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
9
2016
2016年9月18日発行
毎月1回18日発行
通巻377号
（発刊311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

》役立つログの取り方　》PHP入門

Special Feature 1

ログ出力のベストプラクティス

知りたい情報集まっていますか？

システムログから
アプリケーションの
ログまで徹底解説

Special Feature 2

良いPHP、悪いPHP

——すぐ効くWeb開発入門

使いこなせていますか？

きしだなおきの
「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

Extra Feature

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）

9
2016
［目次］
Software Design

contents

Special Feature 1 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

9
2016
［目次］
Software Design

Special Feature 2 第2特集

使いこなせていますか?

良いPHP、悪いPHP すぐ効くWeb開発入門 063

Extra Feature 一般記事

àla carte アラカルト

contents

Column

Development

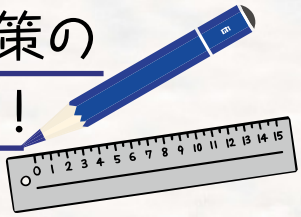
OS/Network

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

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

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

言語のセオリーを徹底解説！

パーフェクトシリーズ

パーフェクト Python

Pythonサポーターズ　著
B5変形判／464ページ
定価（本体3,200円+税）

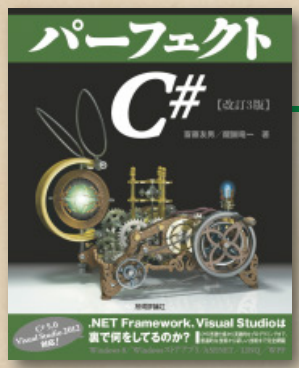
ISBN978-4-7741-5539-5

最新刊！

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
定価（本体3,200円+税）

ISBN978-4-7741-8316-9



パーフェクト C# [改訂3版]

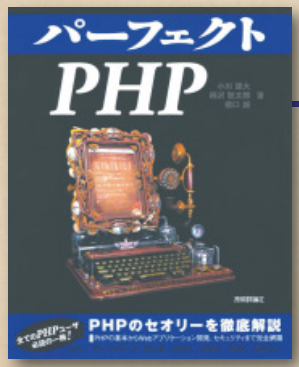
斎藤友男・醍醐竜一　著
B5変形判／608ページ
定価（本体3,600円+税）

ISBN978-4-7741-5680-4

パーフェクト Java [改訂2版]

井上誠一郎・永井雅人　著
B5変形判／592ページ
定価（本体3,200円+税）

ISBN978-4-7741-6685-8



パーフェクト PHP

小川雄大
柄沢聡太郎
橋口誠　著
B5変形判／592ページ
定価（本体3,600円+税）

ISBN978-4-7741-4437-5

パーフェクト Java Script

井上誠一郎・土江拓郎
浜辺将太　著
B5変形判／544ページ
定価（本体3,200円+税）

ISBN978-4-7741-4813-7

パーフェクト Ruby

Rubyサポーターズ　著
B5変形判／640ページ
定価（本体3,200円+税）

ISBN978-4-7741-5879-2

パーフェクト Ruby on Rails

すがわらまさのり・前島真一
近藤宇智朗・橋立友宏　著
B5変形判／432ページ
定価（本体2,880円+税）

ISBN978-4-7741-6516-5

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

第93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。

Rust

新プログラミング言語「Rust」

「Rust」はMozilla Foundationが中心となってオープンソースで開発が進められているプログラミング言語です。並列処理をサポートしたマルチパラダイムな言語を目指しており、手続き型や関数型、オブジェクト指向、並列アクターモデルなどのパラダイムを取り込んだ静的型付けの言語として設計されています。

RustはC/C++のような低レベルな言語としての性質を持っており、OSやデバイスドライバといった、一般的な高レベル言語が苦手とする分野をサポートします。その一方で、不正なメモリ操作によるエラーを防止する安全なメモリ管理や、スレッド間のデータ競合を排除した高い並列処理性能など、高レベル言語の安全性・利便性も合わせ持っている点が大きな特徴です。

それに加えて、Rustの目標には「ゼロコスト抽象化」という項目も挙げられています。これは、高レベル言語のような抽象化を含めて、可能な限り抽象化のコストを下げるということを意味します。たとえば、最適化をすべてコンパイル時に行い、実行時の高速化にコストをかけないしくみなどが挙げられます。

そのほか、Rustが備える特徴的な機能としては次のようなものが挙げられています。

- 所有権システム
- トレイトベースのジェネリクス
- パターンマッチングによる分岐処理
- 型推論
- 最小限のランタイム
- 効率の良いCバインディング

「所有権システム」は、メモリ安全性を実現するために極めて重要なしくみです。Rustでは変数束縛によってメモリ上のリソースに名前を付けるようになっています（一般的な言語で“変数”と呼んでいるものに近い概念です）。この変数束縛は、束縛されている対象リソースの「所有権」を保持します。そして、Rustではリソースに対する束縛が1つだけであることが保証されるため、所有権もデフォルトでは1つの束縛でしか保持できません。このしくみによってメモリ操作の安全性を高めているというわけです。

「トレイト」は、ある型が提供しなければならない機能をコンパイラに伝える機能であり、型を定義する際にその振る舞いを強制することが可能になります。そしてRustのジェネリクスでは、“特定のトレイトを実装する型”のような型パラメータを指定できるようになっています。

これら一連の特徴的な機能によって、ハードウェアに近いレイヤのプログラミングをサポートしつつ、同時に安全性も担保するという難題をクリアしています。

Rust製の新ブラウザエンジン「Servo」

このRustの採用例として注目を集めているソフトウェアに「Servo」があります。Servoは、Mozillaの研究開発部門であるMozilla Researchによって開発が進められている新しいWebレンダリングエンジンです。従来のレンダリングエンジンにくらべて、並列性やモジュール性に優れ、高いパフォーマンスや堅牢なセキュリティを備えることを目標としています。また、モバイルデバイスも含めた幅広いハードウェアに対応するために、低い周波数で動作するマルチコアプロセッサでも高速な動作を実現しようとしています。

Rustは並列処理に優れ、コンパクトなランタイムでも高速に動作する一方で、メモリセーフに設計されていることから、このようなServoのコンセプトに最も適した言語として採用されました。長期的なロードマップとしては、現在Firefoxで採用されているレンダリングエンジンGeckoのコンポーネントを、徐々にServoのコンポーネントに置き換えることが計画されています。

Mozilla Researchは、2016年7月に初めてServoのデベロッパプレビューを公開しました。本稿執筆時点（7月中旬）で公開されているのはMac OS版とLinux版のみですが、Windows版とAndroid版も近日中に公開される予定とのことです。

Rustは低レベルのシステム開発向け言語としては比較的新しい存在ですが、Servoの登場によってその評価は大きく変わる可能性があります。このまま順調に開発が進めば、C/C++に代わる新しい選択肢としての存在感が増してくることでしょう。SD

The Rust Programming Language
https://www.rust-lang.org/
Servo, the parallel browser engine
https://servo.org/

DIGITAL GADGET vol.213

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

広告とデジタルの新しい関係性

広告のSNS活用、VR活用、人工知能活用

広告キャンペーンにおいて、TwitterやFacebook、Instagramといったsnsや、専用アプリを活用したものが、ごく普通に見られるようになりました。

Twitterのハッシュタグを活用したペルー赤十字の「Hashtags for Life」プロジェクトは、Twitterのハッシュタグに自分の血液型を投稿することによって、それを献血登録のデータベースとして活用し、緊急に血液が足りなくなった場合にSNS経由で献血者を募るという運用をしています。

最近日本にも上陸した米国のタコスチェーン店のタコベルでは、Unicodeの絵文字の新規採用分にタコスの絵文字を採用してもらうよう活動を行っていました。その採用を記念して「Taco Emoji Engine」というキャンペーンを実施しました。これは従来のさまざまな絵文字とタコスを組み合わせた変な合成絵文字をツイートして返してくれるというものです。

さらに最近では、広告の分野でのテクノロジーの活用が顕著で、VR技術を活用したもの、人工知能を活用した展開も見られるようになってきました。

NYT VR
http://www.nytimes.com/marketing/nytvr/

New York TimesがGoogle Card boardとGEとminiの協力を得て、報道のためのVR映像を配信。新聞の日曜版をとっている人全員に、VRカードボードが届けられたそう。しくみはVR映像を配信するVrse社の技術によるもので、GEとminiのVR広告もある。新聞社だけあって、取り上げる題材の社会性や臨場感が他のVRコンテンツとは一線を画している。

The Dalí Museum - Dreams Of Dalí
http://thedali.org/dreams-of-dali/

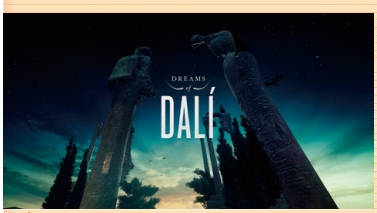
米国フロリダにあるダリ美術館の展覧会のために作られたもので、ダリが描いたシュールな世界の中をVR体験できるもの。ダリの「ミレーの晩鐘の考古学的回想」という作品がもとになっている(**pic.1**)。

Lockheed Martin - The Field Trip To Mars
http://fieldtriptomars.com/

アメリカ航空機・宇宙船の開発製造会社Lockheed Martin社によるもの。スクールバスの車窓全面にディスプレイを搭載してVR的に火星の映像を映し、バス通学を火星探索に変えてしまうプロジェクト。単に風景が映し出されているのではなく、バスの進行方向や移動速度によって映像が切り替わり、まさに火星の上を走るバスの景色が窓から見えるかのよう(**pic.2**)。

Clarif.ai
https://www.clarifai.com/

画像や動画を自動認識し、意味のある分類タグを自動付加するしくみ。



pic.1
シュールレアリズムを代表するダリの世界に没入できるVRコンテンツ「Dreams Of Dalí」

pic.2
The Field Trip To Mars。バスの窓に火星のVR映像を投影

pic.3
人工知能が描いた、レンブラントのテクニックを模倣した絵画

※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。

物の認識はもとより、人の顔の表情や、感情的な要素も読みとって、複数のタグをつけて分類できるAPIサービス。

The Next Rembrandt

https://www.nextrembrandt.com/

人工知能の活用で注目された本作は、画家レンブラントが描いた絵画を筆のタッチまでも機械学習し、レンブラント風の絵画をコンピュータが描くというもの。絵の具のタッチの凹凸まで3Dプリンタと絵の具の塗り重ねで再現している(**pic.3**)。

Cannes Lions 2016より

2016年6月に、広告を中心とした国際クリエイティビティフェスティバル、Cannes Lions(カンヌライオンズ)2016が開催されました。単なる広告からさまざまな製品やサービスにまで分野を拡げており、今年のカンヌライオンズでは、VR技術、デジタル系の演出、人工知能や機械学習を活用したサービスなどが注目を浴び、話題になりました。

その一方、テクノロジーの活用とともに「クラフト」と呼ばれる、職人技や人間味、芸術性なども重視されています。テクノロジーの活用は当然のことで、テクノロジーそのものはもう特別な話題や差別化事項にはならなくなってきています。

いわゆる「広告」の表現だったとしても、単に商品を売るための露骨なものではなく、社会貢献の評価や、ブランドイメージの向上を意図した表現も多くみられます。各部門の入賞作から、とくにモバイル端末や最新テクノロジーを活用した広告戦略のいくつかをデジタルガジェット視点で紹介しましょう。

サイバー部門(Webサイトやデジタル分野の広告)より

●**ニュージーランドのASB銀行 - Clever Kash:**現金を使わない、子供用貯金箱。お小遣いもアプリであげる時代に。子供がお手伝いをすると、象さん型のデジタル貯金箱にお小遣いが貯まる(**pic.4**)。

●**Tourism Australia - Giga Selfie:**オーストラリア政府観光局の広告キャンペーン。ある特定の位置でセルフィー(自撮り)を撮影すると、遠く離れた超望遠レンズで超高精細なセルフィーを撮影してくれるサービス。特殊な機材を用い、一眼レフ720枚分の高解像度写真が見られる(**pic.5**)。

モバイル部門(スマートフォン、タブレット端末向けの広告)より

●**Canon - Photo Coach:**
どういう場所で、どういう写真を撮ればいいのか、街中のデジタルサイネージで写真撮影のコツを教えてくれるキャンペーン。単なるガイドブックには載っていない、リアルなイベント、リアルな場所での撮影の解説を提供。カメラメーカーがカメラ機器そのものをアピールするのではなく、本来の「写真」に着目したキャンペーン(**pic.6**)。

●**Samsung - Try On A Six:**
ライバル社のスマートフォン端末を傾けると自社のスマートフォン端末が同じ方向に傾き、あたかも自分がその端末を持っているかのような気持ちにさせるデジタルサイネージ。Galaxy S6 edgeという液晶画面の左右が曲面になっているスマートフォンのためのもの(**pic.7**)。

デザイン部門より

●**Art Institute Of Chicago - Van Gogh Bnb:**ゴッホの絵画を再現した部屋。シカゴ美術館の展示会に合わせたプロジェクト。シカゴにある部屋をAirbnbで実際に借りることができる(**pic.8**)。

サイバー部門

pic.4
Clever Kash

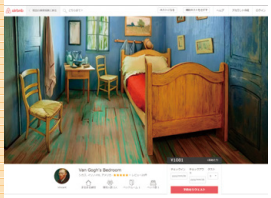
pic.5
Giga Selfie

モバイル部門

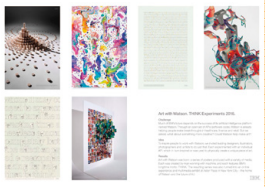
pic.6
Photo Coach

pic.7
Try On A Six

デザイン部門

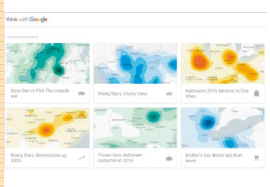


pic.8
Van Gogh Bnb

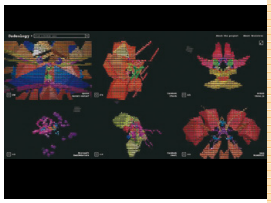


pic.9
Art With Watson

デジタルクラフト部門



pic.10
Google Shopping Insights



pic.11 Codeology
BrainTree Payments

●**IBM - Art With Watson:** IBMの人工知能システムWatsonを活用したアート作品。たくさんのアート作品の事例から機械学習して導き出したもの(**pic.9**)。

デジタルクラフト部門(デジタル環境においてブランドと消費者との優れた体験を生み出したもの)より

●**Google - Shopping Insights:** ECサイトのショッピングの状況を解析し、可視化するサービスが受賞。業界へのツールとしての意義とともに、専門家でなくともわかりやすいグラフ表現が逸品(**pic.10**)。

●**Braintree - Codeology:** ソースコード共有サイトGitHubのプロジェクトを分析し、可視化したもの。有機的な形をもったコードで表現されたプロジェクトは、どれ1つをとっても同じものはない(**pic.11**)。

この先の広告とクリエイティブの形態

テクノロジーでさまざまなことが実現できるとともに、テクノロジーやネットそのものは目新しさがなくなり、ごく当然に扱われるものとなりました。また、たいそうなコストがかかる最先端のテクノロジーではなく、使い古された枯れた技術であっても、ちょっとした工夫や見せ方次第でおおいに役立つということもわかってきました。

テクノロジーがどんなに進歩したとしても、人の感情の動き、人が得る体験、知らない場所への旅行や、人と人とのコミュニケーションによって生まれる新しい発見や共感など、根源的なことはあまり変わらずにあります。それらの根源的な事柄をテクノロジーによって強化できることは、まだまだたくさんあるのだと思い知らされることが多くなってきています。アイデアは誰でも思いつくことであり、そのアイデアを実行することこそに価値があるということですね。SD

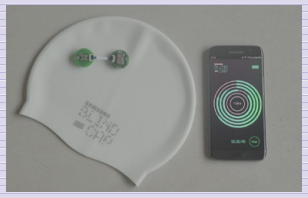
Gadget 1

Samsung Blind Cap

https://www.youtube.com/channel/UCndYDWrz36zgT72kYzMYsoQ

スマートフォン連動の水泳帽

カンヌライオンズ2016モバイル部門金賞。従来、目の不自由な水泳選手は、プールの端に来てターンする際、棒か何かで誰かに教えてもらわなければなりませんでした。Samsungが現在ベータテスト中の通信機能内蔵の水泳帽では、プールの端に近づいてターンするタイミングになったら、プールの外で控えているコーチがスマートフォンで通知すると、選手は水泳帽からの振動でそのタイミングを知ることができます。現在は練習で用いられているデバイスですが、将来的にはより広く使われる可能性がある技術です。



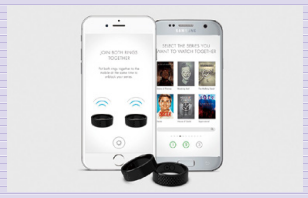
Gadget 2

Commitment Rings

http://seriescommitment.com/

承諾機能付き指輪

カンヌライオンズ2016モバイル部門銅賞。Commitment RingsはNFC内蔵の指輪で、カップルの2人がそれぞれ身につけ、あらかじめ観たい映画や番組を登録しておきます。その後お互いが承認しないと、ネット配信されている動画を観ることができなくなるアプリです。この指輪があれば、一緒に観ようと言っておきながら、抜け駆けすることができなくなります。実際に商品として販売予定で、iOS版、Android版のアプリも予定されています。



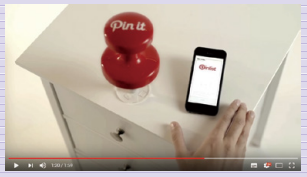
Gadget 3

Tok&Stok - Pinlist

http://www.bizsys.com.br/#/tokstok-pinterest-pinlist/

リアル世界のPinterest

カンヌライオンズ2016ダイレクト部門。気に入った画像をコレクションするネットサービスPinterestを現実世界にもってきたサービス。ブラジルの家具メーカーTok&Stokは、ショールームに設置されている家具にPinterestの機能である“Pin it”のボタンを取り付け、家具を気に入ったユーザがそのボタン押すことで、サイズや写真などを自分のコレクションアルバムに保存できるしくみを構築しました。Pin itボタンはバッテリー内蔵で、Bluetooth Low Energyでピンに一番近いユーザが持っているスマートフォン上のPinlistアプリと連動します。



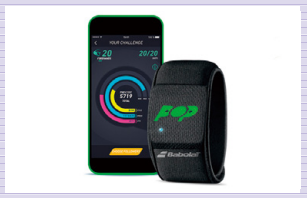
Gadget 4

Babolat Pop

https://www.babolatplay.com/pop

テニスセンサー

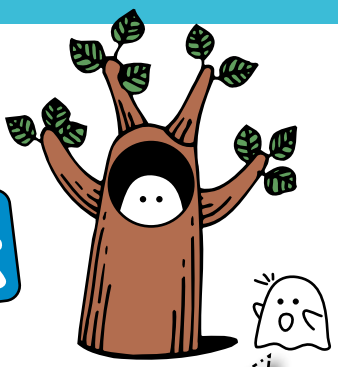
カンヌライオンズ2016モバイル部門銀賞。Babolat Popはテニス中のデータ記録を自動で行うためのデバイスで、ラケットを持つ腕に装着します。プレイ時間はもとより、フォアハンド、バックハンド、サーブの回数、スマッシュなどのスピード、最長ラリーの回数などを意識することなく測りつつプレイできます。Babolat Playというセンサーが内蔵されたテニスラケットとともに利用すれば、さらにさまざまな情報を記録することができます。米国では89.95ドルで販売中。



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

40

結城浩の再発見の発想法



三路スイッチ

三路スイッチとは

三路スイッチとは、**2つの場所のどちらでもオン・オフができるように設計されたスイッチ**のことです。通常のスイッチには「オン」と「オフ」という2つの状態があります。そのスイッチを使って、たとえば**図1**のような回路を作ると、スイッチをオンにすれば照明が点灯し、スイッチをオフにすれば消灯します。

ここで、「階段」に設置された照明を、階上と階下のどちらでも点灯・消灯させたいとしましょう。階上と階下の両方にスイッチを設置するとして、照明を制御しようとしたとたん、困ったことになるでしょう。

なぜなら、2つのスイッチが直列になっていたら、片方のスイッチがオフの状態では、他方でオンにできなくなるからです。これは論理演算x and yで、xがfalseならyがtrueでもfalseでも、結果がfalseになってしまうのと同じです。

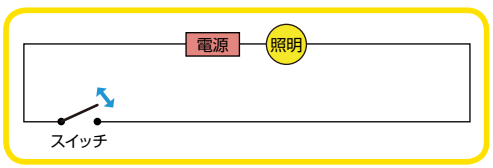
かといって、2つのスイッチを並列にするわけにもいきません。今度は、片方のスイッチがオンになっていたら、他方でオフにできないからです。論理演算x or yで、xがtrueならyがtrueでもfalseでも、結果がtrueになってしまうのと同じです。

三路スイッチはそのような問題を解決します。**図2**のように三路スイッチA、Bを階上と階下に設置します。すると、階上でも階下でも、照明を点灯・消灯させることができるようになります。ただし、配線の数は増えてしまいます。

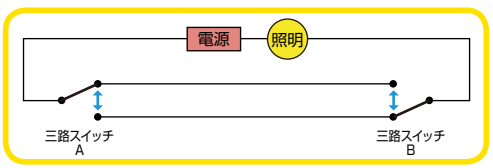
個々の三路スイッチは、単純なオン・オフを行うものではありません。**図2**の三路スイッチAでいえば、左の電線を上につなぐか、下につなぐかを選択しています。三路スイッチAとBの両方が上下の同じ側を選択すれば点灯し、異なる側を選択すれば消灯することになります。これは、xとyが「等しい」という論理演算に相当します(not(x xor y)とも言えます)。

階段の照明を三路スイッチで制御する話をしてきましたが、三路スイッチは階段だけに使われるわけではありません。長い廊下の両端、あるいは広い部屋の入口と出口など、人間がスイッチの制御のためにいちいち移動するのがたいへんな場所や、移動することがそもそも無意味な場所ならどこでも利用できるでしょう(階段を昇るために点灯したのに、昇ったあとで消灯のために降りるのは無意味です)。

▼図1　スイッチ



▼図2　三路スイッチ



三路スイッチの拡張

2ヵ所ではなく、3ヵ所で照明を点灯・消灯することはできるでしょうか。もちろんできます。**四路スイッチ**が存在します。**図3**では中央部に四路スイッチを設置しました。四路スイッチは、電線同士を「そのまま」つなげるか、「クロスして」つなげるかを選択します。3個のスイッチのどれでも、照明を点灯・消灯できるのです。

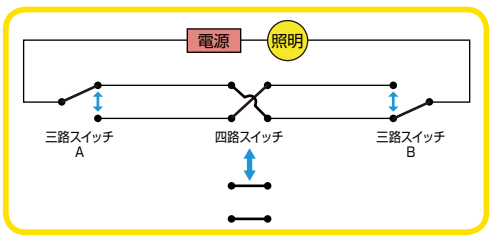
さらにおもしろいのはスイッチの個数はもっと増やせるという点です。しかも新たに「五路スイッチ」を作る必要はありません。四路スイッチを増やせばいいのです。**図4**はスイッチを全部で5個設置した例です。

日常生活での三路スイッチ

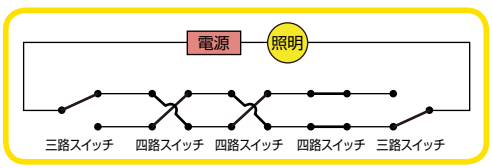
三路スイッチは私たちの日常生活で見かけるものですから、「日常生活での三路スイッチ」という表現は少しおかしいですね。でも、私たちは「必要な機能の再確認」という発想を三路スイッチから学ぶことができます。

「照明を点灯・消灯させる」という機能を実現するため、私たちはスイッチを用意します。でも、そのときに、**点灯や消灯を明示的に指示する必要はないのだ**という点がポイントです。通常のスイッチの場合、「オン」と「オフ」が決まっていて、それを使ってオンなら点灯、オフなら消灯という指示をします。でも、三路スイッチは違います。スイッチ自体ではオンもオフも決まっておらず、ただ2つの状態を切り替えるだけなのです。言い換えるなら、三路スイッチは、現在の点灯状態を反転させていると言えるでしょう。四路スイッチもそうですね。そのままつなぐか、クロスするかを選択して、現在の点灯状態を反転させています。

▼図3　四路スイッチ



▼図4　スイッチを全部で5個にした



別の考え方をしてみます。「照明を点灯・消灯させる」という機能を実現するため、**点灯と消灯の両方のスイッチを用意する必要はない**とも気づきます。たとえば、消灯をタイマーにまかせてしまうなら、人間に必要なのは点灯スイッチだけになります。

さらに別の考え方をすれば、**点灯と消灯のどちらも、人間が指示する必要はない**とも言えます。それに気がつけば、人感センサーを使って、自動的に点灯・消灯させることもできるでしょう。

三路スイッチについて考えているうちに「レンタカー」のことを思い出しました。レンタカーを借りるのは「別の場所に移動したい」からです。でも、A地点からB地点へ移動したいのに、レンタカーを返却するためにA地点に戻ってくるのでは意味がありませんね。そう考えると、レンタカーの**ワンウェイ**（乗り捨て）オプションは自然な発想になります。つまりワンウェイオプションは、**車を借りる場所と返す場所は同じである必要はない**という発想に立っているのです。

公開鍵暗号の**「閉める鍵」と「開ける鍵」は同じである必要はない**というのも似ていますね。

◆　◆　◆

あなたの周りを見回して、1ヵ所で制御するのではなく、2ヵ所で制御したくなるものはないでしょうか。「必要な機能の再確認」をして、2ヵ所で制御する方法を考えてみましょう。それは3ヵ所に拡張することはできるでしょうか。あるいはまた、そもそも人間が制御せずに済む方法はあるでしょうか。

ぜひ、考えてみてください。SD

増井ラボノート

コロンブス日和

第11回　Gear

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

2つのボタンだけで操作できるGear

前回は、大規模な階層情報をズーミング検索するDragZoomというシステムを紹介しました。DragZoomはパソコンやタブレット上のマウスや指の操作でズーミングを行うことによって大規模データの検索を行うシステムですが、ユビキタスコンピューティング環境ではいつでもポインティングデバイスが使えるとは限りません。料理しているとき、歩いているとき、通勤電車で立っているとき、車を運転しているときなど、ポインティングデバイスを使いづらい状況はよくありますが、そういった状況でも大きな階層情報をうまく検索できる方法があれば自由に音楽や番組などを選択できるので便利です。今回は、2個のボタンを使うだけで大規模な階層データを簡単に検索できる「Gear」というシステムを紹介します。

階層データの段階的なナビゲーション

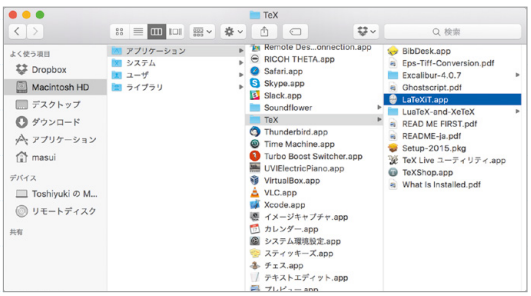
DragZoomのようにGUIを使って階層データをなめらかに検索する方法はこれまでたくさん研究されてきているのですが、残念ながらデスクトップやブラウザ上で広く利用されているものはほとんど存在せず、階層構造を段階的にたどりながらブラウズするやり方が一般的になっています。たとえばMacのデスクトップ画面でファイルを操作するプログラム（ファインダ）では、ファイルの階層構造を視覚化してナビゲーションを可能にする方法がいくつか用意されていますが、どの方法でもマウスやキーを操作して階層構造の親子関係や兄弟関係を段階的にたどっていくことにより階層型ファイルシステムのナビゲーションを行うようになっています（**図1**）。

たとえば1つ下の階層のフォルダの内容を見たい場合、マウスでダブルクリックしてフォルダを開いたり、矢印キーを使って注目行を移動させてから Enter キーを押すといった方法がよく使われています。

現在のパソコンのファイルはすべて階層的に管理されており、ほとんどのファイルにデスクトップからアクセスできるようになっています。パソコンの中の100万個のファイルの中から、ファイルを1つ選ぶのはたいへんなはずですが、実際のパソコン操作においてそれほど苦労している人は多くないでしょう。

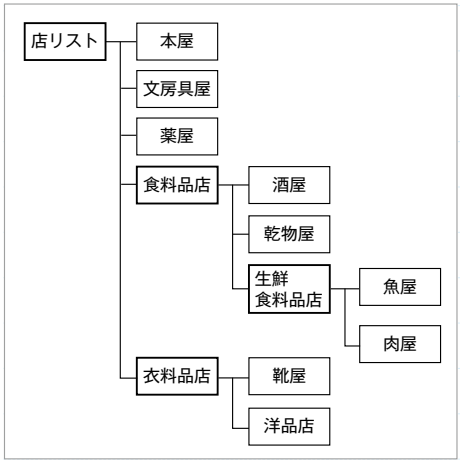
データがきれいに階層的に管理されている場合、それほどたくさんの操作をしなくても必要

▼**図1**　Macのファインダでアプリケーションを段階的に検索しているところ



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

▼図2　あるファイルシステムの階層例



な情報を見つけることができるのが普通です。実際、パソコンのデスクトップ画面でマウスを使って階層のトップから階層をたどって特定のファイルを捜すのに必要な操作は多くても10回程度でしょう。

キーボードで階層構造をナビゲーションする

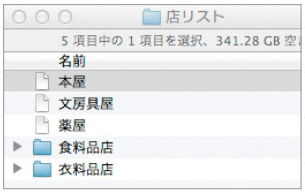
図2のような階層を持つファイルシステムをキーボードでナビゲーションすることを考えてみましょう。

これがファイルシステムの構造になっている場合、Macのファインダでは▲▼◀▶という4個の矢印キーでナビゲーションを行うことができます。

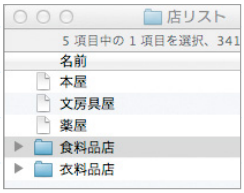
「店リスト」をファインダで表示して、「本屋」を選択すると、表示は図3のようになり、3回▼を押すと、図4のように「食料品店」が選択されます。

「食料品店」は下位階層を持っているので、ここで▶キーを押すと図5のように下位階層が表示されます。さらに▼キーを押すことによって「酒屋」を選択したり、「生鮮食料品店」を選択してから▶を押すことによって、図6のように下位階層を表示できます。

▼図3　「店リスト」をファインダで表示して「本屋」を選択



▼図4　3回▼を押して「食料品店」を表示



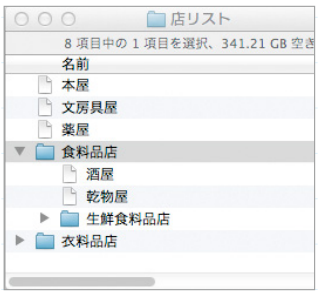
また、図6の状態で◀キーを押すと下位層の表示を消し、図5のような状態に戻すことができます。このように、Macのファインダでは▲▼◀▶という4個のキーを使って階層データのナビゲーションを行うことができます。テレビのリモコンやジョグダイヤルでも階層構造データのナビゲーションのためにほぼ同様の手法が利用されていることが多いようです。

Gear──2つのキーだけで階層構造をナビゲーション

前述のように4個のキーを使って階層データのナビゲーションを行うシステムはたくさんありますが、実はキーを2つしか使わなくても同様のナビゲーションを行うことができます。

Gearでは▲と▼という2つのキーだけを利用してナビゲーションを行います。Gearで「店リスト」を表示すると、ファインダの場合と同じリストが表示されます(図3)。▼を3回押すと前の例と同じように「食料品店」が選択されますが(図7)、そこで操作を中断して一定時間待つと「食料品店」の下位層が自動的に展開され、

▼図5　「食料品店」で▶キーを押して下位の階層を表示



▼図6　「生鮮食料品店」で▶キーを押して下位の階層を表示

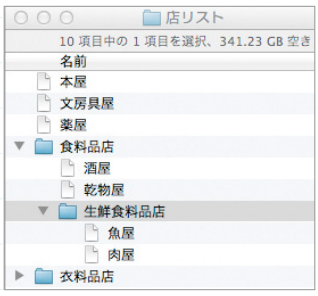


図8のようにその最初の要素が選択されます。

ここで▼を2回押して「生鮮食料品店」を選択したまま一定時間待つと、図9のように下位層が自動的に展開され、最初の要素である「魚屋」が選択されます。

つまり、▶のようなキーを押さなくても、一定時間待つことによって同様の効果が得られることになります。

図7のように食料品店を選択した状態から時間を置かずに▼を押すと、下位層は展開されず、図10のように「衣料品店」が選択されます。またここで操作を止めて一定時間待つと下位層が自動的に展開され、図11のように「靴屋」が選択されます。

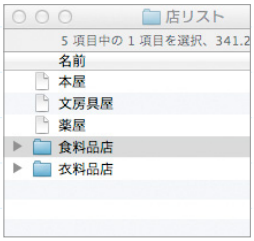
図9の状態から▲を押すと、下位層は自動的に閉じられて図8の状態に戻ります。さらに▲を押すと「食料品店」の下位の層も閉じられ、図7の状態に戻ります。また、図9の魚屋が選択されている状態から▼を2回押すと、「食料品店」の下位層は自動的に閉じられて図10の状態になります。

まとめると、

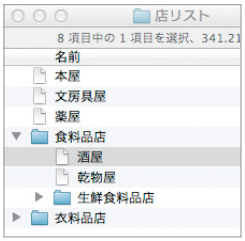
・選択した項目に下位層が存在するときキー入力を行わずに待つと下位層が自動的に展開され、下位層の最初の項目が選択される
・項目リストの端を選択しているとき、さらに▲▼を押すと下位層は閉じられて1つ上の層の項目が選択される

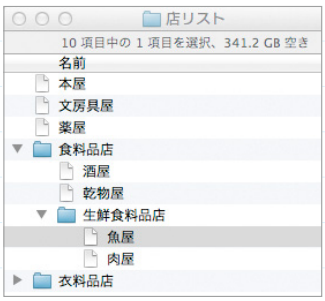
という2つの工夫により、▲と▼だけで階層データを自由にナビゲーションすることが可能になるというわけです。

▼図7 「店リスト」から「食料品店」を選択する



▼図8 一定時間で下位階層が表示される





◀図9 一定時間で下位階層である「魚屋」が表示される

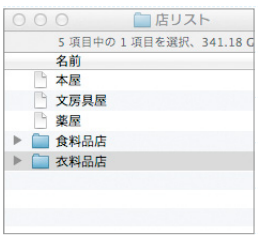
ブラウザでの実装

前述の例は小さな階層的データの上でのGearナビゲーションでしたが、実際には巨大な階層データでも同様のナビゲーションができます。図12のスクリーンショットはブラウザ上に実装したGearを使って各種コンテンツをナビゲーションしているところです。

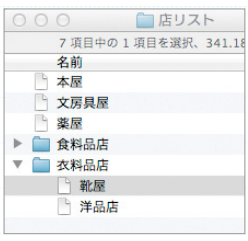
Gearの利点

入力装置は単純であるにこしたことはありま

▼図10 すぐに▼を押すと「衣料品店」が表示される



▼図11 「衣料品店」で少し待つと「靴屋」が表示される



▼図12 WebブラウザへのGear実装

せん。2個のセンサであらゆる操作が可能なのであれば左右の回転操作や何かを押したり引いたりする単純な装置でも操作が可能だということになるので、さまざまな環境で利用が可能だということになります。Gearの操作は▲▼という2つのキーしか必要としないため、圧力センサや回転ダイヤルなどを利用した各種の実装ができます。

たとえばMacの入力装置として**写真1**の「HyperMate」という製品が市販されており、これをGearの入力装置として利用できます。

また**写真2**は、パドルに貼った2個の圧力センサの値を▲▼に割り当てています。板を押したり引いたりすることによってコンテンツのナビゲーションができます。

このように、現状のGUIでは利用されていないような単純なデバイスでもGearの入力装置として利用できるのが大きなメリットです。

Gearの利用環境

Gearの操作には2個の入力装置しか必要としませんから、普通の入力装置が使いにくいような場所でも利用できます。たとえば食卓の裏にさりげなくスイッチや圧力センサを2つ並べておけば、それらを操作するだけでBGMや番組を切り替えることができるでしょう。テレビの前のソファの手すりにローラーを付けておけば、それを回すだけであらゆるコンテンツを選択できるでしょう。

私は自宅の机の前のサブモニタで常にGearブラウザを動かしており、マウスホイールを使ってコンテンツを選択して視聴しています。パソコンで音楽や動画を再生しながら仕事をしている人は多いと思いますが、手近にGear端末があればマウスホイールを回すだけであらゆるコンテンツを簡単に視聴できるのはとても快適です。

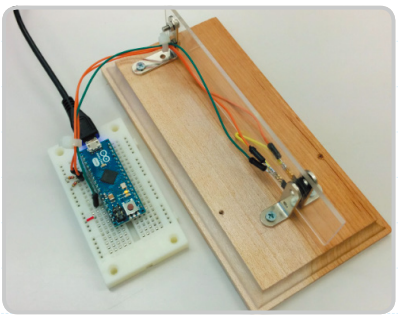
たくさんのボタンを必要とする入力装置ではどうしてもボタンに合わせた設計が必要になり、装置の形状が制約を受けてしまいますが、Gearの場合にはそもそもどこでコンテンツを視聴したいのかを最初に考え、その場所に最も適した入力装置の形態を考え、それらをGearのコントローラに利用するという順番でデザインができることになります。これまでの電子機器は機械の都合にあわせて装置の形状がデザインされるのが普通でしたが、それを根本的に改善できる可能性があると言えるでしょう。

Gearを見たとき「これの何が面白いの？」という顔をする人がいます。確かに、一度Gearを使ってみるとGearの挙動はまったく自然に感じられるので、どこが新しいのかわからなくなってしまうのかもしれません。しかし、これまでボタン2個だけで階層的データをナビゲーションする方法が普及していないことを考えると、Gearはまさに「コロンブスの卵」のような手法だと言えるでしょう。

Gearは確かにコロンブス的要素を持つインタラクション手法であり、どこでも簡単に情報検索を行うための強い味方となります。Gearのためのオシャレな入力デバイスを考えていきたいと思っています。SD

▼写真1　HyperMate

▼写真2　圧力センサを利用した入力装置



第7回 北奔南走!? OSC北海道と沖縄

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

北海道と沖縄は12回目のOSC開催

断酒して1ヵ月半が経ちました。83.6キロから80.4キロと3.2キロ減。おおむね予定に近いペースで減量できています。ただ、この6月から7月に、北海道、そして沖縄と、文字どおり北へ南へ飛び回り、かつ美味しい物をたくさん食べてしまったりしたので、想定以上の減量とはいきませんでした。

北海道と沖縄は、OSC初開催の翌年2005年から毎年開催されており、今回で12回目の開催です。東京一極集中を少しでも解消しようと、少し極端ですが日本の北と南の端でスタートしたOSCが、これまで順調に開催されているのは感慨深いものがあります。北海道は土曜日開催+金曜日に少しだけセミナーを開催する「1.5日開催」ですが、参加者は合計で700名を超える、東京・京都に続く規模の開催地域です。北海道のみなさんがOSCを大事にしていること、さらに道外から「北海道なら行ってみたい」と思えるところが魅力です。そのようにして、どんどんとほかの地域へと出かけて行って、交流を深めてもらえればと思っています。

北海道といえば生ラムジンギスカン

北海道の魅力はやはり「食」でしょう。OSC開催前日にスタッフが集まる「石鍋亭」というお店があります(**写真1**)。新鮮な生ラムジンギスカンと、野菜タップリのモツ鍋が美味しい店で、なぜかOSC北海道のスポンサーでもあります。2005年のOSC北海道開催も、この石鍋亭で飲んでいたことがきっかけとなった、OSCとは縁の深い店でもあります。すすきの交差点に近い好立地ですので、札幌への出張、観光の際にはぜひお立ち寄りください。

OSC北海道終了後の懇親会はサッポロビール園でのジンギスカンパーティー(170名以上参加!)ですが、前夜祭の石鍋亭、さらにその前日も石鍋亭でジンギスカンを食べていて、今回のOSC北海道はジンギスカン三昧でした(**写真2**)。もう少しラーメンやスープカレーを採り入れてもよかったかもしれませんね。その反省を活かして、OSC翌日の朝は二条市場に海鮮丼を食べに行きました(詳細はコラムにて)。

旭川、富良野から高校生を招待

今回のOSC北海道では、旭川工業高校から19名、富良野緑峰高校から7名の高校生が招待参加してくれました(**写真3**)。札幌から旭川までバスで2時間、富良野はさらにその南ですから、本当にはるばる参加してくれました。

参加したみなさんは、朝から夕方までセミナー、展示に参加し、刺激を受けて帰ってくれたようでした。富良野緑峰高校からは4名が懇親会まで参加し宿泊して、翌日は北海道

▼写真1 お世話になっている石鍋亭のマスターと。スタッフポロシャツいただきました

▼写真2 ジンギスカン懇親会終了後、会場前で記念撮影。これでも参加者のごく一部です

▼写真3 懇親会終了後、富良野緑峰高校のみなさんと。高校生らしくジャージです(笑)

大学のキャンパス見学をさせてもらったとのこと。これをきっかけに、いろいろなことに挑戦してもらえればと思います。

全国各地のOSC関係者が沖縄に集結

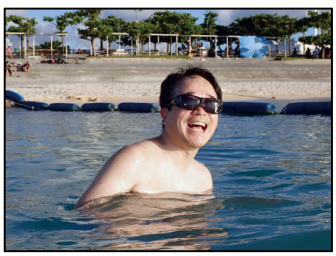
OSC沖縄の開催前日、「OSCサミット@沖縄」に全国各地のOSC実行委員が集まり、各地域の現状報告会を行いました(**写真4**)。各地域共通の課題として、「コミュニティ活動をどうやって活性化するか」「メンバーをどうやって若返りさせるか」ということが挙げられました。たしかに、OSCで知り合ったアクティブな学生や若者が、就職や転職で東京に出てくることが多いと感じます。しかし筆者は、若いうちは東京などで修行して、経験を積んだ後に地域に戻ることはできないかと考え続けています。5〜10年という期間で考えるので今、課題を感じている人には何の解決策にもなりませんが、OSCを始めてまだ10年と少し。蒔いた種が将来芽吹き、花咲くことを期待しながら毎回全国を飛び回っているわけです。若者が挑戦する機会、成長する場を与えて、将来故郷に貢献できる、そんなサイクルを生み出せたらいいですね。

夕陽を見ながらビーチパーティー

OSC沖縄の会場は目の前に人工ビーチがあり、泳いだり(**写真5**)、バーベキューができるので、今回は終了後にビーチパーティーを開催しました。当日は最高の天気で、最高にきれいな夕陽を眺めながら泳いだり、美味しいオリオンビールやサッポロクラシック(北海道からの差し入れ)を飲みながら、交流を深めることができました。OSCサミットに参加した各地域の実行委員は、お互いの地域の開催に協力し合う密約(?)を交わせたようで、大きな成果が得られた2日間でした。SD

▼写真4　OSCサミット後の懇親会で琉装の店員さんと。あれ?沖縄でも石鍋亭??

▼写真5　OSC沖縄終了後、会場目の前のビーチで泳ぎました。沖縄開催ならではですね



Report

どーんと北海道海鮮丼

ジンギスカンばかり食べていたのを反省して、Facebookで「朝から二条市場に行ってくるけど、いい店ないかな?」と書いておいたら、石鍋亭のマスターが「近藤昇商店がいいよ!」と教えてくれたので早起きして行きました。二条市場はすすきのから歩いて10分ぐらいで、大通り公園からも、ほど近い場所にあります。

目的のお店を発見して、お勧めの「けいらん丼」を注文。普通は「鶏卵」ですが、北海道では「鮭卵」、要するにイクラです。出てきてビックリ、北海の海の幸がどんぶりから完全にはみ出しています。「朝ご飯にこれでいいの?」というぐらいのボリューム感です。また、「石鍋亭のご紹介で」と伝えたところ、デザートに完熟メロンをサービスしてもらえました。北海道といえばやっぱり海の幸。朝早くからやっているので、来年のOSC北海道は朝、ここで海鮮丼を食べてから会場に向かおうと誓ったのでした。

▲これが注文した「けいらん丼」。具◀がはみ出ていて、相当引かないと全部写りません

▲地元のお店なので、観光客相手っぽい店が苦手な私も納得の雰囲気です

ツボイのなんでもネットにつなげちまえ道場

第15回

Universal Serial Bus（デバイス編）

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

USB

Universal Serial Bus（USB）は、コンピュータの外部バスの中で、最も使われているものでしょう。基本的にUSBは、パソコンと周辺機器を接続する規格です。USBでは、「ホスト」であるパソコンと、「デバイス」であるキーボードやマウス、プリンタなどの周辺機器、と役割が分かれています。USBでの通信は、ホスト側からの働きかけにより開始されます。ですので、デバイス同士やホスト同士を接続するといったことはできません。

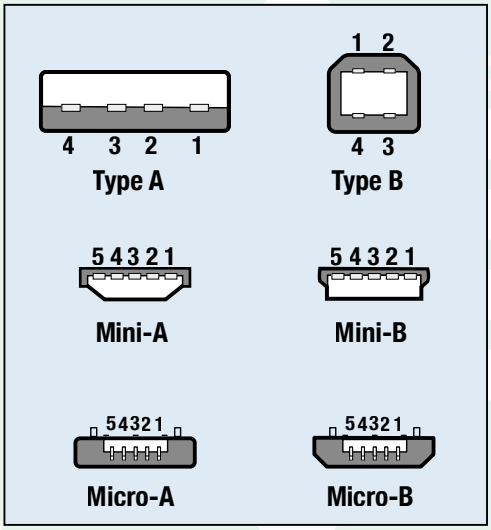
最近のスマートフォンやタブレットは、パソコンなどのホストに接続することも、キーボードやUSBメモリなどのデバイスを接続することもできます。これを実現しているのが、USB On-The-Go（OTGと略されます）という規格です。標準的なUSBコネクタには、5VとGND、D+とD-という信号線の合計4つの端子があります（**図1**）。

ミニやマイクロUSBコネクタでは、これにIDという信号線を加えた5つの端子があります。このIDはOTGに使われます。このIDという信号がGNDにつながっていると、OTGに対応したデバイスはホストの役割をします。ですので、OTGに対応したケーブルのホスト側は、**図2**のようにIDとGNDが接続されています。

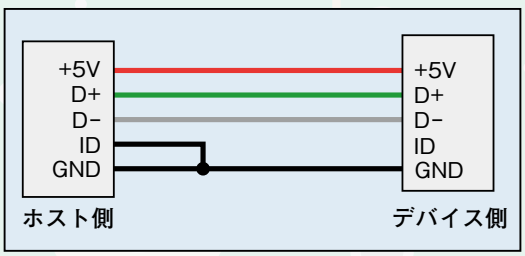
先ほどUSBの信号線はD+とD-という名前だということを記しました。USBでは、このD+とD-という2本の線の間の「差動信号（さどうしんごう）」で伝送をします。差動信号は、「差動」というその名のとおり、1つの信号をD+とD-という2つの信号線の電位差で表します（**図3**）。

1つの信号を、2本の信号線で送るという、一見非効率な方法を採るのには理由があります。大きな理由の1つはノイズです。信号線に外部からノイズが加わったとき、D+とD-に同じようなノイズが加われば、差動信号ではD+とD-の電位差をみているため、ノイズがキャンセルされます。**図3**中の赤い矢印の長さは、電位差を表しています。同じ大きさのノイズが作動信

▼図1　USBコネクタの種類



▼図2　OTGケーブルの結線例



号に入った場合、矢印の長さ(電位差)は、元の信号と変わりません(**図3下**)。

この「差動」というしくみは、高速な伝送を行う信号線で用いられており、USBだけでなく、EthernetやHDMIなどにも用いられています。

話がそれてしまいました。マイコンにも、このUSBのインターフェースが搭載されているものがあり、デバイスにのみ対応しているマイコンや、ホストにも対応しているマイコンなどいろいろな種類があります。mbed LPC1768にもUSBのインターフェースが搭載されていて、デバイスだけでなく、ホストやOTGにも対応しています。今回は、mbed LPC1768のUSBインターフェースを使ってみましょう。とりあえず、今回はmbedにUSBのデバイスの役割をさせてみようと思います。

デバイスクラス

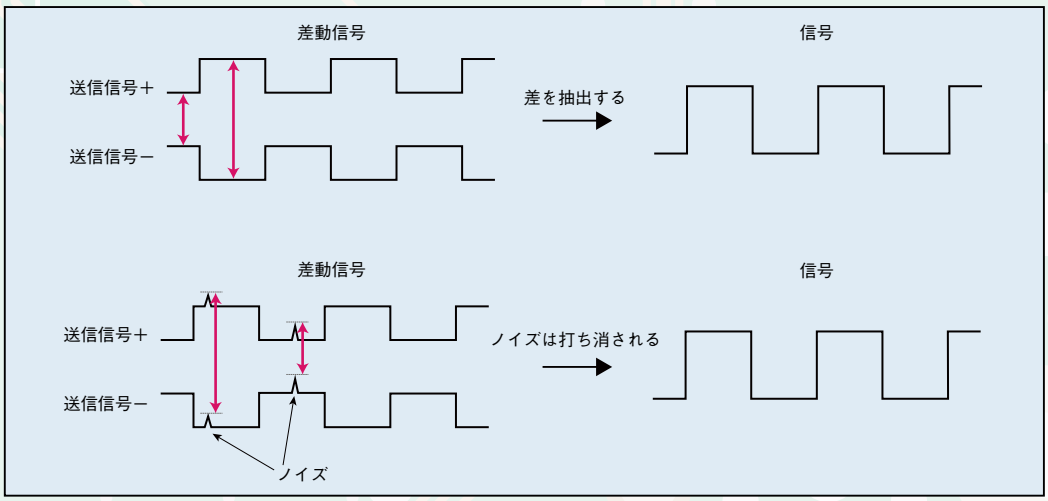
USBでは、周辺機器の機能によってグループ分けされたデバイスクラスと呼ばれる仕様が定義されています。キーボードやマウスであればヒューマンインターフェースデバイス(HID)、USBメモリはマスストレージデバイスといったクラスがあることはみなさんご存じでしょう。パソコンのOSには、この「クラス」に対応したドライバ、「クラスドライバ」があらかじめ含まれていますので、キーボードやUSBメモリを接続したときに、あらためてドライバをインストールする必要がないケースも多くあります。このクラスドライバでは十分にデバイスをコントロールできないデバイスの場合、個別にデバイスドライバが提供されています。

マイコンでは、WindowsやLinuxのように多くのクラスドライバを搭載したOSを使うことができない場合がほとんどです。ですので、USBのホスト機能を使うプログラムを作るときには、それぞれのデバイスやクラスに対応したプログラムも用意しなければなりません。mbedでは、いくつかのクラスに対応しているライブラリが提供されています[注1]。こうしたオフィシャルのライブラリとは別に、USBで接続するBluetoothのアダプタのライブラリ、BlueUSB[注2]なども存在します。

今回はmbedにUSBのデバイスの役割をさせます。デバイスも、デバイスクラスやホスト側のドライバを開発するのであれば独自の振る舞いをしなければなりませんので、振る舞いに応じたソフトウェアを書かなければなりません。mbedでは、デバイスとして動作するためのライ

注1）https://developer.mbed.org/handbook/USBHost
注2）https://developer.mbed.org/users/peterbarrett1967/code/BlueUSB/

▼図3　差動信号の例



ブラリも提供されています[注3]。

マウスを作ってみる

では、実際にmbed LPC1768をUSBデバイスにしてみましょう。ホスト側にドライバがいらなく、簡単に試せるのはHIDデバイスでしょう。また、キーボードと違って、マウスはクリックさせなければマウスポインタが動くだけですから気軽に実験できます。USBMouse_Hello World[注4]というサンプルプログラムがおあつらえ向きですので、これを試しに動かしてみましょう。

まず、ハードウェアの準備をしましょう。今回も、mbed LPC1768[注5]とmbedアプリケーションボード[注6]を使います。**写真1**のように、mbedアプリケーションボードには、複数のUSBレセプタクル[注7]が搭載されています。これらのレセプタクルは用途が異なり、mbedにプログラムを書き込むときなどに使用するUSBが1つ、mbedをUSBホストやデバイスにするためのレセプタクルが1つずつあります(**写真1**)。

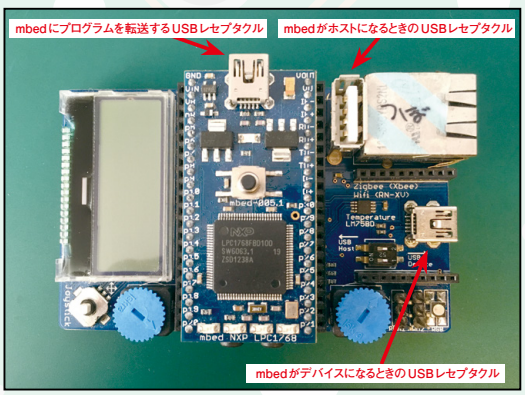
と言っても、mbed LPC1768にUSBのバスが2つあるわけではありません。プログラムを書き込むために使うレセプタクルは、mbed LPC1768の底面に付いているLPC1768とは別のマイコンとつながっています。mbed LPC1768の主要なマイコンであるLPC1768のUSBバスは、アプリケーションボードの2つのUSBレセプタクルに接続されています。

LPC1768にはUSBバスが1つなのに、レセプタクルは2種類付いていることには理由があります。もともと、USBのA端子類はホスト機器に、B端子類はデバイス側の機器に付いていました。しかし、先ほど紹介したOTGの登場などにより、普段はデバイス側の機器がホスト側の役割をでき、Micro-ABといった規格のレセプタクルも存在します。

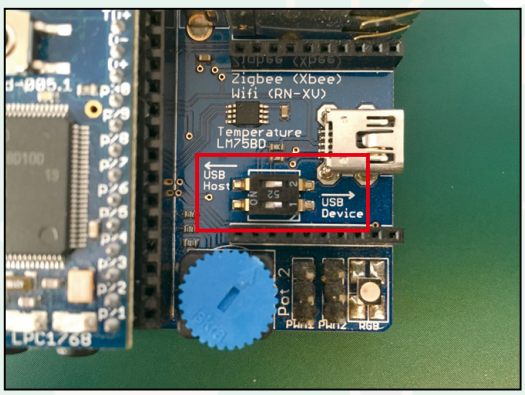
また、アプリケーションボードには、小型のスライドスイッチが付いています。片方には「USB Host」、もう一方には「USB Device」と書かれています。これは、USBバスを15kΩの抵抗でプルダウンするかどうかを切り替えるためのスイッチです。規格で、USBのホスト側の機器のD+とD-の信号線は、15kΩでプルダウンすると決まっています。今回は、mbed LPC1768をUSBのデバイス機器として使いますので、「USB Device」と書かれているほうに切り替えておいてください。**写真2**の状態が、デバイスに使うときの状態で、プルダウンが行われていません。スイッチの切り替えは、ボールペンの先端など、先がほどほどに細いものを使って行

注3) https://developer.mbed.org/handbook/USBDevice
注4) https://developer.mbed.org/users/samux/code/USBMouse_HelloWorld/
注5) http://ssci.to/250
注6) http://ssci.to/1276
注7) レセプタクルというのは、基板や機器側のコネクタのことです。ケーブルの先端のコネクタは、プラグと呼ばれます。

▼写真1　mbedアプリケーションボードのレセプタクル



▼写真2　ホストとデバイスの切り替えスイッチ



います。

次にソフトウェアです。この連載をずっと読んでくださっている方なら、すでに説明は不要でしょう。先ほどのUSBMouse_HelloWorldのページを開き、「Import this program」というボタンをクリックします。すると、オンラインコンパイラの自分の環境に、このプログラムを読み込むことができます。このとき、Updateのチェックボックスにはチェックを入れないようにしてください。あとは、オンラインコンパイラの画面右上にあるターゲットが「mbed LPC1768」になっていることを確認して、「コンパイル」ボタンをクリックすれば、コンパイルが終わり、プログラムのバイナリファイルがダウンロードされます。mbed LPC1768のUSBレセプタクルとパソコンを接続し、「MBED」というドライブにダウンロードしたバイナリファイルをコピーしてください。コピーを終えたら、「MBED」ドライブを取り外して、USBケーブルも取り外します。

今回は、mbed LPC1768をUSBのデバイス機器として使いますので、Temperatureと書かれている温度センサの横にある、Mini-Bのレセプタクルを使います。先ほど外したUSBケーブルを、このレセプタクルに差し込んでください。すると、マウスのポインタが動きます。ここで、このデバイスが手元のパソコンでどう認識されているかを見てみると、**図4**のようになっていました。「製造元ID」が0x1234、「製品ID」が0x0001となっています。

VID、PID

この製造元ID（ベンダID、VID）と製品ID（プロダクトID、PID）は、ホストが接続されたデバイスを識別するために使われます。WindowsなどのOS用にデバイスメーカーから提供されているデバイスドライバには、これらのIDが書かれています。製品IDは製造元が個別に割り振るのですが、製造元IDは、USBインプリメンターズフォーラム[注8]という団体に申し込んで割り当ててもらいます。

先ほどのサンプルプログラムでは、勝手に0x1234というVIDが使われていましたが、これはほかの企業に割り当てられているIDですので、このままデバイスを出荷したりしてはいけません。先ほどのサンプルプログラムであれば、4行目のコンストラクタで、次のように初期化をします。

```
USBMouse mouse(REL_MOUSE, 0x2786, 0xf000);
```

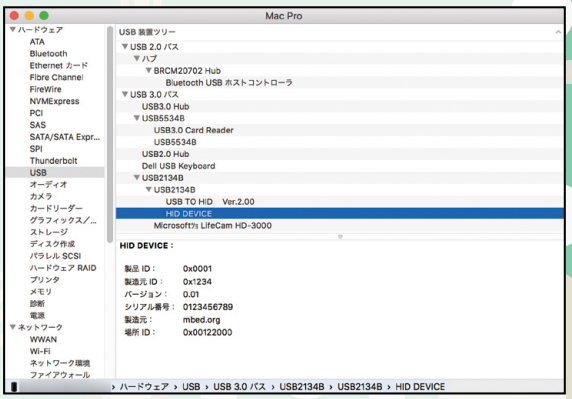
0x2786という製造元IDはスイッチサイエンスに割り当てられたもので、0xf000というプロダクトIDは社内での評価用などに使っているものです。

まとめ

このように、mbedを使えば手軽にUSBデバイスのプロトタイプも作れます。mbedに対応したボードはたくさんありますが、マイコンにUSBの機能が搭載されていなければなりませんし、また、mbedのSDKも対応している必要があります。このような都合で、手軽にUSBデバイスを作れるmbed対応ボードは、LPC1768、LPC11U24、FRDM-KL25Zの3機種だけが掲載されています。SD

注8）http://www.usb.org/developers/vendor/

▼図4　今回作ったUSBデバイスをMacで確認してみた



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9月15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 ザ・ワイヤレス サイレント マウス＆キーボード MK48367G 1名

Windows用ワイヤレスキーボード＆マウス。キーボードにはサイレントキーキャップ、マウスにはサイレントスイッチを採用した静音仕様です。キーボードにはさらに排水機構も備え、飲みものをこぼしてもすぐに掃除可能。ユニークが規格した2.4GHz帯のワイヤレス電波相互ペアリング機能「InteLink」を採用し、レシーバ1つで接続できます。電源は、ともに単3電池1本。

提供元 ユニーク http://www.uniqstyle.co.jp

02 裸族の頭 HDD/SSD引越キット CRAHK25U3 1名

Windowsマシンのシステムドライブを USB3.0経由で、外付け2.5インチSATAのHDD/SSDにまるごと引っ越しできるキット。小容量⇔大容量のコピーも可能で、外付けドライブ接続用にも利用できます。対応OSはWindows 10/8.1/8/7/Vista。

提供元 センチュリー http://www.century.co.jp

03 Spark Summit Tシャツ

2016年6月にサンフランシスコで行われた、ビッグデータ処理基盤Apache Sparkに関するイベント「Spark Summit 2016」のノベルティTシャツ。サイズはLとなっています。

提供元 Spark summit https://spark-summit.org 1名

04 Python 機械学習プログラミング

Sebastian Raschka 著

昨今の人工知能分野の盛り上がりにより、機会学習の注目度が大きく上がっています。本書では、機械学習の各理論を端的に解説、各種ライブラリを使ったPythonによる実装を説明しています。

提供元 インプレス https://www.impress.co.jp 2名

05 インフラエンジニアになるための教科書

寺尾 英作 ほか 著

「単なる技術書ではなく、インフラエンジニアの仕事を支える現場の知識を多くのインフラエンジニア志望者に伝えたい」が、本書のコンセプト。時代によるインフラの変遷にも触れながら解説していきます。

提供元 ソシム http://www.socym.co.jp 2名

06 プログラミング言語 Go

Alan A.A. Donovan ほか 著

Go言語は、Googleによって開発された新しいコンパイラ言語で、エンジニアからの人気を集めています。本書はGoの言語機能と標準ライブラリの長所を、最大限活用できるように書かれた1冊です。

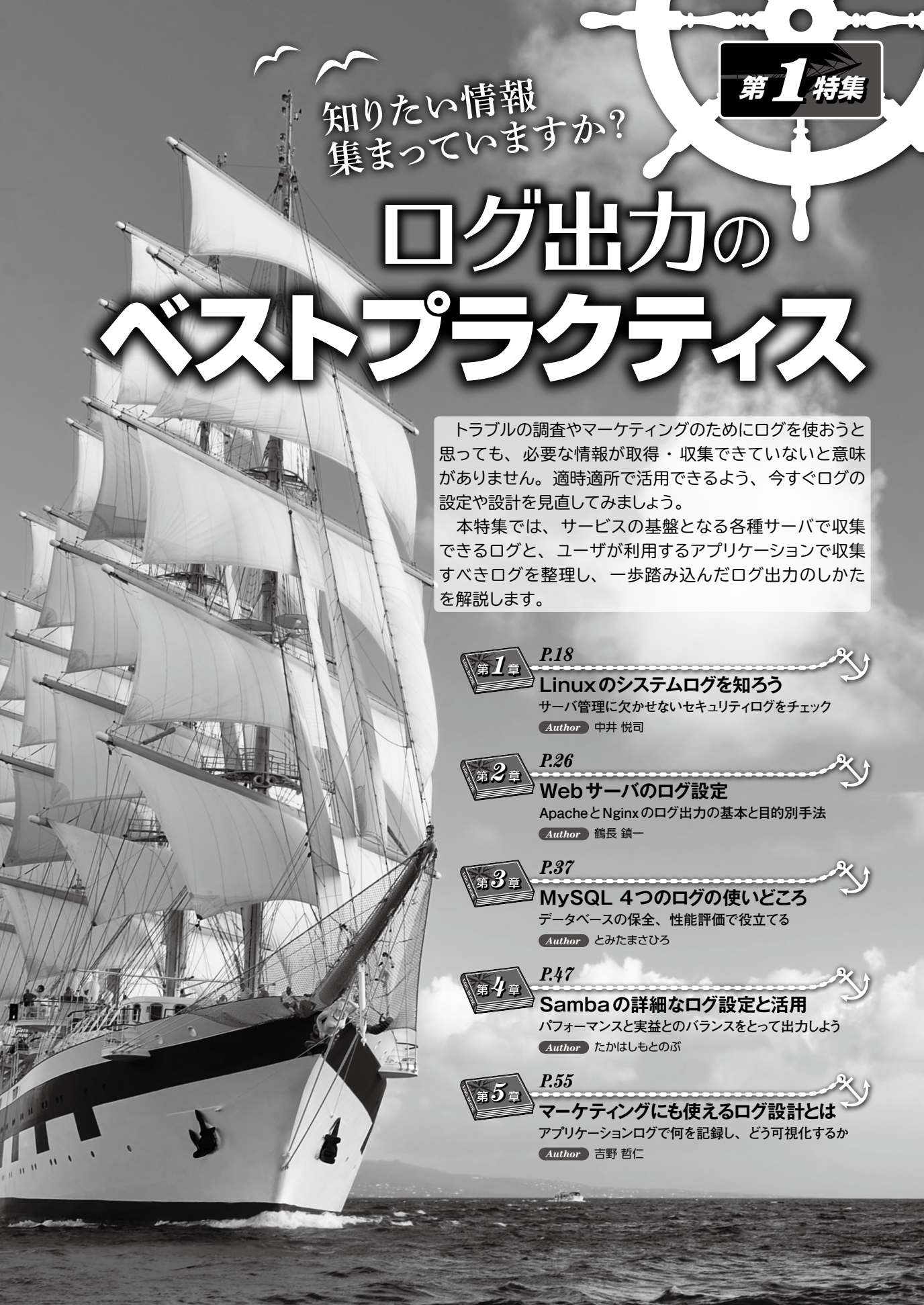
提供元 丸善出版 http://pub.maruzen.co.jp 2名

07 Slack 入門

松下 雅和 ほか 著

今最も注目を集めるチャットコミュニケーションツール「Slack」の入門書。ChatOpsの考え方に触れながら、初めてSlackを利用する方に向けて、基本操作、Hubot・CIツールとの連携方法を解説します。

提供元 技術評論社 http://gihyo.jp 2名



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

トラブルの調査やマーケティングのためにログを使おうと思っても、必要な情報が取得・収集できていないと意味がありません。適時適所で活用できるよう、今すぐログの設定や設計を見直してみましょう。

本特集では、サービスの基盤となる各種サーバで収集できるログと、ユーザが利用するアプリケーションで収集すべきログを整理し、一歩踏み込んだログ出力のしかたを解説します。

第1章

Linuxのシステムログを知ろう

サーバ管理に欠かせないセキュリティログをチェック

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

本章ではLinuxサーバのシステムログについて、CentOS 7（systemd）を題材としてログ出力のしくみを解説します。また、ログ確認において欠かすことのできないセキュリティの観点で、ログの見方や設定方法を紹介します。

Linuxのシステムログ

本特集は、OS、ミドルウェア、アプリケーションなど、サーバ上で稼働するさまざまなコンポーネントの「ログ出力」がテーマです。最近は、クラウドで大量の仮想マシンを起動するシーンも増えたため、多数のマシンからのログをどのように収集・分析するかで頭を悩ますこともあります。しかしながら、まずは、1台のサーバ上でどのようなログが出力されるのか、あるいは、どのようなしくみで出力されるのかを理解することが大切です。

第1章では、Linuxサーバのシステムログにフォーカスして、Linuxにおけるログ出力の基本的なしくみを説明します。さらに、OSが出力するログの中でも、とくに重要となるセキュリティログを取りあげて、ログの見方やログ出力の設定例を紹介します。

なお、systemd導入以前のLinuxを使っている方は、「rsyslogdのしくみ」以降の内容を実務の参考にしてください。また、今後はsystemdの環境が増えると思いますので、この機会にjournaldのしくみも理解しておくとよいでしょう。

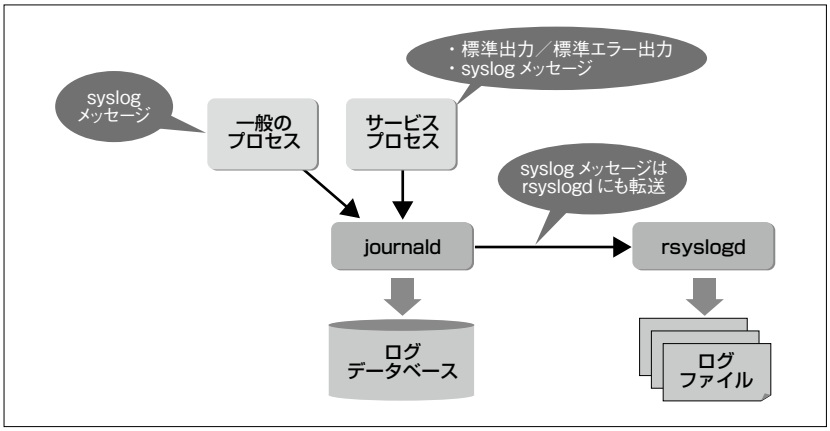
rsyslogdとjournaldの連携

Unix/Linuxのシステムログは、伝統的に「syslogデーモン」によって管理されてきました。ただし、Linuxの進化に伴って、syslogデーモンの実装も進化しています。ここでは、CentOSの最新バージョンであるCentOS 7が採用する、journaldとrsyslogdによるシステムログ管理のしくみを説明します。

CentOS 7では、**図1**に示した、journaldとrsyslogdの連携処理によって、システムログの管理が行われます。一般に、Linux上で稼働するさまざまなサービス（デーモン）やアプリケーションは、syslog関数（syslogライブラリコール）を用いてログを出力します。このログは、journaldが受け取って、独自のデータベースに保存した後、rsyslogdに転送されます。その後、rsyslogdは設定ファイル「/etc/rsyslog.conf」に従って、各種のログファイルにメッセージを記録します。従来、syslog関数で出力されたメッセージは、rsyslogdが直接に受け取っていましたが、その間にjournaldが入る点がこれまでとの違いになります。

それでは、journaldが間に入ることにはどのような意味があるのでしょうか？ これは、systemdが管理する各種サービスのログ出力と関係があります。CentOS 7の環境では、システム標準の各種デーモンやRPMパッケージで提供される標準的なアプリケーションは、systemdのサービスとして管理が行われます。journaldはこれらのサービスから受け取ったロ

▼図1　journaldとrsyslogdの連携処理



グをデータベースに保存する際に、サービスの種類やその他の独自の追加情報をメタデータとして記録します。その結果、システム管理者はjournalctlコマンドを用いて、特定サービスのログを検索することが可能になります。

ログの使い分け

rsyslogdでは、1つのログファイルに複数のサービスのログが混在したり、あるいは逆に、1つのサービスのログが複数のファイルに分かれて記録されることがあります。「セキュリティに関連したログ」など、目的別にログを見る場合はこのほうが便利なこともありますが、特定サービスのログだけをまとめて見る際は少し不便なこともありました。サービスが障害停止した際の問題判別など、特定サービスのログを詳細に確認する際は、journalctlコマンドによるログ検索機能が役に立ちます。

journaldのログ活用

journalctlコマンドを用いて、journaldがデータベースに保存したログを検索する方法を解説します。まず、journalctlコマンドをオプションなしで実行すると、データベース内のすべてのログをまとめて出力します。

```
# journalctl
```

この際、デフォルトではlessコマンドを用いて結果が表示されますので、↑↓キーでログを上下にスクロールして閲覧することができます。画面の右端からはみ出した部分は、←→キーで左右に移動して確認します。lessコマンドが不要な場合は、次のように、--no-pagerオプションを指定します。

```
# journalctl --no-pager
```

特定のサービスのログだけを見るときは、-uオプションでサービス名を指定します。次は、sshd（SSHデーモン）のログを確認する例になります。

```
# journalctl -u sshd.service
```

また、tailコマンドの-fオプションのように、ログに新しいメッセージが追加される様子を観察する場合は、journalctlコマンドに-fオプションを指定します。

```
# journalctl -f -u sshd.service
```

さらに前述のように、journaldはさまざま追加情報をメタデータとして記録しています。「-o

▼図2　journaldが保存するログのメタデータ

```
# journalctl -u sshd.service -o json-pretty --no-pager
..省略..
{
        "__CURSOR" : "s=983ed0ac18074767abcf065afd0e013e;i=3ff;b=e9b62b84a9d04c4487afbaadc
2707ac9;m=15315e4;t=5346b45a3d666;x=93ba269f21ce2698",
        "__REALTIME_TIMESTAMP" : "1465010218063462",
        "__MONOTONIC_TIMESTAMP" : "22222308",
        "_BOOT_ID" : "e9b62b84a9d04c4487afbaadc2707ac9",
        "PRIORITY" : "6",
        "_UID" : "0",  ……❶
        "_GID" : "0",  ……❷
        "_SYSTEMD_SLICE" : "system.slice",
        "_MACHINE_ID" : "ea103ae2d481cfdc698e47636b9fb861",
        "_CAP_EFFECTIVE" : "1fffffffff",
        "_TRANSPORT" : "syslog",
        "SYSLOG_FACILITY" : "10",
        "_HOSTNAME" : "etsuji-tfbook01",
        "SYSLOG_IDENTIFIER" : "sshd",
        "SYSLOG_PID" : "1325",
        "_PID" : "1325",
        "_COMM" : "sshd",
        "_EXE" : "/usr/sbin/sshd",
        "_CMDLINE" : "/usr/sbin/sshd -D",
        "_SYSTEMD_CGROUP" : "/system.slice/sshd.service",
        "_SYSTEMD_UNIT" : "sshd.service",
        "_SELINUX_CONTEXT" : "system_u:system_r:sshd_t:s0-s0:c0.c1023",
        "MESSAGE" : "Server listening on :: port 22.",
        "_SOURCE_REALTIME_TIMESTAMP" : "1465010218062336"
}
..省略..
```

json-pretty」オプションを追加すると、これらのメタデータをJSON形式で出力することが可能です。**図2**の出力例を見ると、メッセージを出力したプロセスのUID/GID（**図2の❶、❷**）など、通常のログメッセージでは確認できない情報が含まれていることがわかります。

なお、先ほどの**図1**では、journaldは、syslogメッセージ、すなわち、syslog関数で出力されたログメッセージを受け取るように描かれていますが、実際にはこのほかのメッセージも受け取るようになっています。1つは、systemdのサービスとして起動したプロセスが標準出力、もしくは、標準エラー出力に書きだした内容です。もう1つは、journaldが独自のAPIで受け取ったメッセージです。これまで、デーモンプロセスが標準出力／標準エラー出力に書きだした内容はそのまま捨て去られることが多かったのですが、systemd/journaldの環境では、これらの内容もログデータベースに記録されており、journalctlコマンドで確認できるようになっています。

ログデータベースの永続保存

journaldのログデータベースは問題判別の際はとくに有用な機能ですが、CentOS 7の環境では注意点が1つあります。インストール時のデフォルト設定では、ログデータベースの内容はディレクトリ「/var/run/log/jounral」以下に保存されており、これは一時ファイルを保存するRAMディスク領域にあたります。つまり、

▼表1　Linuxの主要なシステムログファイル

ログファイル	説明
/var/log/messages	システム関連ログのデフォルトの出力ファイル
/var/log/secure	ユーザのログイン認証など、セキュリティ関連情報を記録
/var/log/cron	cronジョブの実行履歴を記録
/var/log/dmesg	システム起動直後のカーネルログバッファの内容を記録
/var/log/wtmp	ユーザのログイン履歴を記録するバイナリファイル。lastコマンドで参照
/var/log/lastlog	ユーザの最終ログイン記録を保管するバイナリファイル。lastlogコマンドで参照
/var/run/utmp	ログイン中のユーザ情報を保管するバイナリファイル。uptimeコマンド、wコマンドなどで参照

journaldのログデータベースはOSを再起動すると内容が失われてしまいます。あくまで、システム起動後に出力されたログしか参照することができません。

ログデータベースの内容を永続保存する場合は、ディレクトリ「/var/log/journal」を作成した後に、一度システムを再起動しておきます。journaldは、このディレクトリが存在する場合は、こちらにログデータベースを作成して、永続保存するようになっています。ただし、永続保存と言っても、無制限にログをためていくわけではありません。ログデータベースのサイズが、保存用ディレクトリが存在するファイルシステムの全容量の10%以上になるか、あるいは、該当のファイルシステムの空き容量が15%以下になると、古いエントリーから順に削除されていきます。

これらの保存容量を明示的に指定する場合は、設定ファイル「/etc/systemd/jounrald.conf」の「SystemMaxUse」(/var/log/journalの場合)と「RuntimeMaxUse」(/var/run/log/journalの場合)で値を設定します。詳細については、journald.conf(5)のmanページに記載があります。

ちなみに、最近のバージョンのFedoraでは、デフォルトではrsyslogdがインストールされなくなっており、システムログはjournaldだけで管理されます。したがって、Fedoraではデフォルトで「/var/log/journal」が用意されており、ログデータベースは永続保存されるようになっています。

rsyslogdのしくみ

図1に示したように、journaldが受け取ったsyslogメッセージは、そのまま、rsyslogdにも転送されます。rsyslogdは従来と同様に、ログメッセージのファシリティ(Facility：種類)とプライオリティ(Prioirty：緊急度)に応じて、出力先のログファイルを決定します。**表1**は、Linuxの主要なシステムログファイルの一覧ですが、最初の3つがrsyslogdにより出力されるテキスト形式のログファイルになります。出力先のログファイルは、メッセージのファシリティとプライオリティで決まるわけですが、これらはアプリケーションがsyslog関数でログを出力する際にオプションで指定されます。

ファシリティは、メッセージの種類を分類するためのもので、次のいずれかが指定されます。

- auth、authpriv、cron、daemon、lpr、mail、news、security(authの別名)、syslog、user、uucp、mark：システムで規定のエントリー
- kern：カーネルメッセージ
- local0～local7：規定のエントリーに該当しないメッセージについて、アプリケーションが任意に使用可能

プライオリティは、メッセージの緊急度を示すもので、次のいずれかが指定されます。

- debug、info、notice、warning、warn(warning

の別名)、err、error (errの別名)、crit、alert、emerg、panic (emergの別名):この順に緊急度が高くなる(debugは緊急度が低く、emergは緊急度が高いという順番)

rsyslogdはこれらの情報を元に、設定ファイル「/etc/rsyslog.conf」に従って、出力先のログファイルを決定します。**リスト1**は設定ファイルの主要部分の抜粋ですが、各行でメッセージのファシリティに対して、出力先のログファ

▼リスト1　/etc/rsyslog.confの主要部分(デフォルト設定)

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                 /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none                /var/log/messages

# The authpriv file has restricted access.
authpriv.*                                              /var/log/secure

# Log all the mail messages in one place.
mail.*                                                  -/var/log/maillog


# Log cron stuff
cron.*                                                  /var/log/cron

# Everybody gets emergency messages
*.emerg                                                 :omusrmsg:*

# Save news errors of level crit and higher in a special file.
uucp,news.crit                                          /var/log/spooler

# Save boot messages also to boot.log
local7.*                                                /var/log/boot.log
```

column

シェルスクリプトのログファイルを管理していますか?

サーバ運用では、自作のシェルスクリプトを用いて、バックアップなどの定形作業を自動化することがよくあります。このとき、シェルスクリプトのログファイルをどのように管理しているでしょうか? コマンド出力のリダイレクト機能を用いて、独自のログファイルに記録する人もいるようですが、筆者のお勧めはloggerコマンドを用いる方法です。たとえば、シェルスクリプトの中で次のように記載します。

