special feature 1 ログは宝の山

special feature 2 プロセス管理からカーネルを学ぶ

Software Design

2014 8

2014年8月18日発行
毎月1回18日発行
通巻352号
（発刊286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

[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

ログを読む技術

コンピュータに
残ったログは宝の山!

special feature 1

身近なシステムログからウェブやデータベース、そしてビッグデータの基礎まで

故障の原因追求から
ユーザ動向まで
まるわかり!

special feature 2

forkを通して考える・試す・コードを読む
Linuxカーネルのしくみを探る

特別企画

ITビジネスの足下を揺るがす大きなバグ
OpenSSLの脆弱性“Heartbleed”の教訓（後編）

新連載

クラウドシステムインテグレータの心意気!
サーバーワークスの瑞雲吉兆仕事術

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。

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。

第68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。

Swift

iOS/OS Xのための新言語「Swift」

Swiftは、iOSおよびOS X向けのアプリケーション開発のためにAppleによって作られた新しいプログラミング言語です。同社が主催する開発者向けイベント「WorldWide Developers Conference 2014」において発表されると同時に、開発者向けにベータ版の提供が開始されました。

Appleでは、Swiftの特徴として次のような点を挙げています。

モダン

Swiftは、クロージャやジェネリクス、タイプインターフェース、タプル、Optional型といった近代的なプログラミング言語の機能を積極的に採用しています。これによって、複雑さを排除したシンプルなコードで、素早くアプリを開発することができます。

安全な設計

Swiftは静的型付け言語であり、コンパイル時の強力な型チェックによって型安全性が保証されます。さらに、変数の初期化の強制や、配列や整数のオーバーフロー検出、参照カウント方式による自動メモリ管理、記述ミスによるバグ発生を抑制する文法設計など、安全性を高めるさまざまな工夫が施されています。

インタラクティブ

対話形式の実行環境が用意されており、コンパイラ言語でありながらインタプリタ言語のようなスタイルで手軽にコードの実行やデバッグができます。この機能は「Interactive Playground」と呼ばれ、Xcode 6より新たに利用できるようになります。

高速

Swiftのコンパイルと実行にはLLVM技術が採用されており、ハードウェアごとに最適なネイティブコードが生成されるため高速な実行が可能です。Appleの発表によれば、そのパフォーマンスはObjective-Cを上回るとのことです。

Swiftが与えるインパクト

Swiftの登場は、iOSやOS X向けアプリケーションの開発者に対して極めて大きなインパクトを与えるものです。Objective-Cの歴史は古く、それだけに蓄積されたノウハウも膨大な量にのぼります。しかしその一方で、古い設計の言語を拡張し続けてきた副作用として言語仕様が煩雑化し、最適化が難しくなってきているという問題がありました。とくにiPhone/iPadが登場してからは新規の開発者が増え、モダンな開発言語を望む声も少なくありませんでした。

Appleでは従来よりObjective-Cの拡張や開発環境による多言語サポートなどによって、アプリ開発の生産性を向上させるための取り組みを続けてきましたが、それらはあくまでもObjective-Cを中核に据えたものでした。一方Swiftの場合は、最初からObjective-Cの置き換えを狙って設計されたという点で、これまでのアプローチとは異なる位置づけを持っていると言えます。

現在のところ、Swiftの発表は開発者にはおおむね好意的に受け入れられているという印象を受けます。歴史的な事情に左右されないシンプルな言語仕様は、開発の効率を向上させるだけでなく、性能の向上や安定性の強化、より良いUIの実現などにも影響します。新言語から開発者が受ける恩恵は大きく、そのことが歓迎ムードにつながっているものと考えられます。

通常、新しい言語への移行には既存の資産やノウハウが無駄になるかもしれないという懸念が付きまとうものですが、Swiftの場合は既存の実行環境との互換性が高く、Objective-Cと共存させられるという強みがあります。従来のフレームワークにも適合しており、ランタイムもObjective-Cと同じであることから、移行のハードルはきわめて低いと言えるでしょう。

SwiftはiOS 8およびOS X Yosemiteに合わせて正式リリースされる予定で、現在はXcode 6のベータ版によって試すことができます。また、iTunes StoreではApple公式の言語ガイド「The Swift Programming Language」が公開されています。SD

Swift Programming Language
https://developer.apple.com/swift/

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。

Contents

Software Design Aug. 2014

1

［第1特集］

システムログから
WebやDB、
ビッグデータの
基礎まで

ログを読む技術

手がかりを
見いだす
眼力をつくる

021

第1章 ログの基本をおさえておこう 近藤 成 022

第2章 Webサーバのログを見てみよう 近藤 成 037

第3章 MySQLのロギングを見てみよう 近藤 成 052

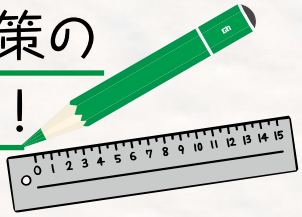
第4章 ログを管理・運用しよう 近藤 成 063
ログローテーションとログウォッチ

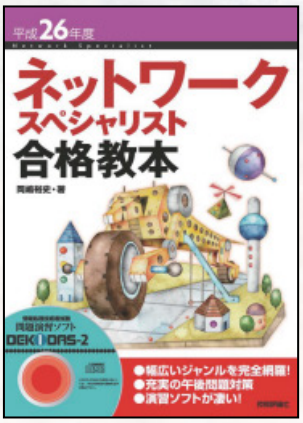
第5章 MSP直伝・プロがやっている 高村 成道 073
ログ監視

第6章 フロントエンドエンジニアもFluentd+MongoDBで実践! 羽田 健太郎 083
小さく始めるログ活用のすすめ

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／624ページ
定価（本体2980円+税）
ISBN978-4-7741-6354-3

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）
ISBN978-4-7741-6355-0

岡嶋裕史　著
A5判／656ページ
定価（本体2880円+税）
ISBN978-4-7741-6099-3

エディフィストラーニング株式会社　著
B5判／392ページ
定価（本体2980円+税）
ISBN978-4-7741-6570-7

濱本常義ほか　著
A5判／592ページ
定価（本体3200円+税）
ISBN978-4-7741-6178-5

左門至峰、平田賀一　著
A5判／328ページ
定価（本体2280円+税）
ISBN978-4-7741-6389-5

大滝みや子、岡嶋裕史　著
A5判／736ページ
定価（本体2980円+税）
ISBN978-4-7741-6111-2

加藤昭、芦屋広太ほか　著
B5判／464ページ
定価（本体1680円+税）
ISBN978-4-7741-6556-1

金子則彦　著
A5判／688ページ
定価（本体3300円+税）
ISBN978-4-7741-6177-8

内田保男ほか　著
A5判／512ページ
定価（本体3200円+税）
ISBN978-4-7741-6101-3

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

Contents

Software Design Aug. 2014

2

［第2特集］

forkを通して考える・試す・コードを読む
Linuxカーネルのしくみを探る 093

Part1 プロセスに見るLinuxカーネルの役割 中井 悦司 094

Part2 「fork」を通してカーネル内部を理解する 岩尾 はるか 104

Part3 ソースコードで見るカーネルの全体像 中井 悦司 115

［一般記事］

大規模Windows環境におけるデプロイ
AWS+Windows環境における大規模ソーシャルゲーム開発／運用の実際［2］ 田中 孝佳 001

ITビジネスの足下を揺るがす大きなバグ
OpenSSLの脆弱性“Heartbleed”の教訓【後編】 すずきひろのぶ 120

ネットワークを眼で読む
使ってみよう! tcpdump 後藤 大地 128

［Catch up new technology］

クラウド時代だからこそベアメタルをオススメする理由［1］ 編集部 198
本当に求められているクラウドサービスとは

［巻頭Editorial PR］

Hosting Department［100］ H-1

［アラカルト］

ITエンジニア必須の最新用語解説［68］Swift 杉山 貴章 ED-1

読者プレゼントのお知らせ 020

SD BOOK FORUM 135

バックナンバーのお知らせ 151

SD NEWS & PRODUCTS 202

Letters From Readers 210

［広告索引］

BOOKS INFOMATION

ISBN978-4-7741-6522-6
四六判・216ページ
定価（本体1,680円+税）

システムインテグレーション崩壊

これからSIerはどう生き残ればいいか？

●斎藤昌義　著

国内の需要は先行き不透明。
案件の規模は縮小の一途。
単価が下落するばかり。
クラウドの登場で迫られるビジネスモデルの変革。

工数で見積もりする一方で、納期と完成の責任を負わされるシステムインテグレーションの限界がかつてないほど叫ばれる今、システムインテグレーターはこれからどのように変わっていくべきか？

日本IBMの営業を経て、数多くの企業にコンサルティングを行う著者が、豊富な図解とともに現状とあるべき姿を解説する。

ISBN978-4-7741-6523-3
四六判／240ページ
定価（本体1,680円+税）

プログラムは技術だけでは動かない

プログラミングで食べていくために知っておくべきこと

●小俣光之　著

開発言語にくわしい。
さまざまなアルゴリズムを理解している。
開発環境を使いこなせる。
ミドルウェアなどの情報を知っている。
OSやネットワークなどの知識がある。

そんな「技術力がある人」なのに、なぜ仕事では通用しないのか？

数々の現場を経験し、いまも現役プログラマー社長として活躍する著者が、「技術を生かして食べていく」ためにはあたりまえのようでいて意外と見すごされている「技術以外」の話を教えます。

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

Contents

Software Design Aug. 2014

3

Column

digital gadget[188]	展示のためのデジタル技術	安藤 幸央	005
結城浩の再発見の発想法[15]	Tradeoff	結城 浩	008
enchant ～創造力を刺激する魔法～[16]	独裁者の帰還	清水 亮	010
軽酔対談 かまぶの部屋 [新連載]	ゲスト:奥谷泉さん	鎌田 広子	014
秋葉原発! はんだづけカフェなう[46]	BLEで遊んでみよう	坪井 義浩	016
Hack For Japan～ エンジニアだからこそできる復興への一歩[32]	島ソン! 電波も届かない離島でのハッカソン	小泉 勝志郎	192
温故知新 ITむかしばなし[36]	カセットテープとデータレコーダ	佐野 裕	196
SDでSF[8]	『太陽の簒奪者』	小飼 弾	208
ひみつのLinux通信[8]	ejectコマンド	くつなりょうすけ	209

Development

サーバーワークスの瑞雲吉兆仕事術 [新連載]	クラウド前夜「スマートフォンの登場」	大石 良	136
るびきち流 Emacs超入門[4]	SKK+AZIKで快適・効率的な日本語入力を!	るびきち	140
シェルスクリプトではじめるAWS入門[5]	AWS利用環境の構築(後編)	波田野 裕一	146
ハイパーバイザの作り方[21]	bhyveにおける仮想シリアルポートの実装(その1)	浅田 拓也	152
Androidエンジニアからの招待状[49]	Androidが生まれ変わる、活躍の場が広がる	嶋 是一	156

OS/Network

RHELを極める・使いこなすヒント .SPECS[4]	Red Hat Enterprise Linux 7に触れてみよう	藤田 稜	162
Be familiar with FreeBSD ～チャーリー・ルートからの手紙[10]	コンパイラ~GCCからLLVM Clangへ	後藤 大地	166
Debian Hot Topics[17]	Debian GNU/Hurdの状況／Squeeze LTSの使い方	やまねひでき	170
レッドハット恵比寿通信[23]	OSSに求められる合理性	平 初	174
Ubuntu Monthly Report[52]	SoftEther VPNをUbuntuだけで使用する	あわしろいくや	176
Linuxカーネル観光ガイド[29]	Linux 3.15の変更点~キャッシュ管理の改善とPM QoS	青田 直大	182
Monthly News from jus[34]	はじめてUNIXで仕事をする人に教えたいこと	法林 浩之	190

Logo Design ロゴデザイン	デザイン集合ゼブラ+坂井 哲也
Cover Design 表紙デザイン	Re:D
Cover Photo 表紙写真	(c) BLOOM image /amanaimages
Illustration イラスト	フクモトミホ、高野 涼香
Page Design 本文デザイン	岩井 栄子、 ごぼうデザイン事務所、 近藤 しのぶ、 SeaGrape、 安達 恵美子 [トップスタジオデザイン室] 轟木 亜紀子、阿保 裕美、佐藤 みどり [BUCH+] 伊勢 歩、横山 慎昌、 森井 一三、 Re:D、 [マップス] 石田 昌治

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

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

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

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

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)

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

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、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

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

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

この個所は、雑誌発行時には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。

DIGITAL GADGET

Volume 188

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

展示のためのデジタル技術

さまざまな展示

首都圏にお住まいの方は、東京ビッグサイトや幕張メッセなどの巨大な展示会場や、ホテルなどで開催されるプライベート展示会など、数多くの展示会に参加する機会があります。企業の展示会の中には、全国各地を行脚して開催されるものもあります。また、各地に企業ショールームがありますし、大型家電ショップも一種のショールーム的役目を果たしています。海外の展示会でさえ、ネットニュースやオンライン動画共有サイトの発達のおかげでさまざまな最新情報を得ることができます。

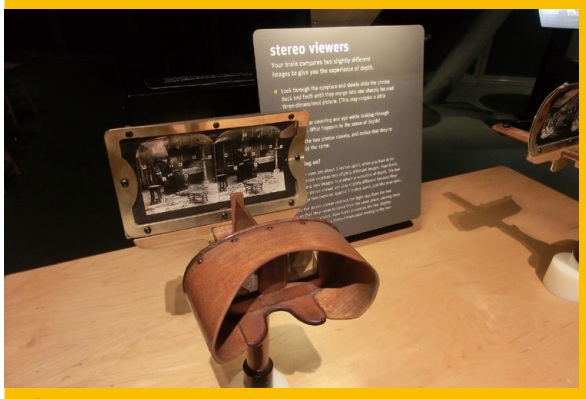
一方、そのような展示会に出展する企業側の方の中には、自社のプロダクトやサービスを展示する役目を仰せつかった人もいるでしょう。投資家たちや株主の前で、限られた時間でデモンストレーションする場面もあるかもしれません。各種展示会の場合、もちろんブースの造作や、華やかなコンパニオン、製品やサービスそのものの魅力も重要ですが、「展示」そのものの工夫も、集客効果やその後のビジネスに影響してきます。

雑貨店のようにわざと混沌とした展示をする場合もありますが、たいていの場合、良かったと思わせる展示は、博物館や美術館のように観る人のことをよく考えた展示であることがわかります。たとえば、観る順番を迷わせないことに着目すると、「説明書きが適切な高さ・位置にあること」「文字が読みやすいこと」「展示ブースが混雑しているときにも、何らかの情報が得られること」などが挙げられます。気配りのあるブースには入りやすい・観やすい雰囲気があり、ブースを離れるときには「必要な情報が得られた」「興味を持った」といった満足感が得られます。

イタリアの著名デザイナーであり、数々の展示会の会場構成を務めたアキッレ・カスティリオーニは、展示における参加者は次の4種類に分かれると述べています。

1. 大雑把な説明で満足する人たち
2. きちんと理解したい人たち
3. 細かいことにもうるさい人たち
4. 子供、若い人たち

これらの分類の中でどの人たちをターゲットにするのか。または、すべての人たちを対象とし、それぞれどのように違った対応をしなければいけないのかを考えておかなければいけません。



サンフランシスコエクスプロラトリアム科学館の立体視の原理を知るための展示


コンピュータヒストリーミュージアムの展示、1969年のキッチンコンピュータ

※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おります。

展示を成功させるために

一般的に常設の展示や長期間の展示でないかぎり、実際に展示するまでの設営時間は短く、100%満足のいく展示にすることはなかなか困難な事柄です。けれども事前によく考えられた準備をしておくことによって、より良い展示を素早く構築するための手はずを整えることができます。注意点は次のような事柄です。

- 目線の高さや目線の動き（キャプション［解説文］の示し方から作品展示そのもの）を考慮する
- 自分の影や反射が映り込まないように。見ている人、そのほかの存在をできるだけ消すように（ブースを区切ったり、入場制限をして順々に見てもらうなども1つの方法です）
- 多くの人が見に来る、あるいは一度に触れる人は1人だけというようなサービスの場合、それ以外の多くの人が見られるディスプレイ映像などを用意しておく
- 人の動き、導線に配慮する。迷わない人の流れを作り出す
- 展示会場では、ネットがない環境やうまくつながらないことを前提に準備しておく。有線LANも検討
- 短時間ですべてを知ることは難しいので、別途Webサイトやパンフレットがあるといい
- すべて準備万端と思っても不慮のミスもある。リハーサルと確認を怠らない
- 液晶画面やディスプレイ画面は省電力モードではなく、見やすく明るくしておく
- スマートフォンの画面やディスプレイ画面などは、メガネ拭きできれいにしておくなど最良の状態で

照明、音、音楽や騒音など、展示の現場でしかわからない条件もいろいろあります。展示空間は日常の仕事場とは異質な空間なので、すべてがうまくいくことを願いつつも、いろいろな事柄がうまくいかない前提で対処していけばよいでしょう。いつも使えているケーブルですら、たまたま何かの不具合で使えないといったことがあるかもしれないのです。そしてそれに気づくのはとても難しいことでもあります。いつもは抜けるはずのない電源ケーブルに足を引っ掛けて抜いてしまったり、転んで飲み物をこぼしてしまったり……マーフィーの法則ではないですが、何か困ったことが起こる可能性が少しでもあれば、その困った事象は運悪く本番で起こってしまうかもしれないのです。

さらに展示の場合、1人だけに説明して、1人に観てもらえばいいわけではありません。解説や説明をする人が必ず一人一人に対応できるとも限りません。実際に触ってみる試用体験や視聴といった1人でしか体験できない展示であれば、順番を待つ人が待つだけになってしまわないような配慮も必要です。ほかにも、一度の展示で満足していただけるものと、何度も足を運んでほしいもの、口コミで多くの人の話題になってほしいものなど、見せ方だけでなく、その周辺の体験もうまく配慮すると広がりが生まれます。

展示のこれから

LED照明が出始めのころは、繊細な色、鮮やかな発色が難しかったようですが、現在では良質のLEDライトの登場で、展示会場や展示物のライティング設計がいろいろと拡張でき、自由度が広がってきたそうです。また、反射しないガラスや透明度の高いガラスの登場で、作品や展示物を保護しつつも身近に感じながら観ることができるようになっています。技術の進歩で展示方法も進歩してきました。

「良かった」と思える展示は、違和感やちょっとした引っかかりもなく、展示物そのものの良さが素直に伝わる環境が作られているのだと考えています。たとえば、文字が横書きであれば、

鶏のふ化の様子を展示したもの。暗いところでも文字が読みやすいようにバックライトで照らされており、展示そのものと説明文が近くに配置されていて相互を順に見やすい

巨大なマッチ棒細工の展示。大きさが比較できる物が置いてあるのと、展示そのものが浮き上がって見えるよう、背景の壁がここだけ青色になっている

爆発的に普及したPDA「Palm Pilot」のスケルトンモデルと、開発初期段階の紙で作られたプロトタイプ。アート作品ではなく、工業製品の場合、その製品が作られる過程を展示することも、その製品を知ってもらい、親しく感じてもらう良い方法である

たいていの言語の場合、左から右に描かれます。つまり視線の移動は左右に流れます。そう考えると、キャプション（説明書き）から、商品やプロダクトといった作品展示への視線の流れがスムーズだと違和感がないですし、逆にそこに不必要な視線の動きが生じると面倒さを感じてしまうのだと思います。

また数多くの展示を観て、良い要素、または良くない要素を細かく分解していくのも良い方法です。単に「良かった」で終わらずに、良いと思った展示のどこが良かったのか、何の要素が「良かった」につながっているのかを、客観的に評価するのです。

さらに、同業種ばかりではなく、他業種の展示、美術館や博物館、スーパーや雑貨店、書店の展示も注意深く観察することで、さまざまなノウハウを学び取ることもできます。

自身の展示も客観的な目線で評価し、改善要素を見つけていくのも重要です。それこそ数日単位でなく、数時間単位でも見直し、改善していく要素が何かしらあるはずです。

魅力のある展示のためには、人の振る舞いや印象を知ることがとても大切なのです。SD

ムーアの法則の説明展示。半導体の集積密度は18〜24ヵ月で倍増するという法則に基づく指数的な増加をネオンランプで示すとともに、ディスプレイによる説明映像、それに手元に置かれたシリコンウェハースで実感のともなう統合展示になっている

GADGET 1

Space Player

http://www2.panasonic.biz/es/lighting/led/special/spaceplayer/

照明型プロジェクタ

Space Playerは天井に設置するスポットライト型の形状をした映像プロジェクタ装置です。従来のスポットライトは光を当てるだけでしたが、Space Playerは色を持った映像を対象物に照射することができます。映像の入力はSDカード、HDMI、無線LAN（Miracast）経由など。本体色は設置場所に応じてホワイトとブラックの2色から選べます。光源はレーザーダイオードで1000ルーメン、1280×800の解像度で動画／静止画の表示が可能です。天井から吊り下げて設置する大型のプロジェクタとは違い、照明器具と同様に配線ダクト取り付けで設置できるのも利点の1つです。

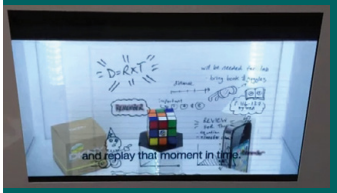
GADGET 2

Transparent Showcase

http://crystal-display.com/digital-signage/transparent-displays-2/clearvue/

透明ディスプレイ

Transparent Showcaseは、ショーケースの箱の前面に透明ディスプレイが設置されたボックスです。透明ディスプレイは、通常の液晶ディスプレイの背面にあるライトが排除されたLEDパネルだけのもので、コンピュータから真っ黒い画面を映す指示を出せば、LEDには何も表示されず向こう側が透けて見えるしくみです。フルハイビジョン解像度で70インチ相当の最大サイズのものから、最小で10インチ相当のものまでいくつかのサイズが用意されています。透明ディスプレイに映し出された映像は、それそのものは発光しないため、ショーケース内の照明用白色LEDが必要です。



GADGET 3

TABLETY

http://workstudio.co.jp/product/tablet/tablety/

タブレットスタンド

TABLETYは盗難防止機能を備えた、フロアスタンドタイプのタブレット端末用展示機器です。スタンドタイプのほかにも壁掛けタイプや、机の上に置くタイプなどがあります。iPadなどのホームボタンが不用意に押されないようカバーをしたり、充電ケーブルが抜けないような配慮が施されています。別売りのセキュリティアラームを使用することもでき、防犯性にも優れています（盗難防止を確約するものではありません）。

GADGET 4

自然光LED薄型ライン照明

http://www.ccs-inc.co.jp/museum/

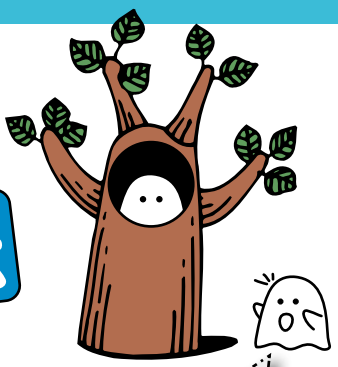
色彩を忠実に再現するLEDライト

自然光LED薄型ライン照明は、幅19mm、高さ12mmと小型で、さまざまな狭所に設置可能なLED照明です。長さは4種類から選択またはカスタマイズも可能。このLED照明に照らされた対象は、自然光のもとで見た色合いにとても近く、微妙な色合いを重視する化粧品や、革製品などの展示にも使われているそうです。また、紫外線や赤外線をほとんど発しないため、対象物の色あせを心配することなく、対象物のすぐ近くに設置することもできるようです。

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

15

結城浩の再発見の発想法



Tradeoff

Tradeoff——トレードオフ

トレードオフとは

トレードオフ(tradeoff)とは、ある量を都合のいい方向に動かすと、別の量が不都合な方向に動いてしまう関係のことです。トレードオフとは「取引」や「交換」という意味で、この用語は「何かを自分が買うためには、その対価として別の何かを支払う必要がある」ということを表現しています。

時間と空間のトレードオフ

プログラミングで起きる典型的なトレードオフは**時間と空間のトレードオフ**です。たとえば、プログラムの高速化を行うための技法として、バッファリングやキャッシュがよく使われます。これは、2回目以降のデータ取得を高速にするため、取得したデータをメモリ上に保存しておく技法です。ここでは「データ取得を高速にする」という「時間」を買うために、「データを保存しておくメモリ」という「空間」を対価として支払っていることになります(**図1**)。

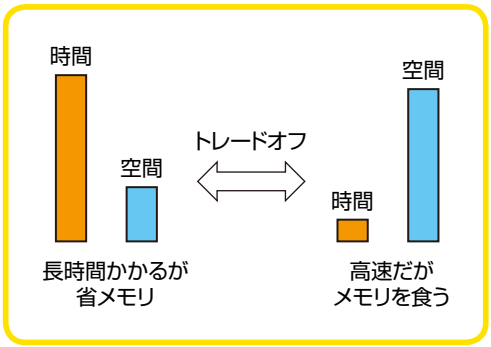
時間と空間のトレードオフでは、いつも空間を対価として支払うわけではありません。たとえば、三角関数や対数関数のような数学的関数をプログラムとして実装する場合、「前もって数表を関数内部に持つ方法(空間を使う)」と「毎回計算によって求める方法(時間を使う)」の2つがあります。メモリに余裕がない環境では、計算を行って実装したほうが都合がいい場合もあるでしょう。これは「メモリ」という「空間」を買うために、「毎回計算する」という「時間」を対価として支払っていることになります。

時間と空間のトレードオフは、時間か空間かの二者択一というわけではありません。時間と空間のどちらをどれだけ買うか(支払うか)には、幅広い選択の余地があります。そしてシステムを設計する人は、環境に合わせて最も良いポイントを選択することになります。たとえば数学的関数を実装する場合、よく使われる値については数表を用意しておき、あまり使われない値は計算を行うという実装も可能でしょう。

コンパイラのトレードオフ

時間と空間のトレードオフ以外にも、プログラミングではあちこちにトレードオフが発生します。

▼図1　時間と空間のトレードオフ



たとえばコンパイラを考えてみましょう。プログラマがコードを書き、それを直接インタプリタとして実行できるなら、コンパイルと最適化のための時間を節約できますが、実行速度は遅くなります。逆に、コンパイルと最適化のために時間を使うなら、実行速度は速くなります。ここではコンパイル時と実行時の2つの時間の間にトレードオフが発生しています。

トレードオフで最適なポイントがどこにあるかは、時代と環境に依存します。コンパイルと最適化にどれだけ時間がかかるか、またそれによってどれだけ実行速度が向上するかの兼ね合いということになるでしょう。インタプリタが内部で中間コードにコンパイルを行うことも、コンパイラがネイティブコードではなく仮想コードにコンパイルすることも、さらに実行時にネイティブコードに変換することも、すべて「時間を何のためにいつ使うか」ということを念頭に置いたトレードオフの結果といえます。

選択は常にトレードオフ

実際のところ「あれかこれか」という選択が生じるところでは、常にトレードオフが存在しているといえます。

これはトレードオフの意味から明らかです。時間であれ、空間であれ、どんな量であれ、いくらでも都合がいいように増加させられるなら、何も悩む必要はなく、選択する必要もないからです。

選択が生じるのは、複数の選択肢のあいだに何らかの相反する関係があるからです。完全にいい選択肢が1つあるなら、それはもう選択肢とはいえません。

トレードオフの原因

ところで、どうして選択が生じるところで常にトレードオフが生まれるのでしょうか。

それは、多くの場合**リソースに限界がある**からです。もしも無限にリソースがあるなら、ほとんどのトレードオフは消えてしまいます。無限に時間を使える場合、無限にメモリを使える場合、無限にお金を使える場合などを考えればわかるでしょう。しかし現実の問題では、リソースを無限に使えることはありません。ですからどうしてもリソースの「やりくり」が発生します。リソースのやりくりをどうするか、これはまさにトレードオフに直面している状況になります。

そう考えてくると、トレードオフに直面したときに最初に行うべきことは、**リソース状況を把握する**ことになりますね。たとえば、時間と空間のトレードオフなら、自分が使える時間と空間がそれぞれどれだけなのかを把握するということです。それは自分が許容できる支払いと、それを対価にして買えるものとを把握することにほかなりません。

もしも、リソース状況を十分に把握することができ、さらに自分が許容できる支払いを把握することができたなら、トレードオフに直面しても、良い判断ができることになるでしょう。

日常生活とトレードオフ

技術的な問題に限らず、私たちの日常生活でもトレードオフは日々発生します。

明日までに2つの作業をしなければならないが、どちらにどれだけの時間を使うべきか。ここでは「明日までの時間」という限られたリソースをやりくりする状況を考えているわけです。

作り上げたプロダクトやサービスの提供価格の問題も、トレードオフと見ることができます。単価が低ければたくさんの顧客に売れますが、多数売らなければ売上が上がりません。一方、単価が高ければ少数の顧客でも売上が上がりますが、そもそもその顧客がなかなか見つからないでしょう。

◆ ◆ ◆

あなたの周りを見回して、トレードオフの関係にあるものを探してみてください。きっと、あらゆるところに見つかるはずです。そこでやりくりされている有限のリソースは何でしょうか。ぜひ考えてみてください。SD

enchant ～創造力を刺激する魔法～

(株)ユビキタスエンターテインメント　清水 亮　SHIMIZU Ryo
URL http://www.uei.co.jp

第16回 独裁者の帰還

君たちのものだよ

アラン・ケイ博士にふたたびお会いできる機会を得られたのは、前回お会いしてからちょうど1年後のCES明けでした。場所は前回と同じカリフォルニア州ロサンゼルス郊外のビューポイントリサーチ社。アメリカ政府の施設で、入室にはパスポートの確認が必要な厳重さです。

再会に感謝の意を示し、現在のenchantMOONを説明し、それから何か助言をください、と言うと、ケイ博士はかぶりをふりました。

「これはもう、君たちのアイデアだよ。私のものではない。君たちのものだ。助言できることはもう何もないよ」

意外な言葉でした。

「君たちのアイデアの、いくつかの点は凄くいい。いくつかの点は、私はそれほど好きではない。なかでも1つとても気に入っているのは、ピクシーダストだ。ピクシーダスト。こんな演出を、私たちは思いつかなかった。魔法だね。まるで」

魔法……魔法か。もはや僕たちエンジニアの想像力はSFではなく魔法物語にインスパイアされるべきなのかもしれません。草木が喋り、ホウキが空を飛ぶ、そんな魔法の世界です。

そんな夢の最中、僕は一方で暗澹たる気分を隠しきれませんでした。その改良は決して簡単なものではなく、実装は理想像とはほど遠いものだったからです。アラン・ケイ氏にまで会っておいて、自分の作っているものの惨めさと言ったら、悲惨そのものでした。

どうにかしなければならない。僕は焦る一方でした。この迷宮から、どうすれば抜け出せるのかと、もがき苦しんでいました。

五里霧中

enchantMOONのメインプログラマで、学生時代から僕と長年の師弟関係にあった近藤誠が去り、それを契機として僕と現場との距離がどんどん離れていきました。僕は開発者たちの居る部屋には一切立ち入ることをやめました。そう求められたからでもあり、そうすべきだと思ったからでもあります。

それは成蹊大学の講義の後、開発者たちに厳しい現実――つまりいま我々が誇りを持って開発し、販売している製品は、単なる欠陥品であるということ――を見せつけた後も大きな変化はありませんでした。もちろんそれもそのはずです。彼らに無謀な挑戦をさせ、完成度の低いソフトウェアのまま製品を発売させることになったのは、トップマネジメントである僕自身の責任だったからです。彼ら自身、納得のいかないまま、しかしこの先どうすればいいのかわからないという五里霧中、暗中模索のまま、闇

雲に時間ばかりが過ぎて行くのでした。

しかし事態の悪化はそれだけに止まりませんでした。ある日のことです。

「最新バージョンを持ってきました」

秋葉原リサーチセンター部長の増田哲郎が僕に1台の端末を差し出しました。僕はそれを少し触り、書き心地を試してみて、それからページをめくってまた書いてみました。

「どうでしょう？　だいぶ高速化されてると思いますが」

「そうか？　どのくらいだと思う？」

「さあ……感覚では10〜20%くらい速くなった気がしますが」

「じゃあこれを見てみろ」

僕は普段持ち歩いているenchantMOONをカバンから取り出し、増田に差し出しました。

「えっ……これ速いですね。清水さんが書いたんですか？」

「いいや。これは1ヵ月前のバージョンだ」

「え……」

「君たちは感覚が麻痺しているんだ。毎日毎日いろんな最適化を試し、その都度、速くなった、遅くなった、というのを感覚でつかもうとしているからだ。昨日、秘書にビデオを撮影して調べてもらったところ、開発中の現行バージョンは前バージョンより50%ほど遅くなっている」

「そ、そんな……」

「間違ってるんだよ。“速さの指標”が。確かに現行バージョンは、ページを次々とめくるということに関しては前バージョンより高速化されている。しかしそれだけだ。実用的にページをめくった直後にペンを走らせたときの速度は却って低下している。こんなことを繰り返していてはだめだ。根本的にアーキテクチャがダメなんだよ。実態に即してない。これは誰もが認めなければならない。それと、ハイスピードカメラによる動作速度の測定、これを毎日のルーチンとしてナイトリービルドごとにやらせなさい」

現場に干渉しない、という約束のもとでできる指示としては、僕にはこれが精一杯でした。

ナイトリービルド

嫌われることを怖れるのをやめよう

しかし現場は幾度も手戻りを繰り返していました。ある部分が速くなったと思えば、また別の部分が遅くなります。資金はどんどん目減りし、ソフトウェアの完成度はいつまで経っても上がらない。そんなことが繰り返されました。

僕は10年来の旧友で最も信頼しているプログラマの布留川英一を密かに呼びだしました。

「いまのやり方じゃ、ぜんぜんダメだ。いったいどうしたんだ。君らしくもないじゃないか」

すると布留川は言いました。

「自分だって精一杯やっている。けれども、物事が前に進まない。誰も大局的な視点でプロジェクトを見ていないからだ。目先の細かいバグや不具合修正に時間を浪費され、腰を据えた最適化もできない。これ以上なんとかしろと言われても、無理だ。“誰か”がきちんとリーダーシップをとって完成させなければ、このプロジェクトはいつまでたってもここから抜け出せない」

珍しく、怒っていたと思います。

そこで僕は理解しました。その「誰か」の役割から逃げ回っているのは、他ならぬ自分なのだと。

部下から嫌われたくない、疎まれたくない、そういう僕の個人的な思いが、「現場の自主性」という美辞麗句を盾にして現場への不干渉という形をとっていました。

しかしこちらからぶつかっていき、嫌われようが疎まれようが、彼らが“正しい道”に戻るた

めに何をすれば良いか、示すべきときに来ていると僕は思いました。部下が育つだとか、才能を開花させるだとか、そういう目に見えないものに過剰に期待し、プレッシャーを与えるのをやめよう。嫌われ、疎まれる自分の人生を受け入れよう。それが彼ら開発チームだけでなく、全社員の人生を預かる私として当然果たさなければならない責任であることは明白でした。

ジェットソン

「今日から開発現場に戻る」

翌週、僕は静かにそう宣言しました。

僕は僕が知る限り最善の方法で開発を立て直す必要性を誰よりも感じていました。全体の処理を効率化し、ユーザインターフェースを改善し、開発者に向けたAPIを整備し、このプラットフォームの上で、作品やアプリを開発する開発者たちにとって、何が必要で、何が不要か、何があるべきで、何があるべきでないか、ユーザが画面を触ったときの手触りをどうすべきで、画面はどのように動くべきか、1コマ単位で指定できる人間をチームに復帰させることにしました。つまり、僕自身です。

本来、100人以上の組織を運営する企業のトップが現場に立ち戻り、ソースコードのレベルまで立ち入って干渉するのは異例のことです。そもそも、僕の会社では過去5年に渡って、僕自身が製品に関連するコードを書かないという暗黙の了解のもとで仕事を進めてきました。しかし、僕たちが開発しようとするenchantMOON、そしてそのOSであるMOONPhaseは、かなり複雑なシロモノで、頭の中に明確なビジョンを描き、強いリーダーシップで牽引しなければ決して本来の姿で完成に向かうことはありません。

