次回は、前回までに構築した、ホスト環境や仮想ネットワーク環境、そしてセキュリティ環境を使って、仮想マシンの作成とゲストOSのインストールを行います。

今回から現実のネットワーク環境、つまり、仮想ネットワークと実ネットワーク、そして外部インターネットの3つのネットワーク環境下での仮想マシンの利用、運用を考えていきます。

今回はまず、仮想マシンの作成を行います。

### 仮想マシンの作成方法

今回は、仮想マシンをGUIの「仮想マシンマネージャー(virt-manager)」とは別のCUI(コマンド)で作成します。CUIコマンドの作成方法には2つあります。

1つ目はPythonスクリプトの「virt-install」コマンドによる方法です。仮想マシン作成に必要な情報を対話型、またはコマンドパラメータで指定して実行することで、仮想マシンを作成し、ゲストOSのOSインストールまで自動的に引き継ぎます。

2つ目は仮想マシン管理ユーザインターフェース「virsh」コマンドを利用する方法です。仮想マシンの情報をパラメータで指定することはできませんが、仮想マシンのフレームをあらかじめ作成しておき、それを仮想マシンとして定義・登録することで仮想マシンを作成します(この方法の詳細は次回)。

### virt-installによる仮想マシンの作成

virt-installコマンドは、KVMのみならずXenでも使用可能な仮想マシン作成、およびゲストOSインストールのコマンドです。

virt-installの処理は、情報の入力処理および仮想マシン作成と、ゲストOSのインストールの2つのステップに分かれています。

情報の入力処理ステップでは、対話型で全情報を入力することも、全情報をそれぞれパラメータとして一括記述することもできます。

virt-installコマンドの入力情報が正しく、起動が正常であれば(パラメータエラーや起動エラーがなければ)仮想マシンの作成は完了し、コマンドも終了します。ただし、ゲストOSのインストール処理は自動的に始まっています。

ゲストOSのインストールはテキストモードとVNCなどによるGUI操作が可能です。

virt-installコマンドによる仮想マシン作成の例が**図1**のようなものです。

ここでは、使用メモリ1GB、ディスク/dev/sda5、CD/DVD起動、MACアドレス指定、NAT変換(デフォルト)、そして、ホスト起動時の仮想マシン自動起動、などをパラメータ指定して、仮想マシンの作成を行い、Windows Server 2008のisoイメージやVNCを使用して

**▼図1　virt-installによる仮想マシン作成の例**

```
 fdisk領域確認（fdisk領域作成は図3）
[root@vhost1 ~]# fdisk -l
デバイス ブート       始点        終点     ブロック   Id  システム
/dev/sda1   *           1          66      524288   83  Linux
/dev/sda2              66        5288    41943040   83  Linux
/dev/sda3            5288        5940     5242880   82  Linux スワップ / Solaris
/dev/sda4            5940       12161    49972000+   5  拡張領域
/dev/sda5            5940        7898    15729421+  83  Linux

 windows-server-2008_r2_sp1_x64.iso確認
[root@vhost1 ~]# ls -al /iso
合計 2907628
drwxr-xr-x   2 root root       4096 12月  4 11:55 2016 .
dr-xr-xr-x. 27 root root       4096 12月  4 12:10 2016 ..
-rw-r--r--   1 root root 2977398784 12月  4 12:00 2016 windows-server-2008_r2_sp1_x64.iso

 virt-installスクリプト実行による仮想マシン作成
[root@vhost1 ~]# ./virt-install_1
virt-install svm1 (win2k8)
#!/bin/bash
echo "virt-install svm1 (win2k8)"
cat $0
virt-install --hvm --name=svm1 --ram=1024 --disk path=/dev/sda5 --cdrom=/iso/windows-
server-2008_r2_sp1_x64.iso --mac='52:54:00:9a:90:01' --autostart --os-variant=win2k8
--network network=default --accelerate --graphics vnc,port=5911,listen='0.0.0.0',
password=Password1,keymap=ja --video=cirrus --noautoconsole --wait=0

Starting install...
ドメインを作成中...                                |    0 B     00:00
Domain installation still in progress. You can reconnect to
the console to complete the installation process.
[root@vhost1 ~]#

 ドメインxmlファイル確認（内容はリスト1）
[root@vhost1 ~]# ls -al /etc/libvirt/qemu
合計 20
drwx------. 4 root root 4096 12月  4 12:15 2016 .
drwx------. 4 root root 4096 12月  3 12:37 2016 ..
drwxr-xr-x  2 root root 4096 12月  4 12:15 2016 autostart
drwx------. 3 root root 4096 12月  3 12:37 2016 networks
-rw-------  1 root root 3146 12月  4 12:15 2016 svm1.xml

 VNCによる仮想マシン上のゲストOSインストールへ
[root@vhost1 ~]# vncviewer localhost:5911 &

TigerVNC Viewer for X version 1.1.0 - built May 11 2016 13:00:50
Copyright (C) 1999-2011 TigerVNC Team and many others (see README.txt)
See http://www.tigervnc.org for information on TigerVNC.
 ...略...
[root@vhost1 ~]#
```

ゲストOSインストールを行っています。作成された仮想マシン設定xmlファイルが**リスト1**です。

**virt-install（表1、図2）の特徴：**

① PV（Paravirtualized、準仮想化）とFV（Full virtualized、完全仮想化）に対応

② 2つのフェーズ処理：仮想マシン設定・作成とゲストOSインストール

③ 仮想マシン設定方法：対話型（--prompt指定）と一括記述型（--prompt指定なし）

④ゲストOSインストール方法：テキスト操作（PV

時のみ)とグラフィクス操作(VNC)
⑤ グラフィクス操作:VNCサーバ(その仮想マシンIPアドレス:指定ポート)にアクセス
⑥ 仮想マシンの削除:仮想マシンの削除と(イメージの場合)ストレージイメージの削除

## 仮想マシン管理virsh

virshは、仮想マシンAPIライブラリlibvirtパッケージ内の「仮想マシン管理ユーザインター

**▼リスト1　virt-installにより作成されたネイティブxmlファイル(/etc/libvirt/qemu/svm1.xml)**

```
<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made using:
  virsh edit svm1
or other application using the libvirt API.
-->

<domain type='kvm'>
  <name>svm1</name>  ドメイン名=仮想マシン名
  <uuid>f496e029-d208-85e7-8a58-dd3614cdaeb2</uuid>  仮想マシンUUID
  <memory unit='KiB'>1048576</memory>                KBメモリサイズ
  <currentMemory unit='KiB'>1048576</currentMemory>  現在メモリ
  <vcpu placement='static'>1</vcpu>                  仮想CPU数
  <os>
    <type arch='x86_64' machine='rhel6.6.0'>hvm</type>   仮想システム情報
    <boot dev='hd'/>   HD起動
  </os>
 ...略...
  <clock offset='localtime'>  TIME設定
    <timer name='rtc' tickpolicy='catchup'/>
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='block' device='disk'> ストレージ
      <driver name='qemu' type='raw' cache='none' io='native'/>
      <source dev='/dev/sda5'/>       実デバイス(作成は図3)
      <target dev='hda' bus='ide'/>   仮想デバイス
      <address type='drive' controller='0' bus='0' target='0' unit='0'/>
    </disk>
    <disk type='file' device='cdrom'> CD/DVD
      <driver name='qemu' type='raw'/>
      <source file='/iso/windows-server-2008_r2_sp1_x64.iso'/> 実isoイメージ
      <target dev='hdc' bus='ide'/>   仮想デバイス
      <readonly/>
      <address type='drive' controller='0' bus='1' target='0' unit='0'/>
    </disk>
 ...略...
    <interface type='network'>            NIC
      <mac address='52:54:00:9a:90:01'/>  MACアドレス
      <source network='default'/>         デフォルトネットワークタイプ=NAT
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
 ...略...
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='5911' autoport='no' listen='0.0.0.0' keymap='ja' ➥
passwd='Password1'> グラフィクスVNC
      <listen type='address' address='0.0.0.0'/> listenアドレス
    </graphics>
  </devices>
</domain>
```

フェース」です。KVMのほか、Qemu、Xen、LXC、OpenVZ、VirtualBox、VMware ESXiなどにも対応しています。

virshは仮想マシンの起動やシャットダウンなど、仮想マシンのすべての操作管理を行います(**表2**)。また、次回に解説する、既存仮想マシンxmlファイルからの新規仮想マシン作成にも使用します。

## 仮想マシン作成中の問題と対策

virt-installコマンドによる仮想マシン作成中に起こるトラブルシュートの例です。

### 論理パーティションが作成できない

仮想マシン用の追加パーティション/dev/sda5を作成できないことがあります。

#### 対処策(図3)

KVM物理ホストのインストール時、ハードディスク/dev/sda内は3つの基本パーティション/dev/sda1、/dev/sda2、/dev/sda3、だけなので、/dev/sda5は拡張パーティション/dev/sda4を作成してから論理パーティションとして追加作成します。

### ディスクデバイスのエラー

virt-installコマンドがストレージパラメータのエラーで異常終了することがあります。

```
size is required for non-existent disk '/dev/sda5'
```

**▼表1　virt-installコマンドの主なパラメータ(https://virt-manager.org/)**

| パラメータの書き方 | 説明 |
|---|---|
| --prompt | 対話型設定。対話型時、ほかのパラメータは無効 |
| --name=仮想マシン名 | 例)VM2 |
| --ram=メモリサイズMB | 例)4096 |
| --disk path=仮想マシンデバイス　または<br>イメージ[, size=イメージ新規作成サイズGB] | 例)/dev/sda5、/var/lib/xen/images/VM2.img, 10GB |
| --autostart | ホスト起動時の仮想マシン同時起動 |
| --os-type=OSタイプ | 例)solaris |
| --os-variant=OS種別(os-type省略可) | 例)solaris10 |
| --network bridge=ブリッジ名　または<br>--network network=default(NAT) | 例)virbr0 |
| --mac=MACアドレス | 例)'52:54:00:00:00:02'(KVM-OUI先頭3オクテット共通) |
| --accelerate | カーネルアクセラレータ使用 |
| --location=インストールURL | 例1)'http://127.0.0.1/media/CentOS_6.5_Final'<br>例2)'http://ftp.iij.ad.jp/pub/linux/centos/6.5/isos/x86_64/' |
| --graphics none(テキスト操作) | ー |
| 　　または<br>--graphics vnc(グラフィクス操作) | |
| 　,port=ポート番号 | 例)5911 |
| 　,listen=待機アドレス | 例)'0.0.0.0' |
| 　,password=パスワード文字 | ー |
| 　,keymap=キーボードマップ | 例)ja |
| 　,passwordvalidto='UTC時間有効期限' | 例)'2016-10-09T15:51:00' |
| --noautoconsole | virt-install実行時にコンソール/vncに自動接続しない |
| --wait=0 | virt-install実行後、すぐに終了する。プロンプトを返す |
| -v, --hvm | 完全仮想化方式の指定、必須 |
| --cdrom=インストールCD/DVDデバイス | 例)/dev/sr0(旧/dev/cdrom) |
| --cdrom=インストールISOメディア | 例)/osmedia/solaris11.iso |
| --location=インストールイメージディレクトリ | 例)/media |
| --pxe | PXEインストール |

**▼図2　virt-installコマンドの指定例**

```
①対話型(最低限の設定のみ)
virt-install --prompt
※質問選択
・Would you like a fully virtualized guest (yes or no)? This will allow you to run
 unmodified operating systems.
(入力例) yes
・仮想マシンの名前は何ですか?
(入力例) VM1
・どれだけのRAMを割り当てますか(メガバイト単位で)?
(入力例) 2048
・What would you like to use as the disk (file path)?
(入力例) /var/lib/xen/images/VM1.img
・どの位の大きさのディスク(/var/lib/xen/images/VM3.img)にしたいですか(ギガバイト単位で)?
(入力例) 40
・インストールCD-ROM/ISOイメージ、もしくはURLの場所はどこですか?
(入力例) /dev/sr0
⇒仮想マシン作成処理⇒ゲストOSインストール(virt-viewer)

②一括記述型/VNC操作……「図1　virt-installによる仮想マシンの作成の例」参照
```

**▼表2　主なvirshサブコマンド**

| 番号 | 書式 | 説明 |
|---|---|---|
| 1 | autostart[ --disable]ドメイン名 | ドメインの自動起動設定[解除](※) |
| 2 | consoleドメイン名 | ゲストのコンソールに接続 |
| 3 | create XMLファイル [--console] | XMLファイルからドメインの作成[および、接続] |
| 4 | define XMLファイル | XMLファイルからドメインを定義登録(ただし起動しない) |
| 5 | undefineドメイン名 | 停止状態のドメインの定義削除 |
| 6 | destroyドメイン名 | ドメインの強制停止 |
| 7 | editドメイン名 | ドメインのXML設定を編集 |
| 8 | startドメイン名 [--console] | (以前に定義した)停止状態のドメインの起動 |
| 9 | shutdownドメイン名 | ドメインを穏やかに停止。通常稼働中のRHEL/CentOSのみ有効 |
| 10 | rebootドメイン名 | ドメインの再起動 |
| 11 | suspendドメイン名 | ドメインの一時停止 |
| 12 | resumeドメイン名 | suspendドメインの再開 |
| 13 | setmemドメイン名 KBサイズ | メモリサイズの変更 |
| 14 | setvcpusドメイン名 仮想CPU数 | 仮想CPU数の変更 |
| 15 | list [--inactive \| --all] | 稼働ドメインの一覧を表示。[停止ドメイン\|全ドメイン] |
| 16 | dominfoドメイン名 | ドメインの情報 |
| 17 | domstateドメイン名 | ドメインの状態 |
| 18 | domidドメイン名 | ドメイン名またはUUIDをドメインIDに変換 |
| 19 | domnameドメインID | ドメインIDまたはUUIDをドメイン名に変換 |
| 20 | domxml-from-native フォーマット ネイティブ設定ファイル | ネイティブ設定をドメインXMLに変換。フォーマット:xen-xm(xmフォーマット)、xen-sxpr(sxpフォーマット) |
| 21 | domxml-to-native フォーマット xmlファイル | ドメインXMLをネイティブ設定に変換。フォーマット:xen-xm(xmフォーマット)、xen-sxpr(sxpフォーマット) |
| 22 | dumpxmlドメイン名 | XML形式のドメイン情報表示 |
| 23 | net-edit default | ドメインデフォルトネットワーク設定の変更 |

(※)「/etc/libvirt/qemu/ネイティブxml設定ファイル」のシンボリックリンクを「/etc/libvirt/qemu/auto/」内に作成。

▼図3　fdiskによる論理パーティション作成

```
[root@vhost1 ~]# fdisk /dev/sda  fdiskによる15GBの/dev/sda5作成
 ...略...
コマンド (m でヘルプ): n
コマンドアクション
   e   拡張
   p   基本パーティション (1-4)
e
選択した領域 4
最初 シリンダ (5940-12161, default 5940):
Using default value 5940
Last シリンダ, +シリンダ数 or +size{K,M,G} (5940-12161, default 12161):
Using default value 12161

コマンド (m でヘルプ): n
最初 シリンダ (5940-12161, default 5940):
Using default value 5940
Last シリンダ, +シリンダ数 or +size{K,M,G} (5940-12161, default 12161): +15G

コマンド (m でヘルプ): p
 ...略...
デバイス ブート      始点         終点      ブロック   Id  システム
/dev/sda1   *           1          66      524288   83  Linux
パーティション 1 は、シリンダ境界で終わっていません。
/dev/sda2              66        5288    41943040   83  Linux
/dev/sda3            5288        5940     5242880   82  Linux スワップ / Solaris
/dev/sda4            5940       12161    49972000+   5  拡張領域
/dev/sda5            5940        7898    15729421+  83  Linux   追加論理パーティション/dev/sda5

コマンド (m でヘルプ): w
パーティションテーブルは変更されました！

ioctl() を呼び出してパーティションテーブルを再読込みします。

警告: パーティションテーブルの再読込みがエラー 16 で失敗しました: デバイスもしくはリソースがビジー状態です。
カーネルはまだ古いテーブルを使っています。新しいテーブルは
次回リブート時か、partprobe(8)またはkpartx(8)を実行した後に
使えるようになるでしょう
ディスクを同期しています。
[root@vhost1 ~]#
```

拡張領域

追加論理パーティション

## 対処策

仮想マシン用のストレージデバイスが確保されていないので、fdiskなどでディスク領域を確保します(**図3**)。

なお、ディスク領域を確保した後も、そのまま、virt-installすると同じエラーとなります。必ず、システムの再起動が必要です。

**Note　KVMで使用するMACアドレス**

KVMのNICで使用するMACアドレスの先頭3オクテットOUI(Organizationally Unique Identifier、管理組織識別子)は、IEEEには登録されていませんが、qemu/kvmでは「52:54:00」と規定しています。

## 次回予告

次回も引き続き、仮想マシン作成を行います(virsh/dumpxmlによる方法)。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12回 EAVや非正規形のテーブル設計を少しずつ修正する方法

前回は、SQLのアンチパターンの1つであるEntity Attribute Value（EAV）を取り上げました。このような設計はできるだけ避けたいものですが、悲しいかな、すでにこんな下手な設計で作られたシステムが存在するのも事実です。そこで今回は、そんな設計を少しずつ修正し、技術的負債を解消していくアイデアを紹介します。

登場人物紹介

生島氏
DBコンサルタント。性能トラブルの応援のため大道君の会社に来た。

大道君
浪速システムズの新米エンジニア。素直さとヤル気が取り柄。

五代氏
大道君の上司。プロジェクトリーダーでもある。

## EAVのコードはメンテナンスしづらい

大阪を中心に20年間システム開発に携わったあと、現在は東京で仕事をしている「SQLの伝道師」ことジーワンシステムの生島です。

「生島さん、ちょっとこれ（**リスト1**）見てもらえませんか」といつものように大道君が相談にやってきました。

「おー、これもEAVやね。どうしたん？」

「既存のコードでこういうのが使われているのを見つけまして……前回の話もあるので、使わないように修正しておこうかと思うんです」


EAV（Entity Attribute Value）というのは前回触れたSQLアンチパターンの1つです。通常ならば「別な種類の値は、別なカラムに格納する」ようにテーブル設計をするのがRDB設計の原則ですが、「値の種類（属性名）を示すカラムと値そのものを格納するカラム」の2つのカラムで汎用的な意味を持たせるような設計とすることでカラムやテーブルの種類を減らす方法のことです。**リスト1**のコードの場合は、type_namesがEAV方式のテーブルで、type_nameで属性名を示し、type_valueに値を格納します。

社内で開発しているECシステムの受注業務モジュールを引き継いだプログラマが、SQLの読みにくさに悩んで大道君に愚痴をこぼしたのがきっかけで発見したそうです。

「どこが読みにくいと思った？」

「SELECT文のFROM句を見て、頭が？？？だらけになりました。やっていることはこういうことですよね？」

と言って大道君が見せてくれたのが**図1**です。**リスト1**のSELECT文の処理ロジックを見える化するとこうなるわけです。

**▼リスト1　EAV方式を名称マスタテーブルに使っている例**

運送会社と配達時間のマスタデータ生成

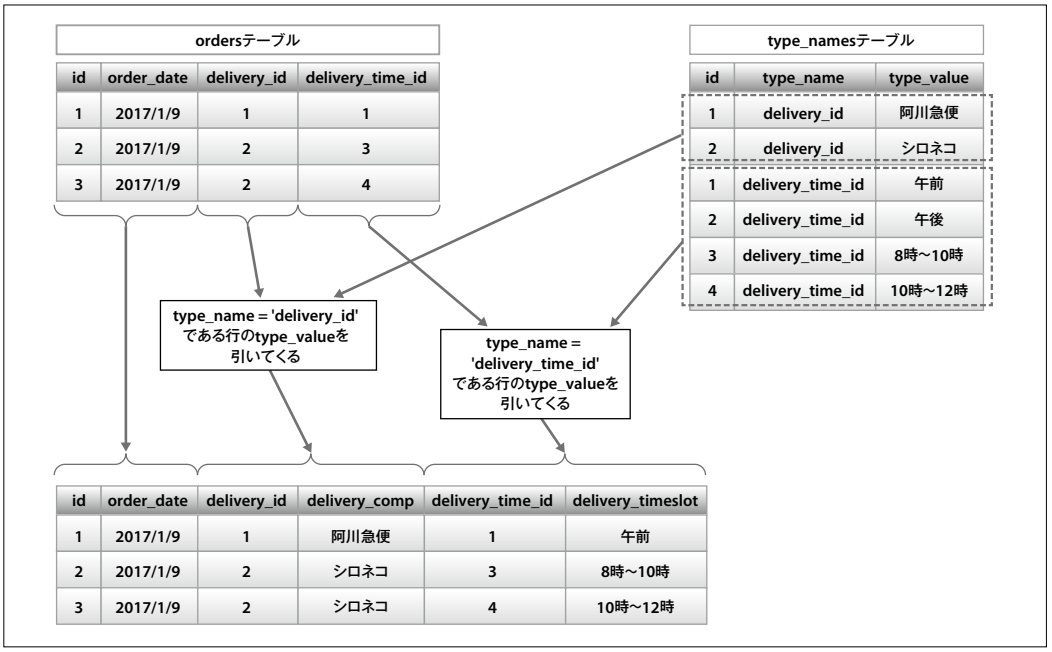
```
CREATE TABLE orders (
  id    SERIAL PRIMARY KEY
  , order_date DATETIME NOT NULL
  , input_employee_id BIGINT NOT NULL
  , delivery_id INT NOT NULL       -- 運送会社
  , delivery_time_id INT  NOT NULL  -- 配達時間
  (..略..)
);

CREATE TABLE type_names (
  id  INT NOT NULL
  , type_name  VARCHAR(100) NOT NULL
  , type_value  TEXT
  , PRIMARY KEY (id, type_name)
);

INSERT INTO type_names(id, type_name, type_value)
VALUES
  (1, 'delivery_id', 'クロネコヤマト')
  , (2, 'delivery_id', '佐川急便')
  (..略..)
  ;

INSERT INTO type_names(id, type_name, type_value)
VALUES
  (1, 'delivery_time_id', '午前')
  , (2, 'delivery_time_id', '午後')
  , (3, 'delivery_time_id', '8時～10時')
  , (4, 'delivery_time_id', '10時～12時')
  (..略..)
  ;
```

運送会社と配達時間を参照するコード

```
SELECT
  o.id, o.order_date
  , o.delivery_id, n1.type_value
    AS delivery_comp
  , o.delivery_time_id, n2.type_value
    AS delivery_timeslot
  (..略..)
FROM orders o
  INNER JOIN type_names n1
    ON o.delivery_id = n1.id
    AND 'delivery_id' = n1.type_name
  INNER JOIN type_names n2
    ON o.delivery_time_id = n2.id
    AND 'delivery_time_id' = n2.type_name
  (..略..)
;
```

同じ名前が別な
意味に多用されて読みづらい

**▼図1　EAV方式のテーブルのJOIN処理**



「そうそう、こういうことやね。EAV使うとどうしてもこういうふうになるわけや」

「type_valueという1つのカラムの情報が、delivery_compとdelivery_timeslotという別なカラムに使われていくというのはどうも馴染めません」

この連載の第2～3回あたりでも触れましたが、SQLはテーブルの一部をタテ方向でもヨコ方向でも自在に切り出して「集合的に扱う」ための言語です。その意味では**図1**でも「テーブルの一部を切り出して」使っていることには違いありませんが、EAV方式を使った場合のSQL文は通常の設計で出てくるSQL文と根本的に違う面があります。その違いを大道君に聞いてみると、

「そうですね……、同じ名前が何度も出てくるので、混乱しやすい気がします」

「そこ、試験に出るよ！(笑)」

## 同じ名前が違う意味で使われている!

別に試験はありませんがまさにそのとおりで、**リスト1**のSELECT文を見ると、type_namesテーブルのid、type_name、type_valueの3種のカラムが2回出てきてそれぞれ違う意味を表しています。通常の手続き型言語のプログラミングにたとえれば、同じ変数名が隣の行で違う意味で使われているようなもので、混乱しないはずがありません。

「ですよね……幸い、と言っちゃなんですけど早めに気がついたので、これを使わないように修正しようと思うんですが、その段取りをどうしようかと」

「すでにこれで書かれたコードがあるわけだね？」

「そうなんです。調べてみたんですが、**図2**に書いたように、EAV方式って本来は分離しておくべきA～Dのテーブルが全部入っているわけで、中身は実質的に多数のテーブルじゃないですか。だからそこに関係するコードは参照系も更新系も非常に多いですよね」

「そうやね」

「となるとそれを一気に修正するのは現実的には難しそうで……」

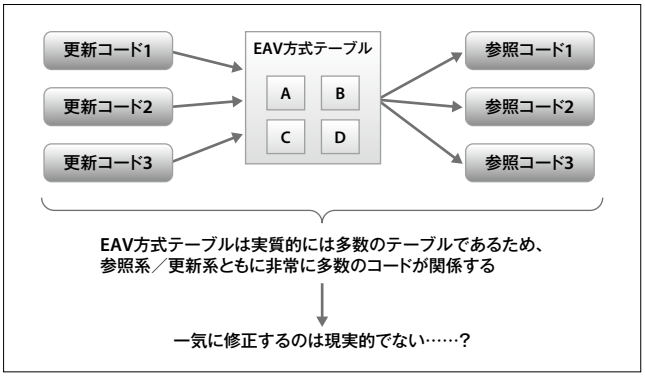
「まあ、そうやね」

「かと言ってこれをそのまま放置すると、メンテナンスしにくいコードが今後も大量に残るわけですから、なんとかしたいわけです」

「うん、それで？」

これまでは私が方法を指示することが多かったのですが、すでに何か策を考えていそうだったので、先をうながしてみました。

**▼図2　EAV方式のテーブルには大量のコードが関係する**




## EAVの名称マスタを少しずつ移行する方法

「これ、EAVを名称マスタ用に使っているので修正方法自体はわりと単純で、type_nameの種類別に個別の名称マスタテーブルに切り出して、それを参照／更新するようにSQLを変えればいいんです。けれど、何しろ数が多いので、少しずつやれないかと思いまして……こんな方法は可能でしょうか？」

と言って大道君が見せてくれたのが**図3**です。

「まず、①EAVから、たとえばDの部分のデータをコピーして、同じデータを持つ新テーブルを生成します。その後、②参照系のコードを順次新テーブルを使うように移行させていきます。すべての移

行が完了するまでは、EAV内のDも残しておきます。両方のDには同じデータが入っているので整合性は保たれます。完了したらEAV内のDを削除。これを繰り返してすべてのEAV内データの移行を終えたら、EAVテーブル全体をDROPする……で、いけるでしょうか？」

「おお、そうかそうか。なるほどね……うん、いけそうやね。でも、更新系のコードはどうするの？」

「やっぱりそこが問題ですよね……。名称マスタ用のデータなんかだと、そもそも更新系のコードがない場合もあるんですけど、全部がそうじゃないです。ただ、更新系は少ないので、③最後に一気に移行させようかと……」

「これがうまくいく前提は、二重に存在するDのデータの同値性が保たれること、やね？」

「そうですね」

「『①新テーブル生成』のあと、『③更新系のコードは一気に移行』の間に、EAV内のDに更新がかかる可能性はある？」

「……あります」

「そこで2つのDの整合性を保つしくみはある？」

「そこなんですけど、何か良い手がないでしょうか」

「まあ、古いテーブルにトリガーかけてやればええよ」

「え？　トリガー使ってもいいんですか？」

「こういうときにこそ使うもんよ」

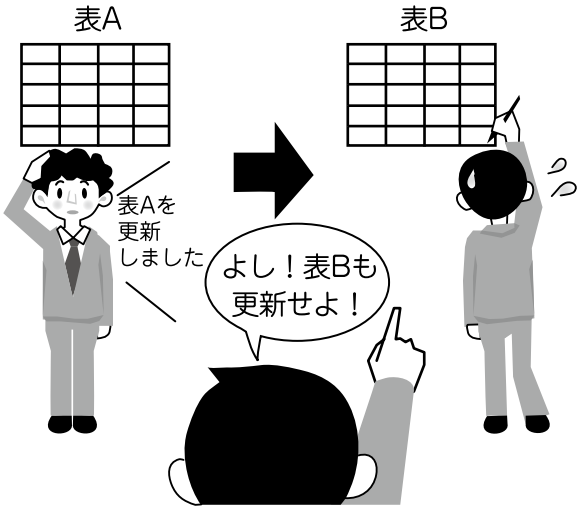
2つのDに同じデータが入っているようにするためには、古いデータ(EAV内のD)が更新されたときに、その内容で新しいテーブルを更新するようにトリガーを組んでやります。**リスト2**がその例です。

## トリガーは便利、けれどご利用は慎重に

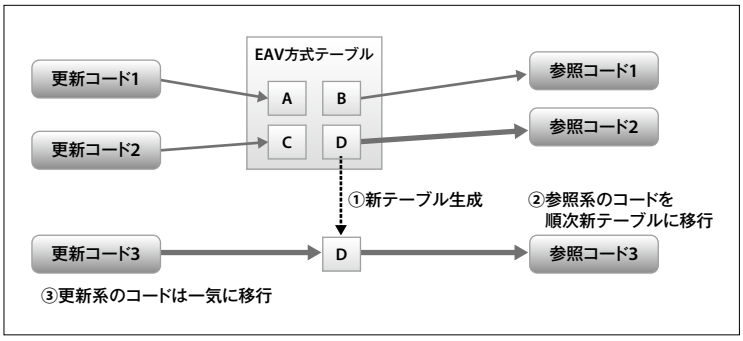
本来ならばDを更新するコードをすべて洗い出してそのすべてを修正する必要があるのですが、トリガーを使えば1ヵ所で済みます。そんな便利なトリガーですが、便利さと危険さは裏表で、トリガーを多用するとアプリケーションから存在が見えない処理が多数走ることになり、つまりプログラマがその存在を忘れてしまう危険を伴います。そこで、弊社では次のような基準に合致するときにトリガーを採用しています。