```
{
<...ここにバックアップ処理のコマンド群を記載...>
} 2>&1 | logger -p local0.info -t "backup"
```

これにより、{}内のコマンド出力の内容がすべてまとめて、syslogメッセージとして送信されます。-pオプションはファシリティとプライオリティの指定で、-tオプションはログの先頭に付与するヘッダーになります。この方法であれば、rsyslogdが管理するシステムログ「/var/log/messages」に記録することができて、ログファイルのローテーション[注A]などに気を使う必要がなくなります。もちろん、rsyslogdの設定で、別ファイルに分けることも可能です。

注A) 1つのログファイルのサイズが巨大になりすぎないように、ログファイルを切り替える処理のこと。

イルをひも付けています。また、プライオリティの指定により、指定以上の緊急度のメッセージのみを書き出します。代表的な設定部分を説明すると、次のようになります。

- ***.emerg**：任意のファシリティで、emerg以上の緊急度のメッセージを出力する
- **mail.***：ファシリティがmailのすべてのメッセージを出力する
- ***.info;mail.none;authpriv.none;cron.none**：任意のファシリティで、info以上の緊急度のメッセージを出力する。ただし、ファシリティがmail、authpriv、cronのものは除外する

セキュリティに関連するログ収集

ここではログの見方や出力設定を学ぶ例として、セキュリティに関連するログを取りあげます。

ログイン認証のログ

Linuxのセキュリティの基本は、なんと言ってもログイン認証の管理です。これまでサーバにログインしたユーザの情報は、**表1**の「/var/log/wtmp」に記録されており、lastコマンドで内容を確認できます。**図3**のように、現在ログイン中のユーザを含めて、過去にシステムにログインしたユーザについて、ログイン時刻／ログアウト時刻を確認できます。

ただし、これだけでは“ログインに失敗した”という事実はわかりません。不正ログインの試みを検知するには、ログインに失敗したという事実を確認することも大切です。これは**表1**の「/var/log/secure」から確認します。このファイルには、OSのユーザ認証にかかわるログがすべてまとめて記録されています。**リスト2**は実際のサーバから取得した例ですが、同じユーザによる、ログイン失敗の記録が連続して何度も記録されており、不正ログインを試みている

▼図3　lastコマンドの実行例

```
# last ⏎
nakai    pts/0        192.168.122.1    Wed Jun  8 16:29   still logged in
reboot   system boot  3.10.0-327.13.1. Wed May 25 09:00 - 16:29 (14+07:29)
root     pts/0        192.168.200.1    Mon Apr  4 16:48 - 17:01  (00:12)
root     tty1                          Mon Apr  4 16:46 - 16:48  (00:01)
root     tty1                          Mon Apr  4 16:45 - 16:46  (00:00)
reboot   system boot  3.10.0-123.20.1. Mon Apr  4 16:45 - 20:25 (28+03:39)
nakai    pts/0        192.168.122.1    Wed Nov 18 14:07 - 15:02  (00:54)
..省略..
```

▼リスト2　不正ログインの試行を示すログ（/var/log/secure）

```
Jun  8 05:16:29 server01 sshd[988]: reverse mapping checking getaddrinfo for ip-11-171.➢
xxxx.xxx [xxx.xxx.xxx.xxxx] failed - POSSIBLE BREAK-IN ATTEMPT!
Jun  8 05:16:29 server01 sshd[988]: Invalid user xxxxxx from xxx.xxx.xxx.xxx
Jun  8 05:16:29 server01 sshd[988]: input_userauth_request: invalid user xxxxxx [preauth]
Jun  8 05:16:29 server01 sshd[988]: Received disconnect from xxx.xxx.xxx.xxx: 11: Bye Bye ➢
[preauth]
Jun  8 05:16:31 server01 sshd[990]: reverse mapping checking getaddrinfo for ip-11-171.➢
xxxx.xxx [xxx.xxx.xxx.xxxx] failed - POSSIBLE BREAK-IN ATTEMPT!
Jun  8 05:16:31 server01 sshd[990]: Invalid user xxxxxx from xxx.xxx.xxx.xxx
Jun  8 05:16:31 server01 sshd[990]: input_userauth_request: invalid user xxxxxx [preauth]
Jun  8 05:16:32 server01 sshd[990]: Received disconnect from xxx.xxx.xxx.xxx: 11: Bye Bye ➢
[preauth]
..省略..
```

▼リスト3　/var/log/secureからユーザの行動を確認

```
① Jun  8 16:01:00 docker01 unix_chkpwd[16539]: password check failed for user (nakai)
② Jun  8 16:01:00 docker01 sshd[16537]: pam_unix(sshd:auth): authentication failure; ➥
  logname= uid=0 euid=0 tty=ssh ruser= rhost=gateway  user=nakai
③ Jun  8 16:01:02 docker01 sshd[16537]: Failed password for nakai from 192.168.200.1 port ➥
  34566 ssh2
④ Jun  8 16:01:04 docker01 sshd[16537]: Accepted password for nakai from 192.168.200.1 port ➥
  34566 ssh2
⑤ Jun  8 16:01:04 docker01 sshd[16537]: pam_unix(sshd:session): session opened for user ➥
  nakai by (uid=0)
⑥ Jun  8 16:01:14 docker01 sudo:   nakai : TTY=pts/0 ; PWD=/home/nakai ; USER=root ; ➥
  COMMAND=/bin/docker info
⑦ Jun  8 16:01:18 docker01 sshd[16561]: Received disconnect from 192.168.200.1: 11: ➥
  disconnected by user
⑧ Jun  8 16:01:18 docker01 sshd[16537]: pam_unix(sshd:session): session closed for user nakai
```

ことがひと目でわかります。**リスト2**では「xxxxx」の記号で伏せ字にしてありますが、ログインユーザ名のほかにログイン元のIPアドレスなども記録されていますので、不正ログインを試みた犯人を探しだす手がかりになります。

ちなみに、ログに記載されている「reverse mapping checking……」というメッセージは、接続元サーバのホスト名がDNSに正しく登録されていないことを示すものです。不正行為のために用意したサーバの場合、ホスト名をDNSに登録せずに使用することも多いので、このようなチェックが行われるようになっています。

あるいは、正しいユーザが正常にログインした場合においても、「/var/log/secure」を見ると、その行動内容を確認できます。**リスト3**の例を見て、どのようなユーザがどのような行動をしたか、読み解くことができるでしょうか？

答えは次のとおりです。ユーザ「nakai」は次のような行動を取りました。

- ①～③……パスワードを間違えてログインに失敗
- ④～⑤……正しいパスワードでログインに成功
- ⑥……sudoコマンドを用いて、root権限で「docker info」コマンドを実行
- ⑦～⑧……ログアウト

sshでログインするだけではなく、sudoコマンドやsuコマンドでユーザ権限を昇格した際も、その記録が残ります。

コマンド実行記録の保存

ログファイル「/var/log/secure」から、sudoコマンドの実行記録を確認できることがわかりましたが、そのほかの一般的なコマンドの実行記録を確認することはできるでしょうか？　これにはいくつかの方法があります。1つは、psacctの利用です。次のコマンドでpsacctのRPMパッケージを導入した後に、psacctサービスを有効化して、起動します。

```
# yum -y install psacct ↵
# systemctl enable psacct.service ↵
# systemctl start psacct.service ↵
```

これにより、システム上で起動したプロセスの情報がバイナリ形式のログファイル「/var/account/pacct」に記録されるようになります。正確に言うと、あるプロセスが終了したタイミングで、そのプロセスの稼働時間と終了時刻の情報が記録されます。このログファイルの内容は、lastcommコマンドで確認します。**図4**の例では、ユーザ「nakai」が起動したプロセス、すなわち、実行したコマンドの履歴を確認しています。このほかにも、--commandオプションで特定のコマンドの記録のみを検索するなども可

▼図4　ユーザのコマンド実行履歴を確認

```
# lastcomm --user nakai ↵
ls                     nakai   pts/0      0.00 secs Wed Jun  8 23:29
w                      nakai   pts/0      0.00 secs Wed Jun  8 23:29
ps                     nakai   pts/0      0.01 secs Wed Jun  8 23:29
grep                   nakai   pts/0      0.00 secs Wed Jun  8 23:29
bash             F     nakai   pts/0      0.00 secs Wed Jun  8 23:29
dircolors              nakai   pts/0      0.00 secs Wed Jun  8 23:29
bash             F     nakai   pts/0      0.00 secs Wed Jun  8 23:29
 ..省略..
```

▼リスト4　端末上の操作を記録するスクリプト

```
.bash_profileの末尾に追加
mkdir -p ~/.termlog
find ~/.termlog -daystart -maxdepth 1 -name "*.log" -mtime +30 -exec rm {} \;
script -afq ~/.termlog/termlog_$USER.$(date +%Y%m%d).$$.log
exit 0
```

能です。

ただし、この方法の場合はプロセス名（コマンド名）が表示されるだけで、コマンドのオプションや実行結果などの詳細は確認できません。場合によっては、ユーザがコマンド端末で行った操作について、すべての記録を保存したいこともあるでしょう。Puttyなど、SSH端末アプリケーションの機能で、端末上の画面表示をログファイルに保存することもできますが、この場合は作業者自身にログを取ってもらう必要があります。作業者の知らないところで自動的に記録するには、どうすればよいのでしょうか？

筆者は、作業ユーザのホームディレクトリにある「.bash_profile」の末尾に、**リスト4**の内容を追加するという方法を用いています[注1]。「.bash_profile」はユーザがログインした際に自動的に実行されるスクリプトです。この中からscriptコマンドを実行しており、これにより、画面に表示された内容をそのままテキストファイルに保存します。この例では、ホームディレクトリ上の隠しディレクトリ「.termlog」の下に、「termlog_<ユーザ名>.<日付>.<プロセスID>.log」というファイル名で保存します。<プロセスID>の部分は、ログイン時に起動したbashのプロセスIDで、同じ日に複数回ログインした場合でも、ログインごとにファイルが分かれるようにしてあります。2行目のfindコマンドでは、30日以上前のログファイルを削除しています。

この方法の場合、ログインしたユーザが意図的にログファイルを削除することもできるので、厳格な監査目的には使えません。それでも、何か問題が起きた際には過去の作業内容をすべて確認できるので、なかなか便利な方法です。

まとめ

第1章では、ログ管理の基礎として、Linuxのシステムログについて解説を行いました。とくに、CentOS 7ではjournaldとrsyslogdが連携したしくみに変わっているので、この点を押さえておくとよいでしょう。またコラムでは、自作のシェルスクリプトにおいても、syslogのしくみが活用できることを説明しました。第2章からのアプリケーションのログ出力においても、syslogと連携する方法と、独自のログファイルを出力する方法のどちらを選択するかは、ログ管理の1つのポイントになるでしょう。SD

注1）「.bashrc」に記載すると正しく動作しないので注意してください。誤って.bashrcに記載すると、scriptコマンド実行時に再度.bashrcが呼び出されるために、無限にscriptコマンドが実行されてしまいます。

Webサーバのログ設定

ApacheとNginxのログ出力の基本と目的別手法

Author 鶴長 鎮一(つるなが しんいち)

本章では、Webサーバ(Apache、Nginx)におけるログ活用や、「アクセスログ」と「エラーログ」の出力設定について解説します。通常では出力されないログを出力する方法や、フォーマットのカスタマイズ方法についても紹介します。執筆にあたり、CentOS 7.2.1511/Nginx 1.10.1/Apache HTTPD 2.4.6を使用しています。

はじめに

Webサーバを効率よく管理し、運用の負担やセキュリティリスクを減らすには、サーバのパフォーマンスをモニタし、問題が発生した際にはいち早く検知するようにします。それにはログの活用が欠かせません。

Webサーバが出力するログはおもに2種類です。クライアントからHTTPリクエストを受け取った際に出力する「アクセスログ」と、リクエストを処理している最中に発生したエラーを出力する「エラーログ」です。本章では2大Webサーバとされる「Apache HTTPD(以降、単にApache)」と「Nginx」について、ログの出力のしかたや、フォーマットの変更方法を解説します。

ApacheもNginxもログフォーマットを柔軟に変えることができ、URL参照元やクライアント情報といった変数をログに加えることができます。昨今ログ解析の重要性が高まり、標準では出力されない値を追加することが多くなっています。既存のフォーマットでは対応できないものでも、状況に応じてフォーマットをカスタマイズすることで最適なログを出力できるようになります。

Apache HTTPDのログ

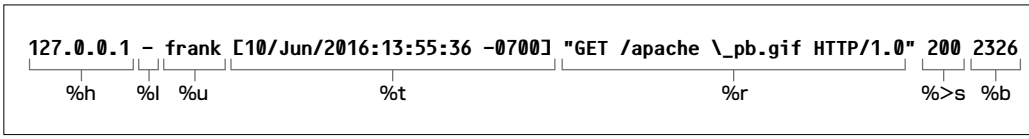
Apache HTTPDログ設定の基本

Apacheには柔軟なロギング機能が備わっており、syslogを経由することなく独自機構でログを出力します。そのためログのフォーマットや出力方法を自由に設定できます。設定はhttpd.confファイル(CentOS 7では「/etc/httpd/conf/httpd.conf」)で行います。

httpd.confでは行単位で設定を行い、1行で1つの事柄を設定します。1行の先頭に設定項目を記述し、空白文字をはさんで設定内容を記述します。設定項目には、「LogFormat」や「CustomLog」といった「ディレクティブ(Directive)」と呼ばれる記述子を使います。おもに使用するディレクティブは、TransferLog/LogFormat/CustomLog/ErrorLog/LogLevelといったものです(**リスト1**)。

それぞれのディレクティブに応じてファイルパスやフォーマットといった設定内容を指定します。なおログのファイルのパスは「/」で始まらない限り、「ServerRoot」ディレクティブで指定されたパスからの相対パスとして扱われます。

▼図1　CLF（Common Log Format）形式のログ



アクセスログ

アクセスログのカスタマイズ

ApacheはHTTPリクエストを受け取るごとに、1行のログをアクセスログに記録します。Apacheのロギング機能を拡張するには、拡張モジュールの「mod_log_config」を使用します。通常Apacheをインストールすれば一緒にインストールされます。**リスト1**のように「TransferLog」ディレクティブでアクセスログを設定すると、「CLF（Common Log Format）」と呼ばれる、**図1**のフォーマットでログが出力されます[注1]。CLFで出力されたログの各フィールドの意味は**表1**のとおりです。

CLF以外のフォーマットのログを出力するには、「CustomLog」ディレクティブを使って次のように設定します。

```
CustomLog    logs/access_log    combined
```

上の設定例では「combined」フォーマットでアクセスログを出力しています。combinedフォーマットはApacheオリジナルのフォーマットです。デフォルトで定義されており、アクセスログの書式として一般的に使われています。CLFで出力されるログに、「参照元のサイト情報」と「クライアントのブラウザ情報」が加わり、より詳細なログを残すことができます（**リスト2**）。

注1）または一番最後に指定されたLogFormatディレクティブで定義された、ニックネームを定義していないフォーマットが使用されます。

▼リスト1　Apache HTTPDのログ出力基本設定

```
# 拡張モジュールの読み込み
LoadModule log_config_module modules/mod_log_config.so
LoadModule logio_module modules/mod_logio.so

# アクセスログの設定（Common Log Format）
# TransferLog    ログのファイルのパス
TransferLog     logs/access_log

# ログフォーマットの定義
LogFormat       "フォーマット定義" ニックネーム
LogFormat       "%h %l %u...."      combined

# アクセログの設定（フォーマット指定）
# CustomLog       ログのファイルのパス ニックネーム
CustomLog       logs/access_log    combined

# エラーログの設定
# ErrorLog        ログのファイルのパス
ErrorLog        logs/error_log

# エラーログのレベル
# LogLevel        ログレベル
LogLevel        warn
```

定義済みフォーマットを利用する

combinedのほか、「common」「combinedio」「referer」「agent」といったフォーマットをデフォルトで利用できます（**リスト3**）。「combinedio」は「combined」ログに送受信バイト数のフィールドを追加したものです。出力には「mod_logio」拡張モジュールが必要になります。詳細な情報が必要な場合は「combined」または「combinedio」を使用します。Apacheのインストール方法によっては、「referer」「agent」が定義されていない場合があります。

フォーマットを独自に定義する

アクセスログのフォーマットを独自に定義することで、詳細な情報を付加したり、必要な情

▼リスト2　Apacheオリジナルのcombined形式のログ

```
172.16.55.1 - - [08/Jul/2016:12:00:12 +0900] "GET /index.html?key1=value1&key2=value2 HTTP/1.1" 200 5 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/601.6.17 (KHTML, like Gecko) Version/9.1.1 Safari/601.6.17"
```

▼表1　フォーマット文字列とその意味

フォーマット文字列	意味
%a	リモートIPアドレス
%A	ローカルIPアドレス
%B	レスポンスのバイト数。HTTPヘッダは除く
%b	レスポンスのバイト数。HTTPヘッダは除く。CLF書式。すなわち、1バイトも送られなかったときは0ではなく、「-」になる
%{*Foobar*}C	サーバに送られたリクエスト中のクッキー*Foobar*の値
%D	リクエストを処理するのにかかった時間、マイクロ秒単位
%{*FOOBAR*}e	環境変数*FOOBAR*の内容
%f	ファイル名
%h	リモートホスト
%H	リクエストプロトコル
%{*Foobar*}i	サーバに送られたリクエストの*Foobar:*ヘッダの内容
%l	(identdからもし提供されていれば)リモートログ名。mod_identがサーバに存在して、IdentityCheckディレクティブがOnに設定されていない限り、-になります
%m	リクエストメソッド
%{*Foobar*}o	応答の*Foobar:*ヘッダの内容
%p	リクエストを扱っているサーバの正式なポート
%P	リクエストを扱った子プロセスのプロセスID
%q	問い合せ文字列(存在する場合は前に?が追加される。そうでない場合は空文字列)
%r	リクエストの最初の行
%s	ステータス。内部でリダイレクトされたリクエストは、もともとのリクエストのステータス。最後のステータスは%>s
%t	リクエストを受付けた時刻。CLFの時刻の書式(標準の英語の書式)
%T	リクエストを扱うのにかかった時間、秒単位
%u	リモートユーザ(認証によるもの。ステータス(%s)が401のときは意味がないものである可能性がある)
%U	リクエストされたURLパス。クエリ文字列は含まない
%v	リクエストを扱っているサーバの正式なServerName
%V	UseCanonicalNameの設定によるサーバ名
%I	リクエストとヘッダを含む、受け取ったバイト数。0にはならない。これを使用するためにはmod_logioが必要

※http://httpd.apache.org/docs/2.4/ja/mod/mod_log_config.htmlをもとに作成

▼リスト3　combined、common、combinedio、referer、agentによって出力されるログの例

```
・combined
192.168.45.128 - - [05/Jul/2016:15:20:33 +0900] "GET / HTTP/1.1" 200 5039 "-" "Mozilla/5.0 ➥
(Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/601.6.17 (KHTML, like Gecko) Version/9.1.1 ➥
Safari/601.6.17"

・common
192.168.45.128 - - [05/Jul/2016:15:22:47 +0900] "GET / HTTP/1.1" 200 5039

・combinedio
192.168.45.128 - - [05/Jul/2016:15:22:47 +0900] "GET / HTTP/1.1" 200 5039 "-" "Mozilla/5.0 ➥
(Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/601.6.17 (KHTML, like Gecko) Version/9.1.1 ➥
Safari/601.6.17" 332 5237

・referer
http://192.168.45.148/ -> /icons/apache_pb.gif
http://192.168.45.148/ -> /icons/poweredby.png

・agent
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/601.6.17 (KHTML, like Gecko)
Version/9.1.1 Safari/601.6.17
```

▼リスト4　アクセスログのフォーマットを独自に定義する

```
#独自フォーマット「custom01」の定義と出力
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %T" custom01
CustomLog logs/access_log custom01
```

報を限定しログのサイズを小さくしたりできます。アクセスログのフォーマットを独自に定義するには、最初に「LogFormat」ディレクティブを使って書式を設定します。定義したフォーマットには一意なニックネームを付けます。

```
#独自フォーマットの定義
LogFormat    "書式"    ニックネーム
```

次に「CustomLog」ディレクティブの引数にログファイルのパスと、使用するフォーマットのニックネームを指定します。

```
#定義したフォーマットでアクセスログを出力
CustomLog   ログファイルのパス   ニックネーム
```

リスト4は設定の一例です。書式には「%h」や「%l」のような記述子を使用します。主な記述子とその意味は**表1**のとおりです。書式は「"（ダブルクォート）」で囲みますが、書式の中に「"」を含める場合は、エスケープシーケンスを使って「\"」と記述します。そのほか「\n（改行）」や「\t（タブ）」といった制御文字を含めることもできます。

ニックネームの代わりに直接書式を設定することもできます。

```
CustomLog logs/access_log "%h %l %u %t ➥
\"%r\" %>s %b"
```

複数のフォーマットを使い分けることで、同時に複数のアクセスログを出力するようになります（**リスト5**）。

▼リスト5　同時に複数のアクセスログを出力

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
CustomLog logs/referer_log "%{Referer}i -> %U"
CustomLog logs/agent_log "%{User-agent}i"
```

LTSV形式でアクセスログを出力する

昨今ログ解析の重要性が高まっています。Apacheの標準フォーマットでは各項目の区切り文字にスペース文字を使うため、ログを解析する際に思ったようにパースできなかったり、抽出したい項目を取り出しづらかったりします。自作スクリプトでログを解析したり、Fluentdのような処理ツールでログを収集したりする際は、プログラムで簡単に扱えるフォーマットでアクセスログを出力するようにします。

さまざまなフォーマットが提唱されていますが、中でも「LTSV（Labeled Tab-Separated Values）」は扱いが容易で拡張性が高い簡易データフォーマットとして人気です。またパーサーも数多くそろっており、ログ解析には不自由ありません。LTSVは、その名のとおり、ラベル（Label）と値（Value）のペアをタブ（Tab）で区切った行指向レコードです。

```
ラベル1:値1［TAB］ラベル2:値2［TAB］ラベル3:値3
```

タブで区切られているため、値にスペース文字を含んでいても問題になりません。またラベルと値は「:（コロン）」で区切られており、パースが容易です。CSV（カンマ区切り）やSSV（スペース区切り）のようにフィールドの位置で値の意味が決められたレコードだと、値が増えた場合の拡張性や、値が出力されなかったときの柔軟性が問題になりますが、LTSVなら各値にラベルが付与されているため、必要な項目だけを任意の順番で記述できます。

Apacheで一般的に使用される「combined」フォーマットをLTSVにするには**リスト6**のよ

▼リスト6　LTSV形式でアクセスログを出力する

```
# 元のcombinedフォーマット
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined

# LTSV化したフォーマットを定義
LogFormat "host:%h\tident:%l\tuser:%u\ttime:%t\treq:%r\tstatus:%>s\tsize:%b\treferer:\%➥
{Referer}i\tua:%{User-Agent}i" combined_ltsv

# アクセスログ出力設定
CustomLog logs/access_log combined_ltsv
```

▼リスト7　LTSV形式で出力されたアクセスログのサンプル

```
host:172.16.55.1    ident:- user:-  time:[08/Jul/2016:02:41:52 +0900]   req:GET /index.html➥
HTTP/1.1    status:404  size:208    referer:\-  ua:Mozilla/5.0 (Macintosh; Intel Mac OS X ➥
10_11_5) AppleWebKit/601.6.17 (KHTML, like Gecko) Version/9.1.1 Safari/601.6.17
```

▼リスト8　LTSV形式で出力されたアクセスログのパース結果

```
host:172.16.55.1
ident:-
user:-
time:[08/Jul/2016:02:41:52 +0900]
req:GET /index.html HTTP/1.1
status:404
size:208
referer:\-
ua:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/601.6.17 (KHTML, like Gecko)➥
Version/9.1.1 Safari/601.6.17
```

うに設定します。出力されたログを見ると、**リスト7**のようにLTSV化されています。

出力されたLTSV形式のアクセスログを、次のような簡単なワンライナーでパースすると、**リスト8**のようにログを閲覧できます。

```
$ cat /var/log/httpd/access_log | sed -e➥
's/\t/\n/g'↵
```

LTSV形式のアクセスログは、SSV形式に比べるとラベル名が入るなどデータが冗長になるためサイズが大きくなります。なおラベルには推奨されているものを使うようにします。詳細は「http://ltsv.org」を参考にしてください。

JSON形式でアクセスログを出力する

構造化が容易なJSON形式のログもたびたび利用されます。JSON形式では全体を「{...}」で囲み、ラベルと値を「:(コロン)」で区切ったものを、「,(カンマ)」区切りで列挙します。文字列の値は「"(ダブルクォート)」で囲み、数値はそのまま記述します。

```
{ "ラベル1" : "値1", "ラベル2" : "値2", ➥
"ラベル3" : "値3"}
```

リスト9のように設定するとApacheのアクセスログをJSON形式で出力できるようになります。出力されたログを見ると、**リスト10**のようにJSON化されています。

ログを見やすく整形すると次のようになります。

```
{
    "host": "172.16.55.137",
    "method": "GET",
    "query": "",
    "referer": "-",
    "remote_addr": "172.16.55.1",
    "request": "/index.html",
    "status": "200",
```

※次ページに続く↗

```
    "time": "[08/Jul/2016:03:45:44 +0900]",
    "userAgent": "Mozilla/5.0 (Mac ..省略.. "
}
```

エラーログの設定

エラーログの出力レベルを設定する

Apacheのエラーログには、リクエストを処理しているときに発生したエラーが記録されますが、それ以外にもApacheの診断情報など、動作に関する重要なエラーが記録されます。サーバの起動に失敗したときや、サーバの動作に問題が起こったときの手がかりとしてたいへん重要です。フォーマットを変更することはできませんが、「LogLevel」ディレクティブでログレベルを変えることができます。通常は「warn」レベルより深刻なエラーだけ出力しますが、ログレベルを「info」に引き下げることで、より詳細にエラーを記録できるようになります（**リスト11**）。

syslog経由でエラーログを出力する

Apache独自のログ管理機構を使用せず、「syslog（シスログ）」に集約することもできます。ローカルホストのsyslogデーモンにApacheのエラーログを出力するには、次のように設定します。

```
ErrorLog syslog
```

デフォルトでは、syslogファシリティに「local7」を使用しますが、次のように、ほかのファシリティを指定することもできます。

```
#userファシリティを指定
ErrorLog syslog:user
```

全ての入出力データをダンプする

拡張モジュールの「mod_dumpio」を使うと、Apacheが受け取ったすべての入力（リクエスト）データと、Apacheから送られたすべての出力（レスポンス）データをエラーログにダンプできます。GETメッセージのクエリ文字列はもちろん、POSTメッセージのボディもログに出力します。

mod_dumpioモジュールがインストールされているか確認し、インストールされていなければ別途追加してください。

```
$ httpd -M | grep dumpio↵
..省略..
 dumpio_module (shared)
```

リスト12のように設定すると、入出力データを**リスト13**のようにダンプするようになり

▼リスト11　ログレベルを引き下げ、より多くの情報をエラーログに出力する

```
# エラーログの出力設定
ErrorLog /var/log/httpd/error_log

# ログレベルをinfoに引き下げる（warmがデフォルト）
LogLevel info
```

▼リスト9　JSON形式でアクセスログを出力する

```
# JSON化したフォーマットを定義
LogFormat "{ \"time\":\"%t\", \"remote_addr\":\"%a\", \"host\":\"%V\", \"request\":\"%U\",↵
\"query\":\"%q\", \"method\":\"%m\", \"status\":\"%>s\", \"userAgent\":\"%{User-agent}i\",↵
\"referer\":\"%{Referer}i\" }" combined_json

# アクセスログ出力設定
CustomLog "logs/access_json_log" combined_json
```

▼リスト10　JSON形式で出力されたアクセスログ

```
{ "time":"[08/Jul/2016:03:45:44 +0900]", "remoteIP":"172.16.55.1", "host":"172.16.55.137",↵
"request":"/index.html", "query":"", "method":"GET", "status":"200", "userAgent":"Mozilla/↵
5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/601.6.17 (KHTML, like Gecko) Version/↵
9.1.1 Safari/601.6.17", "referer":"-" }
```

ます。出力先はエラーログ（error_log）ファイルになります。mod_dumpioモジュールは標準では無効化されているため、「LoadModule...」で有効化します。入力データ（リクエスト）のダンプをする場合は「DumpIOInput On」を、出力データ（レスポンス）のダンプをする場合は「DumpIOOutput On」を指定します。また必ず「LogLevel」ディレクティブでエラーログの出力レベルを「debug」に変更し、さらに引数に「dumpio:trace7」を指定します。設定が完了すると**リスト13**のようなログが出力されます。

すべての入出力データをダンプすると、大量のログが出力されるため、デバッグ作業が完了したら、ダンプ出力を止めるようにします。

▼リスト12　ダンプされた入力データ

```
#拡張モジュール「/mod_dumpio.so」の読み込み
LoadModule dumpio_module modules/mod_dumpio.so

#入力データ（リクエスト）のダンプを有効化
DumpIOInput On

#出力データ（レスポンス）のダンプを有効化
DumpIOOutput On

#ログレベルを変更
LogLevel debug dumpio:trace7
```

▼リスト13　ダンプされた入力データ

```
Fri Jul 08 12:00:12.584409 2016] [dumpio:trace7] [pid 24854] mod_dumpio.c(140): [client 172.16.55.1:55129] mod_dumpio: dumpio_in [getline-blocking] 0 readbytes
[Fri Jul 08 12:00:12.584528 2016] [dumpio:trace7] [pid 24854] mod_dumpio.c(63): [client 172.16.55.1:55129] mod_dumpio:  dumpio_in (data-HEAP): 50 bytes
[Fri Jul 08 12:00:12.584535 2016] [dumpio:trace7] [pid 24854] mod_dumpio.c(103): [client 172.16.55.1:55129] mod_dumpio:  dumpio_in (data-HEAP): GET /index.
html?key1=value1&key2=value2 HTTP/1.1\r\n
```

▼リスト14　デバッグログを出力する

```
LoadModule log_debug_module modules/mod_log_debug.so

#/foo/にアクセスした時にメッセージを出力
<Location /foo/>
    LogMessage "/foo/ has been requested"
</Location>

LogLevel warn log_debug:info
```

デバッグログを出力する

特定のURLにアクセスしたときや、条件を満たしたときだけエラーログにメッセージを出力するには、「mod_log_debug」モジュールを使って**リスト14**のように設定します。mod_log_debugモジュールがインストールされているか確認し、インストールされていなければ別途追加してください。

```
$ httpd -M | grep debug↵
 ..省略..
 log_debug_module (shared)
```

リスト14のように設定すると、**リスト15**のようなメッセージがエラーログの中に出力されます。mod_log_debugモジュールは標準では無効化されているため、「LoadModule...」で有効化します。また必ず「LogLevel」ディレクティブで「debug:info」のように引数を追加してください。条件を設けたり、環境変数を引数に指定することもできます。詳細は「https://httpd.apache.org/docs/2.4/en/mod/mod_log_debug.html」を参考にしてください。

そのほかのログ

フォレンジクス（forensic）ログを残す

セキュリティインシデントの原因究明やアクセス時に発生する問題への対処には、HTTP

リクエストの全内容をフォレンジクス（forensic）ログとして残すようにします。それにはApache拡張モジュールの「mod_log_forensic」を使用し、**リスト16**のように設定します。mod_log_forensicモジュールがインストールされているか確認し、インストールされていなければ別途追加してください。

```
$ httpd -M | grep forensic↵
 ..省略..
 log_forensic_module (shared)
```

リスト17のようなフォレンジクスログが出力されます。フォレンジクスログは1リクエストに対して2行のログを出力します。1回目はリクエストヘッダを受け取った時点で、リクエスト行と受け取ったすべてのヘッダを「|（パイプ）」で区切ったものをログに出力します。2回目はリクエストが処理されたあとに出力されます。2行のログを紐付けられるよう、リクエストごとに割り振られた一意なフォレンジクスIDが、各ログの第1フィールドに記述されます（**リスト17**の「V3845kq-1mG0IxIsl78ZWwAAAAQ」）。1行目には「+フォレンジクスID」が、2行目には「-フォレンジクスID」と記述されるため、IDの組を調べることで、完了していないリクエストがないか確認できます。

ログファイルのローテーション

ログを有効に活用するには、単に記録するだけではなく、記録されたファイルを適切に保管し整理しておくことも重要になります。Apacheのデフォルトでは永久に1つのファイルの末尾に新たなログを追加し続けます。そのため稼働時間が長くなれば、それに比例し巨大なログファイルを生成することになります。そのためCentOSやUbuntuでは「Logrotate」のようなログ管理ユーティリティでファイルを日ごとにローテーションし、古いものは削除しています。

Logrotateを使わなくても、Apacheのログ管理機構だけでローテーションさせることができます。それには「rotatelogs」コマンドを使用します。Apacheをインストールするとrotatelogsコマンドも同時にインストールされます。rotatelogsの引数にはファイル名とローテーション間隔秒数を指定し、**リスト18**のように設定します。

秒数のほか、ファイルサイズを指定することもできます。**リスト19**のように設定すると、ファイルサイズが指定したサイズ（5MB）に達する

▼リスト15　デバッグログ

```
[Fri Jul 08 15:57:15.575031 2016] [log_debug:info] [pid 25457] [client 172.16.55.1:58511] ➢
/foo/ has been requested
```

▼リスト16　フォレンジクスログを残す

```
LoadModule log_forensic_module modules/mod_log_forensic.so
ForensicLog logs/forensic_log
```

▼リスト17　フォレンジクスログ（1リクエストに対し2行出力される）

```
# 1回目
+V3845kq-1mG0IxIsl78ZWwAAAAQ|GET / HTTP/1.1|Host:172.16.55.137|Cache-Control:max-age=0|➢
Connection:keep-alive|Accept:text/html,application/xhtml+xml,application/xml;q=0.9,➢
*/*;q=0.8|User-Agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/601.6.17➢
 (KHTML, like Gecko) Version/9.1.1 Safari/601.6.17|Accept-Language:ja-jp|Accept-Encoding:➢
gzip, deflate|DNT:1

# 2回目
-V3845kq-1mG0IxIsl78ZWwAAAAQ
```

と、ローテーションを実施するようになります。

なお指定されたファイル名にタイムスタンプを付加したものが、ログのファイル名として使用されます。タイムスタンプではなく「20160707」のような日付を用いるには**リスト20**のように設定します。日付フォーマットの詳細は「$ man rotatelogs」で確認してください。

なおrotatelogsは、古くなったファイルを自動で削除しないため、手動で削除する必要があります。

Nginxのログ

Nginxのログ設定の基本

Nginxにも柔軟なロギング機能が備わっており、ログのフォーマットや出力方法を自由に設定できます。設定は「nginx.conf（通常は「/etc/nginx/nginx.conf」）」で行います。設定したい項目ごとに「ディレクティブ」と「値」を使って指定し、行末に「;（セミコロン）」を置きます。エラーログを設定するには「error_log」ディレクティブを使って次のように設定します。

```
error_log  ログのファイルのパス ログレベル;
```

アクセスログを設定するには、「httpコンテキスト」と呼ばれる「http{……}」で囲まれたブロック内で「access_log」ディレクティブを使って行います（**リスト21**）。

最初に「log_format」ディレクティブでアクセスログの書式を設定します。このあとの「access_log」ディレクティブで、ここで定義したアクセスログのニックネームを指定することで実際にログファイルが出力されるようになります。設定例ではニックネームを「main」とし、

```
$remote_addr - $remote_user [$time_local]➢
"$request" $status $body_bytes_sent ➢
"$http_referer" "$http_user_agent" "$http➢
_x_forwarded_for"
```

といった内容のログを定義しています。書式内の「$」で始まる文字列は、Nginxの組み込み変数です。**表2**のような意味を持っています。実際に**リスト22**のようなアクセスログが出力されます。なおNginxには、Apache互換ログフォーマットの「combined」が組み込まれています（**リスト23**）。

アクセスログのパスやファイル名、使用するフォーマットを設定するには「access_log」ディレクティブを使用します。**リスト21**では「/var/log/nginx/access.log」ファイルに、先ほどlog_formatディレクティブで定義したmainフォーマットを適用し出力します。書式を省略するとデフォルト書式の「combined」が適用されます。アクセスログを出力しないようにするには「off」を指定します。

▼リスト18　httpd.conf（1日（86,400秒）ごとにローテーションを実施する場合）

```
CustomLog "|rotatelogs log/access_log 86400" combined
ErrorLog "|rotatelogs /var/log/httpd/error_log 86400"  (※注A)
```

▼リスト19　httpd.conf（ログファイルが5Mを超えたときにローテーションを実施する場合）

```
CustomLog "|rotatelogs /var/log/httpd/access_log 5M" combined  (※注A)
```

▼リスト20　httpd.conf（ログファイル名にYYYYMMDDを用いる場合）

```
CustomLog "|rotatelogs /var/log/httpd/access_log.%Y%m%d 86400" combined  (※注A)
```

注A） Apacheのインストール方法によってはrotatelogsをフルパスで指定する必要があります。

```
access_log  off
```

access_logディレクティブはhttpコンテキストのほか、server/location/ifといったコンテキスト内でも利用できるため、バーチャルサーバやURIに応じてアクセスログを変更することができます。

ログ出力時に圧縮する

Nginxはログを出力するタイミングでログを圧縮できます。**リスト24**のように設定すると前回圧縮してから3秒以上経過したあとにログを圧縮します。圧縮レベルはデフォルトの「1」です。圧縮レベルは最大9まで指定できますが、圧縮率が高くなると、出力にかかる時間やCPUの負担が増えます。

圧縮機能を利用するためにはNginxの「ngx_http_gzip_module」と「zlib」モジュールが必要になりますが、デフォルトで組み込まれています。圧縮レベル1でも1/10に圧縮され、30,000行のアクセスログであれば、2.7MBが28KBに圧縮されます。圧縮されたログを見るには「zcat」コマンドを使用します。

```
$ zcat /var/log/nginx/access.log.gz | more↵
```

▼表2　アクセスログのフォーマット定義に使用したNginxの組み込み変数

変数名	意味
$remote_addr	クライアントアドレス
$remote_user	Basic認証時のユーザ名
$time_local	Common Log形式のローカルタイム
$request	完全なオリジナルのリクエストURI
$status	レスポンスのステータスコード
$body_bytes_sent	レスポンスボディ(ヘッダを含まない)のバイト数
$http_referer	Refererリクエストヘッダの値
$http_user_agent	User-Agentリクエストヘッダの値
$http_x_forwarded_for	X-Forwarded-Forリクエストヘッダの値

▼リスト21　Nginxのログ基本設定

```
..省略..
error_log  /var/log/nginx/error.log warn;
..省略..
http {
..省略..
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;
..省略..
}
```

▼リスト22　実際に出力されたアクセスログ

```
192.168.3.17 - - [14/May/2015:12:09:12 +0900] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 ➥
(Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/600.6.3 (KHTML, like Gecko) Version/8.0.6 ➥
Safari/600.6.3" "-"
```

▼リスト23　Apache互換ログフォーマットの「combined」の定義

```
log_format combined '$remote_addr - $remote_user [$time_local] '
                    '"$request" $status $body_bytes_sent '
                    '"$http_referer" "$http_user_agent"';
```

▼リスト24　Nginxでログを圧縮する例

```
#圧縮レベル1(デフォルト)、最少3秒ごとに圧縮
access_log  /var/log/nginx/access.log.gz main gzip flush=3s;

#圧縮レベル9、最少3秒ごとに圧縮、最少5分ごとに圧縮
access_log  /var/log/nginx/access.log.gz main gzip=9 flush=5m;
```

▼リスト25　LTSV形式でアクセスログを出力する

```
log_format  ltsv  "host:$remote_addr\t"
                  "user:$remote_user\t"
                  "time:$time_local\t"
                  "req:$request\t"
                  "status:$status\t"
                  "size:$body_bytes_sent\t"
                  "referer:$http_referer\t"
                  "ua:$http_user_agent";

access_log  /var/log/nginx/access.log  ltsv;
```

▼リスト26　JSON形式でアクセスログを出力する

```
log_format json '{'
                '"remote_addr": "$remote_addr",'
                '"remote_user": "$remote_user",'
                '"time_local":  "$time_local",'
                '"request":     "$request",'
                '"status":      "$status",'
                '"body_bytes_sent": "$body_bytes_sent",'
                '"http_referer": "$http_referer",'
                '"http_user_agent": "$http_user_agent"'
                '}';

access_log  /var/log/nginx/access.log json;
```

syslog経由でアクセスログを出力する

Linuxの標準的なログ管理システムであるsyslogにエラーログを出力するには、次のように設定します。

```
access_log syslog:server=localhost;
```

リモートのsyslogサーバやデフォルト以外のポート番号（デフォルトはUDP 514番）を指定することもできます。

```
access_log syslog:server=192.168.0.10:515;
```

syslogでログを出力する際に、次のようにシビアリティやタグを設定することもできます。

```
access_log syslog:server=192.168.0.10:515,
           severity=debug,tag=web main;
```

この例ではシビアリティに「debug」を、タグに「web」を、ログフォーマットに「main」を指定しています。なおシビアリティをdebugに指定しても、デバッグログが出力されるわけではありません。

LTSV形式やJSON形式でアクセスログを出力する

よりパースしやすい、LTSV形式やJSON形式でアクセスログを出力できます。LTSVやJSONについての解説はApacheパートを参考にしてください。NginxでLTSV形式のアクセスログを出力するには**リスト25**のように、JSON形式でアクセスログを出力するには**リスト26**のように設定します。

エラーログの設定

デバッグログを出力する

エラーログにデバッグ情報が出力されるようにするには、ログレベルに「debug」を指定します。「error_log」ディレクティブを使って次のように設定します。

```
error_log  /var/log/nginx/error.log debug;
```

デバッグログをリモートのsyslogサーバに出力するには次のように設定します。SD

```
error_log  syslog:server=192.168.0.10 debug;
```

第3章

MySQL 4つのログの使いどころ

データベースの保全、性能評価で役立てる

Author とみたまさひろ　日本MySQLユーザ会　Twitter @tmtms

本章ではMySQLを題材にデータベースのログを考えます。MySQLの運用で確認すべき4つのログでは、ユーザがデータベースに対して行った記録のほかに、パフォーマンスの問題を探るヒントとなる情報や、レプリケーションでも使われるデータがあります。それぞれの役割とログの残し方を知っておきましょう。

MySQLでチェックすべき4つのログ

MySQLで「ログ」というとさまざまな種類のものがあります。システム変数で「log」という文字を含むものは76個もありました。ログと言ってもこれらの中には人間が見るものではなく、MySQLの動作のために内部的に必要なものも含まれています。

今回は、これらの中から人間が見ることができるログとして、代表的な次のものについて説明します。

- エラーログ：mysqldの起動／終了メッセージと実行中のエラーメッセージが記録される
- 一般クエリログ：クライアントから発行されたすべてのクエリが記録される
- スロークエリログ：実行に時間がかかったクエリが記録される
- バイナリログ：クライアントから発行された更新系のクエリが記録される

まず各ログがどのようなものかを説明し、その後に各設定方法を説明します。

エラーログ

エラーログにはmysqldの起動中に発生したエラーメッセージと、mysqldの起動時と終了時の情報が出力されます。デフォルトではmysqldの標準エラー出力に出力されます。

mysqldが異常終了した場合の調査などに役立つメッセージが出力されるので、ファイルに保存するようにしておいたほうが良いでしょう（後述の「エラーログの設定」で解説）。

ファイル形式

エラーログの形式はMySQLのバージョンによって異なります。

MySQL 5.6のログファイルは、「時刻」「mysqldのプロセスID」「メッセージのレベル」「メッセージ」が記録されます（**リスト1**）。時刻は“年-月-日 時:分:秒”の形式です。タイムゾーンはローカル時刻でシステムの設定に依存します。日本のサーバでは通常は日本時間になります。

MySQL 5.7のログファイルは、「時刻」「スレッドID」「メッセージのレベル」「メッセージ」が記録されます（**リスト2**）。時刻はISO8601の形式“年-月-日T時:分:秒Z”でマイクロ秒まで記録されます。タイムゾーンはシステムの設定によらずUTC（Coordinated Universal Time：協定世界時）です。日本時間とは9時間ずれます。

▼リスト1　MySQL 5.6の起動時のエラーログ

```
2016-06-18 21:31:07 13069 [Note] Plugin 'FEDERATED' is disabled.
2016-06-18 21:31:07 13069 [Note] InnoDB: Using atomics to ref count buffer pool pages
2016-06-18 21:31:07 13069 [Note] InnoDB: The InnoDB memory heap is disabled
2016-06-18 21:31:07 13069 [Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
2016-06-18 21:31:07 13069 [Note] InnoDB: Memory barrier is not used
2016-06-18 21:31:07 13069 [Note] InnoDB: Compressed tables use zlib 1.2.3
2016-06-18 21:31:07 13069 [Note] InnoDB: Using Linux native AIO
2016-06-18 21:31:07 13069 [Note] InnoDB: Using CPU crc32 instructions
2016-06-18 21:31:07 13069 [Note] InnoDB: Initializing buffer pool, size = 128.0M
2016-06-18 21:31:07 13069 [Note] InnoDB: Completed initialization of buffer pool
2016-06-18 21:31:07 13069 [Note] InnoDB: Highest supported file format is Barracuda.
2016-06-18 21:31:07 13069 [Note] InnoDB: 128 rollback segment(s) are active.
2016-06-18 21:31:07 13069 [Note] InnoDB: Waiting for purge to start
2016-06-18 21:31:07 13069 [Note] InnoDB: 5.6.31 started; log sequence number 1625997
2016-06-18 21:31:07 13069 [Note] Server hostname (bind-address): '*'; port: 3306
2016-06-18 21:31:07 13069 [Note] IPv6 is available.
2016-06-18 21:31:07 13069 [Note]   - '::' resolves to '::';
2016-06-18 21:31:07 13069 [Note] Server socket created on IP: '::'.
2016-06-18 21:31:07 13069 [Note] Event Scheduler: Loaded 0 events
2016-06-18 21:31:07 13069 [Note] ./bin/mysqld: ready for connections.
Version: '5.6.31'  socket: '/tmp/mysql.sock'  port: 3306  MySQL Community Server (GPL)
```

▼リスト2　MySQL 5.7の起動時のエラーログ

```
2016-06-18T12:32:57.658948Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please ➥
use --explicit_defaults_for_timestamp server option (see documentation for more details).
2016-06-18T12:32:57.659027Z 0 [Warning] Insecure configuration for --secure-file-priv: Current value ➥
does not restrict location of generated files. Consider setting it to a valid, non-empty path.
2016-06-18T12:32:57.659056Z 0 [Note] ./bin/mysqld (mysqld 5.7.13) starting as process 13127 ...
2016-06-18T12:32:57.663629Z 0 [Note] InnoDB: PUNCH HOLE support not available
2016-06-18T12:32:57.663656Z 0 [Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
2016-06-18T12:32:57.663662Z 0 [Note] InnoDB: Uses event mutexes
2016-06-18T12:32:57.663667Z 0 [Note] InnoDB: GCC builtin __sync_synchronize() is used for memory ➥
barrier
2016-06-18T12:32:57.663672Z 0 [Note] InnoDB: Compressed tables use zlib 1.2.3
2016-06-18T12:32:57.663677Z 0 [Note] InnoDB: Using Linux native AIO
2016-06-18T12:32:57.663868Z 0 [Note] InnoDB: Number of pools: 1
2016-06-18T12:32:57.663943Z 0 [Note] InnoDB: Using CPU crc32 instructions
2016-06-18T12:32:57.665021Z 0 [Note] InnoDB: Initializing buffer pool, total size = 128M, instances ➥
= 1, chunk size = 128M
2016-06-18T12:32:57.672652Z 0 [Note] InnoDB: Completed initialization of buffer pool
2016-06-18T12:32:57.674123Z 0 [Note] InnoDB: If the mysqld execution user is authorized, page ➥
cleaner thread priority can be changed. See the man page of setpriority().
2016-06-18T12:32:57.685565Z 0 [Note] InnoDB: Highest supported file format is Barracuda.
2016-06-18T12:32:57.722680Z 0 [Note] InnoDB: Creating shared tablespace for temporary tables
2016-06-18T12:32:57.722753Z 0 [Note] InnoDB: Setting file './ibtmp1' size to 12 MB. Physically ➥
writing the file full; Please wait ...
2016-06-18T12:32:57.906883Z 0 [Note] InnoDB: File './ibtmp1' size is now 12 MB.
2016-06-18T12:32:57.910757Z 0 [Note] InnoDB: 96 redo rollback segment(s) found. 96 redo rollback ➥
segment(s) are active.
2016-06-18T12:32:57.910798Z 0 [Note] InnoDB: 32 non-redo rollback segment(s) are active.
2016-06-18T12:32:57.912642Z 0 [Note] InnoDB: Waiting for purge to start
2016-06-18T12:32:57.963009Z 0 [Note] InnoDB: 5.7.13 started; log sequence number 2526561
2016-06-18T12:32:57.965000Z 0 [Note] InnoDB: Loading buffer pool(s) from /usr/local/mysql-5.7.13-➥
linux-glibc2.5-x86_64/data/ib_buffer_pool
2016-06-18T12:32:57.965500Z 0 [Note] Plugin 'FEDERATED' is disabled.
2016-06-18T12:32:57.969173Z 0 [Note] InnoDB: Buffer pool(s) load completed at 160618 21:32:57
2016-06-18T12:32:57.983855Z 0 [Note] Plugin mysqlx reported: 'X plugin tcp connection enable at port ➥
33060.'
2016-06-18T12:32:57.984179Z 0 [Note] Plugin mysqlx reported: 'Scheduler "work" started.'
2016-06-18T12:32:57.984337Z 0 [Note] Plugin mysqlx reported: 'X plugin initialization successes'
2016-06-18T12:32:57.986085Z 0 [Warning] Failed to set up SSL because of the following SSL library ➥
error: SSL context is not usable without certificate and private key
2016-06-18T12:32:57.986725Z 0 [Note] Server hostname (bind-address): '*'; port: 3306
2016-06-18T12:32:57.986794Z 0 [Note] IPv6 is available.
```

一般クエリログ

一般クエリログはクライアントから発行されるクエリを記録します。デフォルトでは有効になっていません。サーバ起動中に、動的に有効と無効を切り替えることができます。すべてのクエリが記録されるので、常時有効にしておくと膨大な量になってしまうため、必要なときだけ有効にするのがよいでしょう。

たとえば、アプリケーションの操作で予期しない結果が得られた場合、操作の間だけログ記録を有効にすることで、アプリケーションが発行したクエリを知ることができます。O/Rマッパーがクエリの組み立てを行っていて、実際にどのようなクエリが発行されるのかアプリケーション開発者も知らない場合の調査にも役に立ちます。

サーバで実行されるクエリはすべて記録されますが、構文エラーになるようなクエリはログに記録されません（**図1**）。

ファイル形式

一般クエリログファイルには、「時刻」「スレッドID」「コマンド」「引数」が記録されます。時刻の形式はバージョンによって異なります。

MySQL 5.6の時刻形式は年月日（年は下2桁）とローカル時刻（秒単位）で、前の行と同じ時刻の場合は時刻は出力されません。1秒以内に多くのクエリが発行される場合は、grepなどである行だけ抽出してもそのクエリの発行時刻がわかりません。

MySQL 5.7の時刻形式はエラーログと同様にUTCでISO8601の形式（年-月-日T時:分:秒Z）となっており、マイクロ秒単位まで記録されます。5.6と異なり各行に時刻が出力されます。

スレッドIDはクライアントからの接続ごとに割り当てられる番号です。スレッドIDが同じレコードは同じクライアントからのコマンドであることがわかります。

コマンドはクエリのことではなく、サーバ−クライアント間のプロトコルで定義されているコマンドで、Connect、Query、Quitなどがあります。

クライアントからクエリが発行された場合コマンドはQueryとなり、引数にクエリが記録されます。

一般クエリログの見方

mysqlコマンド内で実行する命令の中にはmysqlコマンド自身に対する命令もあります。これらの命令はサーバにクエリとしては送られないため、一般クエリログには記録されません。たとえばpager命令は、mysqlコマンドがクエリ結果表示のために使用する外部コマンドを指定する命令なので、サーバには送られません。

また、useはカレントのデータベースを切り替えるための命令ですが、サーバに対してはUSEクエリではなく、Init DBというコマンド

▼図1　ログの記録例

```
mysql> SELECT MAX(id) FROM t; ↵   ←記録される
+----------+
| MAX(id) |
+----------+
|      123 |
+----------+

mysql> SELECT MAX(unexist) FROM t; ↵   ←エラーだが記録される
ERROR 1054 (42S22): Unknown column 'unexist' in 'field list'

mysql> SELECT MAX() FROM t; ↵   ←構文エラーなので記録されない
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that ➡
corresponds to your MySQL server version for the right syntax to use near ') FROM t' at line 1
```

が送られるように実装されているため、ログ上にもInit DBとして記録されます。

リスト3、4は、mysqlコマンドを使ってroot@localhostがサーバに接続し、testデータベースを選択、テーブルtの作成、tにレコードの挿入、tからの検索を実行、接続を切断したという一連の動作が記録された、5.6、5.7それぞれの一般クエリログの例です。

これらのリストでは同じスレッドIDがまとめてありますが、本来は時系列で記録されているためさまざまなスレッドIDが入り交じっています。あるクエリがどのユーザで発行されたか、どのデータベースで発行されたかを調べるには、該当クエリのスレッドIDと同じスレッドIDのログをさかのぼって、ConnectやInit DBコマンドを見つける必要があります(リストのようにスレッドIDで抽出)。

クライアントが接続を切断するとQuitが記録されますが、厳密には接続が切断されたことではなく、クライアントからQuitコマンドが送られたことを表しています。Quitを送らずに接続を切断するような行儀の悪いプログラムや、クライアントプログラムが不意に落ちてしまった場合はQuitはログには記録されません。

テーブルへの出力

一般クエリログはファイルだけでなく、テーブルに出力することもできます。ログをテーブルに出力するメリットは、SELECTクエリを使用してログの集計を行えることです。

また、テーブルのストレージエンジンはCSVなので、mysqlデータベースディレクトリにCSVファイルがそのまま置かれています(**図2**)。PCにファイルをコピーしてExcelなどの表計算ソ

▼リスト3　5.6の一般クエリログ

```
160702 10:57:45     3 Connect   root@localhost on
                    3 Query     select @@version_comment limit 1
160702 10:57:46     3 Query     SELECT DATABASE()
                    3 Init DB   test
                    3 Query     show databases
                    3 Query     show tables
160702 10:57:47     3 Query     CREATE TABLE t (id INT, value VARCHAR(100))
                    3 Query     INSERT INTO t (id, value) VALUES (123, 'hoge')
                    3 Query     SELECT id, value FROM t
160702 10:57:49     3 Quit
```

▼リスト4　5.7の一般クエリログ

```
2016-07-02T01:30:25.862001Z    5 Connect        root@localhost on  using Socket
2016-07-02T01:30:25.862172Z    5 Query  select @@version_comment limit 1
2016-07-02T01:30:29.093836Z    5 Query  SELECT DATABASE()
2016-07-02T01:30:29.094107Z    5 Init DB        test
2016-07-02T01:30:29.095425Z    5 Query  show databases
2016-07-02T01:30:29.096108Z    5 Query  show tables
2016-07-02T01:30:29.992895Z    5 Query  CREATE TABLE t (id INT, value VARCHAR(100))
2016-07-02T01:30:30.246081Z    5 Query  INSERT INTO t (id, value) VALUES (123, 'hoge')
2016-07-02T01:30:30.290345Z    5 Query  SELECT id, value FROM t
2016-07-02T01:31:07.003081Z    5 Quit
```

▼図2　general_logテーブルはCSV

```
# ls -l /usr/local/mysql/data/mysql↵
 ..省略..
-rw-r----- 1 mysql mysql    4218  7月  3 11:42 general_log.CSV
 ..省略..
```

フトに読み込ませて処理することもできます。

テーブル出力は便利なのですが、MySQLのデータディレクトリと異なる場所に出力することはできませんし、後述するログのローテーションはファイルとは異なる方法になります。すでにログ収集のしくみがある場合は、簡単にはその運用に乗せられないかもしれません。設定によってファイルとテーブルの両方に出力することもできるので、両方に出力しておいて用途によって扱いやすいほうを使うというのも良いでしょう（後述の「ログ出力先を切り替えるには」で解説）。

一般クエリログをテーブルに出力するように設定すると、mysqlデータベースのgeneral_logテーブルに記録されます（**図3**）。general_logテーブルのカラム「event_time」「thread_id」「command_type」「argument」はファイル出力時の情報と同じです。「user_host」カラムはユーザです。ファイル出力時には、あるレコードだけを見てもユーザがわかりませんでしたが、テーブル出力時には各レコードのカラムに記録されます。

スロークエリログ

スロークエリログはクライアントから発行されるクエリのうち、実行に一定時間以上かかったものを記録します。デフォルトでは10秒以上かかったログが出力されます。サーバ起動中に動的に有効と無効を切り替えることができますが、常に有効にしておいても良いでしょう。

ファイル形式

リスト5のように、先頭が「#」の行で、「クエリの終了時刻（Time）」「クエリ実行ユーザ（User@Host）」「クエリにかかった時間（Query_time）」「ロックにかかった時間（Lock_time）」「クエリ結果の行数（Rows_sent）」「クエリ実行時に走査した行数（Rows_examined）」が出力されます。終了時刻の形式は一般クエリログと同様にMySQL 5.6と5.7で異なります。

実際に発行したクエリのほかに「use」や「SET timestamp」が記録されます。useはそのデータベースで最初に出力されるクエリの前に出力さ

▼図3　general_logテーブル

```
mysql> SELECT * FROM mysql.general_log;
+----------------------------+----------------------------+-----------+-----------+--------------+-----------------------------------------------+
| event_time                 | user_host                  | thread_id | server_id | command_type | argument                                      |
+----------------------------+----------------------------+-----------+-----------+--------------+-----------------------------------------------+
| 2016-07-02 14:45:11.752609 | [root] @ localhost []      |         6 |         0 | Connect      | root@localhost on  using Socket               |
| 2016-07-02 14:45:11.753041 | root[root] @ localhost [] |         6 |         0 | Query        | select @@version_comment limit 1              |
| 2016-07-02 14:45:22.319805 | root[root] @ localhost [] |         6 |         0 | Query        | SELECT DATABASE()                             |
| 2016-07-02 14:45:22.320315 | root[root] @ localhost [] |         6 |         0 | Init DB      | test                                          |
| 2016-07-02 14:45:22.323684 | root[root] @ localhost [] |         6 |         0 | Query        | show databases                                |
| 2016-07-02 14:45:22.324346 | root[root] @ localhost [] |         6 |         0 | Query        | show tables                                   |
| 2016-07-02 14:45:28.546195 | root[root] @ localhost [] |         6 |         0 | Query        | CREATE TABLE t (id INT, value VARCHAR(100))   |
| 2016-07-02 14:45:28.547592 | root[root] @ localhost [] |         6 |         0 | Query        | INSERT INTO t (id, value) VALUES (123, 'hoge') |
| 2016-07-02 14:45:28.585006 | root[root] @ localhost [] |         6 |         0 | Query        | SELECT id, value FROM t                       |
| 2016-07-02 14:45:30.786595 | root[root] @ localhost [] |         6 |         0 | Quit         |                                               |
+----------------------------+----------------------------+-----------+-----------+--------------+-----------------------------------------------+
```

▼リスト5　スロークエリログ（MySQL 5.7）

```
# Time: 2016-07-02T07:21:26.477273Z
# User@Host: root[root] @ localhost []  Id:     9
# Query_time: 10.000544  Lock_time: 0.000000 Rows_sent: 1  Rows_examined: 0
use test;                          ←カレントデータベースをtestに切り替え
SET timestamp=1467444086;          ←時刻を合わせる
SELECT SLEEP(10);                  ←時間のかかったクエリ
```

れます。SET timestampはクエリの終了時刻を設定します。「#」で始まる行以外の行をそのまま再実行することで、時刻も合わせて同じクエリを再現できるようにするために、これらの情報が出力されていると思われます。

テーブルへの出力

一般クエリログと同様に、ファイルだけでなくテーブルに出力することもできます。テーブルに出力する際のメリットや注意事項も一般クエリログと同様です。

mysqlデータベースのslow_logテーブルに記録されます(**図4**)。slow_logテーブルのカラム「start_time」「user_host」「query_time」「lock_time」「rows_sent」「rows_examined」「thread_id」は、ファイル出力時の情報と同じです。「db」カラムはクエリが実行されたデータベースで、ファイル出力時にはuseで表されていたものです。「sql_text」カラムは実行されたクエリです。

バイナリログ

バイナリログには更新系のクエリが記録されます。更新系のクエリは一般クエリログにも記録されますが、一般クエリログには参照系クエリも含まれるので膨大な量になってしまうことがあります。参照系クエリの履歴は必要なく、データベースの更新履歴だけを保存しておきたい場合はバイナリログを取得するのも良いと思います。

ただし、バイナリログは通常のログとかなり異なります。ログファイル名も自由にはつけられませんし、名前のとおりバイナリで記録されるため、そのままではログの内容を確認することはできません(mysqlbinlogというツールを使うことでログファイルの内容を表示できます)。

通常、バイナリログは異なるサーバへのデータのレプリケーションのために使用されます。MySQLのレプリケーション機構は、あるサーバで実行された更新系のクエリを他のサーバで実行すれば同じデータになるはずだというアーキテクチャに基づいています。

バイナリログはレプリケーションだけではなく、ディスク故障などでデータが失われたときに元に戻すために使用することもできます。フルバックアップとそれを取得した以降のバイナリログがあれば、フルバックアップからデータをリストアした後に、バイナリログを適用すれば最新状態までデータを戻すことができます。

バイナリログはデフォルトではMySQLのデータ領域と同じディスクに出力されますが、ディスククラッシュに備えるためのバックアップ用途に使用する場合は、異なるディスクに出力されるように設定しておかないと意味がありません。

なお、バイナリログは一般クエリログやスロークエリログと異なり、動的に有効/無効を切り替えることはできません。mysqld起動時に設定しておく必要があります。

MySQLログの設定

MySQLの設定は、コマンドラインオプション、オプションファイル、システム変数による設定の3通りあります。

コマンドラインオプションはmysqld起動時にmysqldコマンドの引数として指定するオプションです。

オプションをファイルに書いておく(オプションファイルにする)ことで毎回同じオプション

▼図4 slow_logテーブル

```
mysql> SELECT * FROM mysql.slow_log;
+----------------------------+----------------+------------+-----------+-----------+---------------+------+----------------+-----------+-----------+---------------+-----------+
| start_time                 | user_host      | query_time | lock_time | rows_sent | rows_examined | db   | last_insert_id | insert_id | server_id | sql_text      | thread_id |
+----------------------------+----------------+------------+-----------+-----------+---------------+------+----------------+-----------+-----------+---------------+-----------+
| 2016-07-02 16:21:26.477273 | root[root] @ [] | 00:00:10.000544 | 00:00:00.000000 |         1 |             0 | test |              0 |         0 |         0 | SELECT SLEEP(10) |         9 |
+----------------------------+----------------+------------+-----------+-----------+---------------+------+----------------+-----------+-----------+---------------+-----------+
```

で起動することができます。同じオプションがファイルとコマンドラインの両方に指定された場合は、コマンドラインオプションが有効化されます。通常は、設定はオプションファイルに記述しておき、一時的に変更したい場合にはコマンドラインオプションで指定するのが良いでしょう。

また、オプションによってはmysqld実行中にシステム変数として動的に変更することができます。変更するにはSETクエリを使用します。これが3つめの設定方法です。

コマンドラインオプション

図5のように、mysqld起動時にオプションを指定します。有効／無効（真偽）を指定するオプションの値は、有効（真）はON/TRUE/1、無効（偽）はOFF/FALSE/0で指定します。値を指定しない場合は有効（真）になります。

オプションファイル

オプションファイル中の[mysqld]グループにオプションを記述しておくと、mysqldが起動時にそれを読み込み、コマンドラインオプションに指定されたのと同様に動作します（**リスト6**）。

オプションファイルは/etc/my.cnf、/etc/mysql/my.cnfなど、複数のファイルが有効です。mysqldがどのファイルから設定を読み込むかは`mysqld --help --verbose`で確認できます（**図6**）。

SETクエリ

mysqld起動中にSETクエリでシステム変数を設定することにより、動的に設定を変更することができます（**図7**）。システム変数名はオプション名の「-」を「_」に置き換えたものです。変更できないシステム変数もあります。また、システム変数を変更するにはクエリを実行するユーザに管理者権限が必要です。

エラーログの設定

エラーログはlog-errorオプションで設定します。log_errorシステム変数で値を参照することはできますが、値を変更することはできません。

log-errorオプションに値を指定した場合はそれがファイル名として使用されます（**図8**）。値を指定しない場合はデータディレクトリ（datadirシステム変数の値）の下の「ホスト名.err」が使用されます。常に有効にしておく

▼図5　オプションで一般クエリログを有効にする例

```
# mysqld --general-log --general-log-file=/var/log/mysql.log
```

▼リスト6　ファイルで一般クエリログを有効にする例

```
[mysqld]
general-log
general-log-file = /var/log/mysql.log
```

▼図6　有効なオプションファイルの確認

```
# mysqld --help --verbose
 ..省略..
Usage: /usr/local/mysql/bin/mysqld [OPTIONS]

Default options are read from the following files in the given order:
/etc/my.cnf /etc/mysql/my.cnf /usr/local/mysql/etc/my.cnf ~/.my.cnf
 ..省略..
```

▼図7　SETクエリで一般クエリログを有効にする例

```
mysql> SET GLOBAL general_log = ON;
mysql> SET GLOBAL general_log_file = '/var/log/mysql.log';
```

▼図8　コマンドラインオプションでエラーログの出力先を設定する例

```
# mysqld --log-error=/var/log/mysql.err
```

ために、オプションファイルに設定しておくのが良いでしょう（**リスト7**）。

一般クエリログの設定

一般クエリログの有効／無効はgeneral-logオプションまたはgeneral_logシステム変数で設定します。ログファイル名はgeneral-log-fileオプションまたはgeneral_log_fileシステム変数で設定します。general_log_fileが設定されていない場合は、データディレクトリの下の「ホスト名.log」が使用されます。

通常はデフォルトのままログ出力は無効の状態にしておき、必要なときにSETクエリでgeneral_logのON/OFFを切り替えるのが良いでしょう（**図9**）。

sql_log_offセッション変数を真に設定することで、そのセッション（接続）でだけ一時的にログを記録しないようにできます（**図10**）。パスワードのような機密にすべき文字列を含むクエリなど、ログに記録させたくないようなクエリを実行する場合に使用できます。

スロークエリログの設定

スロークエリログの有効／無効はslow-query-logオプションまたはslow_query_logシステム変数で設定します（**図11、リスト8**）。スロークエリログのファイル名はslow-query-log-fileオプションまたはslow_query_log_fileシステム変数で設定します。slow_query_log_fileが設定されていない場合は、データディレクトリの下の「ホスト名-slow.log」が使用されます。

long_query_timeシステム変数でクエリ実行時間の閾値を指定します。デフォルトでは10秒です。

ログ出力先を切り替えるには

一般クエリログとスロークエリログの出力先はファイルかテーブル、またはその両方に設定できます。log-outputオプションまたはlog_outputシステム変数で指定します。FILEを指定するとファイル、TABLEを指定するとテーブル、FILE,TABLEを指定するとファイルとテーブルの両方にログを出力します（**図12、リスト9、図13**）。

▼リスト7　オプションファイルにエラーログの出力先を記述する例

```
[mysqld]
log-error = /var/log/mysql.err
```

▼図9　SETで一般クエリログを有効にする

```
mysql> SET GLOBAL general_log = ON;
```

▼図10　一時的に一般クエリログを記録しない

```
mysql> SET sql_log_off = ON;
mysql>  この間のクエリはログに記録されない
mysql> SET sql_log_off = OFF;
```

▼リスト8　オプションファイルにスロークエリログの出力先を記述する例

```
[mysqld]
slow-query-log = ON
slow-query-log-file = /var/log/mysql-slow.log
long-query-time = 1
```

▼図12　コマンドラインオプションでファイルとテーブル両方に出力を設定する例

```
# mysqld --log-output=FILE,TABLE
```

▼リスト9　オプションファイルにファイルとテーブル両方の出力を記述する例

```
[mysqld]
log-output = FILE,TABLE
```

▼図13　SETクエリでファイルとテーブル両方の出力を有効にする例

```
mysql> SET GLOBAL log_output = 'FILE,TABLE';
```

▼図11　コマンドラインオプションでスロークエリログの出力先を設定する例

```
# mysqld --slow-query-log=ON --slow-query-log-file=/var/log/mysql-slow.log --long-query-time=1
```

バイナリログの設定

バイナリログはlog-binオプションで設定します（**図14、リスト10**）。単純にlog-binだけを指定するとデータディレクトリの下の「ホスト名-bin」がファイル名のベース名として使用されます。log-binオプションに値を与えるとその値がファイル名のベース名として使用されます。実際のファイル名は「ベース名.000001」となり、拡張子部分は必要に応じてカウントアップされていきます（**図15**）。log_binシステム変数で有効か無効かの状態を、log_bin_basenameシステム変数でベース名を参照することはできますが、変更することはできません。

バイナリログはもともとレプリケーション機能の一部なので、ログ取得とは直接関係ないserver-idというオプションも指定する必要があります。レプリケーションで使用するにはサーバ間でユニークになる数値を指定する必要がありますが、単にバイナリログを取得するためだけであれば値は何でもいいので、server-id=1などを指定しておけばよいでしょう。

binlog-format=STATEMENTは人間が読んでもわかるように、レコードデータではなくクエリを記録するためのオプションです。レプリケーションやバックアップ用途では指定する必要はありません。

ログファイルの切り替え

同じファイルにログを出力し続けていると、ディスク使用量を圧迫しますし、1ファイルのサイズが大きくなるとファイルの中から何かを探し出す処理も大変になります。そのため、期間やサイズで出力先ファイルを切り替えるような運用が一般的だと思います。

エラーログファイル、一般クエリログファイル、スロークエリログファイルを切り替えるには、リネームしてからログファイルのフラッシュをします（**図16**）。ログファイルが新しく作成され、これ以降のログは新しいファイルに出力されます。古いファイルは使用されなくなるので、ほかの場所に移動するなり、削除するなり自由に扱うことができます。

ログのフラッシュ方法

FLUSH LOGSクエリを使用すると、すべてのログファイルをクローズして再オープンします。特定のログファイルだけをフラッシュす

▼リスト10　オプションファイルにバイナリログの出力を記述する例

```
[mysqld]
log-bin = /var/log/mysql/mysql-bin
server-id = 1
binlog-format = STATEMENT
```

▼図14　コマンドラインオプションでバイナリログの出力を設定する例

```
# mysqld --log-bin=/var/log/mysql/mysql-bin --server-id=1 --binlog-format=STATEMENT
```

▼図15　バイナリログファイル

```
# ls -l /var/log/mysql
-rw-r----- 1 mysql mysql 154  7月16 10:04 mysql-bin.000001
-rw-r----- 1 mysql mysql  32  7月16 10:04 mysql-bin.index
```

▼図16　エラーログファイルを切り替える例

```
# mv mysql-error.log mysql-error.log-20160818
# mysql -uroot -e 'FLUSH ERROR LOGS'
```

ることもできます（図17）。

mysqladminコマンドのflush-logsサブコマンドでも同様の操作が可能です（図18）。なお、MySQL 5.7より前のバージョンでは特定のログファイルだけを指定することはできません。

mysqldに対してSIGHUPシグナルを発行するとすべてのログファイルをフラッシュします。ただしSIGHUPはログファイルのフラッシュだけでなく、権限テーブルのリロードや、スレッドキャッシュ、ホストキャッシュのフラッシュなども行われるので、ログファイルのフラッシュのためだけに使用するべきではありません。

バイナリログの切り替え

バイナリログはフラッシュするとカウントアップした新しいファイル名に出力するため、手動でリネームする必要はありません（図19）。

バイナリログは古いものも含めてmysqldに管理されているため、勝手に移動したり削除してはいけません。不要になったバイナリログを削除するにはPURGE BINARY LOGSクエリを使用します（図20）。

expire_logs_daysシステム変数に日数を設定しておけば、その日数以上経過した古いバイナリログはフラッシュ時に自動的に削除されます。

テーブルに出力している場合

ログをテーブル出力している場合はファイルのように簡単にはいきません。ログ出力先のテーブルはTRUNCATEですべてをクリアすることはできますが、一部をDELETEすることはできません。

古いログを残したままテーブルを切り替えるには、ログテーブルと同じ型の新しいテーブルを作成し、テーブルをリネームします（図21）。このようにすれば古いログはoldテーブルに残り、新しいログは新しく作られたログテーブルに出力されるようになります。SD

▼図17　フラッシュの方法一覧

```
mysql> FLUSH LOGS;                ←すべてのログファイル
mysql> FLUSH ERROR LOGS;          ←エラーログ
mysql> FLUSH GENERAL LOGS;        ←一般クエリログ
mysql> FLUSH SLOW LOGS;           ←スロークエリログ
mysql> FLUSH BINARY LOGS;         ←バイナリログ
```

▼図18　mysqladminコマンドでのフラッシュ方法一覧

```
# mysqladmin flush-logs           ←すべてのログファイル
# mysqladmin flush-logs error     ←エラーログ
# mysqladmin flush-logs general   ←一般クエリログ
# mysqladmin flush-logs slow      ←スロークエリログ
# mysqladmin flush-logs binary    ←バイナリログ
```

▼図19　バイナリログのフラッシュ

```
# ls↵
mysql-bin.000001  mysql-bin.index
# cat mysql-bin.index↵
/var/log/mysql/mysql-bin.000001
# mysql -uroot -e 'FLUSH BINARY LOGS'↵
# ls↵
mysql-bin.000001  mysql-bin.000002  mysql-bin.↵
index
# cat mysql-bin.index↵
/var/log/mysql/mysql-bin.000001
/var/log/mysql/mysql-bin.000002
```

▼図20　古いバイナリログの削除

```
# ls↵
mysql-bin.000001  mysql-bin.000003  mysql-bin.000005  mysql-bin.000007
mysql-bin.000002  mysql-bin.000004  mysql-bin.000006  mysql-bin.index
# mysql -uroot -e "PURGE BINARY LOGS TO 'mysql-bin.000003'"↵
# ls↵
mysql-bin.000003  mysql-bin.000005  mysql-bin.000007
mysql-bin.000004  mysql-bin.000006  mysql-bin.index
```

▼図21　一般クエリログテーブルを切り替える

```
mysql> USE mysql;↵
mysql> CREATE TABLE general_log_new LIKE general_log;↵
mysql> RENAME TABLE general_log TO general_log_old, general_log_new TO general_log;↵
```

Sambaの詳細なログ設定と活用

パフォーマンスと実益とのバランスをとって出力しよう

Author たかはしもとのぶ　Mail monyo@monyo.com　Twitter @damemonyo

本章では、Windowsファイルサーバの機能を提供するSambaのログ設定と、その活用法について解説します。

Sambaについて

SambaはWindowsファイルサーバの機能を提供するプロダクトとして、一般的なLinuxディストリビューションには必ずパッケージが用意されているソフトウェアです。執筆時点での最新版は7月7日にリリースされたSamba 4.4.5で、現在も活発に開発が行われています。

Sambaのアーキテクチャは何度か大きく変更されていますが、今回解説するログ機構については、細かい機能強化はあるものの、基本的には昔から変わっていません。

Samba標準のログファイルの活用

Sambaのログファイルは、標準ではsmbdやnmbdといったデーモンプロセスごとにlog.smbdやlog.nmbdといったファイル名となっています。ログの記録例を**リスト1**に示します。

図1からもわかるとおり、Sambaのログは Apacheでいうところのアクセスログよりはエラーログに近い位置づけの内容となっています。

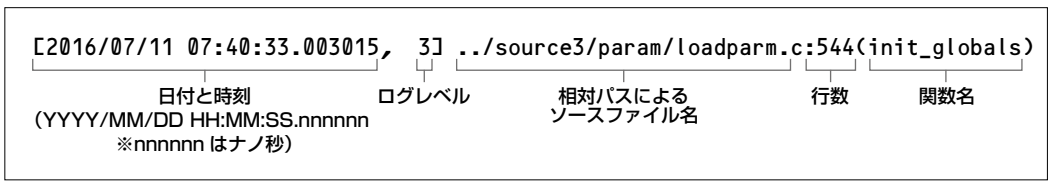
ログの内容は、定型的なヘッダ行と非定型のメッセージ行の2行セットで構成されています。ヘッダ行の形式については、**図1**のようになっ

▼リスト1　smbd起動時にlog.smbdに記録される内容

```
[2016/07/11 07:40:33,  0] ../source3/smbd/server.c:1241(main)
  smbd version 4.2.10 started.
  Copyright Andrew Tridgell and the Samba Team 1992-2014
[2016/07/11 07:40:33,  2] ../source3/lib/tallocmsg.c:124(register_msg_pool_usage)
  Registered MSG_REQ_POOL_USAGE
[2016/07/11 07:40:33,  2] ../source3/lib/dmallocmsg.c:78(register_dmalloc_msgs)
  Registered MSG_REQ_DMALLOC_MARK and LOG_CHANGED
[2016/07/11 07:40:33.002931,  3] ../source3/param/loadparm.c:3653(lp_load_ex)
  lp_load_ex: refreshing parameters
[2016/07/11 07:40:33.003015,  3] ../source3/param/loadparm.c:544(init_globals)
  Initialising global parameters
```

※ここでは後述するログレベル3のログの一部を例示しています。

▼図1　ヘッダ行の形式



ています。

メッセージ行についてはメッセージそのものになります。とくに定型的な形式はありません。

ログファイルの文字コードについて

メッセージ行に日本語ファイル名など、英語以外の文字列などが含まれる場合の文字コードは、Sambaのunix charsetパラメータで制御されます。パラメータのデフォルト値はUTF-8です。値を変更することもできますが、このパラメータはファイル名の文字コードをはじめ、各所に影響しますので慎重に行ってください。

ヘッダ行のチューニング

Sambaにはヘッダ行の記録内容を増減させるパラメータがいくつか用意されています[注1]。次の設定を行うことで、各ヘッダ行ごとにプロセスのPIDとログ記録時のプロセスの実行権限(euid、egid、uid、gid)を記録させることができます。

```
debug pid = yes
debug uid = yes
```

※これ以降の設定も含め、明示的に解説しない限りパラメータはglobalセクションに設定します。

この設定を行った際のログの記録例を**リスト2**に、ヘッダ行の形式を**図2**に示します。

Sambaを構成するsmbdプロセスはクライアントからのアクセスごとに1プロセスが起動されるため、デフォルトの記録では、Sambaが多数のクライアントからアクセスされている際に、複数のsmbdプロセスの記録が入り混じってしまい、解読が困難です。またsmbdプロセスは基本的に認証されたユーザの権限で動作しますが、一時的にroot権限で実行されることもあり、アクセス権がらみのトラブルシューティングで重要な要素となってくることもあります。先ほどの設定により、これらの情報がログに記録されます。

ログ詳細度の設定

図1のとおりSambaのログには、メッセージの重要度(詳細度)を示すログレベルが必ず設定されます。最も重要とされるログはログレベル0となり、以下重要度の順にレベル10までの値が割り当てられています。