僕が現場に戻ると、露骨に不快感を表明する社員もいました。以前なら彼らが退職を匂わせただけで僕は怯んでいたでしょう。しかし、もう僕は別の人間になっていました。このままではenchantMOONは単なる失敗作で、そのコンセプトすらきちんと理解されずに終わってしまう。そのためには何よりもまず、さまざまなしがらみと妥協のもとでどうにもならなくなった最初のソースコードを捨て去り、新しく、正しいコードを書き直そう。ロケットが第一段を切り離すように、さらに身軽で、合目的的なコードを書き直そう。自分の全能力を賭けて、この問題に取り組む必要がありました。

僕は自ら線分の補完アルゴリズムや、フリーハンドで描かれた図形を長方形や円、三角形といった形状に認識させたり、モーフィングさせたりといったアルゴリズムを考案することにしました。新規のアルゴリズムの開発には時間と手間がかかり、スケジュール化が読みづらかったからです。そういう部分は専ら僕が担当し、ほかの開発者たちは現在ある機能のリファクタリングと最適化に全力を傾けさせました。

スカイラブ計画

布留川を始めとするエンジニアたちがMOON Phaseの抜本的な刷新を進める他方、新入社員の福本将悟は、PCやMacなど、enchantMOON以外のデバイスでenchantMOONのコンテンツを閲覧する方法を探っていました。enchant MOONで作られたハイパーテキストをHTML5で再生するためのレンダリングエンジン「Gemini（ジェミニ）」を開発していたのです。

また同時に、コンテンツをアップロードし、ネットを介して共有するためのクラウドサービス「Skylab（スカイラブ）」も準備段階に入っていました。これはアポロ計画以後に打ち上げられた宇宙ステーションSkylabに由来しています。なぜGeminiなのかというと、宇宙の実験室として打ち上げられたスカイラブ計画では、宇宙ステーションとの往復にアポロ宇宙船だけでなく、より小型のジェミニ宇宙船も用いられたという歴史的経緯からです。Geminiは月には行けないけれどもSkylabと地球を結ぶという重要な役割を持っていたことに因んでいます。

福本は複雑なイベントモデルを嫌い、素のHTML5でGeminiのコーディングを進めていました。しかし出来上がったものは、どうも古ぼけて見え、enchantMOONらしさが出ません。じっとコードを見ていた僕はふと思いつき、コードを福本から引き継いで、その場でenchant.jsを使用したエフェクト入りのバージョンを作りました。伏見遼平が開発したtl.enchant.jsを使えば、いとも簡単にエフェクトを実現できたからです。S-IIの画面エフェクトのプロトタイピングにはもともとtl.enchant.jsを使っていました。元のソースコードを使えば、実機以上に狙いどおりのエフェクトが再現されるのは自明でした。このプロトタイプを濱津誠に引き継ぎ、Geminiは完成度を増していきました。

そうして出来上がったGeminiのデビューは、意外なところで果たされることになります。

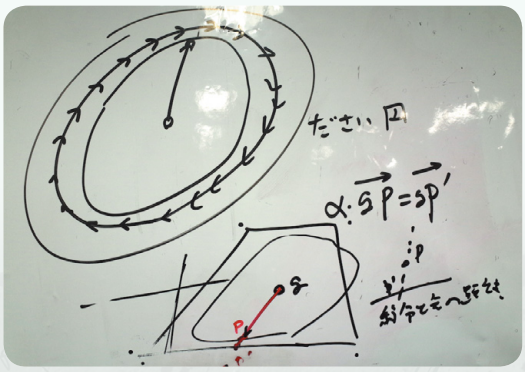
S-IIシークエンス始動

数々の技術者たちの努力により、まるで見違えるように素早くキビキビとした動作になった最新のMOONPhase、そしてSkylab[注1]とも連携し、制作したハイパーテキストコンテンツをほかのユーザやユーザ以外の人とも共有できるレベルまで高めることができた、ある意味で初期コンセプトの完成形に近いバージョン2.9を、僕はサターンV型ロケットの第二段になぞらえ、「S-IIシークエンス」と呼ぶことにしました。

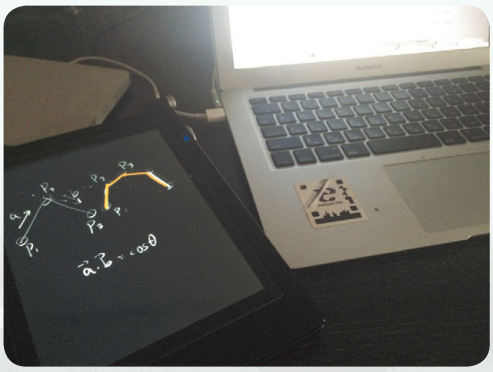
サターンV型ロケットは、名前とは裏腹に実は三段式ロケットです。第一段、S-ICシークエンスは、発射台から打ち上げられ、秒速3.3kmまで加速されます。第一段の燃焼が終了すると、切り離され、第二段ロケット、S-IIシークエンスが開始されて秒速7.0kmまで一気に加速されます。この秒速7.0kmというのは、第一宇宙速度である秒速7.9kmに限りなく近い速度です。このあと、S-IVBシークエンスで7.79kmへ加速され、そこから月へ向かうのです。

大規模なバージョンアップとして予定している3.0の直前、2.9はまさにこの軌道速度へ迫る加速という重要なプロセスを担うS-IIシークエンスなのです。

僕は1年ぶりに内覧会の準備を進めました。そこで用意したのが、enchantMOONで作られた内覧会の告知ページです。第一宇宙速度を意味する方程式をクリックすると、次から次へと印象的なエフェクトで不思議な画像が出てきます。最終的にはATNDのイベント申し込みページにジャンプする、という趣向でした。しかもこのコンテンツは、PCだけでなくiPhoneでもAndroidでも見ることができます。そう。実はこの時点でレンダリングエンジンGeminiをこっそりとお披露目していたのです。2.8までの長い長い燃焼期間を終え、我々はついに月を目指してふたたび加速を始めたのです。SD



モーフィングのためのメモ


補完アルゴリズムを自ら検証

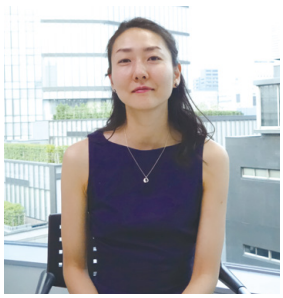
注1） https://skylab.enchantmoon.com/

新連載 軽酔対談

かまぷの部屋

第1献 ゲスト：奥谷泉さん

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



奥谷泉さん

社団法人日本ネットワークインフォメーションセンター（JPNIC）
2000年より現職。JPNICでの主な役割はIPアドレスの管理に関する国内外の調整。ISOC Advisory Council Co-Chair。国内ではアドレス管理の動向紹介や、JPNICでの管理方針の見直しに関する調整に携わっている。
LinkedIn：izumi-okutani

オーストリア・ウィーン Net-grrrlsランチの写真

こんにちは。USP研究所の鎌田広子（かまぶ）と申します。今月から毎回いろいろなゲストを招いて対談いたします。ほろ酔い気分でおもてなししながら、いろいろお話を聞いていこうと思います。

（鎌田）――JPNICにお勤めされているとのことですが、JPNICとAPNICって何がどう違うのですか？

（奥谷）APNICはアジア太平洋地域でIPアドレスの管理を行っている組織です。JPNICは日本国内でIPアドレスとAS番号の管理を行っている組織で、私はIPアドレス管理の調整をしています。といっても数字とにらめっこばかりはなく、海外のカンファレンスに参加したり、テレカン（Tel Con ference）で外国の方と打ち合わせすることが多いです。

――なぜIPアドレス管理を調整する必要があるのですか？

IPアドレスは有限で、どこかの国のものではなくて、世界共通の資源なんです。それを管理しているICANNのIANAが全世界のIPアドレスを管理しているのです。IANAから5つの地域に分けて配っているんですね。北米、南米、中近東、アフリカ、アジアの5つです。APNICはアジアパシフィックで日本はこの中に入ります。IPアドレスはまだしもASは、ルーティングの業務にかかわってないと、馴染みがないですよね。

――ルーティング業務やAS番号って何ですか？

ルーティングは、あるネットワークから別のネットワークにどういうふうに伝わるのか道案内を意味するのですが、AS番号はそのネットワークのかたまりを意味します。それを使って自分のもっているパケットをどう運んでいくのか、IPアドレスとAS番号を使って決めていくんです。ネットを使うすべての方に必要な情報ではないので、馴染みがないのだと思います。

――なるほど。JPNIC歴は長いんですか？　またJPNICに入ったきっかけを教えてください。

非営利団体で働きたくて、仕事探しを始めたことがきっかけです。競争ではなく連携しましょうとか、協力しましょうというパートナーシップを重んじた仕事をしたかったからです。海外との交流もあるだろうと思いました。私自身は技術者ではなく、IPアドレス管理などの調整が主な業務です。また、インターネットのガバナンスに関する動向周知など、JPNICが果たすべき役割はさまざまです。

――非営利団体なのですよね、収益はどうなっているのですか？

会員さんの会費で事業を回しています。職員は25名ぐらいですね。

――少ないですね！　100人ぐらいいるイメージがありました。申請は電子データで管理しているのですか。

そうです。専用のWebシステムで申請いただいて、それで管理しています。オフィスは神田にあって、近くに飲み屋さんがたくさんありま




すよ。

――飲み屋！羨ましいですね。ところで、JPNICはIPアドレスはv4とv6の2つを配っているのですか？

両方ですね。v4が世界的になくなってきているので、枯渇に向けての対策タスクフォースや、v6を利用してもらうような周知、広報活動をしています。アジアの中でもベトナムはとても熱心な国の1つで、技術の習得なども含めv6の普及に対しても積極的です。技術者に話を聞くために、先週も来日していました。

――ところで、大学で『比較文化』を勉強されたということですが。

たとえば、日本語で「甘える」という言葉がありますよね、外国語にはないのですよ。

――へえーそうなんですか？「Depends on」とかは違いますかね。

それだと意味合い的に「依存」に近いですよね。今でも興味の視点は変わっていないです。国際会議でも、仕事の立ち位置だけでなく、国民性も出ます。日本人は和だとかバランスを大事にすると言いますが、本当にそう感じます。

――ええ、本当ですか？　私の周りは個性的な人ばかり……。

日本人は調整役として欧米人に重宝されるんですよ。アジアにもいろんな国があり、欧米人の考え方とは違う国もあります。ですが、日本人は西洋の文化も理解できる部分もあって100%理解しているわけではないですが、雰囲気は掴めているんですよね。一方アジア人でもあるので、アジア的な発想もわかる。国際的な舞台で調整役として間に立つことに日本人が適していると思う場面は、結構あるような気がします。

――いいこと聞きましたね。日本人の良さを知ることで、読者のみなさんのキャリアの視点が広がりそうです。

西洋の考え方ってイデオロギーを重視することが多いです。日本人を含めたアジア人は、正しい正しくないよりも実を取りたいというか、理想と現実を分けて対応する傾向が比較的強い気がするんですよ。

――男女の違いもあると思うのですが、業界やJPNICではどんな感じでしょうか？

スタッフの比率は半分ぐらいが女性です。ただ、業界全体では国内外含め、女性は少ないです。Net-grrrlsというコミュニティ（紹介写真の右側）があって、そこで女性たちとランチをすることがあります。女性同士の会話も楽しいです。海外のカンファレンスに夫婦で参加している方々の中には子供を連れてきているケースもあって、見ていて「いいな」と思いますね。

――そういうやさしい環境であったらがんばれますね。海外出張たくさん行かれているようでうらやましいです。今まで行かれたところで一番よかったところはどこですか？

ブエノスアイレスはまた行ってみたいです。南米は明るくてはしゃぐというサンバのイメージがあったのですが、街の雰囲気がとても落ち着いていて、少し哀愁があって気に入りました。あとお肉がすごく美味しいんですよ。街のごく普通のレストランのウェイターさんも姿勢がきれいで動作が洗練されていました。

――最後に、今年2014年に奥谷さんの出身大学である上智大学に、生成文法とベトナム反戦で有名な「ノーム・チョムスキー」さんがいらっしゃいましたよね。

友人が好きで興味を持ちました。チョムスキー氏は資本主義や体制の矛盾やえぐい部分を鋭く指摘する印象があります。言語学に関する著書も読んでみたいです。

――私も参加したのです。勢いで岩波ハードカバー本を購入しました。まだ読破できてませんが（笑）。今回は楽しいお話ありがとうございました。SD




はんだづけカフェなう

BLEで遊んでみよう

text：坪井 義浩　TSUBOI Yoshihiro　ytsuboi@gmail.com　@ytsuboi
協力：㈱スイッチサイエンス　http://www.switch-science.com/

BLEとは

ご存じの読者も多数いらっしゃると思いますが、BLEとは、Bluetooth Low Energyという低消費電力を特徴とする2.4GHz帯の小電力な無線通信を行う規格の略称です。携帯電話やゲーム機のコントローラなどで広く使われてきたBluetoothですが、バージョン4.0で、このBluetooth Low Energyが盛り込まれました。

Bluetoothという同じ名称が付いていますが、Bluetooth Low Energyのみに対応しているデバイスは、Bluetooth 3.0以前のデバイスとは接続できません。このような事情で、Bluetooth Low Energyのみに対応しているデバイスは、Bluetooth Smart機器と表現されます。一方、従来のBluetoothとBluetooth Low Energyの両方に対応したものは、Bluetooth Smart Ready機器ということになります。たとえば、iPhone 4s以降のiPhoneはBluetooth 4.0を採用していますが、従来のBluetoothもサポートしているため、Bluetooth Smart Ready機器ということになります。

BLEの特徴

BLEの技術的な特徴には、従来のBluetoothよりも進んだ低消費電力という点があります。コイン型電池で1～2年間動かすこともできますが、引き替えに低頻度で少量のデータの通信に向くという特徴もあります。従来のBluetoothは音声を流したりテザリングをしたりと、だらだらと通信をする用途に使われてきました。一方で、BLEが向いているのは、リモコンなどのように、たまに少しのデータ通信を行うといった用途です。

こういった規格の技術的な特徴のほかに、BLEが開発者の間で注目を集めている大きな理由があります。iPhoneやiPadといったiOSを搭載したデバイスで動くアプリケーションと接続する機器を開発するには、AppleとMade for iPhone Program（MFi）の契約を結ぶ必要がありました。しかし、MFi Programに参加するのは、AppleのiOS Developer登録を行うよりも遙かに高いハードルです。しかし、Bluetoothではなく、Bluetooth Low Energyで接続する機器やアプリケーションを開発するときには、このMFi Programの契約を結ぶ必要がありません。

nRF51822

nRF51822は、ノルウェーのNordic Semiconductorが開発した、無線通信モジュールとARM Cortex-M0コアのSoCを搭載したチップです。SoftDeviceと呼ばれるBLEプロトコルスタックがあらかじめ用意されており、比較的手軽にBLEデバイスの開発を行うことができます。比較的手軽といっても、無線通信機器を開発するには、技術と法制度のハードルがあります。

技術的には、高周波回路設計が必要だという点です。デジタル回路とは異なる知識や技術、経験が必要な技術分野ですので、ソフトウェア開発者が気軽に手を出せる分野とは言えません。こういった回路設計を行って、生産されたモジュールがさまざまなメーカーから提供されています。BLEモジュールを使えば、高周波回路設計を行う必要がなくなります。

また、BLEの半導体とアンテナを自分で用意して回路を組んだ場合、外部に電波が漏れない試験環境で動作させるか、あるいは総務省の工事設計認証を得る必要があります。いわゆる「技適」です。この工事設計認証を得たBLEモジュールを利用すれば、認証を受けるプロセスを省略できます。

nRF51822の技術などを得たモジュールも存在するのですが、それでもまだソフトウェア開発者が個人レベルで気軽に試作を行うにはハードルが残されています。こういったモジュールは、基板に搭載することを前提に設計されていて、基板設計などが必要になるという点です。また、ARM Cortex-M0コアの開発を行うには、この技術に関する知識も必要になります。こういった障壁を低くできないだろうかと思っていたところに、筆者も普段から使っているmbed[注1]がnRF51822にも対応するというニュースが流れてきました。

mbed

mbed(エンベッド)については、過去に何度か紹介していますが、あらためて簡単に紹介をします。mbedは、ARMのCortex-Mシリーズのマイコンを手軽に開発できる環境です。mbedは、

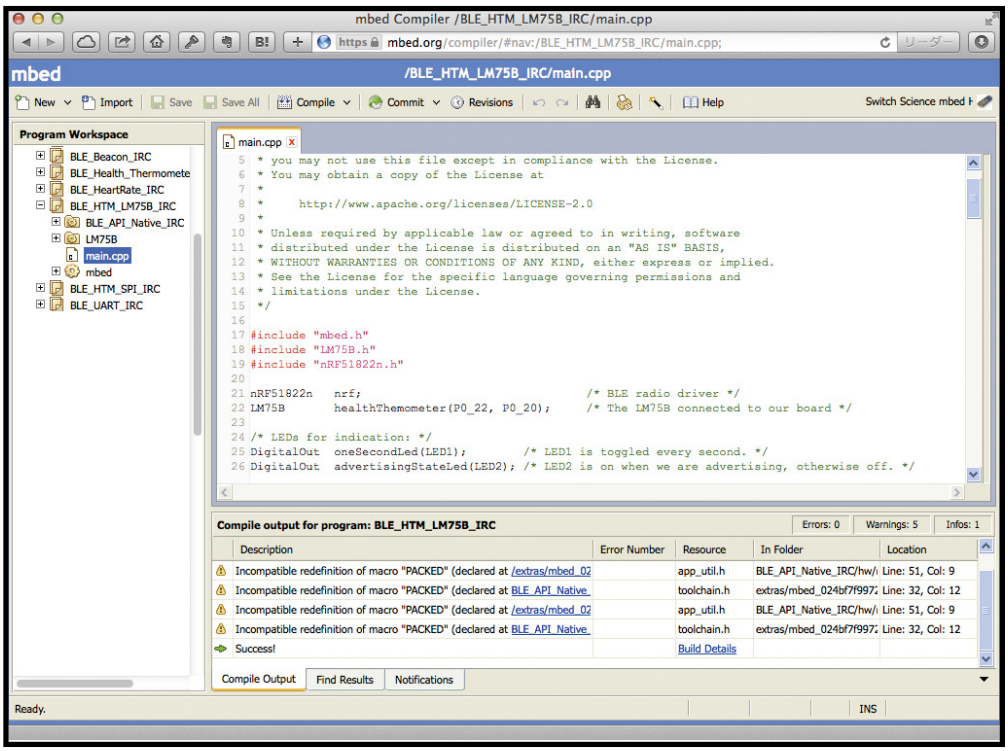
❶セットアップの必要がなく、ブラウザですぐに使うことのできるオンラインコンパイラ(図1)
❷マイコンのI/Oを操作するレジスタの詳細仕様などを意識せず、手早く開発できるmbed SDKと呼ばれるAPI群
❸D&Dするだけでマイコンにプログラムを書き込むことができるインターフェース

で構成されたラピッドプロトタイピングプラットフォームです。

マイコンにセンサなどを接続するときには、一般的にはセンサをコントロールするコードも

注1) http://mbed.org/

▼図1 オンラインコンパイラ



書く必要があります。しかし、mbedのオンラインコンパイラにはコードやライブラリを共有するSNSのような機能もあり、ほかの人が書いて公開してくれているコードを再利用して手早く開発を進めることができます。mbedがnRF51822に対応したことで、nRF51822の詳細仕様を把握する必要なく、手軽にセンサをつなげたりして自分が必要なBluetooth Smart機器のプロトタイプが可能になりました。

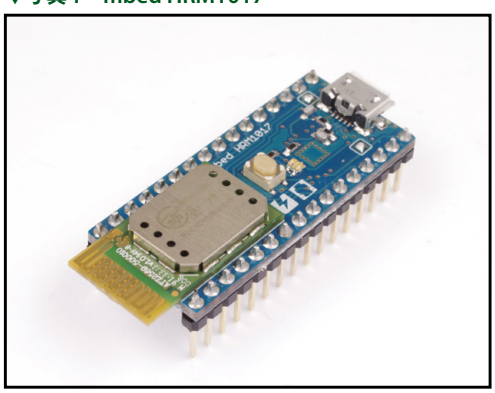
しかし、Nordic Semiconductorの開発したnRF51822-mKITというmbedは、日本の認証を受けていません。このため、認証を得たホシデンのHRM1017というBLEモジュールを使って、国内で適法にBLEの開発ができるmbedを作ってみました。

mbed HRM1017

写真1が、試作したmbed HRM1017です[注2]。ブレッドボードを使ってちょっとした手間で試作ができるような形状にしました。nRF51822は、I²CやSPIといったよく使われるインターフェースを備えています。筆者はまだBLEの学習中ですので、mbedのWebサイトにあったBLEを用いた体温計のサンプルプログラムと、Nordic Semiconductorが配布しているiOSアプリケーションであるnRF Toolboxの組み合わせで遊んでみました。

注2) http://www.switch-science.com/catalog/1755/

▼写真1 mbed HRM1017

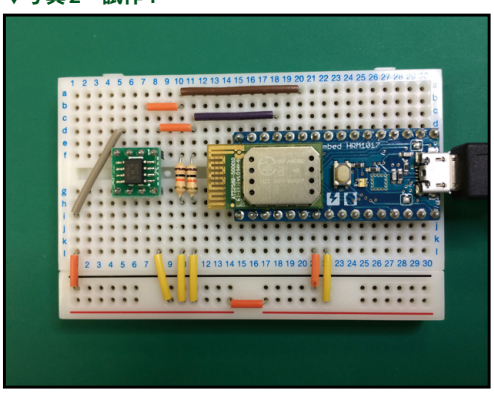


体温計ですので、温度センサをmbed HRM1017に接続してみる必要があります。ここでは、NXPのI²C接続できる温度センサの、LM75Bを使ってみました。mbed HRM1017とLM75Bをブレッドボードの上に組んでみたのが**写真2**です。ブレッドボードには抵抗も2本搭載されていますが、これはプルアップ抵抗というもので、I²Cを使うときには必要になるものです。

サンプルプログラムが使っていたTMP102というセンサのライブラリをLM75Bのライブラリに入れ替えて、オンラインコンパイラでコンパイルすると、HEXファイルがブラウザでダウンロードされてきます。このHEXファイルを、パソコンからはUSBフラッシュメモリのように見えるmbedのドライブにドラッグ&ドロップすると、BLEモジュールにプログラムを書き込むことができます。書き込みが終わると、モジュールの中のCPU(nRF51822)がリセットされ、プログラムが動き始めます。iPhoneのnRF Toolboxでmbed HRM1017に接続すると、**図2**のような画面が表示されました。無事に温度を読み込むことができています。mbedへの慣れもあるとは思いますが、1時間もかからずにこのような実験ができるのがmbedでBLEの試作ができることのメリットです。

温度センサを接続し、気温をBLEで得て、iPhoneで表示できたので、ほかの温度センサを接続してみました。第4回でも扱ったことの

▼写真2 試作1



ある、熱電対です。熱電対というのは、異なる2種類の金属の接点を加熱すると一定の方向に電流が生じる、ゼーベック効果を用いた温度センサです。この電流を測定してSPIで温度を出力するチップを搭載した、スイッチサイエンスの「K型熱電対温度センサモジュールキット(SPI接続)MAX31855使用 3.3V版[注3]」を**写真3**のように接続してみました。このMAX31855というチップのライブラリはmbed.orgで見当たらなかったので、公開されていたArduinoのスケッチを参考にして、mbedのコードに書き直しました。どちらもC/C++ですので、移植はかなり容易に行えます。同じく、コンパイルしてダウンロードしたHEXファイルをドラッグ&ドロップすると、iPhoneで気温を表示することができます。ただし、K型熱電対は、室温くらいの低い温度だと相対的に誤差が大きいので、あまり参考になりません。この熱電対キットの計測できる範囲は、－200～＋1,350℃とのことですので、ライターの炎を熱電対に当てて、温度を測定してみました。Health Thermomitorということで、体温計のデモアプリケーションです。高い温度は表示できないのではないかと心配していましたが、無事に表示されました(**図3**)。

まとめ

このように、BLEを搭載したマイコン、nRF51822をmbedで開発できると、とても気軽にスマートフォンと連動のできるBLE対応機器をプロトタイピングできます。新しい技術を身に付けるのには、実際に手を動かして開発を経験するのが一番の近道だと思います。

mbed.orgには、iBeaconのサンプルプログラムもありますので、iBeaconを試してみたいデベロッパも、気軽に自分のビーコン送信機を作ることができます。また、iOSのデベロッパ登録をしていなくとも、“techBASIC”というiOSアプリケーションを使って、お手持ちのiPhoneでBLEアプリケーションを試作することもできます。

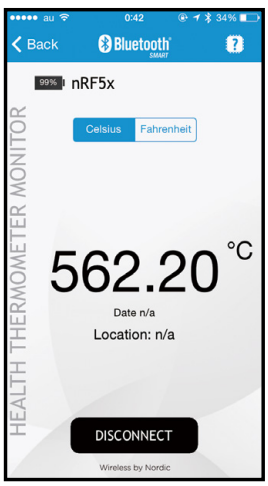
筆者はAndroidにあまり馴染みがなくなってしまったので詳しくはないのですが、API Level 18でBLE関連のAPIが搭載され、Android 4.3以上が動いているBLE対応の端末であれば使うことができるようです。

Bluetooth Low Energyを使うと、スマートフォンに実装できるアプリケーションやサービスの幅が大きく広がります。ぜひ、自分のBluetooth Smartデバイスを作って遊んでみてください。SD

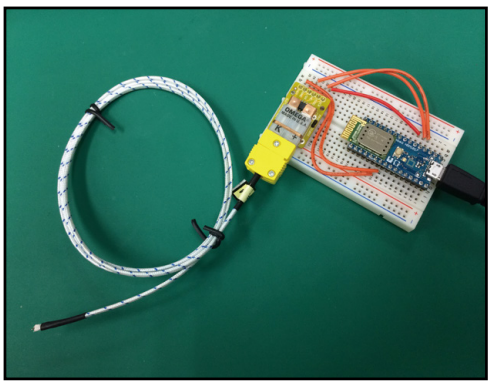
注3) http://www.switch-science.com/catalog/864/

▼図2 nRF Toolboxの画面

▼図3 高温も表示された



▼写真3 試作2



PRESENT 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」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは 2014 年 8 月 17 日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートのご回答について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。ご入力いただいた個人情報は、作業終了後に当社が責任を持って破棄いたします。なお掲載したプレゼントはモニター製品として提供になる場合があり、当選者には簡単なレポートをお願いしております（詳細は当選者に直接ご連絡いたします）。

01 Microsoft All-in-One Media Keyboard

1名

フルサイズのキーセットと一体型マルチタッチトラックパッドを備えたコンパクトなキーボードです。トラックパッドで、スワイプ、ドラッグ、ズーム、クリックタッチの操作ができ、マウスがなくてもパソコンを扱えます。また、カスタマイズ可能なメディアホットキーが備えられており、Web やお気に入りの音楽、写真、映画を指先で操作できます。防滴仕様により、万が一飲み物がこぼれても水分を簡単に取り除けます。インターフェースは、ワイヤレス接続（USB 子機をパソコンに接続）となっており、最大 10m の範囲で操作ができます。

提供元 日本マイクロソフト **URL** http://www.microsoft.com/ja-jp

02 OTG USB メモリ Mobile X20

2名

USB2.0 OTG 対応の USB フラッシュメモリです。1 つのボディに Micro USB と標準 USB（A タイプ）のコネクタを両方搭載しており、スマホやタブレット、パソコンそれぞれに直接接続して使用できます。容量は 32GB になります。

提供元 シリコンパワー **URL** http://www.silicon-power.com

03 ウイルスバスターモバイル

2名

※上の画像は、サンプルです。

iPhone/iPad（iOS）に対応した「ウイルスバスターモバイル」の最新版です。今回提供するのは、パッケージ版と同じ機能、期間で使用できる非売品のカードです。専用サイトから同製品をダウンロード後、アクティベーションキーを入力することで、利用いただけます。

提供元 トレンドマイクロ **URL** http://www.trendmicro.co.jp

04 Red Hat ノベルティグッズ 4点

3名

Red Hat 社のノベルティグッズです。同社のロゴが入った手帳、ファスナー付クリアケース、布細工の帽子のミニチュア、RED HAT ENTERPRISE LINUX 7 のロゴステッカーの 4 点セットで提供します。

提供元 レッドハット **URL** http://jp.redhat.com/

05 Nginx T シャツ & ロゴステッカー

3名

「Nginx ユーザ会 #0」で配布された T シャツです。T シャツのサイズは S、M、L がありますので、ご希望のサイズがあれば応募の際にアンケート内にご記入ください。ロゴステッカーも 1 枚ずつお付けします。

提供元 サイオステクノロジー **URL** http://www.sios.com/

06 OpenStack 入門

2名

中井 悦司、中島 倫明 著／
B5 変型判、208 ページ／
ISBN=978-4-04-866067-9

業務システムとしてプライベートクラウドを構築しようとしているエンジニアのために、OpenStack の構造や考え方・特性を解説する入門書。7 番目のリリースである「Grizzy」を基に解説しています。

提供元 KADOKAWA **URL** http://www.kadokawa.co.jp/

07 パーフェクト Ruby on Rails

2名

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／
ISBN=978-4-7741-6516-5

Ruby や Ruby on Rails に関する基本的なことから、開発や運用に活用するツール、拡張方法など、現場で役立つ知識を中心に据えた、最新の Rails4.1.1 に対応した書籍です。

提供元 技術評論社 **URL** http://gihyo.jp/

第1特集

身近なシステムログからウェブやデータベース、そしてビッグデータの基礎まで

ログを読む技術

―手がかりを見いだす眼力をつくる―

エラー原因の追及からユーザ動向までログは宝の山

「さあ、ログを蓄えて宝の山を目指そう！」と、ひとかたまりの宝を探す冒険、とまではいきませんが、コンピュータが動くとログが蓄積されます。そのログには宝の山と言える情報がたくさん入っています。システムの内側で起きた出来事を記録していくシステムログ、Web サーバに接続してくるクライアントの行動を記録するアクセスログ、データベースの利用状態を記録するクエリログやエラーログ……。サービスの実装も大事ですが、ログを分析したり利用することで、管理・運用の効率が上がったり、次に何をすべきかわかるようになります。意外に知られていない、でも重要なログの扱い方を 70 ページ超の特集で解説します。システムログから MSP のログ監視テクニック、そして Fluentd と MongoDB を利用した現代的なログの使用方法まで一挙に習得してみませんか？

CONTENTS


第1章 ログの基本をおさえておこう 22


第2章 Webサーバのログを見てみよう 37


第3章 MySQLのロギングを見てみよう 52


第4章 ログを管理・運用しよう 63
（ログローテーションとログウォッチ）
Writer 第1～4章 近藤 成


第5章 MSP直伝・プロがやっているログ監視 73
Writer 高村 成道


第6章 フロントエンジニアもFluentd＋MongoDBで実践！
小さく始めるログ活用のすすめ 83
Writer 羽田 健太郎

イラスト 高野 涼香

第1章 ログの基本をおさえておこう

Writer 近藤 成(こんどう じょう)
Mail jj2kon@gmail.com Web http://server-setting.info/

ログは、システムの稼働状況をはじめとして、さまざまな情報を蓄積しています。本特集では、システムログ、アプリケーションのログ、それぞれのログの読み方・扱い方を解説します。これによって、システムに障害が起きたときに原因追及ができるようになったり、分析ができるようになります。まず第1章では、ログの基本をしっかりと学びます。

ログは、いつ、だれが、どこに収集するの?