- 動作を周知徹底できる内容であること
- Excelなどの定義ファイルから自動生成できること

**▼トリガーを使うと、表Aの変更をきっかけにして自動的に表Bも変更できる**




**▼図3　一時的に二重管理を許せば段階的な移行が可能**



今回のトリガーは恒久的なものではなくEAVをなくすために一時的に使用するものであり、かつ、**リスト2**を見てもらえばわかるようにワンパターンのコードが続くため、Excelマクロでカラム名一覧からトリガーを自動生成できる程度の単純な内容です。この程度の単純なものなら「動作を周知徹底する」のも容易ですので、使っても問題ないでしょう。

「まあ何にしてもそれでいけるやろ！　やってみ！」

「やってみます！」

## 非正規形のテーブルを正規化したい

この例のように「下手な設計で作ってしまったけれど今さら作りなおせない」というシステム、身近に心当たりのある方も多いことでしょう。基本設計がまずいままで大量のコードを書いてしまうと、それを一気に修正するのはなかなか現実的ではありません。ですが、やりようによっては少しずつ修正できる場合があります。

前述の例はEAVでしたが、もう1つ、「非正規化設計のテーブルを正規化する」という例を紹介しましょう。

**図4**の上部は非正規化された受注テーブルの例です。テーブルの中に受注ヘッダと受注明細が混在しているため、受注ヘッダに該当する部分に同じ情報が重複して表れています。これは非正規化設計でよくある例ですが、本来は分離しなければなりません。なお、実際には**図4**に示した以外にも多数のカラムが必要ですが、単純化してあります。

「さて、ここで受注テーブルを参照しているコードが多くて、やっぱり一気に修正するのが無理だとする。EAVのときと同じ手が使えるかな？」

「これは……statusみたいに、トランザクションをかけて更新する情報が入っているから、コ

**▼リスト2　新テーブルの生成（左）と更新トリガー（右）**

SELECT文の結果を新規テーブルにコピー

```
CREATE TABLE delivery_comps
(PRIMARY KEY (id))
AS
  SELECT
    id
    , type_value AS delivery_comp
  FROM type_names
  WHERE type_name = 'delivery_id'
;

CREATE TABLE delivery_timeslots
(PRIMARY KEY (id))
AS
  SELECT
    id
    , type_value AS delivery_timeslot
  FROM type_names
  WHERE type_name = 'delivery_time_id'
;
```

旧テーブルへの更新を自動的に新テーブルに反映させるためのトリガー

```
DELIMITER $$

CREATE TRIGGER tr_type_names AFTER INSERT UPDATE DELETE
  ON type_names FOR EACH ROW
BEGIN

  -- delivery_compsテーブルと同期
  IF OLD.id IS NOT NULL AND type_name = 'delivery_id' THEN
    DELETE FROM delivery_comps WHERE id = OLD.id;
  END IF;
  IF NEW.id IS NOT NULL AND type_name = 'delivery_id' THEN
    INSERT INTO delivery_comps (id, delivery_comp)
    VALUES(NEW.id, NEW.type_value);
  END IF;

  -- delivery_timeslotsテーブルと同期
  IF OLD.id IS NOT NULL AND type_name = 'delivery_time_id' THEN
     DELETE FROM delivery_timeslots WHERE id = OLD.id;
  END IF;
  IF NEW.id IS NOT NULL AND type_name = 'delivery_time_id' THEN
    INSERT INTO delivery_timeslots (id, delivery_time)
    VALUES(NEW.id, NEW.type_value);
  END IF;

END;
$$

DELIMITER ;
```

ピーを作るわけにはいかないですよね？」

「トリガーは発火元のトランザクションの中で動くけど、まあ名称マスタと違ってこの種のデータの実体を二重に持つのはやめたほうがいいね。マスタと違ってデータ量も多いし。そこで役に立つのがviewなんよ」

## 正規化してからviewを使って非正規形を再現

手順としては、まず元テーブルを更新しているプログラムを洗い出したうえで、それをすべて、正規化した新テーブルを更新するように修正します。

次に、元テーブルから新テーブルへデータを移行します(**図4**の❶)。

元テーブルをDROP(またはバックアップのためリネーム)し、代わりに非正規形の元テーブルを再現するviewを作ります(**図4**の❷)。当面このviewで元テーブルとの互換性が保たれるため、参照するコードは無修正で動きます。

その後順次、参照系のコードも正規形テーブルを参照するように変えていきます(**図4**の❸)

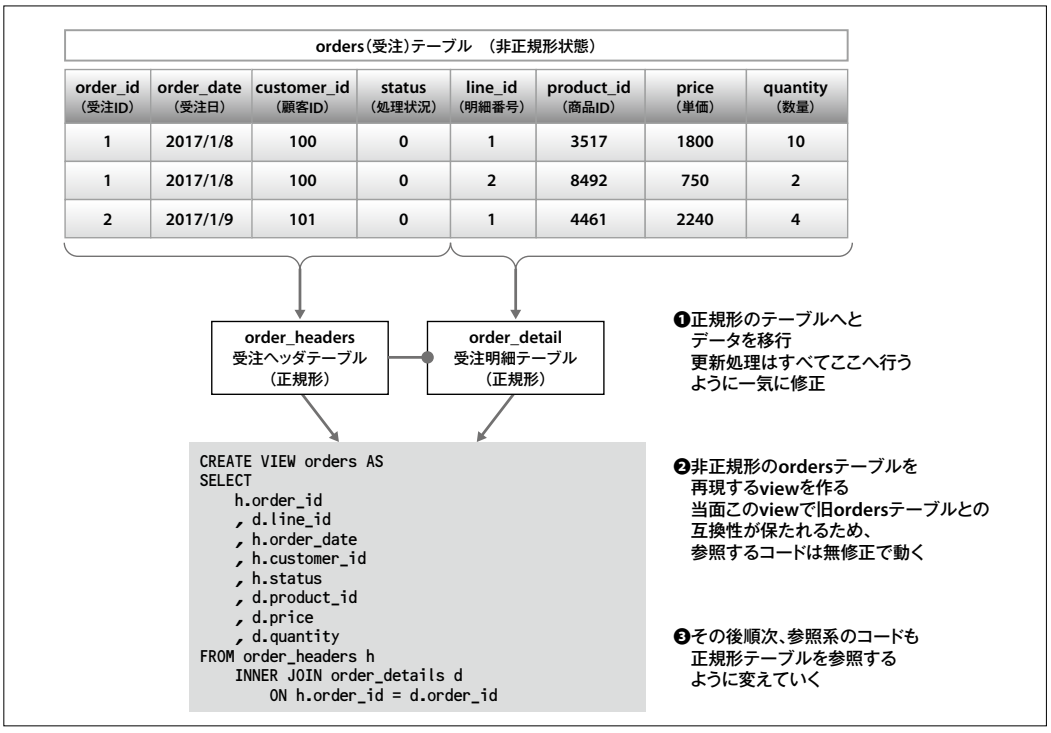
すべての修正を終えたら、orders viewをDROPします。

「あ、なるほど……参照系のコードにとっては『要するに読めればいい』ので、実体を残す必要ないんですね。同じ名前で同じ値が返ってきさえすれば……」

「そういうこと。こんな手を使うこともできるから、やってみるといいよ」

正規形はRDBの基本ではありますが、深く考えずに、あるいは「正規化すると性能が落ちるから……」のような間違った考えをもとに非正規形で作ってしまったシステムも少なくありません。基本設計をひっくり返すのは一筋縄ではいきませんが、ずっと使い続けるシステムで下手な設計を残しておくと、後々禍根(かこん)を残します。こんな方法を参考に、できるだけ解消するように工夫してみてください。SD

**▼図4　非正規形テーブルを正規形に修正**



# 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13回 スマートフォンと人工知能がつながる未来

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

古川 新（ふるかわ あらた）
東海大学 理学部1年／
日本Androidの会
学生部 部長

ディープラーニングの実用化により、近年大きな注目を集める人工知能。私たちに最も身近なスマートフォンと人工知能がどう結びついていくのか、技術的な観点から考えてみるために、TensorFlowをAndroidで動作させるアプリを実際に開発しました。その紹介を通して、今後の人工知能の活用を展望してみましょう。

### 人工知能を使ったアプリケーションの登場

音声アシスタントシステムは、プロダクトとして人工知能がスマートフォンで活用されている事例の1つです。代表的なものはiPhoneに組み込まれたSiriや、Androidの「OK Google」機能です。SiriやOK Googleの高度な音声認識の実現には、人工知能が用いられていることが明かされています。

この「人工知能」とは機械学習によって作られた特化型人工知能（Colum 1参照）のことであり、ディープラーニングと呼ばれる機械学習の発達により、ここ数年で急速に精度が向上してきました。この人工知能の活用は、農業における環境管理や品質評価、自動車の自動運転技術、医療分野における病気の早期発見などに応用されており、これらの研究が活発に行われています。

本稿ではこのような、最も人々に身近なデバイスであるスマートフォンと人工知能の関係にフォーカスを当てます。とくに、この音声アシスタントは音声解析により直感的かつインタラクティブに情報提供を行う機能を持ちます。また、ユーザ個人に特化した情報収集や整理などを支援する機能を持っているため、パーソナルアシスタントシステムとも呼ばれます。

ほかにも、写真に写っている顔を判別して人物ごとのアルバムを自動で作成する機能が、iOS 10の写真アプリやGoogle Photosに導入されています。Facebookには、ユーザのアップロードした画像に写っている人物を自動で判断してタグ付けする機能があります。これらのアプリケーションはどれも機械学習を用いることで、既存のアプリには存在しないユーザエクスペリエンスを実現しています。このように、人工知能を応用したスマートフォン向けアプリ

COLUMN 1

#### 特化型人工知能

ある特定の課題について、賢く振る舞おうとするプログラムのことを特化型人工知能と言います。将棋やチェスの最適な手を見つけ出すプログラムや、写真に写っている人物を判別するプログラムなどがあります。近年実用化が進んでいる人工知能は、この特化型人工知能のことを指す場合がほとんどです。本稿において、「人工知能」とは機械学習によって作られた特化型人工知能のことを指すこととします。

ケーションが市場に現れてきています[注1]。

## スマートフォンから人工知能を利用する方法

機械学習は特化型人工知能を作る手法の1つです。生物の神経細胞(ニューロン)が構築するニューラルネットワークを計算機上でシミュレートしたものを、人工ニューラルネットワークと呼びます。この人工ニューラルネットワークこそが機械学習で作られる人工知能の正体です。人工ニューラルネットワークを構成するニューロンは数式で模され、作られる人工知能は巨大な数学モデルになります。この人工知能が理想的な結果を出力するようになるためには、生物と同様に反復的な訓練が必要です。機械学習とは、膨大な学習用データを用いて体系的にニューラルネットワークの内部表現の最適化を繰り返していくことです。人工ニューラルネットワークはとにかく巨大な数値演算の集合なので、入力を与えて演算を行うことで計算結果を得ることができます。このような膨大なデータや高速な計算性能が必要な人工知能の恩恵をスマートフォンが受けるにはどのような方法があるのでしょうか。

## サーバで実行する

スマートフォンから人工知能を利用する方法の1つは、人工知能を稼働させたサーバを用意し、特定の問題を解決する機能をWeb APIとしてスマートフォンに提供する方法でしょう。これはGoogle Cloud Vision APIのようにサービスとして提供されている事例が存在します。

サーバを用意する場合、運用コストとプライバシーに関する問題があります。人工ニューラルネットワークの機械学習により作られた人工知能を「モデル」と呼びます。モデルは巨大な数値演算の集合です。それを計算し出力を得るには大きな演算コストを伴います。一度の計算であればあっという間でも、多くのリクエストをさばかねばならないサーバでは大きな負荷となります。安定したサービスを提供する場合は、それなりの設備投資が必要になります。たとえサーバが高速に処理できたとしても、ユーザの環境によっては必要な通信帯域を確保できない

注1) "Google voice search: faster and more accurate" https://research.googleblog.com/2015/09/google-voice-search-faster-and-more.html

COLUMN 2

### GoogleとAppleのプライバシー問題

Googleはこのプライバシーの問題について、高度な暗号化によりデータを守ることで、プライバシーよりも利便性を追求する姿勢がうかがえます[注A]。2016年に発表されたGoogle Nowは、ユーザーデータがネットワークを通過することを否定していません。Google Nowは情報の送信を拒否する選択肢も提供していますが、その場合は機械学習による便利なパーソナルアシスタントシステムを十分に利用することができなくなります。Googleが情報を収集する背景には、ユーザから情報を収集して機械学習に用いたほうが、より優秀な人工知能を作ることが可能になるという事実があります。

一方で、Appleはプライバシーの問題に対して非常に慎重です。iPhoneで用いられている人工知能におけるプライバシーに対する取り組みについて、個人情報やプライバシーにかかわるデータはサーバに送信しないとインタビューにて明言しています[注B]。これでは個人に特化した人工知能を用意することが困難になると思われますが、Appleではこの課題を、スマートフォン上で実行する方法をサーバ上での実行と組み合わせることで、ある程度解決しているようです。

人工知能に用いるデータが必ずしも個人情報を含んでいるとは限りませんし、例に挙げた2社のどちらが正しいのかを判断するのは難しいことです。しかし、サーバを使った人工知能の利用にはプライバシーの問題が付随することは意識しなくてはいけません。

注A) "Google's AI Plans Are A Privacy Nightmare" http://gizmodo.com/googles-ai-plans-are-a-privacy-nightmare-1787413031

注B) "An exclusive inside look at how artificial intelligence and machine learning work at Apple" https://backchannel.com/an-exclusive-look-at-how-ai-and-machine-learning-work-at-apple-8dbfb131932b

こともあるでしょう。ゲームの人工知能利用などのように、インタラクティブでリアルタイム性が要求され、リクエスト頻度が高くなることが予期されるようなアプリケーションには向いていません。

とくに重要なのはプライバシーの問題です。この方法ではユーザーデータがネットワークを通過してしまうため、プライバシーの確保に疑念が生じてしまいます。前述のとおり、スマートフォンの強みはユーザとの密接な関係であり、それを活かしたパーソナルアシスタントシステムなどが扱う情報は個人情報と直結します。人の顔の写った写真や、自分の声など、あらゆるデータが勝手にネットワークを通っていくのです。学習のデータとして利用するためにユーザから収集したデータが保存されている可能性も否定できません。たとえ誰にも個人情報が漏れないとしても、自分のデータがインターネットに流れることそのものを嫌う人もいるでしょう。そのため、サーバで実行するには課題があり、Google、Appleともに独自の方針を設けています(Column 2参照)。

## スマートフォンで実行する

ここまでは、サーバでの実行について客観的に事実を述べてきました。実はもう1つ人工知能を利用する方法があります。それは、スマートフォン上で人工知能を実行する方法です。

この方法では、ユーザからのリクエストをリアルタイムに処理するサーバを運用するコストが一切かかりません。人工知能の実行プロセスからネットワーク通信という不安定な要素が排除されるため、サーバ実行よりもインタラクティブなコンテキストに利用しやすく、モデルデータさえあればオフラインでも実行することが可能です。

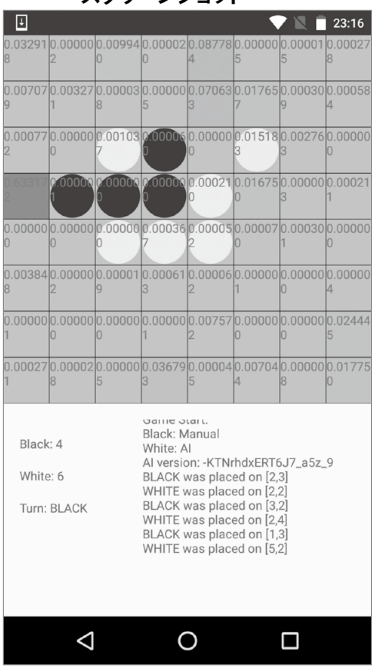
サーバにはない重要なメリットもあります。この方法では、ユーザのデータがネットワークを通過しません。AppleのSiriを始めとする人工知能技術では、この方法をサーバと組み合わせることで、品質を維持すると共にプライバシーを守っているようです。スマートフォン上で計算が完結するため、人工知能に起因したプライバシーの侵害が起きる可能性は根本的にありません。

スマートフォン自身に計算させるため、どうしてもサーバで実行する場合に比べてスマートフォンへの負荷が大きくなります。莫大な回数の反復訓練を行うのは現実的ではありません。

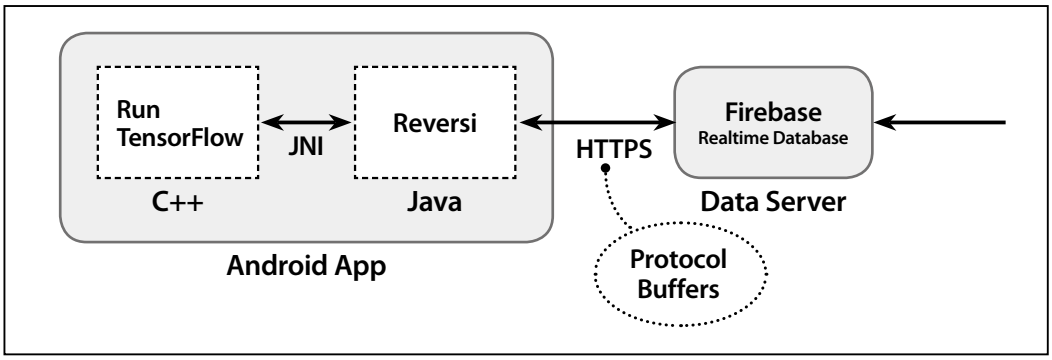
**▼表1　サーバとスマートフォンでの違い**

| | サーバで実行 | スマートフォンで実行 |
|---|---|---|
| 運用コスト | 高 | 低 |
| プライバシーの保護 | 未解決 | 問題なし |
| スマートフォンへの負荷 | 低 | 高 |
| 機械学習による訓練 | 可能 | 困難 |

**▼図1　作成したアプリケーションのスクリーンショット**



▼図2　アプリケーションの構成イメージ




今のスマートフォンの性能で可能なのは、すでに完成しているモデルを使って結果を計算することでしょう。アプリケーションが人工知能に要求するのは結果を得ることであって、人工知能を作る機能ではないからです。サーバのように大量のリクエストをさばく必要はないため、非力なスマートフォンでも結果を計算することに問題はないと考えられます。この点については、一例ですが実際にアプリを開発して検証してみましたので後述します。

両者は一長一短でどちらかが優れているわけではありません(**表1**)。逆に言えば、どちらにも必要とされる局面が存在します。本当にスマートフォン上で実行可能であるならば、プライバシーの保護やコスト削減といった大きなメリットを持つこの手法はスマートフォン開発に広く用いられていくでしょう。

## 人工知能をスマートフォン上で実行するアプリを開発してみた

機械学習で作られた人工知能は本当にスマートフォンで実行できるのか、そして実用に足るものなのかを検証するために、次の条件でAndroidアプリを実際に開発してみました(**図1**)。

・モデルを使った計算はスマートフォンで実行する
・モデルを作る機械学習はサーバで実行する
・スマートフォンからサーバへ情報を提供する(ユーザの任意)

作成したのはリバーシの人工知能と対戦ができるAndroidアプリです。盤面上のマスの数字は、マスごとの「どのくらい最適な手か」を人工知能が数値化したものです。わかりやすいように数値の高いマスほど赤くなるようにしています。**図1**では、4行1列目が人工知能である白にとって最適な手であると判断されています。

機械学習による訓練と実行には、TensorFlowを使用しました。TensorFlowはGoogleによって開発されている汎用数値演算ライブラリです。大規模分散並列処理に特化しており、マルチデバイスに処理を分散し演算をスケールアウトすることができます。機械学習において問題となる莫大な演算負荷もスケールアウトができるため、Google翻訳エンジンなどさまざまなプロダクトに利用されています。GitHub上でオープンソースとして公開されているため、誰でも自由に利用することができます。TensorFlowはC++で作られたライブラリです。Androidアプリには共有ライブラリとして組み込み、JNIを用いてJavaから実行します(**図2**)。TensorFlowをAndroidで実行するための共有ライブラリのサイズは約16MBです。

このアプリではプレイデータをFirebaseのリアルタイムデータベースに送信します。プレイデータの送信はユーザの判断で拒否できるようになっています。Firebaseはスマートフォンアプリケーションでよく利用される汎用的な機能を、クラウドから提供するGoogleのサービスです。データベースに新たなデータが追加されると、訓練用のサーバは自動で機械学習による訓練を行います。訓練で更新された人工知能モデルは定期的にProtocol Buffers形式としてシリアライズされ、データベースサーバにアップロードされます。Androidアプリは最新のモデルデータをローカルストレージにダウンロードし、そのモデルを使用して打つ手をスマートフォン上で計算します。遊べば遊ぶほど訓練され、自動的にユーザへフィードバックされるという一連の品質向上のサイクルが出来上がっています。

より具体的な実装方法については、筆者のブログ[注2]にまとめておりますので、そちらを参照してください。

## 実際に開発することで確認できたこと

設けた条件のとおり、モデルの実行はスマートフォン上で行われるため、ユーザから直接リクエストを受けて結果を返すAPI用サーバは必要ありません。一度モデルデータをインストールすればオフラインでも実行可能です。訓練用サーバはありますが、訓練はユーザのためにリアルタイムに処理を行う必要はなく、常時稼働しているわけではありません。どちらの場合でも訓練コストは必要ですし、API用サーバに比べれば大したコストではありません。また、データベースサーバについても同様です。スマートフォン上で実行することで大幅にサーバ運用コストを削減することができたと言えます。

懸念されたスマートフォンへの負荷も問題となっていません。このアプリを筆者のNexus 5で実行する限りでは、人工知能が結果を算出するのにかかる時間は100ミリ秒未満です。人工知能のモデルサイズもバイナリタイプのProtocol Buffers形式でたったの約17KBです[注3]。もちろん、負荷や容量はモデルの規模や実行頻度、実行環境によって変化します。しかし、まったく使い物にならないほどマシンリソースを酷使するわけではないと考えてよいでしょう。

## スマートフォンは人工知能の顔となるか？

スマートフォンの最たる特徴は人との距離にあります。今やスマートフォンを誰もが持っていて当たり前の時代になりました。ゆえに、「スマートフォンだからこそできること」があります。冒頭で例に挙げたパーソナルアシスタントシステムを始めとするアプリケーションは、機械学習によって作られる人工知能の持つ高精度な解析能力と、スマートフォンが人に最も近いデバイスであるという特徴が有効に活かされています。技術と社会の最も大きな接点であるスマートフォンは、人工知能を利用するデバイスとして活用されていくポテンシャルを十分に秘めています。

実際にAndroidアプリケーションで人工知能を利用してみたことで、スマートフォン上でも人工知能の実行は可能であることが検証されました。スマートフォン自身が人工知能を動作させることも当たり前になっていくのだと思います。SD

注2）http://ornew.net

注3）ちなみに、MNISTによる手書き数字の解析で約99.7%の精度を出せるGoogleのTensorFlowによる畳み込みニューラルネットワークのモデルとほぼ同じサイズです。

COLUMN 3

## コミュニティ活動

文：日本Androidの会　嶋 是一

### ■ABC2016 Autumn開催報告

日本のAndroid開発者のユーザーコミュニティである日本Androidの会が、年に2度開催するAndroidの祭典「Android Bazaar and Conference（ABC）」が2016年11月19日に「ABC2016 Autumn」として開催されました。千葉県柏市にある、柏の葉キャンパスというイノベーション色が豊かな地に400名の方々が集いました（**写真A**）。

「次世代の孵化装置Θ Android ～ミライをつくろう、Androidで作ろう！～」がテーマです。モバイルに関する技術は、スマートフォン8割のシェアを持つAndroidの「上も使って」花開いています。Androidは次世代に続く技術、文化、人材の卵を温める孵化装置の役割を担っています。今どのような卵を温めているのか、今ある孵化の鼓動を集め、そこから見えるミライを共感するために開催しました。「講演（カンファレンス）」と「展示（バザール）」の2つに分かれ、このテーマにちなんだ講演と展示が行われました。とくに人気が高かったのが、レノボから発売されている空間をセンシングできるtangoスマホの実機展示や、本文執筆の古川さんも登壇したディープラーニング、そして今回後援をいただいた総務省から、情報通信国際戦略局長の谷脇康彦様によるデータ主導社会の実現に向けての講演でした。また今回は、Gショックを彷彿させるカシオ計算機のAndroid Wear「WSD-10F」を用いたアプリコンテストを開催しました。日本Androidの会の地方支部からも参戦し、IoTと連携させるもの、お笑いのネタに走るものなど多彩なアプリが発表され、浜松支部の「今何時？」アプリが優勝をしました。

コミュニティの愉しみは、技術やテーマを同じにした参加者や講演者との情報交換ができること、そして自分のアイデアを生かすチャンスがあることです（**写真B**）。次回のABCも計画していますので、スタッフとしてコミュニティ活動に参加されてみてはいかがですか？

・**ABC2016**●http://abc.android-group.jp/2016a/

### ■MaruLabo紹介

学生にディープラーニングの学習・開発環境を提供することを目的としたコミュニティです。2016年12月20日（火）には、定員50名のところ700名を超える申し込みがあった「ディープラーニング・ハンズオン」をGoogleで開催。MaruLaboは基本的に学生自身によって運営されており、学生の参加を募っています。実際的なディープラーニングの開発・学習に必須の高速なGPU環境の提供を1つの特徴としており（計算資源は、企業・個人からの寄贈によるもの）、今後ハンズオンの開催が予定されています。

https://goo.gl/qU21hK

**▼写真A　基調講演会場**

**▼写真B　運営委員集合写真**

### ■Android関連イベント

**○Droidkaigi 2017**

主催：DroidKaigi 2017 実行委員会
日時：2017年3月9日（木）／10日（金）
場所：ベルサール新宿グランドコンファレンスセンター
有料イベント。エンジニアが主役のAndroidカンファレンス。
https://droidkaigi.github.io/2017/

**○技術書典**

主催：TechBooster／達人出版会
日時：2017年4月9日（日）
場所：アキバ・スクエア
一般参加 無料。サークルなどで制作した技術書オンリーの販売会。
https://techbookfest.org/

一歩進んだ使い方のためのイロハ

# Vimの細道

第15回

mattn
twitter:@mattn_jp

## QuickRunで開発を加速する（前編）

書いたプログラムをその都度実行できる便利なプラグインQuickRunを、2回に渡って取り上げます。今回はまず、基本的な使い方とオプションの一覧、そしてQuickRunのカスタマイズにつながる重要なしくみ、runnerとoutputterについて解説します。

### 筆者愛用プラグイン QuickRun

筆者はVimプラグインを見つけては少しだけ試すといったことをよくやっており、日常的に使うプラグインというのはあまり持たないほうなのですが、そんな筆者であってもこれは常にインストールしているというプラグインがいくつかあります。

その1つが今回紹介するQuickRun[注1]です。

QuickRunはthincaさんが開発しているVimプラグインで、現在開いているファイルタイプに見合ったコマンドを実行してくれるコマンドです。今回はQuickRunの基本的な操作方法を、そして次回で便利なカスタマイズ方法を紹介します。

### 基本的な操作

たとえば、Rubyのファイルを開いた状態で:QuickRunを実行すると、そのRubyスクリプトが実行されます。デフォルトで、<leader>-rがキーマッピングされています[注2]。QuickRunのしくみは一見簡単そうに見えますが、デフォルトで数多くのファイルタイプが実行可能となるように設定されており、簡単かつ便利に使えます。

たとえばゼロからソースコードを書いていく場合、トライ＆エラーでだんだんと形を作っていくわけですが、その際に都度、動作を確かめながらコードを書けるため非常に役立ちます。もちろん1枚岩（ソースファイルが1つだけ）であれば、完成しているものを実行するのにも使えます。

:QuickRunはバッファの一部でも実行可能で、ソースコードの一部をビジュアル選択して次のように実行できます。

```
:'<,'>QuickRun
```

※<leader>-rでも実行可能

また、たとえばテキストの中に例文として書かれたコードについては、ビジュアル選択して次のように実行することもできます。

```
:'<,'>QuickRun python
```

このとき、バッファにはスクリプト言語ではないテキストファイルが開かれているので、末尾の「python」のように、実行すべきものが何であるかをQuickRunに引数で教える必要があります。実行できる言語の候補は、:QuickRunの入力のあとにTABで入力補完できます。

注1）URL https://github.com/thinca/vim-quickrun
注2）何も設定していない状態であれば<leader>-rは\rで動作します。

▼表1　QuickRunの対応言語・環境

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| awk | bash | C | Clojure | Coffee | Cpp | Crystal | C# | D |
| Dart | Dosbatch | Elixir | Erlang | eRuby | fish | FORTRAN | F# | Go |
| Groovy | Haskell | io | Java | JavaScript | JSX | Kotlin | Lisp | LLVM |
| Lua | Markdown | Nim | Ocaml | Perl | Perl 6 | PHP | Powershell | Python |
| R | Ruby | Rust | Scala | Scheme | sed | sh | SQL | Swift |
| tmux | Typescript | Vim script | WSH | XQuery | Zsh | | | |

## 実行できるファイルの種類

QuickRunが実行できるのは、PythonやRubyといったスクリプト言語に限りません。たとえばC言語であれば、編集中のソースコードをコンパイルして実行してくれます。Javaも実行できますし、HTMLファイルであればブラウザが起動します。MarkdownであればHTMLに変換され、ブラウザが起動します。

執筆時点(2016年12月)でQuickRunには、デフォルトで51種類のスクリプト言語や変換可能なフォーマットが登録されています(**表1**)。WindowsでもLinux/UNIXでも、ほぼ相違なく動作します。