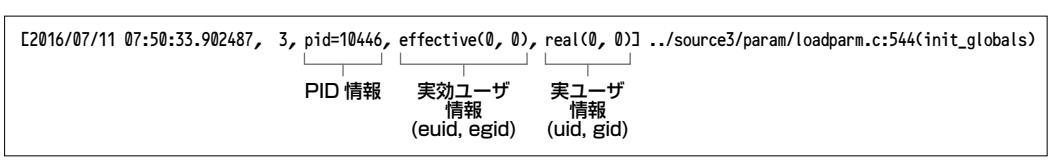
log levelパラメータ(debug levelという別名があります)により、記録するログレベルを制御します。このパラメータのデフォルト値は0ですので、最重要のログ以外は記録されません。より詳細なログを記録するためには、1以上の値を明示的に指定する必要があります。たとえば次の設定を行うことで、ログレベル3まで(レ

注1) これ以外にもいくつかのパラメータがありますが、あまり有用でないため解説は割愛します。

▼リスト2 smbd起動時にlog.smbdに記録される内容(ヘッダ行のチューニング後)

```
[2016/07/11 07:50:33,  2] ../source3/lib/dmallocmsg.c:78(register_dmalloc_msgs)
  Registered MSG_REQ_DMALLOC_MARK and LOG_CHANGED
[2016/07/11 07:50:33.902445,  3, pid=10446, effective(0, 0), real(0, 0)] ../source3/param/➥
loadparm.c:3653(lp_load_ex)
  lp_load_ex: refreshing parameters
[2016/07/11 07:50:33.902487,  3, pid=10446, effective(0, 0), real(0, 0)] ../source3/param/➥
loadparm.c:544(init_globals)
 Initialising global parameters
```

▼図2 ヘッダ行の形式(ヘッダ行のチューニング後)



ベル0から3）のログが記録されます。

```
log level = 3
```

値としては、0から10を指定します[注2]。

このパラメータの値を大きくすればするほど詳細な情報を記録できますが、その分ログファイルの容量が増加することに加え、書き込みのオーバーヘッドでSamba自身の動作も顕著に遅くなります。このため通常運用時には、このパラメータの値は0または1程度に留めておき、重要度の高いメッセージのみが記録されるようにしておくことをお勧めします。

ログレベルの一時的な変更

前述したとおり、実運用環境でログレベルをむやみに上げることはお勧めできません。一方でトラブル発生時にはなるべく詳細な情報を記録させたいところです。ログレベルを恒久的に変更する場合は、前述したlog levelパラメータを設定すればよいのですが、設定を反映するにはプロセスの再起動などが必要なため気軽には行えません。

こうしたときには、smbcontrolコマンドによりログレベルを一時的に変更する機能が便利です。実行例を**図3**に示します。

この例ではログレベルを10に変更後、すぐに0に戻しています。コマンドの文法を次に示します。

```
smbcontrol pid debug N
```

pidとしては、PIDを示す数値のほか、smbdやnmbdといったプロセス名を指定することも可能で、その場合は該当するプロセスすべてが対象となります。

図4のようにdebuglevelオプションを指定することで、現在のログレベルを表示することもできます。

「all:1」という文字列から、現在のログレベ

▼図3　smbcontrolコマンドによるログレベル変更

```
# smbcontrol smbd debug 10↵
# smbcontrol smbd debug 0↵
```

▼図4　現在のログレベルの表示

```
# smbcontrol smbd debuglevel
PID 13973: all:1 tdb:1 printdrivers:1 lanman:1 smb:1 rpc_parse:1 rpc_srv:1 rpc_cli:1 ➥
passdb:1 sam:1 auth:1 winbind:1 vfs:1 idmap:1 quota:1 acls:1 locking:1 msdfs:1 dmapi:1 ➥
registry:1
```

注2）passwd chatパラメータのデバッグの際にパスワード文字列を平文で記録させたい場合など特殊な状況では、より大きい値を指定することが必要な場合もあります。

ログクラスの設定

Sambaのログにはログクラスという概念があり、ログのカテゴリごとにログレベルを指定できるようになっています。設定例を次に示します。

```
log level = 3 passdb:5 auth:10 winbind:2
```

ただし、どのメッセージがどのログクラスに属しているかはソースコードを参照するか、

```
debug class = yes
```

を設定してログクラスをログに記録させるようにしてしばらく様子を見たうえで設定していく必要があります。実質的には、ソースコードを参照できる方でないと活用は難しいでしょう。

ルが1であることを確認できます[注3]。

トラブルの再現手順が確立している場合は、再現直前にログレベルを10にして、再現後すぐに元の値に戻すようにすればよいでしょう。

なかなか再現しないような場合、とりあえずログレベルを3にして様子を見ることをお勧めします。ログレベル3は（開発者でなく）システム管理者用として十分詳細なログが記録されます。

ログファイル名の変更

log fileパラメータにより、ログファイル名を変更できます。これにはSamba変数を設定できるので、たとえば次のように設定することで、すべてのクライアント（IPアドレス）ごとに個別のログファイルを作成できます。

```
log file = /var/log/samba/log.smbd.%I
```

注3）それ以外の文字列はコラムで解説したログクラスごとのログレベルになります。

この場合はクライアントのIPアドレスごとにファイルが別に作成されることにより、多数のログファイルが作成されますので、注意してください。

syslogとの連携

Sambaの標準では、Samba固有のログファイルへの記録と併せてsyslogへのログ記録も行われます。syslogへのログ記録は、syslogパラメータにより制御されます。

デフォルトは、

```
syslog = 1
```

となっており、log levelが1未満、つまり0のログのみがSambaのログファイル以外にsyslogにも記録されます。

syslogにはログの種類を表す「ファシリティ」と、ログの緊急度を表す「レベル」という概念があります。ファシリティについてはdaemonになっていますので、ログは/var/log/messages

column

特定のクライアント間の通信に限って詳細なログを取得する

トラブルシューティングの際には、Sambaサーバ全体のパフォーマンスを落とさないためにも、問題のあるクライアントとの間のログだけを詳細に記録したいこともあると思います。

こうした場合に、たとえばクライアントのIPアドレスを意味するSamba変数「%I」を用いて**リスト3**のような設定を行ったうえで、**リスト4**のようなsmb.conf.192.168.1.1というファイルを別に作成しておくことで、192.168.1.1のクライアントからのアクセスのみログレベルを3にするといった設定を行うことができます。

リスト3の設定が行われていると、192.168.1.1のクライアントからアクセスがあった場合にincludeパラメータによりsmb.conf.192.168.1.1というファイルの内容がその位置に読み込まれます。**リスト4**ではlog levelパラメータの値を3に設定しています。同一のパラメータを複数回設定した場合は、最後に設定したパラメータの値が有効となりますので、これにより、当該クライアントからアクセスした場合のみlog levelパラメータの値が3に設定されます。

それ以外のIPアドレスの場合はincludeパラメータで指定したファイルが存在しないため、このパラメータ自体が無視されlog levelパラメータの値は1が設定されます。

▼リスト3　smb.confファイルの設定

```
log level = 1
include = smb.conf.%I
```

▼リスト4　smb.conf.192.168.1.1ファイルの内容

```
log level = 3
log file = /var/log/samba/log.smbd.192.168.1.1
```

など標準のログファイルに記録されます[注4]。レベルについてはSambaのログレベルに対応しています。詳細は**表1**を参照してください。

Samba固有のログファイルを活用する場合、syslogへ同じ内容を記録してもあまり有用ではないと思います。

```
syslog = 0
```

と設定することで、この機能を無効にすることをお勧めします。

ログファイルの活用

ログに記録される情報はSambaの内部動作に関する情報ですので、基本的にはトラブル発生時の使用が想定されています。

トラブル発生時には、発生時刻周辺のログのメッセージ行の内容を確認していく形で個別に切り分けを行っていくことになります。ソースコードが読める方であれば、ヘッダ行にメッセージを記録したソースコードの部位に関する情報もあるので参考になります。

とはいえ、これらのメッセージはSambaの内部動作についての情報ですので、英語のメッセージから意味が自明である場合をのぞき、正攻法で原因を追っていくのは難しいと思います。そのため、安直ではありますが、次のような作業を行ってみることをお勧めします。

①ログファイルから怪しそうな行（Errorといった文字列があるなど）を抽出する
②そこに記録されているメッセージ行の内容を検索エンジンで検索してみて、類似のトラブルに関する情報がないかを確認する

ありがちなトラブルであれば、これで解決することも多いと思います。残念ですが、解決しなかった場合はメッセージの内容や記録しているソースコードの部位から原因を類推していくしかありません。

注4）コンパイル時以外には変更できません。

ファイルサーバへのアクセスログを取得する

ファイル共有の監査機能を提供するfull_auditモジュールを活用することで[注5]、Sambaで構築したファイルサーバに対する詳細なアクセスログ（監査ログ）を取得できます。このモジュールは標準では有効化されていないため、以降で設定を有効化する手順について解説します。なお、ファイル共有の一般的な設定については、誌面の都合上解説を割愛していますのでご注意ください。

full_auditモジュールの有効化とsyslog設定

full_auditモジュールを有効化するには、該当の共有で次の設定を追加します。

```
vfs objects = full_audit
```

すでにvfs objectsパラメータが設定されている共有の場合は、パラメータ行の末尾にfull_auditというキーワードをスペースで区切って追加します。

full_auditモジュールはsyslogにログを出力しますので、syslogの設定も意識する必要があります。デフォルトではuser.noticeというファシリティとレベルでログが出力されますので、/var/log/messagesや/var/log/syslogといった標準のログファイルに大量のログが出力されま

注5）Sambaでアクセスログを記録するモジュールとしてはほかにもauditとextd_auditというものがありますが、機能が不十分なため解説は省略します。

▼表1　log levelとsyslogのレベル

log level	レベル
0	error
1	warning
2	notice
3	info
4以上	debug

す。このままではほかのログが埋もれてしまって見づらくなってしまうため、アクセスログは別のファイルに切り出しておきましょう。

まずはfull_auditモジュールの設定で、ログを出力するファシリティをlocal1など未使用のものに設定します。

```
full_audit:facility = local1
```

ついでsyslog設定ファイル(CentOSでは/etc/rsyslog.conf)に次のような行を追加してlocal1ファシリティのログを適切なファイル(たとえば/var/log/samba/access.log)に出力するように設定します。

```
local1.*        /var/log/samba/access.log
```

最後に、標準のログファイルにSambaのアクセスログが出力されないよう、除外設定を行います。CentOSの場合はrsyslog.confの次の個所にlocal1.noneというキーワードを設定することで、ファシリティがlocal1のメッセージの出力を抑止できます(**図5**)。

そのほかのディストリビューションでも同様の設定を行います。設定を行ったら、次のようにして忘れずにログファイルを作成してください。

```
# touch /var/log/samba/access.log↵
```

ファイルを作成したら、設定を反映させるために設定ファイルの再読み込み(もしくはsyslogサービスの再起動)を行います。これで**リスト5**のようなアクセスログが/var/log/samba/access.log(だけ)に出力されるようになります。

ここでは解説しませんが、実際に運用するうえではローテーションやバックアップの設定も忘れずに行ってください。

アクセスログの形式

アクセスログの各行の形式は**図6**のようになっています。smbd_audit:に続いてフィールドが「|」で区切って記録されます。

・**プレフィックス**
デフォルトでユーザ名と接続元IPアドレスが|で区切った形式で記録されます。

▼図5　除外設定の例

```
*.info;local1.none;mail.none;authpriv.none;cron.none        /var/log/messages
          追加
```

▼リスト5　アクセスログの出力例

```
Jul 11 08:57:32 centos70 smbd_audit: monyo|192.168.135.1|connect|ok|monyo
Jul 11 08:57:32 centos70 smbd_audit: monyo|192.168.135.1|realpath|ok|/home/monyo
Jul 11 08:57:32 centos70 smbd_audit: monyo|192.168.135.1|realpath|ok|/home/monyo
Jul 11 08:57:32 centos70 smbd_audit: monyo|192.168.135.1|stat|ok|/home/monyo
```

▼図6　アクセスログの形式

```
Jul 11 08:57:32 centos70 smbd_audit: monyo|192.168.135.1|realpath|ok|/home/monyo
```

syslogのヘッダ情報(日付と時刻、ホスト名) / smbd_audit: / プレフィックス / 操作の名称 / 結果 / 対象のパス名

複数の共有でこの設定を有効化し、かつ同一のファイルにアクセスログを記録するように設定した場合は、たとえば次のようにしてログに共有名も記録しておいた方がよいでしょう

```
full_audit:prefix=%u|%I|%S
```

・操作の名称

表2のような名称が記録されます。これらは一部をのぞきLinuxのシステムコール関数名に対応していますので、非常に詳細な挙動を記録できることがわかります

・操作の結果

okもしくはfailが記録されます

・対象のパス名

操作の対象となるパス名が、共有トップからの相対パスで記録されます。操作の種類によってはこのフィールドが存在しない場合もあります

デフォルトでは各操作の成功、失敗がすべて記録されるため、膨大なログが出力されます。このため運用を行ううえでは、次で説明するカスタマイズを行い、記録する操作を制限することがほぼ必須です。

アクセスログのカスタマイズ

full_audit:success と full_audit:failure パラメータにより、記録する操作を制限します。設定例を**リスト6**に示します。

ここでは、アクセスログとしての使用を想定し、

- 共有への接続と切断（connect、disconnect）
- ファイル、ディレクトリの作成、削除、リネーム（ただしファイルの作成は含まない）（mkdir、rmdir、rename、unlink）
- ファイルの読み書き（read、pread、write、pwrite、sendfile）

の成功と、共有への接続と切断の失敗のみを記録しています。

ファイルの作成やオープンも記録したい場合は、次のような設定を行えばよいでしょう。

▼表2　監査可能な主要な操作

項目	操作の名称	説明
	all	すべての操作を含む
	none	すべての操作を除外
ファイル共有の操作	connect	ファイル共有へのアクセス
	disconnect	ファイル共有からの切断
ディレクトリの操作	opendir	ディレクトリのオープン
	readdir	ディレクトリエントリの読み取り
	mkdir	ディレクトリの作成
	rmdir	ディレクトリの削除
	closedir	ディレクトリのクローズ
ファイルの操作	open	ファイルのオープン
	close	ファイルのクローズ
	create_file	ファイルの作成
	read	ファイルの読み取り
	pread	同上
	write	ファイルの書き込み
	pwrite	同上
	rename	ファイルのリネーム
	unlink	ファイルの削除
	chmod	ファイルのパーミッション変更
	fchmod	同上
	chown	ファイルの所有者変更
	fchown	同上
	ftruncate	ファイルサイズの切り詰め
	symlink	シンボリックリンクの作成
	link	ハードリンクの作成
ACLの操作	fset_nt_acl	ACLの設定
	set_nt_acl	
	chmod_acl	ACLの変更
	fchmod_acl	

▼リスト6　アクセスログの設定例

```
full_audit:failure = connect disconnect
full_audit:success = connect disconnect mkdir rmdir rename read pread write pwrite sendfile↵
Tunlink
```

```
full_audit:success = connect disconnect ➥
open close mkdir rmdir rename create_file ➥
read pread write pwrite sendfile unlink➥
```

ただし、とくにディレクトリのオープン操作はディレクトリを一覧するたびに発生しますので、かなりの量のログが記録されます。

リスト6の設定を行った状態で、共有トップにあるtestフォルダ内にあるtest.txtファイルに書き込んだ際に記録されたログの一部を**リスト7**に示します。

これでもかなり大量のログが出力されますが、何とか読み解くことはできるのではないかと思います。

ここでは、パフォーマンスと実益とのバランスをとった設定例として**リスト6**の設定を示しましたが、もちろんセキュリティを優先する場合はより取得する操作を増やしてもかまいません。究極的にはログの容量とSamba、syslogサービス、ディスクI/Oなどのパフォーマンスに問題が発生しなければ、すべての操作を記録することも仕様上はできます。

実際の環境で記録する項目を決めるうえでは試験環境で擬似的にアクセスを行ってどのようなログがどの程度の量出力されるかを確認しつつ、試行錯誤しながら決めていくのがよいでしょう。SD

▼リスト7　アクセスログの例

```
Jul 11 13:20:45 centos70 smbd_audit: monyo|192.168.135.1|monyo|pread|ok|test/test.txt
Jul 11 13:20:47 centos70 smbd_audit: monyo|192.168.135.1|monyo|pread|ok|test/test.txt
Jul 11 13:20:47 centos70 smbd_audit: monyo|192.168.135.1|monyo|pwrite|ok|test/test.txt
```

column

アクセスログを簡易に取得する

実は、Sambaサーバへのアクセス履歴だけであれば本文で解説したfull_auditモジュールを使わなくても簡易に取得できます。

```
utmp = yes
```

という設定を追加することで、Sambaサーバへのアクセスが、Linux標準のログイン履歴に記録されるようになります。この情報はlastコマンドで参照できます。実行例を**図7**に示します。

-wオプションを省略した場合、ユーザ名やホスト名などで8バイトを超える部分は省略されます。2列目の仮想端末名が「smb/」から始まっている行がSambaサーバへのアクセスとなります。

lastコマンドから取得できるのは「誰がいつどこからアクセスしたか」だけですが、パラメータ1つで情報が取得できますので、最低限のアクセス履歴を取得したいという場合は検討する価値があると思います。

▼図7　lastコマンドの実行例

```
$ last -w↵
monyo    smb/24053837 192.168.135.1    Mon Jul 11 10:33   still logged in
monyo    smb/78103915 192.168.135.1    Mon Jul 11 10:31 - 10:33  (00:01)
monyo    smb/27333739 192.168.135.1    Mon Jul 11 10:31 - 10:31  (00:00)
monyo    smb/14568390 192.168.135.1    Mon Jul 11 10:31 - 10:31  (00:00)
```

ユーザ名　仮想端末名　ログイン元ホスト　ログイン時間帯

第5章

マーケティングにも使えるログ設計とは

アプリケーションログで何を記録し、どう可視化するか

Author 吉野 哲仁（よしの てつひと）
ヤフー㈱ ショッピングカンパニー テクニカルディレクター

よく大手のWebサービスでは、ユーザの行動分析やマーケティングのためにログ分析が行われています。そのような用途で活用されるのは、OSやWebサーバのログよりも、むしろアプリケーションで独自に出すログです。しかし、何を記録し、どのように活かせば良いのでしょう。Yahoo! JAPANのノウハウの一端を紹介します。

ログはマーケティングでも活用される時代に

最近では、“ビッグデータ”や“DevOps”のキーワードがもてはやされていることもあり、ログの重要性がますます大きくなってきています。今やシステムの安定稼働だけでなく、行動分析、マーケティングなど、ログの活用範囲は多岐に渡ります。

本章ではアプリケーションのログをどのように残すべきか、Yahoo!ショッピングの事例を交えながら紹介していきます。

どのようなログを残すべきか

基本は“5W1H”

ログを記録するうえで、基本となる考え方は“5W1H”です。「いつ（When）」「どこで（Where）」「誰が（Who）」「何を（What）」「なぜ（Why）」「どのように（How）」。ログを設計するときには、まずこのポイントが押さえられているかチェックしましょう。

いつ（When）

“いつ”とは、基本的には“時間”のことですが、ひとことで時間と言っても、目的によって記録のしかたは異なります。たとえば時間の精度の場合、アクセスログであれば「秒」までで良いですし、多くのトランザクションが同時に集中するシステムでは、前後関係を厳密に見るために「ミリ秒」まで記録する必要があるかもしれません。また、システムのパフォーマンスを計るためであれば「マイクロ秒」まで必要なこともあるでしょう（**表1**）。

目的に応じて、正しい精度の時間を残すようにしましょう。

どこで（Where）

Webサービスにおいては、URLがベースとなります。エラーログを出す際には発生位置（スタックトレース）が出ているといいでしょう。

誰が（Who）

ユーザID、IPアドレスなど、“誰が”を識別する情報はさまざまです。“誰が”の情報は不

▼表1　目的に応じた時間の精度の例

精度	目的
秒	アクセスログ、バッチの処理ログの場合
ミリ秒	在庫システム、カートシステムなど多量のトランザクションを処理するシステムの場合。APIのレスポンスを計測する場合
マイクロ秒	ネットワークやCPUのパフォーマンスを計測する場合

正対策、行動分析、トラブルシューティングを行う際にとくに重要な情報です。目的に応じて必要な情報を選択しましょう。例として、次のものが挙げられます。

・ユーザID
・セッションID
・Cookie
・端末ID
・IPアドレス
・MACアドレス
・User-Agent

何を（What）

「どんな情報が送信（処理）されたか」を記録します。とくにトラブルシューティングにおいては重要な情報です。リクエスト情報やアプリケーションのオブジェクト情報をそのままダンプ（Dump）するケースが多いですが、情報量がたいへん多いため、慣れてきたら出力情報を適切に選定しましょう。

ただし、情報によってはセキュリティの観点からログに記録するべきでないものもあります（詳細は後述）ので、これらのポイントも同時にチェックが必要です。

なぜ（Why）

アプリケーションのエラーログにおいては、エラーIDやAPIのエラーレスポンスなど、エラーの原因を出力します。

どのように（How）

どういった内容の操作（アクション）をされたかを記録します。「どのボタンを押したか」「どのリンクをクリックしたか」「（APIやバッチの場合）どこからCallされたか」が押さえてあればいいでしょう。

◆ ◆ ◆

あらためて整理すると、Webサービスの場合は**表2**のような情報を記録することになるでしょう。

▼表2　Webサービスにおける5W1Hの例

5W1H	記録すべき情報
いつ（When）	時間
どこで（Where）	URL
誰が（Who）	ユーザIDまたはセッションID
何を（What）	リクエスト情報
なぜ（Why）	エラーID
どのように（How）	アクション（ボタン押下、クリックなど）

ログに残してはいけない情報

個人情報

個人情報はログ出力しない。これはまず原則として必要な考え方です。「どの情報が個人情報に該当するか」はデータベースの構成などによって違うため、まずはセキュリティポリシーなどの自社の定義を確認しましょう。

次のように、何らかのロジックでハッシュ化すれば、ログ出力してもOKとなる場合もあります。

```
矢風太郎
  ↓  ハッシュ化
rPiFpYF3U_ruk6Le9M6eGvHB
```

Yahoo! JAPANにおいて、一般の社員が業務の中で個人情報を扱う場合は、厳格なセキュリティルールのもとに行われています。Yahoo! JAPANでは、通常のネットワーク内で動くアプリケーションサーバでは、次のような情報のログ出力は禁止しています。

Yahoo!ショッピングにおける個人情報の例

・顧客の住所
・顧客の氏名
・顧客の電話番号
・顧客のクレジットカード番号
・顧客のメールアドレス
・注文の要望欄

上記の情報は、ログ上では*でマスクされます（**リスト1**）。

▼リスト1　ログ出力イメージ（個人情報は＊でマスクされる）

```
[2016-07-04 12:00:23] [info] Change order order_time ( => 20160704120022)
[2016-07-04 12:00:23] [info] Change order total_price (0 => 670)
[2016-07-04 12:00:23] [info] Change order seller_id ( => 'test-store')
[2016-07-04 12:00:23] [info] Change order bill_first_name (***** => *****)
[2016-07-04 12:00:23] [info] Change order bill_last_name (***** => *****)
```

企業秘密

おもに「売上データ」や「課金データ」など内部情報にあたるものです。「個別の計算結果は良いが、合算した値は出力NG」など、細かい取り決めがある企業も存在します。また、バッチの計算結果の整合性チェックにも使う場合があるため、出力の必要性をよく検討しましょう。

そのほか決めておいたほうが良いこと

日付／時刻の形式

プログラミング言語やソフトウェアによって微妙にフォーマットが違うので、見やすさを重視するなら、できるだけ統一しましょう。

見やすさを重視した日付、時刻形式の例

```
yyyy/MM/dd hh:mm:ss
```

ログ収集システムに連携する場合は、UNIX TIME（1970年1月1日0時0分0秒からの形式的な経過秒数）のほうが都合が良い場合もあります。

文字コード

開発者はあまり意識しませんが、他システムと連携する際には、意外と問題になります。

ログ保管場所

これも言語やソフトウェアによってデフォルトのパスが違うので、できるだけ同じディレクトリの下で管理できるようにするのが理想です。

ログ保管場所（パス）の例

```
/var/log/[hoge]/[アプリケーション名]/
[ファイル名].log
```

ログファイル種類

ログの用途によって、出力ファイルを分けましょう。アクセスログ、バッチログ、スローログ、エラーログなど、最近はログの用途が多くなってきたため、出力ファイルの種類も多様になってきています。

ローテート間隔

ログファイルをどのような間隔でローテートするかを決めます。基本的には日次で行うことが多いですが、サイズが大き過ぎると解析にも時間がかかるので、サイズ単位や数時間ごとのローテーションも考えましょう。

また、ローテートしたあとは圧縮することも忘れずに。ただ、ローテートの間隔設定を間違えると1つのログファイルが巨大になってしまい、gzipなどのデータ圧縮処理でCPUを使い過ぎてサービスに影響が出てしまうこともあるので気をつけましょう（以前、Yahoo!ショッピングでも同じことがありました）。

保存期間

ローテート後のファイル名はどうするか、ローテートしたファイルはいつまで保存されるかを決めましょう。企業によっては、内部統制で最低限の保存期間が決められている性質のログもあるため、確認が必要です。

この設定を怠ると、ログがあふれてディスク容量を圧迫し、サービスが継続できなくなる危険性があります（こういったことは現によくあります）。

ログフォーマット

ログのフォーマットは、まず監視システムや

▼表3　Yahoo!ショッピングでのレベル別対応表

ログレベル	対応	出力基準例
info	不要	バッチの正常終了報告、解析に必要な情報を出すログ
notice	不要	正常とは違うルートで終了したログ
warn	営業時間内	一定の間隔でn回以上発生した時点で対応が必要になるログ
err	営業時間内	1回発生した時点で対応が必要になるログ
crit	即時	一定の間隔でn回以上発生した時点で対応が必要になるログ。サービス継続に影響が出る場合に使用
alert	即時	1回発生した時点で対応が必要になるログ。サービス継続に影響が出る場合に使用

ログ収集システムの仕様に合わせて設計するのが基本です。しかし、自由入力となるメッセージフィールドの仕様は、ある程度決めておいたほうが良いでしょう。その際は“5W1H”を意識しましょう。

サーバ配置によるログ出力制限

Yahoo! JAPANでは、ログを出力するサーバがどのネットワークに属しているかで、出力できるログのレベルが違ってきます。基本的にすべてのサーバは外部ネットワークにポートを開放しておらず、ファイアウォールの内側にいます。外部にサービスを提供するサーバのみ、外部ネットワークに一部のポートを開放しています。

外部ネットワークにポートを開放しているサーバは、開放していないサーバと比べ、侵入やアタックに対するセキュリティリスクは高くなります。そのようなサーバでは、サーバ内に出力できるログは制限されています（**図1**）。ここまでやっている企業もあまりないかもしれませんが、大規模なサーバとネットワークを構築しているYahoo! JAPANでは、セキュリティ強化のためこのようなルールがあります。

運用担当者との認識合わせ

ログの仕様について、運用担当者と認識合わせをしましょう。これは地味ですが、結構大事な作業です。実際に認識合わせをしてみると、「このタイミングでこういうログを出してほしい」など、運用の立場からの要望が出てきます。

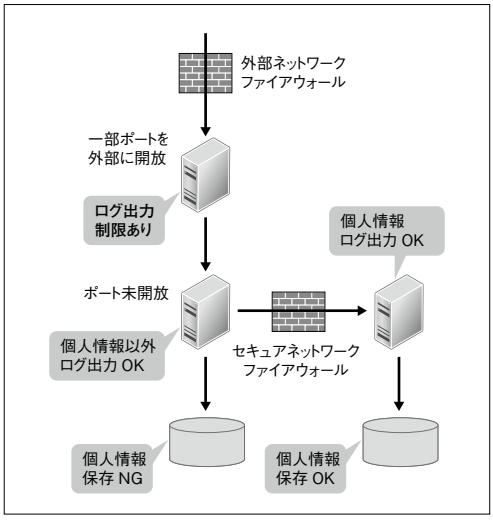
運用フェーズに入ったあとに、「これはどういう意味のログですか？」という問い合わせが運用担当者から来ることもよくあります。これは対応の時間ロスになりますので、運用担当者とのコミュニケーションは事前に行っておきましょう。

ログレベルはサービスのSLAに直結する

ログレベルの定義は非常に重要です。レベルに応じて「今すぐに対応するべきか」「明日、出社してから対応するべきか」が決まるからです（**表3**）。つまり、ログレベルの定義は、そのサービスのSLA（Service Level Agreement、サービス品質保証）を大きく左右します。

一口にログレベルと言っても、使用する言語

▼図1　サーバ配置によるログ出力制限



やライブラリによって微妙に違いがあります（**表4**）ので、個人個人で認識のズレがないように、ログレベルの対応表は作っておいたほうが良いでしょう。

一歩先のことを考えたログ設計

ログ出力がボトルネックに?

非常に大量のトランザクションを処理する場合、大きめのログを一度に大量に出力すると、ログファイルへの書き込み処理がボトルネックとなり、それが積もりに積もって全体の処理レイテンシに影響が出てしまうケースがあります。

その場合、Fluentdなどログ転送技術を活用すると、ログ出力処理自体を本体の処理から切り離して非同期にすることができ、かつ目的に応じて効率よくログを振り分けることができます（**図2**）。

JavaScriptでのエラー収集

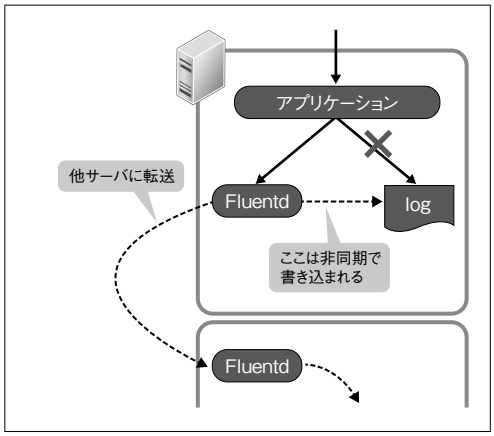
クライアントサイドでのJavaScriptエラーは、サーバサイドでは検知ができません。しかし、JavaScriptのエラーを検知することは、UX（User experience）改善には重要な手がかりとなります。たとえば、フォームのバリデーションエラーのログは、チェックの方法や入力方式を改善させるヒントとなります。

Yahoo! ショッピングでは、一部のツールにJavaScriptエラーをサーバに送信するしくみ（**図3**）を導入し、ログの解析をしています。

JavaScriptのエラーメッセージをサーバに送信する際に重要なのは、なるべく多くのユーザ環境の情報（OSバージョン、User-Agent、デバイス）を集めることです。

現在は、ブラウザの種類、バージョン、デバイス、OSなど、環境が多過ぎてすべてのパターンをテストすることはほぼ不可能に近いです。

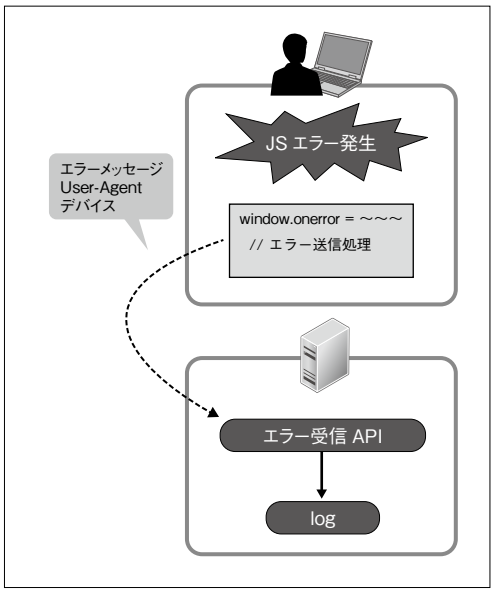
▼図2　Fluentdを使ったのログ出力処理非同期化



▼表4　おもな言語、ライブラリでのログレベル

言語／ライブラリ	syslog	Log4j	PHP	Ruby
ログレベル		TRACE		
	debug	DEBUG	DEBUG	DEBUG
	info	INFO	INFO	INFO
	notice			
	warn	WARN	WARN	WARN
	err	ERROR	ERROR	ERROR
	crit	FATAL	FATAL	FATAL
	alert			UNKNOWN
	emerg			

▼図3　JavaScriptエラーログ収集のしくみ



▼図4　SmartBeat

このしくみを使って、テストではカバーできないユーザ固有環境でのエラーをいち早くキャッチし、改善につなげることができます。

アプリケーションでのログ収集

アプリケーションでログを収集する方法は、すでにたくさんのサービスが世に出ています。Yahoo!ショッピングのアプリケーションでは、ユーザの行動ログは社内ツールやAdobe Analyticsを組み合わせて利用し、エラーやクラッシュログは子会社のツールを使っています。

アプリケーションにおいてはとくにクラッシュログは重要で、当社子会社であるFROSK㈱のSmartBeat（**図4**）を利用して収集しています。このツールではエラーの内容から端末情報、影響のあったユーザ数までかなり詳細に見ることができてたいへん便利なツールです。

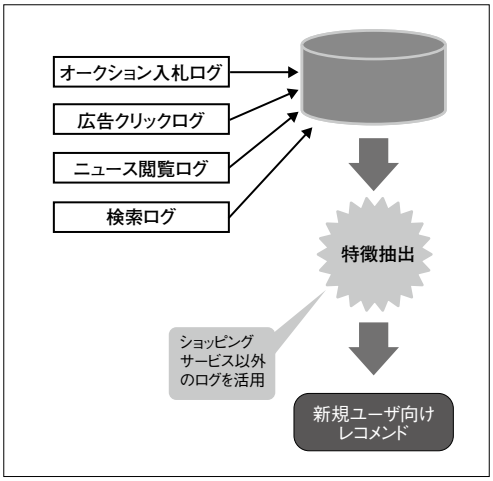
ログの活用事例

ここからはYahoo!ショッピングでのログの活用事例をいくつか紹介します。

新規ユーザ向けレコメンド

Yahoo! JAPANは多くのサービスを提供しているため、Yahoo!ショッピングをまだ利用したことのないユーザもたくさんいます。

▼図5　他サービスのログを有効活用



Yahoo!ショッピングを利用したことのあるユーザであれば、閲覧ログや注文ログから別の商品をお勧め（レコメンド）することができますが、新規ユーザではそうはいきません。その新規ユーザ向けにどう商品をお勧めするか。答えは他サービスにあります。Yahoo! JAPANが提供しているほかのサービスのログを活用することで、新規ユーザにも精度の高いお勧めを実現できます（**図5**）。

エラー状況の可視化

次に、エラー状況の可視化です。Yahoo!ショッピングの一部機能では、Fluentd＋Elasticsearch＋Kibanaでエラーを可視化します。定番の形ですね。

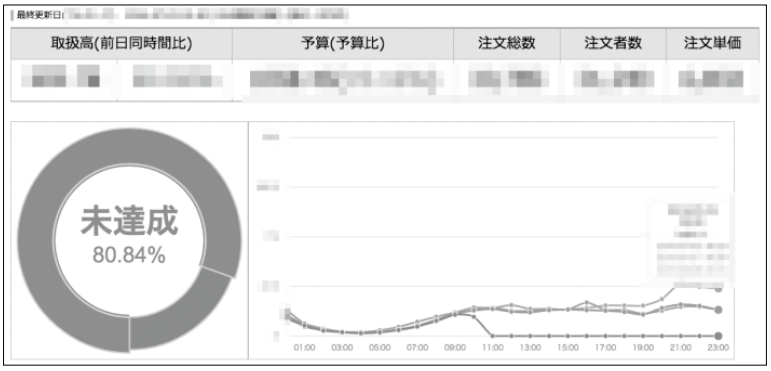
例としては、ショッピングのお問い合わせフォームでエラーとなった数を可視化し、さらにエラー番号単位で集計をかけ、日々の改善に役立てています（**図6、7**）。

サービスの健康状態監視

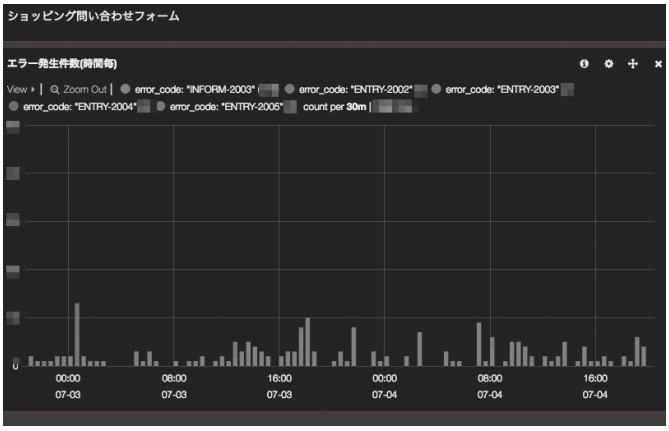
ログはシステムの状態監視だけでなく、“サービスの健康状態”監視にも使えます。Yahoo!ショッピングでの“健康状態”とは、ズバリ「取扱高（Yahoo!ショッピング全体の注文金額の総合計）」です。

この取扱高をリアルタイムで監視できるツールを自前で作成しています（**図8**）。このツールでは日々の目標取扱高が設定されており、その目標に対して現在どのような状態なのかをリアルタイムで閲覧できます。また、前日・前々日・前週と比較してグラフで可視化することで、現在の取扱高が目標どおりに推移しているかを1

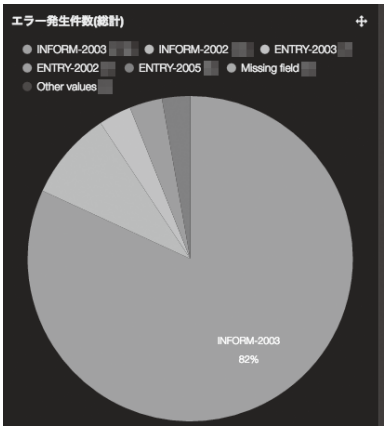
▼図8　取扱高チェックツール



▼図6　Fluentd＋Elasticsearch＋Kibanaを使用したエラー状況の可視化（1）



▼図7　Fluentd＋Elasticsearch＋Kibanaを使用したエラー状況の可視化（2）



時間単位で確認することができます。さらに、1時間ごとにアラートが設定され、前日の同時間比で異常値が出ていないか（取扱高が大幅にずれていないか）をメールで速報する機能もあります。

たとえば、ある商品がテレビで紹介された場合、取扱高は想定より高く推移をします。逆に想定よりも低く推移していた場合、システムのどこかに障害が発生している可能性があります。そういった変化をいち早くキャッチするために、システムだけでなく、“サービスの健康状態”を監視する必要があるのです。

ユーザへのフィードバック

ログを集計した結果をユーザにフィードバックしているところもあります。たとえば、ショッピングカート画面では、カートへの商品の出し入れログを集計し、何人がカートに入れているかを表示しています（**図9**）。

また、受注した注文ログをリアルタイムで配信し、今売れたものを見せるモジュールにも使っています（**図10**）。

ログをマーケティングに活かすには

一般的には、ログを扱うエンジニアと、マーケティング担当者は分かれているケースが多く、かつマーケティング担当者は非エンジニアであることがほとんどです。

非エンジニアに対しては、ログそのものを見せるよりも、やはり可視化してビジュアルで見せるほうがイメージしやすく、発想が広がります。「ログをどう集めるか」ももちろん大事ですが、「**ログをどう見せるか**」もセットで考えることが必要です。そのためには、エンジニアとマーケティング担当者が密に連携することが重要です。

最近ではKibanaをはじめ、多くのログ可視化ツールが出ていますので、まずはフリーのツールから導入してみましょう。

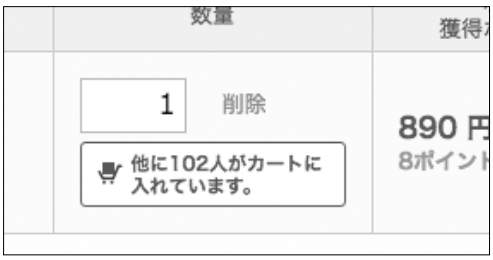
まとめ

一口にログと言っても、非常に奥が深く、設定が面倒くさいと感じるところもあるかもしれません。しかし、数字を伸ばしている企業は間違いなくログを有効活用していますし、運用レベルが高いサービスは質の高いログを出力し、かつ、わかりやすく可視化しているはずです。

ログは単なる文字列ではなく「宝の山」です。みなさんもログを有効活用していきましょう。

SD

▼図9　同じ商品を何人の人がカートに入れているかをユーザに示す



▼図10　今売れた商品をユーザに示す

第2特集

使いこなせていますか？
良いPHP、悪いPHP
――すぐ効くWeb開発入門

RubyやScalaなどが注目されるなか、PHPはその安定性と開発の容易さで、Web開発の分野では今も高いシェアを誇っています。本特集は、PHPを使ううえでのメリット・デメリットを明確にしながら、初心者がWeb開発を行えるようになるためのヒントを提供します。

導入方法と基本文法を押さえたら（第1章）、ライブラリを使ってより高度な機能を簡単に実装しましょう（第2章）。アプリケーションの規模が大きくなりそうならフレームワークを導入して、開発を体系化・効率化しましょう（第3章）。そして最後の第4章では、参加するとPHPの学習がますます捗る、ユーザコミュニティを一挙に紹介します。

Author 濱田 侑弥（はまだ ゆうや）
Twitter @youkidearitai
URL http://tekitoh-memdhoi.info

第1章 基本、押さえていますか？ PHPのはじめ方と学び方 ――環境構築からコーディングまで

PHPは、Web開発の定番プログラミング言語。本章ではプログラミング初心者を対象に、XAMPPを使った環境の構築、文法の基本・データ型とそのハマりどころを解説したあと、「悪いPHP」の例として、Webフレームワークを使わずにCRUDアプリを作ってみます。実行結果をブラウザで確かめながら読み進めましょう。

PHPとは

PHPは、もっとも使われているWebプログラミング言語と言っても過言ではありません。PHP 7もリリースされ、さらに長く使われていく言語となるでしょう。

PHPは、最初Webの便利スクリプトとして使われていたものが、PostgreSQLやMySQLといったリレーショナルデータベースとの親和性の高さから少しずつ実績を積んで信頼を勝ち取っていき、今の地位にまで上り詰めたのだと思います。また日本では、いわゆる「ガラケー」で使われる絵文字や、Shift_JISやEUC-JPなど、早くから「全角文字（マルチバイト）」に対する処理がたくさんあったことも、PHPを大きく進化させた要素の1つではないでしょうか。

本稿では、PHPを通じてプログラミングを学んでいく手がかりを解説します。XAMPP（Windows or Linux、Apache、MySQL、PHP）環境の構築から、言語のおさらい、Webアプリケーション開発まで一通りやっていきましょう。

PHPの環境構築について

「開発環境」について

Webアプリケーションを作成する際に重要になってくるのは、「『本番環境』と『開発環境』を区別しましょう」ということです。開発環境を用意することで、状況を把握するための「トライアンドエラー」を行うことができますから、システムや事業の全体が把握しやすくなります。

技術者にとって、「トライアンドエラー」は非常に重要です。動かしてダメだった部分がなぜダメだったのか、試行錯誤を重ねていくことでエンジニアとして成長ができます。緻密な計画を立ててその計画を忠実に実行するやり方も否定はしませんが、Web系においては“とりあえずやってみる”ということが重要である、と個人的には思います。

本番環境しかない場合、「バグが発生したとき直接手を入れてしまい、サービスがエラーで動かなくなった」「飛んではいけない個人情報が飛んでしまった」などのインシデントが発生しかねません。開発環境においても個人情報などの貴重なデータが入っている場合は、本番環境での運用を想定して、慎重になってください。もしも、慎重になるべき場所がわからなかったら、納得いくまで先輩・上司に聞いてください。

PHPのはじめ方

PHPの環境を整えたい場合には、いくつかの方法が存在します。お手軽なのが、XAMPPと呼ばれるPHPに必要なプログラム一式がすべてそろっているパッケージを用意することです。しかしこの方法には弱点があり、Windowsで動くPHPバイナリはLinux/UNIXのPHPと比べて挙動が多少異なっていることがあるのです。

XAMPPが開発環境としてきっちり動いたことを確認できた、よし本番でLinuxサーバをデプロイ(反映)しようとしたら、あれ、挙動が違うぞと困ってしまうことにつながります。

入門の範囲が外れてしまうため詳しくは触れませんが、もしもVagrantを使うなどして仮想Linux環境を構築できるのであれば、それがベターです。こちらは、本番環境と大きく挙動が違うことはないでしょう。

もしもできるならば、余ったパソコンにLinuxを入れてLANで自宅サーバを組んでみるのも1つの手かもしれません。やはり実物があると人間理解がしやすいです。

開発環境の用意

LinuxにPHPをインストールするには、次のようにディストリビューションごとに提供されているパッケージを使用する方法があります。

```
#Debian
$ sudo apt-get install php5
#Centos
$ sudo yum install php
```

Windowsの場合には、XAMPPをインストールするのが手軽です[注1]。本稿では、初心者のためにXAMPP(PHP 5.6.23)で動かすことを前提に進めていきます。Linuxが動かせるという方は適宜読み替えてください。Windowsの場合、ほかのアプリケーションが80番ポートを使用しているということも少なからずあるので、Apacheを8000番などに変更するのが良いでしょう。展開先のC:¥xampp¥apache¥conf¥httpd.conf の「Listen 80」を「Listen 8000」に変更してください。

注1） URL https://www.apachefriends.org/jp/download.html

▼図1　phpinfo(一部抜粋)

System	Windows NT TEKIMEN-PC 6.2 build 9200 (Windows 8 Home Premium Edition) i586
Build Date	Jun 22 2016 12:08:00
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=c:¥php-sdk¥oracle¥x86¥instantclient_12_1¥sdk,shared" "--with-oci8-12c=c:¥php-sdk¥oracle¥x86¥instantclient_12_1¥sdk,shared" "--enable-object-out-dir=./obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:¥WINDOWS
Loaded Configuration File	C:¥xampp¥php¥php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226,TS,VC11
PHP Extension Build	API20131226,TS,VC11
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring

PHPの動かし方

phpinfo

PHPはWebプログラミング言語として扱われるのが一般的です。ここからは、HTMLが書けることを前提として話を進めていこうと思います。ApacheのDocumentRootに次のような内容のindex.phpを置いてみましょう。お好きなテキストエディタやIDEを開いて書いてみてください。XAMPPであれば、C:¥xampp¥htdocsがDocumentRootに設定されています。

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

これでXAMPP Control Panel[注2]でApacheを起動させ、ブラウザからhttp://localhost:8000/index.phpにアクセスすれば、phpinfoが表示されるはずです(**図1**)。phpinfoとは、インストールされているPHPのバージョン、設定、環境変数などの情報を一通り眺めることのできる関数です。サーバの重要な情報を表示する関数ですから、外部に公開してはいけません。

注2） 展開したXAMPPパッケージのxampp-control.exeから起動できるGUIツール。

▼リスト1　PHPのHello World（helloworld.php）

```
<!DOCTYPE html>
<html lang="ja">
<head>
</head>
<body>
<?php
echo 'Hello World';
?>
</body>
</html>
```

Hello World

お馴染みHello Worldを書くのは**リスト1**のような感じです。これを先ほどと同じようにhelloworld.phpと保存して、http://localhost:8000/helloworld.phpで実行すればHello Worldが出力されます。

入力の受付

リスト2は、ブラウザからアクセスするとき、http://localhost:8000/input_html.php?hoge=hugaとすれば、<?php ?>で囲まれている部分がhugaと表示されます。また、「huga」を好きな文字に入れ替えると、いろいろな文字が表示されるようになったことでしょう。

<?php ?>というタグ

PHPでは、開始タグである<?phpと終了タグ?>の中にプログラムを書いていきます。ほかのプログラミング言語と違ってオリジナルのタグがあるのです。ユニークですね。HTMLを出力するテンプレートにもPHPのプログラムを書けるところがすごくいいですよね。たとえば、

```
<h1><?php echo "Hello World"; ?></h1>
```

をhelloworld_html.phpとしてXAMPP Control Panelのshellで実行すると、次のように出力されます。

```
# php helloworld_html.php
<h1>Hello World</h1>
```

タグの外側はそのまま出力されていることがわかります。また、プログラムだけの場合、終了タグ?>は、プログラムの中に含めないほうが良いとされています。?>のあとに余計な文字列（空白や改行など）があるとそれが出力されてしまい、エラーの原因を見つけるのがたいへんだからです。

データ型

計算させてみる

さて、Hello Worldのほかにも動きを加えたいですね。そこで、簡単な計算をやらせましょう。PHPは計算させることもできます。

```
<?php
echo 1 + 3;
```

これをone_plus_three.phpとして保存し、実行すると、次のようになります（以降、とくに説明のないかぎりXAMPP Control Panelのshellでの実行結果です）。

```
# php one_plus_three.php
4
#
```

▼リスト2　PHPの入力受付（input_html.php）

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
</head>
<body>
<p><?php echo isset($_GET['hoge']) ? htmlspecialchars($_GET['hoge'], ENT_QUOTES, 'utf-8') : ""; ?></p>
</body>
</html>
```

▼リスト3　PHPのif文

```
<?php
$value = '';
$expected = 0;
if ($value == $expected) {
    $output = "$value == $expected is true";
} else {
    $output = "$value == $expected is false";
}
echo $output . PHP_EOL;
```

改行を加えるなら次のように書きます。

```
<?php
echo 1 + 3 . PHP_EOL;
```

これを実行すると、次のようになります。

```
# php one_plus_three.php
4

#
```

PHPのすごいところに、「数値で計算したあと、文字列としてくっつけることができる」ということがあります。なぜなのかはあとで説明しましょう。

PHPには変数もあります。

```
<?php
$value = 1 + 3;
$value = $value + 8;
echo $value . PHP_EOL;
```

出力は12となります。もちろん、PHPでは、条件分岐もできます。if文ですね。**リスト3**を実行すると、次のようになります。

```
# php value_expected.php
 == 0 is true

#
```

もうちょっと見てみましょうか。**リスト4**を実行すると、次のようになります。

```
# php helloworld_value.php
Hello World == 0 is true

#
```

さて、Hello World（文字列） == 0（数値）が

▼リスト4　リスト3を少し変更

```
<?php
$value = 'Hello World';
$expected = 0;
if ($value == $expected) {
    $output = "$value == $expected is true";
} else {
    $output = "$value == $expected is false";
}
echo $output . PHP_EOL;
```

trueとなるのはなぜでしょうか。この理由は「暗黙の型変換」と呼ばれるPHPの機能にあります。==や!=で比較をしようとするときや、echoやprintで出力するときに自動的に型変換してくれます。先ほどのone_plus_three.phpのような「文字列や数値を連結させる」ことも、この機能によってなされています。

このようにさまざまな場面で、暗黙の型変換が“よしなに”やってくれます。使いこなせれば便利です……が、プログラマの意図しない動作も引き起こしやすいのです。

PHPではとくに、データ型ではまりやすい罠が多いため、ここでは基本的に使うデータ型を紹介していきます。まずはスカラー型（大小の比較が可能な型）のデータ型です

論理型（boolean）

値が「真」であればTRUE、「偽」であればFALSEです。論理型はこの2つのみです。

整数型（integer）

値が「整数」である型です。10進数で指定できるほかに、8進数、16進数、2進数で表現できます。また、+と-の符号を付けることもできます。最大値はC言語のlong型に依存しているので、32bitで$2^{31}-1$、64bitで$2^{63}-1$となります。最小値はそれぞれ-2^{31}、-2^{63}です。WindowsではPHP 7以外では$2^{31}-1$が最大値、最小値は-2^{31}になります。

ご自身の環境で最大値を検証してみたい場合には、次のようにPHP_INT_MAXを見てみてください。

```
<?php
var_dump(PHP_INT_MAX);
var_dump(PHP_INT_MIN); // PHP 7以降のみ
```

浮動小数点数（float or double）

値が「浮動小数点数」である型です。実装は倍精度浮動小数点です。整数型の範囲外の値を指定した場合でもこの型になります。浮動小数点数ですので、丸め誤差が発生します。PHPに限ったことではありませんが、厳密な比較に用いるのはお勧めしません。ある程度の桁を許容して比較することが一般的です。

文字列型（string）

一文字8bitの集合体、文字の集まりである型です。バイナリを扱うこともできます。つまり、画像などのデータを読み込んで処理することもできます。マルチバイト文字列を扱う場合には、mbstringというモジュールを使用します。

画像のサイズを測りたいときにはstrlen関数が使えます。

```
<?php
$image = file_get_contents('image.png');
var_dump(strlen($image));
```

マルチバイトの文字列の長さを測りたいときにはmb_strlen関数を使います。

```
<?php
mb_internal_encoding('UTF-8');
$mbstr = 'こんにちは、世界！';
var_dump(mb_strlen($mbstr, 'UTF-8'));
```

文字列型を扱う関数のマニュアルを読むと、「この関数はバイナリデータに対応しています」とある場合があります。これは、PHPを作っているC言語の仕様として、文字列の末端には「ヌル文字」が入っていることと関係しています。もし、PHPにヌル文字が入ってきても、PHPではヌル文字そのものとして扱いますが、C言語では文字列の末端と認識します。バイナリデータに対応していない関数では、ヌル文字で文字の処理をやめてしまいます。それの何が問題なのか。ヌル文字以降をチェックしなくなるわけですからチェックをすり抜けたり、想定しない挙動を引き起こすことになります。なるべく「この関数はバイナリデータに対応しています」という関数を使ってください。

非スカラー型

ここからは、これまでに紹介したスカラー型を複数扱うための便利な複合型を紹介します。

配列（array）

PHPにも配列があります。

```
<?php
$array = array(
  'hoge',
  'huga',
  'bar',
);
foreach($array as $value) {
    echo $value . PHP_EOL;
}
```

配列と言いましたが、実際には「順番付けられたリスト」です。キーを指定することもできますから、いったんコードを**リスト5**のように書き換えてみましょう。これを実行した結果は次のようになります。

```
1 => hoge
0 => huga
2 => bar
```

配列ではないため、さまざまな値をキーにすることができます（**リスト6**）。数値にできるものは暗黙の型変換で整数に変換されます。実行してみると、**図2**のようになります。

▼リスト5　キーを指定した配列

```
<?php
$array = array(
  1 => 'hoge',
  0 => 'huga',
  2 => 'bar',
);
foreach($array as $key => $value) {
    echo $key . ' => ' . $value . PHP_EOL;
}
```

オブジェクト(object)

クラスからnew命令を使って作成したインスタンスを格納できます。このときのデータ型をオブジェクト型と言います……普段こういう言い方をしないのでなんだか不自然ですね。

```
<?php
$obj = new stdClass();
$obj->a = 1;
$obj->b = 'abc';

var_dump($obj);
```

これを実行すると、次のようになります。

```
$ php obj.php
object(stdClass)#1 (2) {
  ["a"]=>
  int(1)
  ["b"]=>
  string(3) "abc"
}
```

初心者向けを大きく逸脱してしまうのでここでは最低限、「newを使ったら変数にオブジェクトが入る」くらいの認識でいてください。オブジェクト指向を使わずともプログラムは書けます。もし、オブジェクト指向での開発が必要なプロジェクトの規模になったら、必要に応じて学習してください。

なぜデータ型の話をしたのか

PHPはデータ型を強く意識することのない、動的型付けと呼ばれるプログラミング言語に分類されます。もともと、そんなに意識することのないはずのものを、どうして誌面を割いて紹介したのか。これには理由があります。

PHPのマニュアルのビルトイン関数のページ[注3]に**図3**のような説明があります。

これはいったいどういうことなのでしょうか。例を使って説明してみましょう。

```
<?php
$str = array();
file_get_contents($str);
```

▼リスト6　いろいろな値をキーに

```
<?php
$array = array(
  'hoge' => 'hoge',
  'huga' => 'huga',
  'foo' => 'bar',
  '3' => 8,
  '-4' => 'x',
  '3.8' => 9,
);
var_dump($array);
```

▼図2　リスト6を実行

```
$ php array.php
array(6) {
  ["hoge"]=>
  string(4) "hoge"
  ["huga"]=>
  string(4) "huga"
  ["foo"]=>
  string(3) "bar"
  [3]=>
  int(8)
  [-4]=>
  string(1) "x"
  ["3.8"]=>
  int(9)
}
```

▼図3　ビルトイン関数についてのマニュアル

> *内部(ビルトイン)関数*
> *注意：関数へのパラメータとして関数が想定しているのとは異なるものを渡した場合、例えば文字列を想定しているところに配列を渡した場合などの場合は関数の返り値は未定義となります。たいていの場合はNULLを返すでしょう。しかしこれはあくまでも規約にすぎず、これに依存することはできません。*

$strのパラメータについて、本来文字列にするべきところを配列にした場合、file_get_contents関数はE_WARNING(警告)を鳴らしながらNULLを返してきます。しかし、これはおかしいです。file_get_contents関数はファイルの読み込みに失敗したらFALSEが返ってくるはずです[注4]。でも、実際にはNULLが返ってくるのです。では、もっと悪いケースを見てみましょう。

```
<?php
$str = array();

if (@file_get_contents($str) !== FALSE) {
    echo "ファイルが読み込めました！";
}
```

返り値のデータ型の厳格なチェックを行ったにもかかわらず、なぜか「ファイルが読み込めま

注3）URL http://php.net/manual/ja/functions.internal.php

注4）URL http://php.net/file_get_contents

した！」という画面が出力されるでしょう。また、@というエラー抑制指定演算子を用いてE_WARNINGを取り除いているので、どこにバグがあるのかもはやわからないですね。

関数の引数には、きちんとマニュアルどおりの型を指定しましょう。データ型を保証できないときはきちんと型チェックを行いましょう（**リスト7**）。

また、これはPHPに限ったことではありませんが、エラーを無視してはいけません。間に合わせで作ったらあとあとにツケがやってきます。そうならないためにも、エラーが発生したらログを取り、できればユニットテストを行いましょう。

CRUDができるアプリケーションを書こう

データベースを用意する

SQLiteというデータベースを使ってみましょう。ここでは、PDO（PHP Data Objects）と呼ばれる、データベースの接続を共通化（抽象化）した拡張モジュールを使用します。PDOにはプリペアドステートメントが装備されており、確実にデータベースからほしい情報をアクセスできます。

ただし、現状では直接PDOを使うことはあまりありません。フレームワークなどにあるO/Rマッパーを使ったほうが便利なケースが多いためです。今回は初心者向けということでせっかくですし使ってみましょう。

ひとまず、SQLiteを使ってみましょう。XAMPP Control Panelのshellから次のコマンドを実行します。

```
# mkdir sqlite
# sqlite3 sqlite/db.sqlite
```

そうすると、SQLiteの操作画面になるので、次のSQLをたたいてみましょう。

```
CREATE TABLE crud_sample (
    id INTEGER PRIMARY KEY , text text NOT NULL
);
```

これでcrud_sampleというテーブルができあがりました。

サンプル作成

さて、これまでの知識を使ってWebブラウザを介したCRUD（Create、Read、Update、Delete）のサンプルを作成してみましょう。サンプルコードを本誌Webサポートページ[注5]に置きました。ただし約束してほしいのは、これは自分のパソコン、ローカル環境上で動かすということです。先に言っておきます。これは「悪いPHP」です。筆者の、Webエンジニアとしての人権を失う覚悟で書きました。理由は後述します。

セキュリティ

このサンプルコードでは、**図4**のようにcurlを使ってリクエストを投げたのちに、ブラウザに戻ると不思議なことに「hoge from curl」という項目が追加されています（**図5**）。

Webアプリケーションはブラウザからのリクエストのみならず、このようにHTTPでの通信ができるのならば、どんなクライアントでも通

▼リスト7　マニュアルどおりの型を渡す。型チェックも行う

```
<?php
$str = "hoge.png";

if (!is_string($str)) {
    exit("不正な引数です");
}

if (file_get_contents($str) !== FALSE) {
    echo "ファイルが読み込めました！";
}
```

▼図4　項目を追加するリクエストを投げる

```
curl -X POST http://localhost:8000/crud_sample.php?mode=add -F "text=hoge from curl"
```

注5）URL http://gihyo.jp/magazine/SD/archive/2016/201609/support

信ができるようになっています。

セキュリティに関しては、IPAの記事「安全なウェブサイトの作り方」[注6]や、書籍『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7]を参照してみてください。

これは悪いPHPです

このサンプルコードを使えば、基本的なCRUD操作は可能になりますが、ログイン機能やセキュリティに関してはほとんど何も考えていません。また「MVC」[注8]のように、テンプレートとプログラムの分離がなされていません。

そもそも、LaravelやCakePHPなどのフレームワークを使ってWebアプリケーションを作るべきです。それでも、フレームワークを使わずにこのアプリケーションを作ったのは、ひとまずは基本的なしくみを理解してほしいと思ったからです。

新入社員のみなさんへ

すでにみなさんは、研修が終わって、プログラミングを業務で本格的に行っている段階に入っているのかもしれません。そうしたときに必要なのは、技術力よりも“他人に頼ること”だと思っています。これは、筆者が他人に頼ることが下手で、1人で抱え込んでたくさんの失敗をしてしまったという経験から得た考えです。

新入社員という立場で、プログラミングについても、もちろん業務についてもわからないことだらけだと思います。ときにはルールにがんじがらめになってうまく動けないといったこともあると思います。とくに、Webアプリケーションの開発というのはさまざまな分野に渡り、それぞれが非常に専門的になっています。それを1人でカバーするのはほぼ不可能なのではないか、と考えると、先輩や上司をなんとか捕まえて頼ること、迷惑をかけてしまうことが必要ではないかと考えるのです。

▼図5　ブラウザでサンプルコードを表示

id	テキスト	編集	削除
5	hoge from curl	編集	削除
4	ABC(*'ω' *)	編集	削除
3	aaaaaホゲ	編集	削除
1	テストABC	編集	削除

追加

もしもそうした、助けてくれる人がいない、という状況でしたらPHP勉強会（**第4章参照**）に顔を出してみてください。え、早く帰れない？「勉強会のために早退します」とか、奥の手として「体調悪いので早退します」とか言って、なんとか時間を作っちゃいましょう。新入社員が1日くらい早退したところでダメになることはありませんよ。

新入社員を教える先輩・上司のみなさんへ

とくに、「初めて部下を持つ」先輩や上司の方にお願いしたいのは、わからないことだらけで戸惑っている新入社員の方をフォローしてあげてほしい、ということです。

やはり、Webの技術1つとってもどれもが高度になってきている昨今、PHPのシステムを動かすとなったらセキュリティ、データベース、文字エンコーディング、フロントエンド、クラウドやVPSなどのホスティングサービス……と、1人でカバーできるほどのものではなくなってきていると思います。

とは言っても、現実には数字（KPI）などの目標と、後輩が思いどおりに動かないことの板挟みになってしまうことがあるかもしれません。しかし、そこで後輩の成長にかけてみてほしいのです。SD

注6） URL https://www.ipa.go.jp/security/vuln/websecurity.html
注7） 徳丸 浩, SBクリエイティブ, 2011,ISBN＝978-4-7973-6119-3
注8） Model View Controllerの頭文字をとったデザインパターン。

Author 後藤 知宏（ごとう ともひろ）
Mail t.goto@chatbox-inc.com
Twitter @mkkn_info
㈱chatbox（http://chatbox-inc.com/）

第2章 効率よく選んでいますか？ PHPのライブラリの選び方・使い方 ――Composerをお勧めする理由

PHPによるWebアプリケーション開発で、ライブラリの存在は欠かすことができません。Webアプリケーション開発歴史の成果がPHPのライブラリ群と言えますので、実にさまざまなライブラリが公開され使用されています。それらを導入するにあたり管理ツールとしてPearが用いられていましたが、現在ではComposerが使用されています。本稿ではComposerの導入から使用方法までいっきに解説します。

ライブラリを導入する理由

プロジェクト開発を進めるにあたり、すべての機能を自分でゼロから開発していくのは非常に骨の折れる作業です。プログラミングの世界では、特定の機能を持ったひとまとまりのコード群をライブラリという形でまとめることがあり、さまざまな機能のライブラリがGitHubなどのサービスを通じてWeb上で公開されています。

無償で利用可能なオープンソースライブラリは、数多くのプロジェクトで利用されるとともに、動作検証として数多くのフィードバックがライブラリ開発者のもとに集められます。多数の動作検証を経たライブラリは、品質も信頼性も非常に高く、プロジェクト開発に大きなメリットをもたらしてくれます。よく言われる「車輪の再発明」といった無駄を減らすこともメリットの1つですし、それに加えて品質の向上にも効果があると言えます。

PHPのライブラリでも同様です。多数のユーザに利用され、動作検証などを経たライブラリは、Webアプリケーションの開発においても大きな導入上のメリットがあります。

しかし、ライブラリのような「いわゆる他人の書いたコード」を無作為にプロジェクト内に取り込み続けると、運用上での問題が生じて来るでしょう。ライブラリ内部にはバグが含まれているケースなどもあるため、更新の対応に備えておく必要があります。ライブラリ内部の動作を気軽に検証してみたり、必要に応じた更新作業を行うためにも、ライブラリとして外部から取り込んだコードは、プロジェクト固有のコードと明確に区別して管理されるべきです。

Composerによるライブラリの管理

多くのプログラミング言語では、ライブラリを管理するためのツール、いわゆる依存管理ツールが提供されています。RubyにBundler、Node.jsにnpmといったツールがあるようにPHPにはComposerと呼ばれるツールがあります。

Composerを通じてライブラリを管理することで、ライブラリの利便性を損なうことなく、柔軟にライブラリの管理を行うことができるようになります。

実際のComposerの利用の流れをふまえながら、ライブラリ管理の重要性を確認していきましょう。

Composerの導入

PHPでライブラリを使用する場合、最近ではComposerと呼ばれるツールを用いたライブラリ運用を採るのが一般的です。Composerはコ

▼図1　Composer公式サイト（英文）https://getcomposer.org/

マンドツールとして提供されており、コマンドプロンプトやターミナル上から、ライブラリのインストールなどの管理作業を行うツールです。そのため、まずは開発環境でComposerコマンドを使用するための、セットアップが必要になります。

Composerの導入手順は、公式サイトにて詳しい手順が掲載されています。公式サイトから「Getting Started」のリンクをクリックするとインストールの方法を確認できます（**図1**）。WindowsとMac OS/Linuxで方法が異なりますので、環境に合わせた方法を選択してください。

Composerの導入：Windows編

Windows環境に向けてはexe形式のインストーラが提供されています。**図1**よりダウンロードを行い展開すると、composerコマンドの展開とパスのセットアップもしてくれます。そのままコマンドプロンプトからcomposerコマンドを実行できるようになります。

Composerの導入：Linux/Mac OS編

Linux/Mac OS環境ではコマンドラインからセットアップ用のダウンローダを用いてセットアップできます（**図2**）。

これを実行してダウンロードできるcomposer.pharという名前のファイルがComposerの本体です。ダウンロードができたらcomposer.pharを/usr/bin/localなどパスの通ったフォルダに配置して準備は完了です（**図3**）。

▼図2　composer.pharのダウンロード

```
$ curl -s https://getcomposer.org/installer | php
```

▼図3　composer.pharの移動

```
$ mv composer.phar /usr/local/bin/composer
```

▼図4　Composerのセットアップ

```
$ mkdir sample
$ cd sample
$ composer init
```

▼リスト1　composer.jsonの例

```
{
    "name": "t_goto/sample",
    "authors": [
        {
            "name": "t_goto",
            "email": "t.goto @chatbox-inc.com"
        }
    ]
}
```

ライブラリのダウンロード

composerコマンドが使用可能になったら、サンプルプロジェクトとして、空のディレクトリを作成し、Composerのセットアップを行いましょう（**図4**）。

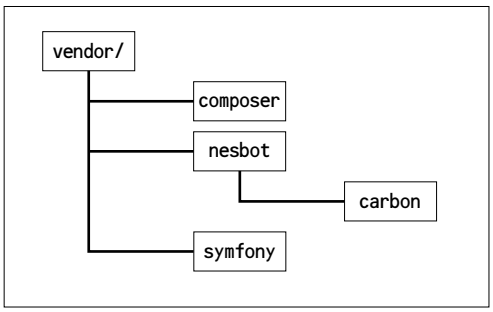
composer initコマンドを実行するといくつかの質問が画面上に表示されます。ライブラリを利用するだけの場合デフォルトの値でも問題ありません。Enterキーを押して質問を進めます。質問が終わるとcomposer.jsonというJSON形式のファイルが作成されます（**リスト1**）。このcomposer.jsonファイルはComposerでライブラリ管理を行うために必要になるファイルで、初期の状態ではプロジェクトや管理者の名前などが含まれています。

リスト1に記載されたプロジェクトや管理者の名前は、デフォルトの設定ではPCの設定が反映されますが、ライブラリをダウンロードするだけ

▼図5　Composerによるダウンロード

```
$ composer require nesbot/carbon
```

▼図6　require後のファイルパス



の利用ケースではこれらが影響するケースはないため、デフォルトのままでかまいません。

準備が整ったところで、実際にライブラリをダウンロードしてみましょう。時刻ライブラリとして一般的に用いられるCarbonをダウンロードするためには、**図5**のコマンドを実行します。

`composer require`コマンドを実行するとvendorディレクトリが作成され、ライブラリのダウンロードが始まります。しばらくするとvendorディレクトリ内に`nesbot/carbon`というディレクトリが作成され、内部にライブラリの本体ファイルが格納されます(**図6**)。

ダウンロードが完了するとcomposer.jsonファイルにダウンロードしたCarbonに関する記述が追記されているのが確認できます(**リスト2**)。

メンバーとの共有

このようにComposerによるライブラリのダウンロードでは、ダウンロードされたライブラリの種類とバージョン番号とが、composer.jsonへ自動で記録されていきます。このcomposer.jsonによるライブラリの一覧管理は、ほかの環境で同じライブラリをダウンロードする際に非常に役立ちます。必要なライブラリの列挙されたcomposer.jsonが共有されている環境では、記述されているライブラリの一括ダウンロードが可能になるのです。

▼図7　ライブラリの一括ダウンロード

```
$ composer install
```

▼リスト2　composer.jsonファイルの中身

```
"require" : {
   "nesbot/carbon" : "^1.21"
}
```

試しにvendorディレクトリを削除し、**図7**のコマンドを叩いてみてください。

削除したはずのvendorディレクトリがふたたび作成されます。その中を確認してみるとCarbonのフォルダも作成されているのがわかるかと思います。

1人の開発者がcomposer.jsonファイルを作成して共有すれば、vendorディレクトリを共有せずとも、ほかの開発者の環境では`install`コマンドを実行するだけでライブラリを導入できるわけです。

Composerを利用したライブラリ管理を行う場合、vendorディレクトリは各開発者が各自の環境でそろえる運用が推奨されています。vendor以下のライブラリ本体は、開発者間で共有せず、代わりにcomposer.jsonファイルを共有し、それぞれの環境でライブラリをダウンロードするのが一般的です。

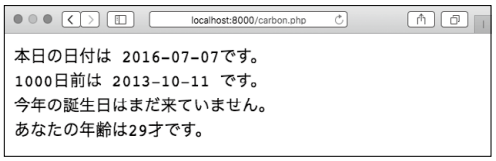
Gitなどのバージョン管理ツールを使用する場合、vendorディレクトリは`.gitignore`ファイルなどを利用してバージョン管理外のファイルに指定してしまうのが良いでしょう。

ライブラリを用いたコーディング

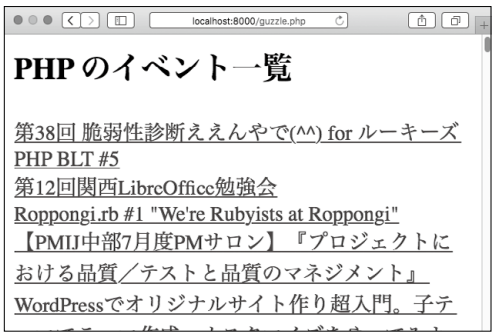
Composerを使ってライブラリの使用準備ができたらライブラリを使ったコーディングに挑戦してみましょう。さまざまなライブラリを利用した処理を確認するために、サンプルアプリケーションを用意しました。次のURL[注1]より

注1）URL http://github.com/chatbox-inc/composer_sample

▼図8　サンプルアプリケーション（public/carbon.phpの実行）



▼図9　HTTPクライアントライブラリ「Guzzle」



ダウンロードし、composer installコマンドを実行してライブラリのセットアップを行ってください。

オートローダのしくみ

PHPで外部ファイルに書かれたプログラムを利用する場合、require命令を用いてファイルを読み込むのが一般的です。

Composerを利用してダウンロードされたライブラリもvendorディレクトリの中にファイルが配置されるため、ライブラリ本体のファイルをrequireすればその機能を利用できるのですが、それではライブラリの数が増えてきた際に、requireが非常に多くなり何かと不便です。

Composerにはオートロードと呼ばれるしくみが用意されており、vendorディレクトリ内のautoload.phpと呼ばれるファイルを読み込むだけでvendor内のすべてのライブラリが個別のrequireなしで利用可能になります。

時刻操作ライブラリ「Carbon」

CarbonはPHP標準のDatetimeクラスを拡張したライブラリで、さまざまな時刻処理を直感的なインターフェースで行えるのが特徴です。

▼リスト3　現在時刻からの減算

```
$now = Carbon::now();
$now->subDay(1000)->format("Y-m-d")
```

▼リスト4　未来判定の関数

```
if($birthDayInThisYear->isFuture()){
    ...
}
```

サンプルアプリケーション内のpublic/carbon.phpをブラウザで確認してみると**図8**のような画面が表示されるはずです。

carbon.phpにおける2行目のrequire文がComposerのオートローダの読み込みです。個別にCarbonのライブラリファイルを読み込むことなく、autoload.phpのファイルを読み込むだけでCarbonライブラリの使用を可能にしています。

Carbonライブラリでは、現在の時刻をCarbon::now()で取得できるほか、生成されたCarbonオブジェクトをベースに時刻関連の演算を行うことができます。

13行目のsubDay関数など生成された時刻を基準とした減算・加算の処理や（**リスト3**）、21行目の現在時刻を基準とした未来判定（**リスト4**）が直感的に行えるのが特徴です。

このほかにも、時刻操作に関する直感的な操作が豊富に用意されているため、時刻を処理する多くの場面でCarbonライブラリが用いられています。より詳しい使い方は、公式ドキュメント[注2]を参照してください。

HTTPクライアントライブラリ「Guzzle」

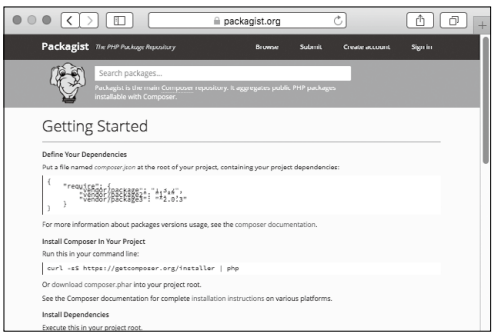
GuzzleはHttpクライアントとしての機能を提供するPHPのライブラリです。サンプルアプリケーション内のpublic/guzzle.phpをブラウザで確認してみると**図9**のような画面が確認できます。

このサンプルアプリケーションでは、IT系のイベント情報サイトConnpassからPHPをキー

注2）Carbon公式ドキュメント（URL http://carbon.nesbot.com/docs/）

▼図10　Monolog

▼図11　Packagist（https://packagist.org/）



ワードにマッチするイベントの情報を一覧で抽出しています。

PHPによるHTTP通信は、file_get_contents関数やcurlによって実装することもできますがGuzzleを用いた記述はより直感的に理解できます。

サンプルファイル6行目からの処理がHTTP通信に関する記述です（**リスト5**）。GuzzleによるHTTP通信はメソッド名、URLによるシンプルな記述で、ステータスコードやボディなどの応答データはすべて戻り値の$resを通じて取得できます。

AWSやTwitter、Slackなどシステム開発で外部サービスと連携する機会は増えており、こうした外部サービスとの連携ではREST APIによるHTTP通信を利用するのが一般的です。

Guzzleを使ったREST API発行のコードは直感的で理解しやすいうえ、asyncによる非同期でのリクエストをサポートするという点も魅力的です。

処理に時間がかかりがちなAPIリクエストなどを非同期に実行することで、全体としてのリクエスト時間を短縮できます。

詳しい使い方などはサンプルコードのほか、公式ドキュメント[注3]を参照してください。

注3）Guzzle、PHP HTTP client（URL http://docs.guzzlephp.org/en/latest/）

▼リスト5　GuzzleによるHTTP通信

```
$client = new GuzzleHttp¥Client();
$res = $client->request('GET',$url);
```

▼リスト6　loglevelを変更してログを調整する