そもそもログは、ソフトウェア[注1](プログラムの意味)が実行された経過情報を出力したものです。つまり、**プログラムが実行されたときに、あらかじめ決められている出力先(ファイルなど)へ、プログラムの処理情報を書き込みます。**

一般的なUNIX系のアプリケーション[注2]のほとんどは、ログの出力先(ファイル名など)を指定ができるようになっており、そこにあらかじめ設定しておけば、そのアプリケーションが起動した際に、その設定情報を読み込み、指定してあるログの出力先(ファイルなど)へ出力されるようになります(**図1**)。

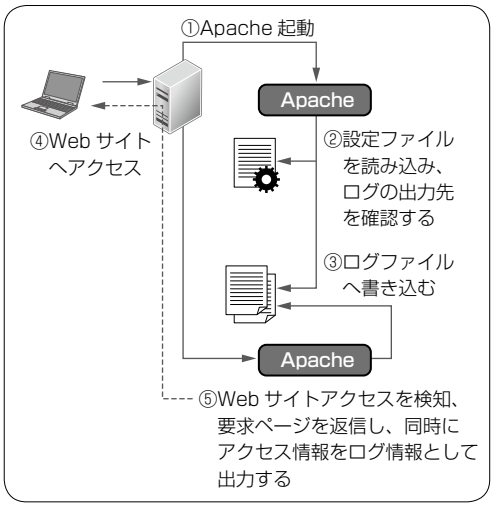
ヒント ここで紹介するCentOSのデフォルトの設定では、ログの出力先は、ほとんどの場合、**/var/log/**配下のディレクトリおよびファイルへ出力するようになっています。

たとえば、**図1**はWebサーバとして有名なApacheのログ出力までの大きな流れです。

①OSからApacheが起動される
②Apacheは、起動時に設定ファイルの情報を読み込む。そこで、ログの出力先を確認する
③ログの出力先にApacheが起動した時のログ情報を出力する
④Apacheが起動完了し、Webサイトへのアクセスが可能となる。そこへ、ユーザからそのWebサイトへアクセスがあったとする
⑤ユーザからWebサイトへのアクセスをApacheは検出し、ユーザへ要求ページを返信すると同時に、日時情報(いつ)とともにユーザ情報(だれが)、要求されたページ情報(何をしたか)をログ情報として出力する

このような流れでApacheのログ情報は出力されます。Apacheのログについては、「第2章 Webサーバのログを見てみよう(p.37)」で解説

▼図1 Apacheのログ出力までの流れ



注1) ここでは、ハードウェアとの違いを強調するためソフトウェアという表現を使っています。ここでの意味は、プログラムと同義です。

注2) プログラムの中で一般的にサービス(デーモン)として提供されているプログラムをここではアプリケーションと呼ぶことにします。

します。

ヒント 図1では、ログの出力先をファイルとしていますが、ログの出力先はファイルとは限りません。データベースやメールなどへ出力されることもあります。

syslogはログの基本です

Unix系OS(Linuxも含む)では、何といってもログと言えばsyslog(シスログと呼ばれる)です。

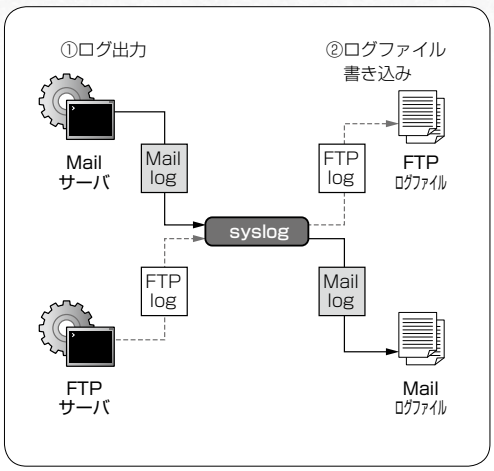
そもそもsyslogは、Mail(SMTP：Simple Mail Transfer Protocol)サーバの代表的なアプリケーションであるsendmail[注3]のログアプリケーションとして開発されたものです。つまり、最初は、単純にMailサーバ専用のロガープログラムだったわけです。これが便利だと気づいた開発者たちが、こぞってほかのアプリケーション(FTPサーバなど)でも取り入れて、sendmailと同じようにsyslogアプリケーションを使ってログ情報の出力を行ったので、デファクトスタンダードになりました。このデファクトスタンダードになったsyslogを体系づけてRFC 3164[注4]にまとめたのがsyslogプロトコルと言われるものです。

ヒント 一般的にsyslogというと、広義のアプリケーションの総称(たとえば、以降で解説するrsyslogやsyslog-ngなどのアプリケーションを含めたもの)として使うことが多いと思っていました。しかし、最近のWeb情報ではsyslogとはプロトコルだという記事をよく見かけるようになりました。個人的に、少し違和感を感じて調べてみるとwikipediaにそれに近い表現で書かれているのを見つけました。想像ですが、その情報を元に、いろいろな方々が書かれたのではないかと思います。このような呼称は、広く普及したほうが正しくなってしまうので、今では、どっちが正しいとは言えなくなってきているのかもしれません。

注3) http://www.sendmail.com/sm/open_source/

注4) RFCとは、Request for Commentsの略で直訳すれば「コメント募集」となります。もともとは、広く意見を吸い上げる意味合いで使用されたようですが、今では少し異なり、インターネットに関する技術の標準を定める団体であるIETFが正式に発行する公開文書を意味します。RFC 3164では、The BSD syslog Protocolが定義、公開されています。

▼図2 Syslogのログ書き込みの流れ



syslogの基本機能①「ログを書き込む」

1つ目の機能はログを書き込む機能です。言い換えれば、ログの出力を管理する機能のことです。たとえば、ログの出力先がファイルの場合は、ログファイルへの書き込み、ログファイルの管理を行う機能のことです。

たとえば**図2**は、Mailサーバ(sendmailやpostfix[注5]など)やFTPサーバ(vsftpd[注6]など)からのログ情報をsyslogがログファイルへ書き込むまでの大きな流れです。

①MailサーバやFTPサーバなどでログ情報を書き込みたい場合、syslogへログ情報を渡す
②syslogはそのログ情報を受け取り、誰からのログ情報か確認し、各アプリケーション用のログファイルへ書き込む

このように非常に単純な流れです。これによってMailサーバやFTPサーバは、ログファイルへの書き込みおよびそのファイルの管理を行う必要もなく、手間が省けて大助かりというわけです。ただ、必ずsyslogがインストールされているとも限らないので、一般的なアプリケーショ

注5) http://www.postfix.org/ [日本語]http://www.postfix-jp.info/

注6) https://security.appspot.com/vsftpd.html

ンは、自前のログ情報出力・管理機能を持っています。もちろん、syslogへの出力機能も、ほとんどのアプリケーションが有しています。

ヒント Windowsでは、このsyslogのログの出力管理機能をイベントログが行っています。

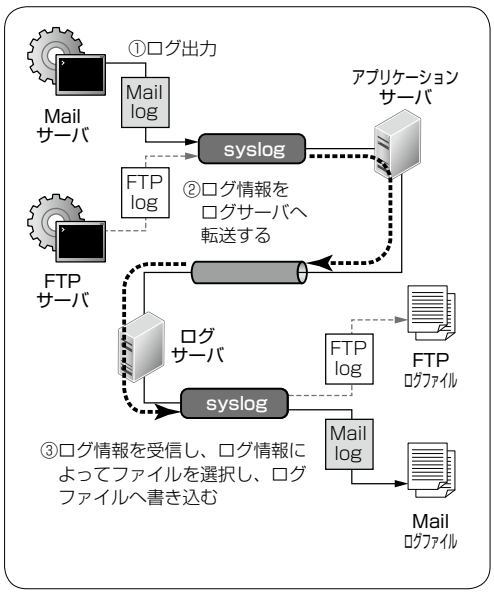
syslogの基本機能②「ログを収集する」

2つ目の機能は、ログの収集管理機能で、複数のサーバのログ情報を1台のログ専用サーバで集中的に収集し管理する機能です。

たとえば**図3**は、**図2**の例に倣ってMailサーバ(sendmailもしくはpostfixなど)や、FTPサーバ(vsftpdなど)からのログ情報をsyslog(アプリケーションサーバ)がログの専用サーバ(ログサーバ)へ送信し一括管理するという大きな流れです。

①MailサーバやFTPサーバなどでログ情報を書き込みたい場合、アプリケーションサーバ内のsyslogへログ情報を渡す
②アプリケーションサーバ内のsyslogは、そのログ情報を受け取りログサーバへ転送する
③ログサーバ内のsyslogは、そのログ情報をアプリケーションサーバから受け取り、誰からのログ情報かを確認し、各アプリケーション用のログファイルへ書き込む

▼図3 syslogのログ収集の流れ



このようにネットワークを介してログ情報を一ヵ所に集めて管理できます。大規模なシステムでも、この機能を利用し、ログサーバの情報管理をしっかり行えば、運用の手助けになることは間違いありません。

このようにアプリケーションを開発する側としたら、ログ情報をsyslogへポイポイと投げておけば、あとはsyslogが何とかしてくれるわけです。確かに便利ですよね。

syslogというアプリケーションは今や利用されていない

CentOS 5では、アプリケーションとしてのsyslog**d**(syslog**デーモン**)は、

・syslogd
・klogd (カーネルログデーモン)

の2つのアプリケーション(デーモン)をsysklogdという1つのパッケージで提供していました。しかし、syslogdは、バージョン1.5(2007年リ

Column syslogがなかった時代

syslogがなかった、あるいはアプリケーションがまだsyslogに対応していなかった時代では、ログの一括管理は自前のUDPによるファイル転送などを駆使して行っていました。その当時は、まだまだ、電話回線＋モデムでの接続がメインで、TCP接続での常時接続などは夢物語のような時代でした。UDP接続で1日に数回、コンピュータ同士を接続するのが普通の時代です。メールもそのUDP接続で1日に数回送られていた時代でもありましたから、メールのやりとりは、文字どおり郵送の手紙感覚でした。今でこそ、ほぼリアルタイムに送受信できるメールもそんな感じだったんですよ。このsyslogも例外でなく、開発された当時は、UDPがメインでした。そのため、ここでの転送もUDPがメインです。

リース）で更新が止まってしまいました[注7]。また、syslogdは、ログ情報の紛失の可能性やネットワーク上のログ情報が暗号化ができないなどのいくつかの問題が指摘されていました。

これらの問題から、とくに多くのLinuxディストリビューションでは、sysklogd（つまり、syslogd、klogd）パッケージは採用されなくなっています。

代わりに採用されているsyslogアプリケーションの後継者には、大きくsyslog-ngとrsyslogの2つがあります。以降、この2つのアプリケーションについて解説します。

syslog-ngの機能とは

syslog-ng[注8]は、syslog New Generationの略で、直訳すれば「次世代syslog」ぐらいの意味でしょうか。この名前のとおり先のsyslogアプリケーションの問題を解決すべく開発されたアプリケーションです。syslogプロトコルのサポートはもちろんのこと、次のような主だった機能が追加されています。

- ログの分類機能
- TCPによるログ情報の送受信（ログ情報の紛失の回避）
- SSL/TLSを使用してセキュアログ（ネットワーク経由の暗号化の実現）
- データベースへのログ出力

などさまざまな機能が盛り込まれています。

rsyslogの機能とは

rsyslog[注9]は、rocket-fast system for log processingの略で、直訳すれば「猛烈に早いsyslog」ぐらいの意味でしょうか。そもそもrsyslogは、標準のsyslogdの後継として始まりましたが、多種多様なソースコードからの入力を変換、結果をさらに多様な出力先への書き込

▼写真1　スイス・アーミーナイフ

このようになんでもかんでも1つのナイフにぶら下がっている様を言いたいんでしょうか、それとも、これだけ機能が豊富だよということでしょうか。いずれにせよ、便利であることは間違いないと思います。

みを可能にすることでロギングのスイス・アーミーナイフのようなもの（**写真1**）へと進化しました（このキャプションは公式サイトの訳です）。

先のsyslog-ng同様、rsyslogもsyslogプロトコルのサポートはもちろんのこと、次のような主だった機能が追加されています。

- ログの分類機能
- TCPによるログ情報の送受信（ログ情報の紛失の回避）
- SSL/TLSを使用してセキュアログ（ネットワーク経由の暗号化の実現）
- データベースへのログ出力

ヒント Webで情報でrsyslogを検索すると、rsyslogは、reliable syslog（信頼性の高いsyslogの意味）の略だというページを多く見つけました。reliable syslogを目指したのは確かですが、名前の由来は、公式サイトで上記のように記載があったので、プロトコルと混同されているのではないかと思います。

CentOS 6では、このrsyslogが採用されています。また、Debianでもrsyslogが採用されており、主要な2つのLinuxディストリビューションで採用されたことによって、rsyslogが今やデファクトスタンダードになりつつあると言っても良いかもしれません。

注7） http://freecode.com/projects/sysklogd

注8） ライセンス：LGPL（core部）、LGPLv2（plugin部）http://www.balabit.com/network-security/syslog-ng/opensource-logging-system

注9） ライセンス：GPLv3（http://www.rsyslog.com）

ここまで、ログ、syslogの基本的な機能を解説してきました。ここからは、実際にCentOS 6を使って具体的な設定、使い方を解説します。また、以降syslogと表記する場合は、狭義の意味でのrsyslogを指すのではなく、広義の意味でsyslog対応アプリケーションの意味として表記することに注意してください。

syslogへコマンドを使って出力してみよう

ここでCentOS 6 + rsyslog環境で実践することで、まずはsyslogが、どのようなものかを肌で感じてもらおうと思います。CentOS 6は、2014年5月現在の最新版6.5です。rsyslogのバージョンは、5.8.10です。また、CentOS 6は便宜上リモートアクセス(SSH接続)して使用します。SSH接続するためには、SSHサーバ(openssh-server)がCentOS側で起動していなければいけません。

もし、SSHサーバ(openssh-server)がインストールされていないようなら、サーバのコンソール画面から次の要領で簡単にインストールできますので、インストールしてみましょう。

```
$ yum install openssh-server
……(省略)……
Is this ok [y/N]: y
……(省略)……
Complete!
```

インストールを終えたら、起動しておきましょう。

```
$ /etc/init.d/sshd start
sshd を起動中:                                [  OK  ]
```

インストールされた状態で、なおかつ、何も変更されていない状態(**初期(デフォルト)状態**)であることを前提に解説します。

リモートアクセスしてみよう

まずは、CentOSへログインしてみましょう。SSHによるリモートログインを行うには、Macでは、Mac OS Xターミナルを使うと良いでしょう。Windowsのターミナルソフトは、コマンドプロンプトです。そもそもデフォルトで**ssh**コマンドが存在しませんので、**ssh**コマンドのインストールを行うか、別のターミナルソフトを使うことになります。ここではターミナルソフトのTeraTermを使ってリモートログインを行ってみます(TeraTermのインストールおよび設定は多くのWebサイトで紹介されていますので、ここでは割愛します)。

コマンドを使ってみよう

まずは、簡単な**ls**コマンド(Windowsでいうところのdirコマンド)を使ってみます(**図4**)。

パラメータの"**-l**"はリスト出力、"**-a**"は、すべてを意味します。実行すると上記のようにすべての情報をリスト形式で出力します。

次に**ps**コマンドを使って、syslogのプロセスを確認してみます(**図5**)。

psコマンドは、現在のプロセスの状態を出力するコマンドです。"**x**"は、呼び出したユーザの所有する全プロセスを出力するという意味で、"**a**"は、端末(tty)を持つすべてのプロセスをリストで出力するという意味になります。ちょっとわかり難いですが、"**ax**"を指定することで全プロセスを出力してくれると覚えてお

▼図4　lsコマンドを試す

```
$ ls -al
合計 20
drwx------. 2 hoge hoge 4096  5月 25 06:33 2014 .
drwxr-xr-x. 3 root root 4096  5月 25 06:33 2014 ..
-rw-r--r--. 1 hoge hoge   18  7月 18 22:15 2013 .bash_logout
-rw-r--r--. 1 hoge hoge  176  7月 18 22:15 2013 .bash_profile
-rw-r--r--. 1 hoge hoge  124  7月 18 22:15 2013 .bashrc
……(省略)……
```

Column

ログはコンピュータの行動記録

ログとは、そもそも英語ではlogと書きます。英和辞典を調べてみると最初に出てくるのが、丸太、材木を切り出すという意味です。次が、測程儀(船の速度を測る器具)や航海(航路)日誌(**写真2**)に記入するなどの意味が出てきます。この2つは、まったく意味が異なるもののようですが、これらを結びつけるものは"船"であり"測程儀"です。船の速度を測る器具(測程儀)に手用測程儀(hand log)というものがあります。これは、木片に長い紐を括り付けた簡単な道具です。使い方も簡単で、木片を海に浮かべ、紐を船上から垂らし、その紐がスルスルと簡単に流れ出るようにして船を走らせるだけです。一定時間内に紐がどれだけ流れ出たかで船の速度を測定するというものです。船の速度でノット(英語：knot、日本語：結び目)という単位が使われるのは、この紐に一定間隔で結び目をつけ、先の計測方法で流れ出た結び目の数を船の速度としたことに由来します。この木片が丸太(log)であり、船の計測が航海日誌(logbook)へと結びついたとされています。

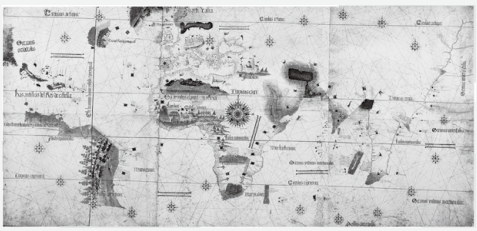
また、コンピュータ、とくにプログラミングの世界では、プログラムが経時(あるいは処理経過)ごとに(「いつ、だれが、どこで、何をした」という情報を)記録することを「ロギングする」、記録したものを「ログ」と呼びます。英語で"記録"はrecordという単語を使うことが多いですが、まさにlogが使われているのは、航海日誌(logbook)のように時間経過(あるいは作業経過)とともに記録を残したことにたとえてのことだと言われています。

さて、本題に入りましょう。なぜログが必要なのでしょう。

それは、大きく2つの理由があります。1つは、不具合の修理、改善のためです。コンピュータシステムに完璧なものはありません。むしろ、コンピュータシステムほどよく壊れるものはないとさえ言えるかもしれません。今でこそ、少なくなりましたが、パソコンが固まる(突然、動かなくなる)のは日常茶飯事でした。もし、コンピュータシステムが完璧で壊れないものであれば、そのような過去を記録した情報(ログ)は、必要ないのかもしれません。しかし、コンピュータシステムに限らず完璧なもの、壊れないものはありません。安全神話が妄想であるように、それを正しく理解していれば壊れたときにどうしようかと考えるでしょう。もし壊れれば、その原因を究明し、修復・修繕し、二度と同じような壊れ方をしないように(英語にもなった)改善(カイゼン)を図るでしょう。その原因究明には、壊れた時の状態・情報が非常に大事です。その貴重な情報がログです。そのログが経時的(あるいは経過的)に記録された情報であることから、壊れた時に何が起こったか、ハードウェア・ソフトウェアを含めてシステムの状態を時間をさかのぼって把握することができるのです。その昔、大航海時代には、航海日誌が安全な航海のための貴重な情報だったように、このログも安定したシステム運用のための貴重な情報なのです。

もう1つは、昨今、注目されているビッグデータに代表される痕跡情報(アクセス情報)としてのログの必要性が高まったことにあるでしょう。これは、ログの特徴である「いつ、だれが、どこで、何をした」という情報から、人の動向や意識、マーケティングの調査などのためのデータマイニングの元データとして用いられたりします。具体的に身近な例として、Webサイトのアクセスログの解析があります。Webサイトのアクセスログ解析では、どのページから入ってきて、どのページで離脱したか、どの地域の人がどれくらいアクセスしたかなどさまざまな分析が行われます。その分析結果は、より人が興味を持つようなページ作りや人を見せたいページへどのように導いていくか、いわゆる導線の張り方を考える材料などに用いられたりします。これらは、先の不具合の修理のように過去のデータから現在を改善するのではなく、過去のデータから未来を予測(改善)するという、同じログの情報でも活用の範囲を広げた1つの解析(分析)方法でもあります。最近では、インターネットの発展とともに、これらの情報活用が非常に注目を集めているだけに、ログというとこちらのイメージが強い方も多いようです。このように、ログが、さまざまな切り口で利用され、改修、改善を図る貴重なデータであることは間違いありません。

▼写真2　大航海時代の海図



くと良いと思います。

“|”(パイプと言います)は、続けてコマンドを実行しなさいということです。つまり、psコマンドを実行したあとの主力を受けて、“|”の後のgrepコマンドを実行しなさいということです。

grepコマンドは、ファイルやコマンドで出力された文字列情報からパラメータ指定の文字列を検索し、ヒットした行を出力するコマンドです。ここでは、psコマンドが出力した内容(文字列)をgrepで指定している文字列“syslog”で検索した結果を画面に表示しています。図5の出力結果からすれば、rsyslogd(rsyslogデーモン)プロセスが、ちゃんと動いています。

ここで紹介したps、grepコマンドは非常によく使う基本コマンドですので、使い方をマスターしておいたほうが良いでしょう。

では、続けて、ログを出力してみましょう。

ログを出力してみよう

SSHでログインできるようになったら、2つのターミナル(ここでの例ではTeraTerm)画面からログインしてください(TeraTermのメニューから[ファイル]−[新しい接続...]を選択することでいくつでも画面を開くことができます)。これから次のように、TeraTermを操作してみましょう(図7)。

▼図5　psコマンドでsyslogのプロセスを確認

```
$ ps ax | grep syslog
  797 ?        Sl     0:00 /sbin/rsyslogd -i /var/run/syslogd.pid -c 5
 1528 pts/0    S+     0:00 grep syslog
```

ヒント　図4はユーザhogeのホームディレクトリの出力結果です。ここで、.bashrc、.bash_profile(bashのユーザ設定ファイル)があるようにLinuxのほとんどのディストリビューションでデフォルトのシェルにbashを採用しています。Bourneシェルと後方互換性を持つbashを含めて、広義の意味でBシェルということがあります。これは、BSD系のCシェル(csh、tcsh)と対比させた言い方となっています。このbashは非常に便利で、コマンド補完や履歴読出しなどいろんな機能があります。たとえばgrepと入力したいけど、綴りを忘れたときは、“gr”まで入力しTABキーを押下すれば、1つしか候補がなければ、“grep”までを入力補完してくれます。2つ以上の候補がある場合は、再度TABキーを押下すれば図6のように候補コマンドが一覧として出力されます。

また、↑キーを押下すれば、前に入力したコマンドが表示されます。1つ前に実行したコマンドがls -alなら、↑キーを1回押下すれば次のように出力されるはずです。

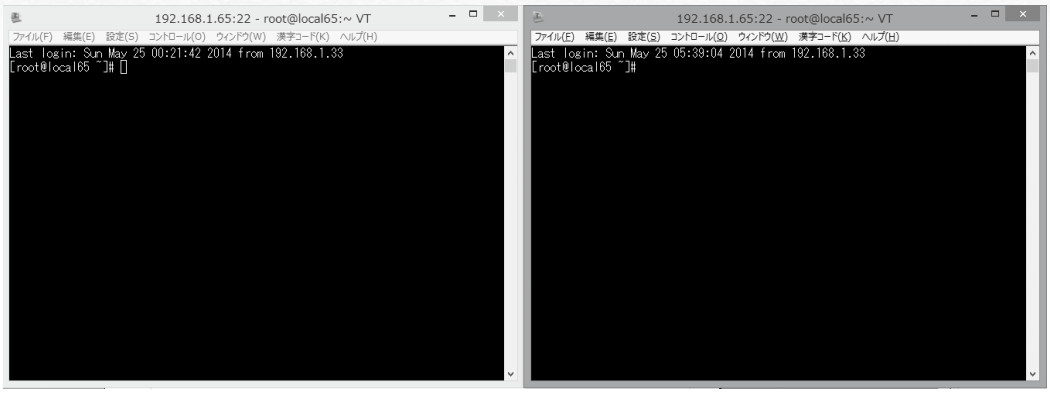
```
$ ls -al
```

これぐらいの機能なら、最近ではWindowsのコマンドプロンプトでもできるようになりましたが、UNIX系のシェルがその元祖です。参考までに、Mac OS XがBSD系のUNIX OSであることを知らない方も多いみたいです。そんな方は最近(Mac OS X 10.0から10.2.8まで)まで、Mac OS Xが採用していたデフォルトシェルが、BSD系のデフォルトシェルであるCシェル(tcsh)だったことも、今はbashなのもご存じないかもしれませんね。

▼図6　bashの入力補完機能

```
$ gr
grefer          grolbp          groupmod        grub-install
grep            grolj4          groups          grub-md5-crypt
grn             grops           grpck           grub-terminfo
grodvi          grotty          grpconv         grubby
groff           groupadd        grpunconv
groffer         groupdel        grub
grog            groupmems       grub-crypt
$ gr
```

▼図7　TeraTerm同時2接続画面イメージ



・一方の画面(図8では、TeraTerm#2)からログ情報の書き込みを行う

・他方の画面(図8では、TeraTerm#1)でリアルタイムにログ情報を確認する

図7のように準備ができたら、ログ情報を出力してみましょう。次のページの**図8**は、ここで準備した2つのターミナル画面に対して、時系列にコマンドを入力する手順を記載したものです。まずは**図8**の手順(1)から順番に、コマンドをそれぞれの端末から入力してみてください。

図8で使っているコマンドは2つだけです。それも非常に簡単な使い方だけです。その2つのコマンドについて以降解説します。

tailコマンド

1つは、**tail**コマンドです。このコマンドは、ファイルの末尾を出力するコマンドです。デフォルトでは、指定されたファイルの末尾の10行を出力します。ファイルが10行に満たない場合、すべての行を出力します。**-f**オプションは、ファイルの内容を常に監視し、表示をリアルタイムに更新するというものです。

```
$ tail -f /var/log/messages
```

と入力した場合、**/var/log/messages**というファイルの末尾10行を出力し、引き続きファイル内容を監視します。ファイルの更新があれば、更新分を画面にリアルタイムに出力します。

loggerコマンド

もう1つは、**logger**コマンドです。このコマンドは、syslogにログ情報を渡すコマンドです。いろいろなパラメータがありますが、ここでは、**-p**オプションだけを使っています。syslogのログ情報には、ログの種別としてファシリティとプライオリティ(優先順位)があります。**-p**オプションでは、そのログ情報のファシリティおよびプライオリティを指定することができます。ファシリティは、日本語での適当な訳がありませんが、ここではカテゴリの意味で良いと思います。

```
$ logger -p mail.info mail-log
```

と入力した場合は、"**mail-log**"というログメッセージをファシリティ＝**mail**、プライオリティ＝**info**を設定したログ情報をsyslogへ渡します。

syslogは、ログ情報のファシリティ、プライオリティに従って出力先(ファイルの場合は、ファイル名になります)を選択し、該当出力先(ここでは、ファイル名：**/var/log/maillog**になります)へログ情報を出力します。

▼図8　同時2接続画面でのやりとりを確認しよう

TeraTerm#1

tail コマンドログファイルをリアルタイムでモニターします。

(1)

```
$ tail -f /var/log/message
…… (略) ……
May 25 07:05:33 local65 root:default-log
```

loggerコマンド発行後、すぐに上記メッセージが出力されることを確認できるでしょう。
CTRL+C でtailコマンドを終了し、続けて別のログファイルを読み込みます

(3)

```
$ tail -f /var/log/secure
…… (略) ……
May 25 07:23:15 local65 root:secure-log
```

loggerコマンド発行後、すぐに上記メッセージが出力されることを確認できるでしょう。
CTRL+C でtailコマンドを終了し、続けて別のログファイルを読み込みます

(5)

```
$ tail -f /var/log/maillog
…… (略) ……
May 25 07:31:22 local65 root:mail-log
```

loggerコマンド発行後、すぐに上記メッセージが出力されることを確認できるでしょう。
CTRL+C でtailコマンドを終了し、続けて別のログファイルを読み込みます

(7)

```
$ tail -f /var/log/cron
…… (略) ……
May 25 07:40:20 local65 root:cron-log
```

loggerコマンド発行後、すぐに上記メッセージが出力されることを確認できるでしょう。
CTRL+C でtailコマンドを終了し、続けて別のログファイルを読み込みます

(9)

```
$ tail -f /var/log/spooler
…… (略) ……
May 25 07:45:11 local65 root:spooler-log
```

loggerコマンド発行後、すぐに上記メッセージが出力されることを確認できるでしょう

```
Message from syslogd@local165 at May 25 ➥
07:48:31...
root:emerg-log
```

loggerコマンド発行後、すぐに上記メッセージが出力されることを確認できるでしょう。これは端末に出力されるのはもちろんのこと、ログファイルにも書き込まれます

TeraTerm#2

(2)
logger コマンドでログ情報をsyslogへ渡してみます。

```
$ logger default-log
```

(4)

```
$ logger -p authpriv.info secure-log
```

(6)

```
$ logger -p mail.info secure-log
```

(8)

```
$ logger -p cron.info cron-log
```

(10)

```
$ logger -p news.crit spooler-log
```

(11)

```
$ logger -p uucp.emerg emerg-log
Message from syslogd@local165 at May 25 ➥
07:48:31...
root:emerg-log
```

loggerコマンド発行後、すぐに上記メッセージが出力されることを確認できるでしょう。このメッセージは緊急時に全端末に送信されるもので、全ログファイルへも保存されます

指定できるファシリティは**表1**のようになります。CentOSでは、デフォルトの設定では11～15の値は、使っていないようです。

指定できるプライオリティは**表2**のようになります。

表1と**表2**のファシリティ名、プライオリティ名を使えることはもちろんですが、数値を使うこともできます。このとき、ファシリティは**表1**の**値2**の列の数値を利用することに注意してください。

```
$ logger -p 16.6 mail-log
```

と入力した場合は、

```
$ logger -p mail.info mail-log
```

と入力した場合と同じ結果を得ます。

また、**-p**オプションを指定しなかった場合は、**user.notice**を指定した場合と同じ動作になります。

```
$ logger default-log
```

と入力した場合は、

```
$ logger -p user.notice default-log
```

▼表2　プライオリティー覧

値	プライオリティ名	概要
0	emerg(panic)	緊急事態(システムが不安定な状態)
1	alert	警報(今すぐ対応が必要な状態)
2	crit	危機的(クリティカル)な状態
3	err(error)	エラー
4	warning(warn)	警告
5	notice	注意(正常だが注意が必要)
6	info	情報
7	debug	デバッグ(問題対処・開発用情報)
16	none	プライオリティなし

※値はRFC 3164[注10]で定義されている値です。また、この値はloggerコマンドで指定できます。
※16：noneは、内部的に使われるものでRFCで定義されているわけではありません。

▼表1　ファシリティー覧

値	値2	ファシリティ名	概要
0	0	kern	カーネルメッセージ
1	8	user	ユーザー・レベル・メッセージ
2	16	mail	メール・システムメッセージ
3	24	daemon	crondおよびrsyslogd以外のシステム・デーモンからのメッセージ
4	32	auth(security)	セキュリティ、認証または認可メッセージ
5	40	syslog	rsyslogdによって内部で生成されたメッセージ
6	48	lpr	ライン・プリンタ・サブシステムメッセージ
7	56	news	ネットワーク・ニュース・サブシステムメッセージ
8	64	uucp	UUCPサブシステムメッセージ
9	72	cron	cronメッセージ
10	80	authpriv	セキュリティ、認証または認可メッセージ(プライベート)
11	88	ftp	FTPシステムメッセージ
12	96	ntp	NTPサブシステムメッセージ
13	104	log audit	セキュリティ、認証に関してOSによっては、4,10,12,14を使い分けることができる。
14	112	log alert	〃
15	120	clock daemon	クロックデーモンに関してOSによっては、9,15を使い分けることができる。
16	128	local0	ローカルで使用(他のアプリケーションからのログで自由に使える)
⋮	⋮	⋮	⋮
23	184	local7	

※値はRFC 3164[注10]で定義されている値です。値2は値の列の数値に2の3乗(2^3)を掛けたものです。また、この値2の数値はloggerコマンドで指定できます。

注10) RFC 3164では、The BSD syslog Protocolが定義、公開されています。

と入力した場合と同じ結果を得ます。

図8に従って行ったコマンドによるログの出力で、**logger**コマンドで指定した**-p**オプションによって、ログの出力先(ここでの例ではファイル)が変わったことに気づかれたと思います。これは、**-p**オプションで指定するファシリティ、プライオリティによって出力先を変えることができるということです。具体的に、rsyslogの設定ファイルで指定、変更することができます。

そこで、次にrsyslog設定ファイルを見てみます。各ログの出力先が、どのように設定されているか確認してみましょう。

▼リスト1　/etc/rsyslog.conf(行番号は説明の都合で付けたもの)

```
01 # rsyslog v5 configuration file
02
03 # For more information see /usr/share/doc/rsyslog-*/rsyslog_conf.html
04 # If you experience problems, see http://www.rsyslog.com/doc/troubleshoot.html
05
06 #### MODULES ####
07
08 $ModLoad imuxsock # provides support for local system logging (e.g. via logger command)
09 $ModLoad imklog   # provides kernel logging support (previously done by rklogd)
10 #$ModLoad immark  # provides --MARK-- message capability
11
12 # Provides UDP syslog reception
13 #$ModLoad imudp
14 #$UDPServerRun 514
15
16 # Provides TCP syslog reception
17 #$ModLoad imtcp
18 #$InputTCPServerRun 514
19
20
21 #### GLOBAL DIRECTIVES ####
22
23 # Use default timestamp format
24 $ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
25
26 # File syncing capability is disabled by default. This feature is usually not required,
27 # not useful and an extreme performance hit
28 #$ActionFileEnableSync on
29
30 # Include all config files in /etc/rsyslog.d/
31 $IncludeConfig /etc/rsyslog.d/*.conf
32
33
34 #### RULES ####
35
36 # Log all kernel messages to the console.
37 # Logging much else clutters up the screen.
38 #kern.*                                                 /dev/console
39
40 # Log anything (except mail) of level info or higher.
41 # Don't log private authentication messages!
42 *.info;mail.none;authpriv.none;cron.none                /var/log/messages
43
44 # The authpriv file has restricted access.
45 authpriv.*                                              /var/log/secure
46
47 # Log all the mail messages in one place.
```

```
48 mail.*                                                  -/var/log/maillog
49
50
51 # Log cron stuff
52 cron.*                                                  /var/log/cron
53
54 # Everybody gets emergency messages
55 *.emerg                                                 *
56
57 # Save news errors of level crit and higher in a special file.
58 uucp,news.crit                                         /var/log/spooler
59
60 # Save boot messages also to boot.log
61 local7.*                                               /var/log/boot.log
62
63
64 # ### begin forwarding rule ###
65 # The statement between the begin ... end define a SINGLE forwarding
66 # rule. They belong together, do NOT split them. If you create multiple
67 # forwarding rules, duplicate the whole block!
68 # Remote Logging (we use TCP for reliable delivery)
69 #
70 # An on-disk queue is created for this action. If the remote host is
71 # down, messages are spooled to disk and sent when it is up again.
72 #$WorkDirectory /var/lib/rsyslog # where to place spool files
73 #$ActionQueueFileName fwdRule1 # unique name prefix for spool files
74 #$ActionQueueMaxDiskSpace 1g   # 1gb space limit (use as much as possible)
75 #$ActionQueueSaveOnShutdown on # save messages to disk on shutdown
76 #$ActionQueueType LinkedList   # run asynchronously
77 #$ActionResumeRetryCount -1    # infinite retries if host is down
78 # remote host is: name/ip:port, e.g. 192.168.0.1:514, port optional
79 #*.* @@remote-host:514
80 # ### end of the forwarding rule ###
```

syslogの設定を確認してみよう

早速、rsyslog設定ファイルを見てみましょう。リスト1がrsyslogの設定ファイル(**/etc/rsyslog.conf**)です。

各行の役割と解説

これだけの行数がありますがコメント(“#”文字列以降はコメントになります)を除けば、実質の行数は数行です。

次に簡単に各設定について解説します。

8行目：ロードするモジュールを指定しています。ここでは、**imuxsock**を指定しています。実際にロードされるファイルは、**/lib/rsyslog/imuxsock.so**です。このモジュールは、ログファイルへの書き込み、管理を行うものです。

9行目：ロードするモジュールを指定しています。ここでは、**imklog**を指定しています。実際にロードされるファイルは、**/lib/rsyslog/imklog.so**です。このモジュールは、カーネルログをサポートするためのモジュールです。p.24で解説したsysklogdパッケージのklogd(カーネルログデーモン)の代替になります。

24行目：ログ情報の出力フォーマットを指定しています。テンプレートとして次のものが用意されています。また、各テンプレートを使った時の**logger -p mail.info mail-log**実行時の出力イメージを記載しておきます。

・RSYSLOG_TraditionalFileFormat

```
May 25 09:29:25 local65 root: mail-log
```

・RSYSLOG_FileFormat

```
2014-05-25T10:27:45.673814+09:00 local65 ➥
root: mail-log
```

・RSYSLOG_TraditionalForwardFormat

```
<22>May 25 10:28:15 local65 root: mail-log
```

・RSYSLOG_SysklogdFileFormat

```
May 25 10:28:30 local65 root: mail-log
```

・RSYSLOG_ForwardFormat

```
<22>2014-05-25T10:28:43.940408+09:00 ↴
local65 root: mail-log
```

・RSYSLOG_SyslogProtocol23Format

```
<22>1 2014-05-25T10:29:00.574099+09:00 ↴
local65 root  - -  mail-log
```

・RSYSLOG_DebugFormat

```
Debug line with all properties:
FROMHOST: 'local65', fromhost-ip: ➥
'127.0.0.1', HOSTNAME: 'local65', PRI: 22,
syslogtag 'root:', programname: 'root', ➥
APP-NAME: 'root', PROCID: '', MSGID: '
 -',
TIMESTAMP: 'May 25 10:29:15', ➥
STRUCTURED-DATA: '-',
msg: ' mail-log'
escaped msg: ' mail-log'
inputname: imuxsock rawmsg: '<22>May 25 ➥
10:29:15 root: mail-log'
```

31行目：ロードする個別設定ファイルを指定しています。具体的に**/etc/rsyslog.d/**配下の**.conf**の拡張子を持つファイルをすべて読み込みます。

42行目：指定されたファシリティ、プライオリティのログの出力先を指定しています。ここでは、ログ情報が***.info;mail.none;authpriv.none;cron.none**のいずれかに該当する場合、**/var/log/messages**へ出力します。

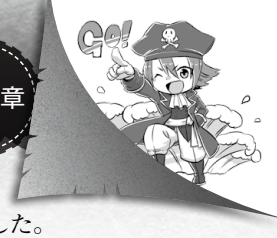
ここで注意してほしいのは、ここで設定しているように、ワイルドカード“*****”が使える点、“**;**”区切りで複数指定できる点です。“***.info**”は、すべてのファシリティ＋**info**プライオリティの意味です。また、さらに注意しておきたい点は、ワイルドカードを使った場合、ログ情報によっては、複数の出力先が該当する場合がある点です。たとえば“**mail.info**”の場合は、出力先としてこの42行目と48行目がそれぞれ該当します。この場合の出力先は、両方になります。

45行目：指定されたファシリティ、プライオリティのログの出力先を指定しています。ここでは、ログ情報が**authpriv.***に該当する場合、**/var/log/secure**へ出力します。

48行目：指定されたファシリティ、プライオリティのログの出力先を指定しています。ここでは、ログ情報が**mail.***に該当する場合、**/var/log/maillog**へ出力します。ただし、ファイル名の先頭に“**-**”（マイナス）記号がついていることに注意してください。これは、ファイルの同期処理（fsync[注10]）を省略するという意味です。少し語弊がありますが、ファイルにちゃんと書き込めたか確認しないということです。そうすることで、syslogでの負荷を軽減させることができるとされています。とくにメールのログは、スパムを含めて負荷が高くなる可能性が大きいためデフォルトとして“**-**”を付加していることが多いです。

52行目：指定されたファシリティ、プライオリティのログの出力先を指定しています。ここでは、ログ情報が**cron.***に該当する場合、**/var/log/cron**へ出力します。

注10) fsyncは、Unix系システムのファイル関連のシステムコールの1つで、メモリ上にあるファイルの内容をストレージデバイス上のものと同期させるために使用するものです。ここでの解説では、同期処理をしないということは、このfsyncをコールしないという意味で使われています。



55行目：指定されたファシリティ、プライオリティのログの出力先を指定しています。ここでは、ログ情報が***.emerg**に該当する場合、全ファイル・全端末へ出力します。