## オプションの指定

QuickRunには実行時オプションが用意されており、実行するコマンドの挙動を変えられます。指定は次の書式で行います。

```
:QuickRun -[オプション] [値] ↵
 -[オプション] [値] ...
```

**表2**に扱うことができるオプションを示します。これらは設定にてファイル種別ごとに指定することもできます。これらをカスタマイズすることで、自分独自のコマンドを作り上げることができます。

## runnerとoutputter

QuickRunが起動する際、大きく分けて2つの機能「runner」「outputter」が働きます。runnerはコマンドを実行するためのしくみ、outputterはその結果を出力するためのしくみです。

### runnerとは

Vimが外部プロセスを起動する方法にはいくつかあり、それぞれに特性が異なります。QuickRunではこれらをユーザ側が指定できるようになっています(**表3**)。ただし、runnerのすべてがあらゆるコマンドに対して有効であるというわけではないため、問題を自己解決できる方のみ指定するようにしてください。

表中で、非同期サポートが△となっている

▼表2　QuickRunの実行時オプション

| オプション | 概要 |
|---|---|
| type | 種別を指定。たとえばテキストファイル内に書かれた別の言語を実行する際に有効 |
| exec | 実行するコマンドの書式を指定。既存の設定を上書きするために使用 |
| command | exec内での%cを指定。たとえば専用のインタプリタを使用する場合に使用 |
| cmdopt | exec内での%oを指定。オプションを付加する場合に使用 |
| srcfile | ソースファイルを指定。srcとは併用不可 |
| src | ソース文字列を指定 |
| args | exec内での%aを指定。プログラム引数を指定する |
| input | コマンドの標準入力に書き込むファイルを指定。=xxxで指定した場合、文字列xxxが書き込まれる |
| outputter | outputterを指定 |
| runner | runnerを指定 |
| mode | ノーマルモードnもしくはビジュアルモードvを指定。通常は自動設定される |
| tempfile | 一時ファイル名を指定。形式：%{tempname()} |

runnerの理由を説明します。これらは非同期をサポートはしているものの、実現方法に懸念事項があるためです。Vim scriptはVim8まで非同期処理をサポートしていませんでした。そこでvimprocやconcurrent_processでは、updatetimeというVimのオプションの値を小さく設定することで、一定間隔にプロセスの状態や出力を監視し、更新があれば所定の処理を実行するという方法で非同期を実現していました。ただしこのupdatetimeは、もともと無操作時にswapfileを更新するための待機時間を設定するオプションであり、この値を小さくすると想定外の問題が起きる可能性がありました。またupdatetimeはグローバルオプションであるため、ほかのプラグインが期待しない動作になったり、画面がチラつくという問題が起きました。Vim8のjobの登場により、これらの問題は解消されました。

## outputterとは

QuickRunでコマンドを実行した際の出力結果をどうするかを指定できます(**表4**)。各outputterでは出力に関するパラメータが設定できます。詳しくは:help quickrun-module-outputterを参照してください。

outputterは、複数のoutputterを組み合わせて実行できます。たとえばコマンドの実行結果が正常の場合にはbufferに出力し、失敗した場合には変数outに結果を設定するといった場合は、次のように実行します。

```
:QuickRun -outputter error:success=buffer:error=variable/out
```

**▼表3　各runnerとその特性**

| runner | 非同期サポート | 特性 |
|---|---|---|
| system | × | コマンドの実行にsystem()関数を使用 |
| shell | × | コマンドの実行に:!を使用 |
| concurrent_process | △ | REPL※をサポートするコマンドの場合、プロセスを常駐させることで次回以降のコマンド実行を高速化 |
| python | ○ | Python拡張にてsubprocessモジュールを使用。Python拡張が有効になっていないと使用不可 |
| remote | ○ | remote通信機能を使用する。ただしLinux/UNIXの場合、X Window Systemが起動していない環境では使用不可 |
| vimproc | △ | Shougo氏が開発しているvimprocを使用 |
| vimscript | × | ファイルの評価に:コマンドを使用。基本的にVim script専用だが、:pyfileなどを使用することでPythonも実行可能 |
| job | ◎ | Vim8のjobを使用。ネイティブに非同期をサポートするため、コマンドからの出力をリアルタイムに表示 |

※REPLとはRead Eval Print Loopの略で、入力を評価して表示する対話形式のプログラムのこと。

**▼表4　各outputterとその特性**

| outputter | 特性 |
|---|---|
| browser | 出力をブラウザに表示する。openbrowser.vimが必要。プレーンな出力がそのままブラウザで出力されるため、コマンドの出力がHTMLでなければならない |
| buffer | 出力をQuickRun専用のバッファに出力(デフォルトの動作) |
| buffered | 出力全体を読み込んだあとbuffered/targetで指定される出力先に流し込む。例)buffered/buffer |
| error | コマンド実行結果が成功かエラーかで出力先を変更。例)error/success=buffer:error=variable/out |
| file | 出力をファイルに書き出す。例)file/name=out.log |
| loclist | 出力結果をロケーションリストに設定 |
| message | 出力をVimのメッセージに書き出す |
| null | どこにも出力しない |
| quickfix | 出力をクイックフィックスに設定 |
| variable | 出力を変数に設定。例)variable:a |
| multi | 複数の出力先を指定する。<br>例)multi/targets=buffer/variable:a |

◆ ◆ ◆

今回はQuickRunの基本的な使い方を説明しました。runnerやoutputterの特性を理解することで、QuickRunの可能性は広がります。次回はQuickRunの設定方法と、拡張方法を紹介します。SD

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33回 Emacsの正規表現（上級編）

Emacsの正規表現の最終回。前半では、正規表現にマッチした行を一覧できるM-x occurをさらに便利にするTipsを紹介します。後半では、複数のクエリによる絞り込み検索ができるhelmとoccurを組み合わせた強力なM-x helm-occurと、結果を編集できるall-extパッケージを紹介します。

### anything.el復活

ども、るびきちです。今まで細々と更新をしていたanything.elでしたが、この際、復活宣言をしました。anything.elとは、筆者が9年間育ててきた候補選択フレームワークで、helmの前身です。helmと袂（たもと）を分かってから、helmは超高速で開発が進められました。

一方でanything.elは、メンテナンス中心にゆったりと育ててきました。その間にanything.elのミッション（開発理念）が固まってきました。「Emacs Lisp学習者に実用的な成功体験、そしてずっと使える安心を提供します」です。あくまでも「ユーザ目線」という点でhelmと明確な差別化を図ります。

とはいえ筆者は、helmを全否定するつもりはありません。helmを使ったすばらしいパッケージは数多く存在し、とくにhelm-swoopには毎度お世話になっています。helmは開発が活発ですので、そのまま使う分には問題ありません。ただ、helmの内部を改造したりhelmプログラミングをしていると、突然の仕様変更の煽（あお）りを受けるリスクがあると言っておきます。

内部を改造したり候補選択インターフェースを使ったプログラミングをするならば、anything.elをお勧めします。与えられたもので満足する人はhelm、自分にぴったりな形へと徐々に進化させたい人はanything.elと明確に線引きします。詳しくは筆者のサイト[注1]を参照してください。

### occurを使いこなす

#### *Occur*バッファを編集する

さて、今回もEmacs正規表現の話題の続きです。バッファ内で正規表現にマッチした行を列挙するには先週お伝えしたとおり、M-x occurを使います（**図1**）。

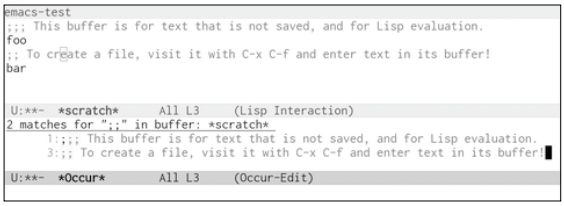
実は、*Occur*バッファを編集することで、元のバッファに編集結果を反映させられます。*Occur*バッファはそのままだとread-onlyですのでe（occur-edit-mode）で編集可能にします。そのあとは、置換などで自由に編集してください。編集結果は、ダイレクトに反映されます（**図2**）。

注1）URL http://emacs.rubikitch.com/anything

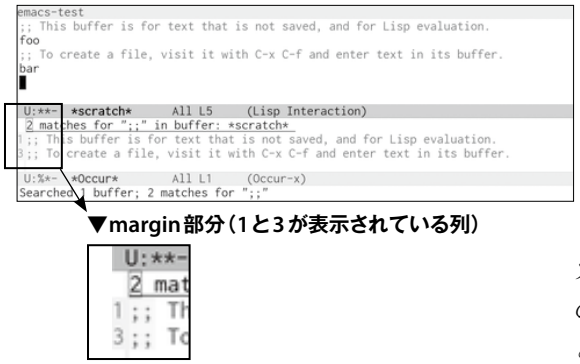
▼**図1** M-x occur RET ;; RET

```
emacs-test
;; This buffer is for text that is not saved, and for Lisp evaluation.
foo
;; To create a file, visit it with C-x C-f and enter text in its buffer.
bar
U:**-  *scratch*       All L5     (Lisp Interaction)
2 matches for ";;" in buffer: *scratch*
       1:;; This buffer is for text that is not saved, and for Lisp evaluation.
       3:;; To create a file, visit it with C-x C-f and enter text in its buffer.
U:%%-  *Occur*         All L1     (Occur)
Searched 1 buffer; 2 matches for ";;"
```

▼図2　*Occur*バッファを編集。すぐに反映される




▼図3　行番号がmarginに表示される




▼margin部分（1と3が表示されている列）




編集が終了し、通常のread-onlyに戻すにはC-c C-c(occur-cease-edit)を使います。

## M-x occurを拡張する occur-xパッケージ

あまり知られていませんが、occurを拡張するoccur-x.elがあります。次の機能を提供します。

・絞り込み検索
・行番号をmargin(左端の空き部分)に表示する

M-x package-install occur-xでインストールし、次の設定を加えます。

```
(add-hook 'occur-mode-hook
          'turn-on-occur-x-mode)
```

絞り込み検索は、正規表現にマッチする行のみを表示したり削除したりする機能です。言ってみれば、前回(2017年1月号)で紹介したように、M-x keep-linesやM-x flush-linesを*Occur*バッファで行うようなものです。

k(occur-x-filter)がkeep-lines相当で、f(occur-x-filter-out)がflush-lines相当です。両方の操作はu(occur-x-undo-filter)で戻せます。複数の正規表現すべてにマッチする行を取り出すには、M-x occurのあと、M-x occur-x-filterを繰り返せばいいです。徐々に絞り込まれていきますので、安心して使えます。もちろん、M-x occur-edit-modeと併用できます。絞り込み検索に加え、*Occur*バッファの編集もこなせるようになれば、編集の幅はかなり広がってきます。

また、行番号をmarginで表示するということは、*Occur*バッファそのものには行番号が含まれないことを意味します(**図3**)。マッチした行そのものだけを、コピー&ペーストできるようになります。*Occur*バッファの内容を貼り付ける際、行番号は邪魔になることが多いです。

## M-x occurで前後の行も表示する

M-x occurでは、grepプログラムのように前後の行も表示できます。たとえば、マッチ行の前後2行を表示したければ、C-u 2 M-x occurと操作します(**図4**)。一方で、普通にM-x occurを実行したあとで前後の行を表示させるには標準では再実行するしかありません。

そこで、occur-context-resize.elをインストールすると便利です。M-x package-install occur-context-resizeでインストールし、次の設定を加えます。

```
(add-hook 'occur-mode-hook
          'occur-context-resize-mode)
```

▼図4　前後2行を表示する

すると*Occur*バッファにて、[+][-]キーで前後の行を1行ずつ対話的に増やせます。「0」で前後の行を表示しなくなります。

とはいえ、気になる行があれば*Occur*バッファから直接ジャンプしたり、M-g M-nなどでたどれます。

*Occur*バッファの表示はほんのちょっとしたことですが、お好みで設定してみてください。

## M-x helm-occur

### パワフルな処理を実現

helmでインクリメンタルに行を絞り込むことで、occurとisearchの良いとこどりができます。つまり、occurのように正規表現にマッチする行を一覧しつつ、isearchのように文字をタイプするごとに表示を変えていくことができます。コンピュータの性能が上がったことで、このようなパワフルな処理ができるようになりました。

それを実現してくれるのがM-x helm-occurです。

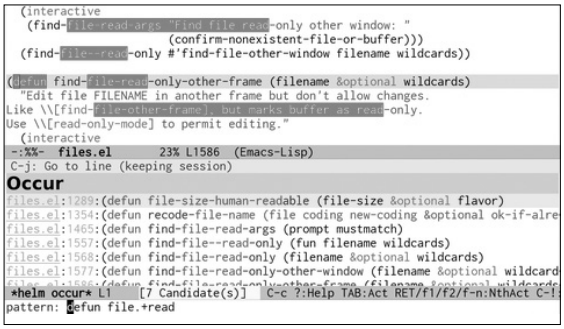
M-x helm-occurは通常のoccurよりもはるかに強力な機能です。なぜなら、次の特徴があるからです。

- 正規表現にマッチする行を一覧できる
- 文字をタイプするごとに表示が変わる
- 正規表現を複数個指定して絞り込み検索ができる
- Migemoが効く（helm-migemo-mode）
- 表示結果を編集できる（all-ext.el）

M-x helm-occurを使いこなせれば、バッファ内の目的とする行をすぐに見つけられます（**図5**）。helmについては2015年3、4月号で取り上げましたが、これから使うにはM-x package-install helmでインストールしてください。この状態でM-x helm-occurが使えます。

M-x helm-occurではいろいろな操作ができます。

**▼図5** M-x helm-occur RET defun file.+read




- Enter：その行へジャンプする
- C-p、C-n、↑、↓：前後行への移動
- C-↑、C-↓：前後行へ移動しながら対象バッファの表示も反映させる
- C-z：対象バッファの該当行を表示する

これらの操作を使ってバッファを縦横無尽に駆け抜けてください。

### Migemoを有効にする

Migemoとは、ローマ字で日本語文字列を検索するすばらしいプログラムです。たとえば、「nihongo」で「日本語」が、「ugoKu」で「動く」が検索にヒットします。

まずはmigemo.elとcmigemoをインストールします。Debian系列のGNU/Linuxならば両者ともパッケージ化されているのでインストールは簡単です。sudo apt-get install cmigemo migemo-elを実行するだけで初期設定まで済み、そのまま使えます。多くのGNU/Linuxにはパッケージシステムがあるため、インストール・設定・管理がとても楽です。

パッケージ化されていない場合は、cmigemoとmigemo.elを別個でインストールし、初期設定も行う必要があります。macOSではbrew install cmigemoで、Windowsでは香り屋氏のサイト[注2]からcmigemoのバイナリを取ってき

注2）URL http://www.kaoriya.net/software/cmigemo

▼リスト1　migemo.elからcmigemoを使う初期設定

```
(when (locate-library "migemo")
  (setq migemo-command "cmigemo")  ; HERE cmigemoバイナリ
  (setq migemo-options '("-q" "--emacs"))
  (setq migemo-dictionary "/usr/share/cmigemo/utf-8/migemo-dict")  ; HERE Migemo辞書の設定
  (setq migemo-user-dictionary nil)
  (setq migemo-regex-dictionary nil)
  (setq migemo-coding-system 'utf-8-unix)
  (load-library "migemo")
  (migemo-init))
(with-eval-after-load "helm-multi-match"
  (helm-migemo-mode 1))
```

ます。migemo.elはMELPAからインストールし、初期設定でHEREと書かれた部分は環境に合わせて書き換えてください。M-x package-install migemoでインストールし、**リスト1**の設定を加えます。

これでisearchでもM-x helm-occurなどでも、Migemoが使えるようになります。

## helm-occurを親切にしたhelm-swoop

helm-swoop.elはM-x helm-occurを親切にしたパッケージです。M-x package-install helm-swoopでインストールします。使い方としてはM-x helm-occurとほぼ同じです。M-x helm-swoopを実行してC-p/C-n/↑/↓を押すと前後行に移動しながら、対象バッファの表示も連動してくれます。ハイライト機能が強化されている分、見やすさはこちらが上です。

筆者のサイト[注3]で詳しく説明しています。

## helmの結果を一覧表示、修正して元のバッファに反映

M-x helm-occurやM-x helm-swoopはM-x occurと違い、Migemoが効くうえに絞り込み検索ができます。

それならば、M-x occur同様にマッチ行一覧を編集できたらいいとは思いませんか？　all.elを拡張する拙作all-ext.elをインストールすれば実現できます。

M-x allはM-x occur-edit-modeと機能的には同じですが、それが登場するはるか昔に書かれたものです。all.el + all-ext.elのメリットは次のとおりです。

- 行番号がoverlayで表示されているため、M-x occur-edit-modeより編集しやすい
- 行番号とその行の内容が書いてあるすべてのhelm情報源に対応
- anything.elにも対応
- C-x hでマッチ行すべてをマークできる
- マッチ行に対してmultiple-cursors.elによる一括編集ができる

all-ext.elもMELPAから、M-x package-install all-extとインストールします。この状態ですでに基本的な設定はできていて、helmからC-c C-aでallに移行できます。*All*バッファはすでに書き込み可能になっていて、編集結果が即反映されるようになっています。

また、マッチ行をC-SPCでマークすることで*All*で編集する行を限定できます。たとえば、絞り込んでも特定行のみを編集対象から外したい場合は、M-aで全行をマークし、除外したい行をC-SPCでアンマークすればいいです。

次の手順ならば安心確実に置換処理ができます。

①M-x helm-occurやM-x helm-swoopを実行（**図6**）
②Migemo対応絞り込み検索

注3）URL http://emacs.rubikitch.com/helm-swoop

③（任意）編集対象の行をマーク（図7）
④C-c C-aで*All*に移行（図8）
⑤M-%やC-M-%などで置換（図9）

通常の置換は置換元にマッチした部分をすべて置換対象にしてしまいますが、all-ext経由の置換は置換対象を明確に絞り込めます。

## おわりに

今回はバッファ内検索・置換の上級編をお届けしました。次回は複数ヵ所を同時編集するmultiple-cursors.elを取り上げる予定です。ただの置換では処理しきれないような複雑な編集も楽々こなせます。

筆者はいろいろなことに興味を持っているため、サイトはサブドメイン単位で再構築しました。「日刊Emacs」改め「新生日刊Emacs」（http://emacs.rubikitch.com/）は日本語版Emacs辞典を目指し、毎日更新しています。やりたいことやパッケージから探せるようにしましたので、Emacsで何か実現したいことがあればぜひとも御覧になってください。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プログラムや文章の添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

**▼図6　M-x helm-swoop RET defun**

```
emacs-test
Error-related messages will still be printed, but all other
messages will not.")

^L
(put 'ange-ftp-completion-hook-function 'safe-magic t)
(defun ange-ftp-completion-hook-function (op &rest args)
  "Provides support for ange-ftp host name completion.
Runs the usual ange-ftp hook, but only for completion operations."
  ;; Having this here avoids the need to load ange-ftp when it's not
 -:**-  *scratch*       9% L584    (Lisp Interaction)
 [C-c C-e] Edit mode, [M-i] apply all buffers
*scratch*
584 (defun ange-ftp-completion-hook-function (op &rest args)
601 (defun convert-standard-filename (filename)
633 (defun read-directory-name (prompt &optional dir default-dirname mustmatch ini
tial)
657 (defun pwd (&optional insert)
670 (defun parse-colon-path (search-path)
683 (defun cd-absolute (dir)
 *Helm Swoop* L1    [149 Candidate(s)] C-c ?:Help TAB:Act RET/f1/f2/f-n:NthAct C-!
Swoop: defun
```

**▼図7　編集対象の2行をC-SPCで選択**

```
emacs-test
                                dir))
                  mustmatch initial
                  'file-directory-p))

^L
(defun pwd (&optional insert)
  "Show the current default directory.
With prefix argument INSERT, insert the current default directory
at point instead."
 -:**-  *scratch*       10% L657    (Lisp Interaction)
 [C-c C-e] Edit mode, [M-i] apply all buffers
*scratch*
584 (defun ange-ftp-completion-hook-function (op &rest args)
601 (defun convert-standard-filename (filename)
633 (defun read-directory-name (prompt &optional dir default-dirname mustmatch ini
tial)
657 (defun pwd (&optional insert)
670 (defun parse-colon-path (search-path)
683 (defun cd-absolute (dir)
 *Helm Swoop* L4    M2 [149 Candidate(s)] C-c ?:Help TAB:Act RET/f1/f2/f-n:NthAct C-
Swoop: defun
```

**▼図8　C-c C-aで*All*に移行**

```
emacs-test
;;; files.el --- file input and output commands for Emacs  -*- lexical-binding:t -
*-

;; Copyright (C) 1985-1987, 1992-2016 Free Software Foundation, Inc.

;; Maintainer: emacs-devel@gnu.org
;; Package: emacs

;; This file is part of GNU Emacs.
 -:**-  *scratch*       Top L1     (Lisp Interaction)
From helm-occur
--------
    601:(defun convert-standard-filename (filename)
    633:(defun read-directory-name (prompt &optional dir default-dirname mustmatch
 initial)




 U:---  *All*           All L1     (All)
```

**▼図9　defun→DEFUNに書き換えてC-c C-cで編集が反映されているのを確認**

```
emacs-test

(declare-function dos-convert-standard-filename "dos-fns.el" (filename))
(declare-function w32-convert-standard-filename "w32-fns.el" (filename))

(DEFUN convert-standard-filename (filename)
  "Convert a standard file's name to something suitable for the OS.
This means to guarantee valid names and perhaps to canonicalize
certain patterns.

 -:**-  *scratch*       9% L601    (Lisp Interaction)
From helm-occur
--------
    601:(DEFUN convert-standard-filename (filename)
    633:(defun read-directory-name (prompt &optional dir default-dirname mustmatch
 initial)




 U:**-  *All*           All L3     (All)
```

# 書いて覚えるSwift入門

## 第22回 謹賀新言語


Writer 小飼弾（こがい だん）　twitter @dankogai

### あらためてSwiftを始めませんか?

本稿を執筆しているのは(とくに政治的に)波乱含みどころか波乱しかなかった感のある2016年末ですが、皆さんに届くころには新年気分も過ぎ、2017年という表記に読者の皆さんも慣れているころだと思います。新言語を学びはじめるにはちょうどいいころあいかと。

本連載はこれまで連載ということもあって、すでにSwiftという言語には本連載開始時点から慣れ親しんでいる読者を対象にしてきたのですが、本誌をはじめて手にとった方、つまり本誌にはある程度の理解はあるけれどもSwiftはまだ知らないという読者に向けて、Swiftとはどんな言語なのかを紹介するころあいでしょう。

### 安全、高速、豊かな表現力

いったいSwiftとはどういう言語なのでしょう？　公式サイトのAbout Swift[注1]には3つの特長が掲げられています。

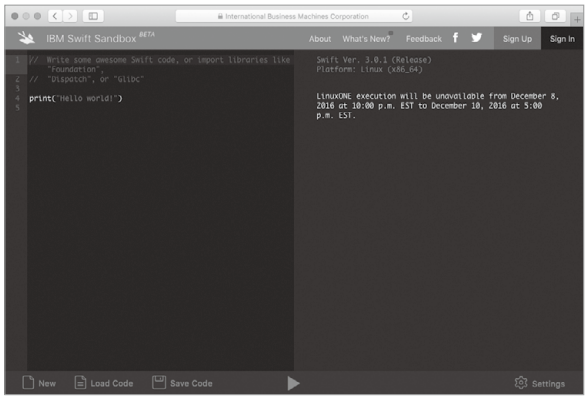
・安全(Safe)
・高速(Fast)
・豊かな表現力(Expressive)

Swiftは静的型を採用していますが、これは安全性に寄与しています。たとえばvar i = 42でInt型として初期化されたものにi = "one"とStringを代入しようとしても、コンパイル時にエラーとして弾きます。SwiftはCやC++やObjective-Cを置き換えることを標榜しているだけあって、同じことを同じように書いた場合、速度はそれらにまさるとも劣りません。それでいて、動的型を採用しているJavaScriptやPerlやPythonやRubyに匹敵する表現力を持ちます。動的言語のように速く書けて、静的言語のように速く実行できる。そのような言語はすべての言語デザイナの夢ですが、Swiftはその夢に最も近い言語だと筆者は感じています。

### Getting Started

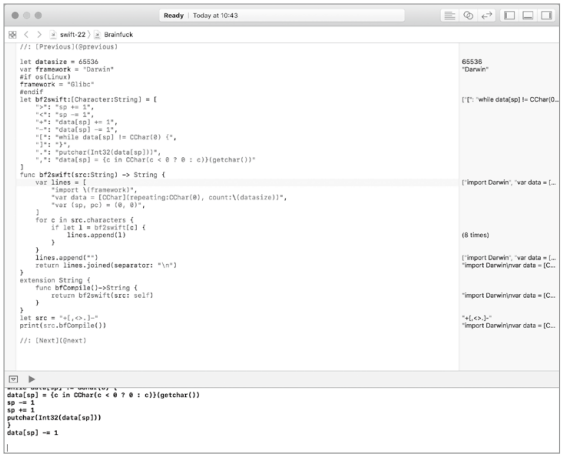
ではさっそくはじめてみましょう。え、Macなんて持ってない？　ご安心を。Swift Sandbox[注2]のおかげで、今やSwiftはWebブラウザさえあればお試しできるのです(**図1**)。

**▼図1　Swiftのsandbox**




注1）https://swift.org/about/
注2）https://swiftlang.ng.bluemix.net/

▼図2　PlaygroundでBrainfuck



▼リスト1　brainfuck.swift

```
let datasize = 65536
var framework = "Darwin"
#if os(Linux)
framework = "Glibc"
#endif
let bf2swift:[Character:String] = [
    ">": "sp += 1",
    "<": "sp -= 1",
    "+": "data[sp] += 1",
    "-": "data[sp] -= 1",
    "[": "while data[sp] != CChar(0) {",
    "]": "}",
    ".": "putchar(Int32(data[sp]))",
    ",": "data[sp] = {c in CChar(c < 0 ? 0 : c)}
(getchar())"
]
func bf2swift(src:String) -> String {
    var lines = [
        "import \(framework)",
        "var data = [CChar](repeating:CChar(0),count:
\(datasize))",
        "var (sp, pc) = (0, 0)",
    ]
    for c in src.characters {
        if let l = bf2swift[c] {
            lines.append(l)
        }
    }
    lines.append("")
    return lines.joined(separator: "\n")
}
extension String {
    func bfCompile()->String {
        return bf2swift(src: self)
    }
}
let src = "+[,<>.]-"
print(src.bfCompile())
```

もともとはApple製品専用開発言語としてスタートしたSwiftですが、オープンソース化にともない、Macだけでなく Linuxも公式にサポートされています。Linux版はWindows 10 Anniversary Editionに搭載されたLinux on Windowsでも動きます。本稿に限らず、本連載ではなるべくプラットフォームに依存しないように当初から心がけているので、本書の連載を理解するにあたってApple製品は必須ではありません。

とはいえ、Swiftの醍醐味をもっとも堪能できるのは、MacのXcodeということになります。とくにPlaygroundは現状Mac版とiPad版しか存在しません。Swiftを100%したかったら、やはりMacが第一選択肢となります。

## 演習：言語処理系を実装してみる

では、さっそく言語処理系を作ってみましょう。え、いきなりハードルが高い？　心配ご無用。ここで実装するのは、その言語でプログラムするよりその言語の処理系を書くほうが簡単と定評のあるBrainfuck[注3]です(**図2**)。**リスト1**はわずか36行ですが、それでも写経するのが面倒だというのであれば、gistにソース[注4]を用意しておきましたので利用ください。

**リスト1**を実行してみると、**図3**のとおりに標準出力されます。**図3**はMacの例ですが、Linuxでは最初のDarwinがGlibcになった以外同じものが出力されるはずです。

これをたとえばecho.swiftというテキストファイルにセーブしたうえで、次のようにシェルからコンパイルして実行してみます。

**注3）** https://ja.wikipedia.org/wiki/Brainfuck
**注4）** https://gist.github.com/dankogai/7020471a37e4e902c70cfe5ff3b7111f

**▼図3　brainfuck.swiftの実行時出力**

```
import Darwin
var data = [CChar](repeating:CChar(0), count:65536)
var (sp, pc) = (0, 0)
data[sp] += 1
while data[sp] != CChar(0) {
data[sp] = {c in CChar(c < 0 ? 0 : c)}(getchar())
sp -= 1
sp += 1
putchar(Int32(data[sp]))
}
data[sp] -= 1
```

**▼表1　CとSwift実装比較**

| BF | C | Swift |
|---|---|---|
| > | sp++; | sp += 1 |
| < | sp--; | sp -= 1 |
| + | data[sp]++; | data[sp] += 1 |
| - | data[sp]-- | data[sp] -= 1 |
| [ | while(data[sp]){ | while data[sp] != CChar(0) { |
| ] | } | } |
| . | putchar(data[sp]) | putchar(Int32(data[sp])) |
| , | data[sp] = getchar() | data[sp] = {c in CChar(c < 0 ? 0 : c)}(getchar()) |

※BF：Brainfuck

```
$ swiftc echo.swift
$ ./echo < echo.swift
```

echo.swiftの中身がそのまま出力されれば成功です。

では処理系中の最後から2番目のsrcを++++++++[>++++++++>++++++++++>+++++<<<-]>.>++.+++++++..+++.>-.------------.<++++++++.--------.+++.------.--------.>+.にしたうえで、もう一度同じようにしてみてください。Hello, world!と表示されましたか？

それではあらためて処理系のソースコードを見てみましょう。こんな小さなコードからでも、次のことが読み解けるはずです(**表1**)。

・変数(var)や定数(let)を定義するにはどうすればよいか
・文字列中に変数を展開(interpolate)するにはどうすればよいか
・型を指定すればどうなるか
　→指定しない場合はどうなるか？
　→bf2swift:[Character:String]の:[Character:String]を消したらどんなエラーが出るか
・辞書(Dictionary)をどう初期化してどう使うか
・配列(Array)をどう初期化してどう使うか
・文字列を1文字ごとに処理するにはどうすればよいか
・既存のデータ型(String)にメソッドを追加するにはどうすればよいか(extension)
・アーキテクチャごとに処理を変えたい場合はどうすればよいか(#if)