```
$loglevel = Logger::DEBUG;
$path = __DIR__.'/../app.log';
$handler = new StreamHandler($path, $loglevel);
$log->pushHandler($handler);
```

Monolog

MonologはPHPのログ機能を提供するライブラリです。サンプルアプリケーション内のpublic/monolog.phpをブラウザで確認してみると**図10**のような画面が確認できます。

このサンプルアプリケーション内では、app.logへのログ出力を行っています。ブラウザからアプリケーションへアクセスするたびに、app.logファイル内にデータが追記されていくのが確認できます。

Monologで記録されたアプリケーションログは、必要に応じて設定からログレベルに応じたログのON/OFFが行えたり、ファイルやメール、DBといった複数の宛先へのログの配信などを切り替えることができるのが特徴です。

サンプルアプリケーション内の18～20行目がログレベルの設定となります（**リスト6**）。コメントアウトされている19行目の記述をコメント解除するとログレベルが変更され、ファイルに記録されるログの量が変わるのが確認できると思います。

開発環境では多くのログ出力を行いたいが、本番環境では不要なログ出力を絞り、重要なエラーログに絞った監視を行いたい、といったケースは多く見受けられます。

Monologによるログ配信では、ログ記述部分のコードに手を加えることなく、ログ側の設定で柔軟にログ記録のしくみを変更できるため、環境に応じたログ配信仕様の変更にも柔軟に対応できるのが特徴です。このほかの詳しい使い

方はMonologのGitHub[注4]ページを参照ください。

さまざまなライブラリ

PHPの世界では、紹介した以外にもさまざまなライブラリが公開されています。Composer経由で手軽に導入可能なライブラリの一覧はPackagistというサイト上で管理されています（**図11**）。

Packagistはすべての開発者に開かれており、GitHubを通じて誰でも気軽に自分のライブラリを登録できます。最近では多くのライブラリがPackagistに登録されComposer経由で気軽にダウンロードできるようになっていますが、Packagistに登録のないライブラリでもプロジェクトに導入できる、というのもComposerの魅力の1つです。

GitHubやBitbucket上に登録された個人リポジトリのライブラリやPEARのライブラリなどに加え、zip形式のライブラリも設定しだいでComposer経由の導入が可能なため、必要なライブラリがPackagistに登録されていなくても対応可能、という点は古いライブラリを利用するケースなどで非常に重宝する機能です。

ライブラリ運用の落とし穴

Composerを通じたライブラリ管理では、vendorディレクトリを共有することなく、ライブラリのコードは各自でセットアップする方法が推奨されています。このような運用は一見手間のように見えるかもしれません。誰かがセットアップしたvendorディレクトリを共有、またはバージョン管理に含めてしまうという方が、各自で`composer install`を実行する手間も省けていいのに……という声はしばしば耳にすることがあります。

しかし、ライブラリはあくまで他人の書いたコードであるため、その運用には十分に気を配らなければなりません。

Composerが一般的に用いられるより前の、古いPHP開発の体制下では、手作業でのライブラリ管理が行われてきました。

ライブラリ配布者のHPなどからライブラリをダウンロードし、プロジェクト内にコピー&ペーストして使用する方法は、単純で簡単な反面、運用上で多くの問題を引き起こしてきました。

ライブラリとプロジェクトの混合

プロジェクト本体とライブラリのコードをまとめて管理した場合、チーム開発などにおけるソース共有はライブラリを含む全体の配布になります。

複数のライブラリを使用する場合、小規模なプロジェクトでもライブラリのファイルによって全体として大きなサイズの配布になってしまったり、SVNなどのバージョン管理ツールを用いている場合、多数のadd履歴が追加されるなど好ましくない結果をもたらします。

またプロジェクト内にライブラリ本体のファイルが含まれている場合、誰かがライブラリのファイルを変更してしまうかもしれません。ライブラリはバグ修正などの更新に耐えられるよう、必ず配布時の状態を維持しておくのが望ましいはずです。

もしライブラリ本体のコードへ次々に変更が加えられてしまったら、ライブラリ本来の動きは次第に失われていき、配布サイトで提供されているようなドキュメントやサポートなどの情報も意味を持たなくなってきます。

ライブラリの更新

ライブラリは、プロジェクト本体とは平行して開発・メンテナンスが進められるため、日々仕様変更や更新などが追加されます。

プロジェクトで使用するライブラリにこれらの更新を取り込む際に、ライブラリに独自の変

注4）Monolog（URL https://github.com/Seldaek/monolog）

更が加えられていると問題が生じるのは言うまでもありません。

ライブラリの更新に対応するためには、ライブラリのバージョン管理とライブラリ本体のコードの維持が必要になります。

プロジェクトとライブラリをまとめて管理する運用では、思わぬ事故でライブラリ本体に変更を加えてしまったり、ライブラリ導入時点のバージョンがわからなくなったりと、ライブラリの柔軟な更新運用が妨げられてしまうケースが多々あり、結果として古いライブラリがいつまでも使われ続けてしまうといった問題が起こりがちです。

依存管理ツールの活用

こうした手運用でのライブラリ管理の問題を解決してくれるのが、依存管理ツールと呼ばれるツールの存在です。依存管理ツールは、ライブラリのダウンロードやインストール、バージョン番号の管理の手法を提供してくれます。プロジェクトメンバーは、ツールを通じてライブラリの導入を手軽に行えるため、プロジェクト内にライブラリの本体をまとめて頒布する必要がなくなります。

古くから存在するPHPの依存管理ツールとしては、PEARと呼ばれるツールが知られています(**図12**)。今でも一部の古いライブラリを利用する際には目にする機会があるかもしれません。PHPにおけるPEARは、古くからPHP向けの依存管理ツールとして提供されてきましたが、ライブラリのインストールに`include_path`のしくみを用いてグローバルにライブラリをインストールするなど、複数の開発環境を並行で動作させるのに不便な点があり、また配布されるライブラリの種類も豊富ではなかったため、あまり一般的に利用されているとは言えない状態でした。

こうした問題を解決してくれたのがComposerです。Composerではプロジェクトごとに閉じたライブラリ管理を行えるほか、ライブラリの公開もGitHubなどのサービスを通じて、誰でも簡単に行えるようになっており、現在、数多くのライブラリがComposer経由で導入できるようになっています。

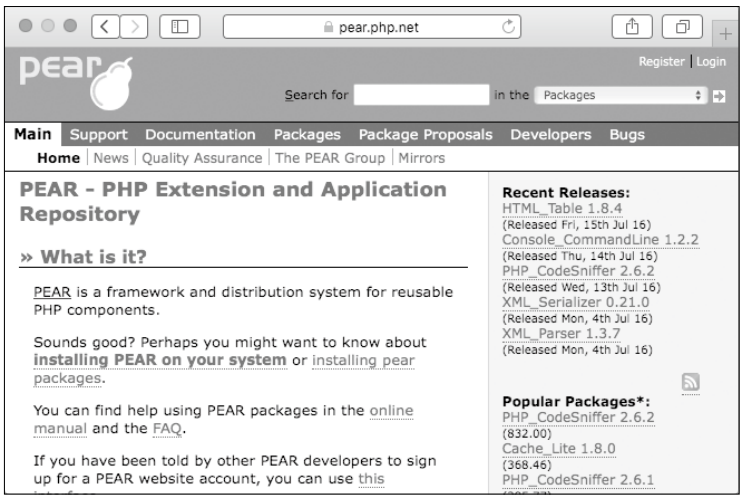
ライブラリとフレームワーク

Composer対応のフレームワーク

最近のPHP開発では、フレームワークを用いた開発が主流となっています。プロジェクト内で利用される特定の機能に関するコードを提供するライブラリと異なり、フレームワークはアプリケーション全体の大きな動きを制御する働きを持っています。

今では、LaravelやCakePHP3、SymfonyなどComposer対応のフレームワークが数多く存在します。Composer対応のフレームワークでは、ライブラリを追加することにより、機能

▼図12　PEAR(https://pear.php.net/)



の追加や拡張が簡単に行えるため、プロジェクトの必要に合わせて柔軟にフレームワークの姿を拡張することができます。

ライブラリの学習効果

開発効率を上げるためにフレームワークを学習するのは非常に大事なことですが、ライブラリに関する学習もまた重要な意味を持っています。

フレームワークは、それぞれに特有の文化を持っているケースが多く、新しいフレームワークの学習は何かと大変なものです。

Composer対応のフレームワークが多く登場する現在、ライブラリに関する知識は多くの開発現場で役立つPHP開発共通のコードテクニックとして役立たせることができます。

フレームワークのライブラリ化

Symfonyなどのフレームワークでは、フレームワークの機能の一部をライブラリとして切り分けて提供しているケースもあります。Symfonyフレームワークにおける Symfony Components は、フレームワーク内部で提供しているユーティリティ関数やサブ機能などをライブラリとして、ほかのフレームワークなどでも利用可能にしたものです。

コマンドラインでのPHP実行をサポートするConsoleやファイル操作をサポートするFilesystem/Finderなどのコンポーネントは FuelPHP や Laravel など多くのフレームワークでも利用されています。

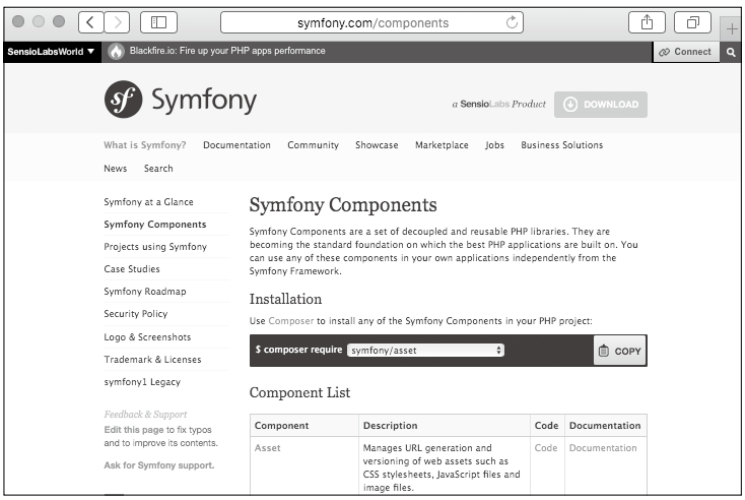
フレームワークの一部を部品化するコンポーネント化の動きは、Composer対応のほかのフレームワークでも見られるようになってきています。フレームワークベースの開発でも自由にライブラリを組み合わせながら細かい挙動を思い思いにカスタマイズできるようになりつつあります。それがSymfony Componentsです（**図13**）。

ライブラリを用いた開発スタイル

ライブラリを用いた開発スタイルは、特定の機能に関する実装の手間を省き、スムーズに開発を続けていくことができます。長期的なコードのメンテナンスも視野に入れ、Composerを用いた適切なライブラリ管理を行うことでライブラリは非常に便利なものとなります。またComposerでは自分で気軽にライブラリを公開することもできるため、複数のプロジェクトで頻繁に利用するコードなどをGitHubを通じてPackagistへ登録してみるのも、プロジェクト開発の大きな助けとなるかもしれません。ライブラリを公開することで、ほかの誰かの役に立ったり、Pull Requestなどを通じてコードに関するアドバイスや改良のアイデアをもらえるかもしれません。ライブラリ運用を通じて再利用性の高いコードをメンテナンスし続けることはプログラミング技術の向上に役に立つはずです。

さまざまなライブラリの利用を通じて、高速なシステム開発のノウハウや柔軟なコーディングのテクニックをぜひ身に付けてください。SD

▼図13　Symfony Components（http://symfony.com/components）



Author はやしりょう
Twitter @ryo88c

第3章 本命はBEAR.Sundayか PHPフレームワークの選び方
――システムの目的から振り返る

Webアプリの開発とは切っても切れない深い関係にあるフレームワーク。オープンソースとして公開されているものに絞っても、さまざまなものがあります。本稿ではフレームワークの選び方と使い方、その特徴を解説します。

フレームワークってなに？

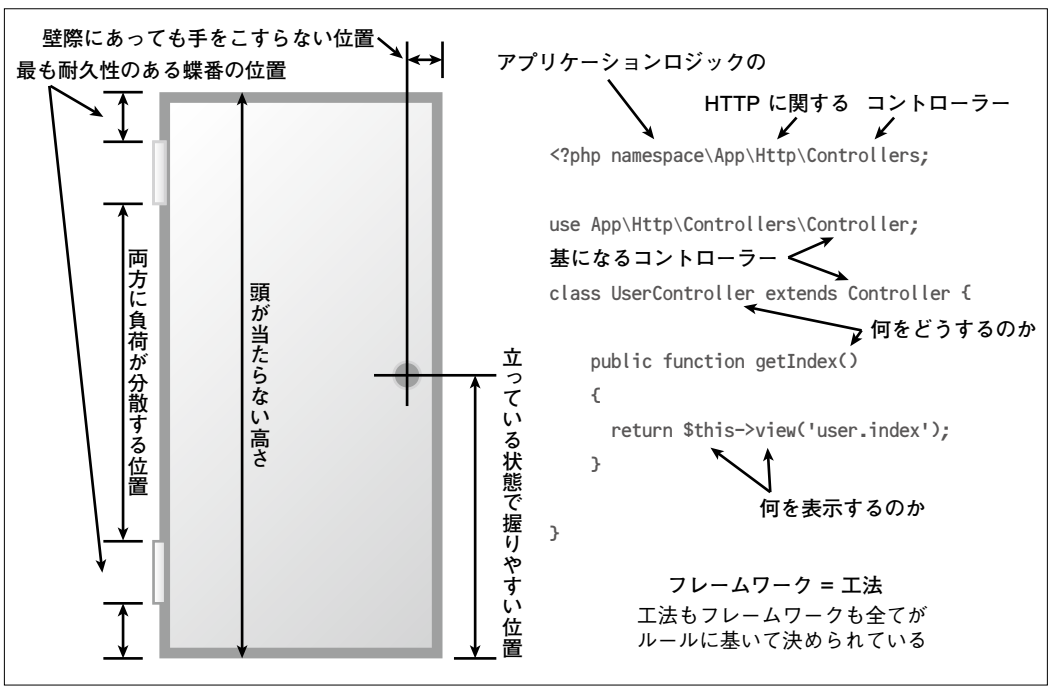
そもそもフレームワークとは何でしょうか。アプリケーションを開発するうえでフレームワークが担う役割について知ることで、開発するアプリケーションに適切なフレームワークを選ぶ方法を理解しましょう。

フレームワーク＝工法

建築工事の世界ではドアの大きさから壁の厚さ、果ては釘の長さにまで、およそ考えつくあらゆる物の材質やサイズとその使い方に厳格なルールがあります。このようなルールのことを工法と言い、道具や資材をどう使うべきか明示します。工法に従って建築工事を進めることで作業工程は画一化します(**図1**)。

アプリケーションにおけるフレームワークは、建築の世界における工法と言えます。一般的なアプリケーションに必要とされる機能が標準で備わっていたり、開発を効率的に進められるよう実装方法を手順化していたり、およそアプリ

▼図1　フレームワークは「工法」である



ケーションの開発に必要とされる要素が体系化されています。また、作業工程が画一化される副次的な効果として、フレームワークを使った開発工数の見積もりをより確かなものにします。

問題解決方法の集合

フレームワークの種類は実にさまざまです。たくさんの機能を標準で備えているものもあれば、ほとんど何も機能が備わっていないものもあります。機能が少ないから役に立たないというわけではなく、フレームワークのコンセプトに基づき「機能を実装しない」ことを選択しています。機能に関するこのような特徴は、どのようなWebアプリを開発するかによりメリットにもデメリットにもなります。

たとえばメールアドレスとパスワードでログインする機能を備えたアプリケーションを開発するとします。ログイン時にはフォームに入力されたメールアドレスが、メールアドレスとして正しい書式のものかどうかを評価しなければなりません。そのような評価機能のことをバリデーション(Validation)と言いますが、この機能を自分で一から実装するのは非常に手間がかかりますし、メールアドレスの仕様や、正規表現といったさまざまな技術への知識が求められます。

アプリケーションの開発はこのような問題を解決することの繰り返しで、フレームワークは開発者に解決方法を提供しますが、その内容には一定のルールがあります。

たとえばLaravelはより多くの種類のデータベースを扱えるようにするため、データベースの種類によらず一定の方法でテーブルを操作できるSchemaというクラスが実装されています。これにより異なるデータベースを利用しているアプリ同士でも、テーブル操作の実装は近似したものになります。

このような解決方法は、フレームワークが対象としているアプリがどのようなものであるかによって決定されます。頻繁に改修されるようなアプリを対象としているフレームワークならば、備わっている機能の多くは改修に関するものとなるでしょう。つまり、フレームワークは対象とするアプリを開発するうえで起こるであろう問題への解決方法の集合であると言えます。

ライブラリとの違い

フレームワークと同じように語られるものにライブラリがあります。フレームワークとライブラリは何が違うのでしょうか。これらの違いを理解することで、アプリケーション開発におけるフレームワークの役どころが見えてきます。

ライブラリとは

フレームワークは工法のようなものであると述べましたが、それに対してライブラリは工法を構成する道具や作業手順そのものと言えるでしょう。フレームワークが文法であるならばライブラリは単語です。単語は単語だけで何らかの事象を示すことができるよう、ライブラリもそれ単体で何らかの機能を提供しています。

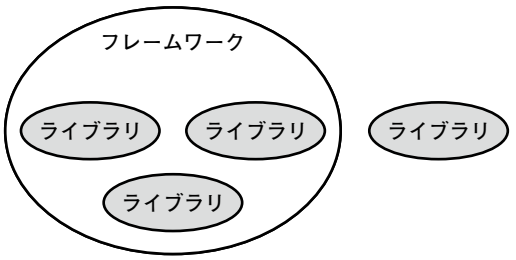
ライブラリとフレームワークの関係

第2章でも紹介されているSymfony componentは、Symfonyを構成する任意の機能を取り出して単体で使えるように作られています。Zend frameworkも同様に機能を単体で使えます。このように、ライブラリとフレームワークの関係は択一的なものではなく、むしろ相補的です(**図2**)。

PHPにはComposerというパッケージ管理システムがあります。依存関係の解決といって、あるライブラリを使ううえで必要なほかのライブラリを自動的にインストールするようなしくみがあり、近年のアプリケーション開発には欠かせないツールとなっています。

ここ数年の間にリリースされているフレームワークのほとんどはこのComposerを導入しており、フレームワークとライブラリの連携が促進されています。Composerが導入されているフレームワークは、フレームワークに標準で備

▼図2　フレームワークとライブラリの関係



ライブラリはフレームワークを構成する機能として動作するほか、ライブラリ単体でも動作する

わっている機能の交換が比較的容易で、アプリケーションの要件に合わせてライブラリを交換することでフレームワーク自体をカスタマイズするような使い方を可能にしています。また、このしくみを利用することでフレームワーク標準の機能を自分が使い慣れているライブラリに切り替えることもできます。

フレームワークのメリットとデメリット

本章ではフレームワークのメリットとデメリットを、フレームワークが持つ特徴の説明を通じて述べます。フレームワークが持つ特徴のメリットとデメリットを知り、フレームワークを選ぶ際の判断材料としましょう。

プログラムをより読みやすいものとする

「計算機プログラムの構造と解釈」という本の序文に次の一節があります。

> “プログラムは、人間に読まれるために書くべきであり、それがたまたま機械でも実行可能であるにすぎない。”

プログラム言語は人間が読むために発案されたものであり、プログラムは人間が読みやすいものとなるよう書くべきであると提言されています。フレームワークも同様の考えに基づき、プログラムの理解を助けるさまざまな機能を備えています。

MVCアーキテクチャのフレームワークにはルーティングという機能が備わっています。たとえば/foo/barというURLをリクエストするとFooController::bar()というメソッドが実行されるようなしくみで、URLを知っていれば対応するクラスとメソッド名がわかりますし、逆もまた然りです。

このようなしくみによってプログラムの読みやすさは向上しますが、読みやすさは説明的な構文によってもたらされることが多く、必然的にコード量が増えます。プログラムの品質を一定に保ち保守性を高めなくてはならないような場合に真価を発揮しますが、実験的なアプリケーションを手早く作らなければならない場合には冗長に感じられることもあるでしょう。

機能を組み合わせてアプリケーションが開発できる

フレームワークには、フォーカスしている問題があり、その問題が解決できるよう最適化されていますので、高機能なフレームワークの場合フレームワークに備わっている機能を組み合わせるだけでアプリケーションの大部分ができあがるようなものもあります。

これはプラスチックのレールを組み合わせてつくった線路の上で電車を走らせるおもちゃのようなもので、レール自体を作る手間が省ける分開発速度が向上し、フレームワークを使う大きなメリットであると言えます。

しかし反面、提供されているレールの組み合わせでは実現できないかたちの線路を作ろうとすると、既存のレールが持っている特徴を深く理解し、それらと統合できるレールを自分で作らなければなりません。フレームワークで考えるならば、フレームワークを構成しているプログラムを熟読して、フレームワークの内部仕様に沿った独自の機能を実装しなければならないということになります。

また、このような場合フレームワーク自身のバージョンアップに伴う後方互換性の管理も必要となり、フレームワークを使う大きな負担で

あり、デメリットであると言えます。

スタートアップのWebサービスがリリースされた直後はピボット(路線変更)が頻繁に起こり、その内容の予測はつきにくいです。これは、事前の分析では適切であると判断されたフレームワークがそうではなくなる可能性があることを意味しています。豊富な機能による開発速度の向上には、柔軟性が犠牲になる場合が多いと理解しておくことが肝要です。

フレームワークの選び方

これまでフレームワーク自身の役割、ライブラリとの関係性、フレームワークが持つメリットとデメリットを紹介してきましたが、本節では状況に応じたフレームワークの選び方について説明をします。

継続開発の有無

アプリケーションには実際に作ってみなければわからないことが数多くあります。そのため、実験的にパイロット版を作ることなどもありますが、このような場合継続して使うことはほとんどありません。

そのようなアプリケーションを開発する場合はフレームワークに高い柔軟性を求める必要はなく、いかに少ない工数でイメージするものを作ることができるかが重要です。O/R マッパーやテンプレート、フォームのバリデーション、URLのルーティングなど一般的なWebアプリに求められる機能がすぐに利用でき、かつ少ないコード量で実装できるフレームワークが良いでしょう。

内製で開発する

「社内の会議で立案された新機能がアプリケーションの仕様上実装できない！」というような状況は避けねばなりませんので、そのような状況において最も重要なのはフレームワークの柔軟性です。

進化論には次の有名な一節があります。

> “強い者が生き延びたのではない。変化に適応したものが生き延びたのだ。”

立案された新機能が実装できないということは、アプリが事業の成長を制限していることを意味します。他サービスと競い合っているようなWebサービスのアプリに最も求められる性能は変化への柔軟性です。

受託開発をするシーンで

受託案件の場合、あらかじめ合意した工数で作らなければなりません。したがって、フレームワークに最も求められる性質は工数見積もりのしやすさです。しかし、工数見積もりのしやすさをフレームワークだけで成立させるのは非常に難しく、設計をシンプルにすることも大事です。

たとえば、データベースの設計をフレームワークに備わっているO/Rマッパーが解釈できるリレーションだけで解決できるようにするなどの工夫が必要です。また、フレームワークに由来する問題が起きた時、初めて使うようなフレームワークでは解決にどの程度の時間が必要か予測が難しい場合がありますので、1つのフレームワークを繰り返し使い熟練度を高めることも大事です。

開発チームの構成で考える

他の選び方とは観点が異なりますが、開発チームの構成はほとんどのケースで考慮すべき重要な指標です。人数はもちろんですが、コミュニケーションの取り方から参加者間の技術力の差など、開発チームの構成はさまざまな要素によって分類ができます。

たとえば、全員がリモートで開発を行っている場合、参加者間の技術力の差が気になるようなことは稀です。これはリモートで開発をしている以上分業が徹底されるからです。反面全員が同じ場所で開発するような場合には技術力の差は非常に大きな意味を持つことになります。

◆ ◆ ◆

このように、開発チームの構成によってあらわれるチームの性質はさまざまですが、基本的には人数と距離感で考えると良いでしょう。人数が多いならばシンタックスシュガー[注1]がしっかりとしていて誰が書いても似たような実装になるフレームワークが適していますし、技術力の差が大きかったり、リモートでの参加が多数を占めたりするような距離感のあるチームであれば、機能同士を疎結合できるような機能を持ったフレームワークの方が開発しやすいでしょう。

開発の現場がおかれている状況を見極めよう

繰り返しになりますが、これらの選び方は択一的なものではなく相補的なものです。

受託案件でもリリース後に継続的な開発が求められるケースもあるでしょうし、技術力に大きな開きのある編成の開発チームでパイロット版のアプリケーションを開発することもあるでしょう。大事なのはそれぞれの状況でフレームワークに求められている性質を見極めることです。たとえ状況に見合ったフレームワークを選べなくても、状況に見合っていないという事実を認識すること自体が重要です。なぜならば、その認識を踏まえて体制や心の準備ができるからです。

フレームワークの紹介

最後に、PHPで実装されているフレームワークを紹介します。ここで紹介されていないフレームワークだからといって、それが有用ではないということではありません。

Zend framework

現在のPHPパーサであるZend Engineの開発チームが前身であるZend Technologies[注2]が開発しているフレームワークです。ほかのフレームワークと大きく異なり、ほとんどすべての機能が単体でも利用できるようコンポーネント化されています。それらのコンポーネントは非常に多くのフレームワークやライブラリで活用されていますので、非常に学習効果の高いフレームワークであると言えます。

Symfony

開発や保守の効率化にフォーカスされており、大規模な開発案件で真価を発揮するフレームワークです[注3]。Zend frameworkと同様に機能がコンポーネント化されており、とくにO/Rマッパーであるdoctrineは非常に有名で、単体で利用されることも頻繁にあります。Zend frameworkと同様、学習効果の高いフレームワークと言えます。

Laravel

スキャフォールドや単体テストなど、従来のフレームワークがフォローしている機能はもちろん、Gulpベースのフロントエンドのタスクランナーが同梱されているなど、昨今のWebアプリケーションに求められているデリバリースタイルが踏襲されている、非常に高機能なフレームワーク[注4]です。

CakePHP

スタンダードでかつ学習コストが低く、フレームワークというもののしくみに慣れるのに適したフレームワーク[注5]です。また、日本人の利用者が非常に多く、日本語で書かれたユースケースが多いという特徴もあります。

CodeIgniter

軽量で速度重視であることで知られているフレームワーク[注6]で、比較的自由度が高いことでも知られています。1人で実験的なアプリケーショ

注1）説明的な構文体系とすることで、プログラムに一定の読みやすさを与えるしくみ。
注2）URL https://framework.zend.com/）
注3）URL https://symfony.com/
注4）URL https://laravel.com/
注5）URL http://cakephp.jp/
注6）URL http://codeigniter.jp/

ンを開発するような場合に適しているでしょう。

FuelPHP

FuelPHPで実装されたアプリケーションを、FuelPHPで実装された、ほかのアプリケーションからライブラリのように扱うことができるHMVC[注7]という珍しいアーキテクチャが採用されているフレームワーク[注8]です。一度実装した機能をほかのアプリケーションでも流用したいというようなニーズのある状況において真価を発揮します。

BEAR.Sunday

BEAR.Sundayは、これまで述べたフレームワークとは根本的に思想が異なります[注9]。ほかのフレームワークが提供するような、たとえばフォームのバリデーションですとか、O/Rマッパーのような機能はいっさい持っていません。

BEAR.Sundayそれ自体は接着剤のようなもので、自身で実装を持たない代わりにあらゆるライブラリを内包することが可能です。バリデーションやO/Rマッパーもライブラリを導入することで利用できます。また、上述したフレームワークのアーキテクチャはすべてMVC(Model View Controller)モデルですが、BEAR.Sundayはリソース指向アーキテクチャによって実装されています。

設計に対する自由度が非常に高く、ほかのフレームワークに見られるようなモデルの構造に対する制約などはいっさいありません。そのため、設計に重きが置かれているような現場で非常に力を発揮します。ちなみに筆者が個人的に最も好んで使っているフレームワークです。加えてアスペクト指向でプログラミングできるので、業務プロセス上には現れないメタ処理を実装から分離できるというメリットもあります。

注7) Hierarchical model–view–controllerの略で、従来のMVCモデルに階層構造を与えた派生的なアーキテクチャ。
注8) URL http://fuelphp.jp/
注9) URL https://bearsunday.github.io/

最後に

いかがでしたでしょうか。今の時代においてWebアプリをフレームワーク抜きで開発することは考えにくく、初学者であろうと熟練した技術者であろうと等しく知識が求められます。駆け足で説明したのでフレームワークを普段から使いこなしているような人にとっては既知の話ばかりで物足りなかったかもしれません。

フレームワークを使うと、フレームワークを使う前には想像もできなかったほどに開発作業が効率化されます。しっかりとしたマニュアルが用意されているフレームワークがほとんどですので、中身をすべて理解していなくとも使うことができます。

筆者はWebアプリの受託開発を生業としているため、これまで非常に多くのフレームワークを利用してきました。どのフレームワークでも言えることは、たくさんの開発者の苦労や工夫が反映されているということです。つまり、フレームワークのソースコードはそれ自体が多くの知見によって成り立っているノウハウの塊と言えます。

自分の手に馴染むようなフレームワークと出会うことができたら、ぜひ一度ソースコードも読んでみてください。きっと使っているだけでは得られない新たな発見があります。

PHPのフレームワークはほとんどがオープンソースで、世界中の開発者が協力しあって改善を続けています。フレームワークを使い、ソースコードを読むことであなたもその協力の輪の中に入ることができるのも、フレームワークを使う魅力の1つです。

筆者もフレームワークを通してさまざまな開発者と出会い、刺激を受けています。あなたともそのような関係になることができればと思い、今回フレームワークというものを紹介しました。うまく紹介できたかわかりませんが、あなたの開発環境がより良いものになれば幸いです。SD

Author 小山 哲志（こやま てつじ）
Twitter @koyhoge

第4章 参加しませんか？ PHPのユーザコミュニティ ――チャットルームからハッカソンまで

現在のPHPコミュニティは、さまざまなものがゆるく広く繋がっているという状態です。本章ではその主なものを紹介しましょう。

総合

日本PHPユーザ会

日本PHPユーザ会は2000年に発足した、日本で最も古いPHPのユーザ会です。ユーザ会と言っても会員名簿があるわけではなく、実態は不確定なゆるふわユーザ会です。現在はほとんど流量がありませんが、php-usersなどのメーリングリストを運用しています[注1]。

PHPユーザーズ（日本語）on Slack

2016年2月に@msngさんが立ち上げた、日本語でPHPの話をするSlackです。各種勉強会やカンファレンスでの宣伝効果もあって、参加者は着実に増え続けています[注2]。

PHPプロダクト

CakePHP

おそらく、日本で最も大きいフレームワークのコミュニティはCakePHPだと思われます。ドキュメントの日本語翻訳を共同で行ったり、CakeFestなど海外カンファレンスの報告会なども開催されています[注3]。

日本Symfonyユーザ会

Symfonyもファンの多いフレームワークです。日本Symfonyユーザ会は定期的にミートアップを開催して、ノウハウの共有をはかっています[注4]。

Laravel Meetup Tokyo

最近ユーザ数を増やしている新進気鋭のフレームワークがLaravelです。実際の案件でLaravelを使っているという話も多く聞くようになりました。関連書籍の出版やリアルイベントの活動も、それらに貢献しているのだと思います[注5]。

FuelPHP＆CodeIgniterユーザの集い

CodeIgniterと、そこから派生する形で生まれたFuelPHPも、シンプルな構造で人気のあるフレームワークです。ここ最近はミートアップの開催は止まっていますが、ドキュメント日本語翻訳などの活動は継続中です[注6]。

地域コミュニティ

PHP勉強会東京

2005年に第1回が開催され、途中幾度かの中断を挟みながら、先日めでたく100回目を迎えた老舗の勉強会です。ここ数年は毎月定期的に開催されており、毎回50名程度の参加者を集めています。驚くのは、これだけの回数を開催しているにもかかわらず、常に1/3から半分程度が初参加だということです[注7]。

PHPBLT

2015年末に始まった、参加者全員がライトニングトークをするというコンセプトで始まった東京の勉強会で、現在まで5回開催されています。比較的“濃い”人たちの“濃い”話題がたっぷり聞けます[注8]。

関西PHPユーザーズグループ

関西のPHPコミュニティも歴史が古く、何度か活発化と停滞を繰り返していましたが、2011

注1）URL http://www.php.gr.jp/
注2）URL https://slackin-phpusers-ja.herokuapp.com/
注3）URL http://cakephp.connpass.com/
注4）URL http://www.symfony.gr.jp/ https://symfony.doorkeeper.jp/
注5）URL https://laravel.doorkeeper.jp/ https://www.facebook.com/groups/laravel.jp/
注6）URL https://atnd.org/events/70728 https://groups.google.com/forum/#!forum/fuelphp_jp
注7）URL https://phpstudy.doorkeeper.jp/
注8）URL http://phpblt.connpass.com/

年に現在の形に落ち着き安定した活動が定着しています。ほぼ毎月開催される勉強会はすでに34回を数え、あとで紹介するPHPカンファレンス関西の主催団体でもあります[注9]。

Sapporo.php

札幌はオープンソースカンファレンスが東京以外の土地で初開催された場所であり、ITコミュニティの古い歴史があります。札幌のPHPコミュニティは、人の入れ替わりによりアクティブな時期とそうでない時期がありましたが、昨年あたりからまた活発な活動が始まっているようです。あとで紹介するPHPカンファレンス北海道の共催団体です[注10]。

Fukuoka.php

最近、IT関係がとみに活発で熱い地域が福岡です。福岡のPHPコミュニティであるFukuoka.phpでは、20～30人規模の勉強会を不定期に開催しています。PHPに限らず、他言語やセキュリティ、インフラなどさまざまなトピックを扱っていて、誰でも歓迎する和やかな雰囲気だそうです。あとで紹介するPHPカンファレンス福岡の主催団体です[注11]。

Nagoya.php

名古屋のPHPコミュニティがNagoya.phpです。不定期に勉強会が開催されているようです。内容は初心者向けから深い話まで、多岐に渡っています[注12]。

カンファレンス

PHPカンファレンス北海道 2016

かつて2012年に一度開催されましたが、その後途絶えていたPHPカンファレンス北海道が今年4年ぶりに開催されました。200人の参加者が集まり、2トラック13個の発表と、7つのスポンサーセッション、そして5つのLTが行われました[注13]。

PHPカンファレンス福岡 2016

福岡のPHPカンファレンスは、@cakephperさんのとあるつぶやきから物事が転がるように動き始めて、ついに昨年開催されたという経緯があります。さて2回目となる今年のPHPカンファレンス福岡は、さらに規模を拡大して開催されました。2トラック20個の発表に2つのスポンサーセッション、8つのLT、さらに懇親会でも8つ程度のLT発表がありました[注14]。

PHPカンファレンス関西 2016

2011年に始まったPHPカンファレンス関西は、今年で6回目を迎えました。ここ数年は、東京のカンファレンスよりも積極的にPHP以外の分野の発表を取り上げていた印象があります。このような各カンファレンスの微妙な味付けの違いを楽しむのも、また楽しいものです。

さて今年のPHPカンファレンス関西では、3トラック17個の発表、6つのLT、懇親会では10個のLT発表がありました[注15]。

PHPカンファレンス 2016（東京）

17回目となる東京のPHPカンファレンス2016は11月3日の開催が決まっていて、現在準備が進んでいるところです。発表やLTの募集、当日スタッフの募集もまだこれからですので、興味がある方はインターネット上の情報に注意していてください（11月3日（木・祝）@大田区産業プラザPiO）[注16]。

PHP Matsuri

カンファレンスではありませんが、PHP Matsuri（祭り）のことも紹介しておきましょう。これは泊まり込みで行われる合宿形式のハッカソンです。各人がおのおのテーマを決めて、まる一日開発三昧を行い、最後にショートプレゼンをするという形です。開発だけではなく、途中さまざまな講演や企画もあります。永らく開催が途切れていましたが、来年初頭を目指して現在準備中だそうで、すごく楽しみです[注17]。SD

注9）URL http://www.kphpug.jp/ https://kphpug.doorkeeper.jp/
注10）URL http://sapporo-php.net/
注11）URL https://www.facebook.com/groups/355557634510818/
注12）URL https://nagoyaphp.doorkeeper.jp/
注13）URL http://phpcon.sapporo-php.net/2016/
注14）URL http://phpcon.fukuoka.jp/
注15）URL http://conference.kphpug.jp/2016/
注16）URL http://phpcon.php.gr.jp/2016/
注17）URL http://www.phpmatsuri.net/

コードを読みやすくするための
6つの書き方

「良いプログラム」のための「良いコメント」

Author きしだなおき
Blog http://d.hatena.ne.jp/nowokay/
Twitter @kis

はじめに

多くのプログラミング言語で、プログラムの処理や構造を記述する「コード」のための文法以外に、自由な文章を記述できる「コメント」のための文法が用意されています。コメントはおもに、コードを読みやすくする目的で使われます。読みやすいコードのためには、良いコードを書くことはもちろんですが、コメントを適切に書くことも大きな助けになります。

コメントには文法の制約がほとんどないので、自由に書けます。そのため、プログラマのセンスや個性が表れやすい部分であるとも言えます。

本記事では、どのようにコメントを書いて考えを書き残せば、コードを読みやすくし、作業をしやすくし、ひいては良いプログラムが作れるのかということを考えていこうと思います。なお本稿で示すコード例は、Javaをベースにした仮の言語です。

良いコード＞だめなコード＋良いコメント

コメントの話をする前提として、だめなコードにどれだけ良いコメントを付けても良いコードにはかなわない、ということを挙げておこうと思います。

コメントはあくまでコードの補足であり、コードだけで十分に読みやすく必要なことがわかるのであれば、コメントは不要です。だめなコードは、読み手を戸惑わせたり読む量を増やしたりと、読みにくくする要素を持っています。そこをコメントで補おうとしても、良いコード並みの読みやすさにはなりません。

極端な例を挙げましょう。

```
// この処理は不要
a = ((4 * a + 6) / 2 - 3) / 2;
```

この処理は、計算すると結局は元のaになるのだとすれば不要です。不要なコードというのはだめなコードの代表です。読まなくていい処理を読ませるということになるので、コードを読みにくくさせています。ここに「不要」というコメントを書いたとしても、このコードを読まなくてよくなるわけではありません。単に読むべき量を増やしています。不要な処理であれば処理自体を削除し、読みやすくしましょう。

もちろん、最初から良いコードを書けるわけではないので、補足としてコメントが必要になることもあるでしょう。しかし、コードが最も読みやすいのは、当然“良いコードに良いコメントが付いた状態”です。本稿で考えるコメントは、「良いコードをさらに読みやすくするためのもの」という前提で進めます。

コメントは必要か

良いコードを書けばコメントは必要ないとい

う話について、もう少し考えてみましょう。

適切に関数や変数などを導入し、関数名や変数名をわかりやすく付けることで、コメントが必要なくなるという考え方があります。しかし、コメントのほうが表現力は高いので、コードをどれだけうまく書いても、コメントの表現力をすべて補うことはできません。またコードは、動作やパフォーマンス、動作の追いやすさに直接つながるものなので、単に読みやすさのためだけにコードを書き換えるということができない場合もあります。

たとえば、次のようなコードがあるとします。

```
a = a ^ 0b1111;
```

これは、変数aの値の下位4ビットを反転させるコードです。しかし、多くのアプリケーションではビット演算が必要になることがあまりないので、XOR演算「^」の振る舞いに不慣れな人も多いと思います。

これをコメントを使わずに読みやすくしようと思うと、関数を導入するという手段が考えられます。たとえば、次のようになります。

```
int invertLower4bit(int n) {
  return n ^ 0b1111;
}
a = invertLower4bit(a);
```

関数名によって、下位4ビットを反転させる処理だということが示されました。けれども、処理系によってはパフォーマンスに影響することもあるでしょう。このようなコードが必要になる状況では、パフォーマンスにも気を使う必要があることが多いかと思います。

反転するビットパターンが増えると、そのパターンごとに関数を定義することにもなりますが、これは現実的ではありません。同じ関数を複数定義するということは避けないといけないので、すでにそのビットパターンを反転する関数が定義されているかどうか、いちいち確認する必要も出てしまいます。

このコードは、次のようにコメントを入れれば、読みやすくするという目的を達成できます。

```
// 下位4ビットを反転させる
a = a ^ 0b1111;
```

別の例を考えてみましょう。次のような例であれば、関数の導入によって読みやすくするということは難しくなります。

```
openDevice();
openDevice();
```

びっくりするコードです。普通にみると、単なる作業ミスです。ここでは確実に動作できるように、同じ処理を繰り返しているのです。

本誌の読者にはデバイスに近いコードを書く人も多いようですので、こんなコードが必要になった経験がある方もいらっしゃるかもしれません。カスタムハードウェアを制御するコードを書くとき、ハードウェア自体もデバイスドライバも市販のものより不安定なために、このようなコードが必要になることもあります。

このコードを説明するような関数を導入するとしたら、次のようになるのでしょうか。

```
void openDeviceTwiceToMakeSure() {
  openDevice();
  openDevice();
}
```

しかしこれは、関数名として良くないと思います。関数は処理を抽象化するものであり、関数名には実装の詳細を含めるべきではないからです。もし二度でも失敗するから三度にしようというときには、関数名を付け替える必要も出てきます。

この関数は次のようにしたほうがいいでしょう。

```
/**
 デバイスを確実にオープンする
 */
void openDeviceCorrectly() {
  // 念のため2度オープンする
  //(現状では1度では失敗することがある)
  openDevice();
  openDevice();
}
```

「良いプログラム」のための「良いコメント」

関数は処理を抽象化するために定義し、実装の説明はコメントで行っています。もしハードウェアやデバイスドライバが安定して、一度だけのopenDevice関数の呼び出しで確実に動作するようになったときには、重複する関数呼び出しを削除すればいいでしょう。

このように、コードの説明のためにはコメントを付けるほうが適切である場面は多くあります。コードはあくまで処理について記述し、コードの意図や制約など、処理に関する情報はコメントで書くようにしたほうがいいでしょう。

コメントの分類

ここで、コメントを分類してそれぞれについて考えてみます。もちろんこのような分類は境界があいまいであるため、はっきりと分けられるものではないですが、考え方の基準として分類があるとわかりやすくなると思います。

最初に、コメントの種類を大きく分けると、コードからわかるものとコードからはわからないものがあります。

コードからわかるコメント

コードに書いてあるコメントというのは、コードをよく読めばわかることを記述したもので、次のようなものが挙げられます。

・コードの直訳
・コードの要約

このようなコメントは、変数や関数をうまく導入するなどしてコードをうまく書くことや、同じようなコードに慣れることで、不要になったり減らせたりする可能性があります。

☑コードの直訳

コードの直訳というのは次のようなものです。

```
// ファイルを開く
openFile();
```

このようなコメントは、関数名などが適切であればなくてもかまいません。もし英語でコメントを書く必要があると、次のようにまったくコードと同じものにもなります。

```
// open file.
openFile();
```

また、次のように構文を説明するものは、記述に慣れてくれば不要になるので、その言語の初学者を対象とするコード以外ではほとんど意味がありません。

```
// iを1増やす
++i;
```

単にコードの直訳であれば、読むべき量を増やしているだけということにもなりがちなので、そのようなコメントは減らしていきましょう。

ただ、コード中のキーワードやシンボル名の多くは英語をベースにして記述するため、英語が非ネイティブであれば日本語に比べて一瞬で識別しにくく、日本語でコメントを書くことで読みやすくなるという効果はあると思います。

☑コードの要約

何行かのコードについて、そこで行っていることをまとめるコメントです。

```
// 三角を描画
graphics.moveTo(100, 10);
graphics.lineTo(150, 70);
graphics.lineTo( 50, 70);
graphics.lineTo(100, 10);
```

関数に付けるコメントも、要約の1つと言えます。代表的なコメントでしょう。

コードからはわからないコメント

コードからはわからないコメントとしては、次のようなものが挙げられます。

・コードの意図
・コードの使い方

・コードのメタ情報
・文法を補完する
・作業情報

コードからはわからない情報がコメントとして記述してあると、コードを読むときに大きな助けになるでしょう。このような内容は、あとから復元することが難しいため、できる限りコメントなどで残しておく必要があります。

☑コードの意図

コードからはわからないコメントの代表としては、コードの意図があります。このコードで実際に何をしたいのかという意図です。

よく、「コメントには、やっていることではなくやりたいことを書くべき」ということが言われます。コードというのは何かがやりたくて書くものですが、やりたいこととその実装が一致しないことも多くあります。そのような場合は、何をやりたくてそのコードを書いたのかという、コードの意図をコメントとして残しておいたほうが良いでしょう。たとえば次のコメントは、単なるコードの直訳です。

```
if (line == null) { // 読み込んだ行がnullのとき
```

次のようにコメントを書き換えて、nullの判定によって実際に何をやろうとしているのかを書くほうが、役に立つコメントだと言えます。

```
if (line == null) { // 読み込みが終了したとき
```

コードの前後や利用のされ方、ライブラリの仕様などから意図を把握できることも多いですが、コードやドキュメントの確認に時間がかかることも多く、コメントとして残しておけばコードの読みやすさが格段に上がります。

☑コードの使い方

おもにクラスや関数、変数に付けるコメントで、その使い方をコメントとして書きます。

```
/**
 読み書きする前にopenを呼び出し、
利用が終わったらcloseを呼び出す
 */
class Device {
  void open();
  void close();
  ...
}
```

設定情報を表す変数には、使い方を表すコメントが必要になることも多いように思います。

```
// 出力先を、fileかdatabaseで指定する
output = file;
```

言語によっては、コメントの書き方によってドキュメント生成ツールが解釈できるようになることもあり、その場合にはツールに対応した書き方でコメントを書くほうが良いでしょう。

☑コードのメタ情報

コードからはわからないコメントとして代表的なのが、著作権表示やライセンスなどコード自体の情報を表すメタ情報です。オープンソースプロダクトのソースコードには必ずといっていいほど、コードの先頭にこのようなコメントが埋め込まれています。たとえば、OpenJDKのソースコードには、**リスト1**のような著作権表示が埋め込まれています。

ライセンスや組織のルールによってソースファイルへの記述が求められていることもあり、工夫によって代替することが難しいコメントです。

☑文法を補完する

構文として用意されていない情報で、「こういった構文があれば表せるのに」というものをコ

▼リスト1　OpenJDKの著作権表示

```
/*
 * Copyright (c) 2001, 2010, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
...(略)...
```

メントとして表すことがよくあります。

たとえば、JavaScriptやPHPなど、関数の引数や戻り値に型を表す構文がない言語では、次のようなコメントを埋め込んで型を明示するということがあります。

```
function /* int */ countWord
    (/* string*/ text)
```

このようなコメントは、その情報からコードの検査を行うためのツールに対応していることも多く、うまく使えばコードのミスを減らし、品質を上げることができます。

☑作業情報

作業のための情報をコメントとして埋め込むこともよく行われます。よくあるのが、これから作業する予定を書いておくTODOです。

```
if (status == 404) {
  // TODO NotFound画面を表示する
  throw new Exception();
}
```

IDEによってはコメント中の「TODO」を一覧できる機能を持つものもあり、そのような機能を活用すると作業の効率を上げることができます。ただ、これらの情報はコメントではなくほかのツールで表すほうが適切な場合も多く、大きなものはタスク管理ツールなど、なんらかのツールの利用を検討したほうがいいと思います。

書いてはいけないコメント

書いてはいけないコメントも挙げましょう。

誤ったコメント

誤った情報は困ります。つまり、実際の処理とは異なった説明をするコメントです。

たとえば次のようなものです。

```
// カウントを半分にする
count *= 2;
```

カウントを半分にすると書いてありますが、実際の処理としてはカウントを倍にしています。

このようなコメントが良くないのは、これではコードが正しいのかコメントが正しいのかわからないということです。

多くの場合はコメントのほうを信じて、コードが間違っていると思ってしまうでしょう。それを確認するには、前後の処理を見直して整合性がとれているかを確認し、動かしてみて意図どおりに動いてみるか確認し、コードの意図を確認するために仕様を確認するというように、いろいろなものを確認していく必要が出てきます。コメントのせいで無駄な作業が発生するのは本末転倒です。

もちろん、最初からこのようなコメントを書くということは少ないと思います。このように、コメントとコードがかけ離れてしまうのは、コードを修正したときが多いでしょう。そして、処理を書き換えたのに関数の説明を書き換え忘れるということは本当によく発生します。作業時はもちろんですが、コードレビューなどでも、処理が変わったときに対応するコメントがないか気をつけましょう。

関係ないコメント

処理とは関係ないコメントも、読む側を戸惑わせてしまいます。コメントに対応する処理がそこにないという意味では、ウソの一種と言えるかもしれません。たとえば次の例です。

```
// 牛乳を買う
openFile();
```

コードに買い物メモを書いてはいけません。思いついたときに目の前に開いてあったのがそのコードだとしても。

これはあくまで冗談ですが、実際によく起こるのは、コードを大きく削除したときにコメント部分の削除を忘れたり、コードの一部を移動したときにコメント部分を移動し忘れたときなどです。これも作業時やコードレビューで気を

つけましょう。

どのようにコメントを書けば良いのか

本稿の本題、良いコメントを書くときのやり方や方針をいくつか挙げてみます。

1 暗黙の条件を記述する

コードが実行されるときに暗黙の前提条件があれば、記述しておいたほうがいいでしょう。

```
// 成功率を求める。totalが0の場合はこの処理は
// 呼び出されない。
rate = success / total;
```

ここでは、割り算の分母が0になることがないことを示して、あらためてチェックを行う必要がないことを示しています。

2 変更時にはまりそうなところに注意書きを書く

コードを不用意に変更するとうまく動作しなくなることに、そのコードの記述中に気づく場合があります。そのときはあとでそのような条件にはまってしまうことがないよう、コメントを残しておいたほうがいいでしょう。

```
// 警告を出す閾値。
// 小さくすると警告が出やすくなるが、
// 5以下だと負荷が高くなり動作に影響する
warningThreshold = 30;
```

このような値の設定の条件以外にも、処理の順番を変えると不具合が出るとわかっている、といった場合もあります。あとでハマるのは時間の浪費になります。気づいたときにコメントを残すようにしましょう。

3 いつもと違うコード

いつもと違うコードを書く必要があれば、コメントでその理由を書いておきましょう。

```
// Lombokが使えないのでgetterを定義
public int getCount() {
  return count;
}
```

普段はLombokというライブラリを使っているが、このようなコードが必要ないプロジェクトの一部で、何らかの事情でLombokが使えないのでこのようなコードが必要になっている、ということを表しています。実際は、そのような事情がどのようなものかも書いておいたほうがいいでしょう。

4 テストに対するコメント

テストコードでは、確認したい事柄に対して代表的な値を選んで処理を呼び出し、振る舞いを確認するようにコードを書きます。そのとき、コードに残るのは代表として選んだ値だけで、何を確認したかったのかという情報は残りません。そのため、何をテストしているのかということがコードからはわかりにくくなります。この場合、たとえば**リスト2**のようなコメントを書けます。テストコードとコメントの内容はほとんど同じですが、ここでは、このテストが処理の境界になる値についてのテストであることを記述しています。すべての値をチェックするのではなく境界だけを選んだということを示しているわけです。このように、どのような方針でテストを行っているかということも、コメントとして書いておいたほうがいいでしょう。

5 コードを書く前にコメントを書く

コメントは、コードを書くのと同時か、そのあとで書くことが多いと思います。そうではなく、コードを書く前にまずコードの概要をコメントとして書くというやり方です。

混み入った処理を書く場合や、まとまった量のコードを書く必要がある場合に、まずはコメントとして日本語や仮コードで処理を書いておくというやり方で、筆者は心の中で「コメント

▼リスト2　テストコードに含まれない情報をコメントで表す

```
//境界のテスト
assertThat(checkAge(19)).isFalse();// 19まではFalse
assertThat(checkAge(20)).isTrue(); // 20からはTrue
```

ファースト」と呼んでいます。そうすると自然に、実装に依存しないコメントを書くことになり、コードの意図に対するコメントになります。

6 書いたほうがいいかどうか迷ったときには書く

さまざまな方針を考えて、コメントを書いておいたほうがいいかどうか迷うことがあります。

書いたコメントが不要だった場合にあとから消すことは簡単ですが、それに比べると、コメントがなくてわかりづらいことに気づいたときにあらためてコメントを書くことは難しいです。書くかどうか迷ったときには、書いておいたほうがいいでしょう。

コメントを書かずに済ます

ここまで、コメントを書くことについて説明しましたが、“コメントを書かない”という方針も考えてみましょう。いくつか、コメントの代わりになるものを挙げていきます。

シンボル名を適切に指定する

関数名や変数名、クラス名など、プログラム中で付ける名前によって、プログラムを読みやすくできます。コメントとして書こうとしていたことをシンボル名にしておけば、コメントが不要になるということはよくあります。

関数や定数の導入

次のように、数値が埋め込まれている場合には、定数を導入したほうがいいでしょう。

```
line = total / 80;// 一行は80文字
```

次のようになります。

```
LINE_WIDTH = 80;
line = total / LINE_WIDTH;
```

また、関数の中の処理が増えて、コードのまとまりが複数あるときは、そのまとまりを関数にしたほうが読みやすくなります。ただ、変数や関数の導入では、

```
LINE_WIDTH = 80;// 一行の文字数
```

のように、それぞれに対するコメントを付けておいたほうがいいので、コメントの総量は減りにくいように思います。

メタ情報(アノテーション)

Javaのアノテーションのように、メタ情報が付けられるしくみがある場合には、それを活用することでコメントの代替になります。また、対応しているツールがあれば、アノテーションによってコードの品質や実行時のデータの内容を検証できるようになります。

たとえばJavaでは、@Deprecatedというアノテーションが用意されているので、「非推奨」のようなコメントを書くよりもいいと思います。

```
@Deprecated
void calc() {
  ...
}
```

代わりにテストを書く

関数の具体的な値に対する結果を例示する場合には、テストを書くことでコメントの代わりになり得ます。テストとして書いておけば、品質の保証にもなります。たとえば次のように、どのような値を与えるとどのような値が返るかをコメントに書くことがあります。

```
/**
 メールアドレスの形式をチェック
 OK:aa@bb.cc
 NG:aa@bb
 */
boolean validateEmailAddress(String address)
```

これをテストとして表すと**リスト3**のようになります。

ただこの場合、ドキュメントになるようなテストと、動作確認のテスト、その初期化コードなどが入り交じることになります。動作がわかりにくい部分についてはコメントとしても書い

▼リスト3　テストとして表す

```
@Test
public void testValidateEmailAddress() {
  assertThat(validateEmailAddress("aa@bb.cc").isTrue();
  assertThat(validateEmailAddress("aa@bb")   .isFalse();
}
```

ておいたほうがいいと思います。

外部のツールを使う

昔、ソースコード管理ツールがあまり普及していなかったり高価であったりしたときには、ソースコードの変更はすべてコメントとして残すということが行われていました。

けれども、今はGitなどのツールが使いやすくなっていて、そのような目的でコメントを付ける必要はほとんどなくなりました。変更の理由についても、コミットのコメントに書いておけます。まとまった量の説明はWikiなどでドキュメント化するということもあります。コードの仕様については別のファイルで管理することも多いでしょう。

このように、コメントではなく外部のツールを活用することでより管理しやすくできます。ただ、コードと別のツールを同時にみるということは煩わしくもあり、また長い間メンテナンスしているうちに、ツールが使われなくなることもあります。

コメントの特性について

最後に、コメントの特性について書いていきます。

コードの種類によってコメントの必要性は変わる

アプリケーションのコードであれば、コードの書き方しだいでコメントが不要になっていくことも多いでしょう。コメントが必要な部分は、ライブラリなどで抽象化されて隠されていくからです。逆にライブラリには、コメントが必要な処理が集まっていくとも言えます。またライブラリの場合には関数の使い方や制約を詳しくコメントに書く必要も出てきます。

例として挙げたように、デバイスに近いコードでは説明が必要な不思議なコードも増えます。

このように、どのようなコメントが必要になるかは、コードの種類によって変わっていきます。そのため、コードの一部をみてコメントに対する全体の方針を決めるのは避けましょう。

なるべくコメントの量をルールにしない

コメントに対する全体の方針として代表的なものに、「コメントの量」が挙げられます。

たとえばある調査で、コードを読みやすくするコメントの分量はコード10行に対してコメント1行だという結果が出たとします。しかし、だからといって、コード10行に対してコメント1行書くということをルールにしないほうがいいでしょう。また、コメントを禁止するのも、コメントの量に関するルールの一種と言えます。

このようなルールは、コメントが適切に書けていないときに発生しやすいと思います。コード10行に対してコメント1行というルールは、コメントが少な過ぎるときに発生します。そのような場合、ルールによってコメントを書くように仕向けるわけです。

ただ先ほども説明したように、コメントの必要量はコードの性質によっても変わります。基準として設定するというのもありですが、あまり強制せず「ムダなコメントは書かない」「必要なコメントは書く」とした結果として、コード10行に対してコメント1行になったり、まったくコメントがなかったりとなるのが理想です。

◆ ◆ ◆

コードを読みやすくするための良いコメントの書き方について説明しました。この記事が、良いコメントを書いて、より良いプログラムを作るための助けになれば幸いです。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2回「物理乱数ハードウェアを作る」

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1回「コンピュータと乱数」(本誌2016年8月号)では、乱数の一般的性質と、シミュレーション用の疑似乱数について紹介しました。今回第2回は物理乱数の応用と、手頃な物理乱数ハードウェアの製作例について紹介します。

物理乱数の性質と応用

物理乱数とは、予測不能な物理現象を観測して得られる情報をもとに作られる乱数列のことです。広く利用されている物理現象としては、次のものがあります。

熱雑音

電気抵抗を持つ物質中の自由電子が熱によってランダムな(規則性のない)振動を起こす現象です。A/Dコンバータや高周波増幅器の入力雑音などの形で観測できるため、比較的容易に使うことができます。

ダイオードのなだれ降伏による雑音

逆方向接続されたダイオードに一定以上の電圧をかけると広帯域にわたってランダムな雑音が発生する現象です。オーディオ機器や無線機器の測定などに利用されています。

発振回路のタイミングのゆらぎ

外部と同期させていない発振回路の構成部品で発生する雑音などによって、発振回路の出力のタイミング(周波数や位相)がランダムに変化する現象です。デジタル回路で実装が容易なため、CPU内部の物理乱数発生器に広く使われています。

◆ ◆ ◆

これらの他にも、半導体レーザカオス[1]や量子力学の不確定性に基づくゆらぎを使ったものなどによって、さらなる高速化が進められています。これらの装置では、Gbps級の性能を目指しています。

本稿では、入手の容易な電子部品を使って製作できる、数十～数百kbpsの生成速度を持つ物理乱数ハードウェアを3つ紹介します[注1]。これらのハードウェアは、組み込み用のマイクロコントローラ(MCU)などを使い、物理現象による雑音源を集め簡単なソフトウェア処理を行って物理乱数を得るという構成を取っています。

物理乱数の課題

物理乱数はランダム性という点では優れていますが、実際に乱数として使うためには次に述べる問題を解決する必要があります。

問題その1

物理現象を電気信号として測定し得られた振幅(信号の大きさ)やタイミングの情報から0と1の2進数の乱数列に変換する際、できるだけ0と1の出現確率がそれぞれ0.5に近くなるように振幅やタイミングの分布を考えておかないと、結果に偏りが生じてしまいます[注2]。

注1) 今回紹介するハードウェアは、あくまで実験用であり、結果に対する一切の保証はありません。筆者および本誌は、運用に使った場合の結果について、一切の責任を負いません。

問題その2

物理現象は外部の要因で変化します。電子回路では接続している電源や信号線などから混入する雑音の影響を受けるため、悪意の第3者がいれば妨害電波を照射することなどで出力を意図的に偏らせてしまうことも可能です。これらを防いで外部の影響を受けにくくするためには、銅板などで覆ってシールドすることや安定した電源の確保など、物理的な攻撃を受けにくくする対策が必要です[注3]。

問題その3

物理現象で得られた乱数列は、そのままでは理想的な一様分布からほど遠いもので、その結果を情報セキュリティの用途にそのまま使うことは推奨できません。統計的な一様性を確保するために出力にハッシュ関数などを使ったり、環境の影響を取り除くため複数の生成装置のXORを取るなどの後処理が必要です。MCUはこのような演算を行うのに適しています。

ランダム性と情報セキュリティ

現在のOSの多くは、予測が非常に困難な乱数列を発生するしくみを備えています。これらのしくみの多くは、コンピュータの中で発生する各種イベントに付随するゆらぎ[注4]を利用しています。これらのゆらぎの情報を物理乱数列として扱い、それをseed(連載第1回参照)として暗号学的強度の高い疑似乱数生成[注5][2]を行うことで、予測が困難な乱数列を大量かつ高速に生成できます。生成結果はパスワードや暗号鍵の生成など、情報セキュリティを高める目的に広く使われています。

しかし、コンピュータの中で起こる現象はある程度は予測可能であるため、そこから得られるゆらぎの量は実用上十分とは言えません。外部からの悪意をもった操作、あるいはソフトウェアの動作の影響などでゆらぎの程度が少なくなってしまうと、ランダム性の高い疑似乱数を作り続けることができなくなります。擬似乱数の種(seed)としての情報は、どんな状態でも十分な量を得られるようにしておく必要があります。

今回紹介する物理乱数ハードウェアは、コンピュータの他の部分とは独立してランダムな情報を継続的に生成することで、それを利用するOSのセキュリティを結果としてより高めることを主たる目的にしています。また、設計と製作の過程がわかっている実装を使うことで、悪意の第3者に乱数列を歪められることを防ぎ、セキュリティ上の問題を防ぐねらいもあります。

今回紹介する物理乱数ハードウェアの概要

今回紹介する物理乱数ハードウェアは次の3つです。関連情報はすべて公開されています。

avrhwrng[3]

筆者の製作したArduinoボードを使ったものです。80Kbps程度の出力が得られます。ハードウェアはCC-BYライセンス、ソフトウェアはMITライセンスです。

NeuG(ノイジー)[4][5]

飛石技術(Flying Stone Technology、FST)の新部裕さんによるものです。STMicroelectronics社(以下STM社)のSTM32F103 MCU(ARM Cortex-M3アーキテクチャ)を使用し、FSTの小型USBデバイスFST-01や、STM社の評価ボードNucleoで動きます。出力は640Kbps程度です。

注2) 今回も前回同様、一様分布あるいは離散一様分布の乱数を得ることを想定しています。

注3) これらのほかにも、接続用の配線が物理的に抜けたりしないように考慮されているか、第3者が触れたりできないような場所にあるか、機器がすり替えられないか、などの一般的なセキュリティの原則は当然守る必要があります。

注4) 例えばマウスやキーボード、タブレットなどの人間による入出力機器の操作や、ネットワーク上のパケットの到着間隔、ハードディスクのアクセス時間などには、一定のゆらぎが観測できます。

注5) 参考文献[2]には、FreeBSD 11.0以降で採用された暗号学的強度の高い疑似乱数生成アルゴリズムFortunaが詳しく解説されています。

ソフトウェアはGPLv3ライセンスです。

rtl_entropy[6]

ソフトウェアラジオR820Tで採取した雑音を処理するためのソフトウェアの名称です。2.4Mbps程度の出力が得られます。ライセンスはGPLv3です。

Arduinoボードを使った物理乱数生成装置avrhwrng

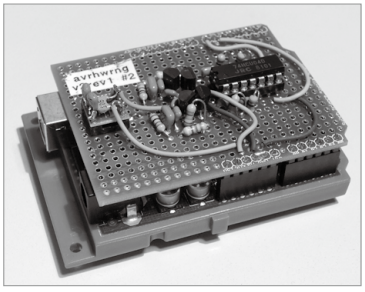
筆者の設計開発したavrhwrngは最初にArduino Duemilanove上に実装し[7]、2015年にはArduino UNO R3でも動作することを検証しました。2016年からはFreeBSDシステムに接続して運用実験を続けています(次回第3回で紹介予定)。本体の写真を**写真1**に示します。

avrhwrngは、トランジスタの一部を逆方向接続したダイオードとして使い、そのなだれ降伏による雑音を増幅して5V系のデジタル信号にする雑音信号生成器を独立して2つ備えています。雑音信号生成器部の回路図を**図1**に示します。左側の2つのトランジスタ(1つはベースとエミッタの間を逆方向接続したダイオードとして使っています)が雑音信号を生成し、右側の増幅器(74HCU04の中のインバータを3つ直列にしたもの)でデジタル信号として増幅を行います。

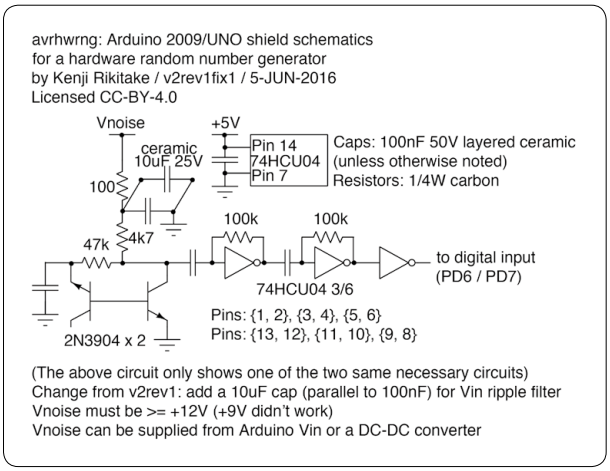
avrhwrngのトランジスタ部分の出力を**図2**に示します。この出力を増幅してデジタル信号とし、2系統分まとめた結果を**図3**に示します。2つの系統の間に相関がないことがわかります。これらの雑音信号をArduinoボード上のMCUであるATmega328Pを使ってサンプリングとフィルタ処理を行い出力します。

avrhwrngで行っているダイオードの逆方向接続による雑音の生成には12V程度の直流電源が必要なため、外部からACアダプタでArduinoボードの動作電源を兼ねて供給する必要がありました。現在は5Vから12Vへ変換するためのDC-DCコンバータを備えており、USB電源のみで動作させることができます。このDC-DCコンバータ部分の回路を**図4**に示します。**写真1**の左側にある8ピンDIPパッケー

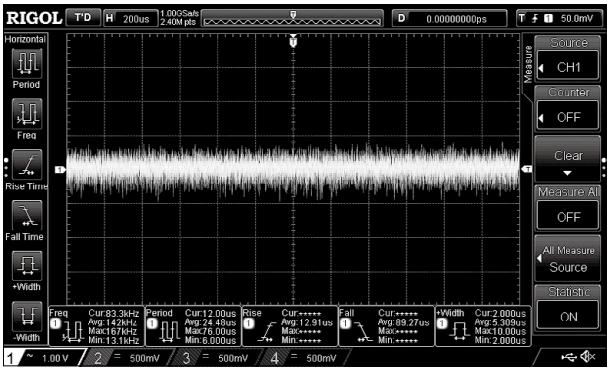
▼写真1　avrhwrngの全体写真



▼図1　avrhwrngの雑音信号生成器部分の回路図



▼図2　avrhwrngのトランジスタ部から出力されたアナログの雑音信号

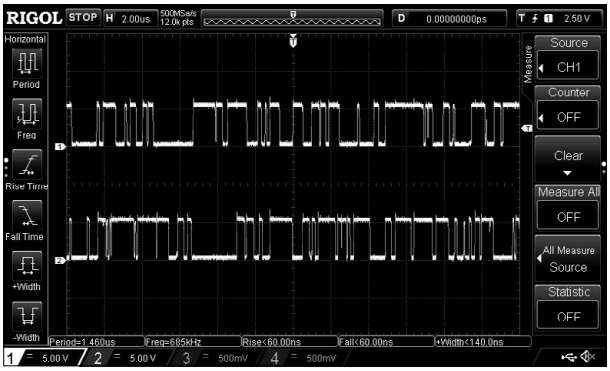


ジがこのDC-DCコンバータの部分です。

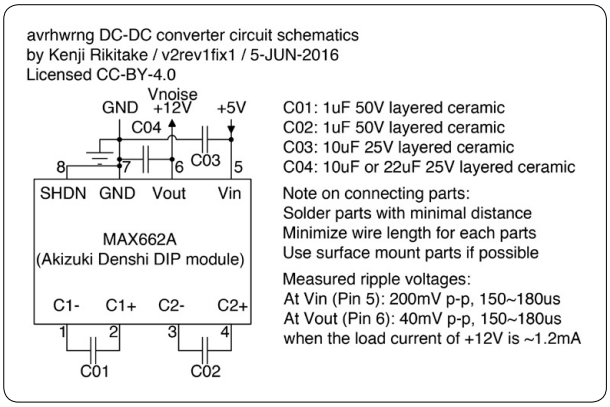
avrhwrngのATmega328Pでは、それぞれの雑音源からの出力を2.875 μ秒の周期で動く内部タイマに同期させてサンプリングし、2ビット得られたところでフォン・ノイマン・フィルタ(**図5**[18][19])にかけて連続した2ビットが同じ状態になることを防ぎます。そしてこの結果から得られた2バイト(16ビット)をバイト毎にXORして、結果を出力します(**図6**)。これによって毎秒約10KBの物理乱数を得ることができます。出力はArduinoボードのUSBインターフェースを介して送ります。ホスト側からはランダムなバイト列を送るUSBシリアルデバイス(速度は115,200bps)に見えます。

ATmega328Pのソフトウェア開発は、8bit AVR MCUの代表的な開発環境であるavr-gccとavr-libc[8]を使って、直接C言語のみで行っています。Arduino IDEのライブラリを使用しなかったのは、

▼図3　avrhwrngの雑音信号生成部2系統が最終的に出力するデジタル信号を並べたもの



▼図4　avrhwrngの5Vから12Vへ変換するDC-DCコンバータ部分の回路図



▼図5　フォン・ノイマン・フィルタの動作原理

0と1の2進数で示されるランダムな信号に対し、2回サンプリングを行い、両方の値が同じ場合は結果を捨てて再試行し、違う場合は有効な値として1ビットの有効な結果とする。

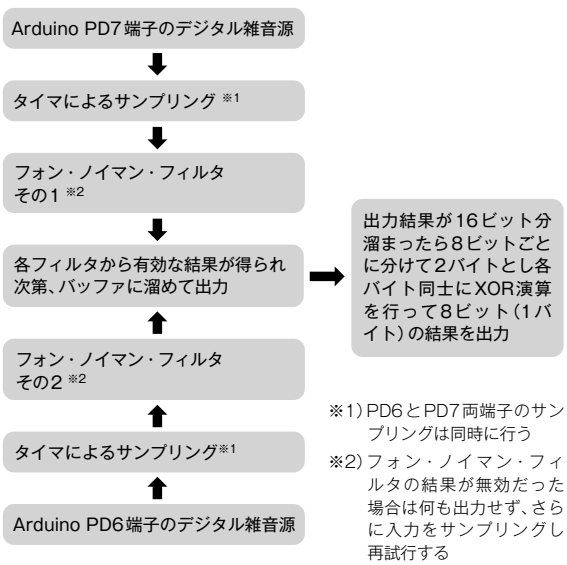
この方法を使えば、サンプリング元の0と1の発生確率が同じでなくても、有効な結果については、0と1の出現確率を同じにすることができる。1951年に提案されたこのアルゴリズムを「フォン・ノイマン・フィルタ」(参考文献[18][19])という。

ただしこの方法では、意図的に1回目の値と2回目の値を違うものに変えた場合には、偏りをなくす効果はまったくなくなる。つまり、1回目と2回目の結果の間の相関が一切存在しない(独立試行である)ことが前提であることに注意が必要。

1回目の値	2回目の値	得られる結果
0	0	無効(再試行)
0	1	0
1	0	1
1	1	無効(再試行)

(※注：上の表では結果には1回目の値を取っているが、2回目の値を取っても確率的性質は変わらない)

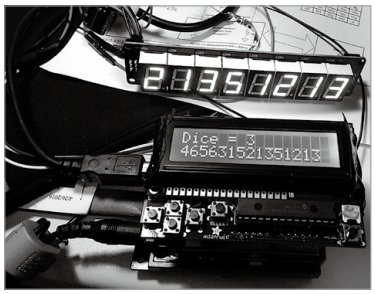
▼図6　avrhwrngの処理の流れ



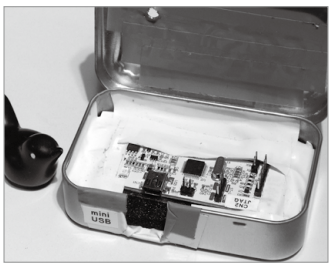
極力高速化を目指したからです。

avrhwrngはホストなしでも独立して動作できるため、他のArduinoのハードウェアと組み合わせることも可能です。筆者は外部電源供給だけで動く1〜6までのサイコロ同様の数字を物理乱数を元に表示する装置avrdice[9]（**写真2**）[注6]を開発しました。この装置ではLCDとLEDで同じ内容を重複して表示するようにし、サイコロの値の変化がわかるようにしています。

▼ **写真2　avrhwrngを応用した電子サイコロ装置avrdice**



▼ **写真3　STM32 Nucleoのデバッガ部を使ったNeuGの実装例[20]**



STM32F103による物理乱数生成装置NeuG

飛石技術が設計販売しているSTM社のSTM32F103 MCUを使ったハードウェアFST-01[10]は、GNU Privacy Guard（GnuPG）の秘密鍵保持と暗号計算をするための暗号トークン用として開発され、そのソフトウェアとしてGnuk[11]が提供されています。NeuGはGnukの中の物理乱数生成部分を独立して実装したもので、概念図[12]によればSTM32F103に内蔵されているA/Dコンバータを使い、基準電圧や外部温度、未使用入力端子の電圧などから得られるゆらぎの情報を内蔵のCRC32計算ユニットでハッシュ処理し、さらにその結果をSHA256を使ってより一様性を高めた情報を出力します。NeuGはUSBのCDC/ACMデバイスであり、ホストからはモデムとして見えるため、avrhwrng同様にシリアルデバイスとして扱うことができます。

NeuGの動くハードウェアとしてはFST-01に限らず、STM社の提供するSTM32 Nucleo F103RB[注7]など市販のボードが利用可能です[13]。また、STM32 Nucleoのデバッガ部分（ST-Link/V2-1）はSTM32F103で動作しているため、この部分のファームウェアを書き直すことでNeuGを動かすことができます（**写真3**[20]）。筆者は数台この方法でNeuGを動かしてFreeBSDへの物理乱数供給源として使っていますが、数週間以上にわたって安定して動作した実績があります。

なお、NeuGとGnukはGPLv3ライセンスで公開され、USBのベンダーID/プロダクトID（VID/PID）は特定非営利活動法人フリーソフトウェアイニシアティブ（FSIJ）[14]が管理しています。FSIJのUSB VID/PIDは個人の利用、あるいは実験用の再配布を伴わない利用だけが既定の条件として許可されているため、その他の商用利用には別の方法でUSB VID/PIDを取得することが必要です。

rtl_entropyとソフトウェアラジオを使った物理乱数生成装置

無線信号を直接デジタル処理して受信するソフトウェアラジオは技術の普及に伴い各種製品が入手できるようになりました。R820Tというチューナーと、RealTekのRTL2832Uという解読器を使ってUSB2.0で接続できるUSBドングル製品が比較的安価にて購入できます[注8]。この受信機はFM放送を聞くためなどのラジオとしても活用できますが、無線受信部の熱雑音を利用

注6） 本稿執筆時ではまだ予定ですが、2016年8月6日/7日の両日に開かれるMaker Faire Tokyo 2016の「理科教育研究フォーラム」にて展示される予定です。
注7） 本稿執筆時（2016年7月上旬）には秋月電子通商（通販コードM-07724）、スイッチサイエンス（PLU# 1618）で入手可能です。
注8） 本稿執筆時（2016年7月上旬）にはAitendoにて「ワンセグRX DVB-T+DAB+FMR820T高性能受信機［R820T］」という名称で入手可能です。

した物理乱数発生器としても使うことが可能です。

rtl_entropyという生成ソフトウェアでは、ソフトウェアラジオ機器を接続するためのライブラリrtl-sdr[15]を使い、内部でさらにフォン・ノイマン・フィルタ、FIPS 140-1に基づく検定[16]、そしてKaminsky debiasing[17]のテストを通ったもののみを出力することで、物理乱数としての品質を高めています。筆者の実験では、毎秒300KB相当の出力を得ることができました。

なお、R820Tを使ったソフトウェアラジオを乱数生成器として使う際は、アンテナ端子にダミーロード注9を接続しないと、十分なレベルの雑音を得ることができませんでした(**写真4**)。また、この機器は電波の受信機として動作していますので、周囲の電磁雑音などの影響を受ける可能性があります。得られる乱数列の量も多いため、ホスト側のCPUの負荷も大きくなります。筆者は数日間の動作は検証していますが、それ以上に安定した物理乱数を生成し続けられるかどうかについては長期動作による検証が必要と考えます。

▼ 写真4　ソフトウェアラジオを搭載したUSBドングルにダミーロードをつけたもの

次回予告

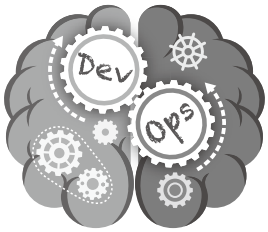
今回は物理乱数発生器の原理と、そのハードウェアの製作例について紹介しました。次回は今回紹介した物理乱数ハードウェアをOSで活用するために必要な知識であるOSの乱数生成の機能としくみ、そしてそれらの利用の仕方について紹介します。SD

参考文献

[1] 内田 淳史、「光のランダム現象を応用した超高速物理乱数生成器の研究開発の最新動向」、レーザー研究、Vol. 39、No. 7、pp. 508-514、https://www.jstage.jst.go.jp/article/lsj/39/7/39_508/_pdf
[2] Niels Ferguson, Bruce Schneier, Tadayoshi Kohno, "Cryptography Engineering: Design Principles and Practical Applications", Wiley, 2010,ISBN-13: 978-0-470-47424-2
[3] https://github.com/jj1bdx/avrhwrng
[4] http://git.gniibe.org/gitweb/?p=gnuk/neug.git;a=summary
[5] 筆者によるNeuGのクローンレポジトリ: https://github.com/jj1bdx/neug
[6] https://github.com/pwarren/rtl-entropy
[7] 船田 巧、「乱数生成シールド」、http://makezine.jp/blog/2009/04/random_number.html
[8] http://www.nongnu.org/avr-libc/
[9] https://github.com/jj1bdx/avrdice
[10] http://no-passwd.net/fst-01-gnuk-handbook/fst-01-intro.html
[11] http://git.gniibe.org/gitweb/?a=project_list;pf=gnuk
[12] http://www.gniibe.org/memo/development/gnuk/rng/neug.html
[13] 新部 裕、「NeuG USB Device を STM32 Nucleo F103でみんなで作ろう」、http://www.gniibe.org/memo/development/gnuk/hardware/stm32-nucleo-f103.html
[14] http://www.fsij.org/
[15] http://sdr.osmocom.org/trac/wiki/rtl-sdr
[16] http://csrc.nist.gov/publications/fips/fips140-1/fips1401.pdf(Section 4.11.1 と 4.11.2)(ただし現在は規格としての意味は持たない)
[17] https://dankaminsky.com/2012/08/15/dakarand/
[18] John von Neumann, "Various Techniques Used in Connection with Random Digits", Notes by G E Forsythe, National Bureau of Standards Applied Math Series, 12 (1951), pp. 36-38. (Reprinted in von Neumann's Collected Works, 5 (1963), Pergamon Press, pp. 768-770.) PDFファイルのURL:https://dornsifecms.usc.edu/assets/sites/520/docs/VonNeumann-ams12p36-38.pdf
[19] D. Eastlake 3rd, J. Schiller, S. Crocker, "Randomness Requirements for Security" (RFC4086), June 2005, https://www.rfc-editor.org/rfc/rfc4086.txt, Section 4.2 "Using Transition Mappings to De-Skew".
[20] 力武 健次、「STM32 Nucleo ボードのST Dongle部にプログラムを新たに書き込むには」、http://qiita.com/jj1bdx/items/2f32d8c8649d7825a9a3

注9) 伝送路のインピーダンスを合わせるための電気抵抗。R820Tの場合は75 Ωのものが必要です。実際には1/4Wのカーボン抵抗で十分でしょう。

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3回 インフラ構成管理入門

連載第3回目は、サーバなどインフラの構成管理を取り上げます。インフラの構成管理はそれ自体がすなわちDevOpsと呼ばれることもあり、開発者（Dev）と運用者（Ops）が密接に協力して開発を進めることで、価値を生み出しやすい部分です。

なぜインフラ構成管理をするのか

アプリが複雑化する今日では、インフラが1台のサーバのみという構成はほとんど見当たりません。場合によっては何十台何百台ものサーバを、適切に設定して改善していく必要があります。今回はサーバなどのインフラの構成管理について、その必要性や概念をあらためて振り返り、導入の流れなどについて説明します。

Infrastructure as Code

プロビジョニングツール[注1]などを使った構成管理は、インフラ管理を簡易化・自動化することで管理者の負担やリスクを少なくし、継続的な改善が可能な状態にすることが目的です。

その中に、Infrastructure as Codeというインフラの状態や更新をコード化して管理を行おうという考え方があります。ChefやAnsibleなどをはじめとする多くのプロビジョニングツールはこの考えをもとにして作られており、すべての要件（Configuration）を人間にもわかりやすいコードにして管理をします。**リスト1**はChefにおいて「recipe」と呼ばれる、サーバの要件を記述した簡単な例です。

これらをサーバ群に適用するという流れで、インフラに変更を加えていきます。「動く手順書」や「動く仕様書」と言うとイメージがつきやすいと思います。

インフラをコード化する利点

サーバの設定や状態をコードで記述する最大の利点は、GitHubなどを使って容易にバージョン管理と共有ができるようになる点です。

サーバの設定を手動で行う体制では、その作業を行うのは一部のサーバにログインできる権限を持った人間に限られるため、管理が属人的になってしまいます。またインフラがブラックボックス化され、管理者が異動や退職などでいなくなった場合に、中がどうなっているか誰もわからないということになりかねません。コードによる構成管理をしておけば、誰もがその「動く仕様書」を読んでサーバがどういう状態になっているかを知ることができます。

▼リスト1 Chefのrecipe例