58行目：指定されたファシリティ、プライオリティのログの出力先を指定しています。ここでは、ログ情報が**uucp,news.crit**に該当する場合、**/var/log/spooler**へ出力します。

61行目：指定されたファシリティ、プライオリティのログの出力先を指定しています。ここでは、ログ情報が**local7.***に該当する場合、**/var/log/boot.log**へ出力します。

設定ファイルの編集方法

リスト1の設定ファイルは、テキストファイルですので、**vi**や**nano**コマンドでTeraTermから簡単に編集できます。**vi**や**nano**がインストールされていない場合は、**yum install nano**のようにインストールしましょう。また、先の設定ファイルを編集した際は、必ず、rsyslogを再起動します。

```
$ /etc/init.d/rsyslog restart
システムロガーを停止中:                [  OK  ]
システムロガーを起動中:                [  OK  ]
```

ここまでsyslogの大きな機能、ログ情報の流れや出力先の設定方法など基本的な解説を行ってきました。また、**logger**コマンドを使って実際にログ情報を出力もしてみました。

ログから必要な情報を見つけよう

本章の最後の解説は、「ログファイルから必要なログ情報をどうやって取り出すか？」です。ログファイルは、随時、追加更新されていますから、ログの種類（ファシリティ、プライオリティの出力先）によっては膨大な量になっていることがあります。

一般的に、何か問題があってログ情報を見たいと思った時、その膨大な量のログファイルを開いて逐一見ることは、ほとんどありません。そのような場合は、ログファイルから必要なデータを抜き出して（絞り込んで）、絞り込んだ情報から、本当に必要な情報を確認していきます。

ログファイルの中から取り出したいログ情報をコマンドで取り出すには、p.28で解説した**grep**コマンドが有効です。たとえば、一番よくあるのは日付によって情報を絞り込む方法です。問題が発生した日時がわかっている場合は、その日時に、どんなログが出力されているか確認したいと思うでしょう。その際、**図9**のように**grep**コマンドを使って必要な情報を取り出すことができます。

catコマンドは、指定したファイルの内容を全出力するコマンドです。

続けて"|"ですから、**grep**コマンドが実行されます。**grep**コマンドでは、**'May 25 09'**の文字列のある行を検索し出力します。これで、

▼図9　ログファイルにgrepコマンドを使用する例

```
$ cat /var/log/messages|grep 'May 25 09'
May 25 09:02:31 local65 root: test
May 25 09:02:45 local65 root: all
May 25 09:02:49 local65 root: lock
May 25 09:26:31 local65 root: mail-log
May 25 09:27:06 local65 kernel: Kernel logging (proc) stopped.
May 25 09:27:06 local65 rsyslogd: [origin software="rsyslogd" swVersion="5.8.1 0" ➥
x-pid="1643" x-info="http://www.rsyslog.com"] exiting on signal 15.
May 25 09:27:06 local65 kernel: imklog 5.8.10, log source = /proc/kmsg started.
May 25 09:27:06 local65 rsyslogd: [origin software="rsyslogd" swVersion="5.8.1 0" ➥
x-pid="1675" x-info="http://www.rsyslog.com"] start
May 25 09:27:12 local65 root: mail-log
May 25 09:28:36 local65 root: mail-log
```

▼図10　grepの文字列処理の例

```
$ cat /var/log/messagesgrep 'signal ¥+[0-9]¥+¥.$'
May 25 08:31:40 local65 rsyslogd: [origin software="rsyslogd" swVersion="5.8.1 0" ➥
x-pid="797" x-info="http://www.rsyslog.com"] exiting on signal 15.
May 25 08:33:05 local65 rsyslogd: [origin software="rsyslogd" swVersion="5.8.1 0" ➥
x-pid="1626" x-info="http://www.rsyslog.com"] exiting on signal 15.
May 25 09:27:06 local65 rsyslogd: [origin software="rsyslogd" swVersion="5.8.1 0" ➥
x-pid="1643" x-info="http://www.rsyslog.com"] exiting on signal 15.
May 25 10:27:36 local65 rsyslogd: [origin software="rsyslogd" swVersion="5.8.1 0" ➥
x-pid="1675" x-info="http://www.rsyslog.com"] exiting on signal 15.
2014-05-25T10:28:13.233089+09:00 local65 rsyslogd: [origin software="rsyslogd" ➥
swVersion="5.8.10" x-pid="1717" x-info="http://www.rsyslog.com"] exiting on signal 15.
May 25 10:28:42 local65 rsyslogd: [origin software="rsyslogd" swVersion="5.8.1 0" ➥
x-pid="1748" x-info="http://www.rsyslog.com"] exiting on signal 15.
<46>1 2014-05-25T10:29:13.731474+09:00 local65 rsyslogd  - -  [origin software ="rsyslogd" ➥
swVersion="5.8.10" x-pid="1778" x-info="http://www.rsyslog.com"] exiting on signal 15.
```

5月25日9時台のログ情報がすべて出力されたことになります。

ヒント 実は**grep**コマンドは、ファイルをそのまま指定できます。たとえば上記のコマンド

```
$ cat /var/log/messages|grep 'May 25 09'
```

は、

```
$ grep 'May 25 09' /var/log/messages
```

と指定した場合と同じ結果を得ます。

ただ、**grep**コマンドは、先にも書いたようにさまざまな用途で使用しますから、ほかのコマンドと組み合わせて使用することが多いです。そのため、ここでは、あえて"|"（パイプ）を使って**grep**コマンドを使っています。

grepコマンドは非常に便利で、正規表現も使えますから何かログ情報の中で検索したい文字列のパターンがあれば、そのパターンで抜き出すこともできます（**図10**）。

図10で指定している

```
'signal \+[0-9]\+\.$'
```

は、暗号みたいですが、これが正規表現です。

この正規表現は、

```
'signal'    :'signal'という文字列がある
' \+'       :半角スペースが1つ以上ある
'[0-9]\+'   :数字が1つ以上ある
'\.'        :'.'という文字列
'$'         :文末である
```

これらの条件がマッチする行を検索しています（各正規表現の記号の意味は**表3**を参照してください）。**図10**の出力例では、文末がすべて**'signal 15.'**になっていることがわかるかと思います。

正規表現を使うと、いろんなパターンで情報を検索、取り出すことができます。また、正規表現は、ほかのコマンドやソフトウェアでも利用できる場合が多いです。通常の検索では数回の検索が必要なことも、正規表現を使えば1回で済むこともよくあります。作業効率の面からも、ぜひマスターしておきたいところです。SD

▼表3　grepコマンドで利用可能な正規表現の代表的なもの

記号	意味
.	改行文字以外の任意の文字列（1文字）
*	直前の1文字の0回以上の繰り返しに一致
^	行先頭
$	行末尾
[ ]	かっこ内の任意の1文字に一致
[^]	かっこ内の任意の1文字に不一致
\+	直前の文字の1個以上の繰り返しに一致
\?	直前の文字の0または1文字に一致
\{n\}	直前の文字のn個の繰り返しに一致
\{n,\}	直前の文字のn個以上の繰り返しに一致
\{,m\}	直前の文字のm個以下の繰り返しに一致
\{n,m\}	直前の文字のn個以上、m個以下の繰り返しに一致

第2章 Webサーバのログを見てみよう

Writer 近藤 成(こんどう じょう)
Mail jj2kon@gmail.com Web http://server-setting.info/

前章では、syslogを含めてログの基本的な事項について解説してきました。この章では、より具体的なWebサーバ(ApacheとNginx)のログについて解説します。まずログの種類を説明します。そしてWebalizerを使用した解析方法を解説します。これでログの見方がより具体的なものになっていきます。

Webサーバのログには種類がある

Webサーバには、**アクセスログ**と**エラーログ**があります。**アクセスログ**は、文字どおり、アクセスした時のログ情報です。もちろん、要求されたページを正常(HTTPのステータスコードでOK(200))に返信した際にも出力されますが、HTTPのステータスコードで「Not Found(404)」などに代表されるエラーステータスを返信した場合もアクセスログに出力されます。たとえば、404であれば、「XXXのページを要求され404を返信した」というログ情報を出力するもので、Webサーバでなぜ要求されたページが見つからなかったかというログ情報ではないことに注意してください(HTTPステータス

▼表1　HTTPステータスコード一覧(RFC 2616[注1]による定義一覧)

ステータスコード	概要
100	継続：Continue
101	プロトコル切替え：Switching Protocols
200	OK：OK
201	作成：Created
202	受理：Accepted
203	信頼できない情報：Non-Authoritative Information
204	内容なし：No Content
205	内容のリセット：Reset Content
206	部分的内容：Partial Content
300	複数の選択：Multiple Choices
301	恒久的に移動した：Moved Permanently
302	発見した：Found
303	他を参照せよ：See Other
304	未更新：Not Modified
305	プロキシを使用せよ：Use Proxy
307	一時的リダイレクト：Temporary Redirect
400	リクエストが不正である：Bad Request
401	認証が必要である：Unauthorized
402	支払いが必要である：Payment Required
403	禁止されている：Forbidden
404	未検出：Not Found
405	許可されていないメソッド：Method Not Allowed
406	受理できない：Not Acceptable
407	プロキシ認証が必要である：Proxy Authentication Required
408	リクエストタイムアウト：Request Time-out
409	矛盾：Conflict
410	消滅した：Gone
411	長さが必要：Length Required
412	前提条件で失敗した：Precondition Failed
413	リクエストエンティティが大きすぎる：Request Entity Too Large
414	リクエストURIが大きすぎる：Request-URI Too Large
415	サポートしていないメディアタイプ：Unsupported Media Type
416	リクエストしたレンジは範囲外にある：Requested range not satisfiable
417	期待するヘッダに失敗：Expectation Failed
500	サーバ内部エラー：Internal Server Error
501	実装されていない：Not Implemented
502	不正なゲートウェイ：Bad Gateway
503	サービス利用不可:Service Unavailable
504	ゲートウェイタイムアウト：Gateway Time-out
505	サポートしていないHTTPバージョン：HTTP Version not supported

注1) RFC 2616では、HTTP/1.1のプロトコルが定義、公開されています。RFC2068の改訂版とされています。
英名：Hypertext Transfer Protocol -- HTTP/1.1

コードの詳細につては、**表1**を参照してください)。

エラーログは、404などに代表されるHTTPのエラーステータスを返信した旨のログ情報でなく、Webサーバで何らかのエラーが発生した場合に出力されます。たとえば、先に書いたように404(Not Found)を返信したログ情報はアクセスログに出力されます。その際、なぜ発生したかの原因がエラーログに出力されることがあります。

静的なページであれば、エラーログには、ファイルが存在しない旨のエラー情報が出力されることになります。また、動的なページであれば、その要因はプログラムのエラーかもしれませんが、ブログツールで有名なWordPressを含め一般的なCMSであれば、要求されたページ情報がデータベースになかっただけなのでエラーログには何も出力されないでしょう。

ヒント

静的なページとは、少し語弊がありますが、URLに対してHTMLで記述されたファイルが1対1で存在するページのことです。動的なページとは、PerlやPHPなどのスクリプトを含めプログラム言語を使用したソフトウェアが要求ページごとに実行され、自動的にHTMLでページを作成、出力するページのことです。このとき、MySQLなどのデータベースを利用する場合が多いです。静的なページは、常に同じアドレスで同じページが表示されるのに対して、動的なページは、アクセスするユーザや時間などさまざまな要因によってページを変動させることができ、自在なページ表現ができる特徴があります。

このように404が出力されたからといってエラーログに出力されるとは限りません。あくまでエラーログはWebサーバがエラーを検出した際に出力されるものであって、HTTPのエラーステータスを返信したことと同義ではないことに注意してください。

ログを出力してみよう(Apache編)

実際にApacheを使ってログを出力してみましょう。CentOSでは、Apacheをhttpdというパッケージ名で提供しています(アプリケーション名もhttpdです)。以降httpdと表記した場合はApacheとします。

まずは、httpd(Apache)をインストールします(バージョンは2.2.15[2.2系])。

```
$ yum install httpd
……(省略)……
Is this ok [y/N]: y
……(省略)……
Complete!
```

インストールを終えたら、起動しておきましょう。

```
$ /etc/init.d/httpd start
httpd を起動中:                            [  OK  ]
```

Apacheの設定ファイルを確認しておきましょう

Apacheの基本設定は、**/etc/httpd/conf/httpd.conf**を編集します。また、デフォルトサイトの設定は、**/etc/httpd/conf.d/welcome.conf**を編集します。

Apacheの基本設定ファイルの447行目付近から**リスト1**のようにログに関する設定を行っています(**リスト1**は、ログに関する設定だけを抜粋しています)。

また、**リスト1**に抜粋していない情報でおさえておきたいものが、サーバのルートディレクトリです。これは基本設定の57行目付近にディレクティブ:**ServerRoot**で設定されています。

```
ServerRoot "/etc/httpd"
```

パスを指定する他のディレクティブでフルパスを指定していない場合は、このルートディレクトリの配下と認識されます。

以降**リスト1**のログ設定について解説します。

ヒント

Webサーバでは、とくに設定ファイルの設定項目(キー情報)をディレクティブと言います。以降で解説するNginxでも設定ファイルの設定項目(キー情報)をディレクティブと言います。英語のdirectiveの"指示"、"命令"などの意味から使われています。

エラーログの設定

ディレクティブ：**ErrorLog**を使用します。次は**リスト1**の例です。

```
Errorlog log/error_log
```

実際のログファイルのパスは、先の**Server Root**の設定＋ここでの設定パスになります。つまり、**/etc/httpd/logs/error_log**になります。

また、関連したディレクティブに、**LogLevel**があります。

```
LogLevel warn
```

ここで、エラーログへ出力するログレベルを指定します。ここの例では警告(Warning)以上の場合にエラーログへ出力されることになります。ここで指定できるログレベルを**表2**に一覧で載せています。その**表2**では上からログレベルが高い順になっています。ここの例でいうと、

▼リスト1　/etc/httpd/conf/httpd.conf(ログ関連のみ抜粋)

```
#
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host will be
# logged here.  If you *do* define an error logfile for a <VirtualHost>
# container, that host's errors will be logged there and not here.
#
ErrorLog logs/error_log

#
# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
#
LogLevel warn

#
# The following directives define some format nicknames for use with
# a CustomLog directive (see below).
#
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined ←図1で詳細解説
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

# "combinedio" includes actual counts of actual bytes received (%I) and sent (%O); this
# requires the mod_logio module to be loaded.
#LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedio

#
# The location and format of the access logfile (Common Logfile Format).
# If you do not define any access logfiles within a <VirtualHost>
# container, they will be logged here.  Contrariwise, if you *do*
# define per-<VirtualHost> access logfiles, transactions will be
# logged therein and *not* in this file.
#
#CustomLog logs/access_log common

#
# If you would like to have separate agent and referer logfiles, uncomment
# the following directives.
#
#CustomLog logs/referer_log referer
#CustomLog logs/agent_log agent

#
# For a single logfile with access, agent, and referer information
# (Combined Logfile Format), use the following directive:
#
CustomLog logs/access_log combined
```

warn以上ですから、warn、error、crit、alert、emergが出力対象となります。

アクセスログの設定

ディレクティブ：CustomLogを使用します。次は**リスト1**の例です。

```
CustomLog logs/access_log combined
```

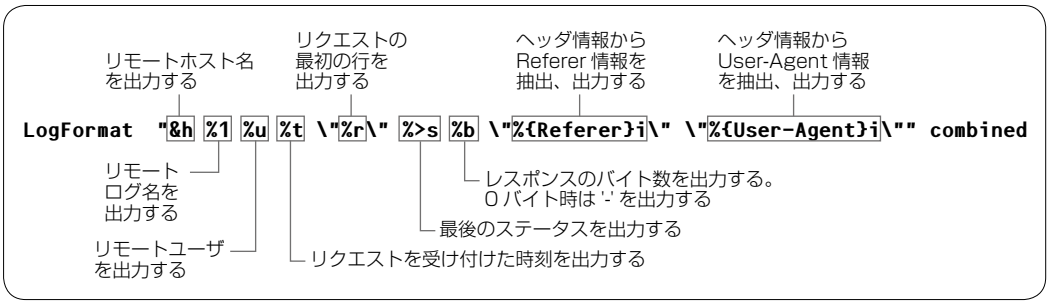
実際のログファイルのパスは、先の**ServerRootの設定＋ここでの設定パス**になります。つまり、/etc/httpd/logs/access_logになります。フォーマットは、combinedを使用します。combinedは、**図1**のようにディレクティブ：LogFormatで定義されています。各パラメータはWebに公開されている書式[注1]に従ったものです。

注1）Apacheのアクセスログで使用できる書式一覧（http://httpd.apache.org/docs/2.2/ja/mod/mod_log_config.html#formats）

▼表2　Apacheのエラーログで指定できるログレベル一覧（http://httpd.apache.org/docs/2.2/ja/mod/core.html#loglevel）

レベル	説明	例
emerg	緊急 - システムが利用できない	Child cannot open lock file. Exiting（子プロセスがロックファイルを開けないため終了した）
alert	直ちに対処が必要	getpwuid: couldn't determine user name from uid（getpwuid: UID からユーザ名を特定できなかった）
crit	致命的な状態	socket: Failed to get a socket, exiting child（socket: ソケットが得られないため、子プロセスを終了させた）
error	エラー	Premature end of script headers（スクリプトのヘッダが足りないままで終わった）
warn	警告	child process 1234 did not exit, sending another SIGHUP（子プロセス 1234 が終了しなかった。もう一度 SIGHUP を送る）
notice	普通だが、重要な情報	httpd: caught SIGBUS, attempting to dump core in ...（httpd: SIGBUS シグナルを受け、...へコアダンプをした）
info	追加情報	“Server seems busy,（you may need to increase StartServers, or Min/MaxSpareServers）...”（「サーバは負荷が高い、（StartServers や Min/MaxSpare Servers の値を増やす必要があるかも）」）
debug	デバッグメッセージ	“Opening config file ...”（設定ファイルを開いている ...）

▼図1　アクセスログcombinedのLogFormat



▼図2　アクセスログの出力先

```
$ ls -l /etc/httpd/
合計 8
drwxr-xr-x. 2 root root 4096  5月 25 14:49 2014 conf
drwxr-xr-x. 2 root root 4096  5月 25 13:11 2014 conf.d
lrwxrwxrwx. 1 root root   19  5月 25 13:04 2014 logs -> ../../var/log/httpd
lrwxrwxrwx. 1 root root   27  5月 25 13:04 2014 modules -> ../../usr/lib/httpd/modules
lrwxrwxrwx. 1 root root   19  5月 25 13:04 2014 run -> ../../var/run/httpd
……（省略）……
```

ヒント

CentOS＋Apacheのログ出力先は、上記の設定では、**/etc/httpd/logs**配下となっていますが、実際に出力されるのは、**/var/log/httpd**の配下となります。それは、単純に**/etc/httpd/logs→/var/log/httpd**へのシンボリックリンクとなっているためです。**図2**のようにコマンド**ls -l**でシンボリックリンクを確認できます。

Webサイトへアクセスしてログを出力してみよう

Apacheが起動していない場合は、手動で起動しておきます。

```
$ /etc/init.d/httpd start
httpd を起動中:          [  OK  ]
```

先の設定を確認できたら、URLにCentOSのIPアドレスを、直接指定(例：**http://192.168.1.65**)してWebブラウザからサイトへアクセスしてみてください。**図3**のようにデフォルトページが見えればOKです。

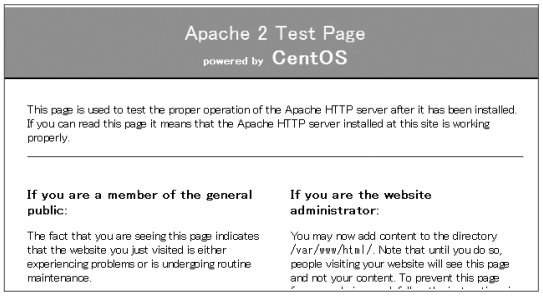
ヒント

SSHでログインできているのにWebブラウザからアクセスしてもデフォルトページを表示できない場合は、iptablesの設定でhttpのポートが規制されているかもしれません。

リスト2はiptablesのデフォルト設定です。SSHが接続できているので、SSH(22番)ポートの設定があるはずです。**リスト2**の9行目がその設定になります。その設定をまねて10行目のようにhttp(80番)ポートの設定を追記します。

この時、必ずSSH(22番)ポートの設定の後に挿入します。順番を間違えるとポートを開くことができないことがありますので注意してください。編集を終えたらファイルを保存し、**図4**のようにiptablesを再起動します。

▼図3　Apacheデフォルトページ



▼図4　iptablesの再起動

```
$ /etc/rc.d/init.d/iptables restart
iptables: チェインをポリシー ACCEPT へ設定中filter       [  OK  ]
iptables: ファイアウォールルールを消去中:                 [  OK  ]
iptables: モジュールを取り外し中:                         [  OK  ]
iptables: ファイアウォールルールを適用中:                 [  OK  ]
```

▼リスト2　/etc/sysconfig/iptables

```
01 # Generated by iptables-save v1.4.7 on Sun May 25 14:07:05 2014
02 *filter
03 :INPUT ACCEPT [0:0]
04 :FORWARD ACCEPT [0:0]
05 :OUTPUT ACCEPT [32:3832]
06 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
07 -A INPUT -p icmp -j ACCEPT
08 -A INPUT -i lo -j ACCEPT
09 -A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
10 -A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT  ←この行を追加
11 -A INPUT -j REJECT --reject-with icmp-host-prohibited
12 -A FORWARD -j REJECT --reject-with icmp-host-prohibited
13 COMMIT
14 # Completed on Sun May 25 14:07:05 2014
```

tailコマンドを使ってログを確認してみよう

図3のようなページが見えていれば、アクセスログ(/var/log/httpd/access_log)が出力されているはずです。tailコマンドで、最新のアクセスログを出力してみます(図5)。

403のエラーステータスが出力されていますが、とりあえず脇に置いて(後で解説します)、続けて、エラーログ(/var/log/httpd/error_log)も確認してみましょう。同じようにtailコマンドを使って、最新のエラーログを出力してみます(図6)。

エラーを詳しく見てみよう

エラーログに何かエラーが出力されていますね。詳しく見てみましょう。まず、日時を確認してみると、ちょうど先のアクセスログで403が出力された日時と同じです。このエラー情報の英語部分を直訳すれば「Optionsにて/var/www/html/のディレクトリ一覧の出力は規制されています」という感じでしょうか。つまり、このエラーのために、アクセスログで規制されている旨のステータスコードである403が出力されていたことが、なんとなく想像できますね。

エラーを回避してみよう

では、このエラーをどうしたら回避できるでしょうか。それは、このエラーが何を意味しているかを、もう少し深く理解する必要があります。勘の良い方はピンと来ているでしょう。

まず、デフォルトサイトの設定(/etc/httpd/conf.d/welcome.conf)を確認します(リスト3)。

Optionsディレクティブで次のように設定されています。

```
Options -Indexes
```

この設定は、URLにファイル名が省略され

▼図5　tailコマンドで最新のアクセスログを出力

```
$ tail /var/log/httpd/access_log
……(省略)……
192.168.1.33 - - [25/May/2014:16:31:28 +0900] "GET / HTTP/1.1" 403 5039 "-" "Mozilla/5.0 ➥
(Windows NT 6.2; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0"
192.168.1.33 - - [25/May/2014:16:31:28 +0900] "GET /icons/apache_pb.gif HTTP/1.1" 200 2326 ➥
"http://192.168.1.65/" "Mozilla/5.0 (Windows NT 6.2; WOW64; rv:24.0) Gecko/20100101 ➥
Firefox/24.0"
192.168.1.33 - - [25/May/2014:16:31:28 +0900] "GET /icons/poweredby.png HTTP/1 .1" 200 ➥
3956 "http://192.168.1.65/" "Mozilla/5.0 (Windows NT 6.2; WOW64; rv:24¥.0) Gecko/20100101 ➥
Firefox/24.0"
```

▼図6　tailコマンドで確認

```
$ tail /var/log/httpd/error_log
……(省略)……
[Sun May 25 16:31:28 2014] [error] [client 192.168.1.33] Directory index forbidden by ➥
Options directive: /var/www/html/
```

▼リスト3　/etc/httpd/conf.d/welcome.conf

```
#
# This configuration file enables the default "Welcome"
# page if there is no default index page present for
# the root URL.  To disable the Welcome page, comment
# out all the lines below.
#
<LocationMatch "^/+$">
Options -Indexes
ErrorDocument 403 /error/noindex.html
</LocationMatch>
```

た場合「もし表示するものが何もなかったとしてもディレクトリ一覧は表示しない」という意味です。エラーログの内容と一致しますね。

さらに、ErrorDocumentディレクティブで「403が発生したら**/error/noindex.html**を出力しなさい」と設定されています。大方は、この**noindex.html**が、先に表示されたデフォルトページだったであろうことに思い至るでしょう。

ここで整理しておきます。まず、IPアドレスだけでアクセスしてみました。つまり、URLにファイル名を指定していません(たとえば、**http://192.168.1.65/index.html**のように**index.html**を指定しなかったということ)から、Apacheでは、あらかじめ設定されている省略時のデフォルトファイルを探します。そのデフォルトファイルは、DirectoryIndexディレクティブで設定します。先のデフォルトサイトの設定(**リスト3**)には、そのDirectoryIndexディレクティブの指定がありませんでしたから、基本設定ファイル(**/etc/httpd/conf/httpd.conf**)の設定内容を引き継ぐことになります。基本設定ファイルを確認してみます。**図7**のように**cat**と**grep**コマンドで検索してみましょう。

ありました。Apacheはファイル名を指定せずにアクセスした場合は、**index.html**か**index.html.var**を探しに行くようになっています。

さて、では、どこのディレクトリの**index.html**か**index.html.var**を探しに行くのでしょう？　これは、先のアクセスがIPアドレスだけを指定したので、デフォルトサイトのドキュメントルートディレクトリはどこか？——というのと同義です。ドキュメントルートディレクトリは、DocumentRootディレクティブで設定します。これは、デフォルトサイトの設定(**リスト3**)にはありませんでしたから、これも先と同様に基本設定ファイル(**/etc/httpd/conf/httpd.conf**)の設定内容を引き継ぐことになります。先と同様、**図8**のように**cat**と**grep**コマンドで検索してみましょう。

ありました。DocumentRootディレクティブには、**/var/www/html**と指定してありますから、ここがドキュメントルートディレクトリということになります。これもエラーログの内容と一致しますね。

さて、ここまで来ると、

> “**/var/www/html**に表示すべきファイル(**index.html**か**index.html.var**)が存在しなかったので、ディレクトリ一覧を出力しようとしたが、規制されていたのでエラー403を出力した。403が発生した飛び先は、**/error/noindex.html**と指定があるのでそのファイルを返信した。”

ということに考えが至るでしょう。では、この考えの裏付けとして、ドキュメントルートディレクトリ(**/var/www/html**)に本当に**index.html**か**index.html.var**が存在しないか、**ls**コマンドで確認してみましょう。

```
$ ls /var/www/html
$
```

▼図7　catとgrepで検索

```
$ cat /etc/httpd/conf/httpd.conf|grep DirectoryIndex
# DirectoryIndex: sets the file that Apache will serve if a directory
DirectoryIndex index.html index.html.var
```

▼図8　catとgrepで検索

```
$ cat /etc/httpd/conf/httpd.conf|grep DocumentRoot
# DocumentRoot: The directory out of which you will serve your
DocumentRoot "/var/www/html"
# This should be changed to whatever you set DocumentRoot to.
#    DocumentRoot /www/docs/dummy-host.example.com
```

何も出力されません。つまり、ここにindex.htmlがあれば、このエラーは回避できそうです。簡単なindex.htmlを作成してみます。

```
$ echo 'test' > /var/www/html/index.html
$ ls /var/www/html
index.html
```

ここでは、echoコマンド(Windowsのechoコマンドと同じで単純に指定された文字列を画面出力します)を使ってファイルへ">"リダイレクトして、"test"という文字列だけのテキストファイル(/var/www/html/index.html)を作成しています。

これで、再度、同じようにWebブラウザからIPアドレスだけを指定してアクセスしてみましょう。ブラウザには"test"の文字だけが表示されるはずです。

ブラウザに表示されたということは、アクセスログに何か出力されているに違いありません。tailコマンドでアクセスログを確認してみましょう。

```
$ tail /var/log/httpd/access_log
……(省略)……
192.168.1.33 - - [25/May/2014:17:10:45 +0900] "GET / HTTP/1.1" 200 5 "-" "Mozilla/5.0 (Windows NT 6.2; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0"
```

今度は、正常なレスポンス200が返信されています。

続けて、同じようにtailコマンドでエラーログを確認してみましょう。

```
$ tail /var/log/httpd/error_log
……(省略)……
```

アクセスログと同じ時間にエラーが出力されていないことが確認できたかと思います。

このようにアクセスログ、エラーログでWebサーバの問題点を見つけ出し、修正、改善を図ることができます。

ログを出力してみよう(Nginx編)

次に、人気のWebサーバ「Nginx」を使って同じようにログを出力してみましょう。

NginxはCentOSにパッケージがありません。そのため、Nginxの公式サイトからインストールするためにリポジトリの設定から行います。

Nginx(バージョン1.6.0)をインストールしよう

①/etc/yum.repos.d/nginx.repoの編集

デフォルトではファイルが存在しないので作成します。

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/
centos/6/$basearch/
gpgcheck=0
enabled=1
```

②yumコマンドでインストール

画面の指示に従いインストールします。

```
$ yum install nginx
……(省略)……
Is this ok [y/N]: y
……(省略)……
Complete!
```

③Nginxの起動

インストールを終えたら、起動しておきましょう。

```
$ /etc/init.d/nginx start
nginx を起動中:                            [  OK  ]
```

Nginxの設定ファイルを確認しておきましょう

Nginxの基本設定は、/etc/nginx/nginx.confを編集します。また、デフォルトサイトの設定は、/etc/nginx/conf.d/default.confを編集します。今回は、ログ以外の話は少し脇に置いて話を進めます。Nginxの基本設

定(/etc/nginx/nginx.conf)は、リスト4のようになっています(ログに関する設定は、設定ファイルの先頭部分にありますので、該当部分を抜粋しています)。

各ログの設定について簡単に解説します。

エラーログの設定

ディレクティブ:error_logを使用します。次はリスト4の例(5行目)です。

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

warn(警告)以上のエラー情報を/var/log/nginx/error.logへ出力します。

ログレベルとして、ここでは警告(warn)を指定していますが、ほかに[debug | info | notice | warn | error | crit | alert | emerg](左からログレベル低→高の順です)のいずれかを指定できます。この値はApacheの表2と同じです。

ここでの例ではwarn以上ですから、warn、error、crit、alert、emergが出力対象です。

アクセスログの設定

ディレクティブ:access_logを使用します。次はリスト4の例(22行目)です。

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

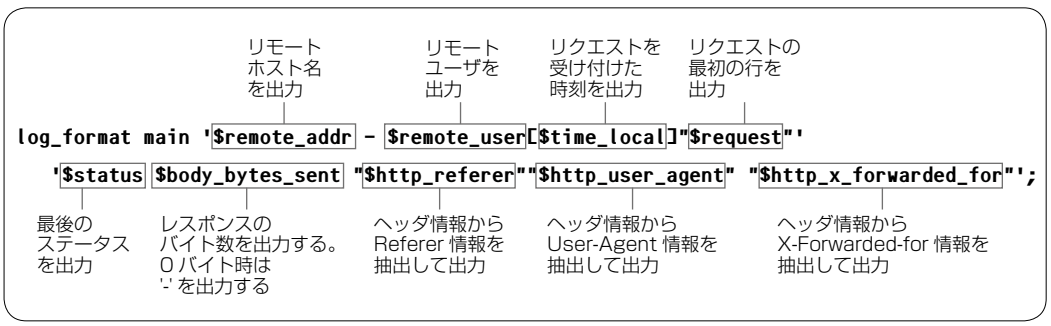
フォーマットmainの定義に従い、/var/log/nginx/access.logへ出力します。フォーマットmainは、ディレクティブlog_format

▼リスト4　/etc/nginx/nginx.conf(抜粋)

```
01
02 user  nginx;
03 worker_processes  1;
04
05 error_log  /var/log/nginx/error.log warn;
06 pid        /var/run/nginx.pid;
07
08
09 events {
10     worker_connections  1024;
11 }
12
13
14 http {
15     include       /etc/nginx/mime.types;
16     default_type  application/octet-stream;
17
18     log_format  main  '$remote_addr - $remote_user [$time_local] "$request"¥'
19                       '$status $body_bytes_sent "$http_referer" '
20                       '"$http_user_agent" "$http_x_forwarded_for"';
21
22     access_log  /var/log/nginx/access.log  main;
23 ……(省略)……
```

←図9で詳細解説している

▼図9　ディレクティブlog_formatの定義



▼表3　Nginxのアクセスログで使用できる書式一覧（http://nginx.org/en/docs/http/ngx_http_core_module.html）：筆者による和訳です。機械的な訳ですがご容赦ください。

変数名	説明
$arg_name	リクエストラインのパラメータ名
$args	リクエストラインのパラメータ値
$binary_remote_addr	バイナリフォーマットのクライアントアドレス。値の長さは、常に4バイトとなる
$body_bytes_sent	レスポンスヘッダ情報を含まないクライアントへの送信バイト数。この値は、Apacheのmod_log_configで定義されている%Bパラメータと同義である
$bytes_sent	クライアントへ送信したバイト数
$connection	接続シリアル番号
$connection_requests	現在の接続リクエスト番号
$content_length	リクエストヘッダの“Content-Length”
$content_type	リクエストヘッダの“Content-Type”
$cookie_name	クッキー名
$document_root	現在のリクエストのルートドキュメントあるいはaliasディレクティブの値
$document_uri	$uriと同じ
$host	次に順で決まる。リクエストラインから抽出されるホスト名、あるいは、リクエストヘッダ情報の“Host”、あるいは、リクエストに一致したサーバ名
$hostname	ホスト名
$http_name	任意のリクエストヘッダフィールド。この変数名末尾の“name”は、リクエストヘッダのフィールド名を小文字に変換し、ダッシュ“-”をアンダーバー“_”へ変換したものになる
$https	SSLモードで接続の場合は、“on”それ以外は空文字
$is_args	リクエストラインがパラメータを持つなら“?”以外は空文字
$limit_rate	この変数を設定すると、制限のレスポンス率の制限できる。;limit_rate参照
$msec	現在時刻（ms）、ログ出力時は、ログ書き込み時の時間（ms）
$nginx_version	Nginxバージョン
$pid	ワーカープロセスのPID
$pipe	パイプによるリクエストの場合は“p”以外は“.”
$proxy_protocol_addr	PROXYプロトコルヘッダからのクライアントアドレス、それ以外は空文字。PROXYプロトコルは事前にlistenディレクティブのproxy_protocolパラメータを設定し有効にする必要がある
$query_string	$argsと同じ
$realpath_root	すべてのシンボリックリンクとともにリアルなパスへ解決されたルートに対する絶対パス、あるいは現在のリクエストのためaliasディレクティブ値
$remote_addr	クライアントアドレス
$remote_port	クライアントポート
$remote_user	Basic認証で指定されたユーザ名
$request	すべてのオリジナルなリクエストライン
$request_body	リクエストボディ。 この値は、proxy_pass、fastcgi_pass、uwsgi_passそしてscgi_passディレクティブによって処理されたロケーションの中で使用できる
$request_body_file	リクエストボディのテンポラリファイルの名前。処理の終わりに、このファイルは、削除される必要がある。 常にリクエストボディをファイルへ書き込むために、client_body_in_file_onlyは、有効になっている必要がある。テンポラリファイル名が、プロキシリクエストやFastCGI/uwsgi/SCGIサーバへのリクエストの中で渡される時、そのリクエストボディは、それぞれのディレクティブproxy_pass_request_bodyをoffにする、fastcgi_pass_request_bodyをoffにする、uwsgi_pass_request_bodyをoffにする、あるいは、scgi_pass_request_bodyをoffにするで無効すべきである
$request_completion	リクエストが完了した場合“OK”以外は空文字
$request_filename	現在リクエスト、ルートに基づいた、あるいはaliasディレクティブ、そして、リクエストURIのファイルパス
$request_length	リクエストの長さ（リクエストライン、ヘッダー、ボディを含む）
$request_method	リクエストメソッド、通常、“GET”or“POST”のいずれか
$request_time	リクエスト処理時間（ms）;クライアントからのリクエストの最初のバイトを読み始めてから、最後のバイトがクライアントへ送信された後にログを書くまでの経過時間
$request_uri	オリジナルの全リクエストURI（パラメータ値も含む）

変数名	説明
$scheme	リクエストスキーマ。“http”or“https”のいずれか
$sent_http_name	任意のレスポンスヘッダフィールド。この変数名末尾の“name”は、レスポンスヘッダのフィールド名を小文字に変換し、ダッシュ“-”をアンダーバー“_”へ変換したものになる。
$server_addr	リクエストを受け入れたサーバのアドレス。この変数の値を計算することは、通常は1つのシステムコールを必要する。システムコールを回避するためには、listenディレクティブで、アドレスを指定し、bindパラメータを使用する必要がある。
$server_name	リクエストを受け入れたサーバ名
$server_port	リクエストを受け入れたサーバポート
$server_protocol	リクエストプロトコル。通常は“HTTP/1.0”or“HTTP/1.1”のいずれか
$status	レスポンスステータス
$tcpinfo_rtt, $tcpinfo_rttvar, $tcpinfo_snd_cwnd, $tcpinfo_rcv_space	クライアントTCP接続情報。TCP_INFOソケットオプションをサポートしているシステム上で利用可能
$time_iso8601	ISO8601標準フォーマットによる時間
$time_local	Common Logフォーマットによる時間
$uri	統一化されたリクエストの現在のURI。$uriの値は、たとえば内部リダイレクトをするとき、またはインデックスファイルを使用するときのように、リクエストの処理中に変更されるかもしれない

で定義されています(**図9**)。

図9はApacheのcombinedの定義(**図1**)にX-Forwarded-For情報を付加しただけのものになっています。各パラメータは**表3**の書式に従ったものです。

Nginxの場合、使用可能な書式はNginxの設定ファイルの中で使用できる変数そのものです。ログのためだけに提供されたものでなく共通的に提供されている変数(**表3**)をそのまま使用できるので、さまざまな情報を出力することができます。ただし、必ずしもその値が設定されているとは限らないので、その点は注意が必要でしょう。

Webサイトへアクセスしてログを出力してみよう

Nginxを起動しておきます。