余裕があったら、次の課題にも挑戦してみてください。

・BrainfuckにはBrainfuck自身で書かれた処理系が存在するが、その処理系を本処理系で生成してみる
・現状ソースコード中のsrcにハードコードされたBrainfuckコードではなく、任意のソースファイルをコンパイルできるようにしてみる
・Swift処理系をBrainfuckで実装してみる

最後はさすがに冗談ですが、ある言語で簡素な言語の処理系を実装するというのは、その言語を覚える一番の近道の1つ。さすがにBrainfuckほど簡単ではありませんが、Lispの実装などはなかなか楽しそうです。

では、今月はこのへんで。SD

# ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

## 第36回 コマンド名の由来

**catコマンドの由来を誰一人知らないSIerで働いていた担当編集は、『たのしいUNIX』でcatの意味をやっと知り、そんな会社は辞めることにしたのはここだけの話です(遠い目)。**


to be Continued

「コマンド名を作るのって難しいなぁ」と仕事で使う小さいスクリプトにすら悩むので、OSSとして公開してるツールや、システムに組み込まれる可能性のあるツールを作ってるヒトたちはスゴイなぁと尊敬します。そういうコードを書く能力もですが、命名スキルもってことで。他人になんて読まれるかってのも検討課題になりますよね。短く、直感的にわかりやすく、口に出しやすいのが理想かと思ってはいますが、実践できてません。ちなみに、私がナンて読むのかいつも忘れるコマンドはfcitxです。「ファイティクス」と読むそうです。綴りも忘れるので補完機能なしでは入力できません。ええ、Input Methodなのでコマンドする機会ないですね。

# Sphinxで始めるドキュメント作成術

小宮 健　Komiya Takeshi　Twitter @tk0miya

## 第23回 Sphinx拡張の作り方

### Sphinx拡張とは

今回のテーマであるSphinx拡張は、その名のとおりSphinxの機能を拡張するモジュールです。外部のツールやサービスとの連携をはじめとして、ドキュメント作成の助けとなる機能をSphinxに追加します。

本連載でも、これまでに数多くのSphinx拡張を紹介してきました(**表1**)。ひとことにSphinx拡張と言っても、追加する機能はさまざまです。過去の調査[注1]では、世の中には200を超えるSphinx拡張が存在しており、さまざまな機能をSphinxに追加することができます。

Sphinxはこのような拡張を可能とするために、たくさんのインターフェースを提供しています[注2]。このインターフェースを理解すれば、Sphinx拡張を簡単に開発できます。今回はいくつかのSphinx拡張を例に、Sphinx拡張の構造や作り方を紹介します。

### ドキュメントツリー(doctree)

SphinxはreStructuredText(以下reST)の処理系であるdocutilsをベースに実装されています。docutilsは2つのステップでreSTファイルを各種フォーマットに変換します。

①入力となるドキュメントソースを読み込み、ドキュメントツリー(以下doctree)に変換する
②doctreeを対象フォーマット(HTMLなど)に変換する

doctreeは、ドキュメントをツリーで表現するdocutils独自のデータ構造です。読み込まれたドキュメントは章(section)や段落(paragraph)、見出し(title)など、ドキュメント構造を表すノードからなるツリーに変換されます(**図1**)[注3]。

Sphinxはドキュメントを表現する内部形式として、このdoctreeを利用しています。入力ファ

注1) 2014年10月時点。
http://sphinxext-survey.readthedocs.io/en/latest/

注2) Sphinx自体もこの拡張用インターフェースを利用しており、一部の機能はSphinx拡張として実装されています。

注3) Sphinxでは`make pseudoxml`コマンドを使うと、変換されたdoctreeをXML形式で出力できます。

**▼表1　本連載で紹介されたおもなSphinx拡張**

| 名称 | 概要 |
|---|---|
| sphinxcontrib-cacoo(第9回) | Cacooで作成した図を取り込む |
| sphinxcontrib-visio(第9回) | MS-Visioで作成した図を取り込む |
| sphinx.ext.autodoc(第18回) | Pythonドキュメントを自動生成する |
| sphinx.ext.intersphinx(第18回) | 外部のSphinxドキュメントを参照する |
| Shuwa Builder(第22回) | ドキュメントを秀和システム社向けの原稿に変換する |
| columnディレクティブ(第22回) | コラム記述用のディレクティブを追加する |

イル(.rst、.mdファイルなど)1ファイルから、1つのdoctreeが生成されます。その後、各doctreeは出力形式に合わせて、LaTeXなどのように1つに統合されて出力されたり、HTMLなどのように個別に出力されたりします。

## 5つのビルドフェーズ

docutilsではドキュメントの変換を2つのステップで実行していましたが、Sphinxでは5つのビルドフェーズで変換を行います(**表2**)。

Sphinx拡張はこれらのフェーズで特定の処理を実行することで、Sphinxに機能を追加します。たとえば、ソースパーサ(Source Parser)は読み込みフェーズで動作する拡張で、特定の形式のドキュメントを読み込んでdoctree形式に変換します。一例を挙げると、recommonmarkはMarkdownを処理するソースパーサで、Sphinxが読み込みフェーズでMarkdown形式のファイルに遭遇すると呼び出され、Markdown形式のドキュメントを解釈してdoctreeに変換します。

また、書き込みフェーズで呼び出されるビルダー(Builder)と呼ばれる拡張もあります。「sphinxcontrib-dashbuilder」はその1つで、doctreeをDash(APIドキュメントブラウザ)形式に変換します。

このように、Sphinx拡張を開発する際は、それがどのビルドフェーズで処理を行う拡張なのかを意識すると良いでしょう。

**▼表2 Sphinxの5つのビルドフェーズ**

| フェーズ | 説明 |
|---|---|
| 初期化 | Sphinx本体や拡張を初期化する |
| 読み込み | ドキュメントソースを読み込みdoctreeに変換する |
| 検証 | doctreeに矛盾がないか検証する |
| 解決 | クロスリファレンスなどの参照を解決する |
| 書き込み | doctreeを対象フォーマットに変換する |

## ロールの定義(sphinxcontrib-textstyle)

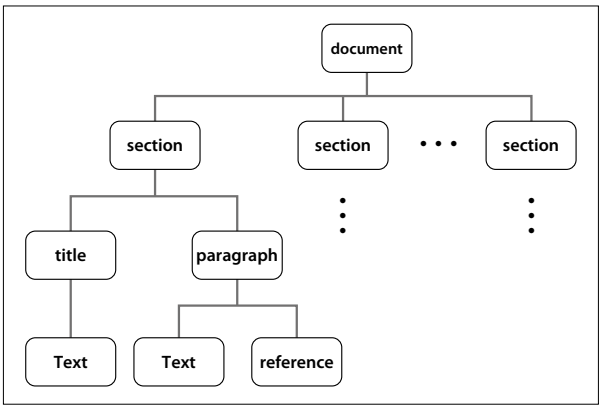
ここからは具体的なSphinx拡張の内部構造を見ていきます。最初に取り上げるのは「sphinxcontrib-textstyle」(以下textstyle)です。

reSTは、ディレクティブとロールという記法の拡張方法を提供します。reST処理系であるSphinxでは、Sphinx拡張からディレクティブやロールを追加できます。textstyleは、ルビや取り消し線などのロールを追加します(**リスト1**)。

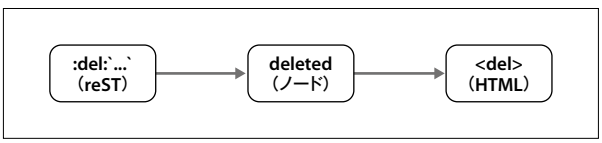
textstyleでは、`:del:`ロールから取り消し線への変換を2つのフェーズを介して行います。まず、読み込みフェーズで`:del:`ロールをdeletedというノードに変換します。そして、書き込みフェーズでdeletedノードをHTMLの<del>タグに変換します(**図2**)。

textstyleはこのフローを実現するため、初期

**▼図1 doctreeによるドキュメント表現**




**▼図2 :del:ロールの処理フロー**




**▼リスト1 sphinxcontrib-textstyleによる取り消し記法の例**

```
Sphinx は :del:`エジプト製の` ドキュメンテーションツールです。
```

化フェーズでいくつかの設定を行っています(**リスト2**)。初期化はsphinxcontrib/textstyle.pyのsetup()という関数で行います。Sphinxはロードされた拡張にsetup()関数が定義されていれば、それを初期化フェーズで呼び出します。

**:del:ロールの追加……❶**

Sphinx APIのadd_role()を使って:del:ロールを追加する。第1引数にはロール名、第2引数にはロール関数を指定する(ロール関数については後述)

**カスタムノードdeletedの追加……❷**

Sphinx APIのadd_node()を使ってSphinx拡張が独自に拡張するノードdeletedを追加する。また、同時にキーワード引数で各種フォーマットへの変換関数(Visitor関数)を指定する(Visitor関数については後述)

次に:del:ロール用のロール関数に指定したdeleted_role()の実装を見てみましょう(**リスト3**)。deleted_role()関数は、Sphinxの読み込みフェーズでreST原稿に:del:ロールが出現するごとに呼び出されます。

ここではユーザが記述したオリジナルのreSTロールrawtextと、そこから抽出されたロールのテキスト部分textを使って、deletedノードを生成しています(❸)。その後、ファイル名や行番号などの情報をノードに付加し(❹)、ノードを返しています(❺)。

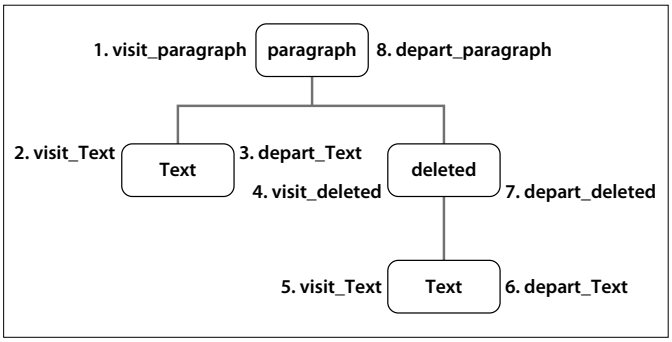
ロール関数は2つのリストを返さなければなりません。最初のリストは:del:ロールが記述されていた位置に埋め込まれるノードの配列、2つ目のリストは:del:ロールが記述されている段落の後ろに埋め込まれるノードの配列です。後者はエラーメッセージ表示などに用いられます。

最後に書き込みフェーズで呼び出されるVisitor関数を見てみましょう。Visitor関数とはvisit関数とdepart関数のペアです。deletedノードのVisitor関数は、visit_deleted()とdepart_deleted()です。

Visitor関数はいずれも、Writerオブジェクトと処理対象のノードを受け取る関数で、doctreeに対象のノード(この場合はdeletedノード)が出現するたびに呼び出されます。visit関数はノードの開始時に、depart関数はノードの終了時にそれぞれ呼び出されます(**図3**)。

そのため、多くのVisitor関数では開始タグをvisit関数で出力し、終了タグをdepart関数で出力します(**リスト4**)。deletedノードに対する

**▼図3　Visitor関数の処理順序**




**▼リスト2　textstyleの初期化処理(抜粋)**

```
def setup(app):
    app.add_role('del', deleted_role)   ←❶
    app.add_node(deleted, html=(visit_deleted, depart_deleted))   ←❷
```

**▼リスト3　:del:ロールの実装**

```
def deleted_role(name, rawtext, text, lineno, inliner, options={}, content=[]):
    text = utils.unescape(text)
    node = deleted(rawtext, text)   ←❸
    set_role_source_info(inliner, lineno, node)   ←❹
    return [node], []   ←❺
```

Visitor関数でも、出力先であるself.bodyに対して<del>タグを出力しています。<del>タグの中身となる削除テキストは、**図3**にあるようにTextノードで表現されており、visit_deleted()とdepart_deleted()の間で、Textノードの Visitor関数によって自動的に処理されます。

なお、textstyleではdeletedノードが子ノードを持つため、visit関数とdepart関数で開始／終了タグをそれぞれ出力しましたが、子ノードを持たないカスタムノードの場合、visit関数だけで処理を完結できます。**リスト5**のhrタグ出力のようにvisit関数の終了時にSkipNode例外を投げると、Sphinxは子ノードの走査とdepart関数の呼び出しを行いません。

## カスタムノードと標準ノード

textstyleは、削除済みテキストを表現するためにカスタムノードを使用します。しかし、拡張で表現したい内容がdocutilsの標準ノードで表現できるのであれば、カスタムノードを使用せずに、標準ノードを使ってドキュメント構造を表現することをお勧めします。

Sphinxはすべての出力フォーマットに向けて、標準ノード用のVisitor関数を提供しています。そのため、Sphinx拡張から標準ノードを利用すれば、新たにVisitor関数を定義せずに済みます。つまり、標準ノードを使えば実装量を減らしながら、さまざまなフォーマットに対応しやすくなります。

標準ノードをうまく活用している拡張の1つに、Excelの表をSphinxに取り込む「sphinxcontrib-exceltable」(以下exceltable)があります。exceltableは標準ノードとして提供されている表や列、セルなどのノードを使って、変換結果を出力します。

docutilsは、表や画像、パラグラフ(段落)などといった一般的なドキュメント構造向けにそれぞれノードを定義しています。したがって、多くの場面で標準ノードが利用できます。Sphinx拡張を開発する際は、あらかじめdocutilsのドキュメント[注4]に目を通しておくと良いでしょう。

## ディレクティブの定義(sphinxcontrib-nicovideo)

次に取り上げるのは、Sphinxドキュメントにニコニコ動画のプレイヤーを埋め込む「sphinxcontrib-nicovideo」(以下nicovideo拡張)です。nicovideo拡張は、nicovideoというディレクティブを提供します。

先ほど紹介したtextstyleではロールを関数として定義しましたが、ディレクティブを定義する際にはクラスとして定義します。nicovideoディ

注4) http://docutils.sourceforge.net/docs/ref/doctree.html

**▼リスト4 deletedノード用のVisitor関数**

```
def visit_deleted(self, node):
    self.body.append(self.starttag(node, 'del', suffix=''))

def depart_deleted(self, node):
    self.body.append('</del>')
```

**▼リスト5 visit関数だけで完結する場合**

```
def setup(app):
    app.add_node(horizonal_line, html=(visit_hr, None))  ←depart関数を指定しない

def visit_hr(self, node):
    from docutils.nodes import SkipNode
    self.body.append('<hr />')
    raise SkipNode  ←子ノードの走査とdepart関数をスキップ
```

レクティブの定義を見てみましょう(**リスト6**)。

まず、ディレクティブのクラスは、docutilsのDirectiveクラスのサブクラスとして定義します(❻)。

このNicoVideoDirectiveクラスには、クラス変数がいくつか定義されています(❼)。これはディレクティブの記法の設定です(**表3**)。Sphinxはこのクラス変数を見て、「nicovideoディレクティブは必須の引数を1つ受け取り(required_arguments = 1)、また、thumbというオプションが指定できる(option_spec)もの」として扱います。

また、NicoVideoDirectiveクラスにはrun()という名前のメソッドが定義されています(❽)。run()メソッドはディレクティブの本体とも言えるメソッドです。reST原稿にnicovideoディレクティブが出現するごとに呼び出され、引数やオプション、コンテンツなどをもとにディレクティブの処理を行います。nicovideo拡張では引数(self.arguments)で指定された動画IDとthumbオプション(self.options)をもとに、nicovideoノードを生成して返しています。Sphinx

**▼リスト6　nicovideoディレクティブの定義**

```
from docutils.parsers.rst import Directive

class NicoVideoDirective(Directive):   ←❻
    has_content = False                    ┐
    required_arguments = 1                 │
    optional_arguments = 0                 │
    final_argument_whitespace = False      ├❼
    option_spec = {                        │
        'thumb': directives.flag,          │
    }                                      ┘

    def run(self):   ←❽
        node = nicovideo(movie_id=self.arguments[0],
                         thumb=('thumb' in self.options))
        return [node]
```

**▼表3　ディレクティブ記法の定義(クラス変数)**

| 変数名 | 初期値 | 概要 |
|---|---|---|
| required_arguments | 0 | 必須のディレクティブ引数の数 |
| optional_arguments | 0 | 省略可能なディレクティブ引数の数 |
| final_argument_whitespace | FALSE | 最後の引数がスペースを許容するかどうか |
| option_spec | {} | ディレクティブオプションの定義(dict形式) |
| has_content | FALSE | ディレクティブがコンテンツを受け付けるかどうか |

**COLUMN**

## あえてカスタムノードを使う

Sphinxに同梱されている「sphinx.ext.graphviz」(以下graphviz)は、DOT記法から生成したグラフ画像をドキュメントに埋め込む拡張です。docutilsでは画像を表すノードとしてimageノードが提供されていますが、graphviz拡張ではあえてカスタムノードを使用しています。

これは出力フォーマットごとに適切な画像形式を選択するためのテクニックです。たとえば、HTMLに変換する場合はHTMLに適したPNG画像を生成しますが、LaTeXへの変換では拡大にも強いベクター形式のPDFデータを生成します。

このテクニックはビルドフェーズの違いを利用したものです。標準ノードを利用する場合、ローダやディレクティブの処理でimageノードを生成することになりますが、読み込みフェーズの時点では出力フォーマットが確定していません。そのため、出力フォーマットに応じて適切な画像を生成することはできません。

一方、カスタムノードを利用する場合は、出力フォーマットごとにVisitor関数を定義できます。この特性を利用して、graphviz拡張では出力フォーマットに適した画像出力を行っています。

標準ノードを利用する場合のメリットと、カスタムノードを利用する場合のメリットをそれぞれ把握して、拡張の特性に合ったアプローチを採ると良いでしょう。

はrun()メソッドが返したノードの配列を、ディレクティブが記述されていた場所に埋め込みます。

残りの工程はロールと同様です。初期化フェーズでSphinx APIのadd_directive()を使ってディレクティブを登録し、Visitor関数にてnico videoノードをHTML要素に変換します。

## Sphinxイベント(sphinx.ext.intersphinx)

その他、Sphinxはさまざまなイベントを提供します(**表4**)。これらのイベントをフックすることで、任意のタイミングでSphinx拡張の処理を実行することができます。

イベントをフックするには、Sphinx APIのconnect()を使ってイベントハンドラを登録します。たとえばSphinxに同梱されている「sphinx.ext.intersphinx」(以下intersphinx)では、**リスト7**のようにハンドラを登録しています。

intersphinxは外部のSphinxドキュメントを参照して、リンクを張る拡張です。たとえばPythonのドキュメントで説明されている関数を、手元のドキュメントから参照できるようにします。これらの外部参照はドキュメントをビルドすると外部リンクに変換されます。intersphinxでは、この動作を実現するために2つのステップで処理を行います。

①外部のSphinxプロジェクトからイベントリ情報(リンク先情報)をダウンロードする
②外部のドキュメントのラベルや関数を参照している個所を、外部リンクに置き換える

ここでは、この2つのステップを実行するタイミングとして、それぞれbuilder-initedイベント(Sphinxの初期化が完了した)とmissing-referenceイベント(クロスリファレンスの参照先が見つからない)を利用しています。

## まとめ&次回予告

今回はいくつかの拡張を例に、Sphinx拡張の構造や作り方を駆け足で紹介しました。今回紹介したほかにも、Sphinxは数多くのAPIや拡張ポイントを提供しています[注5]。これらを利用すると、出力フォーマットやドメインの追加、検索機能の拡張、そしてHTML出力の調整など、多種多様な拡張を作ることができます。

ドキュメントの表現力の向上やビルド処理の自動化、外部のツールやサービスとの連携など、ドキュメント作成の手助けとなるようなアイデアをお持ちでしたら、ぜひともSphinx拡張を作ってみてください。

次回はSphinxと情報デザインについて紹介します。SD

注5) 詳細は開発者ドキュメントを参照のこと。
http://www.sphinx-doc.org/ja/stable/extdev/index.html

**▼リスト7 intersphinxの初期化処理**

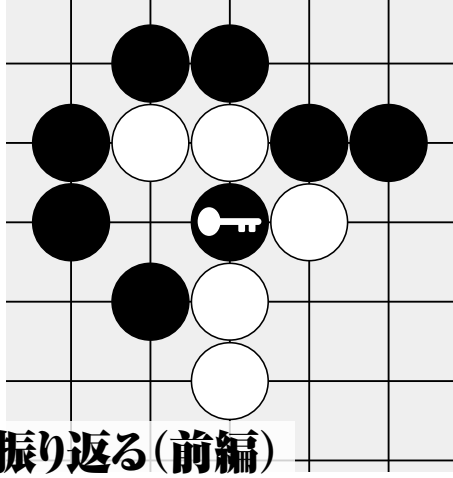
```
def setup(app):
    (..略..)
    app.connect('missing-reference', missing_reference)
    app.connect('builder-inited', load_mappings)
    (..略..)
```

**▼表4 おもなSphinxイベント**

| イベント名 | 概要 |
|---|---|
| builder-inited | ビルダーを含む、Sphinxの初期化が完了したときに発生するイベント |
| source-read | 入力ファイル(reSTファイルなど)を読み込んだときに発生するイベント。読み出したファイルの内容を差し替えることができる |
| doctree-read | 解決フェーズの直前に発生する、doctreeを読み込んだときに発生するイベント。読み出したdoctreeの内容を差し替えることができる |
| missing-reference | 解決フェーズにおいて、クロスリファレンスの参照先が見つからない場合に発生するイベント |
| doctree-resolved | 解決フェーズの最後に発生するイベント |
| build-finished | ビルドが完了したあとに発生するイベント |

# セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



## みんなでもう一度見つめなおそう

## 【第四十回】2016年のセキュリティの状況を振り返る（前編）

**2016年を振り返る形で、注目に値するセキュリティの動きを挙げていきます。情報セキュリティの俯瞰的（ふかんてき）な視点というわけではなく、あくまで筆者の目から見て注目したという前提で挙げていきます。**

### 1月

#### Anonymousが日本を攻撃対象に

「ハッカー集団」と呼ばれるAnonymousですが、直訳してわかるように「匿名の誰か」であり、具体的に特定の人物が統率をとっているわけではありません。誰かが呼びかけ、そこに賛同し行動を起こす。そのときに集まった人々が自分たちをAnonymousと呼んでいる。そう理解したほうがいいでしょう。

彼らが行っている活動の1つに「#OpKillingBay」があります。これは、日本のイルカ漁への対抗として、おもに「.go.jp」のドメイン名を持つサイトに対して、ここ数年継続的に行っているDDoS攻撃です。これは今後も引き続き行われることでしょう。

#### CIA長官のアカウントがクラックされる

ティーンエージャーの手で、CIA長官のJohn Brennan氏のAOLのアカウントがクラックされました。AOLは、昔はパソコン通信で一世を風靡（ふうび）したサービスです。現在では米Verizon社が買収し管理しています。クラックの手法は、同社のカスタマーサービスに電話をし、パスワードをリセットしてもらうという古典的とも言える欺術（ぎじゅつ）[注1]を用いたようです。国家安全保障上の問題はなかったとのことです（今どきAOLを活発に使っているとも思えないですし）が、この事実そのものが話題になりました。

ちなみにAOLではありませんが、筆者も使わずにほったらかしにしているWebサービスが多くあります。もう使っていないメールアドレスで登録してあり、パスワード変更のしようのないサービスもあります。そんなケースを考えると、今回のパスワードリセットの方法が効果的なのも理解できます。

### 2月

#### FBIから2万人分の個人情報が流出

2016年2月8日に、@DotGovsというTwitterアカウントから2万人分のFBI職員の名前、肩書き、電話番号、メールアドレス、国が列挙されているファイルのURLとパスワードが投稿されました。このアカウントはすでに凍結されてファイルは取得できませんが、米Webメディア「Motherboard」の記事によると、このファイルの中身は正しく、さらにFBI以外にDHS（米国土安全保障省）の9,000人分の個人情報も公開されていたそうです。

このツイートには#FreePalestine（「パレスチナに自由を！」の意味）のタグが付いていました。また、公開された個人情報ファイルの最初には“Long Live Palestine, Long Live Gaza,”（「パレスチナ万

注1） Social Engineeringという用語に対し、岩谷宏氏がつけた秀逸な訳語。

歳、ガザ万歳」の意味。）と入っていました。典型的なハックティビズム[注2]に見えます。

これらの情報をどのように入手したかは明らかではありませんが、FBIとDHSというセキュリティがもっとも高そうな政府組織の職員の情報が大量に公開されているのですから、穏やかではありません。

## 病院のPCにランサムウェアが感染

ランサムウェアとは、感染先のPCのハードディスクのファイルを暗号化し使えなくしたうえで、被害者に金銭的な要求をする「身代金（ransam）」目的のマルウェアです。感染を広げたり、金銭的要求をしたりするための情報管理には、既存のボットネットのインフラ（Gameover ZeuSなど）を使う手法を採っています。欧米ではここ2、3年、爆発的な広がりを見せています。犯罪のモチベーションは金銭ですから、広がるのも無理はないと思います。

実務で使うPCに感染すると大きな問題になるのは言うまでもありませんが、米国ではさらに悪いことに、病院でランサムウェアに感染してしまい、暗号化された患者の治療記録などが入っているファイルを諦めるわけにもいかず、ビットコインで40BTC[注3]を払いファイルをもとに戻しました。

## Linux Mintサーバがクラックされる

Linux Mintは使い勝手が良く見た目がかっこよい（これが人気の秘訣だと筆者は考えています）、DebianとUbuntuをベースとしているデスクトップ向けディストリビューションです。欧米では常にトップクラスの人気を保っています。そのLinux Mintのサーバがクラックされ、インストール用ISOファイルが改竄（かいざん）されたものに置き換えられていました。これは非常に大きな問題です。

たとえば、利用（運用）しているGNU/Linuxに、新規のソフトウェアを導入する際や、既存のソフトウェアをアップデートする際のことを考えてみてください。このとき、インストールパッケージをダウンロードするわけですが、たとえ配布用サーバに侵入されインストールパッケージが汚染されていたとしても、パッケージには電子署名がかかっているので、正規のパッケージでないことを検出できます。

ただし、インストール用ISOファイル、つまり一番最初のものだけは、唯一電子署名のメカニズムは使えず、人間の手によって確認する必要があります。それが済めば、検証鍵（公開鍵）を利用できるので、電子署名が使えます。このように最初に使ったものをその後も信じて使う方式を、TOFU（Trust On First Use）と呼び、ベストプラクティスとして十分に運用可能だと信じられています。

しかし、それも最初が正しいという前提があってこそです。一応、各ディストリビューションのサイトにはインストール用ISOファイルのフィンガープリント（SHA256などの出力を用いたファイルの指紋）がありますが、万が一そこまで改竄されていたならば防ぎようがありません。最初が一番の弱点であり、今回はそこを突かれた形になります。

# 3月

## 出会い系サイトの個人情報が流出

出会い系サイト「Mate1.com」の登録ユーザのメールアドレスと平文パスワード2,700万人分が流出し、それが闇サイトで売りに出されたことが発覚しました。業者の持つアカウント情報が数百万人とか数千万人という規模で流出するのは、珍しいことではありません。ここで注目すべき点はパスワードが「平文」で管理されているという点です。

本連載第1回（2013年7月号）でパスワード認証の実装モデルを解説しましたが、このようにパスワードを「平文」で保存しているのは極めて脆弱です。もっとはっきり言えばパスワードのしくみを理解していません。基本的な知識がないままシステムを作り、多数のユーザを抱えて運用している現実があるということがこのケースからわかります。

またパスワードは平文で流出しているので、この

注2） サイバー攻撃を手段として、政治的主張を行うこと。
注3） 2016年2月時点では「1BTC＝約400ドル」「1ドル＝約115円」なので、40BTCはドルで16,000ドル、日本円で約184万円です。

パスワードをほかでも使いまわしているならば（しかも、ユーザアカウントをメールアドレスで管理しているようなサイトで利用しているならば）、アウトです。Webサービスなどで使う際のパスワードは、ひとつひとつユニークにすべきである、という理由がよくわかるケースだと言えるでしょう。

### ニューヨーク州のSCADAに攻撃

米国内サイトに対してサイバー攻撃を繰り返し行っていた7名のイラン人を、米国司法省が手配したと公表しました[注4]。彼らはイランに本社を持つITSecTeam社やMersad社で働いており、その背後にはイスラム革命防衛軍が存在している、と司法省のサイトでは説明しています。彼らは2011～2013年の間、金融業界の会社など46ヵ所に176日間という長期間に渡りDDoS攻撃をしていました。

深く考えなくてはならないのは、ニューヨーク州にあるダムを監視／管理するSCADA（Supervisory Control And Data Acquisition）にもDDoS攻撃を行っていたことです。見方によってはサイバー戦争の前哨戦とも言えるのではないでしょうか。

## 4月

### トルコ国民約5,000万人分の個人情報が流出

トルコ国民の約60%にあたる約5,000万人分の氏名、個人番号、住所がインターネット上で公開されるという事件が発生しました。どうもトルコでは各政党が選挙時に選挙人名簿を入手できるようで、そのデータが外部へ流出したようです。トルコ現政権であるエルドアン大統領を批判しているメッセージも同時に出されていたという報道もあり、これもハックティビズムの一種だと考えられます。

選挙人名簿でかつ個人を特定するためのID（日本でのマイナンバーにあたるもの）が大規模に公開されているわけですから、それを使った詐欺など2次、3次の被害が考えられる深刻な事件です。

このような情報を選挙時に各政党が入手できるというのは、何でも情報が使える政権側との公平性を考えるとアリなのかもしれませんが、FBIやDHSの職員の個人情報が大量に流出する時代ですから、一般の事務処理にプラスアルファ程度の情報管理をしているレベルでは、守り通せるとは思えません。