```
# パッケージマネージャでvimをインストールする
package "vim" do
  action :install
end
# httpdデーモンを起動・自動起動設定する
service "httpd" do
  action [:start, :enabled]
end
```

注1）プロビジョニングとは、サーバなどのインフラを、すぐに提供できるように設定しておくこと。構成管理の1つ。

またアプリのコードと同様に、プルリクエストによるレビューやCIツールによるテストの自動化が可能です。変更をアプリエンジニアが書いて、それをほかのエンジニアがレビューしてテストし、本番環境への適用まで行うというフローが可能になるのです。

またコード化することで、まったく同じインフラ構成を簡単に再現できます。これは、本番とほぼ同一の構成のアプリ開発環境やステージング環境も容易に構築できるということです。これでテストが容易になり、設定の差分を十分検証したうえで本番の構成に適用させられます。

手作業が必要なくなり、ミスが起こりづらくなるのも大きな利点です。管理対象が多くなればなるほどその手順は煩雑になり、人間が作った手順書に沿って人間が実行するとなると、どんどんミスのリスクが上がっていきます。自動化が行われることで継続的なインフラの開発が可能になるのです。

以上の理由から、構成管理を行うことは今や必須であると言えます。「うちのシステムは小規模だから構成管理は不要なのでは？」と思う方もいるかもしれませんが、構成管理をせずに属人化しているとスケールすることも難しくなります。近年では、小さく導入して始められるツールも登場してきていますので、システムが小規模な場合も他人事ではありません。

構成管理をしていない状態をアプリの開発に例えると、バージョン管理ツールを使わずに、共有サーバなどにある1つのソースコードをみんなで触るような状態です。今となっては怖くて開発できたものではないというのは、よくわかるかと思います。

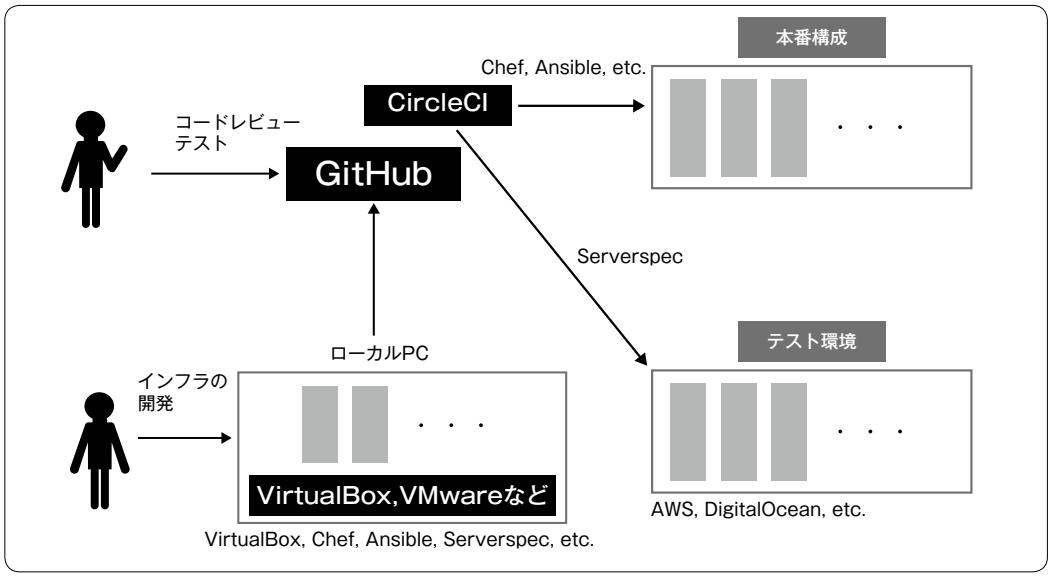
構成管理のツール

インフラの構成管理にはさまざまなツールを利用します。数も種類も非常に多いため、サーバのプロビジョニングに注目して主要なツールの位置づけや利点を整理します。

インフラの開発を補助するツール

インフラをコード化すると、「インフラを開発する」というとらえ方をすることになります(**図1**)。開発するためにはVirtualBoxやVMwareなど、仮想マシンの管理ができるソフトウェアを利用することが多いです。またVagrantは、それらのプロバイダ(仮想化ソフト)のマシンを

▼図1　インフラ開発の流れの例



CUIで管理するためのソフトウェアです。Vagrantfileに仮想マシンの状態を書くことで、仮想マシンの利用がより便利になります。

アプリにテストコードを書くように、インフラにもテストが必要です。テストフレームワークで有名なものにServerspecがあります。立ち上がっているべきデーモンや、listenしているポート番号、設定ファイルの内容など、インフラの望む状態を記述してテストを実行できます。ServerspecはRSpecというRubyのテストツールをもとに書かれており、記述の方法は**リスト2**のように非常に直感的です。ChefやAnsibleとの連携も簡単です。

またGitHubや、CircleCIのようなCIツールもよく利用されます。構成管理の際にはこれらの機能も享受して開発を進めるのが便利です。

サーバのプロビジョニング

構成管理のメインとなるのがサーバのプロビジョニングです。これもさまざまなツールがありますが、中でもChefとAnsibleが有名です。

Chefは Infrastructure as Codeを実現するための代表的なツールです。エージェントソフトウェアであるchef-clientを管理対象のサーバに入れ、定期的にそれがChef Serverにアクセスして自らの状態を収束させる、という動作をします(**図2**)。インフラを記述するコードは、おもにRubyの文法で書けるrecipeとして記述され、cookbookという単位で再利用や配布が行われます。中央のChef Serverに管理を一元化する構成を基本としていて、大規模なインフラで威力を発揮します。

▼リスト2　Serverspecの例

```
# OSがRHELの場合、httpdデーモンを自動起動設定、起動状態にする
describe service('httpd'), :if => os[:family] == 'redhat' do
  it { should be_enabled }
  it { should be_running }
end
# 80番ポートが開いている
describe port(80) do
  it { should be_listening }
end
```

Ansibleも、できることではChefと似ていますが、大きな違いはchef-clientのようなエージェントをインストールしたり、Chef Serverのような中央を作る必要がない点です。SSHでの接続さえ可能ならサーバのプロビジョニングができるため、小規模から大規模までさまざまなインフラに利用できます。コードはYAML形式でplaybookに記述し、roleという単位で再利用します。

ChefもAnsibleもホストごとにパラメータを変えられるので、特別に一部サーバを変更したり、検証環境を構築することも容易です。

これらのツールの特徴は**冪等性**(べきとうせい)です。これは「同じことを何度しても結果は同じになる」という性質のことです。たとえばPostgreSQLのデーモンを立ち上げておくという命令を書いたときに、すでにデーモンが立ち上がっていれば何もしませんし、立ち上がっていなければ立ち上げる動作をします。変更内容を書くというよりも、インフラの望ましい状態(要件)を書き、ツールがサーバ群をその状態に収束させる、というイメージのほうが適しているでしょう。ここが、シェルスクリプトなどで設定作業を自動化する場合と大きく異なる点です。

ここで紹介した以外にもさまざまなツールがありますが、注意すべきことは「構成管理ツールはインフラをブラックボックス化してくれるツールではない」ということです。わからないことを代わりにやってくれるわけではありません。それぞれのコードで、だいたいどういう処理が裏で実行されているのかを理解しておくのが理想です。

またプロビジョニングツールは、JenkinsやCapistranoなどのデプロイを自動化できるツールとも異なります。Dockerによるコンテナ仮想化や、Terraformによるクラウド全体の構成管理ツールともやや立ち位置が違います。ツールの得意分野を理解し、適宜組み合

わせてしくみを作る必要があります。

使いやすい構成管理を実現するまで

構成管理のしくみを新しく作ったり変えたりする場合は、新しいツールの導入をほぼ確実に行うことになります。このとき、ツールを導入する目的を明確化し、複数のことを一度にやらず、1つずつ変えていくのが大切です。

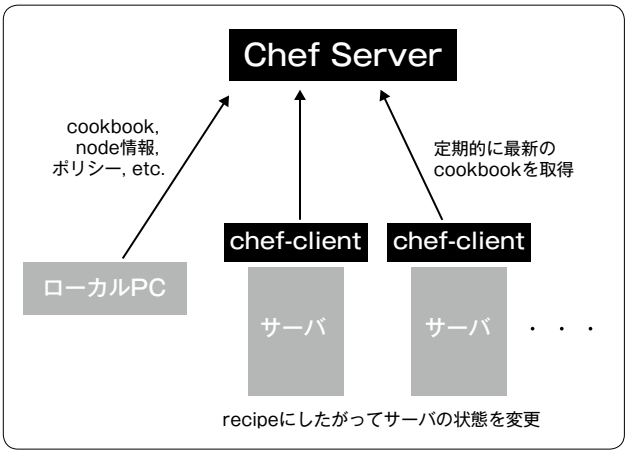
これから新たに構成管理のしくみを導入する場合は、まずチームメンバーの理解をしっかりと得る必要があります。基本的に、構成管理されたインフラは手動で変更を行わないので、作業のフローが大きく変わります。構成管理はチームにとって確実に必要なものといえど、ツールを1つ選定したり、実際にインフラをツールの管理下に移行するまでに、多くの手間と時間がかかります。チームを巻き込んでおかないと、手作業を行うのと同様に“孤独なインフラ管理者”が現れ、ツールの利点が活きなくなります。

筆者が以前いた職場では、短期間しかいないサーバ管理者によってアドホックに変更が行われていたので、サーバ設定の全貌がまったくわからず、ビクビクしながら改善を行っていました。やっと全貌を理解できた自分も近々別の職場に移動することが決まっていたため、このままではまずいと思い、インフラ状態の共有を第一の目的としてChefを導入しました。

現状稼働しているサーバ群は、少しずつ切り出しながら構成管理ツールで作りなおしていくのがベストです。このとき、稼働し始めてしばらくしてから構成管理を始めるという選択が必要な場合もありますが、再現可能なインフラが実現できていない状態を脱するために、なるべく早くにこれを行う必要があります。

バージョン管理の単位によって再利用のしかたが変わり、管理のしやすさも異なってきます。

▼図2　Chef Serverを使った構成管理の例



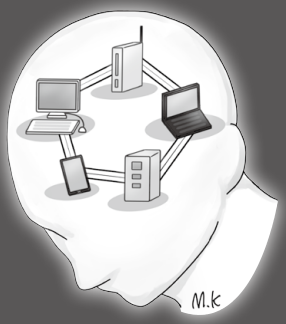
Chefではcookbook、Ansibleではrole単位でバージョン管理をするのがベストプラクティスとされています。オープンソースとして公開されているcookbookやroleなどもあるので、ぜひ使っていきましょう。

導入時にはDigitalOceanやAWSなどのクラウド環境を使って、自動テスト(CI)の環境も同時にセットアップすると良いでしょう。なるべく本番環境とまったく同じ環境を作って、検証に利用するのをお勧めします。ここまでしっかりと構成管理を導入すれば、基本的にアプリの開発と同様の流れで、インフラの開発が行えることでしょう。

◆ ◆ ◆

今回はインフラの構成管理とプロビジョニングのしくみの基礎についてまとめました。構成管理は導入して終わりではなく、随時そのしくみも含めて改善していくものです。インフラを、ビジネスの現場に近いアプリエンジニアにも触れやすい状態にすることで、ビジネスで必要となるインフラを構築することが容易になります。

最近ではAnsibleを始めとして手軽に構成管理が行えるツールが増えてきており、日本語の情報も集まってきています。個人でサーバの構成管理をセットアップしたり、すでにプロジェクトで利用されている構成管理ツールのコードを読むと、学習の助けになるでしょう。SD



仮想化の知識、再点検しませんか?
使って考える仮想化技術

本連載は「仮想化を使う中で問題の解決を行いつつ、残される課題を整理する」ことをテーマに、小さな仮想化環境の構築・運用からはじめてそのしくみを学び、現実的なネットワーク環境への実践、そして問題点・課題を考えます。仮想化環境を扱うエンジニアに必要な知識を身につけてください。

Author
笠野 英松(Mat Kasano)
有限会社 ネットワーク・メンター
URL http://www.network-mentor.com/indexj.html

第4回 仮想環境の構築(その3)~仮想マシンの操作と状態確認

これまでの連載で、次の準備までできました。

・ホストシステム(デフォルト設定のCentOS 6.7上でKVMが動作)
・仮想マシン「kvm1」(Windows 7をOSとしたゲストシステム)
・仮想マシン「kvm2」(FreeBSD 10.3をOSとしたゲストシステム)

今回は、仮想マシンkvm1、kvm2の起動/シャットダウン操作および、ホストと仮想マシンのネットワークの設定状況を確認します。

仮想マシンの起動とOSのシャットダウン

作成した仮想マシンを起動/シャットダウンする方法をまとめておきます。シャットダウンの方法は2通りあり、仮想マシンマネージャー画面で「シャットダウン」するか、仮想マシンのゲスト内で「シャットダウン」します。

仮想マシンの起動

仮想マシンの起動はデフォルトでは手動で行います[注1]。**図1**のように仮想マシンマネージャー画面の「仮想マシンの電源を入れる」アイコン(右向き三角)か、または仮想マシンのアイコン上でマウスを右クリック→[実行]で行います。

また、仮想マシンを起動したままホスト(CentOS 6.7)を停止させると、仮想マシンは起動状態のまま保持され、ホスト再起動時にその状態が再現されます[注2]。

仮想マシンマネージャーでの「シャットダウン」

仮想マシンマネージャー画面でシャットダウンを行う場合は、**図2**のように「仮想マシンをシャットダウン」アイコンからか、または仮想マシンのアイコン上でのマウス右クリックからの[シャットダウン]で行います。

ゲストOSでの「シャットダウン」

仮想マシンのゲストOS内でシャットダウンを行う場合、Windows 7では通常の[スタート]→[シャットダウン]で行います。

FreeBSD 10.3(CUI)では、rootでログイン後のコマンドプロンプト状態(**図3**)から次のコマンドで電源停止まで行います。

```
# shutdown -p now↵
```

なお、FreeBSDではLinuxでのように、

```
# shutdown -h now↵
```

を行うと、キー入力reboot待ち(halt)状態にな

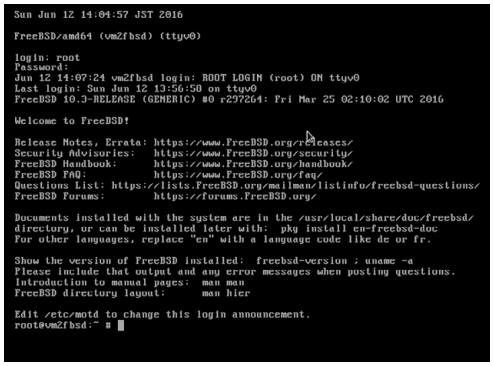
注1) ホストシステムの起動時に自動的にゲストシステムも起動させる自動起動は次回以降に解説。

注2) これについての詳細も次回以降。

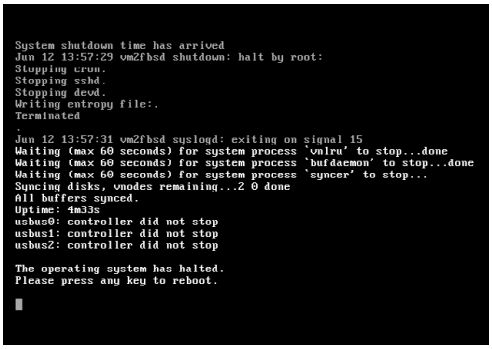
▼図1　仮想マシンの起動方法

▼図2　仮想マシンのシャットダウン方法

▼図3　FreeBSD(kvm2)にrootでログインした画面



▼図4　halt状態のFreeBSD



▼図5　強制終了の確認メッセージ

ります(**図4**)。ここで何かキーを入力するとreboot(再起動)します。この状態のとき、仮想マシンマネージャーの画面(および、virsh listコマンド)では仮想マシンの状態は「実行中」となります。仮想マシンマネージャーのメニューアイコンの[強制的に電源OFF]で終了できますが、このとき**図5**のように確認メッセージが表示されます。このまま「はい」と答えてもいいですが、「-p」オプションでの通常終了をお勧めします。

ゲストOSインストール後の確認

OSインストール後、物理ホストおよび各仮想マシンのそれぞれのシステムで(ログオン／ログインして)、仮想マシンのネットワーク設定(アドレス設定)を確認する必要があります。本連載でのデフォルト処理(前回行った仮想マシン作成とOSインストール)のネットワーク接続はNAT/DHCP接続となります。つまり、ゲストシステムでは自動的にIPアドレスが割り当てられていることになります[注3]。

DHCPの割り当ては物理ホスト内の仮想ネットワーク設定(**リスト1**)で、192.168.122.2～192.168.122.254までの範囲となっています。なお、192.168.122.1は物理ホストの仮想インターフェースに割り当てられていて、192.168.122.0はネットワークアドレスに、192.168.122.255はブロードキャストアドレスに、それぞれ予約されています。

注3) dnsmasq/DHCPのアドレスリース時間は、/etc/dnsmasq.confの「dhcp-range」またはdnsmasqコマンドの「--dhcp-range」オプションの、最後のパラメータで指定するが、両方とも無指定の場合は、デフォルト1時間(man 8 dnsmasq参照)となる。なお、libvirtのネットワーク設定XML「/etc/libvirt/qemu/networks/default.xml」にはリース時間のパラメータはない。

▼リスト1　仮想ネットワークの設定とホストネットワークインターフェースの設定

```
①仮想ネットワークの設定(/etc/libvirt/qemu/networks/default.xml)
<network>
  <name>default</name>                                          設定名
  <uuid>e0a58970-3ef3-4b9d-af36-14b2a6820962</uuid>             識別子
  <bridge name="virbr0" />                                      仮想ブリッジ名
  <mac address='52:54:00:7B:8F:80'/>                            仮想I/F MACアドレス
  <forward/>   仮想LANから物理LANへの転送設定(デフォルト:NATモード)
  <ip address="192.168.122.1" netmask="255.255.255.0">          ホスト仮想IPアドレス
    <dhcp>                                                      DHCP設定
      <range start="192.168.122.2" end="192.168.122.254" />     192.168.122.2～254割り当て
    </dhcp>
  </ip>
</network>
```

```
②ホストネットワークインターフェースの設定(ifconfigの出力抜粋)
実インターフェース
eth0      Link encap:Ethernet  HWaddr 00:17:42:65:C4:A9                          MACアドレス
          inet addr:192.168.0.111  Bcast:192.168.0.255  Mask:255.255.255.0      IPアドレス
仮想インターフェース
virbr0    Link encap:Ethernet  HWaddr 52:54:00:7B:8F:80                         MACアドレス
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0    IPアドレス
```

▼図6　新しい仮想マシンを作るステップ4

```
[root@vm1 ~]# ps ax | grep -v grep | grep dnsmasq↵     dnsmasqプロセスの確認
 2768 ?        S      0:00 /usr/sbin/dnsmasq --strict-order --pid-file=/var/run/libvirt/➘
network/default.pid --conf-file= --except-interface lo --bind-interfaces --listen-address ➘
192.168.122.1 --dhcp-range 192.168.122.2,192.168.122.254 --dhcp-leasefile=/var/lib/libvirt/➘
dnsmasq/default.leases --dhcp-lease-max=253 --dhcp-no-override --dhcp-hostsfile=/var/lib/➘
libvirt/dnsmasq/default.hostsfile --addn-hosts=/var/lib/libvirt/dnsmasq/default.addnhosts

[root@vm1 ~]# more /var/lib/libvirt/dnsmasq/default.leases↵
1465647904 52:54:00:21:f2:d8 192.168.122.194 user1-PC 01:52:54:00:21:f2:d8
1465647886 52:54:00:4c:a7:44 192.168.122.131 vm2fbsd 01:52:54:00:4c:a7:44
 ↑リース満了日時(1970年1月1日00:00:00からの経過秒数)、MACアドレス、IPアドレス、システム名、
  クライアントID(ハードウェアタイプー01=ethernet:MACアドレス)

[root@vm1 ~]# virsh dumpxml kvm1|grep "mac  kvm1のMACアドレス
      <mac address='52:54:00:21:f2:d8'/>
[root@vm1 ~]# virsh dumpxml kvm2|grep "mac  kvm2のMACアドレス
      <mac address='52:54:00:4c:a7:44'/>
```

物理ホスト側での確認

物理ホスト側(CentOS)からの仮想マシン(Windows、FreeBSD)への自動アドレス割り当ての実際を確認します。

図6のように、dnsmasqプロセスが起動していてDHCP自動IPアドレス割り当てを行っていることがわかります。なお、割り当てたIPアドレスとMACアドレスの対応はリースファイル(/var/lib/libvirt/dnsmasq/default.leases)に格納されていて(リース時間――デフォルト1時間――を過ぎると再割り当てまで消える)、MACアドレスは仮想マシン設定ファイル(/etc/libvirt/qemu/*.xml)の内容を見るコマンド(virsh dumpxml)でも確認できます。MACアドレスは仮想マシン作成時に指定(選択)したものです。

図7のコマンド実行で、パケットフィルタiptablesのINPUT(着信)チェインではDHCPサーバ(ホストのbootpsポート)宛着信の、FORWARD(転送)チェインでは仮想マシンからの発信およびその応答の、それぞれのパケッ

▼図7　DHCPサーバ宛着信と仮想マシン側発のパケット許可

```
[root@vm1 ~]# iptables --list↵
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     udp  --  anywhere             anywhere            udp dpt:domain
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:domain
ACCEPT     udp  --  anywhere             anywhere            udp dpt:bootps
 ↑DHCP/udpサーバ宛着信許可
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:bootps
 ↑DHCP/tcpサーバ宛着信許可
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
ACCEPT     icmp --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:ssh
REJECT     all  --  anywhere             anywhere            reject-with icmp-host-prohibited

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             192.168.122.0/24    state RELATED,ESTABLISHED
 ↑仮想マシン宛応答転送許可
ACCEPT     all  --  192.168.122.0/24     anywhere      仮想マシン発転送許可
ACCEPT     all  --  anywhere             anywhere
REJECT     all  --  anywhere             anywhere            reject-with icmp-port-unreachable
REJECT     all  --  anywhere             anywhere            reject-with icmp-port-unreachable
REJECT     all  --  anywhere             anywhere            reject-with icmp-host-prohibited

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

▼図8　DHCPサーバポートの確認

```
 BOOTPポートはDHCPポートと同じ。BOOTPはstatic設定で、DHCPはdynamic設定
[root@vm1 ~]# more /etc/services | grep bootps↵
bootps          67/tcp                          # BOOTP server
bootps          67/udp
```

ト許可を行っていることがわかります。

図8のように、DHCPサーバポートは/etc/servicesファイル内に記述されていますが、BOOTP[注4]ポートはDHCPポートと同じです。BOOTPは固定・半永続設定でIPアドレスを割り当て、DHCPは動的・期間設定でIPアドレスを割り当てる、という違いです。

図9では、ホストの仮想インターフェース側から仮想マシンにpingを行いながらtcpdumpでパケットの流れを見ています。ホストから仮想マシンにpingを打つと自動的に（ルーティング情報（**図10**）にしたがって）仮想インターフェースvirbr0から発信します。また、そのMACアドレスをarpキャッシュで確認しています。

仮想マシンの設定ファイルの保存場所は/etc/libvirt/qemuで、kvm1.xml、kvm2.xmlというファイルで保存されています。各仮想マシンの設定ファイルは本連載で参照するさまざまな情報を含んでいるので見ておくことをお勧めします。今回はMACアドレスやイメージパスの確認で十分です。

仮想マシン1－Windows 7での確認

仮想マシン1（KVM1）のWindows 7での確認は、ログオン後の「コマンドプロンプト」で、次

注4）BOOTP（BOOTSTRAP PROTOCOL、RFC951）：ディスクレスIPデバイスのためのIPアドレス配布プロトコル。BOOTPサーバのデータベースで、クライアントのホスト名／IPアドレス／MACアドレスなどの対応表を持ち、クライアント起動時にその情報を提供する。

▼図9　pingとtcpdump/MACアドレス(arpキャッシュ)確認

```
[root@vm1 ~]# ping -c 5 192.168.122.194↵
PING 192.168.122.194 (192.168.122.194) 56(84) bytes of data.
64 bytes from 192.168.122.194: icmp_seq=1 ttl=128 time=0.708 ms
64 bytes from 192.168.122.194: icmp_seq=2 ttl=128 time=0.574 ms
64 bytes from 192.168.122.194: icmp_seq=3 ttl=128 time=0.652 ms
64 bytes from 192.168.122.194: icmp_seq=4 ttl=128 time=0.570 ms
64 bytes from 192.168.122.194: icmp_seq=5 ttl=128 time=0.580 ms

--- 192.168.122.194 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.570/0.616/0.708/0.063 ms
```

```
※以下は別端末で上記ping実行中にtcpdumpで記録

[root@vm1 ~]# tcpdump -i virbr0↵    仮想インタフェースでのtcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on virbr0, link-type EN10MB (Ethernet), capture size 65535 bytes
13:25:00.113266 IP 192.168.122.1 > 192.168.122.194: ICMP echo request, id 6926, seq 1, ➡
length 64
13:25:00.132825 ARP, Request who-has 192.168.122.1 tell 192.168.122.194, length 46
13:25:00.132853 ARP, Reply 192.168.122.1 is-at 52:54:00:7b:8f:80 (oui Unknown), length 28
13:25:00.133492 IP 192.168.122.194 > 192.168.122.1: ICMP echo reply, id 6926, seq 1, ➡
length 64
13:25:01.114708 IP 192.168.122.1 > 192.168.122.194: ICMP echo request, id 6926, seq 2, ➡
length 64
13:25:01.115514 IP 192.168.122.194 > 192.168.122.1: ICMP echo reply, id 6926, seq 2, ➡
length 64
13:25:05.343119 ARP, Request who-has 192.168.122.1 tell 192.168.122.194, length 46
13:25:05.343141 ARP, Reply 192.168.122.1 is-at 52:54:00:7b:8f:80 (oui Unknown), length 28
^C
25 packets captured
25 packets received by filter
0 packets dropped by kernel

[root@vm1 ~]# arp -a 192.168.122↵    arpキャッシュ情報の確認
? (192.168.122.131) at 52:54:00:4c:a7:44 [ether] on virbr0
? (192.168.122.194) at 52:54:00:21:f2:d8 [ether] on virbr0
```

のように(DHCP設定による)アドレスと接続確認を行っています。

・ネットワーク設定の確認(DHCP割り当てアドレスや「リース取得」、「リースの有効時間」がわかる)

```
C:¥Users¥user1> ipconfig /all
```

・物理ホストの仮想インターフェースへのping

```
C:¥Users¥user1> ping 192.168.122.1
```

・物理ホストの実インターフェース側へのping

```
C:¥Users¥user1> ping 192.168.0.111
```

・物理ホストの実ネットワーク側へのping

```
C:¥Users¥user1> ping 192.168.0.1(既存システムIPアドレス)
```

仮想マシン2－FreeBSD 10.3での確認

仮想マシン2(kvm2)のFreeBSD 10.3の場合には、rootでログインしてから次のコマンドでネットワーク設定(I/Fアドレス)を確認します。

```
root@vm2fbsd:~ # ifconfig -a↵
```

また、/var/db/dhclient.leases.em0(I/F名)内「option dhcp-lease-time」にリース時間3,600秒(1時間)を得ています。

▼図10　仮想マシンのルーティング情報

```
[root@vm1 ~]# route -n↵
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0   ……★
169.254.0.0     0.0.0.0         255.255.0.0     U     1002   0        0 eth0
0.0.0.0         192.168.0.100   0.0.0.0         UG    0      0        0 eth0

 ★=192.168.122.1
virbr0    Link encap:Ethernet  HWaddr 52:54:00:7B:8F:80                          MACアドレス
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0    IPアドレス
```

ゲストシステム構築中・後の問題と対策

ゲストシステムの構築中あるいは後のトラブルをまとめます。基本的に、構築中は前号のステップ4で説明したストレージの問題を除けば、ほぼそのOS関連のものです。構築後および利用時は仮想マシン管理が関係するものもあります。これについては次回以降に解説します。

Windows 7を仮想マシンマネージャー画面からシャットダウンできない

Windows 7で入力待ち(メッセージウィンドウが開いている状態)のとき、仮想マシンマネージャー画面の「仮想マシンをシャットダウン」を行ってもシャットダウンできないことがあります。この場合、Windows 7上で対応する入力を行う必要があります。

物理ホスト側のネットワークから仮想マシンへアクセスできない

仮想マシン宛のパケットは物理ホストの仮想マシンインターフェース発信でしかアクセスできません。物理ホストの実ネットワーク上のシステムから仮想マシンにアクセスするためには、物理ホストのパケットフィルタ(iptables)のINPUTチェインやFORWARDチェインを変更する必要があります。

これについては、仮想マシン利用の回に解説します。

仮想マシン、仮想マシンマネージャーの終了

ゲストOSのシャットダウンではなく仮想マシンや仮想マネージャーの終了を行う場合、仮想マシン画面や仮想マシンマネージャー画面にある「ファイル」メニューの「閉じる」や「終了」で行いますが、仮想マシン画面での「ファイル」メニューからの実行には注意が必要です。

仮想マシン画面メニューの「閉じる」は「仮想マシンを閉じる」の意で、「終了」は「仮想マシンマネージャーを終了する」の意です。ただし、いずれも仮想マシンの状態をそのまま保持します。

次回予告

次回はこれまでで作成／インストールした仮想マシンを2つ(Windows 7とFreeBSD 10.3を個別に)使って発生する問題や不都合などについて、その原因や対応策などを考えてみます。

SD

連載各回では、読者皆さんからの簡単な「ひとくち質問(QA)コーナー」や「何かこんなこともしてほしい要望(トライリクエスト)コーナー」を設けて「双方向連載」にできればと思っていますので、質問や要望をお寄せください。

宛先：sd@gihyo.co.jp
件名に、[仮想化連載]とつけてください

RDB性能トラブルバスターズ奮闘記

原案 生島 勘富(いくしま さだよし) info@g1sys.co.jp ㈱ジーワンシステム
構成・文 開米 瑞浩(かいまい みずひろ) イラスト フクモトミホ

第7回 SQLに小回りの効く記述力を与えてくれるCASE式

これまで本連載では、「複雑なSQLを使って、DBサーバ側で集合操作をして、その結果だけを取得する方法」の効能をいろいろと述べてきました。更新処理でも同様のことが言えます。今回紹介するCASE式とパラメータテーブルはまさにそんなテクニックと言えます。

登場人物紹介

生島氏
DBコンサルタント。性能トラブルの応援のため大道君の会社に来た。

大道君
浪速システムズの新米エンジニア。素直さとヤル気が取り柄。

五代氏
大道君の上司。プロジェクトリーダーでもある。

月末の会員情報更新処理、どうしよう?

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「月末締め処理」に関するお話です。

締め処理というのは、期間中のすべての取引を走査するようなバッチ処理が走ることが多く、性能問題が起きやすい場面です。というわけで、いつものように相談を受けることになりました。

お料理レシピ投稿サイトというとクックパッドが有名ですが、要するに一般消費者が会員登録をして、料理のカテゴリ(和・洋・中など)、使っている食材、所要時間や用途(パーティ用、お弁当用、子供向け、減塩・糖質制限など)といったさまざまな条件で検索できるサービスです。

「生島さん、月末の会員ランク更新処理についてご相談したいんですが」と大道君。

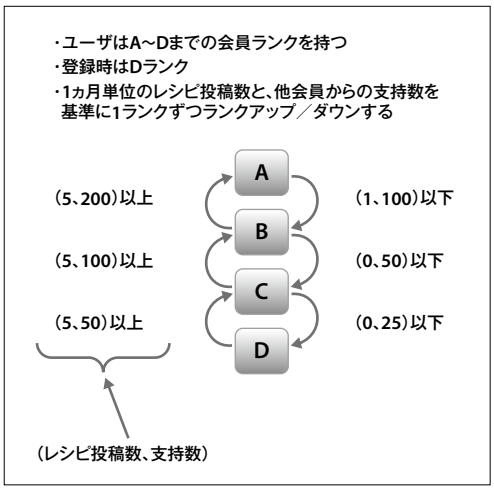
ということで会員ランク制度の概要について聞いてみると、ざっと**図1**のようなものでした。

ユーザの会員ランクはA～Dの4階層があり、会員登録時はDランクですが、レシピを投稿して人気が出ればランクが上がっていき、何もしないと下がっていくしくみです。ランクが上がると便利な機能が使えるようになります。

「つまり毎月末に、その月のレシピ投稿数と、それがほかの会員から支持された数を集計して翌月のランクを決める、というわけやね」

「そういうことです」

▼図1 会員ランク制度概要



なるほど、下手に作るといかにも単純なSQL文を多重ループで発行する「ぐるぐる系」で作ってしまいそうなケースです。もちろん大道君ならそこはわかっているので、そんなことはしないでしょう。

FULL SCANを伴うUPDATEは減らしたい

「それで、何か気になることがあるんかな？」

「**図1**を見てもらうと、ランクの変化が起きる条件が6種類ありますよね。ということは、UPDATE文を6回発行すればいいのかな……と思ったんですが、それでいいんでしょうか？」

「ああ、なるほど。それは良いとこに目をつけたね。逆に聞くけど、UPDATEを6回発行することに欠点があるとしたら、何か思い当たるかな？」

「そうですね……こういう条件のUPDATEだと、インデックスが効かないFULL SCANが走ると思うんですよ。で、それを6回やるのはちょっと無駄が多いんじゃないかな、と……」

「そうやね。全部キャッシュメモリに載ってくれたらまだマシだけど、載らないと毎回HDDを読みにいくしね」

「これ、1回で済ませる方法は何かないんでしょうか？」

「うん、あるよ」

「あるんですね！　教えてください！」

条件項目更新型UPDATEの分割実行に注意

大道君の言うとおり、これを1回で処理する方法はあります。しかしそれだけでなく、実はこの種の処理でUPDATEを複数回に分けて行うことは、別な問題も引き起こすのです。

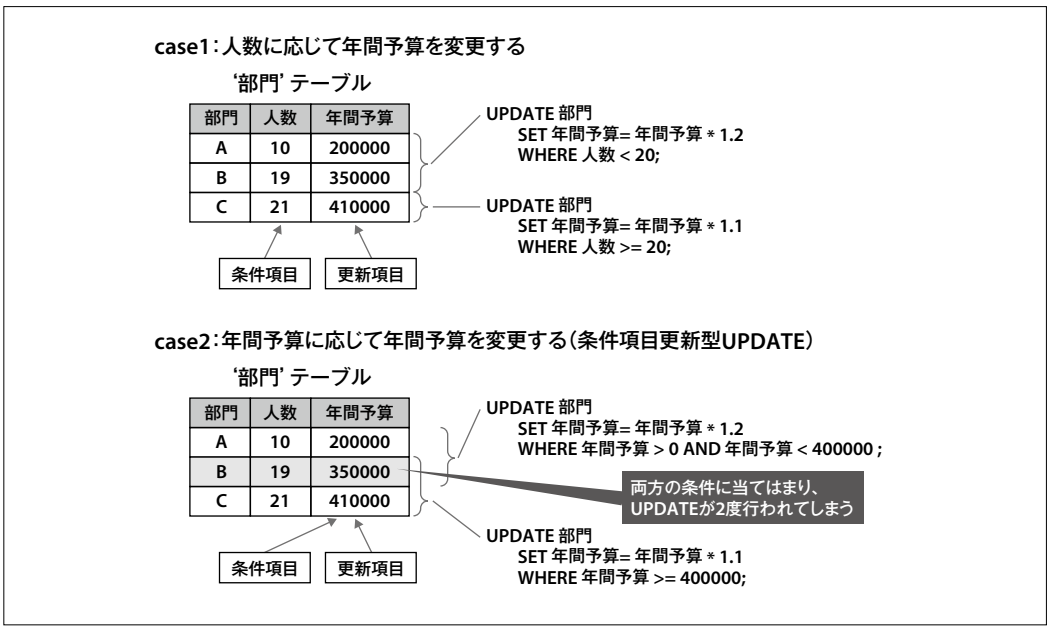
「どんな問題ですか？」

「更新結果が期待と違ってくることがあるんよ」

図2は「部門」というテーブルの「年間予算」項目を一定の条件で更新する処理の例です。case1は「人数」を判断して20人未満なら1.2倍に、20人以上なら1.1倍に更新するもので、このタイプは「条件」判断にダブリがないように設定しておけば問題は起きません。

一方、case2は「年間予算に応じて年間予算を更新する」というもので、よく考えると部門Bは

▼図2　UPDATEを複数回に分けて行う場合に起こりうる問題



35万円を1.2倍すると42万円になり、これは「40万円以上」という条件にも当てはまってしまうため、次のUPDATEでさらに1.1倍されてしまいます。

このように条件項目と更新項目が重複しているUPDATEを複数回行うと、意図せぬ結果を起こしやすいため、それを防ぐためにもUPDATEを1回で行えるならそのほうがいいのですね。

「なるほど……『条件項目更新型UPDATEの分割実行』には注意が必要なんですね」と大道君。

「そうそう、そういうこと！」

「じゃあその解決は……？」

CASE式とパラメータテーブルを活用する

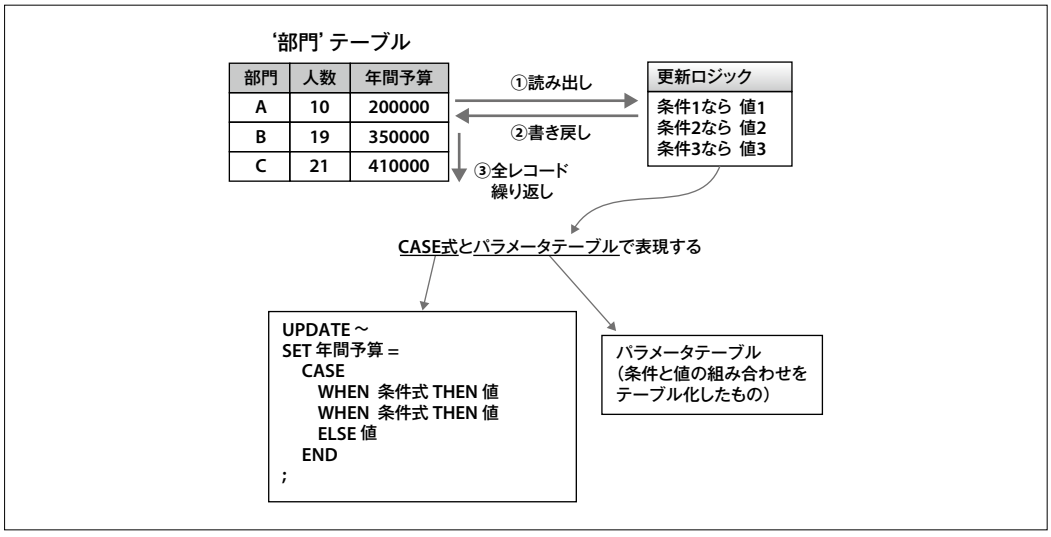
CASE式という、SQL-92から導入された仕様を使います。まずは**図3**を見てください。部門テーブルの年間予算項目を更新するなら、まず①部門Aの1レコード分を読み出し、更新ロジックのどの条件に当てはまるかを判断し、②その結果を部門Aのレコードに書き戻す、という処理を③全レコードについて繰り返す必要があります。これを1回のUPDATEで済ませるためには、「更新ロジック」を1つの式で表現できなければなりません。それを可能にするのがCASE式で、**図3**の左下のような形で使います。「SET 年間予算 = 値」の「値」の部分にCASE式を使うことで、条件に応じて違う値をSETすることができるわけです。

なお、条件と値の組み合わせをSQL文中にハードコーディングすると保守しにくくなるため、実際にはそれらを定義した「パラメータテーブル」を作ってそこから条件と値を引いてくるようにします。

「おお……こんな書き方ができるんですね！」と大道君。

「一見かなり複雑な処理でも、このCASE式

▼図3　CASE式による操作



とパラメータテーブルの組み合わせで劇的に単純化できることがあるから、使い慣れておくとええよ」

OracleではDECODE関数で代用されることもありますが、CASE式はSQLの標準仕様であり、ほかのDBMSでも使用可能ですので、CASEを基本として知っておくことをお勧めします。

会員ランク更新処理を実装しよう!

「今回のレシピ投稿サイトでCASE式を使うなら、どんな処理になるんでしょうか？」

「まあ、まずは条件判断に使うテーブルの構造を見ようか」

必要な部分だけ簡略化して示すと**図4**になります。ユーザがレシピを投稿し、そのレシピに評価がつく、という構造のため、ユーザ対レシピが1対N、レシピ対評価も1対Nの関係です。

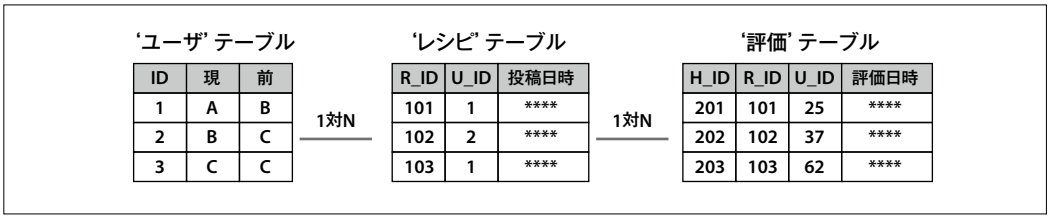
ユーザテーブルの「現」は現在の会員ランク、「前」は前月の会員ランクを表します。

月末の会員ランク更新処理の概要を構造化すると**図5**のようになります。**図5**の上半分ではユーザ、レシピ、評価テーブルのそれぞれIDの数字だけを表示して1対Nの関係がわかるようにしてあります。

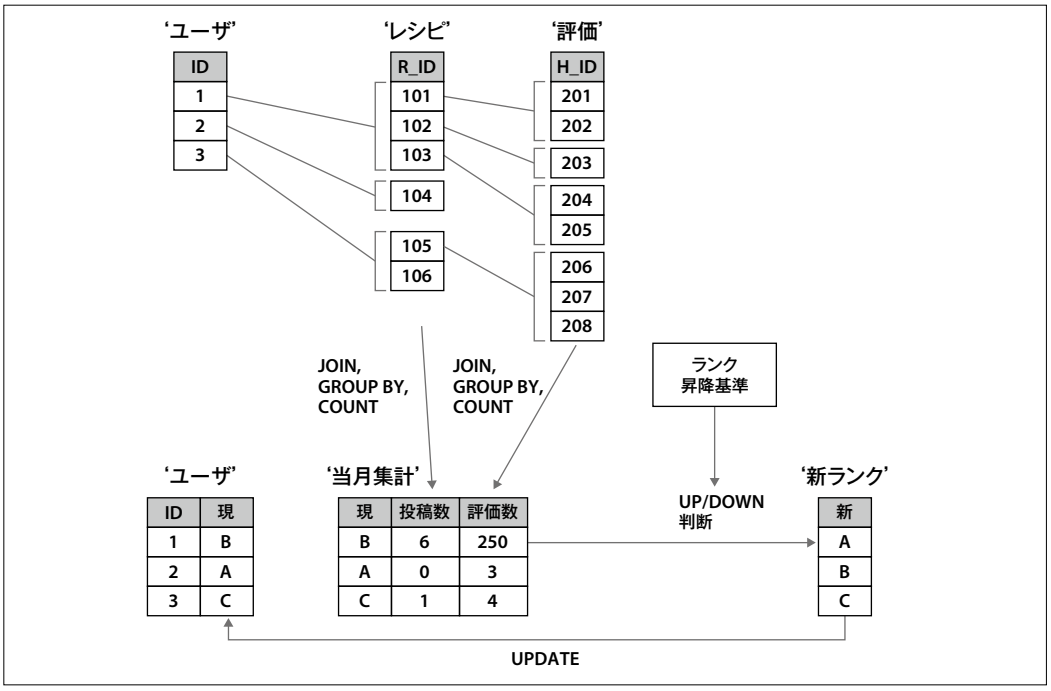
「いったん“当月集計”というテーブルを作るんですか」

「そう、その当月集計テーブルに、ユーザごとのレシピ投稿数と、他会員からの支持評価数を

▼図4　ユーザ／レシピ／評価テーブルの関係



▼図5　レシピ／評価集計、ランク判断ロジック



集計しておく。UPDATEのためのワークテーブルやね」

「はい、集計自体はJOINしてGROUP BYしてCOUNTすればいいわけですね」

「そこはJOINを2段でかけるからちょっと複雑なSQLになるけど、やってることは要するにただの集計だから、難しくはないはずだよ」

「はい、わかると思います」

いったん当月集計をすると、ユーザごとの当月投稿数と評価数(支持を獲得した数)がわかるので、それを「ランク昇降基準」に照らしてランクUP/DOWNの判断をし、新ランクを算出してその値でユーザテーブルの「現ランク」を更新します。

「UP/DOWN判断にCASE式を使うんですか」

「そう。具体的には**リスト1**のように書けばいい」

リスト1のUPDATE文末尾のu.現ランク = CASE～以下がそのコードです。

「これは……現ランクに対して、上がるか、下がるか、そのままか、の判断をしてるわけですよね？」

「そういうこと。2つのWHENでそれぞれ『上がる』『下がる』条件を表し、どちらにも該当しなかったらELSEに来るから現ランクのまま、変わらない」

「ランクのUP/DOWN条件は6種類ありますけど……」

「そこはランク昇降基準を表すパラメータテーブルで吸収する」

図6がそのためのランクパラメータテーブルです。「現ランク」ごとに、ランクUP条件、UP後の新ランク、ランクDOWN条件、DOWN後の新ランクの設定値を保持しています。そのうえで現ランクをキーにして当月集計テーブルとランクパラメータテーブルをJOINすると、現ランクに関係するUP条件とDOWN条件だけが残るわけです。

集計と更新の一発化はできない?

「なるほど……こんなやり方ができるんですね！　おもしろいです！　でもこれ、『一発系SQL』を追求するなら、当月集計テーブルを作るのもやめて、一気にUPDATEしてしまうわけにはいかないんでしょうか？」

「残念ながらそれはできないんよ」

というのは、集計するSELECT文に「ユーザ」テーブルが入ってしまうからです。集計元に入ったテーブルにはロックがかかってしまうため、1

▼リスト1　CASE式を使った会員ランクUPDATE処理

```
CREATE TABLE ユーザ
    ( ユーザID INT     NOT NULL
    , 現ランク CHAR(1) NOT NULL
    , 前ランク CHAR(1) NULL
    -- 以下略
    , PRIMARY KEY (ユーザID)
    );
```

```
CREATE TEMPORARY TABLE 当月集計
    ( ユーザID   INT     NOT NULL
    , 現ランク   CHAR(1) NOT NULL
    , 当月投稿数 INT     NULL
    , 当月支持数 INT     NULL
    , PRIMARY KEY (ユーザID)
    );
```

```
UPDATE ユーザ u
    INNER JOIN 当月集計 s
        ON u.ユーザID = s.ユーザID
    INNER JOIN ランクパラメータ r
        ON s.現ランク = r.現ランク
SET u.前ランク = u.現ランク
    , u.現ランク =
        CASE
        WHEN s.当月投稿数 > r.U投稿数 AND s. 当月支持数 > r.U支持数 THEN r.Uランク
        WHEN s.当月投稿数 < r.D投稿数 AND s. 当月支持数 < r.D支持数 THEN r.Dランク
        ELSE u.現ランク END
;
```

文で同時にUPDATEすることができません。そこでいったん「当月集計」というワークテーブルを作り、その情報をもとにUP/DOWNを判断してユーザテーブルを更新する、という処理を行うわけです。

CASE式はSQLに小回りの効く記述力を与えてくれる

「できないんですか、それは残念です。でも、CASE式っておもしろいですね！　SQLってデータの集合にまとめて同じ操作をするための言語とばっかり思ってましたけど、『同じ操作』のところで意外に細かいロジックも書けるってことですよね、これ？　ますます、SQLでやれることが増えますね！」

さすが大道君、カンがいいです。CASE式を使うと、手続き型言語でif文やswitch文を入れ子にした複雑なロジックを組んでいた部分を大幅に単純化できることがあります。しかも今回のように「パラメータテーブル」と組み合わせると、条件を変更するときも設定データを変えるだけで、コードには変更が及ばずに済むため、保守作業コストを大きく減らすことができます。

簡単に言うと、一定のパターンでの条件判断がいくつもある場合は、こうしたパラメータテーブル方式が使えることが多いのです。しかし、基本設計の段階でテーブルを決めてしまうウォータフォール式の開発スタイルでは、「目に見える項目の洗い出し」が精いっぱいのことが多く、なかなか「パラメータテーブルを作っておこう」という発想が出てこないようです。その意味でも、前回紹介した「テーブル設計を後回し」にする手法をお勧めします。後まわしで良いなら、処理を実装しながら「ここはパラメータテーブル化したほうが良い」と気がついたときに、躊躇（ちゅうちょ）なくそれを採用することができます。その意味でも、「テーブル設計後まわし」方式をお勧めします。

SQLをきちんとわかっていれば、こんな方法でも開発効率および実行速度の両方を向上できますので、一定のパターンでの条件判断が多発するコードを見つけたら、ぜひ、CASE式とパラメータテーブルを使ってみてください。SD

▼図6　ランクパラメータテーブル

	ランクUP条件		UP後の新ランク	ランクDOWN条件		DOWN後の新ランク
現ランク	U投稿数	U支持数	Uランク	D投稿数	D支持数	Dランク
A	NULL	NULL	NULL	1	100	B
B	5	200	A	0	50	C
C	5	100	B	0	25	D
D	5	50	C	NULL	NULL	NULL

※U（ランクアップ）とD（ランクダウン）を同時に満たすことがないように注意して仕様を決めること

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9回 VRアプリをつくろう!

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

野田 悟志(のだ さとし)
日本Androidの会 神戸支部、GDG神戸
URL http://kobegdg.blogspot.jp/
Mail scarviz@gmail.com

Google VRとは

Googleが年に1度、技術者向けに「Google I/O」という大規模なカンファレンスを実施しています。2016年の今回は、「Daydream」という、スマートフォンを使って高品質な没入感を得られるVRプラットフォームを発表しました。このDaydreamとCardboardをまとめたカテゴリを「Google VR」と言っています。

今後はこのDaydreamに対応したスマートフォンが普及していくのではないかと思いますが、まだ発表されたばかりで、端末は発売されていません(今秋あたりから発売予定)。しかし、DaydreamとCardboardは同じGoogle VR SDKを使ってアプリを作ることができます。そのため、今回は従来のCardboardアプリの作成の仕方を説明していこうと思います。

AndroidはKitkat(API19)以降の端末が必要です。Cardboardは1,500円くらいで、Amazonなどで入手できます。5インチ端末まで対応したバージョン1と、6インチ端末まで対応したバージョン2があり、どちらでも構いませんが、バージョン2のほうが扱いやすいと思います。

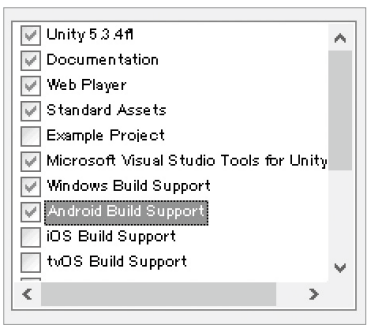
Unityをインストールする

Cardboardアプリは、UnityやUnreal Engineといったゲームエンジンを使うと楽に作成できます。今回はUnityを使用します。

2016年6月の執筆時点では5.3.5が最新バージョンになります。しかし、Canvas(ボタンなどのUIを配置するもの)で不具合が上がっているため、今回は1つ前の5.3.4を使用します[注1]。

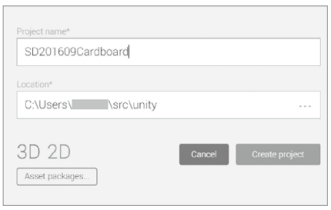
注1) Unity 5.4(7月28日時点)ではこの不具合は解消されています。

▼図1 インストールする対象

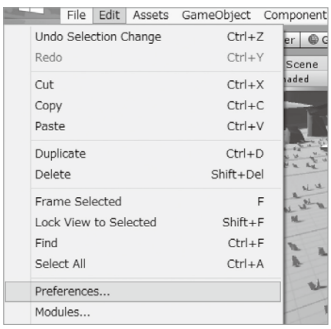


▼図2 上部のNewボタン

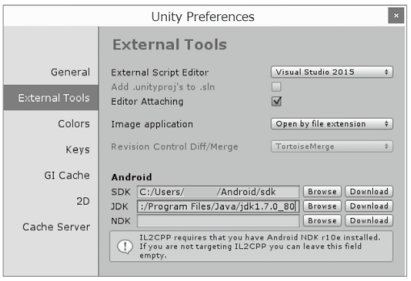
▼図3 新規プロジェクト作成



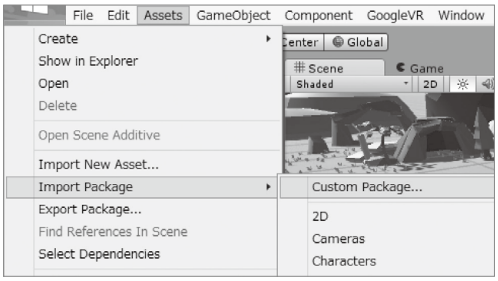
▼図4 Preferencesを選択



▼図5　External ToolsのAndroid設定



▼図6　パッケージのインポート



「Unityダウンロードアーカイブ[注2]」から5.3.4のインストーラをダウンロードしてください。

インストーラを起動し、ダイアログに従って進めていくと、**図1**のようなリストが表示されます。デフォルトでチェックが入っているものに加えて、「Android Build Support」にチェックを入れるようにしてください。

インストールが終わったらUnityを起動します。最初にUnityアカウントでログインする必要があります。まだアカウントを持っていない場合は新規登録し、ログインしてください。

上部にあるNEWボタン(**図2**)を押し、新規に作成するプロジェクト名とディレクトリパス、3Dを指定し、Create projectボタンを押してください(**図3**)。

次にAndroid SDKとJDKのパスを設定します。Unityのメニューから、[Edit]→[Preferences]を選択(**図4**)し、Preferencesダイアログを表示します。External Toolsを選択し、Android SDKディレクトリパスと、JDKのバージョン名ディレクトリをそれぞれ設定します(**図5**)。

Google VR SDK for Unityのインポート

Google VR SDK for Unityをインポートするために、GitHub上のgvr-unity-sdk[注3]へ行き、「GoogleVRForUnity.unitypackage」をダウンロードしてください。

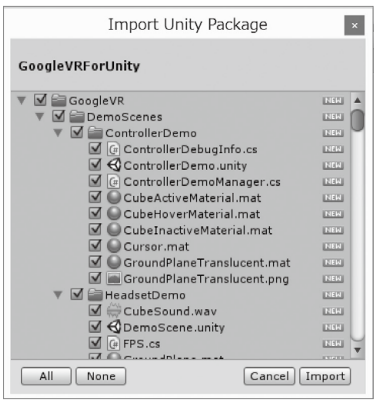
メニューの[Assets]→[Import Package]→[Custom Package]を選択(**図6**)し、先ほどダウンロードしたGoogleVRForUnity.unitypackageを開いてください。Importダイアログが表示され、デフォルトですべてチェックが入っていると思います(**図7**)。そのままの状態でImportボタンを押してください。

簡単なCardboardアプリを作ってみる

AssetStoreから「Low Poly: Free Pack」というAssetをベースにしてアプリを作成しようと思います。該当アプリのAssetStore[注4]をブラウザで開き、「Unityで開く」ボタンを押します。

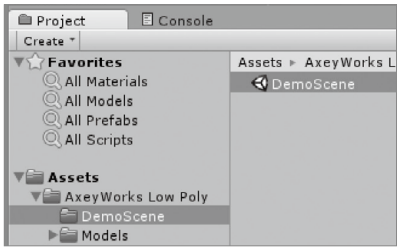
注2) https://unity3d.com/jp/get-unity/download/archive
注3) https://github.com/googlevr/gvr-unity-sdk/
注4) https://www.assetstore.unity3d.com/jp/#!/content/58821

▼図7　Importダイアログ

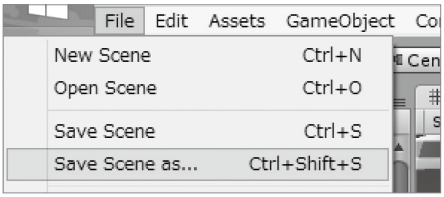


▼図8　AssetStoreの「Low Poly: Free Pack」

▼図9　DemoScene



▼図10　Sceneの別名保存



もし図8のように警告ダイアログが表示された場合は、「アプリケーションの起動」ボタンを押します。Unity側でAssetStoreタブが表示されるので、その中からImportボタンを押し、先ほどと同じようにインポートしてください。

Projectビューの[Assets]→[AxeyWorks Low Poly]→[DemoScene]→[DemoScene]をダブルクリックして開きます(図9)。いったんこのSceneを別名で保存します。メニューの[File]→[Save Scene as]からMainという名前で保存してください(図10)。

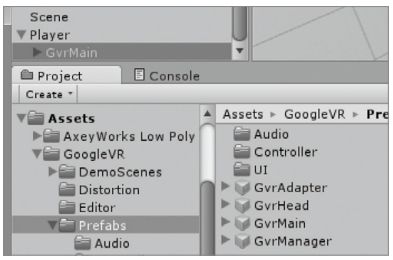
画面を二画面にする(カメラのステレオ化)

メニューから[GameObject]→[Create Empty]を選択し、空のGameObjectを追加します(図11)。名前をPlayerに変更してください。Projectビューから[Assets]→[GoogleVR]→[Prefabs]→[GvrMain]をPlayerの中に入れます(図12)。

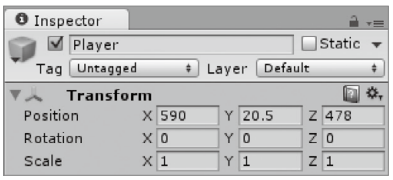
Playerを選択し、InspectorビューでPositionをX：590、Y：20.5、Z：478に変更します(図13)。PlayerをダブルクリックするとSceneビューに、周りに物がある状態で表示されるよ

▼図11　空のGameObjectを追加

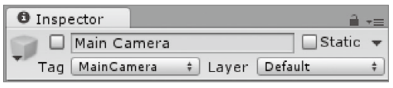
▼図12　GvrMainの追加



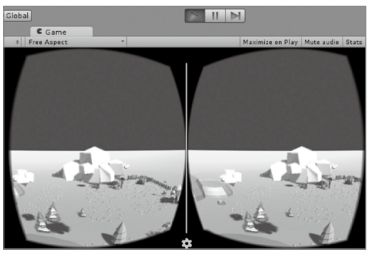
▼図13　Playerの位置の設定



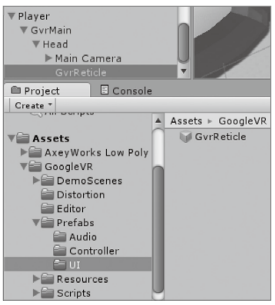
▼図14　Main Cameraの無効化



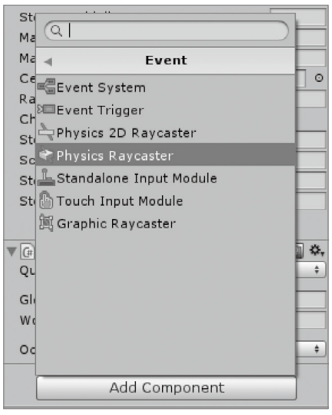
▼図15　ステレオカメラ化の確認



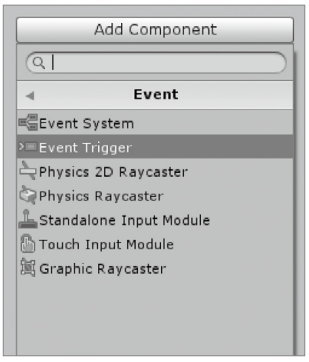
▼図16　GvrReticleの追加



▼図17　Physics Raycasterコンポーネントの追加



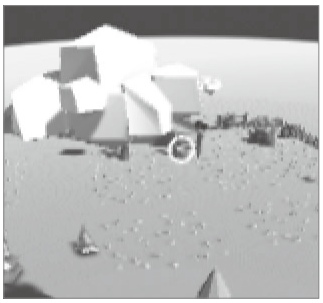
▼図18　Event Triggerコンポーネントの追加



▼図19 GazeInputModule コンポーネントの追加

▼図20 Standalone Input Moduleの無効化

▼図21 レチクルが円に変わる



うになると思います。

Hierarchyビューの Main Cameraを選択し、Inspectorビューでチェックボックスのチェックを外して無効化します(**図14**)。

では早速、再生ボタンを押してステレオカメラ化しているか確認してみましょう(**図15**)。止める場合はもう一度再生ボタンを押します。

視線(Gaze)による操作(ワープ移動する)

今の地点から移動するために、視線の先にワープする処理を入れてみます。

レチクル(視線の照準の点)の表示

Projectビューの[Assets]→[GoogleVR]→[Prefabs]→[UI]→[GvrReticle]を[Player]→[GvrMain]→[Head]に入れます(**図16**)。再生ボタンを押し再生してみると、中央に白い点が表示されていると思います。Altキーを押した状態でマウスを動かすと、ヘッドトラッキングしているように動かせますが、中央の白い点はそのまま中央にあることが確認できると思います。これがレチクルという視線の照準点になります。

視線(Gaze)によるレチクルの反応(レイキャスト)

ワープ先のGameObjectにレチクルが当たった場合、点から円に変化するように処理を加えます。

[Player]→[GvrMain]→[Head]→[Main Camera]を選択し、Inspectorビューで最下部にあるAdd Componentボタンを押し、[Event]→[Physics Raycaster]でPhysics Raycasterコンポーネントを追加します(**図17**)。

ワープ先として、Well_Stone_Water_01(井戸)にしましょう。HierarchyビューのWell_Stone_Water_01を選択し、InspectorビューでAdd Componentボタンを押し、[Event]→[Event Trigger]でEvent Triggerコンポーネントを追加します(**図18**)。

メニューから[GameObject]→[UI]→[Event System]を選択してEvent Systemを追加します。Event Systemを選択し、InspectorビューでAdd Componentボタンを押し、[GoogleVR]→[GazeInputModule]でGazeInputModuleコンポーネントを追加します(**図19**)。同じくInspectorビューで、Standalone Input Moduleのチェックボックスを外します(**図20**)。

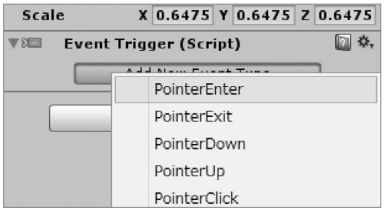
再生して、レチクルを井戸に当てると、点から円に変わることがわかります(**図21**)。

視線(Gaze)の先にワープ移動する

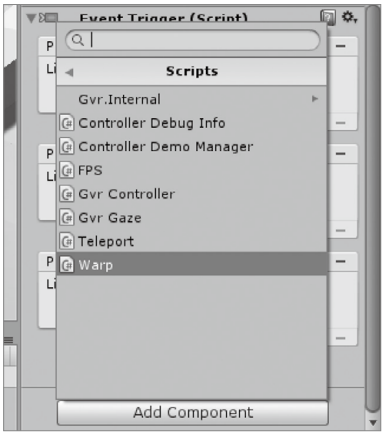
Well_Stone_Water_01(井戸)のInspectorビューのEvent Triggerで、Add New Event Typeボタンを押し、Pointer Enter、Pointer Exit、Pointer Clickの3つを順番に追加します(**図22**)。

ProjectビューのAssetsを選択し、右クリック→[Create]→[Folder]でScriptsフォルダを

▼図22　Event Triggerへ追加



▼図23　Warpスクリプトの追加



▼リスト1　視線の先にワープする処理（Warp.cs）

```
public class Warp : MonoBehaviour, IGvrGazeResponder {
    [SerializeField] private GameObject Player;
    [SerializeField] private GameObject WarpPoint;

    public void OnGazeEnter() {
        Debug.Log("OnGazeEnter");
    }

    public void OnGazeExit() {
        Debug.Log("OnGazeExit");
    }

    public void OnGazeTrigger() {
        // ワープポイントの位置
        var warpPos = WarpPoint.transform.position;
        // 高さの調整
        warpPos.y += 3;
        // 距離の調整（井戸の手前になるようにする）
        warpPos.z -= 5;
        // ワープポイントの位置に移動する
        Player.transform.position = warpPos;
    }
}
```

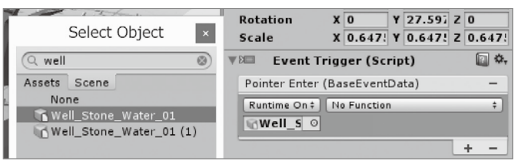
作成し、その中で右クリック→[Create]→[C# Script]でWarp.csを追加します。Warp.csを開き、**リスト1**のように、IGvrGazeResponderインターフェースを実装します。OnGazeEnter、OnGazeExitは対象にレチクルが入ったとき、出たときに呼ばれます。今回はデバッグログを出力するように実装しました。OnGazeTriggerは対象にレチクルが当たった状態で、クリックした場合に発生します。ここでWarpPointというGameObjectの手前に移動するように実装します。WarpPointはこの後、井戸に紐(ひも)づけるようにします。

Well_Stone_Water_01（井戸）のInspectorビューでAdd Componentボタンを押し、[Scripts]→[Warp]を追加します（**図23**）。

同じく井戸のInspectorビューで、Event Triggerの各イベントの＋マークを押し、表示されるボックス（Noneと表示されている）の右横の丸をクリックします。ダイアログが表示されるので、検索Boxで「Well_Stone_Water_01」を入力し、SceneタブのWell_Stone_Water_01を選択します（**図24**）。各イベント処理のコンボボックスで、Warpスクリプトのメソッドを、Pointer EnterにはOnGazeEnter、Pointer ExitにはOnGazeExit、Pointer ClickにはOnGazeTriggerを設定します（**図25**）。

井戸のInspectorビューのWarpスクリプトで、PlayerとWarp Pointのボックスの右横の

▼図24　Warpスクリプトを追加したGameObjectを選択する



▼図25　各イベント処理にWarpスクリプトのメソッドを設定する

丸を押し、ダイアログを表示し、それぞれSceneタブのPlayerとWell_Stone_Water_01を選択します(**図26**)。

再生ボタンを押し、レチクルを井戸に当て、クリックをしてみると、井戸の手前にワープ移動することができるようになりました。

Androidにインストールする

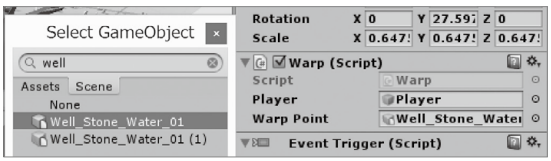
メニューの[File]→[Build Settings]を選択します(**図27**)。

Build SettingsダイアログでAdd Open Scenesボタンを押し、Platformのリストから Androidを選択し、Switch Platformボタンを押します(**図28**)。Player Settingsボタンを押し、InspecterビューのResolution and Presentationの Default Orientation を Landscape Leftに変更します(**図29**)。Other Settingsで、Bundle Identifierに任意のパッケージ名、Minimum API Levelを19(Android 4.4)に設定します(**図30**)。Android端末をPCにつなげ、Build SettingsダイアログのBuild and Runボタンを押し、apkファイルのファイル名と出力先を聞かれるので任意の値を入力し続行すると、Androidにインストールされます。ぜひCardboardにセットして、まわりを見渡したり、ワープ移動してみてくださいね。SD

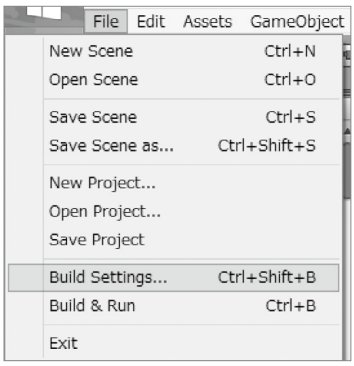
COLUMN VR酔いについて

今回の移動手段はワープでした。もっとリアルに人が歩いているような、少し上下しながら前進するモーションで移動させたほうが良いのではないかと思う人もいるかもしれません。しかし、その場合、実際の体が動いていないのに、視界は体が動いているように見えるため、そのズレから気持ち悪く感じます。このことを「VR酔い」と言います。これは車酔いに似ています。より良いVR体験をしてもらうためには、なるべくVR酔いが起こりにくいコンテンツの作成が求められますので、みなさんも意識してみてください。

▼図26 PlayerとWarp Pointの設定

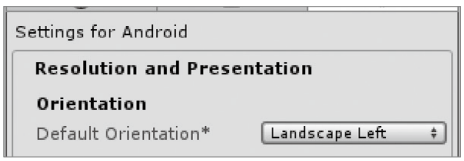


▼図27 Build Settingsの選択

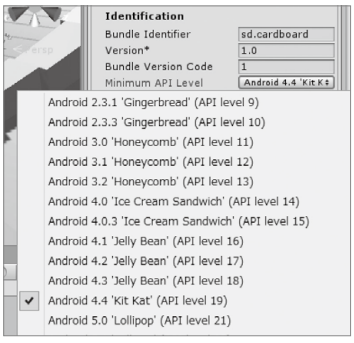


▼図28 Build Settingsダイアログ

▼図29 Orientationの設定



▼図30 Bundle IdentifierとMinimum API Levelの設定



一歩進んだ使い方のためのイロハ

Vimの細道

第11回

mattn
twitter:@mattn_jp

Vimの設定ファイル再点検

今回は初心に立ち返り、Vimの設定ファイルを取り上げます。vimrcとgvimrcの違い、設定時に気をつけるべきオプション（エンコーディングに関する3つのオプション、ambiwidth、autocmd）について解説しながら、どのような方針で設定を加えていくべきなのかを考えましょう。

IDE最新ランキング

Googleサーチエンジンで検索された回数やGoogleトレンドでの動向を基に、開発者が使用する人気の開発環境のランキングを提供する「TOPIDE（Top IDE index）」が2016年7月、IDE（統合開発環境）のランキング情報を公開しました[注1]。

「VimはIDEなのか」という議論もありますが、なんとVimが4位にランクインしました（**図1**）。最初のリリースからもう二十年以上も経っているテキストエディタVimがほかの開発環境を抑えて4位にランクインしたことに、驚くと同時に複雑な気持ちにさえなりました。二十数年前からVimを使っている人もいれば、2016年からVimを始める新入社員もいるのです。二十数年前と言えばその新入社員も生まれていなかったかもしれません。もちろん当時のVimはとても貧弱で、設定できる項目も少なかったかと思います。しかしそれから二十年もの間、Vimはいろいろな機能を取り込みながら、休まずに改良が続けられてきました。

現代でもTraditional viを好む人は数多くいます。Vimを使っていながらも、ほとんど設定しない状態を好む人もいます。しかしせっかくVimを使うのですから、設定のしかたくらいは覚えておいて損ではありません。今回はVimの設定方法をおさらいし、陥りやすいミスについて紹介したいと思います。

注1）URL https://pypl.github.io/IDE.html

Vimの設定ファイルとは

理解しながら設定する

実はVimの設定方法を知らない人はけっこう多く、それを調べるためにGoogle検索を使う人も多いと思います。その結果として、TOPIDE

▼図1　TOPIDE（2016年7月）

Worldwide, Jul 2016 compared to a year ago:

Rank	Change	IDE	Share	Trend
1	↑	Visual Studio	23.35 %	+0.6 %
2	↓	Eclipse	23.07 %	-5.8 %
3	↑	Android Studio	10.09 %	+2.7 %
4	↓	Vim	7.99 %	-0.0 %
5		NetBeans	5.62 %	-0.4 %
6		Xcode	5.22 %	-0.2 %
7		Sublime Text	4.57 %	+0.1 %
8	↑	IntelliJ	4.0 %	+0.9 %
9	↓	Komodo	3.88 %	+0.1 %
10	↑	Xamarin	2.94 %	+1.0 %

の結果になったとも言えるでしょう。皆がVimの設定方法を知っているのであれば、あえて検索しようとは思いませんからね。

GitHubが登場したことで、Vimの設定ファイルをGitHubに置く人も多くなってきました。GitHubの検索ボックスに「vimrc」と入力して検索すると、執筆時点(2016年7月)で9,821個のリポジトリがマッチし、775,379個ものファイルが登録されているようです。設定ファイルを分割して管理しているリポジトリも合わせるともっとあるでしょう。

それだけの数があると、知らないユーザのよくわからない設定を、ちゃんと理解しないまま自分の設定ファイルに取り込んでしまう人も、実は多いんじゃないかと思います。現に、毎週土曜日にオンラインで行われる「vimrc読書会」で題材にする設定ファイルでは、「これおそらくどこからかコピーしたんじゃないか？」といった内容もまれに見られます。設定ファイルのコピー&ペーストをするなとは言いませんが、ちゃんと理解したうえで設定を行うと、今後その設定を変更する際に問題になりにくいだけでなく、その設定がもたらす副作用にも対処できるようになります。

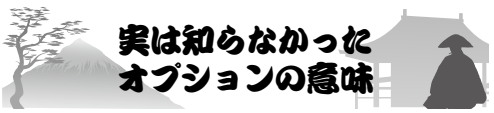
vimrcとgvimrc

最近のテキストエディタであれば、設定用のユーザインターフェースが用意されているものが当然になってきました。画面のチェックボックスにポチポチとチェックを入れるだけの便利なものもあります。しかしながら、VimはUNIXの、しかも化石のように古いテキストエディタです。すべての設定はvimrcという設定ファイルで行います。

Vimの設定ファイルは2種類あります。vimrcとgvimrcです。vimrcはCLI(Command Line Interface)とGUIの両方から読まれる設定ファイルです。かたやgvimrcは、GUI版のみがvimrcのあとに読み込む設定ファイルです。よく間違われるのですが、GUI版はgvimrcだけを読み込むのではありません。

またvimrc/gvimrcは、Vimが動作するOSによって異なるファイル名になります。UNIX系のOSでは「.vimrc」および「.gvimrc」となります。Windowsでは「_vimrc」と「_gvimrc」です。以降ではvimrc/gvimrcと記載します。

設定ファイルはテキスト形式で記述し、set構文もしくは、Vim scriptというVimが内部で実装している制御構文を使って記述します。以降では、vimrcの記述で気を付ける個所について説明していきます。



nocompatible

インターネット上に転がっているvimrcの多くで、先頭部分に付いているのがこの行です。

```
set nocompatible
```

Vimのset構文はオプション名の先頭にnoを付けることで、「そのオプションを無効にする」という働きをします。このオプションはcompatibleオプション、つまりvi互換を無効(no)にしてVimの独自機能を有効にする、という意味になります。あまり知られてはいませんが、実はこれ、正しい記述ではありません。nocompatibleオプションはvimrcもしくはgvimrcファイルを見つけた時点で自動的にnocompatibleとなります。ですので、vimrcファイルを置いているのであれば書く必要がありません。

また、vimrcをVimで編集中に、変更した内容を反映すべく、

```
:source ~/.vimrc
```

とすることがありますが、set nocompatibleの指定があると、いくつかのオプションがデフォルト値に戻ってしまいます。たとえば、historyオプション(コマンドライン履歴)の値を増やしている人もいるかと思いますが、set nocompatible

▼表1　エンコーディングに関するオプション

オプション名	オプションの意味
encoding	Vimの内部エンコーディング
fileencoding	ファイルのエンコーディング
fileencodings	ファイルを開く際に試すエンコーディング群

が実行されると、その瞬間にhistoryの値がデフォルト値である50に戻ります。その後、vimrcの記述のとおり自分の設定内容が実行され、元の増やした値になるのですが、historyオプションは一度減ってしまうと、その瞬間50よりも多く蓄積されていたはずのコマンドライン履歴が50まで削除されてしまいます。

モダンな書き方をするのであればset nocompatibleの記述は削除すべきです。

Vimのエンコーディング設定のハマりどころ

encoding/fileencoding/fileencodingsの設定を正しく理解していない人がたまに見受けられます。簡単に説明すると**表1**のようになります。

fileencodingとencodingの関係

Vimは内部のエンコーディングを変更できるテキストエディタです。fileencodingsにカンマ区切りで列挙されたエンコーディング名に従って、encodingオプションが示すエンコーディングへの変換を試みます。すべてのエンコーディング変換に失敗した場合、Vimはfileencodingの値を空に設定します。また、途中に変換可能なエンコーディングを見つけた場合、そのエンコーディング名をfileencodingオプションに設定します。

ファイルのエンコーディングを変更したい場合、たとえばutf-8のファイルをeuc-jpとして保存するときは、次を実行して:wで保存します。

```
:set fileencoding=euc-jp
```

これにより、以後そのファイルはeuc-jp形式として扱われます。しかし、もしファイルの中にeuc-jpへ変換できない文字が含まれていた場合、Vimはファイルの保存時にエラーを発生させます。euc-jpやcp932よりも、utf-8のほうが扱える文字の数が多いので、encodingはutf-8にしておいたほうが良いということになります。

Vimは、encodingオプションが示すエンコーディングで文字の検出を行います。カーソルを1つ動かしたときにマルチバイト文字を正しく1文字ずつ移動しているのは、この処理によるものです。以前であれば、WindowsではOSのANSIエンコーディングであるシフトJIS、つまりcp932を設定するのが良いとされていましたが、最近のWindow版はencodingをutf-8にしていても、とくに問題が起きることがなくなりました。

fileencodingsは順序が大事

前述したように、Vimはファイルを開く際にfileencodingsに列挙されたエンコーディング名を順に試します。しかし、ファイル内にマルチバイト文字の数が少ないと誤検知してしまう場合があります。fileencodingsの設定で、cp932よりも後にutf-8を書いている設定がありますが、cp932は取り得るバイトのパターンが単純でかつ範囲が大きいのです。

iso-2022-jpのように漢字やカタカナの前にマーカーが入ったり、utf-8のようにバイト列に決まったシーケンスがあると誤検知することは少ないのですが、cp932はとても誤検知を生みやすいエンコーディングです。

たとえば、Linuxの端末で次を実行してみます。

```
$ echo あかさたな | iconv -f cp932 -t utf-8
```

最近のLinuxの端末はutf-8ですので、このコマンドは「utf-8な文字列をcp932な入力としてutf-8へ変換する」というものです。一見このコマンドはエラーになりそうですが、実は正常終了してしまいます。xxdコマンドでバイト列の内容を確認します。

```
$ echo あかさたな | xxd -u
00000000: E381 82E3 818B E381 95E3 819F ➥
E381 AA0A ...
```

cp932の第1バイトは0x81～0x9Fおよび0xE0～0xFC、第2バイトは0x40～0x7Eおよび0x80～0xFCです。ちょうどこの範囲に収まってしまっています。Vimでファイルを開く際にfileencodingsの前の方にcp932があると、本当はutf-8であるにもかかわらず、cp932として開かれてしまいます。とくに必要がないのであれば、cp932は最後に持ってくるべきです。