```
$ /etc/init.d/nginx start
nginx を起動中:                                [  OK  ]
```

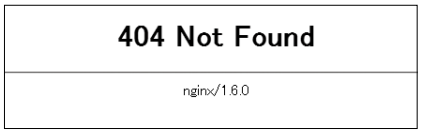
設定を確認できたら、URLにIPアドレスを指定(例　**http://192.168.1.65**)してWebブラウザからサイトへアクセスしてみてください。

tailコマンドを使ってログを確認してみよう

図10のようなデフォルトページが見えていれば、アクセスログ(**/var/log/nginx/access.log**)が出力されるはずです。**tail**コマンドで、最新のアクセスログを出力してみます。

▼図10　Nginxデフォルトページ

▼図11　Nginx 404ページ(例：http://192.168.1.65/asdfasdfagasdgagoajoas.html)




```
$ tail /var/log/nginx/access.log
……(省略)……
192.168.1.33 - - [25/May/2014:19:03:08 +0900] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT6.2;WOW64;rv:24.0) Gecko/20100101 Firefox/24.0" "-"
```

ステータスも200ですから、正しく表示できているようです。

エラーログ(**/var/log/nginx/error.log**)も確認してみましょう。同じように**tail**コマンドを使って、最新のエラーログを出力してみます。

```
$ tail /var/log/nginx/error.log
……(省略)……
```

アクセスログと同時期のログは何も出力されていないことを確認しておきましょう。

あえてエラーを出力してみよう

次に意図的に404エラーを起こしてみます。先のIPアドレス＋ありえないファイル名を入力しアクセスしてみます。

図11のように404のページが表示されたら、先の例と同じように**tail**コマンドでアクセスログを確認してみましょう。

図12のように予想どおりのログ情報です。404を返していますね。続けて、同じようにエラーログも確認してみましょう。

図13のようにアクセスログと同時期に、こちらは「そのようなファイルはありません」というエラーが出力されています。こちらも予想どおりのエラーログです。このようにNginxでもApache同様にログの出力・確認ができました。

次に、Webサイトにどんなアクセスがあったか、アクセスログの解析を行うWebalizerを使ってみます。

Webalizerを使ってログを解析してみよう

CentOSには、Webサーバのアクセスログを解析して、Webサーバで簡単に確認ができるようにHTML形式で結果を出力してくれる便利なツールWebalizerがあります。以下に、このWebalizerのインストールから実際のhttpd(Apache)のアクセスログの解析までを、順を追って解説していきます。ここで使用するWebalizerのバージョンは、V2.21-02です。

①Webalizerのインストール

```
$ yum install webalizer
……(省略)……
Is this ok [y/N]: y
……(省略)……
Complete!
```

②Webalizerのテスト用設定ファイルの作成

Webalizerは、設定ファイルを元にアクセスログの解析を行います。ここでは、テスト用の設定ファイルをオリジナル設定ファイルをコピー、編集して準備します。

```
$ cp /etc/webalizer.conf /etc/webalizer_test.conf
```

コピーした**/etc/webalizer_test.conf**を必要に応じて編集します。29行目あたりのWebサーバのアクセスログファイル指定を確認します。次のようになっていれば、デフォル

▼図12　開くアセスログの確認

```
$ tail /var/log/nginx/access.log
……(省略)……
192.168.1.33 - - [25/May/2014:19:06:14 +0900] "GET /asdfasdfagasdgagoajoas.html HTTP/1.1" 404 168 "-" "Mozilla/5.0 (Windows NT 6.2; WOW64; rv:24.0) Gecko/20 100101 Firefox/24.0" "-"
```

▼図13　エラーログの確認

```
$ tail /var/log/nginx/error.log
……(省略)……
2014/05/25 19:06:14 [error] 3017#0: *4 open() "/usr/share/nginx/html/asdfasdfagasdgagoajoas.html" failed (2: No such file or directory), client: 192.168.1.33, server: localhost, request: "GET /asdfasdfagasdgagoajoas.html HTTP/1.1", host: "192.168.1.65"
```

トのApacheのアクセスログファイルと同じなのでOKです。

```
LogFile         /var/log/httpd/access_log
```

42行目あたりのWebalizerの出力先ディレクトリを確認します。

```
OutputDir       /var/www/usage
```

設定されている出力先ディレクトリが存在するか確認します。インストール時に、デフォルトの出力先ディレクトリは作成されるはずなので、以下のようにディレクトリおよびいくつかのファイルが存在するはずです。もしディレクトリが存在しない場合は作成します。

```
$ ls /var/www/usage
msfree.png  webalizer.png
```

③Webalizerでアクセスログ解析

次のように先に準備したテスト用の設定ファイルを指定してアクセスログの解析を行います。

```
$ webalizer -c /etc/webalizer_test.conf
```

④Webalizerの出力先ディレクトリへのアクセスを許可

Webalizerをインストールした時点で、Webalizer用のApache設定ファイル(**/etc/httpd/conf.d/webalizer.conf**)が作成されますので、それを編集します(**リスト5**)。

Webalizerは、デフォルトの設定でローカルホスト(内部)からのアクセスのみを許容するように設定してあります。そのためネットワーク上の他のPCからアクセスした場合、403でエラーとなってしまいます。

ここでは、すべてリモートで操作していますので、ここもネットワーク上の他のPCのWebブラウザからWebalizerの出力先ディレクトリにApacheを介してアクセスした場合、。その際は、14行目のように、アクセスするPCのIPアドレスを許容するように設定します。編集後

▼リスト5　/etc/httpd/conf.d/webalizer.conf

```
01 #
02 # This configuration file maps the ➢
   webalizer log analysis
03 # results (generated daily) into the➢
   URL space.  By default
04 # these results are only accessible ➢
   from the local host.
05 #
06 Alias /usage /var/www/usage
07
08 <Location /usage>
09     Order deny,allow
10     Deny from all
11     Allow from 127.0.0.1
12     Allow from ::1
13     # Allow from .example.com
14     Allow from 192.168.1.33
15 </Location>
```

にApacheを再起動します。

```
$ /etc/init.d/httpd restart
httpd を停止中:      [  OK  ]
httpd を起動中:      [  OK  ]
```

⑤Webalizerの出力先ディレクトリをWebブラウザからアクセス

URLにIPアドレス+webalizerの出力先(例**http://192.168.1.65/usage/**)を指定して、Webブラウザからアクセスしてみましょう。**図14**のようにグラフが表示されれば、正しく解析できているでしょう。また、**図14**からリンクをたどると**図15**のような詳細データを参照できます。1日当たりのアクセス数や404の返信数など、さまざまなデータを1つのページにまとめて表示できます。

このように、Webalizerを使ってアクセスログを毎日解析すれば、Webサイトのアクセス状態をおおむね把握できます。そこで、次にcronを使って毎日自動更新するように設定してみましょう。

⑥ crontabを編集してWebalizerを自動的に毎日起動

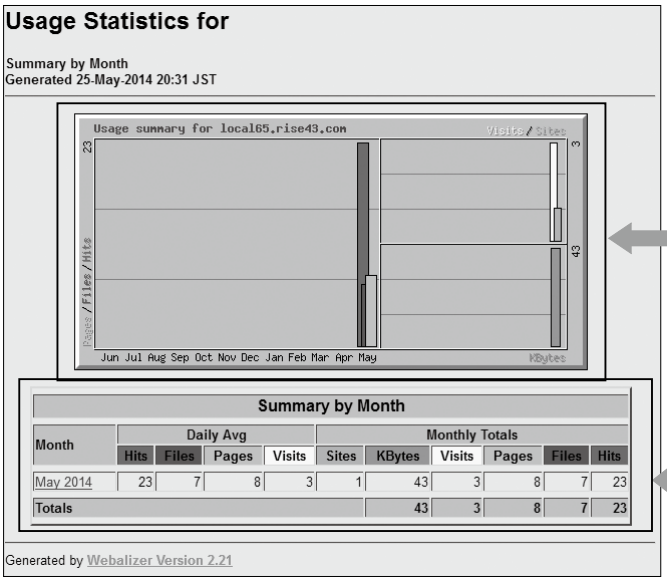
```
$ vi /etc/crontab
……(省略)……
## webalizer
0  4  * * * root /usr/bin/webalizer -c
/etc/webalizer_test.conf
```

このように設定[注2]しておけば毎日4時にWebalizerがアクセスログを解析します。

ここでは、Webalizerのデフォルトの設定だけでApacheアクセスログ解析してみましたが、Webalizerには、ほかにも日本語の設定やIP

注2) /etc/crontabは、テキストファイルです。ここでは、viコマンドを使って編集していますが、nanoコマンドなどでも同じように編集できます。

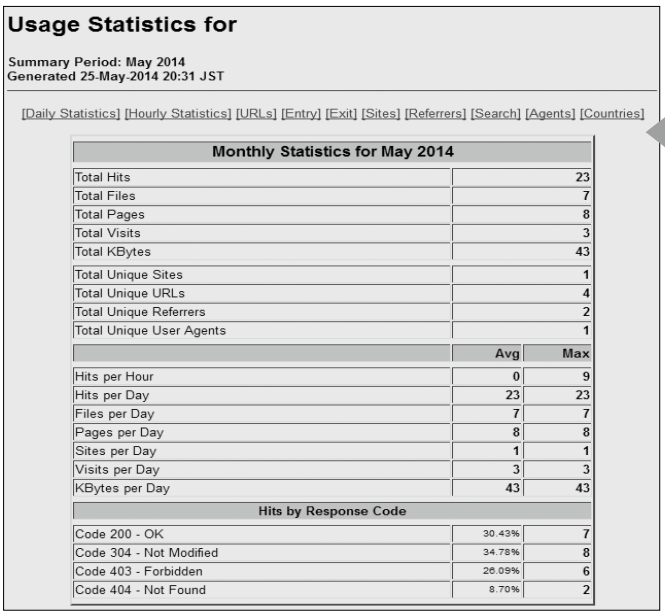
▼図14　Webalizerの最初のページ



直近1年分のアクセスページ数、ファイル数、ヒット数がグラフで表示されます。

さらに詳細な月ごとのアクセスデータが一覧表で出力されます。また、Month列の各年月は、リンクとなっていて、クリックすると**図15**へ遷移できます。

▼図15　Webalizerの詳細のページ



上図のリンクからたどれるさらに詳細なページです。下記に加えてHTTPのステータスコードごとの月合計数など、さまざまな統計データが出力されます。

- 月合計ヒット数
- 月合計ファイル数
- 月合計ページ数
- 月合計訪問者数
- 月合計伝送サイズ
- 月合計ユニークサイト数
- 月合計ユニークURL数
- 月合計ユニークリファラ数
- 月合計ユニークユーザエージェント数

アドレスなどの条件で解析の対象を限定するなど、いろいろな設定ができます。もちろん、Nginxのアクセスログ解析もできます。

WebalizerとGoogle Analyticsの違いって何？（おまけ）

さて、Webサイトのアクセスログ解析では、Google Analyticsという便利なツールがあります（**図16**）。

WebalizerもGoogle Analyticsも同様にアクセス数が出力されます。さて、この違いは何でしょう。先にも解説したように、Webサーバのアクセスログを元に作成されるのがWebalizerの出力情報です。それに対してGoogle Analyticsは、訪問したユーザがWebブラウザのJavaScriptを使ってGoogleのサーバへ通知し、Googleのサーバでログ情報として保管します。その情報を元に、Google Analyticsが解析、出力しています。

つまり、Webalizerの解析結果は、

> Webサーバが「……のページがアクセスされました」と検出した情報を元に解析したものです。

Google Analyticsは、

> Webサイトへの訪問者が「……のページを見ました」とGoogleへ報告した情報を元に解析したものです。

▼図16　Google Analytics画面

ただ、Google Analyticsの場合、訪問者がJavaScriptを有効にしていない場合は、Googleへ報告を行わないため、実際の訪問者数とは差があることがあります。

最近では、Google Analyticsが圧倒的な機能を誇っていますので、Webalizerが陳腐な存在に感じておられる方が多いかもしれません。ただ、Webalizerが出力する情報はリアルなサーバ情報であることは間違いありません。また、Webサーバが出力するログ情報は、アクセス情報だけでなくサーバの負荷やエラー、障害などサーバの状態を細やかに知らせてくれます。サーバの健康状態を監視・管理するには、これに勝るものはないでしょう。そう考えれば、Google Analyticsはログ情報のほんの一面でしかないと気づくでしょう。情報の質がそもそも違うのですから、Google Analyticsがあるからログはいらないという短絡的なものでなく、これらのログ情報と並行して、Google Analyticsの情報を使っていくことが今は求められているのだと思います。SD

第3章 MySQLのロギングを見てみよう

Writer 近藤 成(こんどう じょう)
Mail jj2kon@gmail.com Web http://server-setting.info/

データベースのログは非常に大事です。経営を進めるうえで命とも言えるさまざまなデータを蓄積しているわけですから、有事の際にいつでも回復できるようにしておかねばなりません。あらゆるコンピュータは機械ですので壊れることがあります。そんなときにログが役に立ちます。本稿ではMySQL(Ver.5.1.73)のログを解説します。

MySQLの4つのログ

MySQLのログには大きく分けて2つ、さらに分類すると4つの種類があります。

- **エラー系**
  エラーログ、一般クエリログ、スロークエリログ……問題解決を図る、MySQLの動作に関するログ
- **バックアップ系**
  バイナリログ……データベースの内容をバックアップするためのログ

ここでは、mysqlがインストール、および実行されているものとして以降解説します。もしインストールされていない場合は、**yum**コマンドを使ってmysql-serverをインストールしてください。また、**/etc/init.d/mysqld**でmysqlを起動しておきましょう。

まずは、SQL文を使ってテスト用のデータベース、テーブルの作成を行い、続けてログの解説を行っていきます。

MySQLの基本操作

MySQLのサーバに**mysql**コマンドでログインすると、SQL文でデータベースの操作ができます。**mysql**コマンドは次の形式で使います。

```
mysql [オプション] [データベース]
```

ここでは、オプションは**-u ユーザ指定**ぐらいしか使いませんが、非常に多岐にわたっています。全オプションはMySQLの公式サイト[注1]を参照ください。

では、**mysql**コマンドを使ってログインしてみてください。**mysql -u root**だけでログインできる[注2]はずです。ログインできたら、以降の手順でデータベース、およびテーブルを作成し、データの操作を行ってみましょう。

①データベースの作成

mysqlコマンドでログインしたら、**mysql>**のようにプロンプトが表示されます。そこで、次のようにのSQL文を入力し、テスト用のデータベース(**SAMPLEDB**)を作成します。

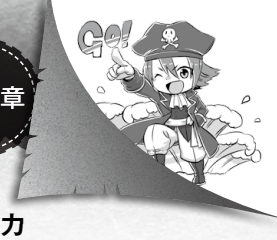
```
mysql> CREATE DATABASE SAMPLEDB;
Query OK, 1 row affected (0.03 sec)
```

ヒント
SQL文: **CREATE DATABASE データベース名;**
これで指定したデータベース名のデータベースを作成できます。

②操作するデータベースの切り替え

使用するデータベースをテスト用のデータベー

注1) http://dev.mysql.com/doc/refman/5.1/ja/mysqlcommand-options.html
注2) MySQLはデフォルトでrootという管理ユーザが存在します。CentOSでは、インストールした時点でrootにパスワードなしでログインできます。これは本来よくありませんが、本稿はログがテーマなので、それらの設定は割愛します。



ス(SAMPLEDB)へ切り替えます。

```
mysql> USE SAMPLEDB;
Database changed
```

ヒント

SQL文:**USE データベース名;**
指定したデータベースを使用できるようになります。Windowsのcdコマンドと似ています。カレントのデータベースを切り替えます。

③テスト用テーブル(SAMPLETABLE)の作成

単純にNAMEという文字列のカラムを持つテスト用のテーブル(**SAMPLETABLE**)を作成します。

```
mysql> CREATE TABLE SAMPLETABLE (NAME ↵
TEXT);
Query OK, 0 rows affected (0.01 sec)
```

ヒント

SQL文:**CREATE TABLE テーブル名(テーブル構造宣言);**
指定したテーブル名のテーブルを作成できます。ここで指定しているTEXTは文字列の意味です。つまり、NAMEというカラム(列)は文字列ですよ、という宣言をしています。Excelならば"NAME"という列を持つSheet(シート名"SAMPLETABLE")を作成するという具合です。

④テスト用テーブルの全出力

```
mysql> SELECT * FROM SAMPLETABLE;
Empty set (0.00 sec)
```

ヒント

SQL文:**SELECT * FROM テーブル名;**
これで指定したテーブルの全出力をします。

空を確認します。

⑤テーブルにデータ(1行)を追加

```
mysql> INSERT INTO SAMPLETABLE (NAME) ↵
VALUES("TARO");
Query OK, 1 row affected (0.00 sec)
```

ヒント

SQL文:**INSERT INTO テーブル名(カラム名) VALUES(値);**
これで指定したテーブルにデータを挿入できます。Excelでいうと行を追加するという具合です。

⑥再度、テスト用テーブルを全出力

```
mysql> SELECT * FROM SAMPLETABLE ;
+------+
| NAME |
+------+
| TARO |
+------+
1 row in set (0.00 sec)
```

⑤で追加したデータ(行)が出力されたことを確認します。最後は、**exit**で**mysql**コマンドを終了します。

```
mysql> exit
Bye
```

SQLは、奥が深いので、ここではログの解説に必要最小限のSQLだけを紹介しました。以降、本節で作成したデータベース:**SAMPLEDB**、テーブル:**SAMPLETABLE**を使ってMySQLのログを解説します。

エラーログを使ってみよう

エラーログは、システムロギングとも言われ、mysql**d**(MySQL**デーモン**)の起動、実行、停止のロギング、また発生したエラー情報などをロギングします。

エラーログの設定を確認しておきましょう

このロギングを行うには、mysqlの設定ファイル(**/etc/my.cnf**)で、次の設定[注3]をします。

```
……(省略)……
[mysqld_safe]
log-error=/var/log/mysqld.log
log-warnings=1
……(省略)……
```

log-error:エラーログのファイル名を設定する。
log-warnings:警告を出力する場合は、1(デフォルト)を設定する。0は出力しない。

注3) ここでは、mysqld_safeセクションにエラーログの設定を行っていますが、そもそも、これは、CentOSのデフォルトの設定をそのまま使用しているものです。mysqld_safeセクションは、mysqld_safeによって読み込まれる設定情報で、CentOSでは、mysqld_safeからmysqldが起動されるため、mysqld_safeセクションにエラーログの設定があります。

ヒント

MySQLでは、設定項目をオプションと呼びます。このオプション名は、バージョンによって異なるので注意しておく必要があります。本稿で使用しているMySQLはバージョンが5.1ですが、MySQL 5.5以降では、オプション名のハイフン"-"は、アンダーバー"_"に置き換えられます。ただし、MySQL 5.5以前からあるオプションは、そのままハイフン"-"を使ったオプション名も利用できます。

たとえば、上記のオプションは次のように変更されています。

```
log-err or log-error → log_error

log-warnings→log_warnings
```

ただ、上記オプションは、いずれも5.5以前からあるので両方とも利用できます(5.1では、ハイフン"-"のみ利用可能)。

エラーログを見てみましょう

エラーログは、CentOSではMySQLをインストールした状態(デフォルト)で出力するように設定されていますので、最初にMySQLを起動した時点ですでに出力されているはずです。

また、CentOS + MySQLでデフォルトのエラーログの出力先は、**/var/log/mysqld.log**です。**リスト1**は、そのエラーログの出力例です。

ヒント

エラーログで**[Error]**情報が出力された場合、その情報はデータベースのファイルが壊れたり致命的なことが少なくありません。その場合データベースの作り直しなどの致命的な対処が必要な場合が多くあります。そのため**[Error]**情報が出力される前の早い段階で**[Warning]**、**[Note]**などの情報に、十分目を光らせ早目早目の対応を心掛けておきましょう

一般クエリログを使ってみよう

一般クエリログは、クエリ(SQL)ロギングとも言われ、mysqld(MySQLデーモン)がクライアントと接続したときの情報、ならびに実行したクエリ(SQL)情報をロギングします。

一般クエリログの設定を確認しておきましょう

このロギングを行うには、mysqlの設定ファイル(**/etc/my.cnf**)で次のような設定をします。

```
[mysqld]
……(省略)……
#  Query log
log=/var/log/mysql/sql.log
……(省略)……
```

▼リスト1　/var/log/mysqld.log([Note]は注意、[Error]はエラー、[Warning]は警告)

```
140525 23:07:04 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql
140525 23:07:04  InnoDB: Initializing buffer pool, size = 8.0M
140525 23:07:04  InnoDB: Completed initialization of buffer pool
InnoDB: The first specified data file ./ibdata1 did not exist:
InnoDB: a new database to be created!
140525 23:07:04  InnoDB: Setting file ./ibdata1 size to 10 MB
InnoDB: Database physically writes the file full: wait...
140525 23:07:05  InnoDB: Log file ./ib_logfile0 did not exist: new to be created
InnoDB: Setting log file ./ib_logfile0 size to 5 MB
InnoDB: Database physically writes the file full: wait...
140525 23:07:05  InnoDB: Log file ./ib_logfile1 did not exist: new to be created
InnoDB: Setting log file ./ib_logfile1 size to 5 MB
InnoDB: Database physically writes the file full: wait...
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
140525 23:07:05  InnoDB: Started; log sequence number 0 0
140525 23:07:05 [Note] Event Scheduler: Loaded 0 events
140525 23:07:05 [Note] /usr/libexec/mysqld: ready for connections.
Version: '5.1.73'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  Source distribution
```

log：一般クエリログの出力の有無を指定する。

一般的に、ここにログファイル名を指定します。オプションキーの“log”だけを指定して、ファイル名を指定しない場合は、**ホスト名.log**のファイル名で保存されます。一般クエリログを出力しない場合は、オプションキーの“**log**”自体を削除します。また、この設定例は古くからある設定方法で、今では次のように設定することで同じことが実現できます。

```
[mysqld]
……(省略)……
#  Query log
general-log=1
general-log-file=/var/log/mysql/sql.log
log-output=FILE
……(省略)……
```

general-log：一般クエリログの出力の有無を設定する(1：出力する、0：出力しない)。

general-log-file：一般クエリログのファイル名を設定する。指定しない場合は、**ホスト名.log**のファイル名で保存される。

log-output：一般クエリログとスロークエリログの出力先を設定(TABLE：テーブルへのログ、FILE:ファイルへのログ、NONE:テーブルまたはファイルにログしない、のいずれか)。

ここでは、出力先ディレクトリを、MySQL用のディレクトリ(**/var/log/mysql**)にしています。このディレクトリは、デフォルトで存在しませんので作成します。また、このディレクトリにmysqldがファイルを作成できるように、ディレクトリの所有者をmysqlに変更しておきます。

▼図1　データの挿入

```
$ mysql -u root SAMPLEDB
Welcome to the MySQL monitor.  Commands end with ; or \g.
……(省略)……
mysql> INSERT INTO SAMPLETABLE (NAME) VALUES("JIRO");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM SAMPLETABLE ;
+--------+
| NAME   |
+--------+
| TARO   |
| HANAKO |
| JIRO   |
+--------+
3 rows in set (0.00 sec)
```

```
$ mkdir /var/log/mysql
$ chown mysql:mysql /var/log/mysql
```

ヒント

chownコマンドは、所有者を変更するコマンドです。パラメータには[所有するユーザ名]:[グループ名＋変更したいディレクトリ名あるいはファイル名]を指定します。

すべての設定を終えたら、mysqldを再起動します。

```
$ /etc/init.d/mysqld restart
mysqld を停止中:     [  OK  ]
mysqld を起動中:     [  OK  ]
```

一般クエリログを見てみましょう

図1のように**mysql**コマンドでテーブルにデータを挿入してみましょう。ここでは、**mysql**コマンドにデータベース名を指定しています。データベース名を指定することでSQL文“**use データベース名;**”を省略できます。つまり、カレントのデータベースは、指定されたデータベース名になります。

次は、一般クエリログ(**/var/log/mysql/sql.log**)を出力してみましょう。**リスト2**は、その出力イメージです。出力された一般クエリログを簡単に解説します。

1行目：mysqldのバージョン表記とログの開始を出力したものです。

2行目：tcpポート番号、ソケット を出力したものです。

3行目：4行目からのタイトルを出力したものです。

6行目：日付、時間、ID番号、コマンド、パラメータがそれぞれ出力されています。日付は、“**140526**”→“**2014/05/26**”のことです。コマンドは“**Connect**”：接続の意味です。また、パラメータ

▼リスト2　一般クエリログをcatコマンドで出力

```
01 /usr/libexec/mysqld, Version: 5.1.73-log (Source distribution). started with:
02 Tcp port: 0  Unix socket: /var/lib/mysql/mysql.sock
03 Time                 Id Command    Argument
04 140526  1:05:04     1 Connect   UNKNOWN_MYSQL_US@localhost as anonymous on
05 1 Quit
06 140526  1:09:03     2 Connect   root@localhost on SAMPLEDB
07 2 Query     show databases
08 2 Query     show tables
09 2 Field List        SAMPLETABLE
10 2 Query     select @@version_comment limit 1
11 140526  1:09:10     2 Query     INSERT INTO SAMPLETABLE (NAME) VALUES("JIRO")
12 140526  1:09:34     2 Query     SELECT * FROM SAMPLETABLE
```

には、“root@localhost on SAMPLEDB”とあります。これは、データベースSAMPLEDBを使用しようとlocalhostのrootが接続してきたという意味です。

11行目：mysqlコマンドでテーブルにレコードを挿入したSQLログです。

12行目：mysqlコマンドでテーブルを出力したSQLログです。

このようにクライアントが接続したログ情報、SQLを処理したログ情報が出力されます。もし、SQLで構文エラーが発生しても一般クエリログには出力されません。発行(実行)したSQLだけが出力されることに注意してください。

スロークエリログを使ってみよう

スロークエリログは、スロークエリ(デバッグ)ロギングとも言われ、mysqld(MySQLデーモン)はlong_query_timeで指定した秒数より時間を要したクエリ(SQL)、またはインデックスを使用しなかったクエリ(SQL)をロギングします。

スロークエリログの設定を確認しておきましょう

このロギングを行うには、mysqlの設定ファイル(/etc/my.cnf)で、次の設定を行います。

```
[mysqld]
……(省略)……
#  Slow Query log
slow_query_log=1
slow_query_log_file=/var/log/mysql/slow.➥
log
long_query_time=1
log_queries_not_using_indexes
log_slow_admin_statements
```

slow_query_log：スロークエリログの出力の有無を設定する(1:出力する、0:出力しない)。

slow_query_log_file：スロークエリログのファイル名を設定する。指定しない場合は、general_log_fileに従う(つまり、一般クエリログと同じになる)。

long_query_time：クエリの処理時間(秒単位)を設定する。この秒数を超えるとスロークエリログに出力される(デフォルトは10秒)。

log_queries_not_using_indexes：インデックスしていないクエリをすべて出力する場合に指定する(これを指定しておくと、先に作成したテスト用テーブルはインデックスしていないため、テスト用テーブル操作の全クエリが出力される)。

log_slow_admin_statements：管理用のステートメント(OPTIMIZE TABLE、ANALYZE TABLE、ALTER TABLEなど)についても同様に処理に時間がかかるものを出力したい場合に指定する。

ここでは、一般クエリログと同じように出力先ディレクトリを、MySQL用のディレクトリ(/var/log/mysql)にしています。このディレクトリが存在しない場合は、作成します。また、このディレクトリにmysqldがファイルを作成できるように、ディレクトリの所有者をmysqlに変更しておきます。

```
$ chown mysql:mysql /var/log/mysql
```

すべての設定を終えたら、mysqldを再起動します。

```
$ /etc/init.d/mysqld restart
mysqld を停止中:        [  OK  ]
mysqld を起動中:        [  OK  ]
```

スロークエリログを見てみよう

図3のように**mysql**コマンドでテーブルを出力します。

次にスロークエリログ(**/var/log/mysql/slow.log**)を出力してみましょう。**リスト3**はその出力イメージです。出力されたスロークエリログを簡単に解説します。

1〜4行目：一般クエリログと同じです(第3章 p.54参照)

5行目：接続ユーザ、ホスト名 が出力されます。

6行目：**Query_time**(クエリ実行時間)、**Lock_time**(テーブルあるいはデータベースがロックされた時間)、**Rows_examined**(処理対象となった行数)がそれぞれ出力されます。

9行目：先に実行したSQL文 が出力されます。

ここで出力されたスロークエリログ情報は、遅延が発生してログ情報が出力されたわけではなく、**log_queries_not_using_indexes**を設定していたために出力されたものです。もし、クエリ遅延が発生したログ情報であれば、先の設定では、**Query_time**(**リスト3**の6行目)が、少なくとも10秒以上になっていないといけません。ここでの出力では、"**Query_time: 0.000201**"とまったく問題ない数値です。

また、**Query_time**は、CPU処理時間でなく、実際にリアクションするまでの時間なので、ここで出力された時間は、そのままユーザへのレスポンス時間に直結します。この値が大きい場合は、ログ情報の実行したSQLを分析することで、データベースでの遅延個所を確定できますし、レスポンス時間の改善を図られるようになるでしょう。

▼図3　テーブルの出力

```
$ mysql -u root SAMPLEDB
Welcome to the MySQL monitor.  Commands end with ; or \g.
……(省略)……
mysql> SELECT * FROM SAMPLETABLE ;
+--------+
| NAME   |
+--------+
| TARO   |
| HANAKO |
| JIRO   |
+--------+
3 rows in set (0.00 sec)
```

▼リスト3　/var/log/mysql/slow.log

```
01 /usr/libexec/mysqld, Version: 5.1.73-log (Source distribution). started with:
02 Tcp port: 0  Unix socket: /var/lib/mysql/mysql.sock
03 Time                 Id Command    Argument
04 # Time: 140526  1:47:41
05 # User@Host: root[root] @ localhost []
06 # Query_time: 0.000201  Lock_time: 0.000095 Rows_sent: 4  Rows_examined: 4
07 use SAMPLEDB;
08 SET timestamp=1401036461;
09 SELECT * FROM SAMPLETABLE ;
```

バイナリログを使ってみよう

バイナリログは、バイナリ(バックアップ)ロギングとも言われ、mysqld(MySQLデーモン)のデータ変更のステートメントをバイナリ情報でロギングします。また、レプリケーションにも使用されます。

ヒント

レプリケーション(replication)は、日本語ではレプリカという言葉がなじみ深いでしょう。複製という意味です。昔は1つの高性能なサーバにデータベースをインストールし、集中管理する方法もよく用いられましたが、輻輳(ふくそう)した場合に1台への負荷が集中してレスポンスが非常に悪くなるという問題がありました。この問題の対策の1つとして、レプリケーションを各サーバに配置することで、少なくともデータベースの読み込みは、各サーバのレプリケーション(複製)に担わせ、負荷を分散させる対策が図られるようになりました。

バイナリログの設定を確認しておきましょう

このロギングを行うには、mysqlの設定ファイル(**/etc/my.cnf**)で、次のような設定を行います。

```
[mysqld]
……(省略)……
#  Binary log
log_bin=/var/log/mysql/bin.log
log_bin_index=/var/log/mysql/bin.list
max_binlog_size=1M
expire_logs_days=1
```

log_bin：バイナリログの出力の有無を設定します。ここにログファイル名を指定することで、出力を有効にします。

log_bin_index：バイナリログインデックスファイル名を設定します。バイナリログファイル名を管理するためのファイル名になります。

max_binlog_size：バイナリログの最大ファイルサイズを指定します。ここで指定したファイルサイズを超えた場合は、ファイルを自動で切り替えます。設定可能な値は、4,096B以上1GB(デフォルト)以下です。

expire_logs_days：バイナリログの保存期間を日数で指定します。この日数を超えたものは削除されます。デフォルト0は、削除しません。

上記以外にもバイナリログに関しては、いろいろ設定できるようになっています。たとえば、**binlog_format**(バイナリロギング形式の設定)は、**STATEMENT**(デフォルト)、**ROW**、**MIXED**のいずれかを指定します。

これ以外にも、まだたくさんあります[注4]。

ここでは、一般クエリログと同じように出力先ディレクトリを、MySQL用のディレクトリ(**/var/log/mysql**)にしています。このディレクトリが存在しない場合は、作成します。また、このディレクトリにmysqld(MySQLデーモン)がファイルを作成できるように、ディレクトリの所有者を**mysql**に変更しておきます。

```
$ chown mysql:mysql /var/log/mysql
```

すべての設定を終えたら、mysqldを再起動します。

```
$ /etc/init.d/mysqld restart
mysqld を停止中:                           [  OK  ]
mysqld を起動中:                           [  OK  ]
```

バイナリログを見てみましょう

図4のように**mysql**コマンドでテーブルにデータを挿入します。次にバイナリログ(**/var/log/mysql/bin.000001**)を出力してみましょう。バイナリログは、名前のとおりバイナリ情報なので、**cat**コマンドなどで出力できません。次のように**mysqlbinlog**コマンドを使ってロギングされている内容を確認します。

```
$ mysqlbinlog /var/log/mysql/bin.000001
```

このコマンドは、バイナリログをテキスト(クエリログ)変換するMySQLのユーティリティ

注4) 詳しくは、http://dev.mysql.com/doc/refman/5.1/ja/server-system-variables.htmlを参照してください。

ツールです。これもmysqlコマンド同様、オプションがたくさんあります。ここでは、バイナリログのファイルを指定するぐらいのことしかしませんが、詳しくは、MySQLの公式サイト注5を参照ください。

リスト4は、バイナリログファイル(/var/log/mysql/bin.000001)を mysqlbinlog コマンドで出力したバイナリログを簡単に解説します。

12行目：at 106の106がロールバックなどで使うポジション番号です(第3章の「バイナリログからロールバックしてみよう」p.60で使用)

24行目：mysqlコマンドでテーブルにレコードを挿入したSQLログです。

注5) http://dev.mysql.com/doc/refman/5.1/ja/mysqlbinlog.html

▼図4　テーブルへのデータ挿入