現在、日本でマイナンバーの利用範囲を拡大する議論が進んでいますが、この流出事件が示すように、利用範囲が広がるほど流出のリスクは増えます。マイナンバーの変更にも手間がかかります。あと10年もすれば、だだ漏れ状態で放置されるような状況になっていて、「法制度を作るときの議論はいったいなんだったのか」という結果になっているかもしれません。

## 5月

### ギリシャの中央銀行へのDDoS攻撃

Anonymousの活動「#OpIcarus」の一環として、ギリシャ銀行へDDoS攻撃が行われました。報道を見る限り、日本のイルカ漁に対する#OpKillingBayの活動と変わりないようなのですが、攻撃の質が変わってきたのか、ギリシャ銀行の対応が遅れたのかわかりませんが、たいへん苦しめられたようです。

### 大手DNSサービス業者NS1へのDDoS攻撃

DNSサービスを提供するNS1社へDDoS攻撃が行われました。DNS水責め攻撃やUDP floodなど複数の攻撃がミックスされ、攻撃パターンも単純ではなかったとのことです。欧州方面のDNSサービスに影響が出たのと、Imgurなどの有名サイトでも影響が出ましたが、短い期間で対応できたようです。

2016年を振り返ると、NS1への攻撃は、このあとの6月、9月の大規模なDDoS攻撃であるMIRAIの前哨戦とも言える位置づけだったことがわかります。同社のブログ[注5]に、一連のDDoS攻撃について言及があるのですが、流量的には20～30Gbps

注4） https://www.fbi.gov/news/stories/iranians-charged-with-hacking-us-financial-sector
注5） https://ns1.com/blog/how-we-responded-to-last-weeks-major-multi-faceted-ddos-attacks

で、パケット数(DNSクエリ数)は10～20Mパケット/秒(pps)だったそうです。また、この程度であれば対応できるとも説明されていました。これから見ると、後のMIRAIの攻撃が常軌を逸したものであったこと、インターネット史上最大の脅威であったということがわかります。

## 6月

### 民主党全国委員会への攻撃

米国の民主党全国委員会(The Democratic National Committee、以下DNC)の内部から電子メールがWikiLeaksに流出しました。Guccifer 2.0というハッカーブログで、自分たちが行ったということを発表しています。しかし、複数のセキュリティ企業および専門家は、ここ2年間の攻撃はCozy BearとFancy Bearと呼ぶ2つのロシアチームが関わっていると分析しています。

この2チームに関しては、DNCだけではなく、ほかの米政府組織への攻撃も長く行っていることが知られています。これらのチームがロシア政府と関係しているかどうかの確証はどこにもありませんが、この手のプロフェッショナルとして動いているチームは、どこかにスポンサーがいるはずです。

今回のDNCの流出事件は、必ずしもメールサーバへの侵入は必要ありません。マルウェアがPCに感染してファイルが流出し、その中に電子メールが含まれていた可能性もあります。この手のマルウェアが流出対象のファイルを選択するとき、電子メールのファイルがあれば、真っ先に流出させます。ですから、ロシアのチームが長年行っていたことと、電子メール流出が別の機会に発生したことは必ずしも矛盾するわけでありません。

このWikiLeaksに流出した内容が、この年行われた米大統領選に大きな影響を与えたことは誰もが認めることです。また、その影響がロシア政府への柔軟な姿勢を見せる候補者であったドナルド・トランプ氏に大きくプラスに働きました。この先もDNCの問題は長く影を落とすことになるでしょう。

### 仮想通貨投資ファンドThe DAOへの攻撃

The DAOは仮想通貨イーサリアム(以下ETH)の投資ファンドです。The DAOがスタートするときに、ドル換算すると1億ドルを越える約1200万ETHもの資金が短い期間で集まったことでも話題になりました。ところが、DAOが開発しETHを管理している分散システムにバグがあり、第三者によって364万ETHの資金が移動させられるというセキュリティ上の問題が発生しました。DAOの仕様ですぐには換金できないようになっているので、「盗む」とは表現せずに「移動」という表現が使われています。

この事件のあと、この攻撃を行ったという者から声明が出ました。「The DAOの方式で仮想通貨を扱うと、スマートコントラクトやブロックチェーン方式の信頼(Confidence)を永遠に損なう」そのことを無視してThe DAOは事業を進めている、という事実を世間に注目させたかったのが動機のようです。その後に、別の攻撃者がまたもや同じバグをついて約700万ETHを移動させました。これで大半が移動させられたことになります。The DAOが、開発したソフトウェアの品質は低いということを、このような形で露見させてしまいました。

80年代からソフトウェア開発に関わってきた筆者の経験から言えば、安定したシステムが運用できるまでには長い期間が必要です。開発期間もリリースしてからの運用期間も短いソフトウェアで100億円を越える仮想通貨を扱うリスクをThe DAOがどう考えていたのか、筆者には理解できませんが、たとえ警告をしたとしても、「欲」に動かされている熱狂的な仮想通貨ブームの中では焼け石に水だったでしょう。不安定な技術の上で多大な価値が動き、それが攻撃を受けるという同様な事件は今後も続くと思います。

## 7月以降は次回

2016年の前半だけで誌面がつきてしまいました。7月以降については次回に議論します。SD

# SOURCES

## レッドハット系ソフトウェア最新解説

第6回

# Red Hat OpenShift Container Platform Part2

前回は、複数台構成のOpenShift環境の構築方法を紹介しました。今回は構築したOpenShift環境のメンテナンス方法や運用時の基本的なポイントなどを紹介していきます。

Author 小島 啓史(こじまひろふみ)
mail：hkojima@redhat.com
レッドハット㈱ テクニカルセールス本部 ソリューションアーキテクト

## OpenShift環境のメンテナンス

前回の記事で紹介したOpenShift環境を主体に解説します。環境のメンテナンスという観点で考えなくてはならない代表的な項目は、アプリケーションを実行するNodeの状況確認や追加・削除といったものがあります。その場合は、OpenShift環境の全管理権限を持つsystem:adminユーザで次のコマンドを実施します。

```
# oc login -u system:admin
# oc get node
# oc describe node
```

補足ですが、AnsibleでOpenShift環境を構築する際には、各Masterでsystem:adminユーザを利用するための認証情報が記載された設定ファイルが、MasterのAnsibleユーザ(前回の例ではrootユーザ)の「~/.kube」ディレクトリに作成されます。上記コマンドはMasterで実施することを前提としますが、もし他のサーバで実施したい場合は、Masterの「~/.kube」ディレクトリをコピーして、atomi-openshift-clientsパッケージをインストールする必要があります。

「oc get」で各リソースの一覧と状況を、「oc describe」で各リソースの詳細情報を確認できます。「oc describe node」を実行した場合は、NodeのIPアドレスや起動中のPod、CPU/Memoryのリソース利用状況などを確認できます。ここで何らかの不具合を確認したため、サーバ修理などで一時的にNodeをOpenShift環境から取り外したいという場合は、「oc adm」コマンドで当該NodeからPodを移動させる必要があります。

```
# oc adm manage-node NODE_NAME ➡
--schedulable=false
# oc adm manage-node NODE_NAME --evacuate
```

「--schedulable=false」でPod配置を無効化し、「--evacuate」で他Node上にPodを作成した後に当該Node上のPodを削除します。当該Nodeの復旧が完了したら、同じく「oc adm」コマンドの「--schedulable=true」オプションでPod配置を有効化することで、ふたたびNodeとして使えるようになります。

## Nodeの追加・削除

アプリケーション実行に必要なCPUやメモリなどのリソースが枯渇している、またはアプリケーション実行サーバを集約したい場合は、Nodeの追加・削除を実施します。Nodeの追加には、

OpenShift環境構築時と同様にInventoryファイルを作成してPlaybookを実行します(**図1**)。

既存のOpenShift環境にNodeを追加する場合は、Inventoryファイルで既存のMaster/Nodeの情報を記載し、[new_nodes]で新しいNodeの情報を記載する必要があります。これにより既存のMasterが管理するNodeの情報に新しいNodeが追加されます。ここでは通常のNodeを追加していますが、Infra Nodeを追加する場合は「region=infra」ラベルをInventoryファイル内で指定してPlaybookを実行します。この時、追加したInfra NodeでRouter Podが起動していない場合は、Router Podのレプリカ数をInfra Nodeの台数に変更することで、Router Podが自動的に新規作成・起動されます。

```
# oc scale rc router-1 --replicas=REPLICA_NUMBER -n default
```

また、「oc delete」で指定したリソースをOpenShift環境から削除できます。Nodeを削除する場合は、当該NodeからPodを移動させた後に「oc delete node」を実行します。この時、Nodeに付けられたラベルを「-l」で指定することで、特定のラベル(zone=zone01など)が付けられたNodeを一括削除することもできます。

```
# oc delete node NODE_NAME
# oc delete node -l zone=zone01
```

そしてPlaybookでOpenShift関連のパッケージやファイルを削除する[注1]ことで、削除したNodeをクリーンアップできます。必要に応じて、Dockerコンテナやイメージもdockerコマンドで削除します。なお、余談ですが、各NodeはGarbage Collection[注2]の機能を備えており、停止した古いコンテナや利用されていないDockerイメージを自動的に削除してくれるため、ディスク容量の圧迫が起きにくくなっています。

## Registry Console

構築したOpenShift環境内の特定のプロジェクトを利用してアプリケーションを開発している場合、デフォルトのopenshiftプロジェクトに

注1) URL http://red.ht/2fY7eFP
注2) URL http://red.ht/2gWDBlj

**▼図1 Nodeの追加**

```
# cat <<EOF > /root/ansible-add-nodes
[OSEv3:children]
masters
nodes
new_nodes

[OSEv3:vars]
ansible_ssh_user=root
deployment_type=openshift-enterprise

[masters]
master1.example.com
master2.example.com
master3.example.com

[nodes]
infra[1:2].example.com openshift_node_labels="{'region': 'infra'}"
node[1..2].example.com openshift_node_labels="{'region': 'primary', 'zone': 'zone01'}"

[new_nodes]
node3.example.com openshift_node_labels="{'region': 'primary', 'zone': 'zone02'}
EOF
# ansible-playbook -i /root/ansible-add-nodes \
    /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-node/scaleup.yml
```

あるDockerイメージ以外も利用したい場合があります。その時は、OpenShiftの機能の1つであるRegistry Consoleを利用すると便利です。Registry ConsoleとはOpenShift環境内のDocker Registryを利用するためのWeb UIを提供するものであり、開発時に利用しているプロジェクト内でのDockerイメージが、プロジェクトの参照・管理権限を持つ開発者間で共有できるようになります。Registry Consoleにアクセスするための URLは、「oc get route」で確認できます(**図2**)。

defaultプロジェクト内で、registry-consoleというルーティング名に紐付けられているホスト名を利用します。**図2**では、「https://registry-console-default.cloudapps.com」にWebブラウザからアクセスします。ログインには、OpenShift環境が利用している認証情報をそのまま使います。ログインすると**図3**が表示されます。

ここで現在ログイン中のユーザが利用しているプロジェクト、プロジェクト内から参照できるDockerイメージ、Docker Registyr内のプロジェクトにログイン[注3]してDockerイメージをpush/pullする方法が確認できます。このような情報を参照すると、**図4**のようなコマンドで、指定したプロジェクトにカスタムDockerイメージをpushできます。

なお、特定のプロジェクトにあるカスタムDockerイメージをOpenShift環境の全ユーザに公開する場合は、system:adminユーザでopenshiftプロジェクトのタグを付けます。

```
# oc tag PROJECT_NAME/CUSTOM_IMAGE:latest openshift/CUSTOM_IMAGE:latest
```

これでopenshiftプロジェクトにあるDockerイメージを利用して、他プロジェクトの開発者がアプリケーションをデプロイできるようになります。ただし、管理者が開発者向けにカスタムDockerイメージを用意することがある場合、適当なユーザにopenshiftプロジェクトの管理権限を追加して、そのユーザ権限を持って管理者が直接openshiftプロジェクトにイメージのpushを実行することを推奨します。

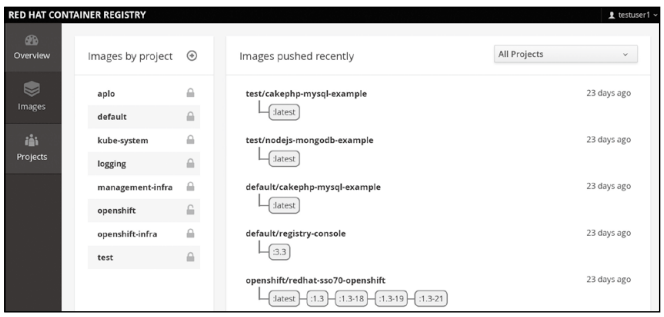
```
# oc adm policy add-role-to-user admin USERNAME -n openshift
```

なお、OpenShift環境ではDockerイメージからアプリケーションを起動する際に、セキュリティを考慮してコンテナ内のプロセスのrootユーザでのプロセス実行を禁止しており、プロセス実行時にはランダムなUIDが割り当てられま

**▼図2　Registry Consoleへのアクセス**

```
# oc get route -n default
  …… (中略) ……
NAME                HOST/PORT
docker-registry     docker-registry-default.cloudapps.com
registry-console    registry-console-default.cloudapps.com
```

**▼図3　Registry Consoleのダッシュボード**




**▼図4　Dockerイメージの Push**

```
# docker login -p    ←Registry Consoleで表示されるログインコマンドを実行
# docker tag CUSTOM_IMAGE:latest \
    docker-registry-default.cloudapps.com/PROJECT_NAME/CUSTOM_IMAGE:latest
# docker push docker-registry-default.cloudapps.com/PROJECT_NAME/CUSTOM_IMAGE:latest
```

注3）URL https://docs.docker.com/engine/reference/commandline/login/

▼図5　OpenShift関連サービスのログ取得方法

```
# systemctl status atomic-openshift-master-{api,controllers}.service ←Masterで実行
# systemctl status atomic-openshift-node.service ←Nodeで実行
```

▼図6　OpenShift環境の情報収集コマンド

```
Podのログを出力
# oc logs POD_NAME

各Nodeで実行して、Podへのルーティング情報を確認
# iptables -t nat -nL

Docker関連の情報を収集しアーカイブを作成
# sosreport -e docker -k docker.all=on
```

す。そのため、コンテナ内のプロセスは任意の一般ユーザで実行できるように、Dockerfileでの一般ユーザの指定と設定／実行ファイルのアクセス権追加(chmod ugo+rxなど)を忘れないようにしてください。

## ログとバックアップ

OpenShift環境運用時の情報収集や問題解決には、他のシステムと同様にログや診断ツールなどを活用します。OpenShift関連のサービスの状況確認やログは**図5**のコマンドで確認できます。

OpenShiftのMaster/Nodeに関するサービスはsystemdに統合されていますので、「systemctl status」コマンドでサービスの状況やログの一部を確認できます。ログレベルはMaster/Nodeの設定ファイル「/etc/sysconfig/atomic-openshift*」に記載されています。デフォルトでは「--loglevel=2」が定義されています。デバッグモードにしたい場合は、「--loglevel=5」に変更して、Master/Nodeの各サービスを再起動する必要があります。また、ocコマンドのログレベルも実行時に指定できます。6から8の数字が指定でき、「--loglevel=8」を指定すると最も多くのログを出力します。

RHEL7ではロギングのしくみとしてjournaldとrsyslogが並行して起動しています[注4]。そのため、OpenShift関連サービスのログも「/var/log/messages」ファイルに出力されます。OpenShift関連サービスのログだけを出力したい時は、journalctlコマンドを使います。ただし、デフォルトではjournaldのログは永続化されていませんので、永続化の設定[注5]を忘れないようにしてください。

```
# journalctl -u atomic-openshift-*.service
```

また、OpenShift環境全体の診断ツールには、「oc adm diagnostics」コマンドがあります。このコマンドでErrorが出力されていないかを確認します。そのほかには**図6**のようなコマンドも使えます。こうした問題解決のノウハウを、弊社サポート契約者向けに公開[注6]していますので参照ください。

バックアップは、etcdのデータディレクトリ[注7]、各Master/Nodeの設定ファイル[注8]、Persistent Storageとして利用しているストレージ、といった要素についてそれぞれ取得する必要があります。また、各プロジェクトのバックアップも、「oc export」コマンドでリソース情報をYAMLファイルに出力する[注9]ことにより取得できます。

## まとめ

今回はOpenShift環境のNode/Dockerイメージ/ログなど、運用時に考慮する代表的な項目の管理方法を解説しました。次回はOpenShiftに関連したSoftware-defined storageの話を紹介します。SD

注4）URL http://red.ht/1GW6p2O
注5）URL http://red.ht/2g9PC8n
注6）URL https://access.redhat.com/solutions/1542293
注7）URL /var/lib/etcd/ (etcdをクラスタ化していない場合は/var/lib/origin/openshift.local.etcd/)
注8）URL /etc/origin/, /etc/sysconfig/atomic-openshift*, /etc/etcd, /etc/sysconfig/docker*
注9）URL http://red.ht/2gqRirJ

43

Debian Developer　やまねひでき　henrich@debian.org

# Mini Debian Conference Japan 2016レポート

# Debian Hot Topics

## Mini Debian Conference Japan 2016開催

開発のフリーズのフェーズも徐々に進んでいるDebianですが、今回は開発のことを離れて、国内で開催されたイベントについて報告します。

2016年12月10日に東京・日本橋のサイボウズ㈱にて、「Mini Debian Conference Japan 2016」が開かれました(**写真1**)[注1]。2トラック、9セッションが行われたこのイベントの模様を、かいつまんでお伝えします。

### Open Build Service in Debian

まずはAndrew Lee(李健秋)さんによる「Open Build Service」(以下OBS)[注2]をDebianパッケージにしたという発表です。以前、本連載内でsnapperとopenQAを紹介しましたが、OBSもそれらと同様openSUSE由来のプロダクトです。

OBSは、openSUSE/SUSE Linux Enterprise Serverのパッケージビルドの中心的なサービス(Debianでいうところの Build daemon[注3]のようなもの)で、そこで動いているソフトウェアもOpen Build Serviceと呼ばれています。

ちなみにOBSはopenSUSEだけではなく、Fedora、Debian、Ubuntuのパッケージもビルド可能という意欲的なものです。さらに単なるパッケージビルドのみならず、パッケージング作業のポータルサイトとしても機能しています[注4](LaunchpadやGitHubのようなものを想像してもらえば良いかと思います)。継ぎはぎで拡張を続けてきたDebianのビルドシステムとパッケージトラッカーよりも後発だけあって、きれいにアーキテクチャが考えられており、スマートにまとまっている印象を受けました。

Andrewさんによると、勤務先のCollabora社[注5]での業務で必要になったために、2年ほど前からずっとパッ

注1) URL http://miniconf.debian.or.jp/
ちなみに同日、同じオフィス内で「LibreOffice Kaigi 2016.12」も併催されていました。

**▼写真1　会場の様子**

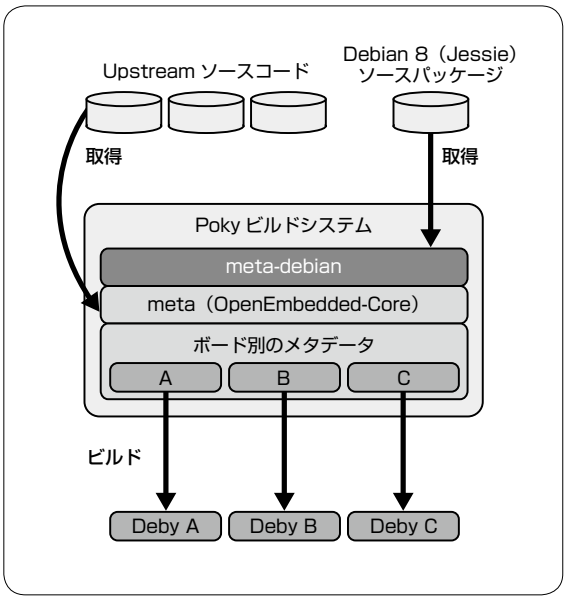
注2) URL http://openbuildservice.org/
旧称はopenSUSE Build Service。
注3) URL https://buildd.debian.org/
注4) URL http://www.slideshare.net/ftake/1-open-build-service
注5) URL https://www.collabora.com/
OSS関連の開発やコンサルティングを行っており、LibreOfficeの主要開発企業の1つでもある。

ケージング作業をしていたとのこと。OBSのフロントエンドを構成するRuby on Rails周りの知識がなかったために苦労したそうで、先日、台湾でRubyConfが開かれたことから、そこにも参加していろいろと参加者に質問していたそうです。OBSのために数十個ものgemを公式Debianパッケージにし、その過程で不明瞭なライセンスの問題に対処したり、埋め込まれているJavaScriptを取り除いて既存のDebianパッケージ側を使うようにするなどの対処をしたり、gemのDebianパッケージが壊れていたのを直したりと、かなりたいへんだったようです。

この記事を執筆している現在、Debianではobs-buildパッケージとその関連パッケージがunstableに投入されています。Collabora社のサイトにはDebian開発者のHector Oronさんによるチュートリアル[注6]があるので、試してみてください。個人的にはupstreamとの3-way mergeを実行する「Merge-o-Matic」が気になっているので、そのうち試してみようと考えています。

注6）Part1 URL https://goo.gl/OSBNqv
Part2 URL https://goo.gl/2rNPMx

▼図1 Deby（Poky＋meta-debian）




## Deby

「Debianソースコードを用いた組込み向けLinux環境作成」という小林良岳さんの発表では、㈱東芝が組み込み機器向けに作成している「Deby」という名のカスタムディストリビューションの紹介が行われました。

Yocto Project[注7]が提供する、組み込みデバイス向けのカスタムビルドを生成する「Poky」というツールがあります。Debyは、このPokyをベースにして、Debianパッケージのソースを使う「meta-debian」というPokyのレイヤ[注8]を追加したものです（**図1**）。

meta-debianは、Debianのソースパッケージをベースに、クロスコンパイルの追加や、東芝が組み込みデバイス向けでは不要だと考えた機能とその依存関係を削除するなどのカスタマイズが行えます。現在のrecipe（パッケージ）数は約500ほどあります。

Debianをベースにすることで、複数のCPUアーキテクチャの利用とそこでの安定性に対して0から試行錯誤するのではなく、Debianでの作業成果の流用が可能になっているわけですね。

組み込み機器ではサポート期間が重要になりますが、Debian安定版の約3年では東芝が期待する期間には足りません。そこで東芝は、Debian安定版よりさらに2年間長くセキュリティアップデートを行っているDebian LTSプロジェクトのスポンサーになり、支援を行っているそうです[注9]。

今後どのような形でUpstreamであるDebianへコントリビューションを行って

注7）URL https://www.yoctoproject.org/ 組み込み製品のためのLinuxベースのカスタム・システムをハードウェア・アーキテクチャに関わらず構築するための、テンプレート、ツール、手段を提供するプロジェクト。
注8）URL https://github.com/meta-debian/
注9）URL https://www.freexian.com/services/debian-lts.html 東芝は最も高額のスポンサーである「プラチナ」。ちなみにもう1社のプラチナスポンサーはGitHub社です（彼らのプロダクトであるGit Hub EnterpriseにDebianを使っている、という話をどこかで見かけましたので、その関係でしょうか……真偽をご存じの方は教えてください）。

いくのかという議論では、

- 組み込み機器ではライセンス確認をきちんとしているので、その確認結果をDEP5形式[注10]で記述するなどの形でフィードバックすれば良いのではないか
- クロスコンパイルについては、Debianではなく、Debyのもう1つのUpstreamであるOpenEmbedded-Core[注11]にフィードバックするほうが良いだろう

などの話が出ました。Downstreamでの開発においてUpstreamとの差異(delta)をどのように吸収していくか(最小化していくか)、というのは重要なポイントですので、両者にとってメリットのある形で実現していくと良いですね。

## OSS版初音ミク?「徴音梅林(ちおんめいりん)」

異色だったのは張正一さんによる「FOSSバーチャルシンガー徴音梅林とLINNEプラットホーム」という発表です。初音ミクをはじめとした「ボーカロイド」については、多くの方がご存じかと思います。これらのボーカロイドはプロプライエタリ・ソフトウェアですので、ソースコードが公開されていないことは当然として、その使用についてもいくつかの制限がついています[注12]。通常の「楽器」は、購入者がその楽器をどのように使用してもかまいません(例として、ギターを歯で弾いたり、ステージ上で破壊・火をつけたりしたミュージシャンが挙げられました)が、ボーカロイドには「制限」があります。

これに対して、山梨大学の森勢将雅博士が開発した「WORLD」[注13]をベースにしたソフトウェア「徴音梅林」[注14]はCreative Commonsライセンス(CC-BY)で提供されているOSSで、ボーカロイドのような使用制限はいっさいありません。その性能はプロジェクトのページからリンクされているYouTube動画で確認できますので、ご一聴ください。

WORLDをベースにすることで、ヤマハが多数所有する特許にも抵触しないなどのメリットもあります。現在、中国語(マンダリン)と日本語(+関西弁(!))に対応しているとのこと。

同じくWORLDを使えるソフトウェアとしてはフリーウェアの歌声合成ツール「UTAU」[注15]があります。徴音梅林がUTAUと違う点は、オープンソースであること、国際化を意識して内部の文字コードはSJISではなくUTF-8を利用していること、さらにクロスプラットフォームを志向しておりLinuxでの利用も可能ということが挙げられます(徴音梅林の「林」は"Lin"uxの意味も含みます)。

もう1つの構成要素である「LINNE platform」は先ほど出てきたWindows用フリーウェアの歌声合成ツール「UTAU」をオープンソースで再実装する試みのようです。いくつかのコンポーネントで構成されますが、現状ではまだ足りない部分が多いとのこと。フロントエンド部分である「Cadencii」は.NETで記述されているものの、Windowsのネイティブコールにかなり依存している問題があり、CadenciiのJava移植版であるJcadenciiは極端に遅いという問題があります。そして何より、Upstreamでの開発が停滞してしまっているため、現在は別のソフトウェア(Fluid Vocal Synthesis system)[注16]の利用を検討しているとのことです。

おもしろい方向でのOSSプロジェクトですので、今後も情報があればまたお伝えしたいと思います。SD

---

注10) 詳しくはコラム「debian/copyrightファイルについて」を参照。現在のソースコードライセンスの記述形式の正式名称は「Machine-readable debian/copyright file」です。
URL https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
同様のものとして、Linux FoundationのSPDX(Software Package Data Exchange)があります。

注11) URL http://www.openembedded.org/wiki/OpenEmbedded-Core

注12) URL http://www.crypton.co.jp/download/pdf/eula_cv01.pdf たとえばユーザが「公序良俗に反する歌詞を含む合成音声を公開または配布すること」は認められていませんし、「映像作品での使用」は別途権利者であるクリプトン・フューチャー・メディア社に許諾を得る必要があります。

注13) URL https://github.com/mmorise/World

注14) URL http://projectmeilin.github.io/ja/

注15) URL http://utau2008.web.fc2.com/

注16) URL http://fluidvocalsynth.weebly.com/

## COLUMN debian/copyrightファイルについて

よくDebianはライセンスについて「厳格だ」「うるさい」と言われることがあります。実際にDebianに含まれているソフトウェアのライセンスについては、ソースパッケージ内のdebian/copyrightファイルで確認できます。しかし、じつは最近までは結構大雑把な記述にとどまっていました(**リスト1**)。

これはあまり良くないだろうとのことで、Debian Enhancement Proposals(DEP)の5つめの提案として挙げられたのが「Machine-readable debian/copyright」(機械的に読み取り可能なdebian/copyrightファイル)です(**リスト2**)。これに従って書くことで、プログラムでライセンスの精査が可能になります。

ただ、実直に記述するのはかなりの労力を要する(そのうえソフトウェアを使用するうえでは、ここを詳細に書いてもユーザには何も変わりがない)ので、メンテナ側もそこまで積極的に作業するにはいたっていません。ツール側でケアする動きもありますが、統一して「このツールを使おう!」という状況にはなっていません。今回のように、利用者側からフィードバックしようという話があると助かりますね。

**▼リスト1　従来のライセンス表記例(developers-referenceパッケージ)**

```
This is the Debian package of the Debian Developer's Reference.  It
was assembled by Christian Schwarz <schwarz@debian.org>, has been maintained
by Adam Di Carlo <aph@debian.org> and is now maintained by Andreas Barth
<aba@not.so.argh.org>.

------------------------------------------------------------------------------
Copyright of the Debian Developer's Reference:

Copyright c 1997, 1998 Christian Schwarz;
(..略..)
This manual is free software; you may redistribute it and/or modify it
under the terms of the GNU General Public License as published by the
Free Software Foundation; either version 2, or (at your option) any
later version.

This is distributed in the hope that it will be useful, but without
any warranty; without even the implied warranty of merchantability or
fitness for a particular purpose. See the GNU General Public License
for more details.

A copy of the GNU General Public License is available as
`/usr/share/common-licenses/GPL-2' in the Debian GNU/Linux distribution
or on the World Wide Web at <URL:https://www.gnu.org/copyleft/gpl.html>.
You can also obtain it by writing to the Free Software Foundation, Inc.,
51 Franklin St, Fifth Floor, Boston, MA 02110, USA
```

自由記述であるため、すべてのパッケージの記述がバラバラ

**▼リスト2　新しいライセンス表記例(jdパッケージ)**

```
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: jd
Source: https://osdn.jp/projects/jd4linux/releases/

Files: *
Copyright: 2006-2012, JD project
License: GPL-2

Files: debian/*
Copyright: 2006-2013, Hideki Yamane <henrich@debian.org>
License: GPL-2

License: GPL-2
 distributed under the terms of the GNU General Public License,
 version 2. On Debian GNU/Linux system you can find a copy of this license
 in `/usr/share/common-licenses/GPL-2'.
```