次は筆者のfileencodingsの設定です。

```
set fileencodings=ucs-bom,iso-2022-jp,➥
euc-jp,cp932
```

fileencodingのグローバルな意味

ここで気を付けておくべきポイントがあります。fileencodingオプションには2つの意味があるということです。1つは現在のバッファに対して設定される、「編集中のファイルのエンコーディング」を示すもの。もう1つはグローバルに働くものです。このグローバルな値は新しい空のバッファが作成された際に、そのバッファのエンコーディングを何にするかを決定します。vimrcで次のように設定します。

```
set fileencoding=utf-8
```

こうすると新規に開いた空のバッファは、編集後に:wで保存するとutf-8で保存されます。たとえば、encodingをutf-8に、かつグローバルなfileencodingをeuc-jpにした場合、euc-jpに存在しない文字を含んだテキストを:wで保存するとエラーになります。オプションの正しい意味を理解しておくことで、保存時にエラーとなっても対応できるようになります。

fileencodingの変更はファイルの変更

Vimにはmodelineというマジックコメント機能があり、Vimでファイルを開いた際にいくらか編集に適した設定を有効にできますが、まれにfileencodingを指定しているものがあります。しかし、fileencodingオプションはVimが検出するオプションです。Vimが検出するエンコーディングと異なる値が設定されている場合、Vimは、ファイルのエンコーディングが変更されたと判断してしまいます。よってファイルを開いた瞬間に、ファイルが変更されるという変な動作になります。

Vimプラグインを書くならscriptencodingはマナー

また、これらのオプションとは別にscriptencodingというコマンドが用意されています。これは、vimrcやVim pluginがどのエンコーディングで記述されているかを明示するためのコマンドです。encodingとファイルのエンコーディングが同じ場合はとくに必要ありませんが、ほかのencodingを使用している場合には、次のようにこのコマンドを使います。

```
scriptencoding utf-8

let g:foobar = 'こんにちわ世界'
```

ファイルにマルチバイト文字が含まれていない場合には、とくに必要ありません。ただしマルチバイト文字が含まれている場合で、かつほかのユーザにそのファイルを公開する場合にはencodingがutf-8でない人のために、このコマンドをファイルの先頭に付けておくのがマナーとなります。

ambiwidth

こちらも、間違って認識されがちなオプションです。Vimのように端末で起動するアプリケーションでよく問題となるのが文字幅問題です。東アジア各国で使われる文字の中には、文字幅があいまいに扱われているもの(East Asian Ambiguous Width)がいくつもあります。端末のように固定幅の文字で表現すべき環境では、これらの文字を1セルで描画するか2セルで描画

するかが決まっていないため、描画が崩れる問題があります。次の例を見てください。

```
こんにちわＸＹです。
```

例として「X」と「Y」は文字幅があいまいな文字とします。環境によっては、次のように2通りの描画のされ方になってしまうことになります。

```
こんにちわXYです。
こんにちわＸＹです。
```

もし、絶対座標でカーソルが「Y」の位置に移動するとした場合、それは行頭から見て何セル目になるのでしょうか。1行目であれば12セル目、2行目であれば14セル目になります。つまり、端末があいまいな幅の文字を2セルで描画するのか1セルで描画するのかによって、Vimが想定している幅と差異が生まれ、結果として描画が崩れるのです。

ambiwidthはこれらのあいまいな幅の文字に対して端末が1セルで描画するのか2セルで描画するのかをVimに教えてあげるオプションです。端末が2セルで描画しているならdoubleを、1セルで描画しているならsingleを設定します。

autocmd

そして、一番多く見かける間違いがこのautocmdです。autocmdは、Vimが用意したイベントに対してユーザ指定のコマンド実行を登録するコマンドですが、このコマンドを複数回実行すると、その分だけイベントが登録されます。vimrcを再読み込みするとイベントが複数回登録されるので、イベントが発動してしまうと処理が複数回実行され、期待しない動作になります。そのためVimではコマンドの解除を行うコマンドも用意しています。

まず、autocmdはaugroupと一緒に使います（**リスト1**）。autocmd!が解除コマンドになります。augroupの中で実行すると、augroupで名付けられたグループのみに作用します。つまりこの例の場合、グループ「MyVimrc」内で登録したイベントのみが解除されます。こうすることで、vimrcを何度も読み直してもイベントが複数登録されることはなくなります。

同じグループ名で異なるイベントの登録をすることはできますが、その都度autocmd!を実行してしまうと、毎回イベントが解除され最後のイベントしか残らなくなってしまいます。

リスト2のように同じグループ名を使いまわして冒頭ですべてを解除するか、**リスト3**のようにカテゴリごとにグループ名を設定し、そのカテゴリごとに解除を実行するのが良いです。

コピーする情報源も選ぶ

これまで説明したように、Vimの設定は知らずにコピーすると失敗する要因が多々あります。誤解を恐れず言うならば、コピーする、参考に

▼リスト1　autocmdの使用例

```
augroup MyVimrc
  autocmd!
  autocmd FileType java setlocal omnifunc=javacomplete#Complete
augroup END
```

▼リスト2　グループ名を使いまわす場合

```
augroup MyVimrc
  autocmd!
augroup END

" java の設定
augroup MyVimrc
  autocmd FileType java setlocal omnifunc=javacomplete#Complete
augroup END

" python の設定
augroup MyVimrc
  autocmd FileType python silent setlocal ts=2 sw=2 sta et sts ai
augroup END
```

する設定ファイルもできるだけVimスキルが高い人のものを選ぶべきです。ただし完全にコピーするのではなく、自らのものにするためにも、意味をきちんと理解して自分専用のvimrcを組み立てるべきです。SD

▼リスト3　カテゴリごとにグループ名を設定する場合

```
" java の設定
augroup MyVimrcJava
  autocmd!
  autocmd FileType java setlocal omnifunc=javacomplete#Complete
augroup END

" python の設定
augroup MyVimrcPython
  autocmd!
  autocmd FileType python silent setlocal ts=2 sw=2 sta et sts ai
augroup END
```

Vim自報

Vim scriptとvimrc読書会のアップデート

lambdaが使えるようになった

パッチ「7.4.2044」からlambdaが使えるようになりました。これまでsort()の比較処理を指定する際、直接「比較式」を書くことができなかったため、別途比較関数を用意する必要がありました。

```
function! s:CompareItem(lhs, rhs)
  return a:lhs["値段"] - a:rhs["値段"]
endfunction

let s:items = [
\ {"名前": "りんご", "値段": 100},
\ {"名前": "みかん", "値段": 180},
\ {"名前": "バナナ", "値段": 90},
\]

let s:items = sort(s:items, "s:CompareItem")
echo s:items
" 【結果】
"
" [{"名前": "バナナ", "値段": 90},
"  {"名前": "りんご", "値段": 100},
"  {"名前": "みかん", "値段": 180}]
```

lambdaの登場により上記のコードが**リストA**のように簡単に記述できるようになりました。->の前に引数を列挙し、後に式を記述できます。関数を用意しなくてもよくなったので便利にはなったのですが、lambdaは式(expression)しか書けません。文(statement)を使うためには本パッチの少し前の「7.4.2008」で入ったexecute()関数を使うのが便利です。lambda、execute()関数についてはもう少し説明が必要ですが、それはまた別の回で紹介します。

▼リストA　lambdaを使う

```
let s:items = [
\ {"名前": "りんご", "値段": 100},
\ {"名前": "みかん", "値段": 180},
\ {"名前": "バナナ", "値段": 90},
\]

let s:items = sort(s:items, {lhs,rhs->lhs["値段"] - rhs["値段"]})
echo s:items
```

vimrc読書会の開催場所が変更されました

毎週土曜夜11時からオンラインで開催されているvimrc読書会ですが、開催場所がlingrからgitterに変更となりました。

・vimrc読書会
https://gitter.im/vim-jp/reading-vimrc

gitterになったことで参加もしやすくなり、Botが改良されて読書会に適した機能が追加され、便利になりました。Vimに関する知識がそれほどない方でも参加していただいてかまいません。どうぞご参加ください。

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29回 カレントバッファを即実行! quickrun(前編)

カレントバッファやregionのコードをすぐに実行できる「quickrun」。プログラムを書いていて、その挙動をすぐに確かめたいといった場合に便利です。今回はquickrunの概要を説明したあと、処理系を選択しての実行や、helmとの連携などを紹介します。

M-x executable-interpret

前回までは4回に渡って、Emacs内でシェルコマンドを実行するあらゆる方法を紹介しました。今回はその流れに引き続いてquickrunパッケージを紹介しようと思います。その名のとおり、カレントバッファを「即実行」するものです。カレントバッファの実行と言えば標準機能のM-x executable-interpretがありますが、その進化形ともいえます。

M-x executable-interpretを実行するとカレントバッファのファイル名がプロンプトに出てきます。Enterを押すとそのまま実行でき(**図1**)、引数を入力することもできます。引数を入力したときは、以後同じ引数を使い回します。

また、M-x compile系列のコマンドですのでM-g M-n/M-g M-pによってエラー行にジャンプすることもできます。「#!」が指定してあるスクリプト言語においては十分便利な機能です。

▼図1 M-x executable-interpret

```
#!/bin/sh
echo Hello
date

echo bye

-:---  quickrun-sample.sh    All L1      (Shell-script[sh])
-*- mode: compilation; default-directory: "~/book/sd-emacs-ren
sai/" -*-
Comint started at Sat Jul  9 09:30:44

/r/book/sd-emacs-rensai/quickrun-sample.sh
Hello
2016年  7月  9日 土曜日 09:30:44 JST
bye
U:**-  *interpretation*    Top L1      (Comint:exit [0] Shell-Co
```

quickrunとは

便利な機能

使っているプログラミング言語がおもにスクリプト言語であれば、M-x executable-interpretでも十分かもしれません。けれどもquickrunは、コンパイル言語やマークアップ言語でさえも「即実行」できるよう進化しています。しかも、その実行には多くの方法が用意してあります。

quickrunには次の特徴があります。

- 40を超えるプログラミング言語に対応
- 60もの処理系について初期設定
- 自分用のコマンドや他言語にも対応可能
- regionのみを部分的に実行可能
  - —実行結果を直下にコメントで出力可能
  - —実行結果に置き換えることも可能
- 文法チェック
- 対話的コマンドはeshellを用いて実行可能
- 標準入力ファイルを用意して実行可能
- 一定時間(デフォルトで10秒)で終了しない場合は強制終了
- 出力方法を指定できる

▼表1 quickrunの対応言語(括弧内はEmacsで使うコマンド)

C(gcc/clang/cl)	C++(g++/clang++/cl)	Objective-C(gcc -objc)	D(dmd)
Fortran(gfortran)	Java(javac and java)	Perl(perl)	Perl6(perl6)
Ruby(ruby/mruby)	Python(python)	PHP(php)	Emacs Lisp(emacs)
Scheme(gosh)	Common Lisp(clisp/sbcl/ccl)	Clojure(jark/clj-env-dir)	Javascript(node/v8/js/jrunscript/cscript)
Coffee Script(coffee)	JSX(jsx)	Markdown(Markdown.pl/bluecloth/kramdown/pandoc/redcarpet)	Haskell(runghc)
Go(go/gccgo)	Io(io)	Lua(lua)	Groovy(groovy)
Scala(scala)	HAML(haml)	SASS(sass)	LESS(lessc)
Erlang(escript)	OCaml(ocamlc)	F#(fsharpc)	ShellScript(shebangによる)
AWK(awk)	Rust(rustc)	Dart(dart)	Elixir(elixir)
TypeScript(tsc)	Tcl(tclsh)	Swift(swift/xcrun)	ATS2(patscc)
R(Rscript)	Nim/NimScript(nim)	Julia(julia)	Gnuplot(gnuplot)

—ブラウザで表示させる
—エコーエリアで表示させる
—ファイルに出力させる
—バッファに出力させる
—変数に代入させる
—それらすべてを組み合わせ可能
—何も出力しないことも可能

・helmまたはanythingインターフェースですべての機能にアクセス可能

対応言語

quickrunは多くの言語に対応しています(**表1**)。Emacs Lispはバッチモードでの実行となります。Markdownなどのマークアップ言語の設定もあります。

とりあえず使ってみよう

そのままM-x quickrun

それでは、さっそくquickrunを使ってみましょう。`M-x package-install quickrun`でインストールしてください。おもな言語にはほぼ対応しています。ここではquickrun-sample.sh(**リスト1**)というシェルスクリプトを例にします。

引数も入力もないので、そのまま`M-x quickrun`を実行します。`M-x executable-interpret`と同じように実行結果が表示されましたが、フォーカスは*quickrun*ウィンドウにあります(**図2**)。そこでqを押せば*quickrun*ウィンドウは消滅します。実行中のプログラムの場合は`C-c C-c`で強制終了できます。

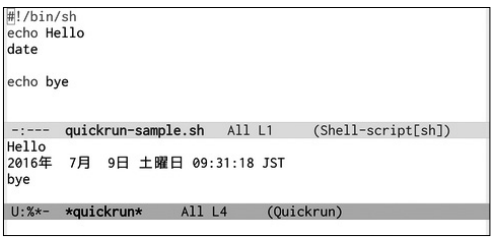
とはいえ、基本形だと`M-x executable-interpret`とあまり変わらないので、わざわざ導入するべきかどうか迷うと思います。それに`M-x executable-interpret`では引数も指定でき、*interpretation*バッファでは入力も受け付けるので、スクリプト言語においてはそれで十分ではないでしょうか。

もちろんquickrunを本稿で取り上げる以上、

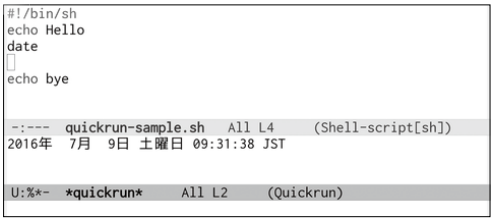
▼リスト1 quickrun-sample.sh

```
#!/bin/sh
echo Hello
date

echo bye
```

▼図2 M-x quickrun

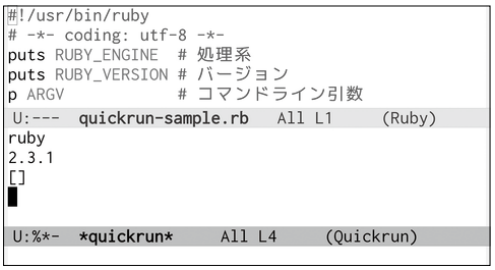


▼図3　dateの行を指定してM-x quickrun-region

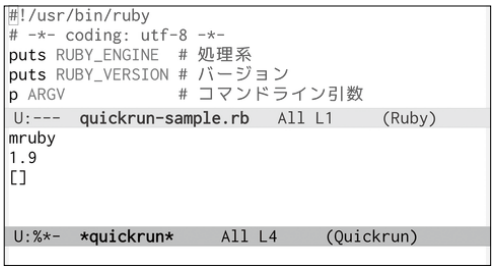


▼図4　dateの行を指定してM-x quickrun-eval-print

▼図5　そのままM-x quickrun



▼図6　C-u M-x quickrun RET ruby/mruby



劣化版ではありません。quickrunにはquickrunの哲学がありますので、どうか読み進めていただきたいと思います。

regionを実行する2つのコマンド

quickrunにはregionのみを実行する機能が存在します。

M-x quickrun-regionはregionのみを実行して結果を表示し(**図3**)、M-x quickrun-eval-printは結果をコメントとしてカレントバッファに挿入します(**図4**)。これらの機能はシェルスクリプトでとくに有用です。

quickrunの応用機能

C-u M-x quickrunでほかの処理系で実行

同じ言語でも、複数の処理系が存在するものがあります。たとえば、Rubyにおいては本家Ruby以外にもmrubyという処理系がquickrunで登録されています。ここでは処理系・バージョン・コマンドライン引数を表示するRubyスクリプトquickrun-sample.rb(**リスト2**)を例にして実行してみます(**図5**)。ほかの処理系で実行させるには、C-u M-x quickrunを使います(**図6**)。

▼リスト2　quickrun-sample.rb

```
#!/usr/bin/ruby
# -*- coding: utf-8 -*-
puts RUBY_ENGINE  # 処理系
puts RUBY_VERSION # バージョン
p ARGV            # コマンドライン引数
```

C-u C-u M-x quickrunで文法チェック

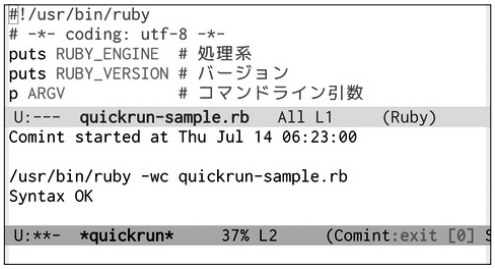
C-u C-u M-x quickrunあるいはM-x quickrun-compile-onlyで文法チェックを行います。

Rubyのようなスクリプト言語の場合は文法が正しいかどうかの静的なチェックを行うだけですが(**図7**)、コンパイラ言語の場合は実際にコンパイルしてエラーが出るか否かをチェックします。コンパイラ言語の文法チェックにはM-x compileのインターフェースを使っており、M-g M-n/M-g M-pでエラー行にジャンプできます。

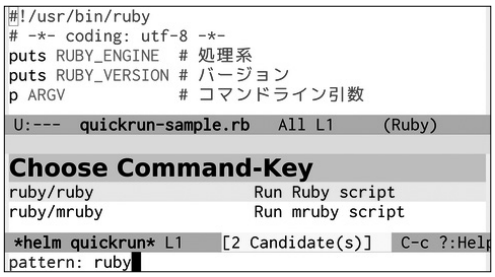
引数を付けて実行する M-x quickrun-with-arg

quickrunにおいて、コマンドライン引数を指定するにはM-x quickrun-with-argを使う必要があります。先ほどのquickrun-sample.rbで「foo bar」というコマンドライン引数を付けて実行すると、**図8**のような結果になります。

▼図7 C-u C-u M-x quickrun



▼図9 M-x helm-quickrun



helm/anythingインターフェース

quickrunには、anythingとhelmのインターフェースが用意されています。

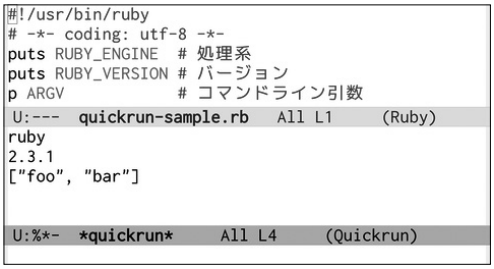
anythingは筆者が9年前から開発・メンテナンスしているパッケージで、強力な絞り込み検索と複数のアクションで1つのコマンドが無数の機能を持つようになり、話題となりました。

helmはanythingの後継版で、とても活発に開発されている超人気パッケージです。本連載でも第11・12回(本誌2015年3月号・4月号)で紹介しました[注1]。

M-x helm-quickrunあるいはM-x anything-quickrunからはquickrunの全機能にアクセスできます。実行すると、まずすべての処理系が候補となって登場しますので、使いたい処理系を絞り込んでください(図9)。そのままEnterを押すとその処理系でquickrunが実行されます。TABを押すとアクションリストが出てきて、多種多様な実行方法を選択できます(表2)。

注1) anythingは、筆者としては完成していると思っているので、かなり前に機能拡充を停止しました。helmは仕様変更が激し過ぎていきなり動かなくなることがよくあるため、anythingは安定志向・elispプログラミングツールとして再興予定です。anythingはまだ死んでいません。

▼図8 M-x quickrun-with-arg foo bar



▼表2 helm-quickrunのアクション

アクション名	相当するコマンド
Run this cmd-key	quickrun
Compile only	quickrun-compile-only
Run with shell	quickrun-shell
Run with argument	quickrun-with-arg
Replace region	quickrun-replace-region
Eval and insert as comment	quickrun-eval-print

◆ ◆ ◆

ここまでquickrunの大まかな機能を紹介しました。次回はquickrunの実行を細かく制御したり、カスタマイズする方法を紹介する予定です。

quickrunは開発中のプログラムをすばやく実行するためのあらゆる方法を提供します。筆者のサイトでもquickrunの紹介[注2]をしています。また、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]も紹介しています。

毎日更新している筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指しています。手元でgrep検索できるよう全文をGitHubに置いています。またEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでもお答えします。「こんなパッケージ知らない?」「挙動がおかしいからなんとかしてよ!」はもちろんのこと、自作elispプログラムの添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

注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18回　APFSとSwift3

Writer 小飼 弾（こがい だん）　twitter @dankogai

APFS――3度目の正直

いよいよSwift 3の解説……の前にOne More Thing。「新製品なきWWDC」と前回言いましたが、実は正真正銘の新製品が1つありました。APFS[注1]。現在のHFS+[注2]に代わる新しいファイルシステムです。ファイルシステムというものの重要性を考えれば、これをスルーするわけにはいかないでしょう。

ファイルシステムの歴史

Apple製品のファイルシステムと言えば、1998年以来一貫してHFS+でした。Jobsの復帰は、技術的には買収元であるNeXTが買収先であるAppleを乗っ取っていく歴史でもあり、macOSは（9までの）Mac OSではなくNextStepの子孫なのですが、唯一乗っ取れなかったのがHFS+です。

HFS+が受けた最初の挑戦が、NextStep由来のUFS(Unix File System)であるのはごく自然とも言えます。OS X登場当時はUFSがHFS+を置き換えると思われていたのですが、HFS+はその挑戦を退けます。HFS+にはユーザごとの所有権やパーミッションといった、OS XというUnixに必要な機能をすべて備えていたからです。UFSにあってHFS+にないのは、スパースファイルやファイル名の大文字小文字の区別ぐらいで、それらは必須ではなかったのです。

次の挑戦はZFSの登場でした。ZFSはこれまでのファイルシステムの常識を覆す画期的なファイルシステムでした。fsckを不要にするトランザクション、パーティションという概念を過去のものにするデータセット、ファイルシステム自体のundoを可能にするスナップショット、エラーを自動検知し、可能であれば自動修復するチェックサム、RAIDホールがないRAID-Z……「Z=最後のファイルシステム」という自信がその名に込められたZFSは、今は亡きSun Microsystemsの最後の遺産でもあります。

ZFSオープン化の理由

画期的だったのは技術にとどまりません。同社はZFSをオープンソースしたのです。おかげでSolaris以外のOSにも移植が進みました。最も熱心だったFreeBSDは、今やZFSが最大の売りの1つにもなっていますし、ライセンスがGPLと非互換ということで公式サポートできないはずのLinuxすら、カーネルモジュールをカーネル本体とは別配布にすることで利用可能になっています。そしてOS Xも、いったんは公式サポートしたのです。OS X v10.5にはリードオンリーとはいえZFSが追加されました。しかしAppleは、v10.6でZFSサポートを

注1） APFS（https://developer.apple.com/videos/play/wwdc2016/701/）
注2） https://en.wikipedia.org/wiki/HFS_Plus

打ち切ってしまいます。

例によってAppleはその理由をつまびらかにしていませんが、最も人気がある仮説はSunがOracleに買収されたことで、OracleがAppleへのライセンスを蹴ったというものです[注3]。かなりの説得力もあり、筆者自身ほとんど説得されていた一方、ZFSがオープンソースであり、OS Xを含む複数OSのサポートがOpenZFS[注4]として結実していることを考えると鵜呑みにし難くもあります。

HFS+がUFSとZFSを退けた話は、拙著『コードなエッセイ[注5]』も取り上げたのでそちらでも確認していただくとして、APFSの登場で、なぜZFSではダメだったのか、筆者にもやっと納得できる理由が得られたように思います。

HFS+をZFSにライブアップグレードするのは不可能だからです。技術的にはさておき、既存のAppleデバイスのユーザが受け入れられる形では。

ファイルシステムのライブアップグレードは、大きなところでは二度行われています。1つは1998年の、MacにおけるHFSからHFS+へのアップグレード。もう1つはWindowsにおけるFATからNTFSへのアップグレード。どちらも共通しているのは、新ファイルシステムが旧ファイルシステムのすべての機能を上位互換としてサポートしていること、そしてアップグレードするのはメタデータのみで、データはそのままだということ(**表1**)。

たとえば最長ファイル名。ZFSは255バイトなのに対し、HFS+は(実はNTFSも)UTF-16で255文字。バイト長で倍も違います。255バイトでも長過ぎに一見思えますが、Twitterのつぶやきをそのままファイル名にペーストするユーザがいないと誰が断言できるでしょうか。

その一方、ZFSではストレージに書き込む情報すべてにチェックサムを付けていますが、APFSではメタデータのみ。もしZFS同様にAPFSもデータのチェックサムを付けようとするとどうなるか？　ファイルシステムのアップグレードの際に、使用ブロックの情報をすべて読む必要があります。それはアップグレード

▼表1　ファイルシステムの機能比較

機能	HFS+	ZFS	APFS	コメント
Copy-on-write	×	○	○	新世代FSの要
スナップショット	×	○	○	
クローン	×	○	○	
RAID	×	○	×	
データセット	×	○	○	既存FSのパーティション相当
チェックサム	×	○	△	APFSはメタデータのみ
暗号化	○	x	◎	
SSD最適化	△	△	○	
タイムスタンプ粒度	秒	ナノ秒	ナノ秒	
最長ファイル名	UTF-16で255文字	255バイト	HFS+以上	ZFSでは互換性不足
旧FSからのアップグレード	NA	×	○	ZFSでは困難

注3) http://dtrace.org/blogs/ahl/2016/06/15/apple_and_zfs/
注4) http://open-zfs.org/wiki/Main_Page
注5) 『コードなエッセイ』当社刊行、ISBN:978-4774156644

に数分ではなく数時間を要するということであり、古くからのパソコンユーザなら何とか耐えられてもスマフォやタブレットのユーザにそこまでの忍耐力は期待できないでしょう。

APFSの真意

APFSのことを最初に耳にした時点での筆者の感想は、「それってZFSと何が違うの?」でした。とくにその要となっているのが**表1**で挙げたようにcopy-on-writeがZFSの真髄でもあります。しかしAppleははっきり表明しました。「18ヵ月後、すべてのAppleデバイスのデフォルトファイルシステムをAPFSに移行する。既存デバイスはそのままアップグレードする」と。

あらためて見てみると、ZFSとAPFSは要となっている技術は共通していても、ユースケースは180度異なるとも言えます。ZFSが威力を発揮するのは、サーバの大規模ストレージ。APFSはパーソナルデバイスの内部ストレージ。ZFSにとってのSSDの役どころはディスクアレイのキャッシュ、APFSにとってのSSDは唯一のストレージ。ZFSにとって暗号化は「あればうれしい」機能、APFSにとって暗号化は「欠かせない機能」……。

APFSはオープンソース化されない?

1つ気になるのは、「APFSはApple製品に最適化」されていることを強調していること。これまでAppleはOSの基礎部分をDarwinとしてオープンソースしてきました[注6]。HFS+まわりのコードもその一環として公開されています。ファイルシステムなきOSというのが現状ありえない以上当然とも言えますが、HFS+とAPFSのどちらでもブートできるとしたら、APFSのコードはDarwinに含める必要はなくなります。

残念ながらその可能性は低くないでしょう。残念**でない**ことに、ファイルシステムの違いはネットが当然のように吸収してくれます。フロッピー、CD-ROM、DVD……リムーバブルドライブはMacからも消えてしまいました。そしてUSBメモリすら、iPhoneやiPadにはそのまま刺さらない。そしてそれをほとんど誰も気にしないとあっては、Apple製品に最適化されたファイルシステムをサポートするインセンティブはさほど高くはない。オープンソースなHFS+すら、Appleの外ではほとんどサポートされているとは言えないのに……。

それでも、Apple製品向けにソフトウェアを開発する我々は、APFSの機能を学ばざるを得ないでしょう。とくにクローンやスナップショットのようなHFS+にはない特長を活かすとしたら。1つ確かなのは、APFSのAPIはSwiftでも提供されるであろうということ。どんなふうになるか、今から楽しみです。

swift.version++ // 廃止される機能

それではいよいよSwift 3の紹介を。といっても、実はSwift 2.2をお使いの皆さんはすでに触れているといっても過言ではありません。Swift 3で廃止が予定されている機能に関しては、すでに警告が出るからです。Swift 3へすぐに移行せずにとりあえずSwift 2.3にとどまるにしろ、警告は今のうちに消しておきましょう。

++ / --

おそらく一番知られているのは[SE-0004][注7]。ほとんどのケースで、+= 1とするだけで対処できるでしょう。どうしても欲しかったら、あらためてオレ演算子として定義してしまえば良

注6) https://opensource.apple.com
注7) [SE-0004]https://github.com/apple/swift-evolution/blog/master/proposals/0004-remove-pre-post-inc-decrement.md

いのです。Swift 3でも警告なしで使えます。

```
prefix func ++(i: inout Int) -> Int {
    i += 1
    return i
}

postfix func ++(i: inout Int) -> Int {
    let o = i
    i += 1
    return o
}
```

C-Styleのfor

［SE-0007］[注8]もSequenceType、あらためSequenceプロトコルがあるSwiftではほとんど不要でしょう。

```
for var i = 0; i < 10; i++ {
  let elem = array[i]
  // …
}
```

より、

```
for elem in array {
  // …
}
```

のほうがはるかにわかりやすいですし、iも欲しければ、

```
for (i, elem) in array.enumerate() {
  // …
}
```

とするだけです。

パラメータvar

［SE-0003］[注9]のパラメーター中のvar禁止とはどういうことかというと、こういうコードが書けなくなるということです。

・Swift2

```
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
}
```

varしたかったら、ブロック内であらためてやれ、と。

・Swift(2|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
}
```

カリー化構文

「インド人は右へ」ではなくて、［SE-0002］[注10]は次のようなコードが書けなくなるということです。

・Swift2

```
func logWithBase(b:Double)(_ x:Double)-
>Double {
    return log(x)/log(b)
}

let log2 = logWithBase(2)
print(log2(8)) // 3.0
```

Swiftにはブロックがあるので、こう書けばOKですし、そのほうがわかりやすいでしょう。

・Swift(2|3)

```
func log(base b:Double)->(Double)->Double {
    return { x in log(x)/log(b) }
}

let log2 = log(base:2)
print(log2(8)) // 3.0
```

注8）［SE-0007］https://github.com/apple/swift-evolution/blog/master/proposals/0007-remove-c-style-for-loops.md
注9）［SE-0003］https://github.com/apple/swift-evolution/blog/master/proposals/0003-remove-var-parameters.md
注10）［SE-0002］https://github.com/apple/swift-evolution/blob/master/proposals/0002-remove-currying.md

実際のところ、カリー化構文を――とくにプロダクションコードで――使っている方は少ないかと思われますが念のため。

パラメータのタプル渡し

ほとんど知られていなかった機能ですが、Swift 2では次のコードがOKでした。

・Swift2

```
func distance(_ x:Double, _ y:Double)-
>Double {
  return sqrt(x*x + y*y)
}
let p = (3.0, 4.0)
print(distance(p))// 5.0
```

パラメータ全体をタプルにして、そのタプルを1つ渡すとすべてのパラメータを渡したのと同等で、言語的には一貫してはいるのですが、バグを引き寄せやすい機能ということで、[SE-0029][注11]で廃止されることになりました。

・Swift(2|3)

```
func distance(_ x:Double, _ y:Double)-
>Double {
  return sqrt(x*x + y*y)
}
let p = (3.0, 4.0)
print(distance(p.0, p.1)) // 5.0
```

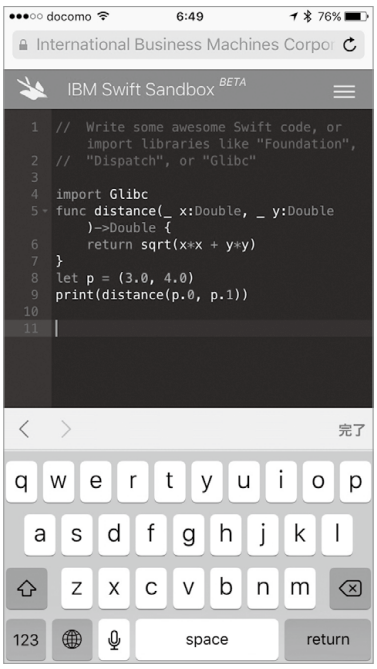
次回に向けての予習

というわけでSwift 3で廃止される機能の紹介だけで今号の誌面が尽きてしまったようですが、オープンソース化されたおかげで、Swift 3の変更はGitHubの**swift-evolution**[注12]でも事前に確認できます。さらにうれしいことに、すでに実装済みの機能に関しては、**IBM Swift Sandbox**[注13]で実際に動かしてみることもできます(**図1**、**図2**)。

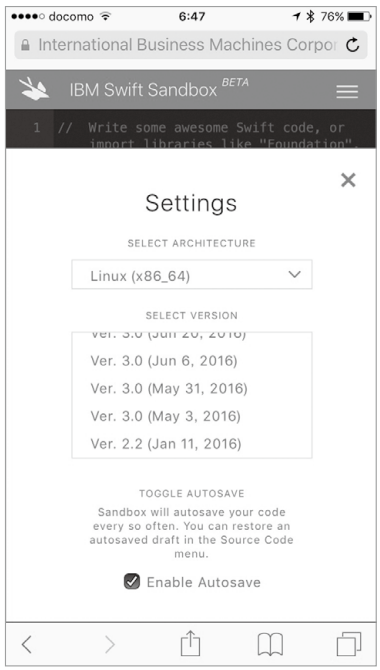
パソコンはもちろん、スマートフォンからでも。Swiftのバージョンを切り替えて試せる点も素晴らしい。

というわけで、次回はいよいよSwift 3の核心に迫っていく予定です。SD

▼図1　IBM Swift Sandboxでの実行例（コード編集）



▼図2　IBM Swift Sandboxでの実行例（複数バージョンのSwiftを指定）



注11) [SE-0029]https://github.com/apple/swift-evolution/blob/master/proposals/0029-remove-implicit-tuple-splat.md
注12) swift-evolution(https://github.com/apple/swift-evolution)
注13) IBM Swift Sandbox(https://swiftlang.ng.bluemix.net/#/repl)

Sphinxで始めるドキュメント作成術

清水川 貴之　SHIMIZUKAWA Takayuki　Twitter @shimizukawa

第18回 ドキュメントを自動生成するautodoc

ドキュメント自動生成における課題

今回は、Pythonソースコードから自動的にAPIリファレンスを生成する、autodocについて紹介します。

もともとSphinxの開発は、プログラミング言語Pythonのドキュメントを作るうえで発生したさまざまな課題を解決するために始まりました。そのときの課題の1つが、ソースコードからのドキュメント自動生成でした。当時のPythonのドキュメントは、Pythonのソースコードに書かれているdocstring(ドキュメント文字列、**リスト1**の①のように書きます)をツールでパースして、LaTeXに埋め込める形式に変換していました[注1]。

docstringを元にドキュメントを自動生成するツールは、Sphinx以外にもいくつかあります[注2]。これらのツールは、APIリファレンスの自動生成が主目的で、チュートリアルや機能別のドキュメントなど、リファレンス以外のナラティブ[注3]なドキュメントの生成には不向きです。

APIリファレンスだけでなく、チュートリアルなどのドキュメントもあったほうがうれしいですよね？　Sphinxを使えば、これまでに本連載で紹介してきた方法で、ナラティブなドキュメントを書けます。さらに、Sphinx本体に同梱されているSphinx拡張sphinx.ext.autodocを使えば、docstringから自動生成されたリファレンスをドキュメントの好きな位置に埋め込めます。

autodocを活用したドキュメント作り

Python製のツールやライブラリのドキュメントは、その多くがSphinxで作成されています。autodocを使うと、とても簡単にAPIリファレンスを作成できるのがその理由の1つでしょう。はじめに、autodocを活用したドキュメントはどんな出力になるのか、そのときのソースはどう書かれているのかを紹介します。

多くのライブラリのドキュメントは**図1**のように、ページ全体の目的を説明するナラティブ

▼リスト1　docstring

```
def fib(n):
    """
    n番目のフィボナッチ数を返します。以下は実行例です。

    >>> fib(5)          ┐②
    8                   ┘
①
    :param int n: n番目のフィボナッチ数を指定。nは0以上の整数。  ┐
    :return: n番目のフィボナッチ数。                              ├③
    :rtype: int                                                  ┘
    """
    return 1 if n < 2 else fib(n - 2) + fib(n - 1)
```

注1）ドキュメントはPDF版とHTML版とがあり、HTMLはさらに別のツールでLaTeXからHTMLに変換していたそうです。 http://sphinx-users.jp/event/20121200_sphinx-advent-calendar/day25-sphinx-past-and-future.html

注2）Sphinxが登場する以前から、Pydoc、Epydocなどがありました。Python以外の言語ではJavadoc、Doxygen、ESDoc、RDocなどがあります。

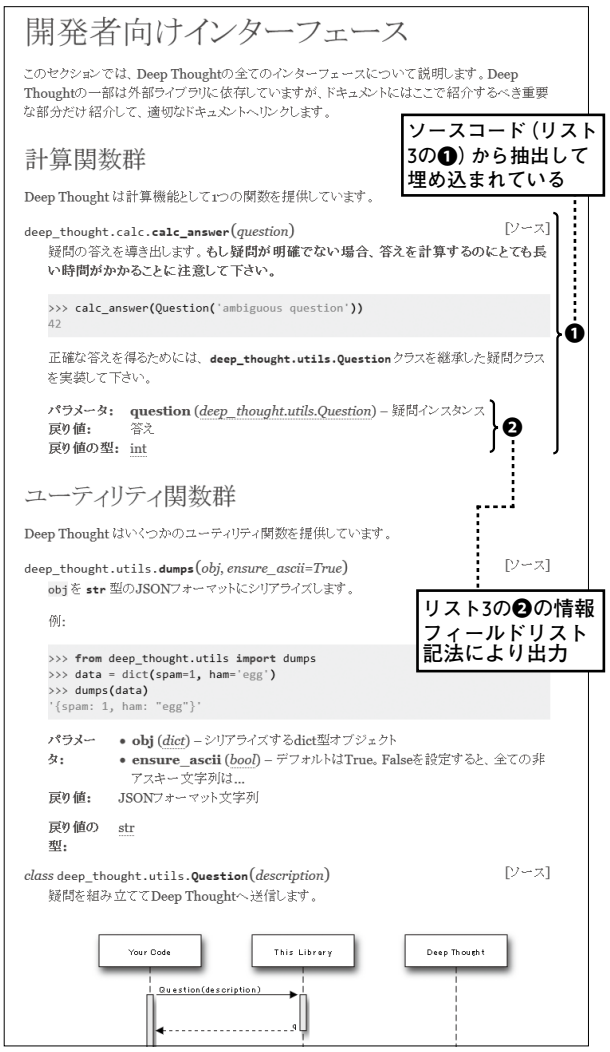
注3）説明的、物語的。ストーリーのあるドキュメント形式。「リファレンス」(辞書的)の対義語。

な部分と、関数やクラスを説明するリファレンス部分とで構成されます。

このようなページを出力するには、**リスト2**のようなドキュメントソースを書きます。このファイルでは、autodocが提供しているautofunctionとautoclassディレクティブを使用しています。これらを使うと、関数やクラスの定義(シグネチャ)と説明文(docstring)をソースコードから自動的に抽出して、ドキュメントに埋め込んでくれます。**リスト2**の❶のautofunctionで参照しているdeep_thought.calc.calc_answerのソースコードには**リスト3**のように書かれています。

一般的に、ドキュメントの自動生成は、ソースコードに書いたのと同じことをドキュメントにも書く煩わしさから解放してくれます。しかし、自動生成だけで作成されたドキュメントはたいていの場合、巨大で読みづらい固まりになってしまい、よく知っているソフトウェアでなければ読むのが困難なものになります。また、ソースコードと直接関係がないドキュメントは書けません。

▼図1　autodocを利用して作成したドキュメント例(一部抜粋)



▼リスト2　interface.rst

```
==================================
開発者向けインターフェース
==================================

このセクションでは、Deep Thoughtの全てのインターフェースについて説明します。
Deep Thoughtの一部は外部ライブラリに依存していますが、ドキュメントには
ここで紹介するべき重要な部分だけ紹介して、適切なドキュメントへリンクします。

計算関数群
====================

Deep Thought は計算機能として1つの関数を提供しています。

.. autofunction:: deep_thought.calc.calc_answer  ←❶

ユーティリティ関数群
========================

Deep Thought はいくつかのユーティリティ関数を提供しています。

.. autofunction:: deep_thought.utils.dumps
.. autoclass:: deep_thought.utils.Question
```

自動生成ドキュメントとは反対に、すべてを手書きで記述する場合、全体を自由に構成できます。ソースコードに書くのが適切ではないようなナラティブな記述、たとえば、チュートリアル、インストール手順、システム構成などもドキュメントに含められます。デメリットとして、関数シグネチャから引数の説明まですべてを手書きでドキュメント側に書く必要があります。そしてこれは、関数の説明は関数のコードの近くに書いておきたい、というプログラマの直感に反しています。実際の問題として、関数の説明をコードの近くに書いておかないと、関数仕様の変更があったときに、その変更内容を漏れなく説明に反映することが難しくなります。

autodocを使うと、ドキュメントの自動生成と手書きのそれぞれの良いところを同時に享受できます。ドキュメント全体の構造は手で書きます。そのため、必要に応じて全体像や使い方の解説文を書き、読者の観点からAPIの説明をわかりやすく並べて提供できます。各APIの詳細な説明は、ソースコードのdocstringから自動的に取得してドキュメントに埋め込みます。

autodocのための docstring入門

docstringはPython 1系のころから言語機能として組み込まれています[注4]。**リスト1**の①のよ

注4）https://docs.python.org/release/1.6/tut/node6.html

▼リスト3　deep_thought/calc.py

```
def calc_answer(question):
    """
    疑問の答えを導き出します。 **もし疑問が明確でない場合、答えを計算するのに
    とても長い時間がかかることに注意して下さい。**

    >>> calc_answer(Question('曖昧な質問'))
    42

    正確な答えを得るためには、:py:class:`deep_thought.utils.Question` クラスを
    継承した疑問クラスを実装して下さい。

    :param deep_thought.utils.Question question: 疑問インスタンス
    :return: 答え
    :rtype: int
    """
    time.sleep(750 * 10000 * 365 * 24 * 60 * 60)
    return 42
```

❶ ❷ ❸

COLUMN

doctestでドキュメント更新漏れを防ぐ

docstring内には、Pythonコードの対話形式での実行例（DocTest記法、**リスト1**の②）を記述できます。これによって、ユーザはその関数をどのように実行すればどんな結果を得られるのか、簡単に把握できます。また、DocTest記法で実行例を書いておくと、doctestと連携できます。doctestはPythonの標準ライブラリです。これを使用すると、DocTest記法を検出して、実行し、期待する値と実際の結果が一致しているかを検査してくれます（**図A**）。doctestによって、そのドキュメントが関数の実装そのものとかけ離れてしまっていないか、関数仕様が変わったときにドキュメントの更新を忘れていないかをチェックできます。

▼図A　doctest実行例

```
$ python -m doctest -v fib.py
Trying:
    fib(5)
Expecting:
    8
ok
1 passed and 0 failed.
Test passed.
```

うに関数やクラスのシグネチャの次の行に、インデントして文字列リテラルを1行または複数行で記述すれば、Pythonインタプリタがこれをdocstringとして解釈します。docstringとは何か、そこにどんな情報が含まれるべきかについてはPEP-0257[注5]で決められていますが、docstringにどんな記法が使われるべきかは決められていません。

Pythonの言語仕様では、たとえばfib関数のdocstringは、`import inspect; inspect.getdoc(fib)`のようにして取得できます。また、Pythonインタラクティブシェルで`help(fib)`と入力すれば、docstringが表示されます。

ほかにもPythonに対応したエディタや統合開発環境では、関数のドキュメントを簡単に見るための機能が提供されているなど、docstringはさまざまなツールで活用されています。

Sphinxは、docstringに書かれたテキストをreSTとして解釈します[注6]。前節で紹介したように、docstringに書かれた内容は自動的にソースコードから抜き出されてドキュメントに出力されます。reSTで書けるので、強調の記法やリンクの記法なども使用でき、HTMLなどに出力したときにより便利な内容を提供できるでしょう。

ただし前述のとおり、docstringの記法は決まっていません。Sphinx以外にも`help(fib)`のように参照したり、統合開発環境で表示したりする場合もあり、そういったツールはdocstringのreST記法を解釈しないでしょう。docstringがそのまま表示されても読みやすいように、マークアップし過ぎないようにしましょう。

Sphinxでは、DocTest記法(p.141のコラム参照)で記述した利用例も自動的にコードハイライトされます。また、doctestビルダーは、DocTest記法を一括で検査する機能を提供しています。doctestビルダーを使用するには、Sphinx同梱のsphinx.ext.doctestを有効にしてください。

注5) https://www.python.org/dev/peps/pep-0257/

注6) Googleスタイルの記法を解釈するsphinx.ext.napoleonもあります。
http://www.sphinx-doc.org/ja/stable/ext/napoleon.html

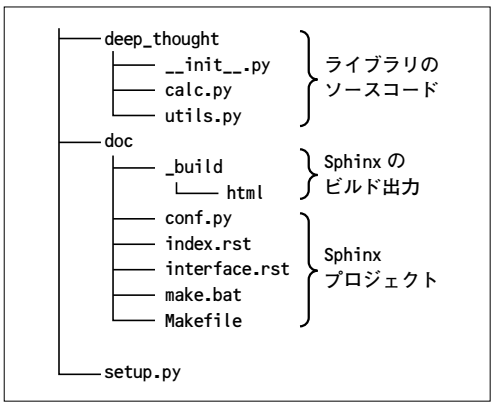
autodocでdocstringの価値を引き出そう

autodocを利用するための設定と、ドキュメントの書き方、そしてautodocのしくみについて紹介します。

autodocを利用するには、conf.pyでsphinx.ext.autodocを有効にしたうえで、Sphinxが対象のPythonソースコードを読み込める必要があります。Pythonのモジュールインポートパスが通っていればどこにあっても良いのですが、たいていは**図2**のようなディレクトリ構造で、自作ライブラリのドキュメントにautodocを使用すると思います。**リスト4**では、sys.pathにconf.pyからみた親ディレクトリを設定して、deep_thoughtパッケージをPythonでimportできるように調整しています。また、autodocを有効にしています。これでautodocを使用する準備が整いました。

あとは、**リスト2**の❶のようにautofunction、autoclassといったディレクティブを使ってド

▼図2　プロジェクトのファイル構成



▼リスト4　conf.pyでautodocを有効化

```
import os
import sys

sys.path.insert(0, os.path.abspath('..'))

extensions = [
   'sphinx.ext.autodoc',
]
```

キュメントに埋め込みたい関数やクラスを指定します。そこまで実施できたら、make htmlでHTMLドキュメントを生成して確認してみましょう。**図1**の❶ような出力が得られます。

autofunctionは、Pythonの関数定義とdocstringから、**リスト5**のようなPythonドメイン[注7]（前回紹介）を活用した記述（reST）を内部的に自動生成します。autodocのディレクティブを使えば、ドキュメントを書く時間を大幅に節約できるでしょう。なお、自動生成したreSTはメモリ内にのみ一時生成されるため、内容を知りたい場合はmake html SPHINXOPTS=-vvvのように実行すれば、確認できます。

関数引数の記述

関数の引数について長々とした文章で説明を書き綴っても、わかりやすいドキュメントにはなりません。引数の説明だということが明確で、どんな型を期待していて、値にどのような意味があるのかが端的に書かれているべきです。

Sphinxでは、関数引数を端的に記述するために情報フィールドリスト記法[注8]を提供しています。これを使うと、整形された読みやすいフォーマットで出力されます（**図1**の❷）。

リスト1の③は情報フィールドリスト記法を使って、引数と戻り値について説明しています。この記法は関数などの説明文でのみ使用できます（正確には、Pythonドメインのディレクティブ内で使用できます）。

情報フィールドリストは:param <型> <引数名>: <説明文>のように書きます。このとき、<型>部分に一致する型が見つかった場合、それが定義されているドキュメントのページへ自動的にリンクが設定されます。ある程度大きなライブラリであれば、そのライブラリが提供するクラスを引数にとる関数などもあるでしょう。そのとき、クラスがどこに定義されているのか、どうやって値を用意するのか、といった説明にリンクされていれば、そのドキュメントはとて

注7）http://www.sphinx-doc.org/ja/stable/domains.html#the-python-domain

▼リスト5　pythonドメインのディレクティブでAPIドキュメントを記述

```
.. py:module:: deep_thought.calc

.. py:function:: calc_answer(question)
   :module: deep_thought.calc

   疑問の答えを導き出します。（後略）
```

注8）http://www.sphinx-doc.org/ja/stable/domains.html#info-field-lists

COLUMN

docstring自動取得の落とし穴

SphinxはPythonが解釈した結果を使ってdocstringを取得しています。このため、対象のソースコードはSphinxドキュメントのビルド時にPythonによって読み込まれます。

importだけで利用環境に影響を与えるようなプログラムをautodocの対象としてはいけません。そのコードが、現在のディレクトリ以下のファイルをすべて削除するような内容の場合、make htmlを実行するとファイルが消えることになります。

一般的に、Pythonモジュールをimportするだけで副作用のある処理を実行するようなコードを書くべきではありません。if __name__ == '__main__':のような実行ガードを記述することで、import時によけいな処理を走らせないようにしましょう（**リストA**）。

▼リストA　if __name__ == '__main__':の使用例

```
import os

def delete_current_dir():
    os.system('sudo rm -Rf .')

if __name__ == '__main__':
    delete_current_dir()
```

このモジュールがimportされた場合、ここは実行されない

も読みやすくなります。

リスト3の❷ではライブラリが提供しているdeep_thought.utils.Questionクラスへリンクされます。**リスト1**の③のような組み込み型やほかのライブラリが提供する型にリンクするにはintersphinx(下記コラム参照)を使ってください。

関数名の参照

ライブラリで定義した関数やクラスは、ライブラリ内のほかの関数などの引数として使われることがあります。たとえば、**リスト3**のcalc_answer関数はdeep_thought.utils.Questionクラスのインスタンスを引数に受け取ります。ドキュメントでは、Pythonドメインのロールを使って、Questionクラスの説明にジャンプできるように書くと良いでしょう。**リスト3**の❸では`:py:class:`で、`.. py:class::`ディレクティブで定義したドキュメントへリンクさせています。同様に、`:py:func:`ロールを使えば`.. py:function::`ディレクティブで記述したドキュメントへの参照を書けます。

説明文中にリファレンスの定義個所へのクロスリファレンスを簡単に設定できるのが、Sphinxの魅力の1つです。autodocを利用すると、定義側のドキュメントはソースコードから自動生成されるため、参照する側のドキュメントを書くことに集中できます。

チュートリアルのようなナラティブなドキュメントを書く場合、使い方などの流れを重視してストーリー仕立ての文章を書きます。しかし、用語や定義の説明を都度挟んでしまうと、文章の流れが分断されてしまいます。ここで紹介した関数やクラスの参照を使えば、細かな定義はリファレンスに任せられ、読みやすいドキュメントを作れるでしょう。

複数のAPIリファレンスページを自動生成

もしライブラリにAPIがたくさんあって、それをすべてドキュメント提供しなければいけない場合、多くの手作業が必要になります。autodocはソースコードの関数定義からドキュメントを自動生成してくれましたが、その機能を使うには、autofunctionなどのディレクティブを使って対象関数やクラスなどをひとつひとつ指定しなければいけません。

モジュール(.pyファイル)内の関数やクラスをすべてドキュメントに埋め込むautomoduleディレクティブを使えば作業は楽になります。**リスト6**はdeep_thought.calcモジュールをautodocで取り込むときの例です。ディレクティブの`:members:`オプションは、対象モジュール内の

COLUMN

intersphinx

intersphinxはSphinxプロジェクトを越えて、別のドキュメントに書かれている定義個所へ自動的にリンクする機能です[注A]。Pythonの組み込み型やdjangoのクラスなど、ほかのSphinxドキュメントで定義されたデータ型に自動的にリンクするには、Sphinx内蔵のsphinx.ext.intersphinxを有効にします。そのためにはconf.pyに**リストB**のように設定してください。

注A) http://www.sphinx-doc.org/ja/stable/ext/intersphinx.html

▼リストB　conf.pyでintersphinxを有効化

```
extensions = [
   'sphinx.ext.intersphinx',
    ...  # 他の拡張利用の指定
]
intersphinx_mapping = {'python': ('https://docs.python.org/3.4', None)}
```

どの関数やクラスをドキュメントに取り込むのかを指定するためのものです。:members:とだけ書いた場合、docstringのある通常の関数／クラスなどが対象となり、:members: calc_answerのように書いた場合、明示したオブジェクトだけが対象となります。:members:オプションを書かない場合、モジュール自体のドキュメント（モジュール先頭のdocstring）だけが対象となります[注9]。

このディレクティブを使えば、労力はだいぶ削減されます。しかし、automoduleディレクティブを1つの.rstファイルにいくつも書くと、1つのページにたくさんのリファレンスが出力されてしまいます。たとえば、10個のモジュールを対象とする場合、各モジュールが10個前後の関数などを持っていたら、合計100個ほどのリファレンスが1ページに出力されるでしょう。だからといって、10個の.rstファイルを用意してそれぞれに**リスト6**のような記述だけを書く作業は、繰り返したくありません。

Sphinxは、この問題を解決する組み込み拡張sphinx.ext.autosummaryを提供しています。autosummaryは、指定されたモジュールそれぞれを個別のページとして生成する機能を提供します。

autosummaryを使うには、conf.pyに**リスト7**のように設定してください。そして、各モジュールの目次となる文書を作成します。たとえばapi.rstに**リスト8**のように対象としたいモジュールやパッケージを列挙します。あとはmake htmlを実行すれば各モジュールそれぞれのリファレンスページが生成されます。また、api.rstから生成されたapi.htmlには、autosummaryディレクティブに指定した各モジュールのリファレンスページへのリンクが提供されます（**図3**）。

まとめ&次回予告

autodocによって、Pythonソースコードから自動的にAPIリファレンスを生成できます。これによりドキュメントを書く人は手書きで書く際の退屈な繰り返し作業から解放されます。そして、読者に読みやすいドキュメントを楽に提供できるでしょう。

これまでの本連載では、Sphinxの機能は汎用的に使えることを強調してきましたが、今回のautodocはPythonソースコードとの連携でのみ利用できます。将来的には言語の垣根を越えて使える機能にしていきたいところですね。

次回はSphinxを使ったWeb APIのドキュメントの書き方を紹介します。SD

注9）オプションの詳細は以下を参照。
http://www.sphinx-doc.org/ja/stable/ext/autodoc.html

▼リスト6　automoduleでモジュール内のドキュメントを一括生成

```
.. automodule:: deep_thought.calc
   :members:
```

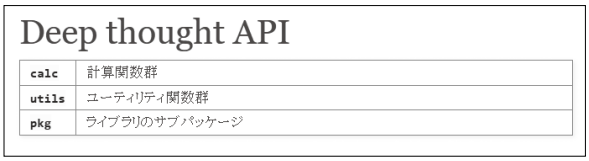
▼リスト8　autosummaryの利用例（api.rst）

```
Deep Thought API
================

.. autosummary::
   :toctree: generated

   deep_thought.calc
   deep_thought.utils
   deep_thought.pkg
```

▼リスト7　conf.pyでautosummaryを有効化

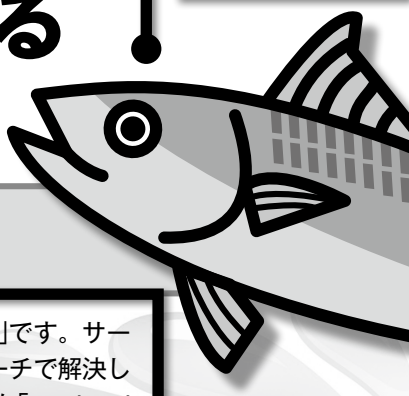
```
extensions = [
    ...  # 他の拡張利用の指定
    'sphinx.ext.autodoc',
    'sphinx.ext.autosummary',
]
autodoc_default_flags = ['members'] # すべてのautodocディレクティブにオプションを設定する
autosummary_generate = True  # autosummaryディレクティブで指定されたモジュールの中間ファイルを自動生成する
```

▼図3　api.rstページの出力例（api.html）



Mackerelではじめるサーバ管理

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



第18回 Mackerel活用事例 ――ガイアックスの場合

Mackerelの活用事例第2弾。GMOペパボに続き、今回は「ガイアックス」です。サーバ監視において、かつてどのような問題があり、現在どのようなアプローチで解決しようとしているのかを解説します。また、筆者が立ち上げにかかわった「Mackerel User Group」についても紹介します。

ガイアックスの福本です。会社では、Reactioなど複数のWebサービスの開発に従事しつつ、サービスのインフラや、デプロイフローの改善などに取り組んでいます。また最近では、Mackerelユーザの有志が集まって設立された、Mackerel User Groupの立ち上げも担当させていただきました。よろしくお願いします。

さて今回は、筆者が所属するガイアックスにおいてのMackerel導入に至るまでの顛末(てんまつ)や、具体的な活用例などについて紹介したいと思います。

Mackerelを導入するまで

Mackerelを導入するまで、ガイアックスではNagiosやCacti、ZabbixなどのOSSを組み合わせてサーバ監視のしくみを構築していました。これらのしくみは、社内の各サービスを開発するエンジニアグループを横断的に支援する、技術開発部のインフラチームが担当しています。このように、OSSなどを活用してサーバ監視のしくみを社内で構築する場合、いずれ次のどちらかの問題に直面するのではないでしょうか？

(1) サーバ監視の設定変更に時間がかかる
(2) サービスごとに監視のしくみが乱立する

1つめの問題です。インフラチームなどのサービスを横断してサポートするチームが、サーバ監視のしくみを管理している場合、「サーバ監視に関する設定を変更する際にはインフラチームに依頼をする」というフローができあがる場合が多いのではないでしょうか。これは、「全サービスのサーバを一括監視しているので、そのしくみをすべてのエンジニアには自由に操作させたくない」という理由かもしれません。あるいは、「サービス側のエンジニアがサーバ監視のしくみを理解していないので、お願いするしかない」という理由かもしれません。

とくにここ最近では、AWSやAzureなどのクラウドを活用して、サーバを簡単に起動／停止できるようになりましたし、これを用いてサービスへのトラフィックに応じてサーバの台数を自動的に増減できる構成を簡単に構築できるようになりました。

こういった状況で、「サーバ監視に関する設定を依頼しなければならない」というフローができあがっていると、サービス側もインフラチーム側もどちらも手間ですし、何より時間がかかり過ぎてしまいます。

そういった場合によく起こるもう1つの問題が、2つめの「サービスごとに監視のしくみが乱立する」です。サービスごとでサーバ監視のしくみを構築できれば、そのサービスにとって最適な形でしくみを構築できますし、何よりサーバ監視の設定も自分たちで行うことができるので、「インフラチームに依頼する」といったフローによって生じるタイムロスがなくなります。

しかし、この解決策も簡単ではありません。まず、サービスを安定して監視するためのしく

みを構築するのは簡単ではありませんし、その保守管理はチームで責任を持たなければなりません。サービスのためのサーバ監視のしくみを構築したエンジニアが、他チームへの移籍や退職などになってしまって、保守管理できなくなってしまった……というパターンは、絶対に避けなければなりません。

ガイアックスでは、これらの問題を解決するために、Nagios、CactiやZabbixなどを活用したこれまでのインフラ監視のしくみも残しつつ、必要に応じてMackerelを導入することになりました。現在、社内のおよそ3割のサービスでMackerelが活用されていて、とくに新規のサービス開発やサービスのリプレイス時のタイミングで、徐々にMackerelへ置き換えるという流れになっています。

ガイアックスでのMackerel活用

PerlからMackerelを使う

ガイアックスではRubyやPHP、Perlなど、サービスごとに異なるプログラミング言語を利用してサービスを開発しています。サーバやインフラを保守／管理するスクリプトや、デプロイ時に行うさまざまなAPI操作については、公式が提供する各種APIクライアント(たとえば、Mackerelであればmkrコマンド)を利用しても良いのですが、そのサービスで使われている言語を活用できれば、チームでの保守管理はいくぶんか容易になります。

筆者が所属する事業ではおもにPerlを利用することが多いので、MackerelのAPI操作については、CPANで公開されているWebService::Mackerel[注1]を活用することが多いです。

ちなみに、RubyからMackerelのAPIを操作するのであれば、本連載第4回(2015年6月号)でも紹介されているmackerel-client[注2]、Node.jsであればmackerel[注3]、Golangであればmackerel-client-go[注4]を、それぞれ同様に利用できます。

ここでは、簡単にWebService::Mackerelの使い方を紹介しましょう。たとえば、次のようなコードだけで、Mackerelのホスト一覧を取得できます(YOUR_API_KEYはMackerelのAPIキーを、YOUR_SERVICE_NAMEはMackerelに設定されているサービス名を設定します)。

```
use WebService::Mackerel;

my $mkr = WebService::Mackerel->new(
    api_key      => 'YOUR_API_KEY',
    service_name => 'YOUR_SERVICE_NAME',
);

my $hosts = $mkr->get_hosts;
```

なお、WebService::Mackerelの使い方については、WEB+DB PRESS Vol.91に掲載されたPerl Hackers Hub「PerlでInfrastructure as Code」という記事でも紹介させていただいたことがあります。もし興味があれば、ぜひそちらの記事もお読みください。

踏み台サーバでのホスト一覧表示

MackerelとWebService::Mackerelの利用例を、もう1つ紹介しましょう。

ガイアックスでは、オンプレミス／クラウド合わせて500台以上のサーバを管理／運用しており、これらの上でさまざまなサービスを提供していますが、これらのサーバにSSH接続する場合には必ず、メンテナンス用の踏み台サーバ(「SSHゲートウェイ」などと呼ぶ場合もあります)を経由しなければならないようにしています。このとき、接続したいサーバのIPアドレスが常に固定であれば、手元のマシンの.ssh/configを適切に設定するなどして、SSHゲートウェイを経由する多段SSH接続を使うことができます。

しかし、クラウドでは頻繁にサーバ(インスタ

注1) URL https://metacpan.org/pod/WebService::Mackerel
注2) URL http://rubygems.org/gems/mackerel-client
注3) URL https://www.npmjs.com/package/mackerel
注4) URL https://github.com/mackerelio/mackerel-client-go

▼リスト1　あるサービスに紐付くサーバの一覧を表示

```
use WebService::Mackerel;
use JSON qw/ decode_json /;

my $SERVICE = 'YOUR_SERVICE_NAME';

my $mkr = WebService::Mackerel->new(
    api_key      => 'YOUR_API_KEY',
    service_name => $SERVICE,
);

my $json = decode_json( $mkr->get_hosts );
my $hosts = $json->{hosts};

for my $host (@{ $hosts }) {
    next unless $host->{roles}{$SERVICE};

    printf "%s ... %s (%s)¥n",
        $host->{interfaces}[0]{ipAddress},
        $host->{roles}{$SERVICE}[0],
        $host->{status};
}
```

ンス)を起動したり、削除したりすることができますし、とくにAmazon Web Services(AWS)のAuto Scalingなどを利用すれば、サービスの負荷によってサーバの台数を自動的に増減することさえできます。

AWSのElastic Compute Cloud(EC2)の場合、起動したサーバのプライベートIPアドレスは起動するたびに異なるものが割り当てられるので、諸般の事情でどうしてもサーバにSSH接続してオペレーションをしたい場合、いったんAWSのコンソールなどでプライベートIPアドレスを確認してから接続しなければなりません。

一方、これらのサーバをMackerelで管理しているのであれば、プライベートIPを含むEC2インスタンスの各種情報を、MackerelのAPI経由で取得できます。そこで筆者たちはWebService::Mackerelを使って、踏み台サーバ(これもEC2インスタンスの上に構築しています)に接続した際に、そこから接続できるEC2インスタンスのプライベートIPアドレスを、Mackerelから取得して表示するようにしています。こうすることで、踏み台サーバにおいて、現在起動しているEC2インスタンスのプライベートIPアドレスと、Mackerelにおけるステータスを確認し、オペレーションが必要なサーバにSSH接続できます。

たとえば、Mackerel上でYOUR_SERVICE_NAMEというサービスに紐付いているサーバの一覧を表示させるには、先ほど紹介したホスト一覧を取得するコードに手を加えて、**リスト1**のようにすれば実現できます。

YOUR_SERVICE_NAMEに、192.0.2.0～192.0.2.2というIPアドレスを持ったserverというロールのサーバと、192.0.2.8、192.0.2.9というIPアドレスを持ったdatabaseというロールのサーバがある場合、このスクリプトの実行結果は次のようになります。

```
192.0.2.0 ... server (working)
192.0.2.1 ... server (working)
192.0.2.2 ... server (working)
192.0.2.8 ... database (working)
192.0.2.9 ... database (working)
```

あとは、このスクリプトを踏み台サーバにSSH接続した際、自動的に実行するようにすれば良いでしょう。

MackerelのWebhookを使う

またまたPerlの話で恐縮なのですが、CPANにはMackerel::Webhook::Receiver[注5]というモジュールが公開されています。このモジュールを使うことで、MackerelのアラートをWebhookで受け取り、任意の処理を行うというPerlのWebアプリケーションを簡単に作ることができます。

リスト2は、Mackerel::Webhook::Receiverを使ってMackerelのアラートをWebhookで受け取り、その詳細(Webhookで通知されるJSONデータ)を、Data::Dumperでダンプするコードです。起動すると、8080番ポートでWebhookのリクエストを待ち受けます。

なお、MackerelのWebhookについては、Mackerelのヘルプの「Webhookにアラートを通知する[注6]というページに、Webhookの設定方法や

注5）URL https://metacpan.org/pod/Mackerel::Webhook::Receiver
注6）URL https://mackerel.io/ja/docs/entry/howto/alerts/webhook

Webhookに含まれるJSONデータのサンプルが掲載されていますので、こちらの情報も活用すると良いでしょう。

ガイアックスでは、このMackerel::Webhook::Receiverを使って、Mackerel公式で通知に対応していない各種サービスとの連携を試行しています。たとえば、Mackerelが何かしらの問題を検知してアラートが発生し、その対応を行った場合、「何が原因で問題が発生して、どのような対応を行ったか」という履歴を残さなければなりません。そのためのひな形を、Webhookを受け取ったタイミングでQiita:Teamやesa.ioといったサービス上に作成するという実験を試したことがあります。ほかにも、自社で開発して現在SaaSとしても提供しているReactio[注7]を含む社内向けサービスと、API経由で連携させたりもしています。

MackerelのWebhookやMackerel::Webhook::Receiverを用いれば、Mackerelが公式に対応していないサービスであっても、Mackerelのアラートと連携させることができます。便利ですので、ぜひいろいろなサービスとMackerelの連携を試してみてください。

Mackerel User Groupについて

最後に、Mackerel User Groupについて紹介させてください。

Mackerel User Groupは、Mackerelを利用しているユーザによって設立された、Mackerelユーザの集まりです。Mackerel User Groupブログ[注8]からの情報発信はもちろんのこと、Mackerel User Group MeetupなどのイベントやSlackなどを通して、ユーザ間の交流と情報交換を活性化していきたいと思っています。

Mackerelの導入を検討していて、疑問に思っていることや興味があることがあれば、ぜひMackerel User Groupのイベントや Slackにご参加ください！

▼リスト2　Mackerel::Webhook::Receiver

```
use strict;
use warnings;

use Mackerel::Webhook::Receiver;
use Data::Dumper;

my $receiver = Mackerel::Webhook::Receiver->new;
$receiver->on(sub {
    my ($event, $req) = @_;
    my $payload = $event->payload;

    print Dumper $payload;
});
$receiver->run;
```

おわりに

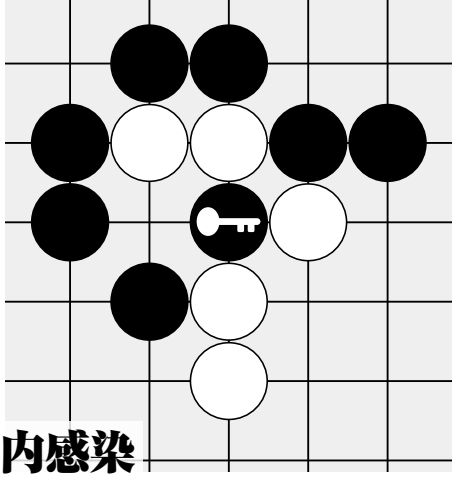
ガイアックスでのMackerelの導入経緯と、社内で活用する際に使っているTipsについて紹介させていただきました。Mackerelは、日本で開発されたサービスであるため日本語の情報が多いこと、そしてその機能や使い勝手もそうですが、サポートの手厚さもすばらしいと思います。困ったことがあれば、問い合わせフォームから連絡を送るとサポートの方がすぐに対応してくださいますし、開発者の方もMackerelが開催するオフィシャルなイベントだけでなく、さまざまな勉強会に参加しておられますので、そういった場所で直接質問や要望を出せるのは、サービスを使う側としてとても安心感があります。

また、最後に紹介させていただきましたが、つい先日Mackerel User Groupも設立されました。Mackerelを日々活用できるユーザさんと交流できるので、日本語の情報が多いことと併せて、Mackerelの導入障壁は今後どんどん下がっていくのではないでしょうか。この機会にぜひ、Mackerelを試してみてください！SD

注7）URL https://reactio.jp
注8）URL http://mackerel-ug.hatenablog.com

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

【第三五回】情報資産とローカルネットワーク内感染

「自分のPCにはたいした情報は入っていないから、不正アクセスされても大丈夫」という人がたまにいます。しかし、そのPCが組織のLANに接続されているなら、考えを改めないといけません。なぜなら、組織内には重要な情報資産があるわけで、そこに内部ネットワークを介してアクセスされ外部に持ち出される可能性もあるからです。

ユーザのとんでもない行動

つい最近ですが、福井県のある町の議会事務局の業務用PCが何者かに遠隔操作されたという事件が発生しました。ここまでは世間でよく聞くマルウェアの感染事例です。しかし、その感染がたいへん興味深いものでした。

まず、職員が職場でアダルトサイトを見ていて画面に「ウイルス感染」という表示が現れたようです。この手のポップアップ誘導はPC以外にも、スマートフォンでも頻繁に見かけます。「無料ウイルス除去ソフトをPCにダウンロード」とか、「スマートフォンの電池の持ちを良くするアプリをダウンロード」など、もっともらしいメッセージが出てきます。

普通はクリックしないでしょうが、ここでクリックしたとしましょう。その場合、外部からアプリケーションをインストールすることになりますから、システムがインストールの警告を表示するなどして、ユーザの許可なしにはインストールはできません。とくにスマートフォンなどでは、開発モードにするといった手間がかかります。インストールの事前準備が必要なので、開発知識がないと難しいでしょう。Androidでは、触っていて偶然に開発モードになってしまうような作りにはなっていません。

しかし、報道を見てみると、予想を越える状況になっていました。

パソコン画面の「ウイルス感染」の表示を見て、そこに記された番号に電話し、電話相手の指示に従って遠隔操作ソフトをダウンロードしたのが原因だった。

（産経WESTの記事[注1]より）

職場でアダルトサイトを閲覧していたという後ろめたさがあったのか、自分でどうにかできると思っていたのか、電話をかければ何でも親切に解決してくれると思ったのか、いくら考えても答えは出ませんが、1つだけわかるのは「ユーザは想像を越える行動を平気でするものだ」ということです。

いずれにしても、正規の方法でアプリケーションをインストールしているわけですから、これではマルウェアは好き放題に何でもできます。もしかすると指定された操作の過程でマルウェア対策ツールを無効にするといった可能性も考えられます。

この指示に従ってインストールしたマルウェアはRAT（Remote Access Tools）というタイプのもので、これにより遠隔からPCが操作されたということです。

低い情報資産評価

先ほどのニュース記事からですが、この遠隔操作

注1）アダルトサイトを職場で閲覧→パソコン外部操作され内部情報流出　町議会事務局長のあきれた危機管理能力
http://www.sankei.com/west/news/160718/wst1607180005-n1.html

されたPCには個人情報もあったようです。

パソコンには議員名簿や議会の資料など開示されているもののほか、事務局長が私的に作成した地元集落の緊急連絡先名簿もあり、名簿には117世帯分の住所、世帯員氏名、生年月日などが記載されていた。

（前述の産経WESTの記事より）

最近は「情報資産」という言葉も聞かれるようになりました。資産／財産として、一般的なのが「不動産」「動産」です。簡単に言えば、「不動産」は土地などで、「動産」とは不動産以外の有体物、つまりモノです。近ごろは、形を持たない著作権や特許権といったものも知的所有権とか無体財産権と呼んで資産／財産として認識されるようになってきました。とはいえ、所有している情報そのものの価値については理解しているとは言い難いものがあります。

たとえば今回のニュースですが、「職場のPCを使って」という言い方から「情報資産が入っているPCを使って」と言い換えたら、とらえ方がずいぶん変わってくると思います。形のない情報資産というものを正しく評価しておらず、低く見積もっているという傾向があるように思います。

「使っているPCにはたいしたものは入っていないから気にしない」

このような言葉は、みなさんも一度や二度は聞いたことがあると思います。そんなことはまずないと思いますが、大幅に譲歩して、その人の情報資産がゼロであったとしましょう。しかし、もしその人のPCが組織内のLANに接続していたならば意味が違ってきます。なぜならば、その人のPCの中にある情報資産の問題ではなく、組織の持つ情報資産の問題になるからです。

その人のPCがマルウェアに感染した、ということだけではなくなります。本来、外部からのネットワークからは安全な環境におかれていた機材までもが、そのマルウェアに感染したPCから組織内LAN経由で、感染していく可能性があるからです。

侵入者の攻撃活動

最初の1台のPCは、RATがしかけられ、外部から侵入した際のゲートウェイになります。ですから、そのPC内にある情報資産がゼロであろうと何であろうと関係ありません。

RATのコントロール下にあるPCは、組織内のLANをターゲットにして探査を開始します。どこにどんなPCやファイルサーバがあるのか調べます。そして、組織内部のLANに接続しているPCへマルウェアの感染を拡大したり、あるいは感染活動はせずに、ほかのPCやファイルサーバ上のファイルを外部に流出させたりするかもしれません。

一般に、内部LANは直接インターネットには接続されていないので、サーバのような外部からの直接のネットワークアクセスはないという想定がされています。そのため内部LANの上でのPCクライアントは必要なセキュリティ・アップデートがされていないケースがあります。パケットフィルタリングなども、組織内の内部ネットワークと公衆ネットワークでは違う設定で運用をしていると思います。つまり、信頼できる内部ネットワークに接続する際には、いろいろなネットワークを介したサービスを使えるようにしていると思います。

内部ネットワークに接続するデスクトップPCは、接続を制限すると、LAN上で動作しているいろいろなサービスが使えなくなります。そのため、内部ネットワークは安全なネットワークとみなして、とにかく接続することを優先にする設定にしておくケースは、よく見かけると思います。

この気持ちはわからないでもありません。組織内でデスクトップPCを使ってドキュメント作成を中心に行っている人たちのためには、もっとも緩い基準で運用しないと、「つながらない」「動かない」と次から次へとクレームが来てたいへんなのは想像がつきます。

ただ、そのような環境は、組織内LAN上のPCが乗っ取られていて外部から操作されている状況では、その侵入者にも有利に働くわけです。たぶん、

このような組織内ネットワーク側からの本格的な攻撃に耐えられるだけのシステムを構築しているところは稀でしょう。ほかのPCへとマルウェアを感染させていくのに、それほど苦労はないと思います。

マルウェア対策ツールは意味をなさない？

先ほどの「指示に従ってマルウェアをPCにインストールする」ケースでは、指示の中にマルウェア対策ツールを無効にする手順も含まれていて、インストールされたマルウェアはスキャンされない／検知されないという可能性が十分に考えられます。

しかし、ほかのPCは人の手によりインストールされるのではなく、侵入するならば外部からの攻撃によりマルウェア(のファイル)が入り込みます。それならば、今度はマルウェア対策ツールが感染したマルウェア(のファイル)を見つけてくれる可能性も十分に生まれてくるわけです。

マルウェアの実行プログラムは外部からの異物ですから、そのマルウェアの動作は標準アプリケーションの動作とは区別できます。少なくともイベントログをみれば、これまでインストールされていなかった謎のアプリケーションが実行されているのを見つけることは可能です。

でも、もしもマルウェアはインストールせず、当然ながら動かしもせず、PCに入っている標準コマンドを外部から動かすだけなら、どうでしょうか？

普通なら、マルウェア対策ツールはファイルをスキャンして疑わしいファイルを見つけて対応してくれます。しかし、マルウェアは入っていませんからスキャンそのものが無意味です。

組織内LANに接続されているほかのPCに対して、外部からの任意のコマンドを動かすことは可能でしょうか？

Windowsのリモート実行コマンド

Windowsであれば、WMIC[注2]のようにリモートホスト上のコマンドを実行するリモート実行コマンドが用意されています。これらはネットワーク側からシステムを管理する際に利用するコマンドです。管理をするオペレータにとっては非常に使い勝手の良いツールです。

遠隔で任意のコマンドを使えるわけですが、勝手に実行できないようにパスワードで制限がかかっています。しかし、パスワードといえども、絶対ではありません。ハッシュダンプツールを用いてパスワードのハッシュ値を入手し、さらにそこからパスワードを探し出すツールにかければパスワードが見つかる可能性は十分にあります。そして、パスワードが見つかれば、好き勝手なことができます。そうなってしまえば、あとは遠隔から任意のコマンドが使えるので、何でもありです。

これらに関しては、JPCERT/CCが提供する情報ページ「攻撃者が悪用するWindowsコマンド(2015-12-02)」[注3](**表1**)を読むとさらにRATを経由して侵入してきたあとで、どんなことをしているか想像がつくと思います。

ファイルをアップロード

たとえば、PCの中にあるファイルをアップロードするにはどうすればいいでしょうか？　マルウェアを使わずに標準のコマンドだけで考えてみます。

まず、Windowsで思いつくのがbitsadminです。bitsadminは、バッチ方式でも簡単にファイルのアップロード／ダウンロードが行えます。ですから、最悪なケースではbitsadminをリモート実行され、PCの中のファイルをアップロードされて情報漏洩することもあり得ます。

最近では、クラウドのサービスを利用するのが日常でも当たり前になっています。当然、窃取したファイルの転送先がクラウドサービスということもあるでしょう。一般論としてクラウドサービスを使う部分でこれまでより良いのは、提供されているAPIを使ってクラウド上のファイルにアクセスでき

注2) WMIC - Take Command-line Control over WMI　https://msdn.microsoft.com/en-us/library/bb742610.aspx
注3) 攻撃者が悪用するWindowsコマンド(2015-12-02)　https://www.jpcert.or.jp/magazine/acreport-wincommand.html

◆表1　攻撃者が悪用するWindowsコマンド（注3のサイトをもとに作成）

▼初期調査でよく使われるコマンド

順位	コマンド
1	tasklist
2	ver
3	ipconfig
4	systeminfo
5	net time
6	netstat
7	whoami
8	net start
9	qprocess
10	query

▼探索活動でよく使われるコマンド

順位	コマンド
1	dir
2	net view
3	ping
4	net use
5	type
6	net user
7	net localgroup
8	net group
9	net config
10	net share

▼感染拡大でよく使われるコマンド

順位	コマンド
1	at
2	reg
3	wmic
4	wusa
5	netsh advfirewall
6	sc
7	rundll32

※wmicは、探索活動などにも用いられます

ることです。ただ、これは標準コマンドだけでは実現できません。それでもクラウドへアップロードするだけのマルウェアを作成すればよいので、それほど難しいことでもありません。またその際の処理ステップは次のようになるでしょう。

- **Step 1：感染PCから情報資産を持っているPCにリモート実行を行う**
- **Step 2：bitsadminを使い外部からマルウェアをダウンロードする**
- **Step 3：マルウェアはクラウドサービスのAPIを持っていてファイルアップロードを行う**
- **Step 4：必要なファイルのアップロードが終わったらマルウェア自身を削除する**

このようなプロセスをたどれば、自分は見つけられることなく、PC上の情報資産を窃取することが可能です。さらに、PCがNASやファイルサーバにアクセスできる状況だと、そのファイルサーバの中身も同様に危険にさらされることになります。

内部からの攻撃には弱い

外部からの攻撃には防壁があり十分な役目を果たしていても、いったん防壁の内側に入られると十分な守りができず崩壊するストーリーはありがちです。「トロイの木馬」というマルウェアの名称も、古代の戦争の際に、大きな木馬の中に兵士が身を潜め、城壁の内側に密かに入り込んで城壁内から敵陣を崩壊させたという話からきています。

一方で、現状で内部も外部も分け隔てなく、すべてに対して同様なセキュリティレベルを想定して防御してしまうと、たいへん使いづらい環境になることでしょう。すでに個人レベルでは、モバイル環境でもデスクトップ環境でも境目がなくクラウドサービスを使っているように、将来的には、組織レベルでも内部ネットワークも外部ネットワークも関係ない、フラットなネットワーク環境のモデルへ変化していくのではないかと予想しています。ですが、いずれにしろ、もう少し先の話であって、今、このときの解決とはならないかもしれません。

守るべき情報資産とは何か

最初の話に戻りますが、自分のPCにはたとえ情報資産が存在していなくとも、そのPCが組織のLANに所属しているなら、組織の情報資産を危険にさらす可能性はあります。

情報資産のことを考えるときは、自分の目の届くところだけではなく、組織全体の情報資産を考えなくてはなりません。「使っているPCにはたいしたものは入っていない」のは、そのとおりなのかもしれませんが、その人の周りにはたくさんの情報資産が存在しているはずです。

そういう観点からユーザの理解を深めていかなければいけないのではないか、と今回の事例を見て深く考えるのでした。SD

SOURCES

レッドハット系ソフトウェア最新解説

第2回

Ansible Tower part2

前回はAnsible Towerの概要とインストール方法を解説しました。今回はAnsible Towerの構造とPlaybookの実行方法を紹介します。

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

Ansible Towerの構造

Ansible TowerでPlaybookを実行するには、Ansible Towerの主な設定項目を理解する必要があります。設定項目の階層図と解説を**図1**と**表1**に記載します。以降のページでは、これらの項目を設定しながらPlaybookの実行手順を解説します。

Ansible Tower CLIのセットアップ

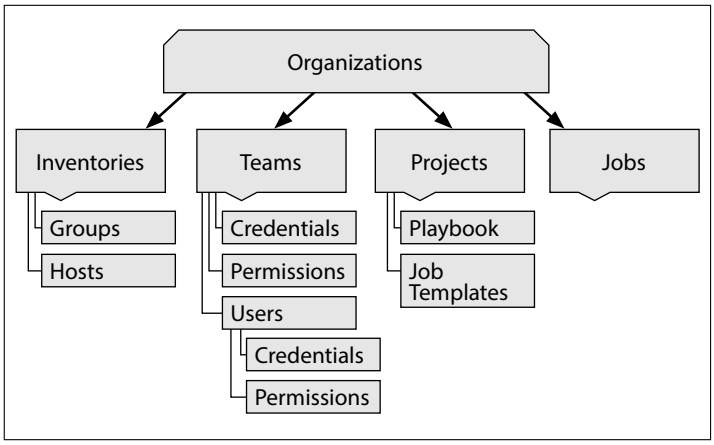
Ansible TowerはWebブラウザから操作できますが、操作手順の自動化や効率的なレビューなどの実現につながるCLI[注1]や、ほかのアプリケーションとの連携のためのREST APIを利用することもできます。本記事では、主にCLIによる操作手順を紹介していきます。まず、Ansible Towerをインストールしたサーバ上で、次のコマンドを実施してCLIをインストールします。