```
$ mysql -u root SAMPLEDB
Welcome to the MySQL monitor.  Commands end with ; or \g.
……(省略)……
mysql> INSERT INTO SAMPLETABLE (NAME) VALUES("SAKURA");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM SAMPLETABLE ;
+--------+
| NAME   |
+--------+
| TARO   |
| HANAKO |
| JIRO   |
| SAKURA |
+--------+
4 rows in set (0.00 sec)
```

▼リスト4　/var/log/mysql/bin.000001(#以降、/* */で括られた範囲はコメント)

```
01 /*!40019 SET @@session.max_insert_delayed_threads=0*/;
02 /*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
03 DELIMITER /*!*/;
04 # at 4
05 #140526  2:11:09 server id 1  end_log_pos 106   Start: binlog v 4, server v 5.1.73-log created 140526 ➥
   2:11:09 at startup
06 # Warning: this binlog is either in use or was not closed properly.
07 ROLLBACK/*!*/;
08 BINLOG '
09 LSSCUw8BAAAAZgAAAGoAAAABAAQANS4xLjczLWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
10 AAAAAAAAAAAAAAAAAAAtJIJTEzgNAAgAEgAEBAQEEgAAUwAEGggAAAAICAgC
11 '/*!*/;
12 # at 106
13 #140526  2:13:18 server id 1  end_log_pos 218   Query   thread_id=2 exec_time=0 error_code=0
14 use `SAMPLEDB`/*!*/;
15 SET TIMESTAMP=1401037998/*!*/;
16 SET @@session.pseudo_thread_id=2/*!*/;
17 SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=1, @@session.unique_checks=1, @@session.➥
   autocommit=1/*!*/;
18 SET @@session.sql_mode=0/*!*/;
19 SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
20 /*!¥C latin1 *//*!*/;
21 SET @@session.character_set_client=8,@@session.collation_connection=8,@@session.
collation_server=8/*!*/;
22 SET @@session.lc_time_names=0/*!*/;
23 SET @@session.collation_database=DEFAULT/*!*/;
24 INSERT INTO SAMPLETABLE (NAME) VALUES("SAKURA")
25 /*!*/;
26 DELIMITER ;
27 # End of log file
28 ROLLBACK /* added by mysqlbinlog */;
29 /*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
```

リスト4のmysqlbinlogコマンドによる出力情報でわかるように全文SQLです。一般クエリログ、スロークエリログとは、その点で異なりリスト4の情報は、そのままMySQLにて実行することができるSQL文になっています。このようにバイナリログから出力されるSQL文をロールバック(用)SQL文ともいい、これを用いて好きな位置へロールバック(戻る)できることから、バイナリログをバックアップログとも言います。

最後に、実際にこのバイナリログを使ってロールバックを試してみましょう。

バイナリログからロールバックしてみよう(おまけ)

フルバックアップ+バイナリログがあれば、フルバックアップを行った時点からバイナリログが存在する範囲内で好きな状態までロールバックできます。ロールバックの前準備からロールバック実施までを解説します。また、バイナリログは、第3章p.58での設定例と同じようにすでに設定されているものとして解説します。

フルバックアップを準備しよう

mysqldumpコマンドでフルバックアップ

図5のようにフルバックを行います。ここで使用しているmysqldumpコマンドは、SQL形式でデータベースをバックアップするコマンドです。このコマンド自体は、コンソールへSQL文を出力するだけなので、“>”リダイレクトを使ってファイルに保存しています。ここで指定しているオプションは、ユーザ名とバックアップ対象範囲(ここではすべて)です。本当に使うオプションは限られていると思いますが、このコマンドもオプションの数がすごいです[注6]。

フルバックアップ状態の確認

図6のようにフルバックアップしたときの状態を確認します。mysqlコマンドは、“-e”オプションを使うと、以降に指定したSQL文を実行できます。つまり、“show master status;”はSQL文で、これをmysqlコマンド1行で実行します。このSQL文は、マスターデータベースの状態を出力したものです。File(バイナリログファイル名)、Position(バイナリログファイルの現在の位置)でフルバックアップを実行したときのバイナリログの正確な位置をしることができます。ここでは、この2つのデータを使いますのでメモしておきましょう。ここまでが前準備[注7]です。ここから先がロールバック可能な範囲になります。

バイナリログを更新してみよう

図7のように、テスト用テーブルに複数のテスト用のレコードを挿入することで、バイナリログを更新します。ここでは、NAME列の

注6) http://dev.mysql.com/doc/refman/5.1/ja/mysqldump.htmを詳しくは参照してください。

注7) 本来、この前準備は、データベースの更新を停止して行う必要がありますが、ここではその手順を割愛しています。

▼図5 フルバックアップの実行

```
$ mysqldump -u root --all-databases > /var/log/mysql/all_backup.sql
```

▼図6 バックアップ状態の確認

```
$ mysql -u root -e "show master status;"
+------------+----------+--------------+------------------+
| File       | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------+----------+--------------+------------------+
| bin.000001 |      218 |              |                  |
+------------+----------+--------------+------------------+
```

“rollback1”から“rollback4”までの4つのレコードを挿入しました。

ロールバックしてみよう

ここで何らかのトラブルが発生し、そのトラブルを解決するために“rollback2”が挿入された位置(“rollback3”の挿入前)までロールバックしなければならなくなったとしましょう。以降、順を追って“rollback2”が挿入された位置までロールバックをしてみます。

①バイナリログの全ファイルをコピー(バックアップ)

```
$ mkdir /var/log/mysql/rollback
$ cp /var/log/mysql/bin* /var/log/mysql/rollback/.
```

ここでは、バイナリログを/var/log/mysql/rollbackというディレクトリに、いったん避難させます。

②“rollback2”の挿入位置を検索

①で退避したバイナリログの中から“rollback2“の挿入個所をmysqlbinlogコマンドで確認します。mysqlbinlogコマンドは、開始日時(“--start-datetime”オプション)、終了日時(“--stop-datetime”オプション)で出力する範囲を指定できます。既知の範囲で指定すると良いでしょう。図8では2014-05-26

▼図7　ロールバック

```
$ mysql -u root SAMPLEDB
Welcome to the MySQL monitor.  Commands end with ; or \g.
……(省略)……
mysql> INSERT INTO SAMPLETABLE (NAME) VALUES("rollback1");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO SAMPLETABLE (NAME) VALUES("rollback2");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO SAMPLETABLE (NAME) VALUES("rollback3");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO SAMPLETABLE (NAME) VALUES("rollback4");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM SAMPLETABLE ;
+-----------+
| NAME      |
+-----------+
| TARO      |
| HANAKO    |
| JIRO      |
| SABURO    |
| SHIRO     |
| SAKURA    |
| rollback1 |
| rollback2 |
| rollback3 |
| rollback4 |
+-----------+
10 rows in set (0.00 sec)

mysql> exit
Bye
```

▼図8　mysqlbinlogコマンドで確認

```
$ mysqlbinlog /var/log/mysql/rollback/bin.000001 --start-datetime "2014-05-26 02:30:00" --stop-datetime "2014-05-26 03:00:00"
……(省略)……
INSERT INTO SAMPLETABLE (NAME) VALUES("rollback1")
/*!*/;
# at 333
#140526  2:56:52 server id 1  end_log_pos 448   Query   thread_id=2     exec_time=0  error_code=0
SET TIMESTAMP=1401040612/*!*/;
INSERT INTO SAMPLETABLE (NAME) VALUES("rollback2")
/*!*/;
# at 448
#140526  2:56:54 server id 1  end_log_pos 563   Query   thread_id=2     exec_time=0  error_code=0
SET TIMESTAMP=1401040614/*!*/;
INSERT INTO SAMPLETABLE (NAME) VALUES("rollback3")
/*!*/;
# at 563
#140526  2:56:56 server id 1  end_log_pos 678   Query   thread_id=2     exec_time=0  error_code=0
SET TIMESTAMP=1401040616/*!*/;
INSERT INTO SAMPLETABLE (NAME) VALUES("rollback4")
……(省略)……
```

02:30:00から03:00:00までの30分を指定してます。

この出力情報から“rollback2”を挿入した位置は、SQL文"INSERT INTO SAMPLETABLE (NAME) VALUES("rollback2")"の位置なので、そのSQL文の後のバイナリログの位置情報“# at”を確認します。ここでは“448”となっています。つまり、データベースをフルバックアップしたときの状態(Position:218)まで戻して、②で確認した位置218～448(ここで確認した位置)までをロールアップすれば良いことになります。

では、この手順にならい、まずフルバックアップしたときまで戻してみましょう。

③フルバックアップした状態までロールバック

```
$ mysql -u root < /var/log/mysql/all_backup.sql
```

ここではフルバックアップファイル(/var/log/mysql/all_backup.sql)を“<”リダイレクトを使ってmysqlコマンドへ渡しています。“<”の後にファイル名を指定すると指定したファイルを読み込み、その内容を“<”の先のコマンドへ渡してくれます。つまり、フルバックアップファイルのSQLを一気にmysqlコマンドを使って実行したことになります。

④フルバックアップした状態まで戻ったか確認

```
$ mysql -u root SAMPLEDB -e "SELECT * FROM SAMPLETABLE ;"
+----------+
| NAME     |
+----------+
| TARO     |
| HANAKO   |
| JIRO     |
| SABURO   |
| SHIRO    |
| SAKURA   |
+----------+
```

ここでは、mysqlコマンド-eオプションを使うことでSQL文を実行しています。NAME列の“rollback1”から“rollback4”までのレコードが見当たりませんから、確かに、フルバックアップの時点まで戻ったようです。

⑤“rollback2”を挿入した位置まで更新

```
$ mysqlbinlog /var/log/mysql/rollback/bin.000001 --start-position 218 --stop-position 448 | mysql -u root
```

ここでは、mysqlbinlogコマンドを使ってバイナリログファイルをSQL変換しています。その際、開始ポジション(--start-position)、終了ポジション(--stop-position)を指定することで、その範囲内のデータをSQLに変換するように指示しています。続けて、“|”パイプ指定し、mysqlコマンドへ渡しています。これで、指定されたポジションの範囲内のSQLをmysqlコマンドは、先のフルバックアップと同じように一気に実行することになります。

⑥意図した位置にロールバックできたか確認

```
$ mysql -u root SAMPLEDB -e "SELECT * FROM SAMPLETABLE ;"
+----------+
| NAME     |
+----------+
| TARO     |
| HANAKO   |
| JIRO     |
| SABURO   |
| SHIRO    |
| SAKURA   |
| rollback1 |
| rollback2 |
+----------+
```

ここでのコマンドは④と同じです。この出力結果には、NAME列の“rollback1”から“rollback2”までのレコードが出力されました。“rollback3”以降は見当たりませんから、確かに意図した位置にロールバックできたようです。

うまくできましたでしょうか？　少し面倒ですが、フルバックアップも含めバイナリログをとっていれば、好きなところまで戻れる点は、もし何かあったときの保険となりえます。その意味でも、このバイナリログは、非常に有効なログであることは間違いありません。SD

第4章 ログを管理・運用しよう（ログローテーションとログウォッチ）

Writer 近藤 成（こんどう じょう）
Mail jj2kon@gmail.com Web http://server-setting.info/

増加し続けるログファイルを管理しなければシステムは破綻してしまいます。そこでログの運用・管理として、ログローテーションがあります。syslog、httpd（Apache）、Nginx、MySQLの設定例を紹介しながら、そのテスト方法を解説します。最後により仕事を便利にするログウォッチの方法を解説します。

ログの保存管理はローテーションが基本

ログ情報はひたすら追加され、ファイルが肥大化することに注意しなければなりません。ディスク容量が格段に増えた現在では、あまり気にしない方も多いようですが、ログをそのままの状態で取り続けると、いつかは破たんします。

そこで必要なのが、ログファイルのローテーション（入れ替え）です。CentOSでは、ログファイルのローテーションを行う`logrotate`アプリケーションが提供されています。

図1は、ログローテーションの簡単な概念図です。**図1**では、5月10日のログ情報が日にちを追うごとにシフトされ4日間の保存期間を終え、最後は削除されるというイメージです。この時、大事なのは保存期間です。

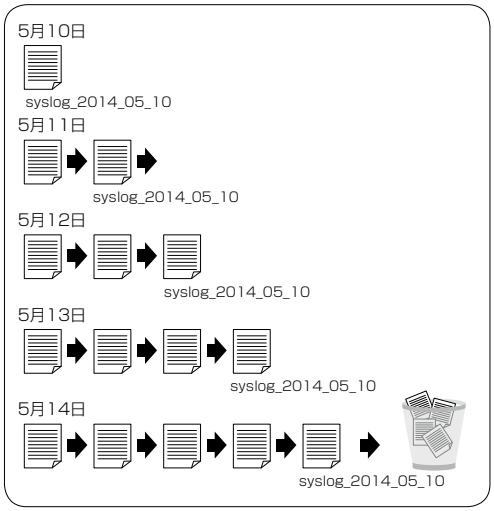
ログの保存期間を決めよう

ログの保存期間は、最長でも3年、あとは1年、半年、3ヵ月、短くて1ヵ月（4週間）という単位でローテーションするのが一般的です。日本版SOX法などの影響で会計に絡むシステムの場合は、ログの保存期間を長くしなければならない場合もあります。また、プロバイダーなどの通信事業を手掛ける会社および団体のサーバは、ネット犯罪への情報提供などもあり、長期間のログを保存する必要があるでしょう。

そうでない限り、ディスク容量が厳しい場合であっても最低1ヵ月程度を目安とします。1週間の保存期間では問題に気づいた時にはログが残っていない恐れがあり、それでは意味をなしません。やはり、短くても1ヵ月（4週間）程度以上のログの保存が必要です。

まずは利用頻度にもよりますが、ディスク容量に10GB程度以上の空きがあるなら、1年分のログ保存期間を設定してみましょう。ディスク容量に余裕がない方は、許される範囲で可能な限り長い期間のログを保存しましょう。最悪の場合でもログが残る可能性が増します。また、運用実績を積み上げれば1日あたりに必要とされるログ容量のより正確な数値がわかります。

▼図1　ログローテーションのイメージ



それでログ保存最長期間を算出できます。より正確なログ保存最長期間から、サーバの運用上の問題点などを鑑み、最適なログ保存期間を決定し、調整しましょう。

とりあえず1年という期間が確保できれば、じっくり腰を据えて最適な保存期間を検討できます。最悪なのは、とりあえず1ヵ月とか、ディスク容量が少ないので10日とか、短い保存期間を決めてしまうことです。とくにサーバの立ち上げ期は、トラブルがつきものです。なるべく長く設定することをお勧めします。

ログローテーションを設定してみよう

実は、ログのローテーションは、rsyslogでも実現できます。ただ、ここまで紹介してきたさまざなアプリケーションのログのほとんどは、それぞれのアプリケーション単独でログを保存する方法のみ(rsyslogでログを管理していません)を紹介してきました。そのため、ここでは、rsyslogのローテーションでなくCentOS 6でデフォルトでインストールされている**logrotate**(Ver.3.7.8)を使ってログローテーションを実施する方法について解説します。

もしインストールされていない場合は、次のように**yum**コマンドを使ってインストールしましょう。

```
$ yum install logrotate
……(省略)……
Is this ok [y/N]: y
……(省略)……
Complete!
```

logrotateは、基本設定ファイル(**/etc/logrotate.conf**)と、個別のアプリケーション用設定ファイル(**/etc/logrotate.d/**配下にアプリケーションごとに設定ファイルがある)で設定します。まず、基本設定ファイル(**/etc/logrotate.conf**)を確認してみましょう。次は、基本設定ファイルから、ログの保存期間に関する情報を抜粋したものです。

```
weekly
rotate 4
```

この意味は次のとおりです。

weekly:ログファイルの切り替えタイミングを1週間ごとに設定します。

rotate 4:ログファイルの保存期間を、ログファイルの切り替えタイミングが4回実施される間に設定します。

つまり、この2つの設定では、1週間ごとにログファイルを切り替え、それが4回(つまり4週間)経過した時にログの保存期間を終える(削除)ということになります。ただし、この設定はデフォルトの設定となりますので、**/etc/logrotate.d/**配下の各アプリケーションの設定ファイルで上記のログファイルの切り替えタイミング、ログファイルの保存期間 を設定した場合は、そちらの設定が優先されます。

続けて、各アプリケーションの設定ファイルを確認してみましょう。**logrotate**は、**/etc/logrotate.d/**配下に各アプリケーション対応の設定ファイルを設置することでログローテーションを実行します。

次に、syslog、httpd、Nginx、mysqlのログローテーションの設定について、簡単に解説します。

syslogの設定を見てみよう

リスト1の例では、切り替えタイミング、切り替え回数が設定されていないので、基本設定の内容が有効になります。つまり、保存期間は4週間となります。

簡単に解説しておくと、"{ }"の間が設定になります。"{"の前に記述されているのがローテーションの対象となるファイル(複数指定可、ワイルドカード使用可)です。各設定項目の詳細は**表1**を参照してください。

▼リスト1　/etc/logrotate.d/syslog

```
/var/log/cron
/var/log/maillog
/var/log/messages     ローテーションの対象ファイル
/var/log/secure
/var/log/spooler
{
  sharedscripts     複数指定したログファイルに対してpostrotateまたはprerotateで記述されたコマンドを1回だけ実行する
  postrotate
      /bin/kill -HUP `cat /var/run/syslogd.pid 2> /dev/null` 2> /dev/null || true
        ローテーションを実行した後に実施するコマンドイメージ
  endscript
}
```

▼リスト2　/etc/logrotate.d/httpd

```
/var/log/httpd/*log {
    missingok
    notifempty
    sharedscripts
    delaycompress
    postrotate
        /sbin/service httpd reload >➥
/dev/null 2>/dev/null || true
    endscript
}
```

▼リスト3　/etc/logrotate.d/nginx

```
/var/log/nginx/*.log {
        daily        毎日切替
        missingok
        rotate 52  daily×52=52日間の保存期間
        compress
        delaycompress
        notifempty
        create 640 nginx adm
        sharedscripts
        postrotate

[ -f /var/run/nginx.pid ] && kill -USR1 ➥
`cat /var/run/nginx.pid`
        endscript
}
```

▼表1　logrotateで使える設定項目一覧

設定値	解説
compress	ローテーションされたログをgzipで圧縮する
create[パーミッション][ユーザ名][グループ名]	ローテーション後に新たな空のログファイルを作成する。ファイルのパーミッション、ユーザ名、グループ名を指定できる
daily	毎日ログローテーションする（正確な日時はcronに依存する）
weekly	毎週ログローテーションする（正確な日時はcronに依存する）
monthly	毎月ログローテーションする（正確な日時はcronに依存する）
ifempty	ログファイルが空でもローテーションする
missingok	ログファイルが存在しなくてもエラーを出さない
nocompress	ローテーションされたログを圧縮しない
nocreate	新たな空のログファイルを作成しない
nomissingok	ログファイルが存在しない場合エラーを出す
noolddir	ローテーション対象のログと同じディレクトリにローテーションされたログを格納する
notifempty	ログファイルが空ならローテーションしない
olddir ディレクトリ名	指定したディレクトリ内にローテーションされたログを格納する
postrotate - endscript	postrotateとendscriptの間に記述されたコマンドをログローテーション後に実行する
prerotate - endscript	postrotateとendscriptの間に記述されたコマンドをログローテーション前に実行する
rotate 回数	指定した回数だけローテーションする
size ファイルサイズ	ログファイルが指定したファイルサイズ以上であればローテーションする
sharedscripts	複数指定したログファイルに対してpostrotateまたはprerotateで記述されたコマンドを1回だけ実行する
nosharedscripts	sharedscriptsの逆。複数指定したログファイルに対してpostrotateまたはprerotateで記述されたコマンドをファイルの数だけ実行する

httpd（Apache）の設定を見てみよう

リスト2も保存期間関連の設定がないので、保存期間は4週間になります。各設定項目の詳細は**表1**を参照してください。

Nginxの設定を見てみよう

リスト3は、Apacheと同様ですが、保存期間の設定を行っています。この例では、毎日切替、52回の切替後削除となりますので、52日間の保存期間となっています。その他の各設定項目の詳細は**表1**を参照してください。

MySQLの設定を見てみよう

MySQLは、rootのパスワードを**logrotate**に通知する必要があるため**/root/.my.cnf**にMySQLのrootパスワードを設定しなければなりません。そのため、デフォルトでは、全コメントアウトとなっています（**リスト4**）。

ほとんどのアプリケーション用設定ファイルについては、保存期間の変更以外は、デフォルトのままで良いと思います。気になる方は**表1**を参照のうえ、設定されるとよいでしょう。

ログローテーションをテストしてみよう

logrotateコマンドに**-d**パラメータを付けて実行すれば、設定に誤りがないか簡単にテストできます。

ヒント

ここで利用したパラメータは次のとおりです。

-d：デバッグ実行
-v：詳細表示
-f：強制的に実行
-m：メール送信のためのコマンドの指定
　例）**-m=/bin/mail**
-s：状態ファイルのパスを続けて指定
　例）**-s=/var/lib/logrotate.status**
一般的に、**-s**は違うユーザが**logrotate**を実行したい場合に、状態ファイルを分けることで混乱を防ぐために使用します。

▼リスト4　/etc/logrotate.d/mysqld

```
# This logname can be set in /etc/my.cnf
# by setting the variable "err-log"
# in the [safe_mysqld] section as follows:
#
# [safe_mysqld]
# err-log=/var/log/mysqld.log
#
# If the root user has a password you have to create a
# /root/.my.cnf configuration file with the following
# content:
#
# [mysqladmin]
# password = <secret>
# user= root
#
# where "<secret>" is the password.
#
# ATTENTION: This /root/.my.cnf should be readable ONLY
# for root !
# Then, un-comment the following lines to enable rotation of mysql's log file:¥
#/var/log/mysqld.log {
#        create 640 mysql mysql
#        notifempty
#       daily
#        rotate 3
#        missingok
#        compress
#    postrotate
#       # just if mysqld is really running
#       if test -x /usr/bin/mysqladmin && ¥
#          /usr/bin/mysqladmin ping &>/dev/null
#       then
#          /usr/bin/mysqladmin flush-logs
#       fi
#    endscript
#
```

logrotateの実行結果が正常な場合は、図2のような詳細情報が出力されるでしょう。図2から指定された設定ファイルに誤りはありません。ただし、“log does not need rotating”と出力されていますので“ログローテーションは必要なし”と判断されています。

このように“必要なし”となるのは、エラーやログファイルが空、あるいはログファイルが存在しない場合を除けば、ログローテーションを最後に実行して、まだ1日以上経過していないために判断されていることが多いです。その場合、先のログローテーション設定ファイルの内容に誤りがないかを確認するために、次のlogrotate状態ファイル[注1]で最終ログローテーション実施日をテスト用に変更して確認すると良いでしょう。

CentOSの場合、その状態ファイルは/var/lib/logrotate.statusです。ここではNginxの実施日を変更してみます。

ここの例では、

```
……（省略）……
"/var/log/nginx/access.log" 2014-5-26
……（省略）……
```

となっていたので次のように日付を1日前に編集し保存します。

```
……（省略）……
"/var/log/nginx/access.log" 2014-5-25
……（省略）……
```

保存後、再度logrotateコマンドを実行してみます（図3）。

今度は“log needs rotating”と出力され、“ロ

注1）通常、運用しているサーバでは、logrotate状態ファイルを直接変更しません。テスト用に状態ファイルをコピーし、logrotateコマンドの-sオプションでテストします。

▼図2　logrotateの実行結果（“ログローテーションは必要なし”）

```
$ logrotate -dv /etc/logrotate.d/nginx
reading config file /etc/logrotate.d/nginx
reading config info for /var/log/nginx/*.log

Handling 1 logs

rotating pattern: /var/log/nginx/*.log  after 1 days (52 rotations)
empty log files are not rotated, old logs are removed
considering log /var/log/nginx/access.log
log does not need rotating
considering log /var/log/nginx/error.log
log does not need rotating
not running postrotate script, since no logs were rotated
```

▼図3　logrotateコマンドの実行結果（“ログローテーションは必要あり”）

```
$ logrotate -dv /etc/logrotate.d/nginx
reading config file /etc/logrotate.d/nginx
reading config info for /var/log/nginx/*.log

Handling 1 logs

rotating pattern: /var/log/nginx/*.log  after 1 days (52 rotations)
empty log files are not rotated, old logs are removed
considering log /var/log/nginx/access.log
log needs rotating
considering log /var/log/nginx/error.log
log does not need rotating
rotating log /var/log/nginx/access.log, log->rotateCount is 52
dateext suffix '-20140526'
glob pattern '-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
previous log /var/log/nginx/access.log.1 does not exist
……（省略）……
removing old log /var/log/nginx/access.log.53.gz
error: error opening /var/log/nginx/access.log.53.gz: そのようなファイルやディレクトリはありません
```

グローテーションは必要"と判断されました。また、もしも設定情報に誤りがあった場合は、図4のようにerror情報が出力されます。

図4は、単純に'aaa'を/etc/logrotate.d/nginxのオプションに追記した場合のエラー情報です。ここでのエラーは、「'aaa'というオプションはありません」という意味になります。

ヒント ここではlogrotateのパラメータにNginxの設定ファイルを使用しました。これはほかの設定ファイルが省略パラメータが多いため、思ったとおりに動作しないためです。

たとえば、httpd(Apache)の設定ファイルでは、ログの切り替え、保存期間が省略されているため/etc/logrotate.confで設定されている値が引き継がれます。そのため、httpd(Apache)を正しく確認するには、

```
$ logrotate -dv /etc/logrotate.conf
```

と全体の設定ファイルを指定しないと正しくテストできません。それに対して、Nginxの設定ファイルは省略されていないので、そのまま設定ファイルを指定できます。

ログローテーションが実行される時刻を知りたい(おまけ)

logrotateは、インストールされた時点で、/etc/cron.daily/logrotate(リスト5)という名前のシェルスクリプトファイルもインストールされます。これは、ディレクトリ名からもわかるようにcronの日毎に実行されるスクリプトファイルです。

単純にlogrotateを実行しているだけですね。もし、失敗したらloggerコマンドでその旨をsyslogへ渡しています。

さて、このlogrotateは、先にも書いたように毎日実行されます。しかし、何時何分に実行されるのでしょうか。

CentOS 5までは、/etc/crontabに次のような記述がありました。

```
……(省略)……
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
……(省略)……
```

▼図4　logrotateコマンドの実行結果(設定に誤りあり)

```
$ logrotate -dv /etc/logrotate.d/nginx
reading config file /etc/logrotate.d/nginx
reading config info for /var/log/nginx/*.log
error: /etc/logrotate.d/nginx:2 unknown option 'aaa' -- ignoring line

Handling 1 logs

rotating pattern: /var/log/nginx/*.log  after 1 days (52 rotations)
empty log files are not rotated, old logs are removed
considering log /var/log/nginx/access.log
log needs rotating
considering log /var/log/nginx/error.log
log does not need rotating
…… (省略) ……
```

▼リスト5　/etc/cron.daily/logrotate

```
#!/bin/sh

/usr/sbin/logrotate /etc/logrotate.conf >/dev/null 2>&1
EXITVALUE=$?
if [ $EXITVALUE != 0 ]; then
    /usr/bin/logger -t logrotate "ALERT exited abnormally with [$EXITVALUE]"
fi
exit 0
```

つまり、これは日毎を指定したジョブはすべて4時2分に実行されることを意味しています。CentOS 5までは**logrotate**もこれに従い、毎日4時2分に実行されていました。しかし、CentOS 6では**/etc/crontab**にこの記述がなくなっています。

CentOS 6では、日毎のジョブは(通常の**cron**とは異なる)**anacron**(パッケージ名:**cronie-anacron**)で実行するようになっています。

リスト6のように、**anacron**の設定ファイル(**/etc/anacrontab**)では、日毎、週毎、月毎のそれぞれの実行するスケジュールが設定されています。

以降に日毎の設定に関して、簡単に解説しておきます。

9行目:これは、0分から45分の間でランダムに実行遅延時間を決めますという指定です。0指定の場合は、ランダム遅延なしとなります。

11行目:開始時刻の範囲は、3時から22時までの間で決めますという指定です。

14行目:**cron.daily**は、1日1回実行します。但し、5分だけは必ず遅延させます。

これらの設定であれば、普通に連続運転している場合、3時6分から3時51分の間(9行目のランダム実行遅延時間に14行目の遅延5分を足したもの、さらに、**cron**によって、時毎ジョブが毎時1分起動となっているため+1分としたもの)でランダムに決定した時刻に**cron.daily**(日毎のジョブ)が実行されることになります。**anacron**のすごいところは、サーバが停止しても再起動した時点で**cron.daily**を実行しようとするところです。

たとえば、CentOS 5では、4時2分にサーバが停止していた場合、5時にサーバが復旧したとしても日毎のジョブは実行されません。しかし、CentOS 6では、同じようにサーバが停止したとしても、5時にサーバ復旧した時点から1分後に**anacron**が起動し5分の遅延処理後、日毎のジョブを実行しようとします。つまり、ちょっとしたサーバの停止時間があっても日毎のジョブは、3時から22時までの間なら取りこぼしなく実行することになります。

CentOS5と同じ動作にするには?

先述のようにサーバが停止しなくても決まった時刻に実施されないことに違和感を覚える方や、会社のサーバなどでどうしても決まった時間に実行してほしい場合があると思います。その場合には次のような対策が考えられます。

CentOS 5までのように定刻にジョブを実行したい場合、

```
RANDOM_DELAY=0
```

とするとランダムな遅延は発生しませんから、上記の設定であれば3時6分に実行されます。分の微調整は、14行目の遅延時間の現行5分の

▼リスト6　/etc/anacrontab

```
01 # /etc/anacrontab: configuration file for anacron
02
03 # See anacron(8) and anacrontab(5) for details.
04
05 SHELL=/bin/sh
06 PATH=/sbin:/bin:/usr/sbin:/usr/bin
07 MAILTO=root
08 # the maximal random delay added to the base delay of the jobs
09 RANDOM_DELAY=45
10 # the jobs will be started during the following hours only
11 START_HOURS_RANGE=3-22
12
13 #period in days   delay in minutes   job-identifier   command
14 1       5       cron.daily              nice run-parts /etc/cron.daily
15 7       25      cron.weekly             nice run-parts /etc/cron.weekly
16 @monthly 45     cron.monthly            nice run-parts /etc/cron.monthly
```

ところを変更すれば良いです。**cron**によって、時毎ジョブが毎時1分起動となっているので、ここの設定を0と設定しても1分となります。設定分＋1分になることに注意してください。

もちろん、サーバが停止していた場合は、先に解説したように復旧後に直ちに実行しようとします。それもやめたい場合は、**cronie-noanacron**を使うほうが良いでしょう。その際は、必ず先に**cronie-noanacron**をインストールして、**cronie-anacron**を削除します。

```
$ yum install cronie-noanacron
……(省略)……
Is this ok [y/N]:y
……(省略)……
Complete!
$ yum remove cronie-anacron
……(省略)……
Is this ok [y/N]:y
……(省略)……
Complete!
```

これだけで、CentOS 5と同じようになります。

ログウォッチでログを毎日チェックしよう

先のlogrotateでログファイルがパンクすることはなくなりました。さあ、あとはログのモニタリング(監視)です。毎日**tail**コマンドでリアルタイムにログを眺めていられるほど時間を持て余しておられる方は、そうそういないと思います。もし、そういう方がいたとしても、ボーっと眺めているだけではだめで、ログ情報の中で問題がありそうなログを見つけ出す必要があります。寝る時間も必要ですし、そんなことを24時間できませんよね。

そこで毎日ログの内容を分析し、問題がありそうな怪しいログを検出してレポートしてくれるツールが**logwatch**です。**logwatch**のレポートは、(デフォルトの設定で)メールで送信されますから、管理者は、毎日、そのメールによるレポートを確認し、問題がないかチェックし、問題があれば、その対応を行うという作業の流れをルーチン化し、効率化を図ることができます。

では、簡単に**logwatch**のインストールから動作確認までを解説します。また、httpdやsyslogに関してはデフォルトのままで**logwatch**が自動でログの分析を行ってくれますが、Nginxの設定はありません。そのため、ここでは、Nginxを追加で分析するように設定する手順も合わせて解説します。

logwatch(Ver.7.3.6)をインストールしよう

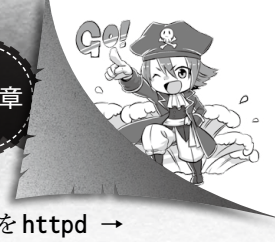
次のように**yum**でインストールします。

```
$ yum install logwatch
……(省略)……
Is this ok [y/N]:y
……(省略)……
Complete!
```

Column anacronの採用で負荷分散

CentOS 6で**anacron**が採用されたのは、もちろん、日毎、週毎、月毎のジョブの取りこぼしがないようするためもありますが、大きな要因は、仮想専用サーバ(VPS)における**cron**の負荷分散と言われています。

レンタルサーバの世界では、VPSは格安で専用サーバのように利用できることもあって大人気です。この人気のVPSにのっているOSはCentOSが非常に多いと言われています。VPSは、所詮複数のユーザで1台のサーバをシェア(共有)していることには変わりませんから、同じOSがのっている場合、ほとんど同じ時間に**cron**ジョブが実行されます。そうすると、CentOS 5では、日毎ジョブが実行される4時2分は異常に負荷が高くなりサーバが不安定になっていました。その昔どこかのレンタルサーバでは、cron設定を変更してくれ……と泣きのお願いメールがユーザへ送付される始末でした。今ではCentOS 6へ切り替わってそれに関連した問題は聞かなくなりましたから、**anacron**による負荷分散は成功しているのではないかと思います。



解析用のNginx用設定ファイルを作成しよう

もともとログ情報は、httpd(Apache)とほぼ同じですから、設定ファイルはhttpd(Apache)のものをコピー、編集して作成します。まずは、必要なファイルをコピーします(**図5**)。

logwatchのデフォルト設定ファイルは、**/usr/share/logwatch/**配下にあります。

そこから、httpd(Apache)用の設定ファイル、ログ設定ファイル、スクリプトファイルをNginx用にそれぞれコピーします。**図5**のように 個別のサービスログ(ここではNginx)を追加する際の各ファイルのコピー先は、**/etc/logwatch/**配下になります。

次にコピーした設定ファイル(**/etc/logwatch/conf/services/nginx.conf**)を次のように編集します。タイトルを**httpd → nginx**へ、ログ設定ファイル名を**http → nginx**へ変更します。

```
……(省略)……
#Title = "httpd"
Title = "nginx"
# Which logfile group...
#LogFile = http
LogFile = nginx
……(省略)……
```

次にコピーしたログ設定ファイル(**/etc/logwatch/conf/logfiles/nginx.conf**)を編集します。**リスト7**のように、httpd(Apache)の設定箇個所を行の先頭に"#"を挿入することでコメントアウトし、Nginxの設定を追記します。

LogFile:監視するログファイル名を指定します。
Archive:アーカイブ化されたログファイル名を指定します(ローテートされたファイル名)。

それぞれ、httpd(Apache)からNginxの設定に合わせて編集します。ここでは、いずれもワイルドカード"*"が使えます。

コピーしたスクリプトファイル(**/etc/logwatch/scripts/services/nginx**)は、怪しいログがないか分析するスクリプトになりますが、httpd(Apache)とログフォーマットが同じなので、そのまま使用できます。

▼リスト7　/etc/logwatch/conf/logfiles/nginx.conf

```
……(省略)……
# What actual file?  Defaults to LogPath ↵
if not absolute path....
#LogFile = httpd/*access_log
#LogFile = apache/*access.log.1
#LogFile = apache/*access.log
#LogFile = apache2/*access.log.1
#LogFile = apache2/*access.log
#LogFile = apache2/*access_log
#LogFile = apache-ssl/*access.log.1
#LogFile = apache-ssl/*access.log
LogFile = /var/log/nginx/*access.log
LogFile = /var/log/nginx/*access.log.1