「Files:」で対象ファイルを、「Copyright:」で著作権者が誰かを、「License:」で適用ライセンスを明示して、適用ライセンスの内容については最後にまとめて記述を行う、という約束

# Ubuntu 16.04 LTSで使用できるUSB無線LANアダプタ7選

Ubuntu Japanese Team あわしろいくや

今回はUbuntu 16.04 LTSで動作するUSB無線LANアダプタを7つ検証したので、そのレポートです。

## USB無線LANアダプタの対応状況

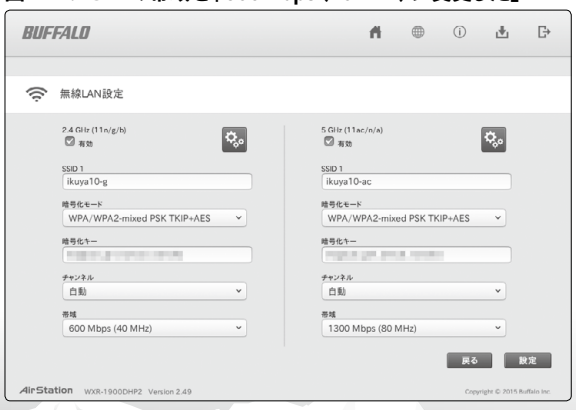
2015年5月号に掲載された、本連載第61回で『Ubuntu 14.04で使用できるUSB 無線LANアダプター7選』と題して今回と同じような記事を書きましたが、約2年も経つと入手が極めて困難になってしまったものや、カーネルのバージョンアップにより対応しなくなってしまったものもありますので、今回あらためてUSB 無線LANアダプタを入手し、検証を行うことにしました。併せて、前回筆者はIEEE 802.11acに対応したルータを所有していませんでしたが、今回、諸事情により購入したので速度の計測もしてみました。

選定した機種は**表1**のとおりで、上6つが802.11ac対応、残りが802.11n対応です。検証したのは16.04/16.04.1のカーネル4.4で、16.04.2/16.10のカーネル4.8では確認していません。

**表1　今回取り上げるUSB 無線LANアダプタ**

| メーカー | 型番 | lsusbの結果 |
|---|---|---|
| プラネックスコミュニケーションズ | GW-900D/GW-900D-BK | 2019:ab30 |
| プラネックスコミュニケーションズ | GW-450S | 2019:ab32 |
| アイ・オー・データ機器 | WN-AC867U | 04bb:0952 |
| アイ・オー・データ機器 | WN-AC433UA | 04bb:0953 |
| バッファロー | WI-U2-433DM | 0411:0242 |
| エレコム | WDC-433DU2H | 056e:4007 |
| アイ・オー・データ機器 | WN-G300UA | 0bda:8178 |

**図1　2.4GHzの帯域を「600Mbps（40MHz）に変更した」**




## 使用するルータ

今回検証に使用したルータは、バッファローのWXR-1900DHP2です。使うかどうかはさておき、いろいろな機能があるのと、802.11nでも600Mbpsの速度が出るので選択しましたが、残念ながら無線LANアダプタ自体が300Mbpsまでしか対応しないものしか購入できなかったため、今回は試せませんでした。ルータもデフォルトでは使用する帯域が20MHzになっており、これだと300Mbpsまで出ません。よって40MHzに変更しています（**図1**）。

## ドライバーのインストール

前述のとおり今回別途ドライバ（カーネルモジュール）の必要なモデルが6つありますが、インストールする方法はすべて同じです。**図2**のコマンドを実行してください。

すべてエラーが出ていないことを確認し、無線LANアダプタを挿入してください。

とくに問題なく動作することを確認したら、次のコマンドを実行してください。これでカーネルのアップデートごとにカーネルモジュールの再インストールが不要になります。

```
$ sudo make uninstall
$ sudo make -f Makefile.dkms install
```

## GW-900D/GW-900D-BK

第61回(2015年5月号)でも紹介したGW-900Dは終息で、現行モデルはGW-900D-BKです。プラネックスコミュニケーションズの開発者向け情報[注1]によると内部的には同じものとのことです。USB 3.0モデルとしては価格も安めですのでうれしいです。ドライバのインストールが必要です。

iwconfigの結果は**図3**のとおりです。

注1) https://www.planex.co.jp/support/taiou/kisyu/developer_wifiusb.shtml

## GW-450S

これも第61回で使用しました。ドライバのインストールが必要です。5GHzにしか対応していないのは注意点ですが、価格も安くて小さいのはうれしいところです。

iwconfigの結果は**図4**のとおりです。

## WN-AC867U

USB 3.0で比較したかったので購入しました。ドライバのインストールが必要です。正直なところかなり大きいので、扱いはけっこう難しいです。

iwconfigの結果は**図5**のとおりです。

## WN-AC433UA

アンテナが長いモデルは、802.11nで使用する場合に有効ではないかと思います。ドライバのインストールが必要です。

iwconfigの結果は**図6**のとおりです。

**図2　別途ドライバインストールの例**

```
$ sudo apt install git dkms
$ git clone https://github.com/abperiasamy/rtl8812AU_8821AU_linux.git
$ cd rtl8812AU_8821AU_linux/
$ make
$ sudo make install
$ sudo modprobe rtl8812au
```

**図3　GW-900D-BKのiwconfigの結果**

```
wlx0022cfe65987  IEEE 802.11AC  ESSID:"（略）"  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency:5.64 GHz  Access Point: （略）
          Bit Rate:867 Mb/s   Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=100/100  Signal level=100/100  Noise level=0/100
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

**図4　GW-450Sのiwconfigの結果**

```
wlx0022cff2744f  IEEE 802.11AC  ESSID:"（略）"  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency:5.64 GHz  Access Point: （略）
          Bit Rate:434 Mb/s   Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=98/100  Signal level=100/100  Noise level=0/100
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

## WI-U2-433DM

コンパクトで802.11nにも対応していていいですが、若干割高感があります。ドライバのインストールが必要です。

iwconfigの結果は**図7**のとおりです。

## WDC-433DU2H

これもアンテナが長いモデルで、エレコム製のものもあったほうがいいかと思って選択しました。ドライバのインストールが必要です。

iwconfigの結果は**図8**のとおりです。

## WN-G300UA

**表1**を今一度見ていただきたいのですが、このモデルのベンダIDはアイ・オー・データ機器(04bb)ではありません。調べてみたところどうもチップベンダ(Realtek)そのままですので、ドライバのインストールが不要です。とにかく簡単につなぎたいという場合にはお勧めです。とはいえ速度が安定せず、なぜかビットレートも300Mb/sになりません。

iwconfigの結果は**図9**のとおりです。

**図5　WN-AC867Uのiwconfigの結果**

```
wlx3476c56c028d  IEEE 802.11AC  ESSID:"(略)"  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency:5.64 GHz  Access Point: (略)
          Bit Rate:867 Mb/s   Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=100/100  Signal level=100/100  Noise level=0/100
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

**図6　WN-AC433UAのiwconfigの結果**

```
wlx3476c52db4e4  IEEE 802.11AC  ESSID:"(略)"  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency:5.64 GHz  Access Point: (略)
          Bit Rate:434 Mb/s   Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=100/100  Signal level=100/100  Noise level=0/100
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

**図7　WI-U2-433DMのiwconfigの結果**

```
wlx8857ee94524d  IEEE 802.11AC  ESSID:"(略)"  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency:5.64 GHz  Access Point: (略)
          Bit Rate:434 Mb/s   Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=98/100  Signal level=100/100  Noise level=0/100
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

**図8　WDC-433DU2Hのiwconfigの結果**

```
wlxbc5c4c57310a  IEEE 802.11AC  ESSID:"(略)"  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency:5.64 GHz  Access Point: (略)
          Bit Rate:434 Mb/s   Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=100/100  Signal level=100/100  Noise level=0/100
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

## ベンチマーク

ベンチマークについてはいろいろと前提の説明が必要です。場所は筆者の自室で電波暗室などではないため、電波干渉などがあり、安定していないことが考えられます。とはいえ直線距離で2メートル程度しか離れていないので、比較的良好な条件であるとは思われます。同じような速度にならない場合は、筆者はウサギ小屋に住んでいるのだなと思っていただければ幸いです。

送受信それぞれを計測しましたが、scpコマンドを使用してUbuntu 16.10のisoイメージ(ubuntu-16.10-desktop-amd64.iso)を送信しました(**表2**)。約1.5 GBです。

計測は原則として3回行って最も速いものにしました。802.11nはタイミングによってばらつくこともありましたが、電波干渉を受けやすいという性質上、やむを得ないと思います。ただしWN-G300UAは例外で、場合によっては1桁違っていました。あくまで参考値で、この値はほぼ出ないと思っていただいていいです。

## 考察

今回はすべてRealtek製のチップが採用されているもののみを検証しました。MediaTek製のチップを採用しているモデルも市場にたくさん出回っていますが、今回少し試した限りでは速度がまったく出ないとか、16.04のカーネル(4.4)ではなんとか動作するものの16.10のカーネル(4.8)では動作しないとか、容易に解決できそうにない大きな問題があったため、見送っています。WI-U3-866D[注2]は認識したもののアクセスポイントの検出ができなかったので外しています。

新規に購入する場合は、用途をよく考えるのがいいでしょう。とにかく速いほうがいいという場合はUSB 3.0のモデルになりますが、いつも大きなファイルをやりとりするわけではないでしょうし、性能と価格のバランスのいいUSB 2.0モデルが妥当ではないでしょうか。5GHz帯でしか使用しないのであればGW-450Sがお勧めですが、やはり2.4GHz帯も使用したいというのであればWDC-433DU2Hがお勧めです。とはいえ若干割高感はありますが。

WN-G300UAはお勧めではありませんが、価格も安いですし、いざというときのために持っておくのは悪くないでしょう。SD

**図9　WN-G300UAのiwconfigの結果**

```
wlx3476c541afc5  IEEE 802.11bgn  ESSID:"(略)"
          Mode:Managed  Frequency:2.437 GHz  Access Point: (略)
          Bit Rate=150 Mb/s   Tx-Power=20 dBm
          Retry short limit:7   RTS thr=2347 B   Fragment thr:off
          Power Management:off
          Link Quality=70/70  Signal level=-27 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:5   Missed beacon:0
```

**表2　各モデルのベンチマーク結果**

| 型番 | 周波数 | 送信 | 受信 | 備考 |
|---|---|---|---|---|
| GW-900D | 5GHz | 36.2MB/s | 23.0MB/s | USB 3.0接続 |
| GW-900D | 2.4GHz | 20.8MB/s | 22.0MB/s | USB 3.0接続 |
| GW-900D | 5GHz | 35.4MB/s | 22.7MB/s | USB 2.0接続 |
| GW-900D | 2.4GHz | 20.0MB/s | 18.8MB/s | USB 2.0接続 |
| GW-450S | 5GHz | 28.7MB/s | 22.4MB/s | 5GHzのみ対応 |
| WN-AC867U | 5GHz | 34.6MB/s | 22.7MB/s | USB 3.0接続 |
| WN-AC867U | 2.4GHz | 20.8MB/s | 20.3MB/s | USB 3.0接続 |
| WN-AC867U | 5GHz | 35.4MB/s | 22.4MB/s | USB 2.0接続 |
| WN-AC867U | 2.4GHz | 20.5MB/s | 21.7MB/s | USB 2.0接続 |
| WN-AC433UA | 5GHz | 28.7MB/s | 22.4MB/s | |
| WN-AC433UA | 2.4GHz | 11.1MB/s | 11.6MB/s | |
| WI-U2-433DM | 5GHz | 28.7MB/s | 22.4MB/s | |
| WI-U2-433DM | 2.4GHz | 11.3MB/s | 12.4MB/s | |
| WDC-433DU2H | 5GHz | 28.2MB/s | 22.7MB/s | |
| WDC-433DU2H | 2.4GHz | 11.5MB/s | 11.7MB/s | |
| WN-G300UA | 2.4GHz | 6.0MB/s | 4.9MB/s | 2.4GHzのみ対応 |

注2) http://buffalo.jp/product/wireless-lan/client/wi-u3-866d/

# 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

システム管理者の視点でファイルシステムを眺めます。データ圧縮や、ファイルシステムの木構造をうまく活用する方法を見ていきます。

## 第10回 ファイルシステムについてもう少し深く

システム管理者はもちろん、ソフトウェアエンジニアにもある、システム停止による緊急の深夜呼び出し。原因の調査や問題の解決をするには、何をしなければいけないのでしょうか。今回は、さまざまな障害のうち、ファイルシステムの問題に対処できるようになることを目指しましょう。

ファイルシステムが木構造であること、コピー、削除、移動、リンクの仕方を知っただけで、Unix上のオペレーションが自在にできるわけではありません。バックアップの仕方、使っているディスクの空き状況を確認すること、使っているファイルシステムの制限・特性の理解、ディスクが不足になった場合の対応方法や、ファイル・システム中のファイルやディレクトリを一括で処理する方法の修得を目指してください。

### ファイルシステムの状態

まずは、ディスクの利用量などファイルシステムの状態の確認方法を見ていきます。

#### du – Disk Usage – ディレクトリ内の使用量を調べる

duは、ディレクトリ階層を再帰的にたどり、使用量を集計するコマンドです。オプションを指定しなければ、カレントディレクトリから、それぞれのディレクトリのサイズを集計した情報を列挙します。単位はブロック(Linuxでは1,024byte、macOSでは512byte)で、1ブロックのサイズは環境変数やオプションで変更できます。

duの実行例(macOS)
```
$ du↵
145424  ./bin
8   ./Caskroom/identify/.↵
metadata/551/20161107035636.943/Casks
8   ./Caskroom/identify/.↵
metadata/551/20161107035636.943
0   ./Caskroom/identify/551
...略...
0   ./var/run/dbus
0   ./var/run
4632    ./var
15240728    .
```

各ディレクトリの合計だけわかればいいときは、-sオプションを指定します。

duの実行例-sオプション付き(Linux)
```
# du -s /usr↵
10488368    /usr

# du -s /usr/*↵
295504  /usr/bin
2528    /usr/games
172832  /usr/include
3270800 /usr/lib
3807056 /usr/local
20  /usr/locale
18240   /usr/sbin
2061324 /usr/share
860060  /usr/src
```

この例では、/usrの下をアクセスしました

ので、念のためroot(スーパーユーザ)権限で実行しています。アクセスできないところがあると、下記のようなエラーメッセージがでます。エラーがあっても、コマンドは中断しないで読めなかった部分をスキップします。

duでパーミッション(権限がないディレクトリを参照しようとした場合のメッセージ例)

```
du: cannot read directory './var/backups/Remote/nakajim/fml/spool/ml/asgs0/tmp': Permission denied
```

-k、-mを付けるとKB、MBをブロック単位として集計してくれます(さらにLinuxでは、-Bオプションによって、ブロックサイズを明示的に指定できます)。

### 人間に読めるよう出力してくれ

人は、桁が多い場合カンマ(,)で区切るか、KB、MB、GB、TBなど、ある程度桁を丸めてもらえないと理解に時間がかかります。du、ls -l、この後に紹介するdfコマンドなどは、サイズを表示する際のデフォルト単位は、Byteやブロックだったりで、カンマによる桁の区切りもありません。コマンドの出力情報をプログラムが処理する場合は、正確であること、無駄がないほうが便利です。

du、ls、dfなどは、人が読みやすい形で表示するオプション-hが用意されています。

```
$ du -sh
148K    .
$ ls -lh

total 240
-rw-r--r--@ 1 masa  staff    97K  6  6 15:48 fig10.pptx
-rw-r--r--  1 masa  staff   324B 11 19 15:41 out_of_10.txt
-rw-r--r--@ 1 masa  staff    15K 11 23 19:01 unix_command_explorer_10.txt
```

Unixの世界は、コマンドやプロセスがうまく連携できることを重視しますので、コンピュータ優先(Computer First)です。この世界では、あなたがコンピュータでないことを明示的に宣言「-hオプション」しましょう。

-aオプションを指定すれば、個々のファイルサイズも列挙します。

```
$ du -a
```

duで、ディレクトリ同士を比較してどこが多く領域を取っているか確認できます。ログファイルが無用に増えてしまったなどの状況が一目で確認できるので便利です。しかし、ファイルの数が多い、ディレクトリ階層が深いところにduをかけると時間がかかります。たとえばルート(/)から全部duをかけるなどという大胆な構想はいけません。容量が動的に増える可能性があるのは、ログとデータベースの領域だから/var/log、/var/lib/mysql、Webアプリケーションディレクトリ/siteの下など、的を絞ってduを使いましょう。

-dオプションで表示するディレクトリの深さは指定できますが、使っている領域はすべてスキャンして合計します。サブディレクトリの数が多いときには、有効なオプションです。duの結果を**sort -nr**にパイプで結果を渡してやれば、消費量の多いディレクトリから並べ替えることができます。

### STEP UP! ls -lとディレクトリのサイズ

ls -lで表示される5番目のフィールドの「サイズ」、ファイルの場合はまさにサイズです。しかし、ディレクトリのときは何なのでしょう。ディレクトリ内にあるファイルの合計でしょうか。違います。duの結果と比較してみてください。ディレクトリの場合は、ファイルシステム中で、そのディレクトリのメタ[注1]情報(ディレクトリ内にどんなファイルやディレクトリがどのくらい入っているのかなど=ディレクトリエントリというデータ構造)が使っている領域です。この領域がどのように増減するかは、Linuxの(extX)やmac OSなど、ファイルシステムによって異なります。

## df – Disk Free – ファイルシステムの状態を見る

ファイルシステムごとに、どの程度の容量があって、どのくらい使われているかを確認するには、**df**コマンドを使います(**図1**)。

表形式で、ファイルシステムごとに1行ずつ情報が出力されます。**図1**のように、macOS

注1) 「メタ~」は、情報システムの世界でよく使われる接頭語です。扱う対象と上位の(対象を管理上する、対象を定義するなど)ものを区別するときに使います。例)メタデータ、メタ言語。

**▼図1　dfの実行例（macOS）**

```
$ df
Filesystem    512-blocks      Used Available Capacity   iused     ifree %iused  Mounted on
/dev/disk1    1872985888 899070744 973403144    49% 112447841 121675393   48%   /
devfs               366       366         0   100%       634         0  100%   /dev
map -hosts            0         0         0   100%         0         0  100%   /net
map auto_home         0         0         0   100%         0         0  100%   /home
/dev/disk5     16674816   6427752  10247064    39%    803467   1280883   39%   /Volumes/Secured
```

などのBSD系Unixのdfは、オプションを指定しなくてもディスク容量や使用状況に加えて、i-nodeの利用状況も表示します。

### i-node

Unixのファイルは、それぞれ1つのi-nodeと呼ばれる管理情報に対応付けられているということを第5回（本誌2016年9月号）で解説しましたが、この機会にもう少し詳しく理解しておきましょう。i-nodeには、次のような情報が保存されています。

・ファイルの所有者・グループ
・アクセス権限
・ファイルの大きさ
・ファイルシステム中のデータの位置
・制作、最終使用、最終変更日時
・リンクカウント
・種別

プログラムは、i-nodeをインデックスとして実データが保存されている領域にたどり着くことができます。i-nodeは、1つのファイルシステム（ディスクパーティション）中では、必ずユニークになるように番号（ID）が振られています。逆に、異なるファイルシステム（ディスクパーティション）では、i-node番号のユニーク性は保証されません。

Linuxでは、デフォルトではi-node情報が表示されません。i-nodeの状態を確認したければ-iオプションを指定します（**図2**、**表1**）。

dfもオプションで、-m 1block=1MB、-k 1block=1KBなどブロックサイズを指定できます。

### STEP UP! i-node領域

通常ファイルシステムを作るときは、システムのデフォルトでi-node数が確保されます。しかし、ファイルシステムの活用のされ方、「たとえば、a）小さいファイルをたくさん作る場合、b）大きなファイルを少数作る場合」によっては、a）の場合i-node領域が足りなくなってしまう、b）の場合無駄にi-node領域がディスクを専有してしまっている、という状況になります。ディスク容量がシステム運用に十分ある場合は、あまり気にすることはありませんが、際どい計画を求められているときには、ファイルシステムを作るときに、i-node領域を指定するようにしましょう。

**▼表1　dfで表示されている情報の意味**

| フィールド名 | 意味 |
|---|---|
| XXX-blocks | 利用可能領域（ブロック数、1ブロックのサイズはXXX） |
| Used | 利用済み領域 |
| Available | 残り利用可能領域 |
| Capacity/Use% | 利用されている領域% |
| ifree/IFree | 利用可能i-node領域 |
| %iused/IUse% | 利用されているi-node領域% |
| Mounted on | どのディレクトリにマウントされているファイルシステムか |

**▼図2　-iオプションを付けたdfの実行例（Ubuntu）**

```
$ df -i
masahiro@ubuntu0:~$ df -i
Filesystem      Inodes  IUsed   IFree IUse% Mounted on
udev           1007464    531 1006933    1% /dev
tmpfs          1011787    788 1010999    1% /run
/dev/sda2      7266304 360572 6905732    5% /
tmpfs          1011787      8 1011779    1% /dev/shm
tmpfs          1011787      6 1011781    1% /run/lock
tmpfs          1011787     17 1011770    1% /sys/fs/cgroup
/dev/sda1            0      0       0     - /boot/efi
cgmfs          1011787     13 1011774    1% /run/cgmanager/fs
tmpfs          1011787     36 1011751    1% /run/user/1000
/dev/mmcblk0p1 3899392  59878 3839514    2% /media/masa/EXT4XCI64G
```

## STEP UP! ファイルシステムのマウント

HDDやSSDなど、物理ドライブは分割して使うことができるのはご存じでしょうか。分割したかたまりをディスクパーティションと呼びますが、通常1つのディスクパーティションには1つのファイルシステムを作って使います。

ディスク(パーティション)は、ファイルシステムを作ってから、適当なディレクトリの下にマウントmountできます。マウントすることで、複数のファイルシステムを横断して、まるで1つのドライブのように使えます(図3)。

### mount – MOUNT – ファイルシステムをマウントする

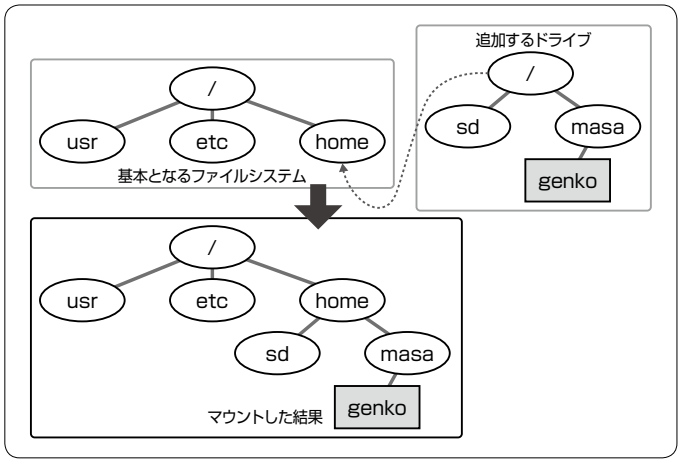
ファイルシステムをマウントするには、mountコマンドを使います。USBドライブなどのプラグ&プレイできるドライブは、コンピュータに接続するだけで自動でマウント(automount)されることもあります。現在マウントされているデバイスを確認する場合にも、mountコマンドを使います。

mountされているファイルシステムの確認する例
```
$ mount
/dev/disk1 on / (hfs, local, journaled)
devfs on /dev (devfs, local, nobrowse)
map -hosts on /net (autofs, nosuid,
automounted, nobrowse)
map auto_home on /home (autofs,
automounted, nobrowse)
/dev/disk5 on /Volumes/Secured (hfs,
local, nodev, nosuid, journaled, noowners,
mounted by masa)
```

出力結果は、onの前のデバイスが、その後ろのディレクトリにマウントされていることを示しています。()括弧内は、マウント時のオプション、どのようなフォーマットのファイルシステムかなどが表示されています。

macOSでは、mountが実装されていますが、独自のツールdiskutilを使います。

▼図3 ファイルシステムのマウント




## 圧縮したままファイルを操作する

ディスク容量に限りがある場合、ファイルを圧縮するなどして効率的にファイルシステムを使わなければなりません。ログなどテキストデータは圧縮による効果が大きく、容量を削減することができます。

### gzip、bzip2

Unixには、ファイルの圧縮ツールも用意されていてgzip、bzip2がよく使われます。違いは、圧縮アルゴリズムで、圧縮効率、圧縮速度です。gzipのほうが一般に圧縮率は低いのですが、高速だと考えて良いでしょう。使うときには、圧縮ファイルをほかのシステムで使うこと、コンパチビリティなどを考慮して選択します。

圧縮は、元のファイルはそのままで、圧縮ファイルを作りながら進行するので、残り領域が極端に少ない状態まで放置していると圧縮できないことがあります。早めにアクションをとりましょう。

### 圧縮したまま、ファイルを操作

圧縮したままファイルを操作できるコマンドには、zmore、zless、bzmore、bzless、zcat、bzcat、zdiff、bzdiff、zcmp、bzcmp、zfgrep、zegrep、bzfgrep、bzegrepなどがあります。ページャーをはじめ、ほとんど必要なものはそろっています。これまでの連載でも出てきたコマンド名の前に、圧縮形式に対応した接頭語gzip=z、bzip2=bzが付いた名前になっています。

## 木構造をまとめて操作する

### tar – Tape ARchiver – アーカイブする

テープデバイスが活躍していた古(いにしえ)の時代から存在しているコマンドで、今も頻繁に使われているのがtar(ターと発音)です。tarは、木構造のまま、ファイルとディレクトリを1つのファイルにまとめ(アーカイブし)て管理します。tarでまとめたファイルは、tarball(ターボール)と呼びます。拡張子は慣用的に.tarを使います。

tarのオプションは、-に続かないものもあります。cはアーカイブを作る、tはアーカイブの中身を確認する、xはアーカイブから取り出す、です。vオプションを一緒に指定すると、詳細なメッセージが出ます。fオプションで、ターゲットのtarballを指定します。

次のコマンドは、ディレクトリ.以下をアーカイブして、ファイル"mytarball.tar"に出力する例です。

```
アーカイブを作る基本形：カレントディレクトリ以下をmytarball.tarにアーカイブする
$ tar cvf mytarball.tar .
```

tarでアーカイブする対象のディレクトリは、絶対パス指定(/xxxというように、/で始まる形式)しないようにしましょう。古いバージョンのtarには、アーカイブファイル内に本当に/からのパスが付けられて、展開するときに望んだ出力先と違う所に展開されてしまうことがありました。このようなシステムも残っているかもしれませんが、新しいtarは絶対指定した場合は、アーカイブ内のファイルパスはすべて先頭の/を取り除いてアーカイブしてくれます。それでも、無用なパスが長々とついているのは、展開したときに、とても深いディレクトリを作ってしまったりと良いことはありませんので、パス指定は適切に行うようにしましょう。

```
$ tar cvf tmp.tar /tmp
tar: Removing leading `/' from member names
/tmp
```

```
アーカイブの内容を確認する基本形
$ tar tvf mytarball.tar
```

次のコマンドは、mytarball.tarの内容をカレントディレクトリに取り出します。

```
アーカイブからファイルを取り出す基本形：カレントディレクトリからの相対パスで、mytarball.tarに記録されている情報を取り出す
$ tar xvf mytarball.tar
```

tarを使うと、オーナー・パーミッション情報を保存したままコピーすることもできますし、別のシステムへの移動も容易です。次の例では、-Cオプションを使って、出力先のディレクトリを、デフォルトのカレントディレクトリから変更しています。

```
tar | tarを使って、カレントディレクトリ以下を、属性そのまま/home/taroにコピーする
$ tar cf - | tar xf - -C /home/taro
```

デフォルトでは、tarはファイルを圧縮しませんが、圧縮形式を扱うこともできます。zオプションならgzip形式、jオプションならbzip2形式です。

```
tarと圧縮の組み合わせ（アーカイブの内容の確認）
$ tar tvzf mytarball.tar.gz
$ tar tjvf mytarball.tar.bz2
```

ファイル名には、アルゴリズムに応じて、.tar.gz、.tgz、.tar.bz2、.tbzなどをサフィックス(接尾辞。ここではファイル拡張子の意)として付けるのが習慣です。

圧縮アルゴリズム(圧縮していない場合も)に合ったオプションを指定しないと、アーカイブの内容を確認したり展開したりできないので注意しましょう。たまに、上記のネーミングコンベンションに従わなかったり、間違ったサフィックスが付いていることもありますので、おかし

いと思ったら、fileコマンドでファイルの内容を確認してみましょう。

```
$ file abc.tar
```

### find – FIND – 木構造に従ってファイルを見つける・操作する

先に紹介したdu -aを使えば、特定のディレクトリ以下のファイルを再帰的にファイルやディレクトリサイズを集計します。findコマンドは、木構造に従ってファイルやディレクトリを列挙して、さらにそれらに対して指示を送ることができます。

findのオプションは、Linuxでは新しいスタイルで、macOSは従来からのスタイルです。大きな違いは、検索を開始するディレクトリを省略できるのがLinuxで、macOSでは省略できません。オプションはたくさんありますが、従来からのスタイルで記述する習慣をつけておけば良いでしょう。

最も使うのは、-printオプションでしょう。見つかったファイルやディレクトリ名を標準出力に出力します。

findの例
```
$ find . -print
.
./.gitignore
./.git
./.git/description
./.git/FETCH_HEAD
./.git/config
./.git/objects
./.git/objects/38
... 中略
./rogue5.4.4-ant-r1.1.2/pack.c
./rogue5.4.4-ant-r1.1.2/rooms.o
./rogue5.4.4-ant-r1.1.2/save.o
./r-o-m
```