```
# yum -y install python-pip
# pip install ansible-tower-cli
```

次にCLIのセットアップをしていきます。セットアップやAnsible Towerの操作には`tower-cli`コマンドを利用します(**図2**)。`tower-cli`ではhelpオプションも用意されていますので、使い方を逐次調査しながら実施していくことができます。

Ansible TowerのCLIのセットアップが完了すると、カレントユーザのホームディレクトリにAnsible Towerを操作するためのホスト／ユーザ名／パスワード情報が記載された、「`.tower_cli.cfg`」

▼図1　Ansible Towerの階層図



注1) URL https://github.com/ansible/tower-cli

▼表1　Ansible Towerの設定項目

項目	解説
Organization	下記4項目の上位項目
Inventory	Playbookの実行対象となるホスト。Ansible Towerでは、Inventory単位でPlaybookの実行対象を指定する
Team	ロールベースのアクセス制御を実現する単位。Teamに所属するユーザは、Organizationの管理者により割り当てられたPlaybook実行時の認証情報(Credential)と、Inventory/Job Templateを利用できる。Inventory/Job Templateのアクセス権限は、Permissionで指定する
Project	Ansible Towerで実行するPlaybookを管理する。Playbookの格納先は、Manual(Ansible Towerのローカルディレクトリ)/Git/Subversion/Mercurialを指定できる。Ansible TowerのJobとしてPlaybookを実行するためには、Project/Credential/Inventoryを紐付けたJob Templateを作成する必要がある
Job	Job Templateを利用したPlaybookの実行や実行履歴を管理する。また、Jobのスケジューリングも可能

▼図2　Ansible TowerのCLIのセットアップ

```
# tower-cli config username admin
# tower-cli config password adminユーザのパスワード
# tower-cli config host http://localhost
# cat /root/.tower_cli.cfg
[general]
host = http://localhost
username = admin
password = adminユーザのパスワード
# tower-cli user list
== ======== ================ ========== ========= ============
id username email            first_name last_name is_superuser
== ======== ================ ========== ========= ============
 1 admin    メールアドレス                                 true
== ======== ================ ========== ========= ============
```

が作成されます。セットアップの最後に、Ansible Towerのインストール時に自動的に作成されるadminユーザが正常に表示されるかを見ることで、tower-cliの動作確認をしています。

Playbookの実行

いよいよPlaybookの実行を解説していきます。ここでは、とある部署が管理しているホストに対して、ユーザを作成するための簡単なPlaybookを実行することを想定しています。その場合の、一連のコマンド実行例を紹介していきます。

最初に、InventoryとCredentialを作成します。Inventory01という名前のInventoryを作成し、IPアドレス(192.168.124.19)を持つサーバをGroup01に所属するホストとして登録します(**図3**)。ここで登録するホストは、IPアドレスまたは名前解決できるホスト名である必要があります。ssh-copy-idでSSHの公開鍵を対象ホストにコピーしたあとに、公開鍵をCredential01として登録します。

次に、Project01という名前のProjectを作成します(**図4**)。ここでは、Ansible Towerのローカルディレクトリを Playbookの格納先として指定するため、作成したPlaybook(user-create.yaml)を格納しているディレクトリ、sample-project01を指定します。なお、Ansible TowerではProjectに登録するディレクトリは、/var/lib/awx/projectsに作成する必要があります。

最後に、Job Templateを作成してJobを実行します(**図5**)。Group01/Project01/Credential 01を紐付けた、job-user-create01という名前のJob Templateを作成します。前の手順でPlaybookの格納先となるディレクトリのみをProjectに登録していますので、user-create.yamlをここで登録します。作成したJob Templateを指定して、最後にJobを実行するとPlaybookが実行されます。Jobを実行すると、Ansible Towerでは**図6**のような画面で実行結

▼図3　Playbookのコマンド実行例（Inventory、SSH公開鍵の作成）

```
# tower-cli organization create --name Organization01
# tower-cli inventory create --name Inventory01 --organization Organization01
# tower-cli host create --name 192.168.124.19 --inventory Inventory01
# ssh-copy-id root@192.168.124.19
# tower-cli credential create --name Credential01 --username root --password redhat --kind
ssh --ssh-key-data /root/.ssh/id_rsa
```

▼図4　Playbookのコマンド実行例（Playbookのためのディレクトリ作成など）

```
# mkdir -p /var/lib/awx/projects/sample-project01
# cat << EOF  > /var/lib/awx/projects/sample-project01/user-create.yaml
---
- hosts: all
  tasks:
        - name: create user
          user: name=james
EOF
# tower-cli project create --name Project01 --organization Organization01 --scm-type manual
--local-path sample-project01
```

▼図5　Playbookのコマンド実行例（Job Templateの作成と実行）

```
# tower-cli job_template create --name job-user-create01 --job-type run --inventory
Inventory01 --project Project01 --playbook user-create.yaml --machine-credential Credential01
# tower-cli job launch --job-template job-user-create01
```

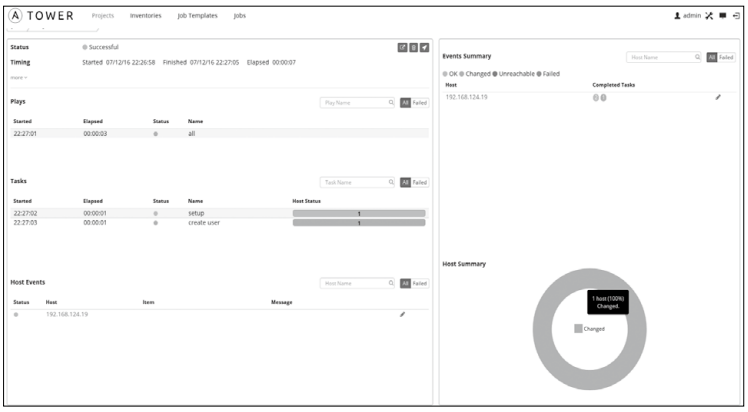
果を確認できます。Ansible TowerのGUIでは、どのユーザがいつ何のJobを実行したかといった履歴以外にも、標準出力されたPlaybookの実行結果や作成されたユーザ情報なども確認できます。

ここまではAnsible Towerのadminユーザで操作してきましたが、各部署の運用担当者にOrganizationの管理ユーザを割り当てて、InventoryやProjectの作成を委任できます（**図7**）。

こちらの操作で、前に作成したCompany01の管理者としてuser01を登録することで、user01権限を持つ運用担当者がAnsible Towerを利用して、Company01に紐付けられたInventoryやProjectなどを管理できるようになります。また、Organization間ではリソースが隠蔽されますので、各部署に割り当てられたサーバだけをAnsible Towerで管理できるようにもなります。

ほかにもPermissionの設定を行うことで、非エ

▼図6　Jobの実行結果



▼図7　Organizationの管理ユーザの割り当て

```
# tower-cli user create --username user01 --password redhat --email user01@example.com
# tower-cli organization associate_admin --user user01 --organization Organization01
```

▼図8　inventoryのインポート

```
# tower-manage inventory_import  --inventory-name=Inventory01 --source= Inventoryファイルまたはディレクトリ
```

▼図9　Amazon EC2上で管理しているサーバの登録方法

```
# tower-cli credential create --name ec2-Credential01 --kind aws --username AWSのアクセスキー ↵
--password AWSのシークレットキー
# tower-cli group create --name ec2-group01 --source ec2 --credential ec2-credential01 ↵
--inventory Inventory名 --update-on-launch true --overwrite true
# tower-cli group sync ec2-group01
```

ンジニアの方に対してJob Templateを実行する権限だけを与えて、ボタンを押すだけの安全なPlaybookの実行環境を用意できます。こうした手順の詳細については、Ansible Towerのユーザーガイド[注2]を参照ください。

Inventoryのインポート/Dynamic Inventory

前のInventory作成手順では逐次ホスト名やIPアドレスを指定していましたが、ホストやグループの情報を記載した既存のInventoryファイルをインポートしたり、AnsibleのDynamic Inventory[注3]を利用することができます。インポートの場合はtower-cliではなく、Ansible Tower付属のtower-manageコマンドを利用します(**図8**)。

また、AnsibleのDynamic Inventoryを利用して、Amazon EC2上で管理しているサーバを登録する場合は**図9**のような手順を実行します。

ec2-Credential01としてAWSのアクセスキーとシークレットキーを登録し、その認証情報が紐付けられたec2-group01という名前のグループを作成します。そしてsyncを実行すると、Dynamic Inventoryが実行されてEC2上にあるホストがec2-group01に登録されます。グループ作成時に--update-on-launch, --overwrite,を指定することで、Job実行時、またはsync実行時に、すでに存在しないホスト情報を自動的に削除してくれます。

Ansible Tower 3.0.0のリリース

米国時間2016年7月19日に、Ansible Tower 3.0.0がリリースされました。v3.0.0での主な変更点は次のようになります。

- Web UIの刷新(v2.4.5と基本概念は同じ)
- Dynamic Inventoryの拡張(Red Hat CloudForms/Red Hat Satellite 6にも対応)

詳細はこちらで公開されているWebページ[注4]での情報を参照ください。

Ansible/Ansible Towerの導入支援

レッドハットでは本記事のようなお客様への製品紹介のほかにも、プロフェッショナルサービスでAnsible、またはAnsible Towerによるシステム構成管理の導入支援なども行っています。

Ansibleの導入支援に興味のある方は、ぜひ気軽にsales-jp@redhat.com宛にお問い合わせください。SD

注2) URL http://docs.ansible.com/ansible-tower/latest/html/userguide/index.html
注3) URL http://docs.ansible.com/ansible/intro_dynamic_inventory.html
注4) URL http://docs.ansible.com/ansible-tower/latest/html/release-notes/index.html

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第34回 ❖タイムスケジュールでプログラムを実行

◉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。

必要最小限のサービス

FreeBSDはプロジェクトが配布しているインストーラ経由でインストールした場合、デフォルトではほとんどのデーモン（サービス）が動作しない設定になっています。必要な機能があればその都度有効化して利用するといったように、シンプルでセキュアなほうにデフォルトの設定が振られています。

システムのデフォルトの挙動は/etc/defaults/ディレクトリにまとまっています。

```
% tree /etc/defaults/ ↵
/etc/defaults/
├── bluetooth.device.conf
├── devfs.rules
├── periodic.conf
└── rc.conf

0 directories, 4 files
```

/etc/defaults/rc.confがとくにデーモン（サービス）の有効化・無効化や、起動時のオプションなどを設定するファイルです。簡単にデーモン（サービス）の有効化・無効化に関連する設定行を抜き出すと**図1**のようになります。だいたい「**デーモン名**_enable="YES"」となっているものが有効化で、YESがNOになっていれば無効化されています。YESになっていると、必要に応じてシステム起動時に自動的にそのデーモン（サービス）が実行されるしくみになっています。

▼**図1 デフォルトで有効になるデーモンなどの設定**

```
% uname -sr ↵
FreeBSD 10.3-RELEASE-p4
% grep -E ^[^#][a-z]+_enable+='"YES"' ↵
/etc/defaults/rc.conf | awk '{print $1}'
devd_enable="YES"
cleanvar_enable="YES"
gptboot_enable="YES"
hostid_enable="YES"
syslogd_enable="YES"
cron_enable="YES"
crashinfo_enable="YES"
dmesg_enable="YES"
virecover_enable="YES"
newsyslog_enable="YES"
mixer_enable="YES"
```

実際にFreeBSDを必要最小限設定でインストールした場合にデーモンとして起動してくれるのは、デバイスの状況変化に応じてアクションを取るためのdevd(8)、システムやデーモン（サービス）のログを収集するためのsyslogd(8)、指定されたスケジュールで指定されたプログラムを実行するcron(8)、メールのやりとりを管理するsendmail(8)くらいで、それ以外のデーモン（サービス）は動作していません。逆に言えば、これらのデーモン（サービス）は汎用オペレーティングシステムとして動作するための基本的なデーモン（サービス）ということになります。

タイムスケジュールで処理を行うcron(8)

今回は指定したタイムスケジュールで指定されたプログラムを実行するcron(8)を取り上げます。システムを運用するためのいくつかの機能がcron(8)

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。

経由で実行されているため、なんらかのサービスを開発した場合に、実はcron(8)で定期的に実行しているだけとか、cron(8)を使って仕事量が少なくなる夜間にバッチ処理を走らせているだけとか、そういった運用が実は多いからです。そんなわけで有益で大切な機能なのですが、今ではこの存在を知らない方もいるようです。

cron(8)もしくはこれに類する機能はいくつか異なる方法で利用でき、設定ファイルもいくつかあります。今回はシステムが利用する設定ファイル/etc/crontabを読むとともに、どのように設定するのかを説明します。

システムレベルのタイムスケジュール設定ファイル/etc/crontab

FreeBSD 10.3-RELEASEの/etc/crontabは**図2**のようになっています。#から始まる行はコメント行で、空行も設定としては意味をもたず、それ以外の行が意味のある内容になっています。cron(8)はこのファイルに書いてある内容をそのまま読み、指定されたタイムスケジュールどおりに、指定されたコマンドを実行していきます。

必要最小限の設定だけ抜粋すると**リスト1**のようになります。左から5列目までは「いつ」を指定する項目です。6個目が「誰」の権限で実行するかで、7個目以降が実際に実行するコマンドやプログラムとなっています。わかりやすいようにまとめると次のようになります。

何分 何時 何日 何月 何曜日 誰 実行する内容 ...

それぞれの項目では、基本的に「何分:0～59」「何時：0～23」「何日：1～31」「何月：1～12」「何曜日：0～7」のように数字が指定できます。

曜日の指定は0または7が日曜日を意味しています。1が月曜日、2が火曜日、3が水曜日、4が木曜日、5が金曜日、6が土曜日です。項目が*になっている場合にはすべてに対して一致します。たとえば「何時」の指定が*になっているなら、ここは0から23まですべて指定したのと同じことになります。

/etc/crontabでは*/5といったようにスラッシュも使われていますが、スラッシュは指定した数字分だけスキップせよ、という指定になります。たとえ

▼図2 FreeBSD 10.3-RELEASEの/etc/crontab

```
% cat /etc/crontab ↵
# /etc/crontab - root's crontab for FreeBSD
#
# $FreeBSD: releng/10.3/etc/crontab 194170 2009-06-14 06:37:19Z brian $
#
SHELL=/bin/sh
PATH=/etc:/bin:/sbin:/usr/bin:/usr/sbin
#
#minute hour    mday    month   wday    who     command
#
*/5     *       *       *       *       root    /usr/libexec/atrun
#
# Save some entropy so that /dev/random can re-seed on boot.
*/11    *       *       *       *       operator /usr/libexec/save-entropy
#
# Rotate log files every hour, if necessary.
0       *       *       *       *       root    newsyslog
#
# Perform daily/weekly/monthly maintenance.
1       3       *       *       *       root    periodic daily
15      4       *       *       6       root    periodic weekly
30      5       1       *       *       root    periodic monthly
#
# Adjust the time zone if the CMOS clock keeps local time, as opposed to
# UTC time.  See adjkerntz(8) for details.
1,31    0-5     *       *       *       root    adjkerntz -a
```

ば「何分」の部分が/5になっているなら、「5分飛ばせ」という意味になります。「何分」の列では、*は0から59までを意味するので、つまり*/5で「5分ごとに実行せよ」という意味になります。

1,31といったカンマ区切りの指定がありますが、これは1と31の両方とも、という指定です。0-5という指定も1ヵ所あります。これは0,1,2,3,4,5と指定したのと同じです。範囲指定ということになります。つまり*の指定は、-を使って最初から最後まで範囲指定したことと同じになりますし、カンマ区切りで全候補を記述したこととも同じになります。

crontab(8)を読んでみよう

crontab(8)ではほかの書き方も用意されていますが、システムがデフォルトで用意しているcrontab(5)の内容を読むだけなら、前述したルールを知っているだけで大丈夫です。**リスト1**を見ながら、一通り読んでみましょう。

最初は①です。この行は「*/5 * * * *」がスケジュールの指定で、実行権限はroot、実行対象は/usr/libexec/atrunとなります。何時、何日、何月、何曜日がすべて*になっているので、つまり何分に記述された*/5がスケジュールということになります。よって、「5分ごとにroot権限で/usr/libexec/atrunを実行せよ」という指定になります。

at(1)についてもそのうち取り上げようと思いますが、at(1)で実行が指定されたものの、指定された時刻になっても実行されなかったジョブは優先度の低い待ち行列に追加されることになります。この待ち行列からジョブを持ってきて定期的に消化していくのが、/usr/libexec/atrunです。システムの負荷が高いと、待ち行列入りしたままなかなか実行されないこともあるのですが、cron(8)で、実行できるときにユーザのジョブを実行しています。

次の行も指定内容はatrunと同じです(②)。11分ごとに実行せよ、というスケジュールになります。/usr/libexec/save-entropyというのは/dev/randomからランダム値を抽出して、/var/db/entropy/以下にsaved-entropy.[1-8]といった名前で保存するためのプログラムです。このプログラムが11分おきに起動され、ランダム値が保存されることになります。このランダム値は、システム起動時に/dev/randomの乱数のシードとして使われることになります。システム起動時において特定の条件だと(たとえば起動時十分な乱数性を確保できない)擬似乱数が十分な乱数性を確保できないという懸念があり、このように乱数を保存することで、次回のシステム起動時における乱数性の確保につなげているというわけです。

次に行ってみましょう(③)。「いつ」の指定は「0 * * * *」となっています。この指定だと何分の0だけが指定されていることになりますので、1時間に1回、たとえば13時になった段階で1回、14時になった段階で1回といったように、1時間に1回だけnewsyslog(8)というコマンドがroot権限で実行されることになります。

newsyslog(8)は、おもに/var/log/ディレクトリに保存されるログファイルをローテーションすることが目的のコマンドです。ファイルがある一定サイズを超えたら新しいファイルにするとか、古いファイルは圧縮しておくとか、そういったことをさせています。

次の行(④)は「1 3 * * *」となっていますので、毎晩午前3時1分に「periodic daily」というコマンドをroot権限で実行せよ、という指定になります。

▼**リスト1　必要最小限だけ抜粋した/etc/crontab**

```
#minute hour    mday    month   wday    who     command
*/5     *       *       *       *       root    /usr/libexec/atrun ……①
*/11    *       *       *       *       operator /usr/libexec/save-entropy……②
0       *       *       *       *       root    newsyslog ……③
1       3       *       *       *       root    periodic daily……④
15      4       *       *       6       root    periodic weekly……⑤
30      5       1       *       *       root    periodic monthly ……⑥
1,31    0-5     *       *       *       root    adjkerntz -a……⑦
```

FreeBSDサーバを枕元において寝たことがある方はご存じかもしれませんが、FreeBSDは午前3時1分になるとハードディスクがゴリゴリ音をたてます。秘密裏に株価予測プログラムが動き出すとかビットコインマイニングプログラムが動き出すとかそういうことではなく、おもにファイルシステム上のヘルスチェックや、毎日調べたほうが良いステータスのチェックなどが行われています。午後3時1分に動き出すのが嫌であれば時刻を変更するか、コメントアウトして処理そのものを動作しないようにするという手もあります。

次の行(⑤)もperiodic(8)の実行を指定したものです。「15 4 * * 6」となっていますので、「毎週土曜日の午前4時15分」に「periodic weekly」を「root権限で実行せよ」という指定になります。

periodic(8)はデイリー、ウィークリー、マンスリーで実行する定期チェックを担当するプログラムで、実際にはこのようにcron(8)経由で実行されています。そして次の行(⑥)もperiodic(8)の実行に関するものです。こちらは「30 5 1 * *」ですので、「毎月1日の午前5時30分」に「root権限」で「periodic monthlyを実行せよ」ということになります。

システムがデフォルトで設定しているスケジュールは次の設定で最後です。⑦では「1,31 0-5 * * *」となっています。この指定で「毎日午前0時1分、午前0時31分、午前1時1分、午前1時31分……午前5時1分、午前5時31分」に「root権限」で「adjkerntz -a」を実行せよ、という指定になります。

カーネルは通常、協定世界時の時刻情報を保持しています。PCやサーバのハードウェア側は現地時間(日本だと日本標準時)の時間を保持しています。adjkerntz(8)はこの2つの間の値を適切に調整するコマンドです。夜間の間に微妙な時刻の調整を行っていることになります。

と、このように、普段は気にすることなく使っているかと思いますが、実際にはシステムにそのように実行するようスケジュールが書き込まれ、cron(8)が指定されたスケジュールに従ってプログラムを実行しているだけというのが、動作の本当のところだったりします。

◆ ◆ ◆

以上、cron(8)の説明でした。cron(5)とその設定ファイルcrontab(5)はシステム運用でもっとも基本的な機能の1つなのですが、最近は知らない方も多いようですので取り上げました。気にしたことがなかったのであれば、これを機に一度使ってみてはいかがでしょうか。SD

column

periodic(8)は必要か?

デフォルトの設定は最小限で汎用的に使えるものが想定されていますので、periodic(8)も動作しますし、必要最低限のステータスチェックも行います。しかし、periodic(8)が実行するデイリーのチェック処理はディスクに対して軽い処理というわけでもありませんので、この時間帯に重いバッチ処理などを実行していると、システムはだいぶ苦しい状況に追い込まれます。

また、AzureでもAmazon EC2でもKVMでもbhyveでもVMwareでも良いのですが、仮想環境でFreeBSDをホストしている場合、periodic(8)の処理の一部やadjkerntz(8)などの処理は実行する意味がなくなることがあります。ntpd(8)の実行などもそうなのですが、そういった処理は仮想環境のホスト側で処理するので、ゲスト側で処理すると"メガネonメガネ"というか、無駄な処理になるからです。

このため、エンタープライズで利用するようなケースであったり、本番環境で運用するようなケースでは、こうした処理が実行されないようにcrontab(5)を編集することがままあります。periodic(8)が実行されているタイミングで何かシステムの反応が遅くなるとか、負荷が高くなるといったことがあれば、それら機能を無効化することを検討してみてください。スケジュールの設定をコメントアウトしてからcron(8)を再起動すれば設定が反映されます。

LibreOffice 5.2の新機能

Ubuntu Japanese Team／あわしろいくや

今回は、本誌発売後にリリース予定のLibreOffice 5.2の新機能や変更点についてです。

LibreOffice 5.2概要

LibreOffice 5.2は、タイムベースリリースの方針に従って半年に一度のリリースが行われているLibreOfficeの新バージョンです。8月第1週のリリースを目指して開発が進んでいます。

5.1から引き続きユーザインターフェースの変更点が多いのですが、Calcには比較的大きく手が入っています。また、特筆すべき変更点としては、ついにUbuntuからでもGoogle Driveにアクセスできるようになりました。

全般

リモートファイルで Google Driveのサポート

これまでも何度か取り上げてきましたが、LibreOfficeとしてはリモートファイル機能でGoogle Driveをサポートしていたのですが、Ubuntuを含むLinux用のバイナリではGoogleの認証をパスできませんでした。それが、ようやくファイルの読み込みと保存ができるようになりました。

認証は通常のパスワード認証のほか、2段階認証にも対応したため、Googleのアカウントを持っている場合は誰でもGoogle Driveにアクセスできるようになるはずです。

なお、Google Driveへのアクセスにはクライアント IDとクライアントシークレットが必要で、LibreOfficeではconfigureで指定する必要があります。言うまでもなくこれらは個人ないし団体ごとに取得する必要があり、オフィシャルバイナリとUbuntu用バイナリでは別のものを使用することになります[注1]。Ubuntu用のバイナリで使用するクライアントIDとクライアントシークレットは5.1.0の段階で取得済み[注2]ですので、5.2でも有効な状態でリリースされることが期待できます。

なお、Microsoft OneDriveサポートは5.2でも見送られています。

テンプレート

テンプレートのウィンドウが一新されました（**図1**）。カテゴリの切り替えがタブからプルダウンになり、テンプレートが多数ある場合でも簡単に目的のものが絞り込めるようになりました。

また、テンプレートを開くのはもちろん、編集やデフォルトのテンプレートを変更することも簡単にできるようになりました。LibreOfficeのデフォルトのテンプレートは日本語で使うぶんには適した状態になっていない部分もあるので、簡単に変更できるのは朗報でしょう。

注1） ちなみにLibreOfficeはDebianとUbuntuでほぼ同じソースからビルドされていますが、このあたりには違いがあります。

注2） https://bugs.launchpad.net/bugs/1389936

さらに、標準ツールバーの保存アイコンのオプションに、[テンプレートとして保存]が追加されました(**図2**)。この機能を使用すると、テンプレートのカテゴリを選択して保存できます。そればかりか、ここから直接デフォルトのテンプレートにもできます。

テンプレートのインポートとエクスポートと移動が簡単にできるようになったのも特記すべきことでしょう。意外なことに、今まではない機能でした。エクスポートは選択したテンプレートをテンプレート形式で保存します。移動はカテゴリ間を移動します。インポートは、カテゴリを選択後テンプレートファイルを選択します。現在表示しているカテゴリにインポートするわけではないのは注意点でしょう。

なお、この変更はGoogle Summer of Code(GSoC)2016の成果によるものです。GSoCの成果は通常毎年2月リリース版に盛り込まれるのですが、このプロジェクトは開始後わずか4週間で所定の作業がおおむね終わってしまいました。よって5.2のフィーチャーフリーズに間に合ったという極めて珍しい事態になったのです。残作業としては、次のバージョン(5.3を予定)でLibreOffice Templates[注3]のデータを取得できるように変更する開発があり、これはまだ取り込まれていません。

余談ですが、5週目以降は絵文字を簡単に入力できるようにするツールバーの開発を行っているようですので、次のバージョンでは簡単に絵文字の入力ができるようになっているかもしれません[注4]。

ツールバー

ツールバーが3つ追加されました。いずれも[表示]-[ツールバー]で表示のオンオフができます。

まず最初はWriterとCalcに追加された[標準(シングルモード)]で(**図3**)、これは基本的な機能を提供するツールバーと、書式設定ツールバーを統合して1つのツールバーにしたものです。これをオンにする場合は[標準]と[書式設定]ツールバーを非表示にします。そうすると、ツールバー1行分縦に広く使えるようになるということです。言うまでもなく昨今のワイドディスプレイでは横のスペースには余裕がありますので、フルHD程度の解像度で使用している場合はこのツールバーを使用するといいのではないでしょうか。もちろん[標準]と[書式設定]ツールバーのすべてのアイコンがあるわけではないので、すべてのケースで切り替えできるわけではありませんが、少なくとも書式ツールバーはサイドバーからでも

図1　新しいテンプレートのウィンドウ

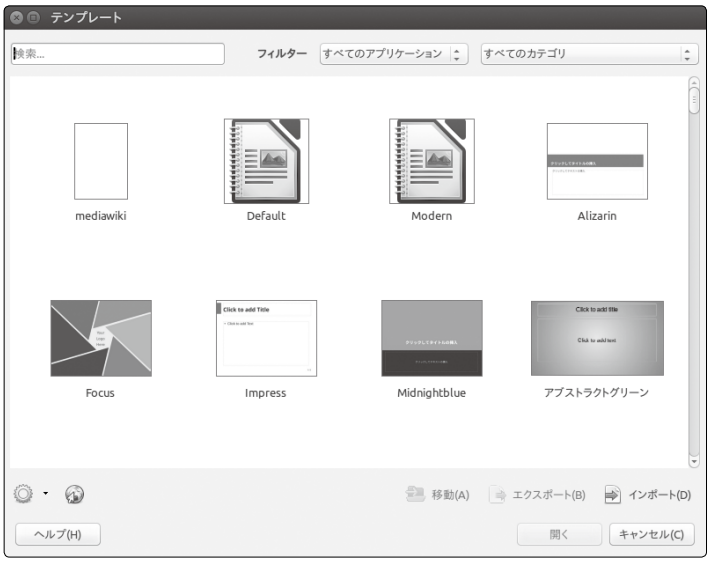


図2　テンプレートとして保存が追加された

図3　Writerの[標準(シングルモード)]ツールバー

注3) http://templates.libreoffice.org/

注4) 現状でもオートコレクトを使用すれば簡単に絵文字の入力ができますが、記法を覚えなくてはいけないので若干ハードルは高めです。

変更ができるため、問題とはならないでしょう。

ほかには機密ツールバーも追加されています。これはわかりにくいですが、デジタル署名をするためのツールバーです。Writerだけではありますが、差し込み印刷ツールバーも追加されています。

検索と置換

検索と置換ウィンドウが一新されました(**図4**)。新たに[前を検索]と[次へ検索]ができるようになりました。

デジタル署名の強化

オランダ国防省のスポンサードで[注5]、デジタル署名機能が強化されています。まずは1つのドキュメントに単数の著者による複数の署名ができるようになりました。あとSHA-256のハッシュ関数が使用できるようになり、それに伴ってOOXMLの署名のインポートとエクスポートができるようになりました。

曲線

Draw/Impressにはもともとあったのですが、Writer/Calcの図形描画ツールバーからもベジエ曲線や多角形の曲線を描けるようになりました。

ファイルへ出力

LibreOfficeでは、これまでも印刷ウィンドウからファイルへ出力できましたが、[オプション]タブの[ファイルへ出力]にチェックを入れる必要があり、直感的ではありませんでした。5.2からはプリンタの1つとして表示されるようになったため、とても使いやすくなりました。とはいえどの程度使いどころがあるのかは疑問なところがあります。ちなみに出力形式はOSによって異なり[注6]、UbuntuではPostscript形式となります。

Writer

変更の追跡バー

変更の追跡機能は、簡単にいえばバージョン管理機能ですが、メニューをたどるほか[表示]-[ツールバー]-[変更の追跡]でツールバーを表示して使用します。今回[標準]ツールバーにこのツールバーの表示と非表示を切り替えるアイコンが追加されました。

相互参照にフィルター

[挿入]-[フィールド]-[他のフィールド]-[相互参

注5) イタリア国防省はすでにLibreOfficeに移行していますが、この機能の追加によりオランダ国防省でもLibreOfficeが採用されるのかもしれません。資金を提供して機能を追加してもらい、それを採用することによって全体のコストを下げることができ、かつLibreOfficeユーザのメリットにもなるというのは、理想的な状況といえます。日本でもこのようになるといいのですが、なかなか難しいです。

注6) ちなみにWindowsではPRN形式です。

図4　新しい検索と置換ウィンドウ

図5　新しいブックマークウィンドウ

照]タブに、フィルタ機能が追加されました。多数の相互参照を使用している場合でも、簡単に絞り込めるようになりました。

ブックマークの挿入の強化

これまでの[ブックマークの挿入]は本当に挿入することしかできず、操作も直感的ではありませんでした。5.2ではそれが見直され、ただ挿入するだけではなくその場所に移動(ジャンプ)したり、削除やブックマーク名の変更もできるようになりました(**図5**)。なお、ブックマーク名は自動で割り当てられるようになったので、数が多くない場合はそれに任せるのもいいでしょう。

Calc

セルの固定が簡単に

これまでセルの固定をしたい場合、固定する行と列の1つ下をアクティブにし、[表示]-[行と列の固定]をクリックする必要がありました。5.2では[表示]-[セルの固定]のサブメニューに[行と列の固定]のほか、[最初の行を固定]と[最初の列を固定]が追加されました。たいていの場合は最初の行か列を固定するでしょうから、アクティブなセルを移動する必要がなくなったので便利といえます。

隣接するセルからも枠線を削除

これはちょっとわかりにくいのですが、たとえば6×6のセルに罫線を引いたとします。真ん中の4×4から罫線を消す場合、通常であればほかのセルと隣接する罫線は残ります(**図6**)。しかし、新たに追加された[隣接するセルからも枠線を削除]にチェックを入れると、隣接するセル、すなわちこれまで罫線が残っていたセルからも罫線が消えます(**図7**)。どういった場面で役に立つ機能なのかはよくわかりませんが、今までの方法で同じことをしようと思うとすごく面倒だったので、大幅に手数を減らすことができるようになりました。

図6　従来は隣接するセルの線は残る

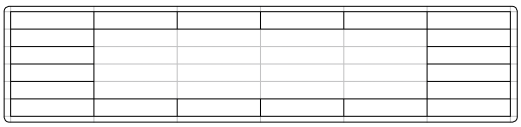
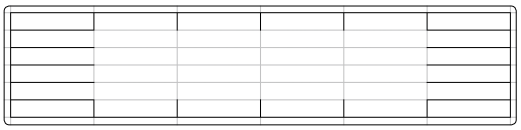


図7　[隣接するセルからも枠線を削除]にチェックを入れるとこうなる



ステータスバーの機能を同時表示可能に

Calcの画面右下に、合計や平均などを簡易表示するステータスバー機能があります。これまではどれか1つしか有効にできませんでしたが、5.2からは複数同時に表示できるようになりました(**図8**)。

関数のツールチップ

セルに直接関数を入力する場合、ツールチップが表示されるようになりました(**図9**)。これで関数を完全に覚えてなくても入力できるようになったので、ウィザードを使う機会が減ることが期待できます。なおこの機能は、個人名は伏せますが日本人の大学生によって実装されました。

新しい関数

新しい関数が追加されています。**表1**にまとめましたのでご覧ください。おおむねMicrosoft Excel 2016との相互運用性を向上させるために追加されていますが、RAWSUBTRACTは計算の精度を上げる

図8　複数の項目にチェックを入れることができるようになった

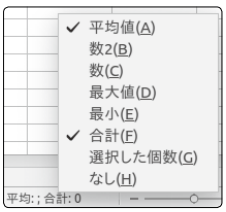


図9　SUMで始まる関数をツールチップで表示している

ために追加されています。

仕様変更のあった関数

WEEKDAY関数は、Microsoft Office 2010以降と同じく種類(タイプ)に11～17が追加されました。日曜日を1とする17は、日本では便利そうです。

ワイルドカードのサポート

Calcは一部の関数でワイルドカードが使えるだけで、Microsoft Excelほど広範囲に使用できなかったのですが、このたび大幅に見直されました。なお、実際に使えるワイルドカードは、任意の一文字の"**?**"と、空白を含むすべての文字の"*****"で、エスケープには"**~**"を使用します。

ただし、デフォルトでは正規表現を使用するようになっており、変更するには[ツール]-[オプション]-[LibreOffice Calc]-[計算式]の[一般的な計算]を変更します。

キーアサインの変更

数式入力ボックスで Shift +Enter を押した場合、これまでは1つ上のセルに移動していましたが、5.2からは改行になりました。

これまで Shift +F4 はセルの参照に割り当てられていましたが、5.2からはこの機能は無効になりました、[ツール]-[オプション]-[LibreOffice Calc]-[互換性]-[キーバインディング]を[OpenOffice.org互換]に変更すると、従来どおりの挙動になります。

現在アクティブなセルの行を全選択する場合は Shift +スペースキー、列を全選択する場合は Ctrl +スペースキー、全シートを選択する場合は Ctrl + Shift +スペースキーとなりました。ただし通常 Ctrl +スペースキーはインプットメソッド(Fcitx)にアサインされているため、変更しない限り使用できません。

表1　新しい関数

関数	用途
RAWSUBTRACT	誤差を丸めず減算
FORECAST.ETS	実績から予測値を求める7つの関数
CONCAT	Excel 2016の同名の関数と互換
TEXTJOIN	Excel 2016の同名の関数と互換
IFS	Excel 2016の同名の関数と互換
SWITCH	Excel 2016の同名の関数と互換
MINIFS	Excel 2016の同名の関数と互換

Draw/Impress

サイドバーのアニメーションの設定

Impressのサイドバーにある[アニメーションの設定]は、これまで追加時に[+]アイコンをクリックしてダイアログから適用する項目を選択していましたが、5.2からはこのダイアログがなくなり、サイドバーから直接設定するようになりました。

スライドのプロパティをサイドバーのプロパティに追加

スライド全体のプロパティをサイドバーから変更できるようになりました。[書式]-[ページ]を開く必要がなくなったので、非常に便利になりました。

図形描画ツールバーのオンオフ

標準ツールバーに、図形描画ツールバーの表示と非表示を切り替えるアイコンが追加されました。一時的に縦のスペースを広くしたい場合に便利でしょう。

Ubuntuで5.2を使用する

いつものように、10月にリリースされるUbuntu 16.10はLibreOffice 5.2をデフォルトでインストールするでしょう。また、やはりいつものようにPPA[注7]でも配布され、16.04などでも使用できるようになるでしょう。

同時にsnapパッケージでも配布されるようになるはずですので、PPAからインストールするのに抵抗がある場合は使用してみるといいのではないでしょうか。執筆段階では、少なくとも5.2 Beta2は配布されています[注8]。SD

注7) https://launchpad.net/~libreoffice

注8) https://skyfromme.wordpress.com/2016/06/14/libreoffice-5-2-0-beta2-as-a-snap-package/

39

Debian Developer　やまねひでき　henrich@debian.org

DebConf16レポート（前編）

Debian Hot Topics

DebConf16開催

Debian開発者会議「DebConf」が南アフリカ・ケープタウンのケープタウン大学で開催されました[注1]（**写真1、2**）。参加者が集まりやすいヨーロッパや北米ではなく初のアフリカ大陸での開催ということで、例年より参加者は少なめではありましたが、それでも280名超が100以上のセッションやイベントを精力的にこなしました。今回と次回では、そこからいくつかトピックをかいつまんで紹介したいと思います。

セッションビデオについてはDebConf16のサイトにリンク[注2]がありますので、そちらをご覧ください（有志がYouTubeにもアップロードしているようなので、そちらを検索いただいても良いでしょう）。

▼写真1　宿の学生寮からの眺め。晴れた日にはテーブルマウンテンが見える

▼写真2　ケープタウン大学構内の様子

注1）URL https://debconf16.debconf.org/
注2）URL https://debconf16.debconf.org/talks/
注3）URL https://annex.debconf.org//debconf-share/debconf16/slides/13-we-need-you-to-release-debian.pdf

Debian 11のコードネームは「Bullseye」

リリースチームのEmilio Pozuelo Monfortさんのセッション「We need you to release Debian」[注3]では、Debian 9“Stretch”のリリースに向けての状況と、フリーズを短くするための取り組みへの協力依頼が語られました。

Debian 9のリリーススケジュールについては以前からとくに変更なく、2017年2月5日に開発フリーズが宣言されています。「現状、修正が必須となるリリースクリティカルバグ（RCバグ）は1,062個あり、そのうちの300個超がGCC 6がらみ、別の300個超がdebhelperの旧バージョン依存で、さらに別の330個程度がキーとなるパッケージに存在している。まずはこれらのバグを修正していく必要がある」と述べられました。リリース対象アーキテクチャについては**表1**のようになっています。

そして早くもDebian 11のコードネームが発表されました。これまでどおりToy Storyのキャラクターから WoodyとJessieの愛馬

「Bullseye」が選ばれています(**表2**)。

systemd in Debian -- a status update

DebianではJessieから正式に採用されたsystemdですが、そのsystemdのDebianでの現状についてのセッションです[注4]。

「systemd自体の開発はかなり速いスピードで行われている。そんな状況でも、Upstream側で入念に作りこまれているユニットテストに加え、Debianパッケージ側でも基本的な振る舞いのテストをするautopkgtest群を加えることで、品質を担保しつつすばやくUpstreamの変更に追随して、いつでもunstableに投入できている」とのことです。なお、Upstreamでのsystemdの開発はGitHubを利用しており、すべてのプルリクエストに対してCI(Continuous Integration)が走るようになっているようです。システムの中核を成すソフトウェアなのに「枯れている」とは言いがたい変更が続々と加えられるsystemdですが、大きなリグレッション(デグレード)を避けつつリリースできているのはこのおかげですね。

パッケージとしては、systemdパッケージ群の分割(systemd-containerなど)を行って、インストールされるライブラリパッケージを最小になるようにしているとのこと。その際には、ほかのディストリビューションとも協力しているそうで、昨年行われたsystemd conferenceでは、systemd関連のパッケージ名をなるべくディストリビューション間で共通にするように話し合いなどを行ったようです。

また、先日のリリースでは、libsystemd-shared.soというライブラリに共通コンポーネントをまとめるようにした結果、パッケージサイズが50%削減されたとのこと。筆者が実際にビルドして確認したところ、バージョン230-1では35MBだったのが、バージョン230-7では16MBと大幅に削減されていました[注5]。

現在作業中なのが、systemd採用に従って不要になったレガシーな機能やパッケージの依存関係を削除することです。セッションでは、次の項目が取り上げられていました。

- initscriptsパッケージの依存関係整理(完了)
- sysv-rcパッケージのプライオリティ降格
- insserv対応パッチの削除
- rcS対応パッチの削除(完了)

initscriptsはシステム起動/シャットダウン用スクリプト集のパッケージです。このパッケージに含まれていた/lib/init/vars.shファイル(多数のinitスクリプトが参照しています)をsysvinit-utilsパッケージに移動することで、パッケージの削除が可能になりました。同様のパッケージであるsysv-rcは、プライオリティが「important」ですので自動的にインストールされてしまいます。しかし、いくつか見つかっている問題を処理すれば、プライオリティを落としてインストールされないようにできそうだ、とのこと。両方合わせても削減されるサイズは微小ですが、こういう積み重ねが大事なのでしょう。

systemd自身でも、SUSE由来のinsserv用

注4) URL https://annex.debconf.org/debconf-share/debconf16/slides/89-systemd-in-debian--a-status-update.pdf

注5) systemdソースパッケージからビルドされる全バイナリパッケージの総計での比較。

▼表1　Debian 9のリリース対象のアーキテクチャ

現状、対象になっているもの(と追加候補)	対象にするか検討中のもの
amd64/i386 armel/armhf/arm64 mips/mipsel/(mips64el) ppc64el s390x	kfreebsd-i386/amd64 powerpc sparc64

▼表2　Debianのコードネーム

バージョン	コードネーム	リリース時期
Debian 8	Jessie	2015年5月
Debian 9	Stretch	2017年春予定(おそらく5月)
Debian 10	Buster	未定(2019年春?)
Debian 11	Bullseye	未定(2021年春?)

のDebian固有パッチ(/etc/insserv.conf(.d/)を利用する機能)[注6]を削除するなどの取り組みを進めています。

もう1つはrcS周りの整理です。/etc/rcS.d/配下のinitスクリプトは、システム起動時のハードウェア周り、たとえばディスクのマウントやネットワークなどの初期化作業などを実施します。これをsystemdのネイティブサービスに置き換えることで、これらのinitスクリプト間での依存関係でループが起きてトラブルが発生するのを避けられるようになる、と説明がありました。「rcSのinitスクリプトは、kFreeBSDやHurdの場合やinitにsystemdではなくOpenRCなどを使う場合に参照するので、きちんと直したほうが良いのでは?」という質問については、「systemdの観点からは、rcSではなくsystemdネイティブなserviceファイルをパッケージに入れることで対応してもらう」との回答でした。

なお、Debianには積極的に新しいバージョンのsystemdが投入されていますが、systemdが提供する全サービスを有効にしているかというとそうではなく、次のような例外があります。

- ネットワークインターフェース処理には、現在、Debianで利用しているifupdownのリプレースが必要になるため、systemd-networkd[注7]についてはまだ有効にしていない
- Ubuntuではネットワーク名前解決のコンポーネントとしてsystemd-resolved[注8]を推しているが、Debianでは有効にしていない。採用はUbuntuでの反響を見てから決めたい、とのコメント
- 同様にブートローダーであるsystemd-boot[注9]も有効にはしていない

注6) Debianでsystemdを採用する以前にinitに使っていました。

注7) URL https://www.freedesktop.org/software/systemd/man/systemd-networkd.service.html

注8) URL https://www.freedesktop.org/software/systemd/man/systemd-resolved.service.html

注9) UEFI専用のブートローダー。現状のGRUBを置き換えることになるか? URL https://www.freedesktop.org/wiki/Software/systemd/systemd-boot/

このあたりは、同じようにsystemdを採用しているディストリビューションであっても、違いが出るところですね。

CDNサービス「deb.debian.org」

Tollef Fog Heenさんが発表したのは、Debianパッケージの CDNサービス[注10]「deb.debian.org」です[注11]。これまでも、Debianパッケージリポジトリ用のCDNサービスは複数存在していました。第1世代のCDNが、現在、AWSでソリューションアーキテクトとして活躍していらっしゃる荒木靖宏さんが実装した「cdn.debian.net」、次に第2世代として「httpredir.debian.org」、そして第3世代が今回の「deb.debian.org」という位置づけになるでしょうか。

日本国内にいる限り、パッケージリポジトリは現状のftp.jp.debian.orgを指定するのが最適です。しかし、頻繁に海外へ移動を行う場合や、利用者の地理的な特定ができないdockerなどのマシンイメージでミラーを指定する場合には、CDNであるdeb.debian.orgを指定しておくのが最適でしょう。deb.debian.orgを利用するには、/etc/apt/sources.listに**リスト1**の設定を追加します。

これまでのCDNと比較すると、deb.debian.

注10) Contents Delivery Networkの略。1サーバに負荷が集中しがちなコンテンツデータ配信について、各地にコンテンツをコピーしたエッジサーバを配置し、ユーザから最も近いエッジサーバからコンテンツを配信することで、負荷分散と快適なサービスの提供を可能にする。今回名前が出てくるFastly以外ではAkamai、CloudFlare、MaxCDNなどが有名。

注11) URL https://debconf16.debconf.org/talks/97/

▼リスト1　deb.debian.orgの設定のしかた

```
DNSのSRVレコードをサポートしていないjessieのaptの場合
deb http://cdn-fastly.deb.debian.org/debian ↵
jessie main  contrib non-free
deb http://cdn-fastly.deb.debian.org/debian-↵
security jessie/updates main

stretch以降のaptの場合
deb http://deb.debian.org/debian sid main ↵
contrib non-free
```

orgは通常のパッケージ／セキュリティアップデートパッケージ／デバッグシンボルパッケージ／移植版のdebian-portsと、広い範囲でパッケージが取得可能になったのと、クライアントにとってより最適なエッジのキャッシュが利用できるようになっているのが利点です。

現状の実装では、バックエンドのCDNとして過去にTollefさんが勤務していたFastly[注12]を利用しているようです[注13]。今のところまだ実験段階のサービスで、ほかのCDNサービスとも協力していく予定であり、今後正式なリリースが待たれます。

ScreenをDebianインストーラに統合

日本からの参加者の発表では、ロジャー清水さんによるターミナルマルチプレクサ[注14]のGNU ScreenをDebianインストーラに統合したという発表「GNU Screen comes to Debian Installer」がありました[注15]。とくに画面出力やキーボード接続が存在しないARM組み込み系などのヘッドレスマシン[注16]でのシリアルコンソール経由のインストールや、IBMのPowerシリーズで使われているppc64elアーキテクチャや汎用機であるs390xアーキテクチャなどへのSSH経由でのインストール作業の際、簡単に仮想コンソールを切り替えられるようになります(**図1**)。これらの機器では、通常のPCとは違ってインストール最中にシェルに切り替えて機器固有の設定をコマンドで実施する必要があるため、Screenで仮想コンソールが使えるのは大きなメリットです。

注12) URL https://fastly.com

注13) Fastlyというフリーではないサービスに依存しているなどの問題は存在しますので、ここはどう折り合いをつけていくのかが、今後の課題になりそうです。

注14) 1つのターミナルで仮想ターミナルが利用できるようにする機能。ScreenやTmuxがある。

注15) URL https://annex.debconf.org/debconf-share/debconf16/slides/86-gnu-screen-comes-to-debian-installer.pdf
URL https://debconf16.debconf.org/talks/86/

注16) たとえばNASなど。

Debianインストーラのバージョン Stretch alpha7「以降」[注17]での利用が可能になっているので、ぜひお試しください。

“jessie and a half”リリースの検討

さて、DebConf中には発表以外にもメーリングリストでの議論が活発に行われています。Steve McIntyreさんは、jessie-backportsに含まれている新しいカーネルやX関連パッケージを「jessie and a half」としてリリースするのはどうか、という提案を行いました。過去にDebian 4.0“etch”に対して「etch and a half」[注18]として、サポート対象のハードウェアを増やすためにLinuxカーネルやXドライバの更新を行ったことがあります。Debian 8“Jessie”でもこれと同様に行おう、という提案です。

Jessieリリース後、ARM64アーキテクチャやamd64アーキテクチャのクライアント(つまりは普通のPC)のSkylakeプラットフォームで

注17) 執筆時点ではdaily imageのみになります。URL http://cdimage.debian.org/cdimage/daily-builds/daily/arch-latest/amd64/iso-cd/などを参照。

注18) URL https://www.debian.org/releases/etch/etchnhalf.ja.html

▼図1　インストール時の仮想コンソール切り替え

上部に仮想コンソールが4つ切り替えできることが表示されている
仮想コンソールは CTRL ＋ A で切り替え可能

は、大きな変更が続々と起きています。今回はそれらの新しめの機器での対応を主目的としているようです。

以前の「etch and a half」との違いは、Debian 9 "Stretch"のリリースまで、継続的にbackportsリポジトリからカーネルを更新する予定であるところです。とくに主だった反対は行われず、作業ターゲットの日程としては7月下旬あるいは8月頭という話でしたので、この号が発売されるころには実施されているかもしれません。名称についてはもっと良いものがあれば、それでかまわない("Jessie Backport August 2016"などはどうか？)とあるので、最終的には若干変わっているかもしれませんね。SD

COLUMN カンファレンス参加への道(下準備から現地到着まで)

DebConfの申し込みは開催3、4ヵ月ほど前にオープンし、Webから申し込みができます。ちなみに参加にあたっては特別な資格は「不要」で、どなたでも無料で参加できます。さらに、早めに登録を行うと宿泊や食事のスポンサー枠を受けることもできます。渡航費については、必要であれば費用の申請を行い、それをDebConfチームが審査して可否を決めます。この審査に通るためには、これまでにDebianへの何らかのContributionを目に見える形で行っているか、をアピールできるかどうかが肝です。

参加までに準備が必要だと筆者が思ったことを**表A**にまとめておきました。次回のDebConfに限らず、今後、初めて海外IT系カンファレンスに行くという方は参考にしてください。

現地空港から会場までの移動ですが、DebConfでは参加者がWikiで自身の乗る飛行機／列車の時間を記載して移動のコーディネートをし、複数人で待ち合わせてタクシーシェアして会場入りするスタイル[注19]を採っていますので、意外に安心です。筆者は空港に着いてすぐに、先に会場入りしていた知り合いのDebian開発者に見つけられ、彼らの運転するレンタカーでケープタウン大学入りしました。大きな都市であればUberを使うのも良いでしょう。

注19) URL https://wiki.debconf.org/wiki/DebConf16/TravelCoordination/Arrival

▼表A　カンファレンス参加チェックリスト

内容	チェック	コメント
パスポート	□	今すぐ取ろう！　GPGキーサイン交換にも使える
クレジットカード	□	キャッシュ機能を有効に。付帯保険の確認を忘れずにすること
カンファレンス申し込み	□	早めの申し込みだと割引や宿／食事のスポンサーが受けられたりする
休暇の取得	□	誰かが休んでも単一障害点にならない職場づくりの機会だと思おう
航空券購入	□	トランジットに気をつけて
ESTA(eTA)登録／ビザ申請	□	北米行きの場合は要注意
ノートPCと電源アダプタ	□	現地で壊れたり盗られたりしても泣かない。ディスクは暗号化しておこう
電源変換プラグ	□	いくつかの国で使えるものを買っておこう
SIMロックフリー端末	□	安いので構わない。モバイルルータでも良い
モバイルバッテリー	□	複数持ちがお勧め
スーツケース	□	帰りのお土産を入れるスペースがあると良い
着替え	□	長期滞在の場合は洗濯するかランドリーサービスを利用しよう
虫除け／日焼け止め／薬	□	日焼け止めは現地で購入したほうが強力なのが手に入って良いかも
他の参加者へのお土産など	□	せっかくだから何か手土産があると喜ばれる。宣伝したいことがあれば資料やステッカーなども持参
印刷したカンファレンス情報	□	開催地の住所／連絡先電話番号は必ず控えておく
印刷したEチケット	□	旅費の補助を受ける際には必須※。スマートフォンでも見れるとなお良い
オンラインチェックイン	□	搭乗24時間前から可能
現地通貨	□	空港で両替。少額ならレートはあまり気にしない

※カンファレンスへの参加で旅費の補助を受ける場合、現地で開催チームに旅費の精算を求められることがあり、そのような場合に必要になってきます。PDFファイルがあれば印刷すれば……と思っても、プリンタがうまく動かない場合や、そもそもプリンタがない場合もありますので、データだけでなく印刷物も準備しておきましょう。

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

前回に引き続きファイル操作を紹介します。今回はファイルやディレクトリの移動、リンク、リダイレクトについて解説します。

第5回 ファイル操作の基本(その2)

エンジニアに必要な基礎能力は、問題解決力や読解力、表現力、対話力、論理思考能力など多岐にわたっています。いずれも一朝一夕に身につくものではありませんので、日々の努力と経験が重要です。そうした能力の中で、初心者のうちに必ず身につけておいたほうが良いものを1つ挙げるなら、「タイピングスピード」です。とにかく素早くたくさんのプログラムや命令をコンピュータに入力できる能力が邪魔になることはありません。入力が遅ければ、キーボードをたたいているうちにせっかく考えていたアイディアやアルゴリズムが頭の中から消滅してしまうこともあります。ライバルエンジニアが1,000行書き上げる間に、入力が4倍速ければ4,000行のコードが書けます。キーボードの音高らかにタイプしているので後ろからよく見ていると、入力した先からバックスペースで修正ばかりしていて、一向に前に進まない人がいますが、間違いが多いのも思考が中断してしまいます。タッチタイピング(手元を見ない)で、素早く、間違いなく入力できるようになりましょう。

そんな能力はすでに高いレベルに達しているというあなたには、2番目に効果がある「整理整頓」をおすすめします。;-)

続・ファイル操作

移動とリンク

前回のファイル操作基本1で紹介した、ディレクトリ・ファイルの作成、複製、削除はもうばっちりですね。さらに先月から十分に経験を積んだ皆さんは、CLI環境(テキスト情報だけ)でもファイルシステム中で迷うことはないでしょう。弊社の新人さんも、要領を得て調子がでてきたのか、口調も軽やかになってきました。

さて、前回やり残したファイルの移動とリンクを体験しましょう。実践環境が残っている人は、そのまま続けて動作を確認できます。

mv—MoVe—ファイルやディレクトリを移動する

mvはファイルやディレクトリを移動したり名前を変えるコマンドです。移動対象がディレクトリの場合、その下にあるファイルやディレクトリも移動します。

図1のように../d1をそっくりd1の下に移動してみましょう。

```
$ pwd
/Users/masa/Work/d2   masaはユーザ名が入る
$ mv ../d1 d1
```

簡単ですね。

STEP UP! mv応用例:複数のファイルの拡張子をまとめて変更する

Windows系のファイルシステムでは、ファイル名は<ファイル名>.<拡張子>の形式が一般的です。Unixとは異なり、拡張子('.'以降の文字列)にファイルの種別を示す意味があります。ここでは、特定のディレクトリ内にあるファイルの拡張子をまとめて変更する方法を探ってみましょう。

拡張子.dmgを.isoに変更してみます。

現在のファイルの状態
```
$ ls↵
a.xxx a.dmg b.yyy c.iso d.dmg
```

まず単純にmvを使って、次のように指定できるでしょうか……

失敗例
```
$ mv *.dmg *.iso↵
```

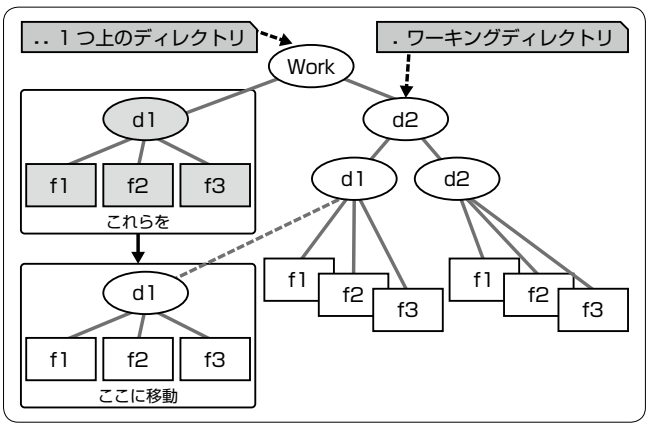
この失敗は前回のcpの例でもあったように、移動先が複数であることが原因です。mvの代わりにechoを使って出力を見れば一目瞭然です。a.dmgとd.dmgをc.isoに移動しろ、という指示になってしまいますね。

```
$ echo *.dmg *.iso↵
a.dmg d.dmg c.iso
```

解決するには、前回のように複数のmvコマンドを発行する必要があります。今回は、シェルの繰り返し制御を使ってみましょう。bashではforによる繰り返し処理ができますので、次のように記述します。

bashでの技
```
$ for f in *.dmg ; do mv ${f} ${f%.*}.iso; ↵
done↵
```

▼図1 移動する



ここではforとシェル変数f、拡張子を取り除くためのパターンマッチ演算%.*を使っています。ちょっと見慣れない文字の並びで気後れしてしまいそうですが、頑張って理解しましょう。bashのこのような文字列展開の機能については、また別の回に改めて詳しく解説しようと思います。

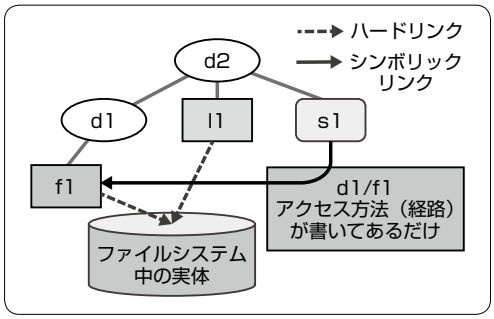
FreeBSDなどは、cshもしくはその拡張版であるtcshが標準のシェル[注1]です。shの系統と、cshの系統の両方を理解し操作できるようになれば、ほとんどの環境で困ることはないでしょう。ここでは参考として、前の処理と同じ動きをcshで記述してみます。

```
% foreach i ( *.abc )↵
foreach? echo $i $i:r.iso↵
foreach? end↵
```
← 行頭のforeach?は、foreach処理内での入力プロンプト

ちなみにsh系のプロンプトは$ですが、cshの標準プロンプトは%です。foreachとendは、独立した行に書くと決まっているので、bashのように1行では書けません。:r演算子で、拡張子を取り除いた文字列が得られます。見た目はちょっと違いますが、bashの技と同じ操作になっていることが確認できます。

注1) サン・マイクロシステムズの共同設立者である、ビル・ジョイがcsh、viの開発者です。氏は、70年代の終わりから80年代前半に、UCBでBSD Unix開発において中心的な役割を果たしていました。そうした経緯もあり、FreeBSDなどBSD系Unixでは標準シェルがcshということがあります。

▼図2　ハードリンクとシンボリックリンク



作業現場でよく現れるこの手の短いスクリプトは、イディオムとして覚えておきましょう。

シンボリックリンクとハードリンク

Unixファイルシステムは、異なる名前で同じファイルやディレクトリにアクセスする方法、リンクを2種類提供しています(**図2**)。

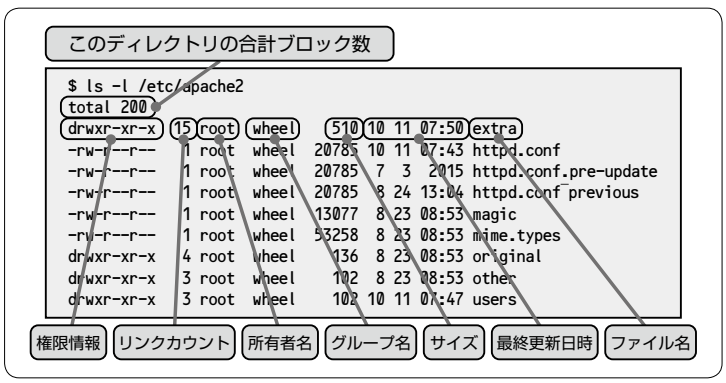
ls -lの情報再び

ls -lの情報を再び[注2]確認しましょう(**図3**)。

リンクカウントに注目してください。リンクカウントは、ファイルがリンク(参照)されている数で、i-node内にあります。ファイルが作られた状態ならリンクが1つでリンクカウント1、ハードリンクされるたびにリンクカウントも増えます。rmコマンドなどでファイルを消す操作は、実際には参照情報だけ削除してリンクカウントを1つ減らします。リンクカウントが0になるとファイルの実体が占有していた領域を解放します(**図4**)。

注2) Software Design 2016年6月号、172ページの図7で確認しました。

▼図3　ls -lの実行例と意味(OS X 10.11.4の例)



ln—LiNk—ハードリンクを作る

lnコマンドを使って、リンクとリンクカウントを実際に試してみましょう。

```
$ ls -l d1/f1↵
-rw-r--r--  1 masa staff  0  2 13 14:36 d1/f1

$ ln d1/f1 ./l1↵
$ ls -l d1/f1 ./l1↵
-rw-r--r--  2 masa  staff  0  2 13 14:36 ./l1
-rw-r--r--  2 masa  staff  0  2 13 14:36 d1/f1
```

リンク前に1だったd1/f1のリンクカウントが、もとのd1/f1も、新しく作ったl1も、ともに2になっています。

i-node番号を確認してみましょう。lsに“-i”オプションを指定してみます。

```
$ ls -i d1/f1 ./l1↵
34202749 ./l1   34202749 d1/f1
```

同じi-node(この場合34202749)が共有されていることがわかりますね。i-node番号は1つのファイルシステム内でユニークな番号です。ハードリンクでは、ファイルシステムをまたいだリンクは作れないことに注意してください。

ln -s—LiNk -s—シンボリックリンクを作る

シンボリックリンクはファイルやディレクトリへのアクセスの方法(パス)を示したものです。リソースの実体はないので、アクセス先がどこかに移動されたり削除されると、デッドリンクになります。

では、早速シンボリックリンクを作ってみましょう。シンボリックリンクは、lnコマンドに-sオプションを指定して作ります。

```
$ ln -s d1/f1 ./s1↵
$ ls -F↵
d1/ l1  s1@
$ ls -Fl s1↵
lrwxr-xr-x  1 masa  staff  5  2 13 14:43
s1@ -> d1/f1
```

上の例では、d1/f1を参照するs1というシンボリックリンクを作成しました。lsに"-F"オプションを指定したので、s1にシンボリックリンクである'@'がついていることが確認できます。

試しに、もとのd1/f1を削除してみましょう。

```
$ rm d1/f1↵
$ ls -lF s1 d1/f1 l1↵
ls: d1/f1: No such file or directory
lrwxr-xr-x  1 masa  staff  5  2 13 14:43
s1@ -> d1/f1
```

もとのファイルは消え、先に作ったハードリンクl1のリンクカウントは1減っています。s1はまだ、d1/f1へのアクセスを示したまま存在していますが、次のように"cat"コマンドを使ってファイルにアクセスしてみると、

```
$ cat s1↵
cat: s1: No such file or directory
```

エラーメッセージが示すように、参照先に実体がないことがわかります。

ハードリンクも、シンボリックリンクも実体を複製しませんので、大きなファイルが対象であってもディスク容量の消費は気になりません。バックアップのとき、既存のファイルに別の名前やアクセス方法を提供したいときなど便利に使えます。

STEP UP! システムによっては異なるファイルシステムへの移動はできない

mvコマンドは内部でrename(2)システムコールを呼び出して、ファイルの実体はそのままに、参照情報を書き換えます。ファイル実体のサイズは大きいこともありますが参照情報はほんのわずかです。参照情報だけを書き換えることで、ファイルをすべてコピーする無用な負荷がかかりません。

異なるファイルシステム間を移動するには、ファイルの実体も移動させなければなりません。そのためシステムによっては、mvで異なるファイルシステム間の移動ができない場合があります。

例：異なるファイルシステムでは動かないときに出力されるエラーメッセージ

```
"inter-device move failed, unable to
remove target: Is a director"
```

逆にmvで異なるファイルシステム間での移動ができるシステムは、ファイルシステムが異なる場合には(1)コピー先にすでにファイルがあるなら削除しておく(2)コピーを実施(属性やパーミッション、構造を維持。シンボリックリンクは辿らない)(3)コピー元を削除、という一連の動作をしています。

ファイルシステムをまたいで、mvと同じ処理をする

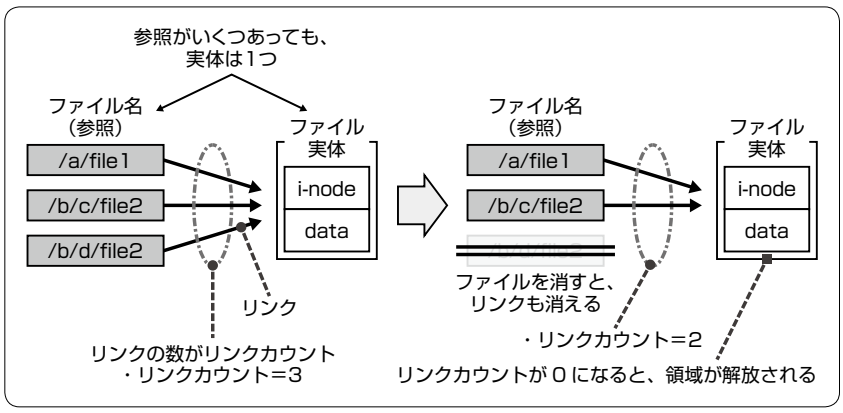
```
$ rm -f コピー先 && \↵
$ cp -pRP コピー元 コピー先 && \↵
$ rm -rf コピー元↵
```

木構造を再帰で克服

Unixのファイルシステムは階層構造です。rm -r、mkdir -pなど、階層構造をたどって処理できることを紹介しました。すでに紹介したコマンドにも、階層構造に対応できるものがあります。

rmdirはディレクトリの中が空でなければディレクトリを削除できませんでし

▼図4 リンクカウントとファイルの実体



た。rmdir -pは、サブディレクトリ内のディレクトリにファイルがなければ、深層にあるディレクトリまでまとめて削除できるオプションです。もちろんrm -rなら、ファイルも含めて削除できるので、あまり使う機会はないかもしれません。

先月のディレクトリマップ作成では、ディレクトリの各階層に対してlsを実行して確認しました。ls -Rとオプションを指定すれば、対象のディレクトリ以下を再帰的にたどって見ることができます。各ディレクトリごとに空白行で区切られて表示されています。それぞれのセクションのはじめの行の○○:の○○が、ディレクトリパスです。

ls -Rの例(Ubuntu)
```
$ ls -R
.:
d1  d2

./d1:
f1  f2  f3

./d2:
d1  d2  f1

./d2/d1:
f1  f2

./d2/d2:
f1  f2
```

オプションがコマンドによって、-R、-r、-pなどと異なっています。Rとrはrecursive(再帰)、pはparents(親)の意味です。mkdir、rmdirなど、parentsでの指定はユーザ自身、最下層がどの位置かわかっていて、そこから親ディレクトリをたどるというイメージです。一方rmなどのrecursive指定は、指定したポジションから下の階層を根こそぎ取ってくるというイメージですから、下の階層にどのくらいファイルやディレクトリがあるかわからない状態で指定するのは危険です。今後紹介するコマンドにも、これらと同様なオプション指定で階層的処理に対応したものがあります。

ファイルのリダイレクト

Unixは、プロセスを起動するとデフォルトで3つのファイルが開かれています。3つのファイルは、それぞれ0:標準入力(stdin)、1:標準出力(stdout)、2:標準エラー出力(stderr)です。番号は、ファイルディスクリプタと言い、プロセス内でファイルをオープンするごとに、空いている若い番号から順に割り振られるユニークなIDです。

各コマンドが単純に出力する場合は標準出力に出力され、入力を求めると標準入力から入力されます。標準出力は、デフォルトでは画面への出力、標準入力はキーボードからの入力で、標準エラー出力も画面への出力に設定されています。

シェルはこれらの入出力先をファイルにしたりコマンドと連結する[注3]ことができます。

リダイレクト

出力先や入力元をファイルに変更する機能=リダイレクトは、後に紹介するテキスト処理で多用するファイルとコマンドのやり取りのしくみですので、ここでマスターしておきましょう。

何も指定しなければ「標準入出力」は画面とキーボードです、これをファイルなどに変更するには、次の記号を使います。

・>:ファイルへ出力(上書き)
・<:ファイルから入力
・>>:ファイルへ追記

例で確認しましょう。

```
$ ls /bin -FC > ls_out.txt
$ cat ls_out.txt
[*       df*          launchctl*  pwd*    tcsh*
bash*    domainname*  link*       rcp*    test*
(..中略..)
date*    kill*        pax*        stty*
dd*      ksh*         ps*         sync*
```

注3) これをパイプでつなぐと言いますが、パイプの話は改めて。

ls_out.txtにlsの結果が書き込まれました。続けて、既存のファイルに対して同様にリダイレクト処理をしてみます。

```
$ ls /usr/bin > ls_out.txt↵
$ cat ls_out.txt↵
2to3*                mdls*
2to3-*               mdutil*
(..中略..)
mdimport*            znew*
mdimport32*          zprint*
```

ls_out.txtは上書きされています。もう一度/binの内容を、'>>'のリダイレクトによって書き込んでみます。

```
$ ls /bin >> ls_out.txt↵
$ cat ls_out.txt↵
2to3*                mdls*
2to3-*               mdutil*
(..中略..)
date*   kill*   pax*   stty*
dd*     ksh*    ps*    sync*
```

ファイルに、新しい処理の結果が追加されました。次は、入力のリダイレクトも組み合わせてみましょう。**tail**コマンドは、ファイルの終わりから10行分を表示します。コマンド引数で、入力するファイルを指定できますが、指定しなければ標準入力からです。ここではリダイレクトを意図的に使っています。

```
$ tail < ls_out.txt > ls_out2.txt↵
$ cat ls_out2.txt↵
mdimport*            znew*
mdimport32*          zprint*
[*      df*          launchctl*  pwd*    tcsh*
(..中略..)
dd*     ksh*         ps*         sync*
```

ls_out.txtをtailに入力して、結果をls_out2.txtに出力できました。

STEP UP!

リダイレクト処理は、シェル内でコマンドを子プロセスとして起動する際に、forkされた子プロセス内で標準入出力を一度closeしてから、指定された入出力ファイルをopenすることで、ファイルディスクリプタ（0、1、2、……）の対象をファイルに置き換えます。その状態で、コマンドをexecすれば実行されたコマンドのファイルディスクリプタ（0、1、2、……）は、すでにオープンされたファイルとなっているわけです。出力先としてopenするときに、上書き／追記の指定がされています。

既存のファイルを空（0bytes）にする

リダイレクトを理解したところで、既存のファイルを空にする方法を紹介しましょう。さまざまな方法がありますが、よく使われているのは/dev/nullという（読んでも、書いても常に）空っぽの特殊ファイルを、cpしたりcatしたりする方法です。