# If the archives are searched, here is ↵
one or more line
# (optionally containing wildcards) that tell
where they are...
#If you use a "-" in naming add that as well -mgt
#Archive = archiv/httpd/*access_log.*
#Archive = httpd/*access_log.*
#Archive = apache/*access.log.*.gz
#Archive = apache2/*access.log.*.gz
#Archive = apache2/*access_log.*.gz
#Archive = apache-ssl/*access.log.*.gz
#Archive = archiv/httpd/*access_log-*
#Archive = httpd/*access_log-*
#Archive = apache/*access.log-*.gz
#Archive = apache2/*access.log-*.gz
#Archive = apache2/*access_log-*.gz
#Archive = apache-ssl/*access.log-*.gz
Archive = /var/log/nginx/*access.log.*.gz
……(省略)……
```

Nginxのlogwatchをテストしよう

図6のように**logwatch**コマンドを使って簡単に動作確認ができます。

logwatchコマンドのパラメータについて簡単に解説します。

--print:画面に出力します。
--service:サービス名を指定します。ここでは、nginxを指定しています。
--range:ログの解析範囲を指定します。ここ

▼図5　httpの設定ファイルのコピー

```
$ cp /usr/share/logwatch/default.conf/services/http.conf /etc/logwatch/conf/services/nginx.conf
$ cp /usr/share/logwatch/default.conf/logfiles/http.conf /etc/logwatch/conf/logfiles/nginx.conf
$ cp /usr/share/logwatch/scripts/services/http /etc/logwatch/scripts/services/nginx
```

▼図6　logwatchでNginxをテストする

```
$ logwatch --print --service nginx --range all

 ################### Logwatch 7.3.6 (05/19/07) ####################
        Processing Initiated: Mon May 26 13:33:16 2014
        Date Range Processed: all
      Detail Level of Output: 0
             Type of Output: unformatted
          Logfiles for Host: local65.rise43.com
  ################################################################

 -------------------- nginx Begin ------------------------

 Requests with error response codes
    404 Not Found
       /asdfasdfagasdgagoajoas.html: 1 Time(s)

 --------------------- nginx End -------------------------


 ####################### Logwatch End #########################
```

では、all(すべて)を指定しています。

図6の出力結果に第2章p.38でテストした404エラーが正しく検出されました。問題なさそうですね。

> **ヒント**
> **logwatch**の**cron**設定は、インストールした時点で、**/etc/cron.daily/0logwatch**という名前のシェルスクリプトファイルがインストールされますので、すでに毎日自動的に実行されるようになっています。そのため**cron**設定は不要です。

logwatchのレポートをチェックしていれば完璧というわけではありません。しかし、そのレポートにある小さな兆候も見逃さないようにしていれば、早期の問題の検出につながります。

おわりに

本稿を書くにあたり、少しでも初心者の方々にログに興味をもらってもらうために航海日記の話からsyslogの歴史なども織り交ぜながら「ログとは何？」を解説してみました。

インターネットによる利便性は、昨今の多発するクラックやそれに伴う改ざん、情報流出等々、さまざまな危険と隣り合わせです。その危険を減らすには、サーバを最新状態に保つとともにログの収集が不可欠です。しかし、そのログは近年重要性が増すとともに肥大化が進んでいます。さらには、クラウド化、ネットワーク分散化が進み、ログ収集・管理が非常に大事な要素になってきています。そこで、最近注目されているのが、あらゆるログの収集、解析・出力に柔軟に対応できるように設計されたfluentdです。fluentdは、syslogと異なりファシリティやプライオリティなどではなく最近のアプリケーションらしくタグで管理され、インプット、アウトプットの豊富なプラグインを用いてさまざまなログの収集、解析・出力に対応できるように設計されています。ここで紹介したsyslogも収集の対象にできます。

興味のある方は、ぜひ第6章をご一読ください。入門編を解説しています。さらに深く知りたい方は、Webや本などでも紹介されていますので、調べられると良いと思います。

少し脇道に逸れましたが、ログは問題検出の1つの手がかりであり、問題解決および改善のための貴重な情報です。また、より良いサーバ構築のヒントでもあります。これからますます重要性の高まるなか、本稿を読まれた皆さんがログをより有効活用されることを願っております。SD

第5章 MSP直伝・プロがやっているログ監視

Writer ㈱ハートビーツ　高村 成道(たかむら なりみち)
Mail takamura@heartbeats.jp

システムに不具合が発生したとき、ログは原因切り分けや障害検知に役立つ貴重な情報です。しかし、ログに出力されたメッセージをただ眺めているだけでは何がなんだかわかりません。そこで本章では、ログから障害原因を特定するコツを伝授します。また、障害検知のしくみであるログ監視について、MSP(Management Services Provider)の現場での経験を基に詳しく紹介します。

膨大なログから原因を掴むコツ

一言でログといっても、ログを出力をするソフトウェアによってその内容はさまざまです。また、ログファイルのサイズは、大きいものでは数百GB程度に達することもあります。そのような膨大な量のログから必要な情報を抽出することは簡単ではありません。闇雲に最初から最後まですべてに目を通していては日が暮れてしまいます。そのため、ログの確認はポイントを押さえて行う必要があります。ここでは、障害原因を調査する際のログ確認のコツを紹介します[注1]。

時刻で絞る

ログ確認の基本は、障害が発生した時刻付近のメッセージを確認することです。多くの場合、障害が生じた時刻の前後に原因となるエラーメッセージが出力されます。**図1**は、Linuxサーバのシステムログファイルに対して日付(6月4日)と時間指定(20:00～20:05)を指定して抽出する例です。なお、ログファイルを確認するときはシステムに負荷をかけないようにするために、**cat**や**less**コマンドでファイルの内容をすべてを出力するのではなく、**tail**や**head**コマンドでファイルの一部分だけを出力することも大切です。また、**nice**や**ionice**などを用いて、処理を低優先度に実行することで、より負荷を下げることができます(**図2**)。

ノイズを取り除く

ログファイルを確認するときに注意しなければならないのは、障害発生時刻に近いメッセージのすべてが障害原因であるとは限らないということです。障害時刻付近で確認したエラーメッセージが障害発生前から出力されており、それが障害発生後も継続して出力されていた場合、

注1) 実行環境：CentOS 6.5／Apache HTTP Server 2.2系／MySQL Community Server 5.5系／Python 2.6系

▼図1　tailコマンドによるメッセージ確認

```
$ sudo tail -n 1000 /var/log/messages | grep "Jun  4 20:0[0-5]"
Jun  4 20:01:20 test-01 dhclient[97109]: DHCPREQUEST on eth1 to 192.0.2.3 port 67
Jun  4 20:01:20 test-01 dhclient[97109]: DHCPACK from 192.0.2.3
Jun  4 20:01:21 test-01 dhclient[97109]: bound to 192.0.2.4 -- renewal in 15950 seconds.
Jun  4 20:04:31 test-01 ntpd[64182]: synchronized to 203.0.113.110, stratum 2
```

▼図2　niceコマンドで負荷軽減

```
$ sudo nice -n 19 ionice -c3 tail -n 1000 /var/log/messages | grep "Jun  4 20:0[0-5]"
```

それは単なるノイズ(障害に直接関係のないメッセージ)であることが多いです。見通しをよくしたり勘違いを防止するためにも、ノイズはgrepの-vオプションを用いて抽出対象外にすることをお勧めします(図3)。

キーワードで絞る

時刻による絞り込みだけの場合、同時刻に大量のログが出力されていた場合に困ってしまいます。そこで有効なのが、キーワードを用いた絞り込みです。

障害時に出力される文字列

ログ確認を行う時点ですでに障害原因があらかた推測できている場合、その障害のときに必ず出力される文字列というものがあらかじめわかります。この場合は、その文字列を基に検索するのが得策です。検索にヒットすればその障害が発生しているということがわかります。ヒットしない場合はその障害は発生していないということがわかります。たとえば、Apache HTTP Server(以降、Apache)のエラーログに対して"server reached MaxClients"と検索することでApacheが最大同時接続数に達していたかどうかがわかります(図4)。この確認方法はあくまでも小手調べ程度に用いるようにしましょう。というのも、この絞り方は検索文字列がとても具体的であるため、ほかの確認すべきメッセージもフィルタリングしてしまっている可能性が高いのです。同時に発生しているほかのエラーを見落とすことで本当の原因を特定できなくなってしまうというリスクもありますので、本章で紹介するほかの手法と組み合わせて使うようにしましょう。

エラー文字列

未知の障害が発生した場合に有効なのが、エラー時によく出力されるログレベルの文字列をキーワードにすることです。ログレベルは多くのソフトウェアで用いられています。それぞれの優先順位や意味する内容はソフトウェアごとに多少異なりますが、大まかには表1のとおりです。ソフトウェアごとにログレベルの文字列は多少異なります。Warningの場合、"Warn"のように省略して出力されることもあれば、"warning"のように小文字のみの場合もあります。そのため、ログレベルの仕様を把握するか、ヒットしやすいパターンを指定するようにしましょう。検索でヒットしやすくするためのお勧めはgrepの-iオプションです(図5)。このオプションを指定することで大文字小文字を区別

▼表1 ログレベル

ログレベル	意味
Fatal、Critical	致命的なエラー障害
Error	致命的なエラー情報またはエラー情報
Warning	警告情報
Info、Note、Notice	一般的な(操作)情報
Debug、Trace	デバッグ情報

▼図3 grep -vでメッセージ抽出

```
$ sudo tail -n 1000 /var/log/messages | grep "Jun  4 20:0[0-5]" | grep -v "dhclient"
Jun  4 20:04:31 test-01 ntpd[64182]: synchronized to 203.0.113.110, stratum 2
```

▼図4 "server reached MaxClients"で検索

```
$ sudo tail -n 1000 /var/log/httpd/error_log | grep "server reached MaxClients"
[Thu Jun 05 17:51:43 2014] [error] server reached MaxClients setting, consider raising the
MaxClients setting
```

▼図5 ログレベル"warn"を含むメッセージをgrep -iで抽出

```
$ sudo tail -n 1000 /var/log/hoge/applications.log | grep -i "warn"
```

しないで検索できます。

メッセージの量に着目する

エラーメッセージには何も出力されていないにもかかわらず、サイトが見られなかったりメールが送られなかったりとサービスに影響が出ている場合は、要求に対して単にシステムの処理性能が追いついていないことが原因かもしれません。その場合は、メッセージの量(行数)を比較してみましょう。ここでは、Apacheのアクセスログ(**図6**)を集計するコマンドとして、時刻で集計する場合とアクセス元IPアドレスで集計する例を紹介します。

時刻ごとにアクセスログを集計した場合、アクセスの増減を読み取ることができます。**図7**の例では、09:36付近でアクセスが急増していることがわかります。集計結果を可視化するとアクセスの様子がわかりやすくなります。ここでは**uniq -c**の結果を可視化するツールである**c2g**を紹介します[注2]。これを用いると、**図8**のように数字が棒グラフになります。**Cacti**や**Munin**などのモニタリングツールを導入することで、データの可視化をより強化できますが、今回は省略します。

アクセス元IPアドレスごとに集計をすると、どのIPアドレスからアクセスが多いかということがわかります。**図9**の例では、198.51.100.6からのアクセスが非常に多いということがわかります。特定のIPアドレスからの接続が極端に多い場合は、不正アクセスや検索エンジンロボットによるスキャンの可能性がありますので注意しましょう。

注2) https://gist.github.com/eidantoei/999146

▼図6　出力対象のApacheログ(サンプル)

```
198.51.100.100 - - [05/Jun/2014:09:36:47 +0900] "GET /redmine/stylesheets/application.css ➥
HTTP/1.1" 200 8903 "https://example.com/redmine/themes/farend_basic/stylesheets/application. ➥
css?1390511924" "Mozilla/5.0 (X11; Linux x86_64; rv:29.0) Gecko/20100101 Firefox/29.0"
```

▼図7　時刻ごとにアクセスログを集計(分単位)

```
$ sudo tail -10000 access_log | grep ➥
"05/Jun/2014" | cut -d ':' -f 2,3 | sort ➥
| uniq -c
     383 09:35
    3253 09:36
    1120 09:37
    1196 09:38
     933 09:39
     355 09:40
     348 09:41
     219 09:42
     370 09:43
     218 09:44
     313 09:45
```

▼図8　c2gによるアクセスログ増減の可視化

```
 383 [|||                              ] 09:35
3253 [|||||||||||||||||||||||||||||||| ] 09:36
1120 [||||||||||                       ] 09:37
1196 [||||||||||||                     ] 09:38
 933 [||||||||                         ] 09:39
 355 [|||                              ] 09:40
 348 [|||                              ] 09:41
 219 [||                               ] 09:42
 370 [|||                              ] 09:43
 218 [||                               ] 09:44
 313 [||                               ] 09:45
```

▼図9　アクセス元IPアドレスごとにアクセスログを集計

```
$ sudo tail -10000 access_log | grep "05/Jun/2014:09:36" | awk '{print $1}' | sort -n | ➥
uniq -c | sort
      6 198.51.100.1
      8 198.51.100.2
     34 198.51.100.3
     40 198.51.100.4
    130 198.51.100.5
   3000 198.51.100.6
```

比較対象の選び方

メッセージの量の比較対象は慎重に選びましょう。たとえば、セールか何かで1日中アクセスが多い場合、数分前のメッセージ量と比較してもアクセス増と判断できません。前日、3日前、1週間前など、通常のアクセス状況と比較することが大切です。サービスの利用ユーザの増加に伴い、通常のアクセスが多くなった場合にはサーバのチューニングやスペックアップをすることで対策しましょう。

現場での障害原因切り分け例

確認したログファイルから一度に原因が特定できれば楽なのですが、実際にはそんなに簡単にはいきません。ただ、根本的な原因ではないにしても障害には関係のある大切なログであることはたしかです。ここでは、これまでに紹介した方法が実際にMSPの現場でどのように使われているのか、例を基に説明していきます。

MySQL Community Server(以降、MySQL)が突然停止するという障害が起きました。このとき、まず確認するのがMySQLのエラーログです。障害時刻付近のメッセージを抽出した結果が**リスト1**のとおりです。ここではMySQLが停止したことが問題となっているため、それに関係のありそうなログを中心に見ていく必要があります。また、ERRORやFATALなどの文字列も見落とすことなく確認しましょう。このような基準でログを確認すると、「mysqldが停止していたので起動しようとしたが、メモリ確保ができなかったため、また停止した」ということがわかります。mysqldとはMySQLのプロセス名です。しかし、すでに1行目のログ出力の段階でmysqldが終了しているため、MySQLのログを確認しただけでは根本的な原因は不明なままです。しかし、2回目の起動はメモリが確保できないために失敗しているということがわかりました。ここからは予測ですが、もしかすると始めにプロセスが終了した原因もメモリ周りが関係しているかもしれません。そのため、次にシステムログを確認します。MySQLのエラーログにおいて1行目の時刻よりも前で、なおかつメモリに関係しそうなログを確認したところ、**リスト2**のようなログが出力されていました。

やはり、メモリ不足によって障害が発生していました。**リスト2**のログメッセージからでOOM Killer(Out of Memory Killer)によって

▼リスト1　/var/lib/mysql/hogehoge.com.err

```
 1  140612 20:42:09 mysqld_safe Number of processes running now: 0  // mysqld_safeがmysqld
    が0であることを確認
 2  140612 20:42:09 mysqld_safe mysqld restarted    // mysqldの起動を実施(すでにこの時点で終了
    している！ )
 3  140612 20:42:15 [Note] Plugin 'FEDERATED' is disabled.
 4  140612 20:42:15 InnoDB: The InnoDB memory heap is disabled
 5  140612 20:42:15 InnoDB: Mutexes and rw_locks use GCC atomic builtins
 6  140612 20:42:15 InnoDB: Compressed tables use zlib 1.2.7
 7  140612 20:42:15 InnoDB: Using Linux native AIO
 8  140612 20:42:15 InnoDB: Initializing buffer pool, size = 256.0M
 9  InnoDB: mmap(274726912 bytes) failed; errno 12
10  140612 20:42:15 InnoDB: Completed initialization of buffer pool
11  140612 20:42:15 InnoDB: Fatal error: cannot allocate memory for the buffer pool// "Fatal
    error"。buffer pool 用のメモリ確保に失敗
12  140612 20:42:15 [ERROR] Plugin 'InnoDB' init function returned error.
13  140612 20:42:15 [ERROR] Plugin 'InnoDB' registration as a STORAGE ENGINE failed. //
    InnoDBの登録失敗
14  140612 20:42:15 [ERROR] Unknown/unsupported storage engine: InnoDB
15  140612 20:42:15 [ERROR] Aborting        // 異常終了中
16  140612 20:42:15 [Note] /usr/libexec/mysqld: Shutdown complete    // mysqld停止
17  140612 20:42:15 mysqld_safe mysqld from pid file /var/run/mysqld/mysqld.pid ended
```

mysqldが強制終了させられていたことがわかりました。ここまで切り分けることができれば、あとは障害の再発防止をする対策を実施するのみです。このケースではメモリ使用量が高くなりがちなApacheとMySQLに対して、メモリやコネクション設定のチューニングを行うことで解決しました。

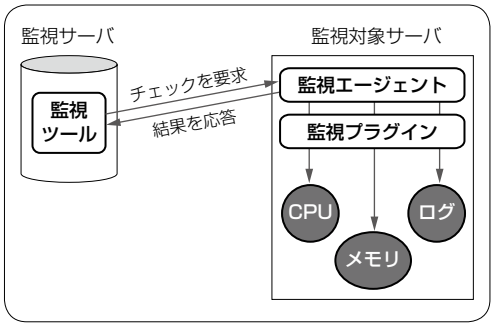
ログの情報を基に原因を切り分けることは障害対応において非常に重要なスキルです。ここでは、あるログから別のログへリンクするようなケースを例にとって説明しました。原因切り分けの手法は、複数のログファイルを確認する方法以外にもあります。個人的には、ソースコードに対してエラーメッセージを検索(grep)する方法がお勧めです。この方法によりライブラリのバグを検出できたこともあります。もちろん、怪しいと思われる文字列を基にググるという方法も、単純ですがとても有効な手段です。

ログから障害を検知する

ログメッセージに対して検索するパターンが決まれば、あとはそのパターンが出力されているかどうかを調べるだけです。いつでも障害を検知できるようにするためには、24時間365日ログを確認しなければなりません。そこで必要となってくるのが監視です。監視とは、対象のサービスやサーバの状態が正常であるかどうかを継続的に確認し、報告を行うことです。具体的には、数分ごとに監視プラグインを実行することでサーバのステータスを確認し、異常があればメールなどで通知します。ここでは、代表的な監視ツールである**Nagios**で監視することを前提に話を進めていきます。

それでは、ログ監視のしくみについて見ていきます。Nagiosでは、ログ監視のプラグインとしてcheck_logが用意されています。監視プラグインを用いた監視は**図10**のとおりです。

▼図10　監視プラグインを用いた監視



▼リスト2　/var/log/messages

```
 1 Jun 12 20:42:05 www kernel: [29204168.349016] httpd invoked oom-killer: gfp_mask=0x201da,
   order=0, oom_score_adj=0
 2 Jun 12 20:42:05 www kernel: [29204168.349033] httpd cpuset=/ mems_allowed=0
 3 Jun 12 20:42:05 www kernel: [29204168.349077] Call Trace:
 4 Jun 12 20:42:05 www kernel: [29204168.349090]  [<ffffffff8143eb2b>] dump_stack+0x19/0x1b
 5 Jun 12 20:42:05 www kernel: [29204168.349108]  [<ffffffff814449ba>] ? error_exit+0x2a/0x60
 6 Jun 12 20:42:05 www kernel: [29204168.349126]  [<ffffffff814444bb>] ? retint_restore_
   args+0x5/0x6
 7 Jun 12 20:42:05 www kernel: [29204168.349137]  [<ffffffff811191e9>] oom_kill_
   process+0x1a9/0x310
 8 Jun 12 20:42:05 www kernel: [29204168.349153]  [<ffffffff81208495>] ? security_capable_
   noaudit+0x15/0x20
 9 Jun 12 20:42:05 www kernel: [29204168.349161]  [<ffffffff81119939>] out_of_
   memory+0x429/0x460
10 ...
11 Jun 12 20:42:05 www kernel: [29204168.573052] Out of memory: Kill process 19632 (mysqld)
   score 78 or sacrifice child
12 Jun 12 20:42:05 www kernel: [29204168.573068] Killed process 19632 (mysqld) total-
   vm:1542800kB, anon-rss:132108kB, file-rss:0kB // mysqldを強制終了
```

check_log

check_log(**図11**)の実行例は次のとおりです。1つめの引数に、監視対象のログファイルを指定します。2つめは、一時ファイルを保存するパスを指定します。この一時ファイルには、1回前に監視プラグインを実行した時点でのログファイルが保存されます。3つめは、監視したいパターンを指定します。

```
$ ./check_log.sh -F logfile -O oldlog -q query
```

なお、監視プラグインの初回実行時には古いログファイルが存在しないため、初期化処理として事前にcheck_logを一度実行する必要があります。**図11**中の④の終了コードは、Nagiosがサーバステータスを決定する際に用いられます。0でない場合は、アラートを発報します。

より柔軟なログ監視

管理者の要望に合った監視をしようとした場合、check_logでは機能が足りないケースが出てきます。筆者がこれまで現場で経験してきたケースにはたとえば次のようなものがあります。

- サーバに負荷をかけずにログ監視をしたい
- 不要なアラート発報は避けたい
- 複数行のメッセージを監視したい
- ログローテートが有効な場合でもロスなく監視したい

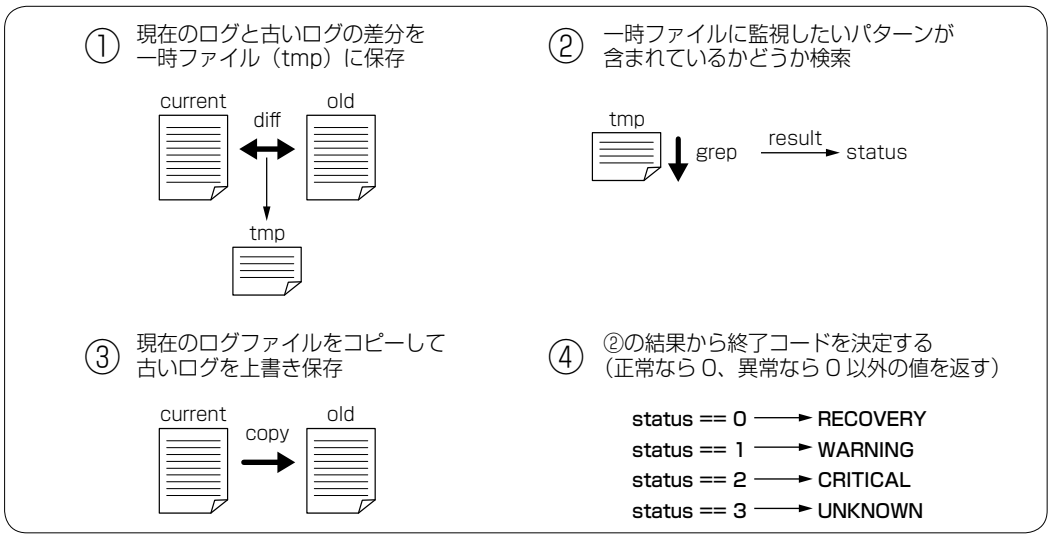
これらの要望を満たすために、筆者はcheck_log_ngという監視プラグインを新たに作成しました(**図12**)。これ以降は、現場での要望・問題をどのような方法で解決したのか紹介します。check_log_ngの詳しい使い方についてはGitHub[注3]を参照してください。

サーバ負荷を抑える

check_logでは、ログ監視を行う度に対象のログファイルをコピーしています。このしくみの場合、対象のログファイルのサイズが大きければ大きいほどサーバにI/O負荷がかかってしまい、本来優先すべきサービスに影響が出てしまいます。また、ログファイル1つ分、余計にディスク領域を消費することになります。これらの問題は、seekファイルを用いたログ監視を行うことで解決できます。seekファイルとは、監視終了時点の読み込み位置を記憶するファイ

注3) https://github.com/nari-ex/check_log_ng

▼図11　check_logのしくみ



ルです。擬似コードは**リスト3**のとおりです。

seekファイルを用いるログ監視は、対象のログファイルすべてを読み込まずに、前回の続きから検索を行います。そのため、毎回ファイルすべてを読み込むcheck_logと比べてI/O処理が減り、サーバへの負荷を軽減できます。

seekファイルを用いたログ監視において、1つ注意しなければならない点があります。それはseekファイルをむやみに更新してはいけないという点です。監視エージェント以外がseekファイルを更新してしまうと、その間の更新差分を監視エージェントでは監視できなくなってしまいます。テストや複数監視拠点からの並行監視を行う場合は、それぞれ個別のseekファイルを指定するようにしましょう。

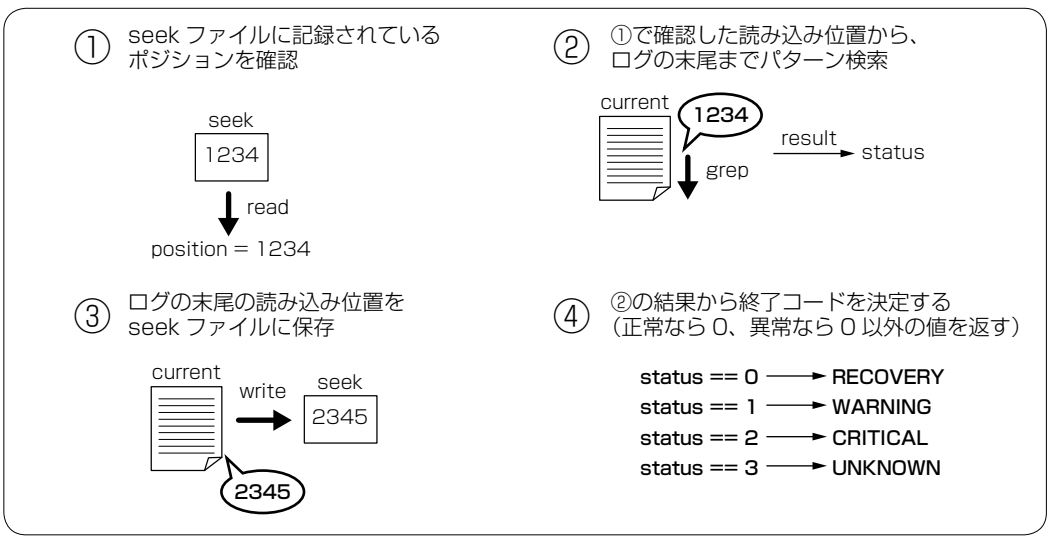
不要なアラート発報を抑制する

監視をするうえで困るのは、対応する必要の

▼リスト3　seekファイルを用いたログ監視

```
 1 STATUS = 3 # 0: RECOVERY, 1: WARNING, 2: CRITICAL, 3: UNKNOWN
 2
 3 def pattern_match(logfile, posision, pattern):
 4     f = open(logfile, 'r')
 5     f.seek(offset, 0)  # 読み込み位置を position まで移動
 6     status = 3
 7
 8     for line in f: # 1行ごとに処理
 9         # status に0～3の値を返す
10         result = 監視パターンマッチ(pattern, line)
11         ステータス更新(result) # resultの値によってSTATUSを変更
12
13     end_position = f.tell()   # 検索し終えた位置を取得
14     f.close()
15     return end_position
16
17 def check_log(seekfile, pattern, logfile):
18     start_position = シークファイル読み込み(seekfile) ←①
19     end_position = pattern_match(position, pattern)   ←②
20     シークファイル更新(end_position, seekfile) ←③
21     sys.exit(STATUS) ←④
```

▼図12　check_log_ngのしくみ



ないアラートを発報してしまうことです。「アラート発報したものの実際には対応不要だった」というケースは、ログ監視では珍しいことではありません。このようなことを再発させないために、対応不要なログメッセージについてはアラート発報を抑制します。たとえばApacheのエラーログ監視の場合、サービスに直接関係のない出力は無視したいことが多いです。

次のログは、ロボット型検索エンジンに対する命令を記述するためのファイルが存在しないということを意味しています。検索エンジンがサイトをスキャンするときにしかアクセスしないファイルですから、実サービスの表示に直接的な影響はありません。しかし、監視文字列が"error"の場合、検索エンジンがサイトをクロールするたびにアラートが発報されてしまいます。

```
[Mon Jun 09 12:02:23 2014] [error] ➥
[client 203.0.113.28] File does not ➥
exist: /var/www/html/robots.txt
```

このような問題を解決するために、監視プラグインに対して除外パターンを指定可能にしました。監視プラグインの変更内容は単純で、監視パターンマッチの前に除外パターンのマッチを行うというものです。擬似コードは次のとおりです。これは、**リスト3**の8～11行目を拡張するものです。

```
for line in f: # 1行ごとに処理
    # status に0～3の値を返す
    if not 除外パターンマッチ(negative_ ➥
pattern, line):
        result = 監視パターンマッチ➥
(pattern, line)
```

複数行対応

監視対象のメッセージにはさまざまな種類があります。それらがすべてsyslog形式とは限りません。一部のJavaアプリケーションでは、**図13**のように一度に複数行、メッセージが出力されます。**図13**のメッセージは2行目から7行目まで1つのエラーに対するログであるため、1行ごとに区別せず扱う必要があります。しかし、先ほどまでの監視実装では1行ごとにマッチングを行うため、このログに対して監視パターンと除外パターンの両方を指定して監視を行った場合、除外パターンが正常に動作しません。たとえば、監視パターンとして"ERROR"、除外パターンとして"Blocks should be registered"を指定した場合、"ERROR"を含む2行目には除外パターンが存在しないためアラートを発報してしまいます。

また、エラーごとにまとめるだけでは不十分な場合があります。たとえばこのアプリケーションでは、1行目のメッセージが出力されるときには必ず2行目のメッセージも出力されるとしましょう。この場合、1行目のメッセージにマッチするような除外パターンを指定したとしても、2行目でERRORの文字列があるためにアラート発報をしてしまいます。これらのメッセージはほぼ同時に出力されるため、エラーごとかつ時刻ごとにログをまとめることで対策ができます。

複数行のメッセージをまとめる際の基準については、ログ監視の仕様に沿って設定する必要

▼図13　複数行に渡るエラーメッセージ出力

```
1 2014-06-09 20:36:12 [ERROR] [ForgeModLoader] The following problems were captured during
this phase
2 2014-06-09 20:36:12 [ERROR] [ForgeModLoader] Caught exception from LogisticsPipes|Main
3 java.lang.IllegalArgumentException: Slot 882 is already occupied by ecru.MapleTree.block.
ecru_BlockMiniStairs@50ae8e0e when adding logisticspipes.blocks.LogisticsSolidBlock@64e6c903
4  Blocks should be registered before postInit for NEI to do proper conflict reporting
5         at codechicken.nei.IDConflictReporter.blockConstructed(IDConflictReporter.java:83)
6         at net.minecraft.block.Block.<init>(Block.java:347)
7         at net.minecraft.block.BlockContainer.<init>(SourceFile:9)
8         at logisticspipes.blocks.LogisticsSolidBlock.<init>(LogisticsSolidBlock.java:40)
```

があります。そのため、現在はメッセージをまとめるためのキーを正規表現で指定可能にすることで複数行のメッセージの対応をしています。

フォーマットは、正規表現の組を指定します。はじめの正規表現がキーとなり、もう1つのほうがメッセージとして扱われます。たとえば、先ほど例に挙げたメッセージの場合は

```
^(%Y-%m-%d %T \[\S+\]) (.*)
```

のようなフォーマットを指定します。これにより、1行目については、"**2014-06-09 20:36:12 [ERROR]**"がキー、それ以降の文字列がメッセージとなります。なお、パターンマッチは、キーとメッセージの両方に対して行われます。このパターンにマッチしなかった場合、その文字列はメッセージとして扱われます。その際のキーは、前回までのキーを使いまわします。これによりキーごとに正しくパターンマッチを行うログ監視を実現しています。なお、メッセージをまとめる場合、まったく関係のないメッセージをまとめてしまう危険性もありますので、フォーマットの指定には十分気をつけましょう。

擬似コードは**リスト4**のとおりです。このコードは、先ほどの**リスト3**のコードの8～11行目を拡張するものです。forループ内では、一つ前のキー(pre_key)を保持しており、このpre_keyと行ごとのキー(cur_key)を比較することで処理を分岐しています。pre_keyは1つ前のループでセットされます。cur_keyについてはループの先頭(2～8行目)でセットされます。

2つのキーが一致する場合は、メッセージをまとめる必要があります。そのため、メッセージをバッファに追加し監視パターンマッチを行わずにループの先頭に戻ります。2つのキーが一致しない場合は、エラーメッセージの区切りを意味しますので次の行とは区別して扱う必要があります。そのため、保留していたバッファをまとめ、監視パターンマッチをします。その後、変数を初期化処理してループの先頭に戻ります。なお、ループ前の変数の初期化やループ

▼リスト4　パターンマッチを使ったログ監視

```
 1  for line in f:
 2      # 現在のキーとメッセージをセット
 3      if フォーマットに一致:
 4          cur_key = m.group(1)
 5          cur_message = m.group(2)
 6      else:
 7          cur_key = pre_key
 8          cur_message = line
 9  # pre_keyが存在しない場合、pre_keyに現在のキー、バッファに行全体をセットしてループの先頭へ戻る
10      if pre_key is None:
11          pre_key = cur_key
12          line_buffer.append(line)
13          continue
14
15      # 現在のキーと1つ前のキーが等しい場合、メッセージをバッファに追加する
16      if cur_key == pre_key:
17          line_buffer.append(cur_message)
18      # 現在のキーと1つ前のキーが異なる場合、監視パターンマッチを実行
19      else:
20          # バッファに保留していたメッセージをまとめる
21          message = ' '.join(line_buffer)
22          if not 除外パターンマッチ(negative_pattern, message):
23              result = 監視パターンマッチ(pattern, messsage)
24              ステータス更新(result)
25          # 次回のループのために変数を初期化する
26          pre_key = cur_key
27          line_buffer = []
28          line_buffer.append(line)
```

を抜けたあとのバッファの処理については省略しています。

複数ファイル対応

ログローテートが行われる場合、監視はローテート前とローテート後のファイルの両方をケアする必要があります。監視と監視の間にログローテートされた場合、古いほうのログファイルに検知対象の出力がないか見落とさないようにするためです。ローテートしたログファイルのサフィックスは数字や日付などさまざまです。

この問題を解決する方法はシンプルで「古いほうのログファイルも監視する」ということになります。そのため、check_log_ngでは複数のログファイルに対して更新日時を確認することで対応しています。複数ログ指定はglobを用いて行います。たとえば、システムログを指定する場合は

```
/var/log/messages*
```

のように指定します。末尾にアスタリスクを付与することで、"/var/log/messages"で始まる任意の長さのファイル名が対象となります。これにより、"/var/log/messages"に加え"/var/log/messages-20140618"や"/var/log/messages.1"といった名前のファイルも監視対象となります。

実装は、check_log関数のラッパー関数を追加することでできます。globで指定されたパターンにマッチするファイルのリストを取得し、それらすべてに対してcheck_log関数を実行します。これだけでも動きますが、ローテートした古いログファイルすべてをスキャンするのは負荷のことも考えると危険です。そのため、指定した時間(デフォルト：1日)よりも前に更新されたファイルはcheck_log関数の実行を終了するようにしています(**リスト5**)。なお、ローテート後にseekファイルが重複しないようにするために、ファイルのinodeを基にseekファイルを生成するオプションも実装してありますが、それについては割愛します。

まとめ

いかがでしたでしょうか。ログは保存しているだけでは何の効果も発揮しません。ログから必要な情報を得ることで初めて役に立ちます。この記事を通して、ログを用いた障害検知や原因切り分けの手助けができれば幸いです。

ログ監視については、現場で実際にあった要望とそれ対する解決策についても紹介しました。監視プラグイン、ログ監視のパターンやロギングの設定を調整することで、自分の環境に最適なログ監視を実装しましょう。SD

▼リスト5　複数ファイルのログファイル監視