.docで終わるファイル名のものだけ探してみます。*によるグロッビングを使っていますが、findの前にシェルにグロッビングを展開されてしまわないように\で*をエスケープします。

findの例2
```
$ find . -name \*.doc -print
./rogue5.4.4-ant-r1.1.2/rogue.doc
```

findなら、ディレクトリ階層以下、特定のファイルだけを削除できます。次の例では、-execオプションで、rmを実行します。rmの引数に、見つかったファイル名(これは記号{}に収納される)を指定し、コマンドの区切りである;を\でシェルからエスケープしています。ちょっと記号が多くて混乱するかもしれませんが、イデオムとして覚えておくと便利です。

find -execの例
```
$ find . -name \*.doc -exec rm {} \;
```

-typeを指定すると、特定の属性のものだけを抜き出すことができます。

find -typeでディレクトリだけを表示する例
```
$ find . -type d  -print
.
./.git
./.git/objects
...略...
./rogomatic-r2.0.2/src/.deps
./rogue5.4.4-ant-r1.1.2
```

ファイル操作において、とにかくfindは強力です。確実にマスターしましょう。

## 今回のまとめと次回について

今回は、ファイルシステムの状態の確認、圧縮操作、木構造をたどってのファイルの操作を紹介しました。次回は、テキスト処理(その2)の予定です。SD

### 今回の確認コマンド

【manで調べるもの
(括弧内はセクション番号)】
du, df, mount, zmore, zless, bzmore, bzless, zcat, bzcat, zdiff, bzdiff, zcmp, bzcmp, zfgrep, zegrep, bzfgrep, bzegrep, tar, file, find, diskutil(macOSのみ)

February 2017

NO.64

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

# もはや生活の一部!? ベテランのコミュニティ運営術

今回は、2016年11月に大阪で行った研究会の模様をお伝えします。

## jus研究会 大阪大会

**■ITコミュニティの運営を考える**
**【講師】たなかとしひさ（LILO／OSM関西）、やまぐちあゆみ（子供とネットを考える会）、榎 真治（jus／LibreOffice日本語チーム）、法林 浩之（jus）**
**【日時】2016年11月12日（土）15:00～15:50**
**【会場】大阪南港 ATC ITM棟10階**

関西のLinuxユーザズグループ「LILO（Linux Install Learning Osaka）」や、OpenStreetMapのコミュニティ「OSM関西」で活動するたなかさん、「子どもとネットを考える会」で活動するやまぐちさんという、関西のITコミュニティで長く活動されている2人をお迎えして、関西オープンフォーラム（KOF）の中で開催しました。参加者は14人でした。

パネリストからそれぞれITコミュニティ運営で困っているお題を1つずつ出してもらい、みんなでそれに回答していくスタイルで進めました。

### ■コミュニティ運営は生活の一部？

最初のお題は、たなかさんの「運営感を持てませんが、これでいいでしょうか？」でした。コミュニティ活動は顔を洗うように生活の一部とのことです。

やまぐちさんも、コミュニティ活動が日常（子育て）に直結していることもあり、常に「これ、家で使えそう、人に紹介できそう」という意識で物事を見ており、やはり運営というより生活の一部だそうです。法林さんも同様とのこと。筆者（榎）は、LibreOfficeでは少し運営感があると話したところ、意識高い系だと言われてしまいました。

### ■一般人にリーチするには

2つめのお題は、やまぐちさんの「啓発活動なので一般の保護者にリーチしたいが、どうすれば良いか？」です。学校でチラシを配布してもらっても、男の子はチラシをカバンから出さないのだそうです。

たなかさんはLILOでもLinux普及に3年はかかったと言います。OSM関西でも2年間はほぼ宣伝活動しかしていなかったそうです。当時は、売名と言われても看板を出すことをやっていました。

筆者からはLibreOfficeは一般の人に使ってほしいのにギークの人しか知らないこと、LibreOfficeのイタリアコミュニティではマーケティングの人が引っ張っていて、ボランティアを集める際にはマーケティングの人も募っていたことを紹介しました。意図的にそちら方面の人を集めるのもありではという議論になりました。

そのほかに、LibreOfficeからはサッカー場でブースを出す試みをしたという話があり、やまぐちさんからも図書館でのインターネットお悩み相談をした経験の紹介があり、外に出て行くことも大事ではないかとの議論にもなりました。

### ■コミュニティは惰性で良いのか

筆者から3つめのお題として「コミュニティが惰

性になったときにどうするのか？」を伺いました。

たなかさんによるとLILOは惰性で良くて、終着点としては30年くらいたったある日にサーバがダウンして解散だそうです。「まだやっていたんだ」というくらいがちょうど良いと言います。

やまぐちさんは、まっちゃ139勉強会で宴会担当を10年ほどやっており、立ち上げ当初は年2、3回開催していたのが、今は年1回程度だそうです。メンバーがほかのコミュニティをやったり別の方向を向いたりしているので、これで良いかなとのことです。子供とネットを考える会は「世の中に良い資料があるのに知られていないのがもったいない、知らせたい」との思いが活動のベースにあるので、子供が成長しても続けられそうという話もありました。

法林さんからは、jusではインターネットが流行りだしたときなど、何度か新しい方向に舵を切ったことが紹介されました。「やっている人がおもしろいと思ったから、そちらに進んだのでは？」とのこと。最近10年くらいは年中行事化しており惰性感は強くなってきていますが、大きいイベントということもあって、やった感はあるとのことです。

### ■南港は遠い？

法林さんから4つめのお題として「KOFの会場として南港は遠いが、どこでやったらいいですか？」が出されました。

たなかさんからは「遠いから行けないということで消滅したコミュニティも多く、コミュニティの継続的活動をするうえではどこに地域基盤を持つかは大事」という指摘がありました。LILOは関西全体であり、活動が活発なころは各地を回っていたそうです。広いので1ヵ所にしてはいけないという意識だったそうです。KOFの規模になると転戦はしないほうが良いだろうという議論もありました。

また、子供を連れて来られるので南港は便利という話や、やまぐちさんから「会場の開放感や、一般のお客さんにふらっと参加してもらいやすいという点で南港はいいのでは」という話がありました。

「OSC東京やデブサミ関西に比べると便利」、「コミュニティイベントはお金がないので、不便なところでやっていることも多い」との意見もありました。

### ■遠方から参加してもらうには

会場からは、「エリア外からの参加はどう思いますか？」というお題をいただきました。「ほかのエリアから参加いただけるのはありがたい、ただ遠方からの参加はたいへんではないか？」という話になったのですが、広島からの参加者によると、この会場は「広島から（新幹線で来て）新大阪で降りればさほど気にならない」とのこと。子供とネットを考える会は啓発について一緒に考える会がほかにあまりないらしく、エリア外からの参加者が多いそうです。

会場からの2つめのお題として、「遠隔で参加したい」という話がありました。筆者からはLibreOfficeでも中継はやりたいが、トラブルが起こりがちで張りついて担当できる人がいないと厳しいこと、人が確保できていないことを紹介しました。

長らくKOF中継班を担当されていた、たなかさんによると、KOFでは1年目からステージを中継しており、それは来られない人に雰囲気を味わってもらうためにやっているとのことです。初めてLinux Conference '98で中継したときには、回線がパンクするほどアクセスがあったそうで、会場に行けない人は情報に飢えているのだろうとのことです。

やまぐちさんからは「情報モラルやセキュリティのイベントでは、その場で信頼できる人と直接話をしたいというニーズがあるので、内容を選ぶ必要がある」との指摘がありました。セキュリティでディープな勉強会は意図的に配信していないそうです。

### ■終わりに

ゲストの2人は長くコミュニティで活動されており、時間軸を見据えた話は印象に残りました。また、いつにも増して笑いが多い研究会でした。

動画をWeb[注1]で公開しているので、詳しい内容に興味を持たれた方はご覧ください。SD

注1） URL https://www.youtube.com/watch?v=z9ww4TouVVI

# 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
関 治之　Hal Seki
Twitter @hal_sk

# 第62回 シビックテック祭り「Code for Japan Summit 2016」

Code for Japanの関です。今回は2016年11月18〜20日にかけて開催された、シビックテックのお祭り「Code for Japan Summit 2016」についてレポートをしたいと思います。

自分たちが住む地域の課題をテクノロジーを用いて解決する活動（シビックテック）は、海外だけでなく日本各地でも行われています。Code for Japan Summit（以下サミット）とは、シビックテックの普及啓発のために、Code for Japanのメンバーや行政職員、各地で活動しているシビックテック関係者が一緒になってシビックテックの活動を紹介し、悩みを共有し、相互に学ぶための場です。

今年は、横浜市金沢区さんの協力のもと、横浜市金沢区総合区役所にてサミットを開催しました。全国からエンジニア、デザイナー、行政職員などシビックテックにかかわる多くの方々が参加して、活発な議論が行われました。今回のサミットのテーマに掲げた「Voyage（出航）」にふさわしく、新たな仲間との出会いや新たな活動のスタートの契機となるイベントになったと考えています。会場の装飾も、船出というキーワードにからめてこだわっています（**写真1**）。

◆**写真1　VOYAGEにからめて、いろいろな仕掛けをしてみました**


参加者は3日間の延べ人数で約650人。正確には把握できていませんが、ユニークにすると400人ほどが参加いただいたと思います。関連ツイートは、450以上にものぼりました。

## サミット1日目

1日目は、金沢区長もお呼びして、「シビックテックと地域協働のネクストステップ」というパネルディスカッションを行いました。

区長に金沢区の地域協働に関する取り組みをうかがった後、この連載でもおなじみの、Hack For Japanの及川卓也さんに登壇してもらい、シビックテックが次のステップに進むにはどういったことが必要かをお話いただきました[注1]。

及川さんからは、ご自身もかかわっていたSpending.jpのプロジェクトをケースにして、ビジネスとして考えたときに必要となる価値設定や、人に届けるために必要なこと、プロダクトマネジメントの必要性を伝えてもらいました。

その後、Code for Kanazawaの福島さんも参加し、どのように地域の人達とつながって、継続的に活動をしていくかを話し合いました。印象的だったのは、Code for KanazawaではNPOや地域の人達にも同団体に入ってもらって、一緒にプロジェクトを起こしていっているという点でした。そうしないと「Code for Kanazawaに頼めば何かやってくれる」という感じでお客さん立場になってしまうことで、活動が続かなくなることが多いとのことです。

注1　スライドURL：https://t.co/0Kc0qymV0h

その後は会場を分けて、分科会に。さまざまなセッションが行われましたが、筆者が参加していたセッションを中心にご紹介します。

## インターナショナルセッション

Code for AmericaのMonique Baena-Tenさんや、台湾のシビックハッカーであるT.H. Scheeさんが来日していたので、英語のみのセッションを企画しました。人数が20名くらいだったので、車座になって皆でディスカッションを行いました(**写真2**)。

アメリカ、台湾でのシビックテック事情をうかがい、Code for Ibarakiの柴田重臣さんからも日本の各地の状況をシェアしました。アメリカの大統領選の影響についてもリアルなところを聞けて、とても勉強になりました。THさんは、"Civic Tech is like a ocean"と表現しましたが、シビックテックを進めていくことは、荒波を航海するようなもの。良いときもあれば、悪いときもあります。そんなときに、このように各地の活動をシェアし、世界中で頑張っている人達がいることを知ることは、とても良い機会だと思いました[注2]。

## ブリゲイド相談会

Code for Japanと連携しながら各地で課題解決を行っているCode forコミュニティを、「ブリゲイド」と呼びます。このセッションでは、ブリゲイド同士が集まって相互に悩みを相談する相互相談会を行いました。活動内容の見える化や、より良いコミュニティ運営のやり方について、多くの意見が出ていました。筆者の参加したテーブルでは、各地の情報を共有するハッシュタグを作り、Code for Japan側でそれらを拾ってニュースレターを定期的に配信するなどのアイデアが出ていました。実施を検討したいと思います。

## Code forフェロー大報告会

Code for Japanでは、企業と自治体間の組織の壁を壊して、ともに考える人材づくりを行う目的で、企業の従業員が自治体の職員として役所内で3ヵ月間働く、人材育成プログラムを行っています。これまで、8自治体に18人のフェローが派遣されています。実際に派遣を体験した4名が参加して、パネルディスカッションを行いました。

◆**写真2　Code for AmericaのMoniqueさんも参加**


セッションでは、奥野和弘さんから「鯖江に行ってびっくりしたのが、言葉が通じないこと。アメリカ人とのほうがはるかに話が通じる！　いかにプロトコルに頼っていたのかわかった」といった話がでたり、神戸に行っていた松村亮平さんから「請負とコーポレートフェローシップの違いは上下関係があるかないか。対等の関係で課題に向かえる」といった話、同じく神戸に行っていた宮崎光世さんからは「市というのは市民への良い発信の手段を意外と持っていない。新聞は年齢層が高い。ネットをうまく活用する余地がたくさんある」といった話が出てきました。

そのほかにもたくさんのセッションやワークショップがあり、そこかしこで情報交換がされていました。ちなみに、1日目で人気のあったセッションが「シン・ゴジラセッション」(**写真3**)。

自治体職員がシン・ゴジラをテーマに防災を語るというもの。そのほかにも政府職員によるIT戦略のワークショップがあったり、政府や自治体の職員による企画が多くあるのも、Code for Japan Summitの特徴です。

## サミット2日目

サミット2日目は、Code for AmericaのMonique

注2　THさんのスライドURL: **https://drive.google.com/file/d/0BxiwX2rqs8dzTW16alZKdEJlNWc/view**

Baena-Tanさんによるキーノートスピーチから始まりました。MoniqueさんはCode for Americaで、地域活動の取り組みのユーザ調査を指揮してきました。おもに、組織的な最優先事項とコミュニティのニーズを明らかにする、参加型デザインの調査プロセスの開発をブリゲイドと行ってきました。またCode AcrossやNational Day of Civic Hackingといった全米中の行動を促す日（national days of action）のプログラムの開発やコーディネーションなど、ブリゲイドのネットワークマネージメント支援も行ってきました。

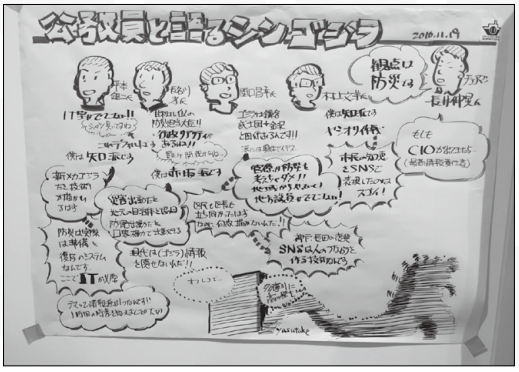
行政と地域コミュニティが協働をするための具体的な方法やヒントが満載のプレゼンでした。

1. 一般的な言葉、その場にいる人が理解のできる言葉を使いましょう
2. 期待値をコントロールしましょう
3. コミュニティの中間で引き合わせましょう

という3つの方法論や、各地のブリゲイド向けの活動のアドバイスもありました。情報発信や内部のコミュニケーションツールとして、MeetupやMedium、Slack、Loomio、Google Docsなどを活用しているそうです。

最後に締めくくりの言葉として言ってくれた、「You are not alone. あなたは一人ではありません。この課題に向き合っているのは我々だけではありません。行政の中にも市民の中にも行政に限らずさまざまなバックグラウンドを持った人たちがいて、それぞれ今の政治がうまくいくように取り組んでいます。それを忘れないでください。」という言葉は、多くの人の心に響いたようです。

◆写真3　シン・ゴジラセッションのグラフィックレコード




その後、1日目と同じく分科会セッションに入りました。筆者が参加したセッションからいくつかご紹介します。

## 公共システムの開発を成功させるには

Code for Japan初のフェローとして2014年から福島県浪江町に勤務し、町民向けアプリの開発を行った後、2016年4月からはIT専門官として神戸市に勤務している吉永隆之さんがセッションチェアとなり、同じく浪江町の町民向けアプリ開発プロジェクトにかかわった合同会社インフォラウンジの肥田野正輝さん、千葉市でオープンデータ施策やちばレポの立ち上げにかかわり、Code for Chibaの活動にも積極的に参加している千葉市役所 広報広聴課長の松島隆一さんにお話をうかがうセッションでした。

公共システムの開発は、通常の企業が行うものよりも制約がたくさんあります。随意契約が難しく、開発企業を好きには選べない問題や、アジャイル開発に向かない契約形態、異動などの影響で役所側とベンダー間の知識の非対称性が大きくなりがちで丸投げになってしまう、などです。

吉永さんから、浪江町のシステム開発で実際に体験した話をシェアしてもらいました。吉永さんからは公共調達に関して、行政システムのオープンソース化や、アジャイル開発ができる会社を見つけること、段階的な開発プロセスの導入などを提言していました。

松島さんが言っていた印象的なこととして、「システム調達の面から考えると、役所のフロント部分のシステムはブルーオーシャン。多くのベンダーが入り込んでいる既存部分を攻めるより、シビックテック側は外側の部分を攻めるほうが良いのでは」「役所の内側のシステムを変えていくには、シビックテック側の人が役所の中に入っていく必要がある。CfJの関さんが神戸市のCINO（Chief Innovation Officer）になったり、吉永さんが市の職員になったりという動きがもっと起こると良いのでは」という点がありました。

## Government × Startup

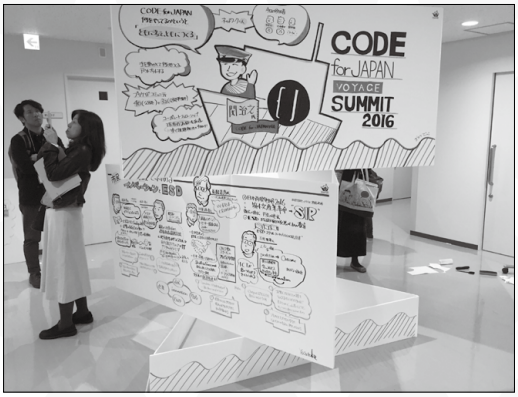
こちらは筆者が担当したセッションです。福岡市、神戸市、大阪市など、スタートアップを支援する自治体が増えてきました。世界的に見ても、市職員がスタートアップとタッグを組んで共に解決に挑むサンフランシスコ市や、4,500万ドル規模のスタートアップ支援を発表したホーチミン市など、自治体とスタートアップの結びつきが注目されています。

神戸市でスタートアップ支援を推進する多名部重則さんや、経済産業省でスタートアップと協働した経験を持つ津脇慈子さんにお話を聞きました。筆者からは、サンフランシスコの先進事例として、自治体が提示した地域課題をスタートアップが支援するStartup in Residenceプログラムを紹介させてもらいました。

多名部さんは神戸市で500 Startupsを始めとするスタートアップ支援施策を推進している人です。神戸市のスタートアップ支援は、市長から「なにかワクワクすることを企画してくれ」という話があったことからスタートしたとのこと。スタートアップが成長するには大きなエコシステムが必要という考えから、神戸市が支援するスタートアップは神戸市になくてもかまわないという姿勢で、神戸市外からも広くスタートアップを集めています。

津脇さんは経済産業省のプロジェクトで、スタートアップとアジャイルプロセスのシステム開発を行ったのですが、そのときに「発注側と開発側との信頼関係がないとアジャイル開発はちゃんと回らない」という点を強く感じたそうです。プロジェクト開始後、経産省側から「なにがどう進んでいるかわからなくて不安」という意見が出始めたときに、実際に開発現場に足を運んで、バックログの説明やタスクカンバンの作業内容の説明などをしてもらったところ、安心できたしコミュニケーションもうまくできるようになったとのこと。そのときの経験を活かし、今進めている地域IoTのプロジェクトでは、企業側に経産省の担当がついて、規制緩和についての取り組みを進めるなど、「ともにつくる」側に回るやりかたを実施しているそうです。Code for Japanのコーポレートフェローシップでは、企業の人材が自治体に派遣されますが、その逆のパターンもあるのだなと勉強になりました。

◆写真4　グラレコタワー




お二人と話す中で、行政がスタートアップを育成する取り組みが広がるには、行政側とスタートアップが立場を越えて同じ目的意識を共有し、プロジェクトベースで協働をしていくことが重要なのだなと感じました。とはいえ、「行政と付き合うのは面倒」と敬遠してしまうスタートアップもまだまだ多いと思います。Code for Japanとしては、スタートアップとの連携や起業家育成について、政府にいろいろと提言をしていきたいと考えています。

## グラフィックレコーディング

Code for Japan Summitは、日本でもかなり早い段階（2年前のサミット）で大規模なグラフィックレコーディングチームで会場のセッションを記録したのですが、今回も新たな取り組みをしました。多くのセッションをレコーディングして張り出しただけでなく、**写真4**のように、タワー状にレコーディングを組み合わせて、立体的に展示を行っています。

この前で記念写真を取っている人もたくさんいて楽しかったです。グラレコチームの皆さん、どうもありがとうございました！

★　★　★

2017年もサミットは実施する予定ですので、ご興味を持っていただいた方は、Code for JapanのFacebookグループなども覗いてみていただければと思います。SD

温故知新
# ITむかしばなし

第62回

# シャープMZ-2861
## ～MZの最後に輝く究極の16ビットパソコン～

速水 祐（はやみ ゆう） http://zob.club/ twitter @yyhayami

## はじめに

シャープ製のパソコンMZシリーズも、最後に輝いたマシンがありました。しかし、その高い性能の輝きは時代背景や広報メッセージの迷いにより一瞬で、ほとんど目立たない存在で終わってしまいました。今回は、このホビーパソコンMZ-2861（**写真1**）に光を当てます。

## MZ-2861とは

MZ-2000、MZ-2200と続くMZシリーズでは、1985年9月にスーパーMZと呼ばれたMZ-2500が発売されました。このMZ-2500は、MZ-80B → MZ-2000の非互換性の問題の反省を活かして、MZ-80BとMZ-2000の双方のマシンの互換性を有し、CPUに6MHzの動作周波数のZ80Bを載せて、当時最高速の8ビットホビーパソコンとして登場しました。

また表現力の高いグラフィック機能を有し、解像度640×400ピクセル16色と、ゲーム用として活用できる320×200ピクセル256色の表示が可能でした。画面のハードウェアスクロールもサポートされていたため、縦スクロールシューティングゲームに最適なハードであり、電波新聞社から発売されたMZ-2500版XEVIOUS[注1]は、それまでのほかのパソコンの移植版とは一線を画す出色の出来でした。

しかし、1985年はNECから88シリーズの地位を確立するPC-8801mkIISRが発売され、シャープ製のパソコンは、MZとは異なる事業部から発売されていたX1ターボがホビーユーザから支持を集めていました。ですから、出遅れたMZ-2500は十分な販売実績を残せませんでした。

起死回生を期して1987年4月に登場するのがMZ-2861です。MZの後の数字286は、搭載されていた当時の最高速16ビットCPUのIntel 80286（以下i80286）の286と、最初の2がMZ-2000シリーズであることを示しています。MZ-2861は、8ビット機として完成されたMZ-2500との互換性を有し、I/O関連はZ80系の安定したハードを使い、それにi80286に最適化した新規開発のハードウェアを加えた究極の16ビットパソコンとして開発されました。

当時のi80286を載せた16ビットパソコンの多くは、その前の8086を使ったパソコンが前身にあり、その互換性に引きずられて自由にi80286用の設計ができなかったのです。

またMZ-2861のグラフィック機能は、高い表現力を持っており、MZ-2500の持つ出力に加えて640×400ピクセルで65,536色を表示できました。

## ワープロ専用機互換のMZ書院

シャープからは同じ1987年3月にX68000（68HC00 10MHz搭載）が発売されており、当時ホビーユーザに圧倒的な支持を得て、多くの高機能なゲームなど

**▼写真1 MZ-2861**

**注1）** ゼビウス。ナムコ（現在のバンダイナムコエンターテインメント）から1983年に発表された、縦スクロールのシューティングアーケードゲーム。

※本記事の執筆にあたり、2016年12月号「ITむかしばなしスペシャル」でも登場した、古籏一浩氏に写真や貴重な情報を提供いただきました。

のソフトウェアが発表されました。それに対してMZ-2861はスプライト機能はないものの、CPUの高速性では68HC00に比べて圧倒的に有利であり、ユーザの支持さえ得られれば、MZの地位の復活となるはずだったのです。

そこで、シャープがとった販売戦略は、なんとMZ-2861という名を前面に出さずに「MZ書院」と名を付すことだったのです。当時のシャープ製ワープロ専用機「書院」は、評判がよく、世の中はワープロブームでした。富士通でもFM-16βを諦め、富士通製のワープロOASYSの機能を持つFM-Rが発売されたのもこの年です。また、PC98シリーズ(以下PC98)が16ビットパソコンとして地位を固め、ビジネスユースだけでなくホビーユースでもPC98の時代になってきていました。PC98は、ワープロとしてジャストシステムの一太郎と、表計算としてLotus 1-2-3を使うことが一般的でした。当時は、漢字変換(ATOKなど)の変換スピードを上げるために辞書をRAMディスクに置くこともさかんに行われていました。

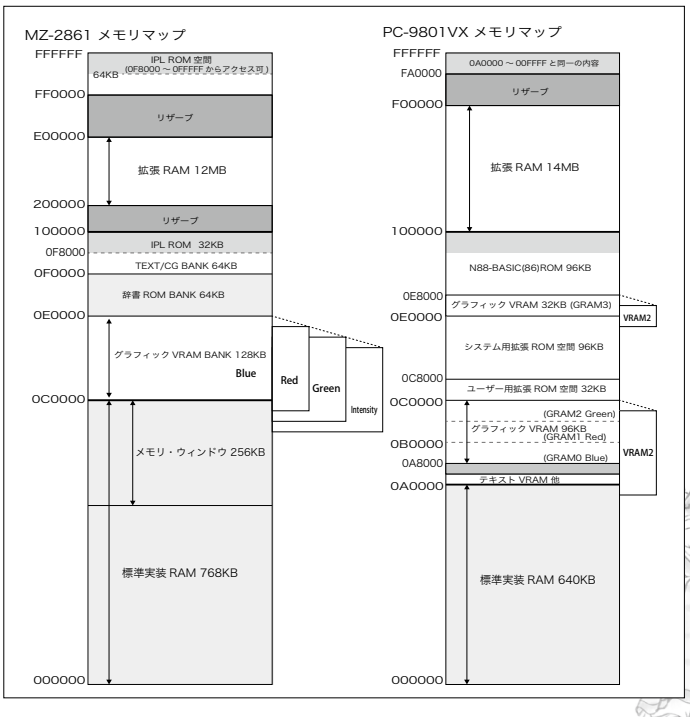
MZ-2861は、メインメモリ空間に辞書ROMエリアを設けることで、RAMディスク以上の高速な漢字変換を実現しており、付属のMS-DOS 3.1には、カナ漢字変換プロセッサとして書院の変換が使えるようになっていました。さらに、MZ-2861はPC98ソフトウェアエミュレーション機能もついていたのです。すなわちMZ-2861上でPC98用の一太郎やLotus 1-2-3を動作させることができたのです。

しかし、この年にはEPSONからハードウェア的にPC98と互換のPC-286が発売され、ソフトウェアエミュレーションでは速度的にとても対抗できなかったと思われます。当然、販売実績も上がらず、新たなソフトウェアはほとんど発表されず。MZ-2861の最後の数字の1が2になることはなく、MZシリーズは終焉を迎えたのです。

同じi80286が載ったPC-9801VXとMZ-2861のメモリマップを比べてみました(**図1**)。i80286における1MB以上のメモリ空間(最大16MB)は、プロテクトモードに切り替えないとアクセスできません。そのためには、i80286で用意されている命令を使いますが、リニアモードに戻る命令は用意されておらず、CPUに強制的にリセットをかけて、リニアモードに移行するしかありません。実行時間も要するため、PC98VXではいくらプロテクトメモリを増設しても、せいぜいRAMディスク程度にしか使えない状況でした。MZ-2861は、設計当初からプロテクトメモリを使うことを前提として考えられているため、リニアモードで扱えるメモリエリアに256KBのメモリウィンドウが用意されており、広大なプロテクトメモリ空間をリニアモードから扱えるようになっていました。

MZ-2861は、新規のi80286マシンとして優れた設計になっており、最適な動作を行うことができる実力を持っていたのですが、一瞬しか輝けませんでした。SD

**▼図1　PC-9801VXとMZ-2861のメモリ構成の比較**



# うまくいく チーム開発のツール戦略

第6回 「すぐに使える障害管理テンプレート」なら簡単！ JIRAでラクラクIT運用業務

Author リックソフト㈱ 斎藤 智昭、山本 鮎美、大塚 和彦

## Excelでのインシデント管理の苦しさ

みなさん、筆者はExcelが大好きです。思いついたらExcelで、課題一覧、帳票作成（もちろんExcel方眼紙）、システム構成図（もちろんExcel方眼紙）、報告書（もちろんExcel方眼紙）、プロジェクト管理……。なんて自由なツールなんでしょう！ この自由さが大好きです。ときに自由過ぎて周りに迷惑をかけてしまいますが、それでもやめられないのがExcelです。

某SIerに在籍していたころ、Excelでインシデント管理を行っていました。障害連絡が入るとExcelファイルに記入するので、記入件数がどんどん増えていきます。ステータスも記入しなければなりません。運用チーム数名で、ファイルサーバ上にあるExcelファイルを黙々とメンテナンスしていきます。このような日々の業務の中では、次のような問題が発生します。

- **利便性の問題**：ほかの誰かが使っていると、「読み取り専用」でしかファイルが利用できない
- **データ破損の問題**：ファイルの破損、もしくは上書きミスで一部のデータが消失する
- **情報連携と共有の問題**：各種の帳票がすべて別々のファイルなので連携ができていない。手作業で情報連携と共有を行うので生産性が悪い

世間にはITILというITサービスマネジメントのフレームワークがあり、それに準拠したツールがあるので、そのツールを使えば問題は改善するはずです。しかし、一見大きくなさそうで実は大きい問題として、これまでの報告書と書式が違うので読みにくいなどの問題が出てきます。