```
cpを使った例
$ cp /dev/null file↵

catとリダイレクトを使った例
$ cat /dev/null > file↵
```

どちらも、同じように動作します。

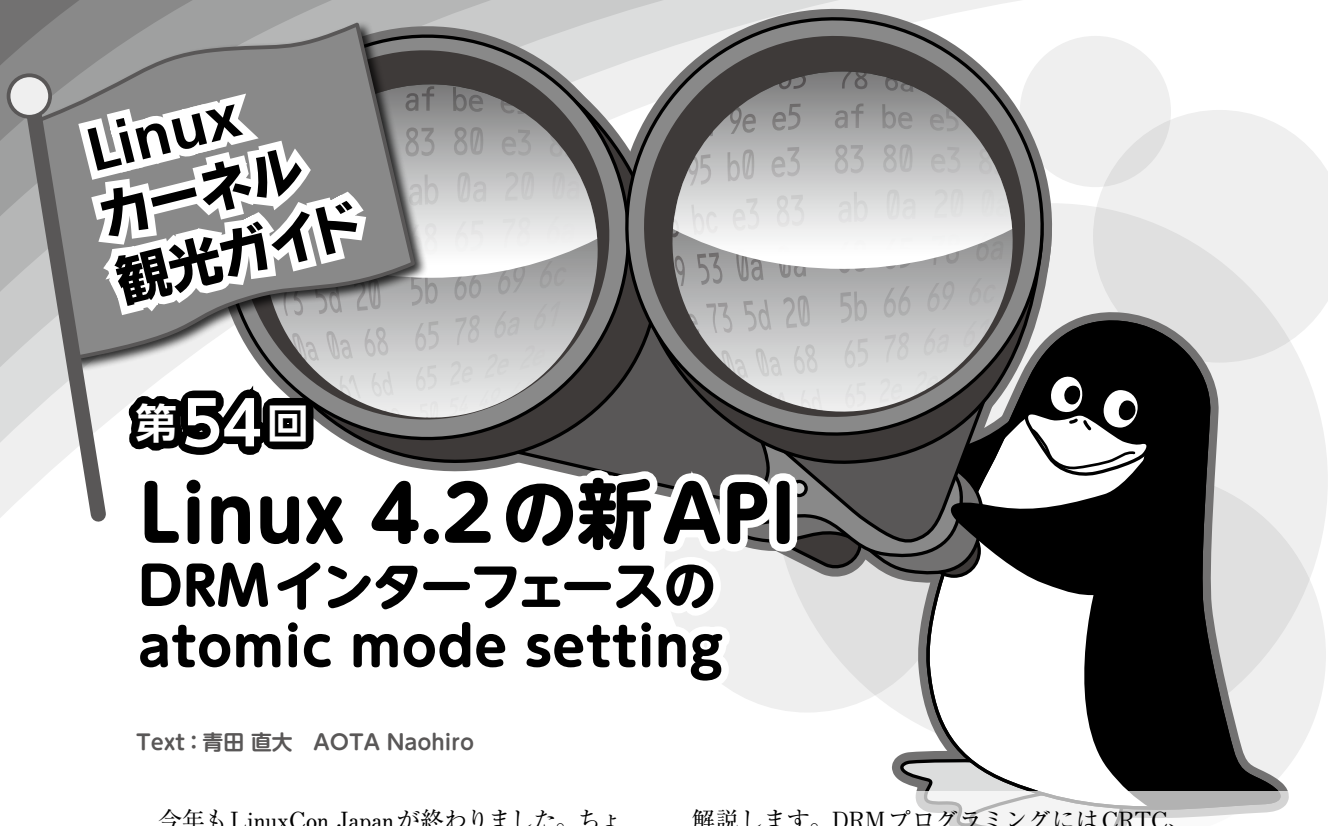
今回のまとめと次回について

移動やリンクの方法とリンクカウントや参照のしくみを理解しました。ファイルシステムの階層構造をたどるオプションを使ってサブディレクトリを処理できるようになりました。そして、リダイレクトを理解しました。これで、ファイルの基本操作は大丈夫ですね。

来月はちょっとレベルを上げて、キャラクタベースゲームの金字塔ローグと、ローグを自動で解くシステムロゴマチックを題材に、ソースコードからのビルド手順を見ていきます。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）】
mv(1), csh(1), ln(1), unlink(1), unlink(2), cat(1), tcsh(1), rm(1), rmdir(1), mkdir(1), ls(1), close(2), open(2), tail(1), null(4)
【以下はbashのhelpコマンドを使って確認】
for



第54回

Linux 4.2の新API
DRMインターフェースの atomic mode setting

Text：青田 直大　AOTA Naohiro

今年もLinuxCon Japanが終わりました。ちょうどRCのタイミングがいい時期だったので、「LinuxConでLinux 4.7がリリースされるかな？」と思っていましたが、そうはいかなかったようです。なんにせよ、この記事が出るころにはLinux 4.8の新機能patchも出そろっていることでしょう。

今月は、Linux 4.2のDRMの新APIであるatomic mode settingについて解説します。

DRM 概要

XやWaylandのような複雑なものから、boot splashを実現しているシンプルなplymouthまで、さまざまなアプリケーションがモニタへの描画を行います。これらのアプリケーションはlibdrmというライブラリを使い、/dev/dri以下のデバイスファイルにioctl(DRM API)でやりとりを行うことで、解像度を設定し、画面の描画に使われるバッファを設定し、ちらつかないように適切なタイミングでバッファの切り替えを行っています。

Linux 4.2ではカーネルのDRMインターフェースに新しくatomic modesetというものが追加されました。atomic modesetとはどういうものでどのように使うのかを見ていきます。

まずDRMを使ったプログラミングの概要から解説します。DRMプログラミングにはCRTC、Plane、Encoder、Connector、FrameBufferと5種類のobjectが登場します(**図1**)。

CRTCとはCRTコントローラの略です。CRT(ブラウン管)のディスプレイはほとんど見なくなりましたが、ソースコードの中ではしっかり生き残っています。これはビデオカード側の描画バッファであるスキャンアウトバッファを示すobjectです。Planeとは描画元と描画先をつなぐobjectで、どのFrameBufferのどの領域からどのCRTCのどの領域へ描画するかを管理します。Planeを使うことで、ビデオカードによる描画の合成が実現されています。

たとえばマウスカーソルは、カーソル用のPlaneがあれば、それを使って表示されています。Encoderは、CRTCから受け取ったデータをモニタへと出力するまでを管理するobjectです。どのCRTCに接続可能であるかを示す情報を持っています。Connectorは、信号の最終的な出力先でモニタにつながっている部分を管理するobjectです。出力先の種類(VGA、HDMI、Display Portなど)や、ディスプレイのID(EDID)といった情報を持っています。FrameBufferはアプリケーションが描画するデータを書くバッファを管理するobjectです。

アプリケーションの側からまとめてみましょう。アプリケーションはFrameBufferに描画したいデータを書き込みます。ビデオカードはPlaneに設定されたデータをもとに定期的にCRTCのバッファにFrameBufferに描画されたデータを合成します。合成されたデータはEncoderを通じてConnectorに送られ、モニタへの表示が行われます。

DRMプログラミング：描画先の選択

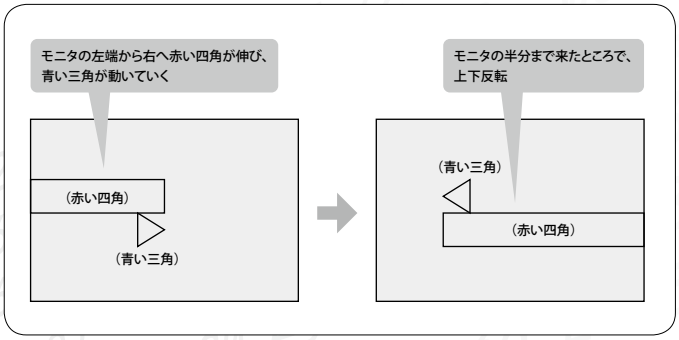
では、DRMを使ったプログラミングを見ていきましょう。今回はサンプルとして、メインのバッファに左から右へと伸びていく赤い四角を描き、その下にPlaneを使って青い三角形を右向きに、赤い四角の先頭部分と一緒に進むように合成するコードを書きます(**図2**)。さらに、赤線が画面の半分までいったところで描画スクリーンを180度回転してみます。すなわち、画面の半分までいったところで、これまで左端から右に伸びていた赤い四角が、右端から左へと伸びていくようになり、青い三角は赤四角の上に表示されることになります。

DRMのプログラミングは、まずカードデバイスをopenすることから始まります(**リスト1**)。/dev/driの下にビデオカードごとに“**card***”というファイルがあり、これをopenします。DRMプログラミングは、ここでopenしたファイルデスクリプタにioctlを発行することで実現されます。**リスト1**の例では**ioctl**をラップする**libdrm**を使っています。

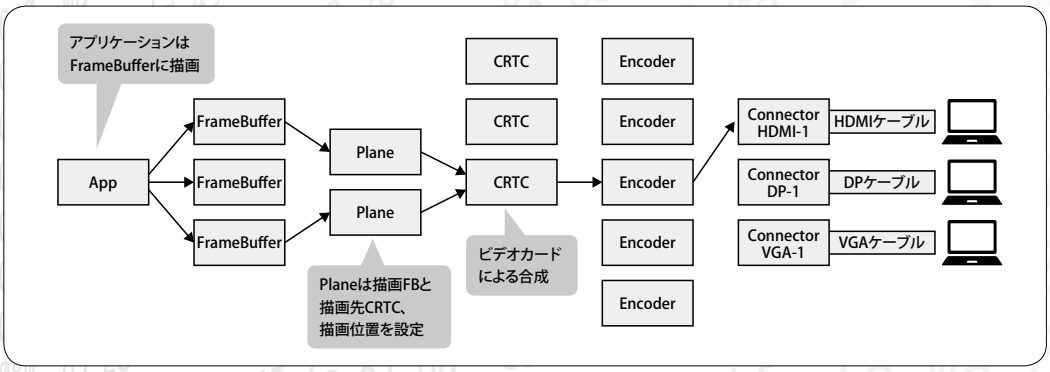
まず、**drmSetClientCap()**を使って**UNIVERSAL_PLANES**という機能を有効にします。CRTC全体を覆うPrimaryのPlaneやマウス用のカーソルPlaneも、通常のPlaneのように見えるようにする設定で、あとのatomic mode settingと動作をそろえるために有効にしています。

次に**drmModeGetResources()**を使って現在の環境を調べていきます。この環境からConnectorの数がとれるので、ひとつひとつ**drmModeGetConnector()**でConnectorの情報を取得していきます。**リスト1**の例ではモニタに接続されていて、かつEncoderが設定されているConnectorを探しています。適切なConnectorを発見したら、そこに接続されているEncoder、さらにそのEncoderにつながったCRTCを調べてそれらのIDを取得します。また、解像度やリフレッ

▼図2　DRMを使ったプログラミング例



▼図1　DRMによる画面出力の概念



シュレートなどのmodeの設定を読み込んでおきます。

Planeの取得

次に描画に用いるPlaneを取得します(**リスト2**)。**drmModeGetPlaneResources()**を用いて、Planeの数を取得し、各Planeを**drmModeGetPlaneResources()**で取得して調べていきます。PlaneはどのCRTCにも使えるわけではないので、描画先のCRTCに対応しているかどうかをまず調べます。

次にPlaneの種類を調べます。Planeには PRIMARY、CURSOR、OVERLAYの3種類が存在します。PRIMARYはCRTCの全体を覆うベースとなるPlaneです。CURSORは単語のとおりマウスカーソル用のプレーンです。OVERLAYは自由に使うことができるPlaneです。赤い四角を描くメインのPlaneとしてPRIMARYを、青い三角を描くPlaneとしてOVERLAYを選ぶ必要があります。種類を取得するにはPlaneの"**type**"という"プロパティ"を読みます。**drmModeObjectGetProperties()**で、該当Planeのプロパティのリストを取得し、**drmModeGetProperty()**を使って各プロパティの名前を調べ、"**type**"であればその値を読み込むといったコードになります。

▼リスト1　描画先の選択部分

```
int main(int argc, char **argv)
{
    drmModeConnector *conn;
    uint32_t crtc_id = 0, enc_id, conn_id;
    drmModeCrtc *saved_crtc;

    // カードデバイスのopen
    const char *card = "/dev/dri/card0";
    int fd = open(card, O_RDWR | O_CLOEXEC);
    FAIL_ON(fd < 0);

    // UNIVERSAL_PLANES を有効に
    FAIL_ON(drmSetClientCap(fd, ➥
DRM_CLIENT_CAP_UNIVERSAL_PLANES, 1));

    drmModeModeInfo mode;
    drmModeRes *res = drmModeGetResources(fd);
    FAIL_ON(!res);
    for (int i = 0; i < res->count_connectors; ++i) {
        conn = drmModeGetConnector(fd, res->connectors[i]);
        FAIL_ON(!conn);
        if (conn->connection != DRM_MODE_CONNECTED ||
            (enc_id = conn->encoder_id) == 0) {
            drmModeFreeConnector(conn);
            conn = NULL;
            continue;
        }

        // 適切なConnectorを発見
        drmModeEncoder *enc = drmModeGetEncoder(fd, enc_id);
        conn_id = conn->connector_id;
        crtc_id = enc->crtc_id;
        memcpy(&mode, &conn->modes[0], sizeof(mode));
        drmModeFreeEncoder(enc);
        drmModeFreeConnector(conn);
        break;
    }
⋮
```

FrameBufferの作成

次にFrameBufferを3つ作成します(**リスト3**)。2つはPRIMARY Planeに設定するメインの描画用で、1つはOVERLAY Planeに設定するために使います。**create_fb()**はサイズwidth x heightのFrameBufferを作成する関数です。

ioctl(DRM_IOCTL_MODE_CREATE_DUMB)でDumb Bufferを作り、**drmModeAddFB()**でFrame Bufferを作り、**ioctl(DRM_IOCTL_MODE_MAP_DUMB)**とmmapでFrameBufferの領域をプログラムのメモリにマップします。このmapの部分にRGB値を描くことで画面への描写が行われます。

modeとPlaneの設定

青三角のFrameBufferへの描写(コードは省略)が終われば準備は完了です。CRTCにPrimaryのFrameBufferを座標(0,0)から描画するように設定し、読み込んでおいたmodeを使って解像度を設定します(**リスト4**)。プログラムがコンソールで動いていれば、この設定をしたところでコンソールが消えて真っ黒に

▼リスト2　Planeの取得部分

```
    uint32_t plane_id = 0, main_plane = 0;
    drmModePlaneRes *plane_res = drmModeGetPlaneResources(fd);
    FAIL_ON(!plane_res);
    for (int i = 0; i < plane_res->count_planes; ++i) {
        drmModePlane *p = drmModeGetPlane(fd, plane_res->planes[i]);
        // 描画先CRTCに対応しているか?
        if (!(p->possible_crtcs & (1 << crtc_index)))
            continue;
        // Planeの種類の確認
        drmModeObjectProperties *props = drmModeObjectGetProperties(
            fd, p->plane_id, DRM_MODE_OBJECT_PLANE);
        for (int j = 0; j < props->count_props; j++) {
            drmModePropertyPtr prop =
                drmModeGetProperty(fd, props->props[j]);
            if (strcmp(prop->name, "type") == 0) {
                uint32_t type = props->prop_values[j];
                if (type == DRM_PLANE_TYPE_PRIMARY)
                    main_plane = p->plane_id;
                else if (type == DRM_PLANE_TYPE_OVERLAY)
                    plane_id = p->plane_id;
            }
            drmModeFreeProperty(prop);
        }
        drmModeFreeObjectProperties(props);
        drmModeFreePlane(p);
    }
    FAIL_ON(main_plane == 0 || plane_id == 0);
⋮
```

▼リスト3　FrameBufferの作成

```
struct framebuffer {
    uint32_t width, height;
    uint32_t dumb;
    uint32_t fb;
    uint32_t pitch;
    uint8_t *map;
    uint32_t size;
};

void create_fb(int fd, struct framebuffer *fb, uint32_t width, uint32_t height)
{
    fb->width = width;
    fb->height = height;

    // create dumb buffer
    struct drm_mode_create_dumb creq;
    memset(&creq, 0, sizeof(creq));
    creq.width = width;
    creq.height = height;
    creq.bpp = 32;
    FAIL_ON(drmIoctl(fd, DRM_IOCTL_MODE_CREATE_DUMB, &creq));
    fb->dumb = creq.handle;
    fb->size = creq.size;
    fb->pitch = creq.pitch;

    FAIL_ON(drmModeAddFB(fd, width, height, 24, 32, fb->pitch, fb->dumb, &fb->fb));

    // map buffer
    struct drm_mode_map_dumb mreq;
    memset(&mreq, 0, sizeof(mreq));
    mreq.handle = fb->dumb;
    FAIL_ON(drmIoctl(fd, DRM_IOCTL_MODE_MAP_DUMB, &mreq));
    fb->map = mmap(0, fb->size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, mreq.offset);
    FAIL_ON(fb->map == MAP_FAILED);

    return;
}
⋮
```

なります。X上で動かしている場合、XがCRTCを握っているのでここで失敗します。さらに、青三角のFrameBufferをCRTC上の位置を指定して合成させます。CRTC上の座標(0, ybase(=赤い四角の下))からLINE_WIDTH × LINE_HEIGHTの領域に、FrameBufferの座標(0,0)からLINE_WIDTH × LINE_HEIGHTの領域を合成しています。プログラム中で16bitシフトしているのは16bitの位置に小数点が置かれるためです。

またこのプログラムでは、最後に元のコンソールに戻せるように、saved_crtcに現在の画面設定を保存しています。

描画メインループ

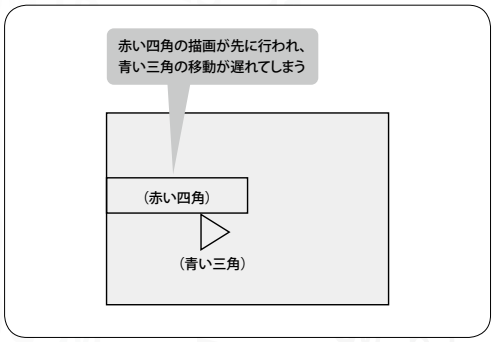
描画を行うメインループに入ります。**リスト5**では幅STEPずつの赤い四角を描画し、メインのバッファを切り替え、overlayの表示位置を更新しています。描画と切り替えから見ていきましょう。現在表示されているFrameBufferは`fb[front]`になっています。`fb[front]`に直接描画を行っても画面は更新されますが、タイミングによっては描画途中のデータが見えてしまい、画面のちらつきの原因となります。そこで先ほど2つ用意したFrameBufferを使って、ユーザには見えない「裏側」のFrameBufferに描画を行い、描画完了した時点でバッファを交換します。

裏側に描画作業を行い、`drmWaitVBlank()`で次のCRTCへのスキャンアウトまで待ちます。`drmModeSetCrtc()`でメインのFrameBufferを入れ換え、`drmModeObjectSetProperty()`で画面の回転を行い、`drmModeSetPlane()`で青い三角が描画されているPlaneを移動します。

以上のようなコードを書けば、とりあえずFrameBufferとPlaneを使ったプログラミングができます。ここで「メインのFrameBufferの切り替え」「メインのFrameBufferの回転設定」「Planeの位置調整」「Planeの回転設定」と4つの作業を4つの関数で行っていることに注目してください。これら4つの関数呼び出しはlibdrmを通じて、4回の`ioctl()`に対応します。`ioctl()`を呼び出せば、その時点でmodeやバッファが切り替わってしまいます。もしカードやドライバの問題でPlaneを動かす`ioctl`の前後に0.1秒ずつかかってしまう場合(少し無理そうな仮定ですが……)、メインの赤い四角が伸びるのがモニタに描画されたあとにPlaneが動いてしまい、**図3**のように常に赤い四角を青い三角が追いかけるような形に描写されてしまいます。

このように異なるCRTCやPlaneに対する操作がatomicに行われないことで、画面が乱れた

▼図3　表示がずれてしまう



▼リスト4　modeとPlaneの設定

```
:
// OVERLAY用Framebufferへの描画(省略)
:
    // save current CRTC
    saved_crtc = drmModeGetCrtc(fd, crtc_id);
    // set mode
    FAIL_ON(drmModeSetCrtc(fd, crtc_id, fb[front].fb, 0, 0, &conn_id, 1, &mode));
    // set plane
    if (drmModeSetPlane(fd, plane_id, crtc_id, fb[overlay].fb, 0, 0, ybase,
                LINE_WIDTH, LINE_HEIGHT, 0, 0, LINE_WIDTH << 16, LINE_HEIGHT << 16)) {
        fprintf(stderr, "failed at %d\n", __LINE__);
        goto clean;
    }
```

り意図しない描写が行われることになります。Linux 4.2ではこうした問題を解決するために、atomicにCRTCやPlaneへの操作を行うAPIが追加されました。

atomic mode setting

では、libdrmでatomicに操作を行うコードを見てみましょう。**リスト6**は描画が完了し、atomicにFrameBufferの切り替えやPlaneの移動を行う部分の抜粋です。まず**drmModeAtomicReq**のデータ**areq**を**drmModeAtomicAlloc()**を使ってallocationします。そして、**drmModeAtomicAddProperty()**を使って変更内容を**areq**に追記していきます。**drmModeAtomicAddProperty**は、操作対象のオブジェクトのID、そのオブジェクトで変更するプロパティのID、変更後の値を引数にとります。ここではメインのPlaneのFrameBufferを切り替え(FB_ID)、回転を設定(ROTATION)し、Planeを移動(CRTC_X, CRTC_Y)しています。areqのデータ構築が終われば、書き換えタイミングを待って**drmModeAtomicCommit()**を使い、FrameBufferとPlaneへのプロパティの更新をatomicに実行します。

こちらのAPIであれば、たとえPlaneの移動に時間がかかったとしても青い三角と赤い四角が一緒に進み、描写の問題は起きません。

まとめ

今回はLinux 4.2のDRMの新機能、atomic mode settingについて解説しました。Xなしでモニタに描画できるDRMプログラミングはなかなかおもしろいので、ぜひ試してみてください。SD

▼リスト5　描画メインループ

```
    for (int x = 0; x < width; x += STEP) {
        int back = front ^ 1;
        // draw main buffer
        // (省略)
⋮
        front ^= 1;
        drmWaitVBlank(fd, &vbl);
        FAIL_ON(drmModeSetCrtc(fd, crtc_id, fb[front].fb, 0, 0,
                    &conn_id, 1, &mode));
        FAIL_ON(drmModeObjectSetProperty(
            fd, main_plane, DRM_MODE_OBJECT_PLANE, ROTATION,
            (x > width / 2) ? 4 : 0));
        if (x < width / 2) {
            if (drmModeSetPlane(
                fd, plane_id, crtc_id, fb[overlay].fb, 0,
                max(0, x + STEP - LINE_WIDTH), ybase,
                LINE_WIDTH, LINE_HEIGHT, 0, 0,
                LINE_WIDTH << 16, LINE_HEIGHT << 16)) {
                fprintf(stderr, "failed at %d\n",
                    __LINE__);
                goto clean;
            }
        } else {
            // 座標が変わり，回転させるだけで同様
            // (省略)
        }
    }
```

▼リスト6　atomic mode setting

```
        front ^= 1;
        areq = drmModeAtomicAlloc();
        FAIL_ON(!areq);
        if (x < width / 2) {
            drmModeAtomicAddProperty(areq, main_plane,
                        prop_id[FB_ID], fb[front].fb);
            drmModeAtomicAddProperty(areq, main_plane,
                        prop_id[ROTATION], 0);
            drmModeAtomicAddProperty(areq, plane_id,
                        prop_id[CRTC_X],
                        max(0, x + STEP - LINE_WIDTH));
            drmModeAtomicAddProperty(areq, plane_id,
                        prop_id[CRTC_Y], ybase);
        } else {
            // 省略
        }
        drmWaitVBlank(fd, &vbl);
        if (drmModeAtomicCommit(fd, areq, 0, NULL)) {
            fprintf(stderr, "failed at %d\n", __LINE__);
            perror("");
            goto clean;
        }
        drmModeAtomicFree(areq);
```

September 2016

NO.59

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

北と南でも議論、コミュニティそれぞれの課題と解決策

6月に札幌、7月に沖縄で行った研究会の模様をお伝えします。いずれも、ここ1年ほど続けているITコミュニティをテーマに取り上げました。

jus研究会 札幌大会

■ITコミュニティの運営を考える
【講師】小楠 聡美（TEF道）、八巻 正行（LOCAL）、法林 浩之（日本UNIXユーザ会）、榎 真治（日本UNIXユーザ会/LibreOffice日本語チーム）
【日時】2016年6月18日（土）15:15～16:00
【会場】札幌コンベンションセンター 204会議室

札幌大会は、ソフトウェアのテストや品質管理に関するコミュニティ・TEF道を運営している小楠さん、同じく道内コミュニティの支援をするコミュニティ・LOCALの理事を務める八巻さんを講師にお迎えしました。参加者は20人でした。

■新人へのフォローはどうしている？

まず小楠さんから「新しく入ってくる人がコミュニティに馴染（なじ）めず定着しないのだが、どうやってフォローしているか」というお題が出されました。これに対しては、ほかのコミュニティでもフォローはあまりできていませんが、初参加者には必ず主催者から声をかける、全員で自己紹介をする、お菓子を提供して共通の話題を作るなどの工夫をしています。

■コミュニティのミッションは決めているか

次に榎さんから「団体のミッションや方向性はどうやって決めているか」というお題が出ました。LibreOfficeなどの開発コミュニティはミッションを明確にしやすいですが、TEF道は勉強したい人が集まっているだけで、団体としての目標は設定していないとのことです。LOCALは一般社団法人なので、理事会で方向性を決めて総会で決議しています。

■コミュニティの高齢化にどう立ち向かうか

八巻さんからのお題は「コミュニティの高齢化問題」でした。LOCALには学生部もあるのですが、就職で北海道を離れるケースが多く、高齢化問題への根本的な対策にはなっていません。新しい人を入れていかないと高齢化は避けられないので、TEF道では参加登録サイトをDoorkeeperにして新しい人が入りやすいようにしています。また、筆者と榎さんが実行委員を務める関西オープンフォーラムでは、年1回のイベント以外に今年から月例の勉強会を始め、そのリピーターをスタッフに勧誘しています。

■なぜ北海道コミュニティは盛り上がるのか／マンネリ化対策／コミュニティの解散

筆者からは「北海道のコミュニティやイベントはなぜこんなに盛り上がるのか？」というお題を出しました。道民である八巻さんと小楠さんは、道外から見て楽しそうに見えるからではないかと答えました。北海道という土地柄に人々を開放的にさせる何かがあるのかもしれません。

今回は会場からも質問を受け付けました。「長くやっていると同じネタが繰り返し出てきてマンネリ化するのはどうすれば避けられるか？」という質問

には、やりたいことの選択肢を増やせばマンネリ化しない、マンネリだと思えば自分が発表すれば良い、同じ話題でも4、5年経つと違う議論になるので気にしない、などの回答がありました。

「コミュニティには終わりの美学はあるか？」という質問には、目的を達成したら解散しても良いのだが、まだ目的を十分に達成した感触がないため解散を考えていないという回答が多かったです。

jus研究会 沖縄大会

■ITコミュニティの運営を考える
【講師】米須 渉（JAWS-UG沖縄）、西島 幸一郎（ハッカーズチャンプルー）、榎 真治（日本UNIXユーザ会/LibreOffice日本語チーム）、法林 浩之（日本UNIXユーザ会）
【日時】2016年7月2日（土）15:15～16:00
【会場】沖縄コンベンションセンター 会議場B1

沖縄大会は、AWSのユーザグループ・JAWS-UGを運営する米須さん、開発系コミュニティが集まるイベント・ハッカーズチャンプルーを運営している西島さんをお迎えしました。参加者は27名でした。

■コミュニティの目的とは

最初の議題は榎さんから出た「コミュニティって何？」でした。各講師からは、コミュニティにはイベント系のように人の輪を作ることに主眼が置かれているものと、開発系のように継続的に何かを作って貢献することを目指すものとの2種類があること、マイナーなもののほうが、コミュニティができやすいことなどのコメントがありました。JAWS-UG沖縄の場合は立ち上げ時の苦労が多く、うまく回るようになるまでに2、3年かかったそうです。

■勉強会の満足度を測るには

続いては西島さんの「勉強会やイベント参加者の満足度をどう測るか？」というお題でした。これに対しては、懇親会への参加率で測るとの意見がありました。しかし学生は金がないので懇親会に参加する人が少ない、沖縄では自家用車で来ているので帰る人も多いという課題もあるようです。ほかにはツイートの量で測定するなどの回答がありました。

■どうやって人を集めるか

米須さんからのお題は「どうやって新しい人を集めていますか？」でした。LibreOfficeでは、世界的にはGoogle Summer of Codeなどを利用して若者を集めることに成功していますが、日本ではまだできていないようです。学校関係はコミュニティ活動に関心の高い先生や先輩が引率する形で若い人が参加するケースが多いですが、特定の人に依存しているため永続性がないのが問題です。

また、若い人たちは自ら新団体を旗揚げするのもアリとの意見が出る一方で、開発系コミュニティで団体が分裂すると開発リソースが分断されて致命傷になるのでやめてほしいという意見もありました。

筆者からは「集客について、地元のキーマンに相談する以外の手段はないか？」というお題を出しました。沖縄ではコミュニティ数もそれほど多くなく、メンバーも重複しているので、コミュニティに参加するような人に情報を伝えるのはさほど難しくないとのことです。地域によっては行政などが持っている企業リストを通して情報を伝えることもできますが、自分たちが来てほしいような参加者が増えるかというと疑問であるという意見が多かったです。

■イベントは土日開催か、平日開催か

最後に米須さんから「土日開催と平日開催はどちらが集客できるか？」というお題が出されました。全体的には、平日は業務に役立つ勉強会が多く、趣味に近いものは土日に開催される傾向があります。

沖縄では平日に開催すると運営側が仕事を休まないと対応できないため週末の開催が大半ですが、最近は勉強会の回数が増えてきて、土曜日は複数のコミュニティが勉強会を行うことも珍しくなくなってきました。そこでGo言語の勉強会は平日の開催を試行しているそうです。SD

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
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha
佐伯 幸治　SAEKI Koji
Twitter @widesilverz

第57回 熊本地震での活動紹介（減災インフォ・情報支援連絡会議・Civic Tech Live!）

先月号のIT DARTによる熊本地震での活動に続き、今回は減災インフォの活動、情報支援連絡会議、Civic Tech Live!についてお届けします。

減災インフォの活動

減災インフォ[注1]は、平時には災害に関する情報収集と発信、災害時には遠隔・後方からの情報支援に取り組んでいるボランティア団体です。熊本地震では、発生時から情報の整理を始めていき、Webサイト、Twitter、Facebookで発信に取り組んできました。

原稿執筆時点（7月2日）では、発災から時間が経っていることもあり更新頻度は落ちていますが、振り返ってみるとWebサイトではカテゴリ別に13本の記事を作成、SNSではおもにTwitterで6月まではほぼ毎日のように熊本地震に関して発信しています。減災インフォの活動については、これらWebサイトやSNSでの記録を元に活動を紹介していきます。

現地状況の把握

発災時にまず取り組んだのが、どこでどのような災害情報が発信されているかを把握するための情報発信元を整理することでした。手始めとして、簡単な説明を添えて企業や団体をリストにして記事として公開しました。このリストには、大手メディア、現地メディア、内閣府防災・関係省庁、ボランティア団体、自治体、支援コミュニティといった企業や団体が含まれています。このリストを起点にして、メディアの報道や省庁の発表資料、現地の生の情報などを効率的に入手できるようにしました。

次に自治体の状況に関しての情報整理に着手しました。震度や避難者数などを含めた自治体別の被害情報一覧、自治体ホームページやSNSアカウントの有無と発信状況、熊本県内の自治体のTwitterアカウントリストをまとめました。自治体別の避難情報については、発表資料をもとに地図上に落とし込むことで、一見してわかりやすくなるような表現に取り組みました（詳しくは後述「理解しやすくするための見える化」に記載）。

これら自治体別の情報については、減災インフォでは平時の活動として、1,741自治体の公式ホームページやSNSアカウントなどの情報を整理する取り組みを進めており、今回の熊本地震ではこの活動が役に立ちました。平時に集めた自治体の情報は、減災インフォのサイトやオープンデータ[注2]として公開していますので、興味のある方はぜひご覧ください。

支援したい方向け情報の取りまとめ

前述の「現地状況の把握」でもわかるように、発災からしばらくは被害状況に関する情報が多く発信されていましたが、次第に支援に関する情報が増えてきました。そこで減災インフォでもボランティアや寄付に関する情報をまとめていきます。ボランティアでは「心構え」、「全国社会福祉協議会（社協）の情報」、「ボランティアツアー情報」、「自治体主体のボランティア情報」、「ボランティア向け宿泊施設」などの項目で記事を作成して随時更新、必要であれば社協関係の団体やYahoo!ボランティアなどへのリンクも設けて、ボランティア情報が入手しやすくなるような内容としました。

注1　http://www.gensaiinfo.com/

注2　http://linkdata.org/work/rdf1s4103i

寄付については、「義援金を受け付けている団体・企業」、「支援金を受け付けている団体・企業」、「ふるさと納税を利用した寄付を募っている自治体」の3つにわけて情報をまとめて更新していきました。

被災された方向け情報の取りまとめ

避難所が開設されていくとともに、自治体や省庁からは避難所生活における情報が発信されるようになります。熊本地震では災害関連死としてエコノミークラス症候群が大きく取り上げられたことから、厚生労働省や内閣府防災、政府応援情報[注3](今回の熊本地震で開設された省庁からの発信をまとめてアナウンスするTwitterアカウント)、自治体などの公的機関が心身の健康にまつわる注意喚起の情報を頻繁に発信しており、共有も多くされていたため、減災インフォでも健康に関しての取りまとめに着手しました。

また避難所生活からしばらく日数が経つに連れ、住まいやお金のことなど生活再建に関する情報を多く目にするようになります。これらの情報は有用かつ大切な内容ではあるのですが、普段の生活を過ごしている限りでは馴染みのないものが多く、被災された方にとってわかりづらいという声も聞かれたので、理解しやすくするためにイラストをメインにした表現での取り組みにもチャレンジしました(次項参照)。

理解しやすくするための見える化

理解しやすくするための見える化の取り組みとしては2つ挙げられます。1つは自治体別の避難者数について、自治体名と数字が掲載されている発表資料をもとに地図上に落とし込んだ取り組みです。これはデータを地図上に落とし込んで視覚的に表現できるCartoDBを利用しています[注4](**図1**)。

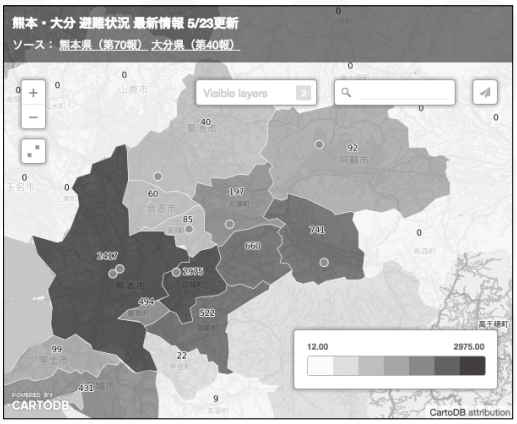
もう1つは住まいやお金に関する生活再建や健康などにまつわる情報をイラスト化したことです。省庁や自治体などから提供される情報は信頼性の担保はされているものの、文字が主で内容が複雑だったこともあり、そのままではとっつきにくく、人によっては敬遠してしまうかもしれません。そこで内容をイラスト化して簡易的に表現することで、読み込むきっかけとして利用してもらったり、自治体職員や士業の方々に説明資料として利用してもらうことを目指しました。内容のイラスト化については、グラフィックレコーダーの方々が集まっている団体のグラフィックレコーダーネットワークに協力を仰ぎ、「み絵るヘルプ」というプロジェクトを起ち上げてTwitterやFacebookでの発信に取り組みました[注5](**図2**)。

各団体や企業との連携

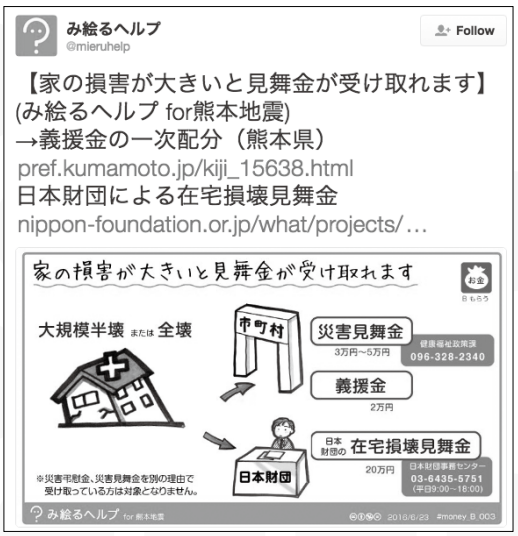
減災インフォはボランティアでの活動なのでどうしてもリソースが限られてしまいます。そこで熊本地震では、減災インフォと同様にHack For JapanのスタッフがかかわっているIT DARTに加え、社協関係の団体や大学生を中心とした支援コミュニティなどと連携しながら、手を広げすぎることなく、減災インフォでできることに配慮しつつ情報支援に取り組んでいました。また情報発信については、災害時ツイートの勉強会などで協力いただいたTwitter社からツイートのプロモーション枠を提供してもらう機会もあり、平時からの活動やつながりを災害時に活かして、連携していくことの重要性を実感しています。

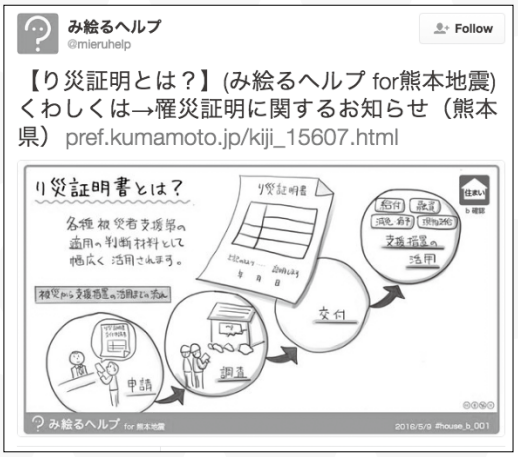
注3　https://twitter.com/kantei_hisai http://www.kantei.go.jp/jp/headline/saigai/kumamoto_hisai.html
注4　http://www.gensaiinfo.com/blog/2016/0420/3028/
注5　https://twitter.com/mieruhelp https://www.facebook.com/mieruhelp/

◆**図1　CartoDBを利用した避難者数のマッピング**



◆図2　Twitterアカウント「み絵るヘルプ」からのツイート。内容によっては自治体など元情報へのリンクも貼られています





情報支援連携会議

5月29日に「熊本地震 情報支援連携会議」が開催されました。この会議は情報支援に取り組んでいる団体・コミュニティ・組織・個人の方々が集まり、それぞれの取り組みを共有して、今後の支援につなげていくことを目的としたものです。会議では各団体が各自のテーマで発表を行いました（**表1**）。詳しくは減災インフォのサイトに各発表のレポートやスライドがアップされていますのでそちらをご覧ください[注6]。

また、熊本県でITリテラシーを解消する活動に取り組んでいるモバイル・ネットワーク研究所の松川さんより、発災時のことや復興に関する取り組みについて話を聞く機会もありました。なお松川さんは、復興支援として物産購入サイト「熊本 買って支援」[注7]を開設していますので、ぜひサイトを訪れてみてください。情報支援連携会議では多様な立場からの情報共有がなされ、より多面的に熊本地震を理解することができました（**写真1**）。

防災×シビックテック

5月26日に、Hack For Japanスタッフの関治之さんが代表を務めるCode for Japanによる「Civic Tech

注6　http://www.gensaiinfo.com/blog/2016/0604/3792/「イントロ編［熊本地震 情報支援連携会議］」
注7　http://kumamotokatte.jimdo.com/

◆表1　情報支援連携会議で行われた発表

テーマ	タイトル	発表団体
通信	通信環境（キャリア、WiFi）と今後の課題	iSPP
NPO連携	NPO連携火の国会議の運営について	JVOAD
NPO支援	ボランティア団体向け情報機器提供と情報サービスの支援	IT DART
全般	災害情報を共有する仲介役を目指して	防災科学技術研究所
	熊本地震から～シビックテックと防災	Code for Japan
	フェーズ毎の今回の活動と7つの特長～試行・連携のトライアル	減災インフォ・TKM47・み絵るヘルプ
地図	基盤情報としての地理空間情報と災害情報支援における課題	東京大学 空間情報科学研究センター
	避難所情報の集約と課題	IT DART
	大学生を中心としたgoogleMAPの展開	Youth for Kumamoto
企業の現地支援	Yahoo! JpanのIT支援と、熊本現地チームの連携	Yahoo! Japan
災害ボランティア	熊本ボランティア情報ステーションと福岡市ボラバス支援	ボランティアインフォ
	災害ボランティアセンターの情報発信支援	災害IT支援ネットワーク
マッチングプラットフォーム	スマートサプライを活用した物資支援	スマートサバイバープロジェクト

Live!」が開催されました。このイベントではみんなが当事者意識を持って、ITの力で社会をより良くするシビックテックをテーマに、飲み、語り合いながら、自分たちに何ができるかを考えます。5月に開催されたこのイベントでは、熊本地震にてボランティア活動に携わった方々に、被災地での活動内容や課題などを中心に防災とシビックテックを題材としたライトニングトークをしていただきました。その活動報告の中から、いくつかご紹介します。

熊本地震における特徴

NPO法人ETICローカルイノベーション事業部の川口枝里子さんによる熊本の現地での被害状況や特徴などのお話では、被災地でのさまざまな被害状況を説明いただきました。とくに避難生活を余儀なくされている家屋の倒壊状況については、5月16日時点で全壊、半壊、一部破損の合計で85,506棟あり、熊本県内で被害が集中している地域もあれば、少なかった地域もあるという非常に局所的な状況だったようです。また、東日本大震災と比較して、津波被害が起きなかったこともあり、復旧のスピードは非常に早く進んでいます。緊急支援の段階をいち早く終えた地域から、現地の人たちへバトンをうまく渡していき、いかに後方支援をしていくかが重要な取り組みになると締めくくられました（**写真2**）。

プリンターに課題があった

災害支援ITネットワークの代表である柴田哲史さんからのお話は、被災地での支援経験が豊富な人たちによる災害ボランティア活動支援プロジェクト会議、通称「支援P」[注8]での活動報告となりました。現地社協に代わって特設サイトの立ち上げや各災害ボランティアセンターのIT支援を行われたそうです。ネットワークの構築からパソコンの運搬とセットアップ、ホームページからFacebookページの運用までをこなしていましたが、その活動の中で非常に苦しめられたのが業務用複合機による印刷が1台のパソコンからしかできない問題でした。

これは非常時において、複数台からの印刷を行うための環境整備にかかる準備が整わず、家庭用のプリンターのほうが有効とのことでした（最新の業務用複合機ではこの辺りの問題は解決しているそうです）。

このほかにもさまざまな取り組みの紹介がありました。東日本大震災からの教訓が活かされ、初動の早さが目に見える形となって表れている報告会でした。

最後に

前号、今号とお届けした熊本地震における活動ですが、熊本の復興はまだこれからとなり多くの支援が必要です。現地でのボランティアだけでなく、物産購入の支援、観光支援、情報支援など、いろいろな支援の形がありますので、支援活動に参加してみてはいかがでしょうか。また減災インフォの活動は、平時、災害時にかかわらず、常に試行錯誤で取り組んでいる状況ではあるのですが、活動メンバーを募集していますので、興味をお持ちの方は減災インフォのサイトからお問い合わせください。SD

注8　http://www.shien-p-saigai.org/

◆写真1　情報支援連絡会議で開催されたワークショップ

◆写真2　熊本の被害状況を説明する川口さん

温故知新
IT むかしばなし

第58回

Intel 80286 ～究極の16bit CPUだったのか～

速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami

はじめに

16bit命令アーキテクチャで動作するIntelの16bit CPU(8088、4.77MHz、外部データバスは8bit)が搭載されたIBM-PCは、当時の8bit CPUに比べて動作スピード的にはあまり差がないものでした。しかし、1984年に発売されたPC/ATは、さまざまな拡張と高速な動作を実現した現在のPCアーキテクチャの礎となるものでした。今回は、このPC/ATに搭載されx86 16bitアーキテクチャの最後を飾る高速／高機能な16bit CPUであるIntel 80286(以下i80286)についてのお話をしましょう。

i80286とは

8bitのIntel 8080の命令アーキテクチャを伝承して16bit化したIntel 8086(以下i8086)は、IBM-PCに遅れること1年の1982年に発売されたNEC PC-9801に搭載され、我が国における16bitコンピュータのスタンダードになりました。i8086/i8088が載ったパソコンが広く使われるようになるとその機能の不足が問題視され、高機能化を求める声が高まりました。それに応えるよう、i8086を高機能化して登場したCPUがi80286でした。

1982年2月の発表のあと、1984年8月に6MHzの動作クロックのi80286を搭載したPC/ATが登場しました。PC-9801シリーズでは、1986年にPC-9801VXが10MHzのi80286が搭載されて発表されました[注1]。

i80286は、次のような機能強化が行われています。

❶CPU実行速度の大きな高速化
❷メモリ領域の拡大
❸信頼性の高いメモリ保護と割り込み管理機能の追加
❹高速性と信頼性を両立させたマルチタスクの実現

CPU実行速度の高速化

i80286の内部ブロックは大きく変更され(**図1**)、4つの内部ユニットに分割されています。これらのユニットが並列に動作することでパイプラインを実現しています。i8086でもバスユニットと実行ユニットが分割されており、命令の読み込みであるフェッチ動作と命令の実行動作を並列に実行できましたが、フェッチと実行には要するクロック数に差があるため、効果的なパイプライン動作は実現できていませんでした。i80286では、実行ユニット(EU)と命令ユニット(IU)を分割することで、ユニットごとの機能を低減でき、各ユニットの動作時間の差を少なくして効率的なパイプラインを実現しています。

またバスユニット(BU)に6byteのプリフェッチキュー(i8086にも存在する)と命令ユニットにデコード済みの3つの命令を格納するエリアにより、各ユニットの実行時間の差を吸収しています。

さらに、高機能化したバスユニットによりメモリのアクセスが速く動作しています。以上のような機能アップにより、かなりの高速化が図られていますが、倍の高速化までは達しないはずです。では**表1**のように3倍近い高速化はどうして実現しているのでしょうか。

実は、i80286には、システム

注1) 1985年5月にハイレゾグラフィック機能を有したPC-98XA i80286 8MHzが発表されていますが、互換性からPC-9801シリーズとはしていません。

クロックとプロセスクロックの2種類の外部クロックがあります。i8086では動作クロックといえばシステムクロックを指しますが、i80286ではプロセスクロックとしており、プロセスクロックはシステムクロックの1/2なのです。すなわちPC/ATのプロセスクロック 6MHzのi80286は、システムクロックが12MHzなのでした。

メモリ領域の拡大

i80286では、従来のi8086と互換の16bitアーキテクチャであるリアルモードと拡張されたプロテクトモードがあり、21bit以上(アドレス：FFFFHより上)のアドレスを使用するにはプロテクトモードにする必要がありました。

プロテクトモードでは、セグメントレジスタはセグメントセレクタとなり、機能が変わります(**図2**)。メモリブロックの情報をセットされているテーブルが物理メモリ上にあり、セグメントセレクタはそのテーブルのインデックス値を示すことになります。それが示すメモリブロック情報を使って、**図2**のように❶～❹の手順でアクセスが行われます。

マルチタスクOSの実現

マルチタスクOSに必要な機能が備わっているため、それを活用したMS-DOSのレベルアップとなるOSの登場が期待されていました。それがOS/2であり、Microsoft社とIBM社の共同で開発が進められていたのですが、開発は大きく遅延しました。

おわりに

Unixの原型を築くために活用されたDEC社のミニコンピュータPDP-11は、完全な16bit CPUであり、使えるメモリエリアも当初は64KBでしたが、利用が広がるにつれてメモリの拡張が要求されi80286と似たメモリブロック管理テーブルによるメモリ拡張が行われました。しかし、その直後の1977年に32bitコンピュータ VAX-11が登場してPDP-11の役割は急速に減速してしまいます。それから8年後のx86におけるCPUの変遷も同様であったのかもしれません。

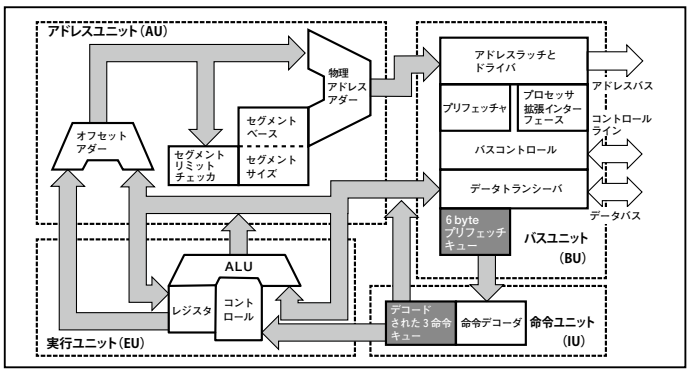
i80286は、16bit CPUとして究極な機能を持っていましたが、それはほとんど生かされずに高速なi8086互換のCPUとして使われ、その優れたアーキテクチャはi80386に引き継がれていったのです。SD

▼表1　i80286のスピード

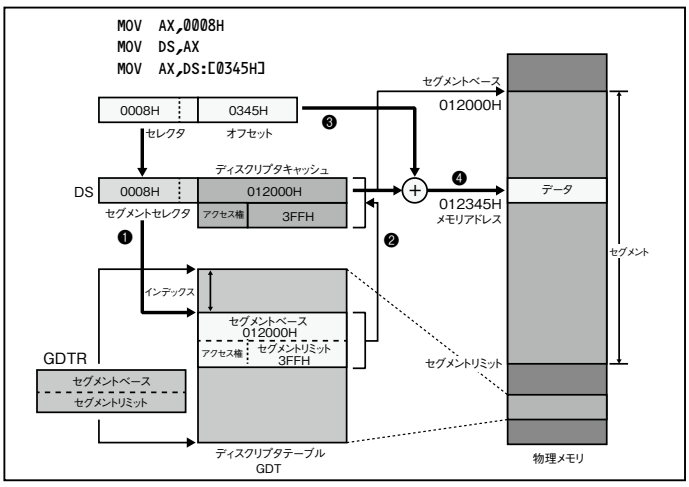
	i8086 5MHz	i80286 10MHz
Ratio to PC9801	1	6
データ転送命令	必要クロック数	
レジスタ←即値	4	2
メモリ←レジスタ	8+EA※	3
レジスタ←メモリ	9+EA※	5

※Effective Address(実行アドレス)に要するクロック数

▼図1　i80286の内部ブロック図



▼図2　プロテクトモードでのメモリアクセス動作



Hardware

オープンハードウェアとWeb技術でWoTの世界を多くの開発者に
CHIRIMENプロジェクト始動！

Author CHIRIMEN Open Hardware コミュニティ 赤塚大典

CHIRIMEN とは

CHIRIMENとはWebデベロッパのためのwot[注1]デバイス開発環境である。センサやアクチュエータもすべて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を開発できる。

開発環境はOSとしてB2G（Boot to Gecko）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ている。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する。

本稿ではCHIRIMENのハードウェアとソフトウェア、その開発方法、開発主体であるCHIRIMEN Open Hardwareコミュニティの活動内容について紹介する。

CHIRIMEN開発環境

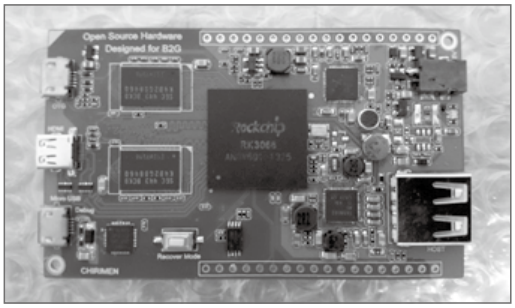
CHIRIMENは、**写真1**に示すハードウェアおよびその上で動作するソフトウェアの双方で成立する。

・ハードウェア

開発した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合わせている。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となる。また利用しているSoC[注2]にはGPUを搭載しているため、マルチメディアの再生も可能である。Wi-Fiなどのネットワーク環境は搭載されていないが、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ためである。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となる。**表1**に具体的なハードウェア仕様を示す。

・ソフトウェア

執筆現在、搭載されているOSはFirefox OSの開発版、



▲写真1　CHIRIMEN

B2Gである。B2GはFirefoxブラウザと共通のブラウザエンジンGeckoを搭載したWebコンテンツ・アプリケーション実行環境を備えたOSであり、OS起動時にWebブラウザだけが起動する。したがってそのアプリケーションはすべてWeb技術で構成される。CHIRIMEN Open Hardware projectでは、GPIOおよびI²CをWebページからJavaScriptだけで制御可能にするAPI「WebGPIO」および「WebI2C」を開発し、その仕様をW3Cに提案中である。同APIをGecko上に実装しており、アプリケーションやWebページと同じコンテキストから周辺ハードウェアの制御を可能としている。

参考までに、本環境で「Lチカ」を実現するコードを**リスト1**に示す。詳細は次号にて述べるが、JavaScriptに詳しい方であれば、Webページ上で何かを点滅させる方法と大差ないことがおわかりいただけると思う。

・アプリケーション開発

CHIRIMENをUSB接続したホストPC上で、B2Gアプリケーションの開発同様、Firefoxブラウザに搭載されたWeb IDEを用いて開発する。WebページおよびGPIO、I²Cの制御すべてをJavaScript、HTML、CSSで開発するため、WebページとWoTデバイスアプリケーションはほぼ同等に扱うことが可能で、Webページ・アプリを作るようにWoTデバイスアプリケーションを開発できる。なお、CSSのみでデバイスを制御するPCSS（Physical CSS）という開発環境も平行して開発している。スタイルシートが画面の中の表示を変えるものでなく、実世界をもスタイリングしてしまう思想へ昇華されているおもしろさがある。

これらの環境によって開発の敷居が低くなり、Webデザイナを含めたWebにかかわるさまざまな方がデバイス製作にもかかわっていただけるものになると信じ、また、そうなってほしいと考えている。

注1) WoT（Web of Things）：Internet of Things（IoT）というスローガンが知られているが、そこで共通化されるのはネットワークのレベルである。WoTはその上位のコンテンツ・サービス・アプリケーション実行環境を共通化し、それらの連携をより容易にすることを目指している。

注2) SoC（System on Chip）：従来コンピュータを構成するにはCPU、MMU、GPU、I/Oブリッジなど多数のプロセッサを組み合わせる必要があったが、超小型化が進むに従い、これらのプロセッサを1つのパッケージに統合したLSI（SoC）が主流となってきた。

```
document.addEventListener("DOMContentLoaded", function() {
  (blinkThread = (function*(param) {
    const port = yield getGPIOPort(198, "out", blinkThread);
    let value = 1;
    setInterval(() => {
      port.write(value);
      value = value === 1 ? 0 : 1;
    }, 1000);
  })()).next();
});

function getGPIOPort(portNumber, inOrOut, thread) {
  let port = null;
  navigator.requestGPIOAccess().then(gpio => {
    port = gpio.ports.get(portNumber);
    return port.export(inOrOut);
  }).then(() => {
    thread.next(port);
  });
}
```

▲リスト1　Lチカのコード例

CHIRIMEN Open Hardwareコミュニティの活動

我々の具体的な活動としては、❶CHIRIMEN開発環境のハードウェアおよびソフトウェアの開発、❷その標準化、❸ユースケースの開発、❹ワークショップや展示、などがある。

このような活動を支える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、運営している。本コミュニティは上記のように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、国内外あわせて多種多様な職種の方々で構成されている。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カルチャも併せ持つのがCHIRIMEN Open Hardwareコミュニティの特徴である。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る。また、さまざまな意味で国際的活動となっているのも特徴といえる。

❶CHIRIMEN開発環境の構築

WebGPIOおよびWebI2Cの実装、より開発のしやすい環境の提案や実装、教育用途を対象としたキット提供、およびさまざまなセンサ、アクチュエータとの連携テストや確認を行っている。また、Linux+Firefoxをベースとした開発環境の構築も提案されており、時期をみて取りかかることになると思われる。

❷標準化

WebGPIOおよびWebI2C APIの仕様策定と標準化をW3Cのコミュニティグループ Browsers and Robotics Community Group内で行っている。

❸ユースケースの開発

開発環境だけでなく、どのようなアプリケーションが考えられるか、プロトタイプを製作し実験している。このプロトタイプが新しいWebの使い方を示す糸口となる。

❹ワークショップや展示

ワークショップも比較的頻繁に開催している。そこではタッチ＆トライ的な側面のイベントもあれば、実際にモノを作るイベントなども行っている。

これらの活動方針は、基本的にはFacebook Group[注3]、Slack、および月に一度を目処に開催されるマンスリーミーティングで決定される。このようにCHIRIMENの検討は多種多様な技術、職種、思想、国が交わるコミュニティ活動により成り立っており、執筆時点においても道半ばにある。もし、Webデザイン、組み込み技術、ハード開発など、少しでも自身の得意分野・興味分野をお持ちの方がいるならば、CHIRIMEN Open Hardwareコミュニティに参加いただきたい。きっとどこかでご自身のスキルが役に立ち、未完成のCHIRIMENをより良いものにできると確信している。一緒にいいものに作り上げていかないだろうか。

なお、CHIRIMEN環境でのより詳しい開発手順やCHIRIMEN Open Hardware projectの誕生した背景などは次号に掲載予定である。乞うご期待。

注3) https://www.facebook.com/groups/chirimen/

●表1　ハードウェア仕様

SoC	RK3066 ARM Cortex A9 1.6GHz dual core, Mari 400 GPU quad core
Memory	DDR3 1GB（RAM）
Storage	NAND Flash 8GB, MicroSD slot × 1
Size	80mm × 48mm
Power	5V 1A via dedicated power connector
Interface	micro HDMI female、USB（microUSB × 1（OTG), USB × 1, microUSB × 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）, I^2C × 2, UART × 2, SPI × 2, Audio analog stereo IN × 1/ OUT × 1, PWM × 1, Analog IN × 1

CONTACT
CHIRIMEN Open Hardware project
URL **https://chirimen.org/**

Software

グレープシティ、「SpreadJS 9J」のリファクタリング版を提供開始

グレープシティは7月27日、ブラウザ上でMicrosoft Excelライクなユーザインターフェースを実現できるJavaScriptライブラリ「SpreadJS 9J」のリファクタリング版を提供開始した。

SpreadJSは、HTML5のCanvas上にクライアント側UIウィジェットとしてスプレッドシートを描画し、Excelライクな操作性と、スパークラインや条件付き書式、グループ化やフィルタリングなど高度な機能も含めた豊富なExcel互換機能を提供する。

今回のリファクタリングでは約800KBあったコアモジュールを約400KBまで軽量化、これまでjQueryの参照が必須であったものを非依存にし、高速化、さらにAPIをよりシンプルに使いやすくするように見直し、可読性と記述性を高めた。また新機能として、次の4つが搭載された。

- **Excel入出力機能の強化**
  これまで提供していた`ExcelIO`サービスに代わり、クライアントサイド`ExcelIO`と`ExcelIO`コンポーネントの2種類のExcel入出力機能を提供。クライアントサイド`ExcelIO`は、クローズドな環境でも利用できる
- **印刷機能**
  専用APIによるシートの印刷機能を提供。ブラウザでの画面印刷では見切れていたシート全体の印刷などが可能になる
- **テーブルスライサー**
  テーブル作成時に利用できるスライサー機能を追加。ボタンによる視覚的なフィルタリングが可能になる
- **新テーマへの対応**
  新たにExcel2013スタイル（3種類）、Excel2016スタイル（2種類）にも対応

なお、リファクタリング前の製品は「SpreadJS Classic」として、製品パッケージに同梱される。価格は120,000円（税抜き）となっており、対応ブラウザはInternet Explorer 9以上、Microsoft Edge、Chrome、Firefox、Safari 5.1以上、iOS（Safari、Chrome）。

CONTACT
グレープシティ㈱ URL http://www.grapecity.com

Report

「LinuxCon+ContainerCon Japan 2016」開催、SUSE Linux CTOインタビュー

The Linux Foundationによる国際技術カンファレンス「LinuxCon+ContainerCon Japan 2016」が7月13～16日にかけて椿山荘（東京都文京区）にて開催された。セッション登壇などのためSUSE Linuxの経営陣が来日、CTOのThomas Di Giacomo氏に取材の機会を得た。

――SUSE Linuxのこれまでの活動について

SUSEは1991年からエンタープライズLinuxを開発してきました。フォーチュン掲載企業100社のうち2/3は当社のOSを利用しています。最近ではOpenStackへの貢献も活発です。HPC（High Performance Computing）については東工大とも協力関係にあります。ここ数年、さまざまな面で二桁台の成長を果たしています。当社は、初期の10年はLinuxのエンタープライズ使用に注力してきました。そして次の10年はMicrosoftやSAPなどとのパートナーシップの拡大です。さらにIntel、AMD、富士通とも協力関係を深めてきました。2010年からはデータセンターの近代化に力を注いでいます。これにはCephなどのSDS（Software Defined Storage）やResource Orchestrationが挙げられます。さらにOpenNFV、OpenDaylightにもカーネルアップデートで対応しています。また、PaaS基盤としてCloudFoundryにもかかわっています。これからの課題としてはARMへの対応で、年末には正式発表します。SAPについては推奨環境です。とくにSAP HANAの95%はSUSEで稼働しています。

――Red HatはRHELのMicrosoft Azure対応で活発なプロモーションをしているが、SUSEの戦略はどうか

Microsoftとは10年以上協力関係にあります。技術的な優位点としてはHyper-VのサポートとSUSE Managerの利用です。さらに、AzureでのHPC対応はSUSEだけです。

――日本市場の見込みについて

いわゆる無償のLinuxは、ビジネスでたしかに使用できますが管理が手間です。SUSEの導入でそれが解決できます。日本市場は少し特殊で、ほとんどの顧客がIBMのメインフレーム上でSUSEを使用していて、それが安定したビジネスになっています。SAP HANA対応は戦略上のアドバンテージです。

CONTACT
ノベル㈱ URL **http://www.novell.com/ja-Jp**
The Linux Foundation URL **http://www.linux-foundation.jp**

ユニーク、
電子マネー残高表示機能付パスケース「miruca」発売

㈱ユニークは、電子マネーの利用環境に合わせた、ICカード電子マネー専用の残高表示機能付パスケース「miruca（ミルカ）」を7月中旬に発売した。

mirucaは、電子マネーの残高が確認できるパスケース。液晶下のボタンを押すと液晶部分に電子マネーの残高が表示され、また電子マネーを使った際には、残高が自動で一定時間表示される。ケース本体にはラバーコーティングが施されており、手に馴染む滑りにくい加工になっている。電源はCR2016電池1個。

対応電子マネーは、Suica、PASMO、nanaco、ICOCA、Kitaca、manaca、TOICA、PiTaPa、はやかけん、nimoca、SUGOCA、Suica・PASMOを搭載したICキャッシュカード・クレジットカードなど。価格は2,980円（税抜き）。

▲miruca

CONTACT
㈱ユニーク URL **http://www.uniqstyle.co.jp**

ソラコム、
新サービス「SORACOM Gate」「SORACOM Door」を発表

㈱ソラコムは7月13日、IoTプラットフォーム「SORACOM」において、2つの新サービス「SORACOM Gate」「SORACOM Door」を発表した。

SORACOM Gateは、IoTシステムとデバイスとの間に仮想的なL2ネットワークを張り、両者の直接通信を実現するサービス。これにより、セキュリティを担保しつつ、プライベートIPアドレスを利用してサービスにアクセスしたり、SSHなどでログインしてメンテナンスを行ったりすることが可能になる。

SORACOM Doorは、任意の環境に構築されたユーザのシステムとSORACOMとを、AWS Direct Connectを活用して専用線で直接接続するサービス「SORCOM Direct」の兄弟サービス。Directでは専用線で接続していた部分を、インターネット上に構築したVPNトンネルで代用する形での接続を可能とするもので、安全な経路での通信をより安価に実現できる。

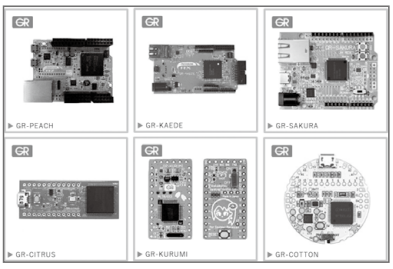
また同社は同日、SORACOM Global PoCキットの申し込み受付を開始した。本キットは世界の120を超える国と地域で、SORACOMのすべてのサービスが利用できる。価格は49,800円（税込み）で、30回線（30枚のSIM）の初期費用・基本料金、およびSIMの送料を含んでいる。

CONTACT
㈱ソラコム URL **https://soracom.jp**

ルネサスエレクトロニクス、
「GRデザインコンテスト2016」のエントリー受付開始

ルネサスエレクトロニクス㈱は7月23日、「GRデザインコンテスト2016」のWebエントリー受付を開始した（**http://gadget.renesas.com/ja/contest/2016/**）。

本コンテストはルネサスエレクトロニクス㈱が提供するプロトタイピングボードを用いたアイデア作品のコンテスト。1次選考（9月5日）では形にする前のアイデアが審査され、選出された100名に対して、「GADGET RENESAS」のワンボードコンピュータが提供される。2次選考（11月18日）ではそのボードを用いたプロダクトの完成度や実用性などが審査される。最終選考（12月10日）では、その作品のデモが審査される。賞金総額は150万円。1位は賞金50万円となっている。



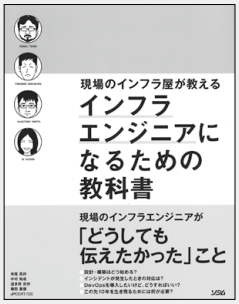
▲GADGET RENESASのボード。6種類の中から1つを選択する

CONTACT
ルネサスエレクトロニクス㈱
URL **https://www.renesas.com/ja-jp**

Python機械学習プログラミング

Sebastian Raschka 著／㈱クイープ 訳／福島 真太朗 監訳
B5変形判／464ページ
4,000円＋税
インプレス
ISBN＝978-4-8443-8060-3

機械学習は、コンピュータで人間並みの学習能力機能を実現させる研究で、科学シミュレーションや音声認識など、さまざまな分野への活用が期待されている。機械学習のソフトウェアの実装には、豊富なライブラリと平易な文法を持つPythonが使われることが多い。本書では、SciPy、NumPy、scikit-learn、matoplotlib、pandasといったライブラリを使ったPython 3によるプログラムを示しながら、機械学習において必要とされる処理を説明していく。そこで使われるアルゴリズムは数学の理論と複雑な数式で詳細に説明されており、技術書というよりは科学書の印象を受ける。書籍の難易度は高いが、Pythonのコードは読みやすいので、ひとつひとつの処理を追いながら、少しずつ理解していけるだろう。

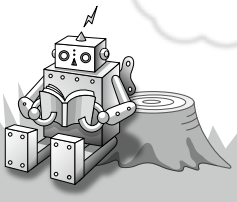


インフラエンジニアになるための教科書

寺尾 英作、中村 知成、波多野 安衣、横田 真俊、JPCERT/CC 著
B5変形判／296ページ
2,680円＋税
ソシム
ISBN＝978-4-8026-1043-8

本書はインフラ技術者の業務全般について解説する。最近この手の本が多いが、技術そのものよりシステムの構築／運用に関するノウハウを扱っているためか、本（著者）ごとに重きを置いているテーマや、書かれているノウハウに違いがあるのが興味深い。本書は、章ごとにその分野の専門家が執筆しているのが特徴だ。とくに、VPS／クラウド／ベアメタルなど各種インフラサービスの特徴について論じた2章と、システムの検討／構築／運用／障害対応などを扱った3、4章が、著者の経験に裏づけられた情報が多く盛り込まれていると感じた。インフラの分野も技術の流行り廃りが著しいが、本書ではコンテナ技術、Immutable Infrastructure、ChatOpsなどの新しい話題にも触れている。最新の情報に遅れないよう目を通しておきたい。

SD BOOK REVIEW



プログラミング言語Go

Alan A.A. Donovan、Brian W. Kernighan 著／柴田 芳樹 訳
B5変形判／464ページ
3,800円＋税
丸善出版
ISBN＝978-4-621-30025-1

2009年にGoogleから発表されたコンパイル言語「Go」はシンプルな言語仕様と高速性で、エンジニアから人気を集めている。本言語の開発にはC言語の開発者の1人であるケン・トンプソンがかかわり、また本書の著者には「K&R本」のブライアン・カーニハンが名を連ねるなど、豪華な顔触れだ。本書はそんなGoについて、プログラミング言語の基本（文法、制御構造、データ型、関数など）を前半で説明し、Goの特徴的な機能（並行プログラミング、リフレクション、低レベルプログラミング）を後半に説明している。とくに並行プログラミングについては、時計サーバやチャットサーバを作りながら、Goの並行プログラミングを実現している2つの要素「ゴルーチン」「チャネル」を詳細に解説している。

Slack入門

松下 雅和、小島 泰洋、長瀬 敦史、坂本 卓巳 著
A5判／208ページ
1,980円＋税
技術評論社
ISBN＝978-4-7741-8238-4

本誌2016年1月号でも特集したチャットコミュニケーションツール「Slack」の日本初の解説書。親しみやすいUIとたくさんのサービスと連携できる機能（App Directory）が人気を呼び、Slackはエンジニアを中心に今もユーザ数を増やしている。最近では大手企業での導入も始まっているようだ。本書は「入門」と冠しているものの、中級者向けにSlackを活用するためのHubotやCIツールとの連携についても触れているため、ChatOpsを始めたい方でも参考にできる内容になっている。また、Slackは日本語化されておらず、英語が苦手な方にとってはまとまった資料として有用である。視覚的な情報が多く、またフルカラーなので、Slackの楽しげな雰囲気が伝わってくる1冊になっている。

ひみつのLinux通信

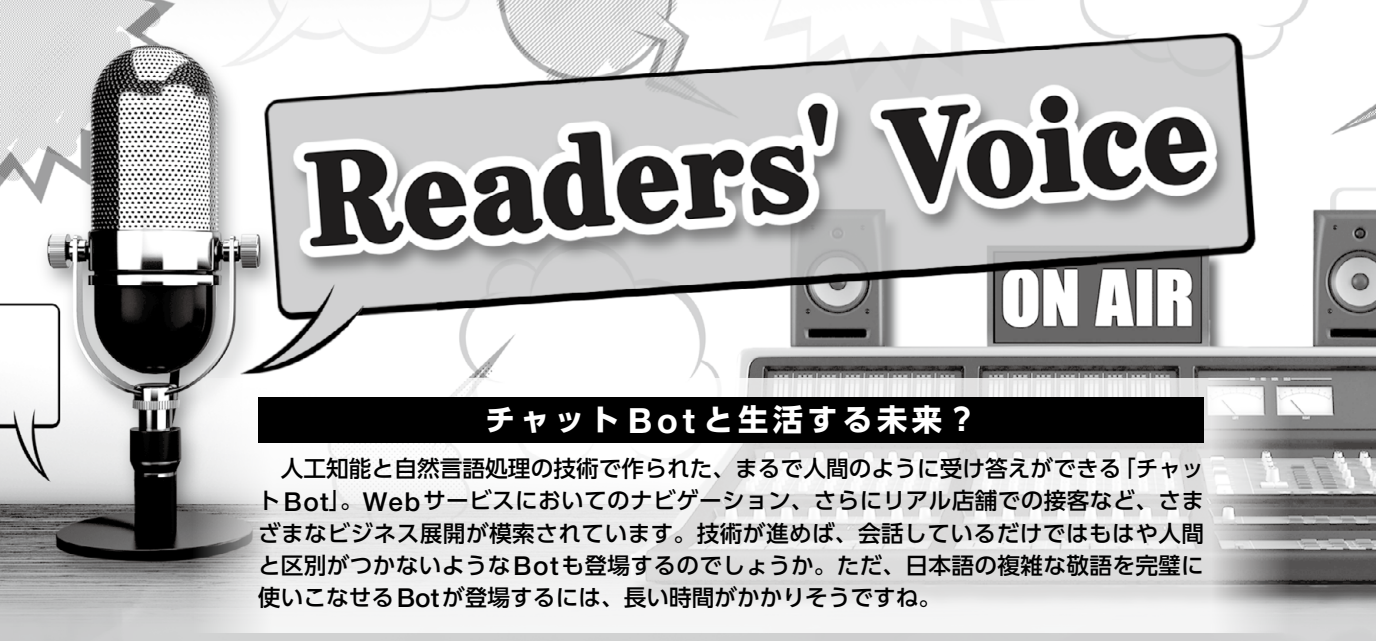
作)くつなりょうすけ
@ryosuke927

第31回 モジュールを読み込むように進化したい

トリニティがnmapでポートスキャンするシーンでほくそ笑んだ、君！すでに老害ですから！——と自分にブーメランする担当編集の咆哮が読めるのは本誌だけ。

to be Continued

歌って踊れるエンジニアが夢ですが、歌えないし踊れないしエンジニアとしても特に自信はありません。この業界、勉強は欠かせないのに時間は有限、現金も有限、お腹も空くし、睡眠も削れない。手軽にスキルアップを、Linuxカーネルモジュールのようにinsmodやmodprobeできないか脳内検討した。映画『MATRIX』にもそんなシーンがあったね。ああいう世界がそこまで来てるんだ、ってワクワクしましたね。機械の世界には恐怖しましたが。アレが普通になると学校とか行かなくていいから、さらにコミュ障になりそうだし、そういうロボットのほうが先に実用化されそうだし、アレはオレに有用じゃない！——って結論にいたりました。やっぱり、日々努力して歌って踊れるエンジニアになります。



チャットBotと生活する未来？

人工知能と自然言語処理の技術で作られた、まるで人間のように受け答えができる「チャットBot」。Webサービスにおいてのナビゲーション、さらにリアル店舗での接客など、さまざまなビジネス展開が模索されています。技術が進めば、会話しているだけではもはや人間と区別がつかないようなBotも登場するのでしょうか。ただ、日本語の複雑な敬語を完璧に使いこなせるBotが登場するには、長い時間がかかりそうですね。

2016年7月号について、たくさんの声が届きました。

第1特集　プログラマが知っておくべきTCP/IP

プロトコルとネットワークモデルの解説から、C、JavaScript、PHP、Python、Rubyでのネットワークプログラミング、さらにはWiresharkでのパケットキャプチャまで、プログラマが手を動かしながらTCP/IPを学べる大特集でした。

抜け落ちている知識の総ざらいができました。 尾崎さん／東京都

及川さんが書いたというのを聞いて買いました。 twoさん／東京都

通信の基本ですよね。初心に帰らなくっちゃ。 kmさん／愛知県

私たちが日ごろお世話になっているインターネットの基礎であるプロトコルについて、あらためて学びなおすことができる良い企画でした。 オミオさん／宮城県

普段意識していないTCP/IPについて学習できたのが良かった。 thさん／新潟県

TCP/IPのしくみからWiresharkまでの流れが良かった。 三浦さん／大阪府

こういう基礎的な記事は人に教えるとき役に立つし、己の記憶のリフレッシュもできるので定期的に読んでいきたい。 うたさんさん／大阪府

Software主体の雑誌にネットワークの記事があるのがうれしい。 raihennさん／東京都

久しぶりにWiresharkを触ってみて、かなりの機能が追加され、かつ便利になっていることに驚きました。 山下さん／東京都

低レベルな部分からわかりやすい説明がされている。 上藤さん／広島県

プログラマサイドのTCP/IPの本質が理解できた。 高尾さん／群馬県

より深く学びたい読者向けに、参考文献リストを特集末尾に掲載しても良かったかもしれません。 匿名希望さん／大阪府

アプリ開発者といえども、イマドキTCP/IPの知識なくして最適な開発はできないです。 ほまれさん／千葉県

TCP/IPはネットワークの基本中の基本なので、ネットワークに携わるエンジニアには必須の特集でした。C言語から利用するのは初心者には若干ハードルが高いと思いますが、基礎をちゃんと知るという意味ではすばらしい特集だったと思います。 bsdhackさん／神奈川県

インフラエンジニア向けよりは、プログラマ向けに重きを置いたTCP/IP特集。普段から利用しているインターネットのしくみを、それこそコード単位で紐解くことができました。

第2特集 手を動かして学ぼう正規表現入門

プログラミングやエディタの作業を効率化する「正規表現」を基本から解説。Webツール「Rubular」で、書いた正規表現がきちんと動くか試しながら知識を身に着けていく、実践的な特集でした。

正規表現の理解が深まり良いと思いました。多くの人に使ってもらいたいですね。 masaki_hashimo2さん／沖縄県

正規表現の説明がかなりわかりやすいため、これから正規表現を学習する方

7月号のプレゼント当選者は、次の皆さまです

①HHKB Professional BT
向山裕介様（東京都）

②USBアナログメーター
井上裕貴様（東京都）

③Vivaldi Tシャツ&ステッカー
小林福嗣様（東京都）

④『Unix考古学』
林正紀様（埼玉県）、福田和真様（神奈川県）

⑤『プリンシプル オブ プログラミング』
西上博士様（兵庫県）、諸星大樹様（愛知県）

⑥『Amazon Web Services クラウドネイティブ・アプリケーション開発技法』
西谷幸治様（大阪府）、長浜均様（神奈川県）

⑦『基礎からのWebアプリケーション開発入門』
永島薫様（福岡県）、いくす様（埼玉県）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

にはもってこいだと思いました。個人的には正規表現の確認サイトを知らなかったので、勉強になりました。伊藤さんの記事は丁寧で本当にすばらしい。これからの記事も楽しみにしています！
ノリオさん／大阪府

正規表現はなかなかとっつきにくいので勉強になります。　ともさん／兵庫県

正規表現って敷居が高いのですが、これをうまく引き下げて説明されているように感じました。　鈴木さん／熊本県

正規表現はミニ言語という視点はおもしろい、と思いました。
泥臭い環境しか知らない。さん／奈良県

どうしてもわかりにくい正規表現をWebツールを使って解説してくれるのはすごく助かると思います。もっと早く読みたかった。
romeosheartさん／長崎県

サーバをいじる際などに正規表現が使えると便利と思いつつ、なかなか憶えられなかったが、例がわかりやすくとっつきやすくて良かったです。
kusaさん／東京都

「HTMLをCSVに置換」の項、仕事でも役に立ちそうです。
ＹＹさん／神奈川県

基礎部分の理解があるだけで、文字列処理の効率が全然上がると思う。
阿部さん／岩手県

今さらながら、正規表現を利用したものを組む際はググったりしていたので、再度勉強になった。
massakiiiiさん／福岡県

今まで都度都度の独学だったが、きちんと勉強しようという気になった。
kybさん／東京都

使っているプログラミングがRubyだったため、正規表現をよく利用するのでおもしろかったです。 井上さん／埼玉県

一度覚えてしまえば普段の作業が一気にカイゼンされる正規表現。必要だとは思っていてもなかなか勉強する機会がなく、この機に勉強できて良かったという声が多かったです。

表紙について

弊誌の表紙の変遷ですが、2013年度「鳥」、2014年度「犬」、2015年度「野生動物」となっています。そして今年2016年度は、ご存じのとおり「猫」です。

娘が猫好き。癒やされる。
西原さん／兵庫県

子猫に癒やされた。
白川さん／東京都

猫がかわいいです。
風のピエロさん／長野県

やっぱ、にゃんこでしょ！
南雲さん／埼玉県

一目見て貴誌と分かるデザインが安心感を与えてくれます。
サバ鼻炎さん／東京都

猫は、エンジニアの方にはとくに人気の高い印象の動物ですね。内容ももちろんですが、表紙にもご注目ください！

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
October 2016

2016年10月号
定価（本体1,220円+税）
192ページ

9月17日発売

［第1特集］意外と説明できない?
Webサーバとは何か? なぜ動くのか?
CGI、サーブレットからNode.js、Railsまで一挙解説

［第2特集］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!

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2016年7月号
●P.172　連載「Linuxカーネル観光ガイド」リスト2中の下から10行目
［誤］ if (TICKET_SLOWPATH_FLAG &&
__ticket_t head;
［正］ if (TICKET_SLOWPATH_FLAG &&
static_key_false(&paravirt_ticketlocks_enabled)) {　←この行抜け
__ticket_t head;
●P.174　左段10行目
［誤］ただし、SPIN_THRESHOLD（=2）
［正］ただし、TICKET_LOCK_INC（=2）
■2016年8月号
●P.102　連載「アプリエンジニアのための［インフラ］入門 第2回」右段上から21行目
［誤］192.168.0.0/216
［正］192.168.0.0/16

SD Staff Room

●8月には『小悪魔女子大生のサーバエンジニア日記』のaicoさんが挿絵を描いた、『独習Python入門』（著 湯本 堅隆=ござ先輩）が発売されます。10月には一昨年の特集『ポートとソケットがわかればTCP/IPがわかる』が同じくaicoさんのイラストで書籍発売予定です。ぜひ買ってください。（本）

●文鎮化したNexus7を冷蔵庫に入れておいたら治ったと友人に聞き、ロッカーに眠っていた、買ってすぐに画面が乱れて電源の入らなくなったWinタブを思い出した。まずは充電して……と電源をいれたら無事起動! ギリで無料10化も間に合った。しかし、自前Nexus7文鎮は復活しませんでした。（幕）

●仕事の帰り道、目の前で信号が赤に変わる。何気なく脇にある公園の石垣に目をやると、そこにへばりつく黄色いトカゲと目が合った。微動だにしない彼の気持ちは知らず、“久しぶりだな”とのんきに独り言。それから信号が変わるまで二人でだるまさんごっこをした。明日はもう少し早く帰ろう。（キ）

●NHKの連ドラ「とと姉ちゃん」にハマっています。平日は見られないので、土曜の放送だけを見ています。ほかの月～金の内容は頭の中で勝手に想像しています。が、伊藤淳史なる俳優さんが演じている男性が何者なのかがいまだにわかりません。とと姉ちゃんと親しくしていて重要人物ぽいのに……。（よし）

●地元の友人連中に混ざりたくて、十何年ぶりに某TCGを始めようかと思案中。自分の小学生時代からは状況が変わっているようで、ガチ勢の友人たちと対等に対戦するためのデッキを作るには、なんと1万円超は掛けなければならない雰囲気。一年に一度会うか会わないかの友人なので、悩みどころですね。（な）

●来年用の年賀状素材集本に掲載予定のちびぬいができあがりました。これを撮影してもらってポストカードにします。撮影には小物も必須であれこれ手作りするのですが、毎年1つ、2つ冬っぽい素材が足りなくなり慌てて購入するはめに…夏なのでお店ではなかなか欲しいものがすぐ手に入らないのがつらいところです。（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9月号

発行日
2016年9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。