このような問題に対応するため、「Atlassian社が開発するJIRA Software（以下JIRAと表記）にアドオンを組み合わせてインシデント管理業務の運用を改善し、Excel方眼紙帳票にデータを差し込んで帳票として出力する」という夢のような方法をお伝えします。

## インシデント管理ツールとしてJIRAを使ってみよう

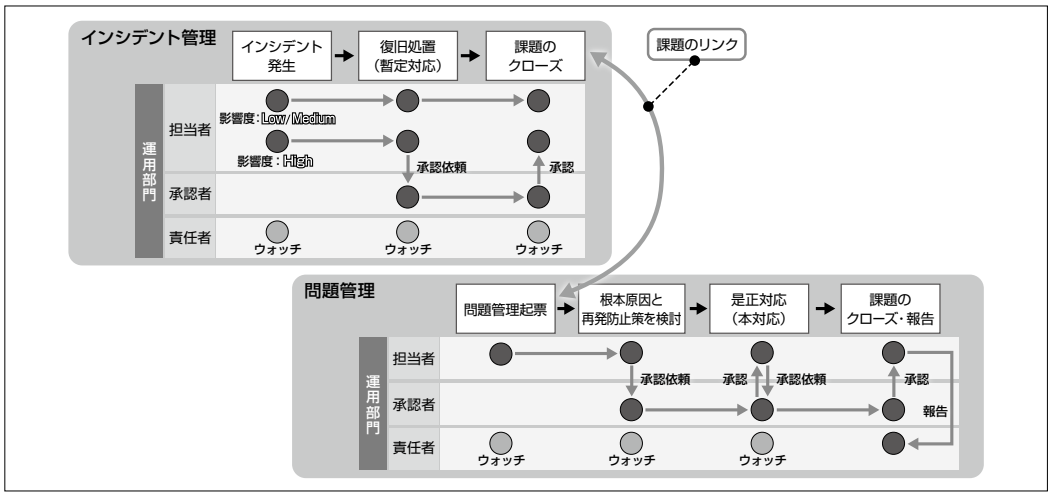
ITILでは、通常どおり業務を遂行できないシステムの状態を「インシデント」、インシデントを引き起こす根本原因を「問題」と呼びます。インシデントが発生したとき、迅速にサービスの復旧処置を施すまでを「インシデント管理」、根本原因を特定して再発を防止する是正処置を実施するまでを「問題管理」として別々に管理・解決していくことでITサービスの品質を向上させることを目標としています（**図1**）。ここでは、インシデント発生から解決までの流れを、JIRAを早期に業務利用できる「すぐに使えるテンプレートシリーズ」の障害管理テンプレートを利用して操作と合わせて説明します。このテンプレートのURLは、

**https://www.ricksoft.jp/solution/template/incidents.html**

となります。



▼図1　インシデント管理の概念




## インシデントの復旧処置

### a. インシデント発生!

ユーザから問い合わせを受けたら、インシデントをJIRAに登録します(**図2**)。インシデントの解決策の蓄積と傾向分析のため、受けた問い合わせはすべて登録することが重要です。インシデントには、システムが使えないなどの障害回避要求だけでなく、パスワード変更や情報開示などのサービス要求も含まれます。

### b. 担当者にインシデント対応を依頼する(担当者の割り当て)

ユーザからの問い合わせ内容を課題に登録したら、インシデントの担当者をアサインします。このとき、アサインされた担当者には、インシデント対応を割り当てられたという内容の通知が送られます。その担当者がJIRAにアクセスすると、ダッシュボードに課題が表示されます(**図3**)。

▼図2　JIRAの課題作成画面


▼図3　JIRAのダッシュボード


### c. ビジネスへの影響度と緊急度、優先度を決める

インシデント対応をアサインされた担当者は、影響度と緊急度、優先度を決めます。影響度(ビジネスにおけるインパクト)は、インシデントによりサービスを利用できなくなるユーザの数や、システムの規模などから判断します。緊急度は、インシデントを解決するのに必要とする時間から判断します。影響度が高くても、運用回避や代替手段の利用が可能な場合は緊急度は低くなります。優先度は、影響度と緊急度から算出できます。

### d. 対応策を調査する

続いて、障害回避要求かサービス要求かの分

類をします。テンプレートでは「種別」という項目で選択します。サービス要求の場合は、用意してある手順を準備します。障害回避要求の場合は回避策を調査し、過去に同様のインシデントが起きていれば、その回避策のように対応します。インシデントを課題として蓄積することにより、その回避手順(ワークアラウンド)も蓄積されていきます。ここでは、インシデントの原因が不明な場合でも特定はしません。また暫定回避ができない場合は、対応不可としてクローズし、問題として管理します。

### e. インシデントを解決する

調査した回避策により、インシデントを解決します(**図4**)。テンプレートで「作業開始」を実行して調査内容を入力し、「対応中」ステータスに遷移させます。影響度の高いインシデントの場合は、クローズする前に対応内容の承認を受けます。テンプレートには、承認待ちの課題や解決状況を簡単に把握できるように、専用のダッシュボードが用意されています。

### f. インシデントをクローズする

サービスが復旧していることを確認したら、インシデントをクローズします。インシデントがクローズされると、この回避手順は既知のワークアラウンドとなり、同様のインシデントが起きた際に役立ちます。

◆ ◆ ◆

以上で、インシデント発生から解決までの「インシデント管理」の一連の活動を説明しました。

次に、インシデントの根本原因を調査しインシデント発生を防止する「問題管理」の活動を説明します。

**▼図4　JIRAの作業開始画面**

## インシデントの是正処置

### a. 問題管理を起票する

暫定回避し、再発する(または再発する可能性のある)インシデントや、暫定回避できなかったインシデントを「問題」としてJIRAに登録し、根本原因と再発防止策を調査して解決します。テンプレートでは、該当するインシデントを課題リンクで関連付けし、特定された問題の基礎情報を入力します。

### b. 問題解決の担当者を割り当てる

問題の基礎情報を課題に登録したら、問題解決の担当者をアサインします。このとき、アサインされた担当者には、問題解決を割り当てられたという内容の通知が送られます。

### c. 根本原因・再発防止策を調査する

問題解決にアサインされた担当者は、インシデントの場合と同様に優先度を決め、調査を開始します。 このとき、インシデント管理と同様に、過去の問題から同様の事象がないか調べます。解決済みであれば、その原因・解決策を参考にします。問題の原因が特定できたら、解決策、再発防止策を検討し、是正対応の作業内容について上長へ承認を求めます(**図5**)。

### d. 問題を解決する(是正対応)

上長から承認を受けたら、調査した解決策をもとに是正対応を実施します。対応が完了したら、対応の結果を記載して上長へ承認を求めます。

### e. 問題をクローズする

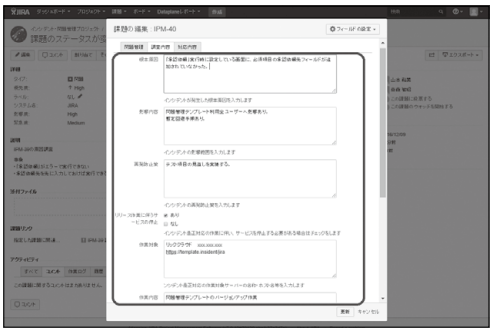
是正対応の内容について上長から承認を受け

たら、問題をクローズします。クローズされた問題は、今後同じような事象が発生した場合のナレッジベースとして蓄積されていきます。

## 帳票出力で管理者に報告

JIRAの課題の出力機能は標準ではXML、Word、印刷用しかなく、Excelでの出力はできませんが、JIRAの拡張機能として追加できるアドオンでExcel出力できるものがいくつかあります。弊社で提供している「すぐに使える障害管理テンプレート」ではExcel出力できるアドオンを使用しているので、JIRAを使用している場合でもExcelファイルの報告書をそのままテンプレートとして使用できます。Excelファイルの報告書の入力項目にJIRAのフィールド項目を設定し、JIRAにテンプレートとしてアップロードすれば、すぐに使用できます(**図6**)。

**▼図5　JIRAの課題編集画面**



## おわりに

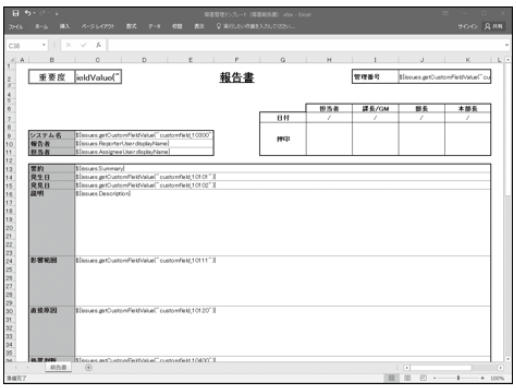
JIRAは、インシデント管理で重要な「問題発生」から「問題解決」までの間、情報を共有しながら効率的に業務を進めるためのベース機能を持っています。インシデント管理業務をすぐに始められる「すぐに使える障害管理テンプレート」を適用することで、今回紹介した機能を使うことができます。Excelをやめるのではなく、Excelとも仲良くするためのインシデント管理ツールをぜひ試してみてください。SD

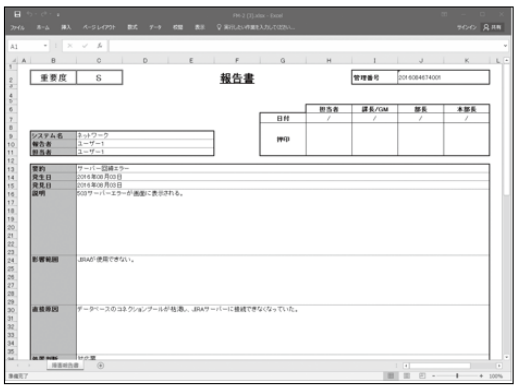
**※本連載の過去記事は技術評論社のWebサイト「gihyo.jp」でもご覧になれます。**

http://gihyo.jp/ad/01/atlassian



**▼図6　(左)報告書のテンプレート　(右)ダウンロードされる報告書**





Service

## Packet Host、ARMサーバを使ったベアメタルクラウドサービスを日本で提供開始、ソフトバンクが全面支援

米Packet Host社は2016年12月16日、ARMサーバを使ったベアメタルクラウドサービスの日本での提供を開始した。またこれに伴い、同日にローンチイベントが開催され、そこではソフトバンク㈱が本サービスの日本展開に際して、全面的な支援を行うことが発表された。

Packet Hostは2014年7月設立、現在従業員が二十数名ほどのスタートアップで、ベアメタルクラウドを提供するクラウド事業者。2016年11月にARMアーキテクチャのサーバマシンを提供開始し、サービス開始2週間で400社への提供を行った。ソフトバンクはその実績と革新的なサービスに注目し、出資を決めたそう。現在Packet Hostは、Docker、Kubernetes、MesosなどクラウドネイティブやIoTでの利用、RHEL、Ubuntu、Core OSおよびFreeBSDの各OSの64-bit ARMサポートのため、エコシステムとの協力を進めているとのこと。また、今回の日本での提供開始と同時に、サンノゼ、ニュージャージー、アムステルダムに加え東京リージョンも開設した。

今回提供が開始されたのは、「ARMv8-Aサーバ」を用いたベアメタルクラウドサービス「Type2Aサーバ」。ARMサーバを使用している理由は「低消費電力×高密度」で、インテルアーキテクチャサーバと遜色ないパフォーマンスを、従来比10分の1の省電力と、1サーバラックあたり7,300コアという高密度（＝省スペース化）によって、はるかに低コストで実現できるそう。現在の利用料は85円／時間、つまり1コアあたり1円以下／時間である。

またクリエーションライン㈱が、日本でのType2Aサーバの展開において提携を発表しており、ベアメタルARMクラウド上でのIoTデータ解析ソリューションの提供を予定している。

▲左：ソフトバンク㈱専務取締役エリック・ガン氏、右：米Packet Host社CEOザッカリー・スミス氏

**CONTACT**
**Packet Host** URL https://www.packet.net
**ソフトバンク㈱** URL http://www.softbank.jp

Hardware

## ラズベリーパイ財団創設者来日、Raspberry Pi開発の経緯と日本での展開について

2016年12月15日、ワンボードコンピュータRaspberry Piを提供する英ラズベリーパイ財団の創設者の1人、エベン・アプトン氏が来日し、Raspberry Pi開発の経緯や日本での展開についての記者発表会を行った。

Raspberry Pi誕生のきっかけとなったのは、エベン氏が英ケンブリッジ大学で教鞭をとっていた2006年、コンピュータサイエンスを受講する学生の数が年々減少していることだった。エベン氏はその原因として、Commodore 64やMSX、X68000、BBC Microといった8ビット・16ビットのマイクロコンピュータがゲーム機やパソコンに取って代わられ、プログラミングのできるコンピュータが子供部屋からなくなったからだと分析している。

そして、氏がRaspberry Piの開発に乗り出したのが2007年。「プログラミング可能で、多くの言語に対応しているもの」「子供の興味を引くもの」「小型で強度が高く、子供が学校へ持っていくことのできるもの」「教科書の価格程度の25ドルといった安価なもの」という4つの基準を満たしたマシンを開発することで、学生とプログラミングのギャップを埋められると考えた。

2011年、当時の財団メンバー6名が計25万ドルの資金を持ち寄って、最初の1万台を中国で製造。売り出そうとした2012年のはじめ、英アールエスコンポーネンツ社とのライセンス契約が成立して大量生産につながり、初日に10万台が売れた。そして順調にバージョンとモデルを重ね、今に至る。

またエベン氏は、アールエスコンポーネンツ㈱が11月10日に提供開始したRaspberry Piの国内生産モデル「日本製Raspberry Pi 3 Model B」にも触れ、「Made in Japan」のRaspberry Piを見られた喜びを語った。こちらはRSオンライン（**http://jp.rs-online.com**）から購入できる。

▲ラズベリーパイ財団エベン・アプトン氏

**CONTACT**
**Raspberry Pi財団** URL https://www.raspberrypi.org

Service

## グレープシティ、「Spreadsheet for Salesforce」発売

グレープシティ㈱は2016年12月7日、Salesforce向けソリューションシリーズ「GrapeCity for Salesforce」の第2弾となる「Spreadsheet for Salesforce」を発売した（第1弾は同年3月発売の「Barcode for Salesforce」）。

Spreadsheet for Salesforceは、SalesforceにExcelライクなユーザインターフェースを提供するAppExchange注アプリ。SalesforceのデータをExcelのような柔軟な一覧形式で参照したり編集したりでき、またExcelから直接コピー&ペーストでデータを一括入力できるので、業務担当者のデータ更新作業を大幅に効率化できる。Salesforceと完全に統合されており、プラグインなどのソフトウェアはいっさい不要。MacやWindowsで動作するブラウザだけですべての操作を行える。また、SalesforceのSNS機能であるChatterにも対応しており、新規ToDoの登録や行動履歴などのChatterフィードもSpreadsheet上で利用できる。

本製品はサブスクリプションによるクラウドサービスで、1ユーザの月額利用料は1,500円（税別）。最低10ユーザからの契約で、1年単位の自動更新となっている。

**注)** ユーザ企業がSalesforce製品用に開発した画面、機能などを購入できるサービス。

**CONTACT**
**グレープシティ㈱** URL http://www.grapecity.com

Topic

## 「Ruby biz Grand prix 2016」、表彰式が開催

2016年12月15日、「Ruby biz Grand prix 2016」の表彰式が行われた。

Ruby biz Grand prixはビジネスの領域において、プログラム言語Rubyの特徴を活かした新たなサービスを創造し世界へ発信している企業、団体および個人を対象とするグランプリ。審査委員長はまつもとゆきひろ氏。開催2回目の今年は29点のエントリーがあった。受賞企業は次のとおり。

●受賞一覧

| | | |
|---|---|---|
| グランプリ | ㈱MISOCA | 請求書作成サービス「MISOCA」 |
| | ラクスル㈱ | ネット運送サービス「ハコベル」 |
| 特別賞 | ㈱アクトキャット | 自動コードレビューサービス「SideCI」 |
| | ㈱ジモティー | 広告掲示板「ジモティー」 |
| | ユニファ㈱ | ポータルメディア写真サービス「るくみー」 |
| グローバル賞 | 独Planio社 | プロジェクト管理サービス「Planio」 |
| ソーシャルイノベーション賞 | 正興ITソリューション㈱ | 健康管理ソリューション「Health-Ledger」 |
| | ㈱メドレー | オンライン診療アプリ「CLINICS」 |

**CONTACT**
**Ruby biz Grand prix** URL http://rubybiz.jp

Service

## ネオジャパン、ビジネスチャット「ChatLuck」の新バージョン（V2.0）を提供開始

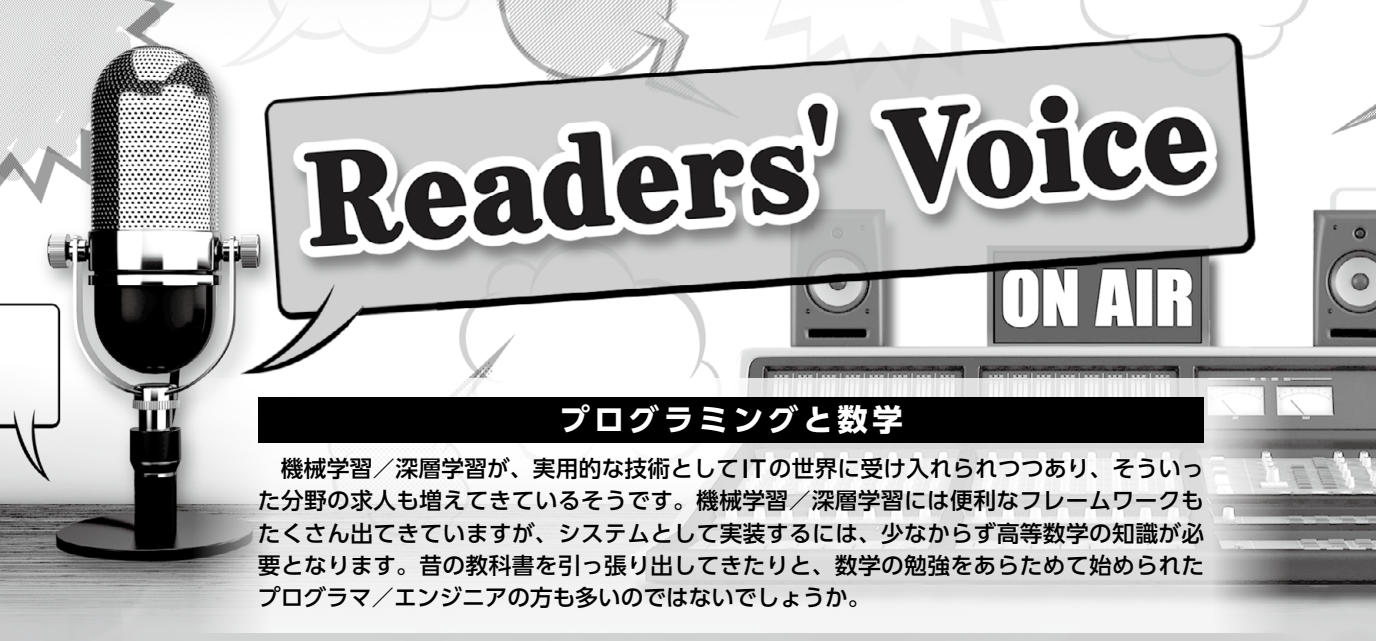
㈱ネオジャパンは2016年11月22日、「ChatLuck」の新バージョン（V2.0）の提供を開始した。

本製品は、PCやスマホなどから利用する企業向けのリアルタイムチャットツール。ビジネスに特化した管理／セキュリティ機能を備え、製造業、サービス業、地方自治体を中心に採用されている。V2.0では、次に挙げるように、スマホ版の機能を大幅に強化した。

- GPSを用いて、メンバーの位置情報をリクエスト・収集・確認する機能と、自分の位置情報をメンバーに共有する機能を搭載。訪問先の指示、集合場所の連絡、災害時の安否確認などのシーンで活用できる
- スマホで動画を撮影して投稿できるようになった
- スマホで撮影した写真を投稿する際に、画質を軽量化して送信する機能を追加。これによりデータ通信量を削減し、よりスムーズなやりとりが行える
- スマホへのプッシュ通知を行う曜日や時間帯を設定可能。業務時間外の通知を抑制する場合に活用できる

そのほか、管理者によるシステムメンテナンスなどの一斉通知が送れる機能など、多数の改善が行われている。

**CONTACT**
**㈱ネオジャパン** URL http://www.neo.co.jp




## プログラミングと数学

機械学習／深層学習が、実用的な技術としてITの世界に受け入れられつつあり、そういった分野の求人も増えてきているそうです。機械学習／深層学習には便利なフレームワークもたくさん出てきていますが、システムとして実装するには、少なからず高等数学の知識が必要となります。昔の教科書を引っ張り出してきたりと、数学の勉強をあらためて始められたプログラマ／エンジニアの方も多いのではないでしょうか。

## 2016年12月号について、たくさんの声が届きました。

### 第1特集 NoSQLの教科書

ハッシュテーブルのしくみ、MongoDB、Couchbase、Redis、NoSQLとして使うMySQLと、今のNoSQL業界を横断するような特集でした。気になった／使いたくなったDBはありましたか？

**普段は意識していなかったハッシュの奥底まで踏み込んでいてよかった。**
**りょうじさん／東京都**

**いまだに実案件でNoSQLを使う機会に遭遇していないのですが、いつか使いたいと狙っています。**
**オトさん／神奈川県**

**業務でKVS（Redis）を利用しているため、再度基本を学べて良かった。**
**massakiiiさん／福岡県**

**DBの基本概念から解説してあり、わかりやすい。**
**林さん／愛知県**

**これまでNoSQLはmemcachedしか使ってこなかったので参考になりました。MySQL Clusterはおもしろそうなので検証してみたいと思います。**
**犬棟梁さん／埼玉県**

**同じKVSのNoSQLでもいろいろな特色があることがわかり、とても参考になりました。次回機会があればグラフ型データベースについても取り上げていただけるとうれしいです。**
**出玉のタマさん／大阪府**

実用段階に入って久しい各種NoSQL、使いたいと考えている読者の方は多いようです。実装するシステム、扱うデータに合わせてSQLと使い分けられれば、強い武器になることでしょう。

### 第2特集 文字コード攻略マニュアル

プログラミングでは避けては通れない文字コードについて、文字どおり0と1から解説しました。第2章からは実践編として、HTML、Java、Ruby、MySQLにおける文字コードのハマりどころとその回避策を扱いました。

**文字コード関係の問題はいつも付け焼刃の知識でしのいできたので、あらためて体系的に整理でき、たいへん勉強になりました。**
**南雲さん／埼玉県**

**もう文字コードはUTF-8の時代なんですね。まだShift_JIS指定しています。読める人いないんじゃないかしら。ちょっと気になります。**
**Tayuさん／千葉県**

**文字化けさせないためにも文字コードの理解は不可欠だと思いました。**
**永作さん／東京都**

**文字コードについては最近の動向をちゃんと理解できるようになりました。**
**栗原さん／大阪府**

**UTF-8で統一かと考えていましたが、まだまだShift_JIS（CP932）の呪縛は消えませんね。**
**いとうはじめさん／宮城県**

**そういえば、最近はShift_JISの0x5Cに起因する文字化け問題、あまり見なくなった気がしますね。UTF-8普及が進んでますよね。**
**のりぃさん／埼玉県**

**今回の文字コードの特集はたいへん役に立ちました。ありがとうございます。このようなまとまった情報は、今まであまりなかったような気がします。普段文字コードエラーが出ると、その場しのぎで処理していました（内緒ですが）。**
**山根さん／京都府**

## 12月号のプレゼント当選者は、次の皆さまです

**①Wi-Fiホームルータ「Aterm WG2600HP2」**
濱田弘様（兵庫県）

**②ウイルスバスタークラウド＋デジタルライフサポート プレミアム1年版**
オブジェクト脳192様（大阪府）

**③Qiitaノベルティ Tシャツ（Mサイズ）**
**ブルー：**NGO VAN KHANH様（東京都）
**グレー：**大平圭佑様（東京都）

**④『Ansible徹底活用ガイド』**
菊地謙様（愛知県）、青木悟様（神奈川県）

**⑤『確かな力が身につくPHP「超」入門』**
つばき様（愛知県）、高瀬行夫様（石川県）

**⑥『インフラエンジニアの教科書2』**
Sakura Onishi様（東京都）

**⑦『ポートとソケットがわかればインターネットがわかる！』**
福田悟史様（埼玉県）、岩永良太様（東京都）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

文字コードについて体系的に学べて良かったという声が多く寄せられました。そのほか、文字化けで痛い目をみたという読者の声も多かったです。本特集は、文字化けなどのトラブルシューティングにも役立つ内容でした。

### 第3特集 ITむかしばなしスペシャル

連載記事「ITむかしばなし」の特別版。日本のIT黎明期から活躍されている9人の方に、ご自身の経験と当時の様子を振り返っていただきました。

**若手エンジニアとしては、過去を知る機会ができて、ためになりました！**
**ぴょうへいさん／大阪府**

**初期のインターネットダイヤルアップ接続とユーザ認証を読んでいて、なんとも懐かしい気分になりました。**
**yoshitakaさん／神奈川県**

**おじさんなので昔話は懐かしい限りです。**
**FMタウンズ世代さん／埼玉県**

**大学でFM-11（Intel 8088）にLISP処理系を実装したのを思い出しました。RAMを空けるためFloppyでSwap機構も作りましたが、それも含めすべてアセンブラ記述でした。**
**e-jさん／神奈川県**

**リアルタイムで経験した懐かしい話が多かったです。あの当時、もっといろいろとしていれば、今はもうちょっと違ったのかな、という回顧的な妄想もあったり。(^^;;**
**福田さん／神奈川県**

現役世代には懐かしく、若手世代には新鮮なお話でした。ほかの産業に比べればまだまだ歴史の浅いIT業界ですが、それでも現在に至るまでの先人の試行錯誤が確かに存在し、今の便利さにつながっているのだと実感できました。

### 一般記事　[次世代言語] Elixirの実力を知る（後編）

並行処理が書きやすく、耐障害性の高いWebアプリを作るのが得意な言語として注目が集まる「Elixir」の入門記事です。後編では、プロセスによる並行処理とmix・Phoenixを使ったアプリ開発について解説しました。

**最近いろんな場面で少しずつ見かけるようになった言語ですが、まだ情報が少ないので、勉強する機会ができて良かったです。**
**村橋さん／北海道**

**年末年始の休みにでも試そうかと……。**
**うたさんさん／大阪府**

**正直、言語が多過ぎて業務の上ではある程度絞っている。しかし、システムを刷新する際の参考になるし、こういう記事を読むことで新しいアイデアの発想につながる。**
**s5k6s842さん／沖縄県**

**Phoenixを使ってアプリケーションを作ってみたくなりました。**
**tomato360さん／東京都**

どんどん新しい言語が出てきて、すべてを試すことはとてもできないですね。ただ、言語の強みや弱みを押さえておくことで、未来における選択肢の1つにすることができます。並行処理に興味がある方、Elixirがお勧めです。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

# 次号予告

Software Design
March 2017

**2017年3月号**
**定価（本体1,220円+税）**
**192ページ**

**2月18日発売**

**［第1特集］マーケティング&サービス向上に役立つ**
## ログ&データ分析基盤入門

**［第2特集］**
## 実践DevOps！【インフラ自動化】完全マニュアル

**［特別企画］**
## PG-Stormとは何か（前編）

**［一般記事］**
## なりすましメール対策
## Amazonペイメントのしくみ

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

### お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2017年1月号
●P.128　連載「セキュリティ実践の基本定石 第39回」左段下から14行目
［誤］CNCのサーバには外部から101/tcpポート
［正］CNCのサーバには外部から48101/tcpポート

### SD Staff Room

●ちょっとした運動の後、僧帽筋を痛め首がまわらなくなって2週間、そして現在1ヵ月すぎても背中の痛みが取れない。治療では背中・肩・首の筋肉がカチンカチンだそうだ。自覚症状がないので肩こりをしないタイプと自分を決めつけていたけど、体中がいろいろダメになっている。メンテナンスが要る時期に来たなぁ。（本）

●自分がファイルシステムを意識したのは、DISK BASICやMS-DOSのFDだった。1980年代はPCゲームが流行り、プロテクトがかかったFDがいろいろあった。規格外のトラックに情報があったり、回転数を落として同一トラックにセクタを増やしてあったり。シーク音で何かわかった頃が懐かしい。（幕）

●猫カフェ初体験！　土曜のお昼前、食事を終えてやや眠くなった猫がウロウロ、ゴロゴロしている座敷部屋は完全なまったり空間。ごきげんをうかがいながら、なでさせてもらう。たまにフギャっと怒られて、すみません、と手を引っ込めたりしながら撮影。かわいい寝顔を撮影した、この瞬間の幸福感が◎。（キ）

●近所の本屋が雑貨屋化してきています。本よりも利益率の高い雑貨の品数を増やすからには、経営難なのでしょう。倒産したら困るので、Amazonの利用は控えてその本屋で本を買っています。同様に、近所の街並みがチェーン店だらけになると嫌なので、できるだけ個人店を利用するようにしています。（よし）

●取材の仕事でラスベガスに行ってきました。ストリップ地区のホテルはどれも巨大で豪華絢爛。ダウンタウン地区のフリーモントストリートにはカジノが立ち並び、天井は電飾アーケードになっていて、深夜なのに昼間のような明るさ。なんと空港にまでスロットマシーンが並んでいました。業が深い街です。（な）

●11月は寒い日が続いて積雪。12月は冬至を迎えているにもかかわらず、暖かい日が続いて師走とは思えない季節外れの天候でした。北極の海氷が例年より多く溶けていることもあり、この先はぐぐっと寒くなる日もあるとか。2月の受験シーズンに雪が降らないことを祈りつつ。穏やかな年になりますように！（ま）



ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

**Software Design**
**2017年2月号**

発行日
2017年1月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱


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


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