```
def check_log_multi(seekfile, logfile_pattern, neg_pattern, pattern, logfile):
    # 対象のログファイルを指定されたファイル名のパターンを基に取得
    logfile_list = get_logfile_list(logfile_pattern)
    # ファイルごとにcheck_log関数を実行
    for logfile in logfile_list:
      check_log(seekfile, pattern, neg_pattern, logfile)

def check_log(seekfile, pattern, neg_pattern, logfile):
    start_position = シークファイル読み込み(seekfile)

    # 更新されていない古いログはチェックを行わない
    if ファイルの最終更新時刻 < (現在時刻 - ユーザ指定の時間):
        return False

    end_position = pattern_match(position, pattern, neg_pattern)
    シークファイル更新(end_position, seekfile)
    sys.exit(STATUS)
```

第6章

フロントエンジニアもFluentd＋MongoDBで実践! 小さく始めるログ活用のすすめ

Writer 羽田 健太郎（はねだ けんたろう） Retty㈱
Twitter @jumbOS5

最近何かと話題のFluentdという技術があります。「ログデータがどうとか、何かと面倒なんでしょう？」という声を聞いたりもしますが、それは本当に勘違いです。“すぐ使えるログ活用”ということで、本章ではFluentd＋MongoDBで複数のログを1ヵ所に集計するシステムの構築手順やtips、活用例などを、Retty㈱での経験をもとにお話します。

Fluentdについて

使ってますか？ ログ

Retty㈱の羽田と申します。実名性グルメサービスRettyでスマホエンジニアをやっています。スタートアップの企業ではスマホエンジニアと言えど、自分でAPIを書いたりインフラ周りを手伝ったりと、分野を横断してタスクが振ってくることもしばしば……。

本稿はインフラ担当じゃない筆者がFluentdを使い、複数のサーバから出るログデータをログサーバ（MongoDB）に集計し、それを使うためのクエリを書いて利用した経験から、その過程で考えたことや運用の注意点などを紹介します。インフラ以外のフロントエンジニアの方や、ディレクタの方にもログに興味を持ってもらうことを目的とした内容ですので、経験者の方には退屈な記事になってしまうかもしれませんがご容赦ください。

Fluentdとは？

Webサービスを運用するサーバの維持には、サーバの増減やその管理に日々の多くの時間が割かれています。現在弊社では、AWS（Amazon Web Services）の登場により、台数の管理や増減にかかるコストは下がったものの、運用やそこに集まるサーバごとのログ情報を活用する手段を模索中です。ログを活用するには、それらのデータを一元的に管理できる別の機能が必要になります。そういった複数個所にたまるログなどのデータを一元的に処理するために生まれたシステムがFluentdです。

Fluentdは Treasure Data社[注1]の開発したログ管理ツールで、オープンソースであり、さまざまな環境で動作することから近年いろいろなところで活用事例が報告されています。

・はてなブログ
https://speakerdeck.com/shibayu36/fluentd-mongodb-kibanawoli-yong-sitahatenaburoguabtesutofalseshi-li

・COOKPAD
http://www.slideshare.net/hotchpotch/20120204fluent-logging

・NTTPCコミュニケーションズ
http://www.slideshare.net/keithseahus/big-datafluentd

大小さまざまな企業がこぞってFluentdを利用している理由には、次のようなことがあると考えています。

注1） http://www.treasuredata.com/jp/

Fluetndの特徴

[1]導入が容易

本稿後半で説明するtd-agent(Fluentdの安定版)を用いたログの集計でも説明しますが、Fluentdは導入がとても容易で、confファイルにも小難しいことは書かずに運用が始められます。

[2]カスタマイズ性に富んでいる

Fluentdにはさまざまなプラグインがあり、その導入もgemで管理されているため拡張性に富んでいます[注2]。Amazon S3や各データストアとの連携、ログを飛ばす前にデータを加工したり条件を追加するといったドキュメントや事例も、Web上に多くあります。プラグインはRubyで記述されているため、Rubyが書ければ自分のほしい機能も自作して導入できます。

[3]ログの集計と構造化

ログが使われない理由として集計が面倒だったり、それに見合うだけのメリットがないと言われてきました。しかしFluentdの登場により、容易に構造化されたログを取り出すことができるようになり、それと連携した可視化のアプリケーションが増え、生のデータでもユーザに非常に使いやすい形で扱えるようになりました。

サービスを成長させるために必要なデータを、もっとエンジニアと近づけることのできる最初の一歩として、Fluentdは最も適した技術だと筆者は考えています。

Webにおけるログの活用

ログを使う

Webサービスにおけるログとはどんなものがあるでしょうか？　ユーザのアクセスログ、各プロセスの吐き出すログ、エラーログ、スロークエリ……1つのサーバだけでも大量のデータがストリームで流れていきます。これらのデータは止まることなく、サービスの運用とともに増加し、サービスの成長とともにその加速度も増します。こういったデータを1つ1つサーバごとに見ていく時間は、どこの企業にもないでしょう。

Rettyも例外ではなく、社内において「SEOの観点から各Webサーバ(Apache)のアクセスログ、それとレガシーなコードを改善するためにエラーログを集計・解析できる体制を作りたい」という要望があがりました。そこで普段はスマホエンジニアの筆者ですが、環境の構築と設定を担当することになりました。

とりあえず一番イケてる方法を試してみた

Fluentd + Elasticsearch[注3] + Kibana[注4]でイケてる感じのシステムを組んで、ログをグラフィカルに表示させてみたりといろいろ試してみたのですが……「これは別にほしくないし、グラフの活用法があまり見えない」と言われてしまい。ここらあたりの技術はちょっと試す分にはオーバースペックなんだと感じました。

とりあえずやってみよう程度の期待値で始まったプロジェクトのため、あまり時間も割けずにどうすればいいんだろう……と悩み、インフラ部隊と相談して決まった方針が「コマンド一発で必要な数字が出せる状態にすべき」ということでした。

集計しておくべきログ

どんな小さなWebサービスでも、自分でサーバを持っている限りログは出力されます。上記のようにさまざまなログが出力される中で、私たちWebサービスエンジニアがとくに注目すべきログには何があるでしょうか？　筆者はWebサーバのアクセス／エラーログだと思います。

注2) http://fluentd.org/plugin/
注3) http://www.elasticsearch.org/
注4) http://www.elasticsearch.org/overview/kibana/

アクセスログ

Apacheのアクセスログは**図1**のように、いつ、どこから、どこにアクセスがあったかを見ることができます。出力される情報の形式は**リスト1**のようになっていて、httpd.confに記述されています。細かいプロパティの説明は省きますが、リモートホストとそのアクセス時間、レスポンスプロパティ、ユーザエージェントなどです。そのほかにもリクエストのメソッド名や応答時間、リクエストに含まれるヘッダやレスポンスのヘッダの情報などもログとして出力することができます。

アクセスログにはユーザのページごとのアクセスや、アクセス元の情報が入っています。たとえばここから、Googleのbotがアクセスしてきているかなどを読み取ることができます。

エラーログ

サービスによってログ出力の規約などは異なるとは思いますが、Apacheのエラーログにはエンジニアの意図していないクエリの結果や、処理結果がたくさん詰まっています。ここを見過ごしながらサービスを成長させるのは、負債が増えていくのと同じです。エンジニアにはこのエラーログを読み取り、コードやサービスの抱える問題を判断し、処理することが求められます。

▼リスト1　httpd.confに記載されているログ設定例

```
LogFormat "%h %l %u %t \"%r\" %>s %b ➥
\"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

Fluentd＋MongoDBという選択

Rettyでは「ログの可用性」というところに注目して、ログサーバのデータストアにはMongoDBを使っています。理由としてはCapped Collection（後述）の存在とAggregation.Framework（後述）のようなクエリの多様性、それに加えて既存の取り組み事例が多く、すぐに導入できそうだったことが挙げられます。

Apacheのエラーログとアクセスログを集計できる環境を1〜2週間程度で仕事の合間に作りました。実際に作業にあてた時間は非常に少ないので、環境構築だけなら実質2、3日あれば十分だと思います。一番時間がかかったのはFluentd関連のconfの設定と、ログを取り出すクエリを書いたり、ログの形式を検討することでした。

それでは次から設定の手順を説明します。

FluentdとMongoDBの導入

導入手順

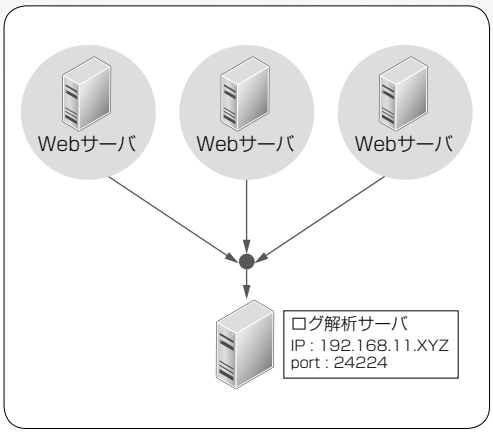
図2のように、複数のWebサーバからログ解析サーバにログを送信するような状況を仮定して設定を行います。送信側と受信側各々に必要なツールの導入と、その設定手順を述べていきます。本稿での実行環境はCentOSです。用いるツールとしてはFluentdの安定版のtd-agent[注5]を使います。

注5） https://github.com/treasure-data/td-agent

▼図1　アクセスログ（例）

```
XXX.XXX.XXX.XXX - - [11/May/2014:20:54:03 +0900] "GET /appRank/api/public/ranking/favorite/853444791 HTTP/1.1"➥
200 2393 "http://xxxxxxxxxxxxxyyxxxxxxxxx.com" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2)➥
AppleWebKit/537.75.14 (KHTML, like Gecko) Version/7.0.3 Safari/537.75.14"
XXX.XXX.XXX.XXX - - [12/May/2014:00:51:15 +0900] "GET /muieblackcat HTTP/1.1" 404 210 "-" "-"
XXX.XXX.XXX.XXX - - [12/May/2014:00:51:15 +0900] "GET //phpMyAdmin/scripts/setup.php HTTP/1.1" 404 226 "-" "-"
XXX.XXX.XXX.XXX - - [12/May/2014:00:51:16 +0900] "GET //phpmyadmin/scripts/setup.php HTTP/1.1" 404 226 "-" "-"
XXX.XXX.XXX.XXX - - [12/May/2014:00:51:16 +0900] "GET //pma/scripts/setup.php HTTP/1.1" 404 219 "-" "-"
XXX.XXX.XXX.XXX - - [12/May/2014:05:24:29 +0900] "HEAD / HTTP/1.0" 200 - "-" "-"
```

▼図2　Fluentdのためのサーバ構成



※サーバOS：CentOS release 6.4 (Final)

送信側（Webサーバ）手順

[手順1] td-agentの用意

次のようにviで/etc/yum.repos.d/td.repoを作成し、**リスト2**の内容に編集します。

```
$ vi /etc/yum.repos.d/td.repo
```

作り終えたら、次のようにtd-agentをインストールします。

```
$ sudo yum install -y td-agent
```

[手順2] pos用のディレクトリの用意

図3のように任意のディレクトリにaccess.posとerror.posを作成します（本稿では/var/lib/fluent/に置くこととします）。そして、それぞれの権限を書き込みができるよう変更してください。

posファイルに読み込んだ位置を記録しておくことで、td-agentを再起動した場合でも前まで読み込んだ位置から再開してくれます。これを指定しない場合はFluentdの起動時に、td-agentのログに警告が出力されます。

[手順3] td-agent.confの書き換え

この設定は、後述するプラグインを導入する個所で述べます。

▼リスト2　/etc/yum.repos.d/td.repo

```
[treasuredata]
name=TreasureData
baseurl=http://packages.treasure-data.➥
com/redhat/$basearch
gpgcheck=0
```

▼図3　posファイルの作成と権限の変更

```
$ sudo touch /var/lib/fluent/access.pos
$ sudo touch /var/lib/fluent/error.pos
$ sudo chmod 777 -R /var/lib/fluent/
```

▼リスト3　/etc/yum.repos.d/10gen.repo

```
[10gen]
name=10gen Repository
baseurl=http://downloads-distro.mongodb.➥
org/repo/redhat/os/x86_64
gpgcheck=0
enabled=1
```

受信側（ログ解析サーバ）手順

[手順1] MongoDBのインストール

次のようにviで/etc/yum.repos.d/10gen.repoを作成し、**リスト3**の内容に編集します。

```
$ vi /etc/yum.repos.d/10gen.repo
```

作り終えたら、次のようにMongoDBをインストールします。

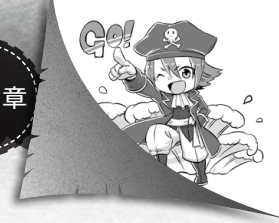
```
$ yum update
$ yum install mongo-10gen mongo-10gen-➥
server
```

試しに起動してみましょう。mongoのプロセスをバックグラウンドで起動するには--forkのオプションが必要になります。

```
$ mkdir /data/db
$ mongod --fork --logpath /var/log/log
$ mongo
```

[手順2] portの解放

UDP、MongoDBで用いるポート群の解放を行います。詳細は割愛します。



[手順3]td-agentのインストール

送信側の手順1と同じ操作でtd-agentをインストールします。

以上がconfファイルの記述を除いた、必要なモジュールのインストールと設定です。

次は、必要なプラグインの説明とconfファイルの設定についてです。

プラグインを導入する

td-agentを導入するとバンドルされたgemがついてくるので、これを用いて管理を行います。試しに次のようなコマンドを叩いてみると、今入っているプラグイン一覧が見られるはずです。

```
$ /usr/lib64/fluent/ruby/bin/fluent-gem ➥
list |grep fluent
```

便利なプラグインが豊富にありますが、本稿ではデータを使うために最低限必要なコマンドだけを入れます。MongoDBにデータをストアするためのプラグイン「fluent-plugin-mongo[注6]」です。

標準のFluentdだとテキストファイルにダンプすることしかできないので、jsonで送られてくるデータを構造化して管理するためにこのプラグインを使います。

td-agent.confを書くための前知識

td-agent は /var/etc/td-agent/td-agent.conf のファイルの設定を起動時に読み込んでデータの管理をします。実際の設定内容を解説する前に、ここでconfの記述の基本的なルールであるインプットプラグインとアウトプットプラグインの説明をしましょう。

インプットプラグインに関する仕様と設定

インプットプラグインの設定は次のようにsourceディレクティブに記載します。

```
<source>
  type インプットプラグインの種類の指定➥
(tail、execなど)
  その他パラメータ(利用するインプットプラグ➥
インに応じて必要なパラメータを追加指定)
  tag タグを設定する
</source>
```

アウトプットプラグインに関する設定

アウトプットプラグインの設定は、次のようにmatchディレクティブに記載します。

```
<match タグパターン>
  type アウトプットプラグインの種類の指定➥
(file、stdoutなど)
  その他パラメータ(利用するアウトプットプラ➥
グインに応じて必要なパラメータを追加指定)
</match>
```

matchの後ろにタグのパターンを指定し、特定のタグが付与されたログデータをどのアウトプットプラグインで処理するかを指定します。

td-agent.confの設定

送信側(Webサーバ)

送信側のtd-agent.confは、**リスト4**のようにします。

①：tailプラグインを使ってApacheのアクセスログのファイルの更新をフックし、formatに指定した正規表現にマッチする行であればログとして出力します。Apacheのログフォーマットがデフォルトのフォーマットのままであれば"Apache"と記述しても使えます。

②：tagには、アクセスログであることと、どのサーバからのログかわかるようにaccess.web1のようにつけました。エラーログに関しても同様です。

③：最後にログを集約するログサーバの設定として、portやhostなどの設定を記述します。

このconfファイルを各サーバごとに用意して、web1の部分の設定のみ各サーバで固有のタグ

注6) https://github.com/fluent/fluent-plugin-mongo

をつけるようにします。正規表現の記述が間違っていたりするとtd-agentの起動に失敗します。失敗しても起動時には何もエラーが出ないので、起動しないようなら/var/log/td-agent/td-agent.logを確認してみましょう。

受信側(ログ解析サーバ)

受信側のtd-agent.confは、**リスト5**のようにします。

①：forwardを最初に指定することで、受信側のログサーバとして24224のポートでログを受け取ります。

②：今回は使いませんが、confファイル中でincludeを用いて外部ファイルの設定を読み込むことができます。たとえばサーバごとに異なる設定を書きたい場合など、confファイルが見にくくなるのをさけるためにファイルを分けるといった利用が可能です。

③：アクセスログの受信部です。各サーバからaccess.web1、access.web2……などのタグを持ったログが入ってきます。

④：mongo-pluginによってMongoDBへのストアが可能になっているので、typeとして指定して、用いるデータベースとコレクションの指定を行います。コレクションはMySQLでいうところのテーブルに相当します。

⑤：include_tag_keyを指定することにより、ログについているtagをストアされるレコードに"tag"というキーで付与することができます。これで同じコレクション内に複数のサーバからのログを収集しても、それぞれがどのサーバからのログなのかを識別することが可能です。

⑥：インストールしたMongoDBのportなどの設定をここでします。

⑦：エラーログもアクセスログと同様に記述します。ストア先のコレクションのみ別のも

▼リスト4　送信側のtd-agent.conf

```
# 送信側の/etc/td-agent/td-agent.confの設定
# アクセスログの設定
<source>
# ①
  type tail
format /^(?<host>[^ ]*) [^ ]* [^ ]* [^ ]* \[(?<time>[^\]]*)\] "(?<method>\S+)(?: +(?<path>[^ ]*) ↵
+\S*)?" (?<code>[^ ]*) (?<size>[^ ]*) (?<restime>[^ ]*)(?: "(?<referer>[^\"]*)" "(?<agent>[^\"]*)")?$/

time_format %d/%b/%Y:%H:%M:%S %z
  pos_file /var/lib/fluent/access.pos
  path /var/log/httpd/access_log
# ②
  tag access.web1
</source>

# エラーログの設定
<source>
  type tail
  format /^\[[^ ]* (?<time>[^\]]*)\] \[(?<level>[^\]]*)\] (?<message>.*)$/
  time_format %b %d %H:%M:%S %Y
  path /var/log/httpd/error_log
  pos_file /var/lib/fluent/error.pos
  tag error.web1
</source>

# ③
<match *.web1>
  type forward
  flush_interval 3s
  <server>
  # ログサーバのhostを設定
  host 192.168.11.XYZ
  port 24224
  </server>
</match>
```

のにします。
⑧：正規表現であてはまらなかったログなどが出力されます。

confファイルは頻繁に書き換えたりするのでGitで管理するといいでしょう。状況に応じてbranchを切り替えて、使うconfを変えるのもいいでしょう。そのときにはtd-agentを再起動することを忘れずに。

使ってみる

必要なモジュールのインストールも完了し、confも書き終えたらいよいよ起動しましょう。MongoDBの起動を確認してから、送受信側の両方で次のようにtd-agentを起動します。

```
$ sudo service td-agent start
```

起動に失敗している場合、

```
$ sudo service td-agent status
```

とコマンドを打つと、“td-agentが停止していますがPIDファイルが残っています”のようなメッセージが出ます。このときは/var/log/td-agent/td-agent.logを確認してみてください。confの記述を間違えていたり、正規表現がログの形式と合っていない場合などが原因でtd-agentのプロセスが落ちます。またログファイルへのアクセス権限があるかどうかの確認も忘れずに。

これまでの設定により、たまるログは**リスト6**のようになっています。これでjsonデータがWebサーバから逐次送られてくるようなシステムが構築できました。

▼リスト5　受信側のtd-agent.conf

```
# 受信側の/etc/td-agent/td-agent.confの設定
# 送信側の設定を受信側でも設定しておく（口を開けっ
ておくイメージ）

# ①
<source>
  type forward
  port 24224
</source>

# ②
# include web_server.conf/*.conf

# ③
<match access.*>
  # plugin type
  # type mongo_replset
  type mongo

# ④
  # mongodb db + collection
  # dbとcollectionを指定
  database log
  collection access

# ⑤
  # set tag_name
  # 送信側で付けたタグをkey値でセット
  include_tag_key true
  tag_key tag

# ⑥
  # mongodb host + port
  host localhost
  port 27017

  # interval
  flush_interval 10s
  buffer_chunk_limit 10m
</match>

# ⑦
<match error.*>
    # plugin type
    #type mongo_replset
    type mongo

    # mongodb db + collection
    database log
    collection error

    ## set tag_name
    include_tag_key true
    tag_key tag

    # mongodb host + port
     host localhost
     port 27017

    # interval
    flush_interval 10s
    buffer_chunk_limit 10m
</match>

# ⑧
<match **>
  type file
  path /var/log/td-agent/no_match.log
</match>
```

▼リスト6　たまるログ（抜粋）

```
●アクセスログ例
{ "_id" : ObjectId("53044f6b1ed75a0e83b8f278"), "time" : ISODate("2014-02-19T06:29:54Z"), ➥
"host" : "XXX.XXX.XXX.X", "method" : "GET", "path" : "/images/topics/markers/marker10.png", ➥
"code" : "200", "size" : "5078", "restime" : "1045", "referer" : "http://retty.me/area/PRE13/➥
STAN5888/PUR1/", "agent" : "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)", ➥
"tag" : "access.web1" }

●エラーログ例
{ "_id" : ObjectId("5378d5c41ed75a365660308d"), "time" : ISODate("2014-05-18T15:46:06Z"), ➥
"level" : "error", "message" : "PHP Notice:  Undefined index: HTTP_HOST in /var/www/html/➥
public/index.php on line 61", "tag" : "error.web1" }
```

ログの活用

何を取り出すのか

ここまでの手順で常に複数サーバからのログがログ解析サーバに蓄積されるようになりました。次はこのログからデータを取り出してみましょう。MongoDBからデータを取り出すにはMongoQueryを書きます。

JavaScriptで記述され、MySQLのようにカラムが一様でないような情報に対しても柔軟にアクセスできるフレームワークが用意されています。詳しいクエリはリファレンス[注7]を読んでもらうとして、ここでは頻繁に使うクエリのみ抜粋します。

【MongoDBの基礎的なクエリ】

●document数の取得

```
> db.access.count();
```

●access collectionのdocumentの取得

```
> db.access.find();
```

●最新の10件の取得

```
> db.error.find().sort({time:-1}).➥
limit(10);
```

●web1サーバのタグの情報のみ取得

```
>db.error.find({"tag":"error.web1"})
```

●messageを“HTTP_HOST”という単語でgrepして検索

```
> var grepQuery = new RegExp("HTTP_HOST");
> db.error.find({message:grepQuery});
```

●指定時間内のクエリを取得

```
> db.error.find({time:{$gt:ISODate➥
('2014-05-18 00:00:00'), $lt : ➥
ISODate('2014-05-18 24:00:00')}})
```

●Aggregation.Frameworkによるクエリ（指定時間内のmessageごとの行数を集計する、MySQLにおけるgroupbyのようなもの）

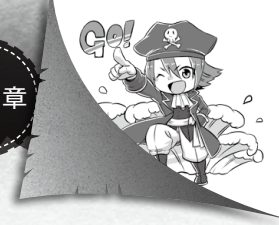
```
> var start = ISODate('2014-05-18 ➥
00:00:00');
> var end = ISODate('2014-05-18 ➥
24:00:00');
> db.error.aggregate([
  {$match :{time:{$gte:start,$lt:end}}},
  {$group : { "_id" : "$message" , ➥
"count" : { $sum : 1} }},
  { $sort : { count : -1 } },
  ]);
```

Rettyにおけるエラーログの集計と活用

Rettyでは、ログのメッセージに記述されたログ情報から重要度を判別し、その行数をカウントしています。**リスト7**のようなコードで、指定の文言をmessageに含むドキュメントをカウントアップするようなコードを用いています。

こうすることで過去に追加されてしまった良

注7）http://docs.mongodb.org/manual/reference/operator/query/



▼リスト7　エラーログ修正のクエリ

```
var start = ISODate('2014-02-19 00:00:00');
var end = ISODate('2014-02-21 24:00:00');
const ACCESS_TAG_HEADER = "access.";
const ERROR_TAG_HEADER = "error.";

"time:"+start+" - "+end;

function getServerTagName(name, type){
        switch(type){
        case 0:{return ACCESS_TAG_HEADER + name ;}break;
        case 1:{return ERROR_TAG_HEADER + name ;}
        break;
        }
}

function getErrorLevelCtn(level, tag, grep){
var errorCountMethod;

  if(level){
    level = ",level:'"+level+"'";
  }else{level = "";}
  if(tag){
    tag = ",tag:'"+tag+"'";
  }else{tag = "";}
  if(grep){
    var grepQuery = new RegExp(grep);
    grep = ",message:grepQuery";
  }else{grep = "";}
  errorCountMethod = "db.error.find({time:{$gte: start, $lt: end}"+level+tag+grep+"}).count()";
  return eval(errorCountMethod);
}

function showAllServerLog(){
  print("level[error]:"+getErrorLevelCtn("error"));
  print("level[notice]:"+getErrorLevelCtn("notice"));

  print(getErrorLevelCtn("","","PHP Fatal error"));
  print(getErrorLevelCtn("","","\[EMERG\]"));
  print(getErrorLevelCtn("","","\[ALERT\]"));
  print(getErrorLevelCtn("","","\[CRIT\]"));
  print("");
  print(getErrorLevelCtn("","","PHP Warning"));
  print(getErrorLevelCtn("","","PHP Notice"));
  print(getErrorLevelCtn("","","\[ERR\]"));
}

showAllServerLog();
```

くないコードや、新規追加されたコードで、テストでは検知されないようなerrorやwarningを吐き出しているものを見つけ出します。これらを撲滅することもエンジニアには大事な仕事です。Rettyでは毎週Apacheのエラーログを確認して、問題のある個所の修正だけでなく、それを産むコードを共有することでノウハウの共有とサービスの改善を行っています。

これにより、新しくジョインして来てくれたエンジニアと過去に書かれたコードについて議

論する場ができ、エンジニアが作りっぱなしで終わらない体制をとっています。

まとめ

いかがだったでしょうか。小さく始めるログ集計ということで、本稿ではログ活用の導入を書かせていただきました。Fluentdは今ほんとうに熱い技術の1つで、筆者のようなフロントのエンジニアでも簡単に環境が構築でき、MongoDBを使えばクエリもなじみのJavaScriptで書けるので、サーバに眠るログを無駄にすることなく活用できます。

いきなりABテストに使ったり、効果測定にバンバン使おうとするのではなく、「とりあえずなるはやでログを使える環境にしておく」という視点で見ると、Fluentdは最善の選択肢だと思います。その先でサービスに対して有効なログの抽出の仕方が見えてくると思います。最後まで読んでくださりありがとうございました。SD

Column

TIPS

MongoDBに関するtipsを紹介します。

[1]キー名

MySQLと違い、MongoDBではレコードにキー値が含まれていて、レコードごとにデータと共に記録されます。そのため、キー値が長いと容量圧迫の原因となるのでなるべく短い名前を使うべきです。

[2]mongotop、mongostat

MongoDBの監視をしたければこのコマンドを使います。mongotopはtop、mongostatはvmstatのような機能を持ちます。

[3]クエリファイルとしての実行

いちいちMongoDBのshellに入ってコマンドを打ち込むのは面倒なので、処理を外部ファイル化してコマンド一発で呼び出せるようにしましょう。使い方は簡単です。書きたい処理やshellをquery.jsのように保存しておいてディレクトリに置きます。そして次のように実行します。

```
$ mongo < query.js
```

また、その結果をテキストなどに保存したい場合は、

```
$ mongo < query.js > result.json
```

などのようにして出力することが可能です。

[4]cronとしての実行

定期的にクエリを実行してその結果をファイルに保存しておきたい場合などは、cronに登録しておくといいでしょう。

```
#crontab -e
```

でcrontabを開き、次の行を追加します。

```
$ 5 19 * * *  mongo < query.js > ➥
result_`date +\%Y\%m\%d`.log
```

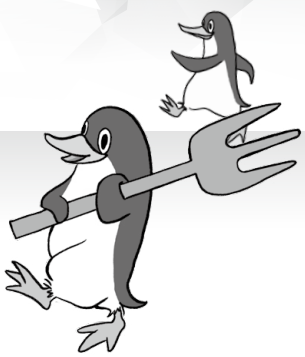
これで「毎日夜19時5分にquery.jsの結果をjsonとして保存する」処理が登録されます。

[5]Capped Collection

MongoDBにはCollectionの容量を固定長にすることで、古いデータを自動で消して新しいデータのみを指定容量で残してくれる機能があります。エラーログなどはあまり過去のデータを保持している必要がないので、固定長にするのが良いかもしれません。fluentd-mongo-pluginはこの機能も提供してくれるので、オプションでこれを指定できます。

forkを通して
考える・試す・コードを読む

Linuxカーネルのしくみを探る



カーネルとはOSの核となる機能のことです。CPU、メモリ、入出力装置などのハードウェアを抽象化することで、アプリケーションが共通の命令で各ハードウェアを扱えるようにします。また、アプリケーションを効率よく動作させるために、プロセスやメモリなどのリソースを管理します。

信頼性が高く、処理の速いLinuxシステムを構築／運用する場合、Linuxカーネルの知識を求められることもあるでしょう。しかし、今のカーネルは機能の数が多く、処理の内容も複雑です。そこで、本特集では数あるカーネルの機能の中から「プロセス管理」に注目して、そのしくみをひも解きます。プロセス管理はシステムのパフォーマンスにも直結する要素ですので、実際の開発や運用に活かせる知識もたくさん見つかるはずです。

Part 1 プロセスに見るLinuxカーネルの役割　中井 悦司　P.094

Part 2 「fork」を通してカーネル内部を理解する　岩尾 はるか　P.104

Part 3 ソースコードで見るカーネルの全体像　中井 悦司　P.115

Part 1 プロセスに見るLinuxカーネルの役割

レッドハット株式会社　中井 悦司(なかい えつじ)
Twitter @enakai00

本パートでは、「プロセスとは何か」「forkとは何か」というところから解説を始めます。カーネルの複雑な役割や動作を学ぶ前に、まずLinuxでは複数プログラムの同時実行をどうやって実現しているのか、その基本的なしくみを理解しましょう。

カーネルを本気で学ぶ第一歩

この特集では、Linuxカーネルについての理解を深めます。Unix/Linuxを本当の意味で理解するには、カーネルを避けて通ることはできません。その一方で、カーネルのしくみについて、「実感」を持って理解して、その知識を役立てられるようになるには、少しばかり高いハードルがあるのも事実です。カーネルの「役割」を学ぶことは、それほど難しいものではありませんが、普通にLinuxを使っていても、一般のユーザにはカーネルの「動作」は見えません。そのため、どうすれば、そのしくみにまで踏み込んで理解できるのか途方にくれてしまいます。

そこで、この特集では、Linuxの「プロセス」を足がかりに話を進めます。この後で説明するように、ユーザから見たLinuxの主役はプロセスです。プロセスの動作のしくみを理解すると、その背後にあるカーネルのしくみが見えてきます。このPart1では、Linuxのプロセスのしくみを徹底的に学びます。その後、Part2で、これらのしくみがカーネル内部でどのように実現されているのか、もう一歩踏み込んだ解説を行います。

とくに、新しいプロセスを生み出すしくみが、プロセスの「fork(フォーク)」です。カーネル内部において、どのようにプロセスが生み出されて実行を開始するのか——forkのしくみを丁寧にひも解きながら、カーネルを理解する「コツ」を実感していただきます。

そして、カーネルのしくみがわかるようになると、実際のソースコード(カーネルソース)にも興味がわいてくるでしょう。最後のPart3では、カーネルソースを入手して、ソースコードを「散策」する方法を紹介します。カーネルソースをいきなりすべて理解するのは不可能ですが、カーネルのしくみの中で、興味がわいた部分のソースを覗きこむ方法を知っておくと便利です。カーネルのしくみがわかってきたら、その後には、「ソースコードを読む楽しみ」——そんなすばらしい世界も待っています。

Linuxの主役はプロセス

Linuxを利用すると、1台のサーバで、さまざまなプログラムを同時に実行することができます。今では当たり前のことですが、筆者が小〜中学生のころ(1980年代)に使っていた初期のPCでは、同時に実行できるプログラムは1つでした。好きなゲームのプログラムを起動すると、そのPCはゲームの実行だけを続けます。ワープロを使いたくなったら、ゲームは終了して、あらためて、ワープロのプログラムを起動します。

一方、Linuxサーバでは、WebサーバとDB

サーバを同時に起動しつつ、SSHでサーバにログインしてviエディタを起動してファイル編集までできてしまいます。Linuxでは、サーバ上で稼働するそれぞれのプログラムを「プロセス」として管理しながら、複数のプロセスを同時に実行することでこれを実現します。

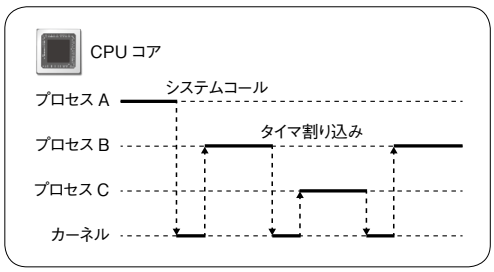
正確に言うと、すべてのプロセスが完全に「同時」に実行されるわけではありません。たとえば、2コアのCPUが搭載されたサーバであれば、それぞれのCPUコアが1つのプログラムコードを実行するので、まったく同時に実行できるプロセス(プログラム)は2つだけです。そこで、Linuxカーネルは、複数のプロセスに対して、CPUの処理時間を順番に割り当てることで、これらを擬似的に同時実行します。Linuxカーネル自身も1つのプログラムコードですので、**図1**のように、1つのCPUコアの上では、各プロセスのプログラムコードとカーネルのプログラムコードが順番に実行されていきます。

このように、さまざまなプロセスの実行を通して、ユーザの役に立つ仕事を行うのがLinuxサーバの役割です。先ほど筆者が小～中学生のころの話をしましたが、当時、こんな川柳(?)を耳にすることがありました。

「コンピュータ　ソフトがなければ　ただの箱」

コンピュータにおけるソフトウェアの重要性を詠んでいるようですが、現代的に言うならば、そのソフトウェアを実行する「プロセス」こそが、Linuxの主役と言えるでしょう。この後は、Red Hat Enterprise Linux(RHEL)6.5の環境を前提に、プロセスの状態を確認するコマンドなどと併せて、Linuxにおけるプロセス管理のしくみを解説していきます。もちろん、Linuxカーネルの役割は、プロセス管理だけではありません。カーネルの役割の全体像については、Part3で説明します。

▼図1　CPUコアが実行するプログラムの切り替え



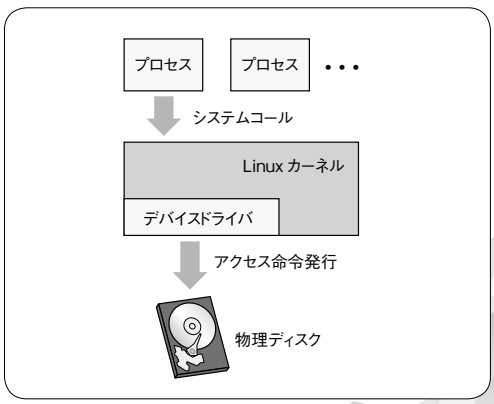
プロセススケジューリング

プロセス管理におけるカーネルの役割について理解を深めるため、**図1**において、CPUコアで実行されるプログラムがプロセスからカーネルに切り替わるタイミングを考えます。これには、大きく2つの場合があります。

・システムコール
・タイマ割り込み

「システムコール」は、プロセスのプログラムコードが、カーネルに何らかの処理を依頼する場合です。たとえば、**図1**のプロセスAはC言語のプログラムで、関数read()でファイルからデータを読み出すとします。このとき、実際にディスク装置にデータの読み出し命令を発行するのはカーネルの役割です(**図2**)。関数read()は、Part2で説明するシステムコールのしくみによって、カーネルに処理を切り替えます。その後、カーネルは、適切なデバイスドライバを用いて、物理ディスクにアクセス命令を発行し

▼図2　システムコールによるカーネルへの処理依頼



ます。

この後、実際にデータが読み出されてメモリ上に配置されるまで、しばらく時間がかかります。この間、プロセスAは何もすることがありませんので、CPUコアの処理時間を無駄にしないよう、カーネルはほかのプロセスに実行を切り替えます。

そのほかのシステムコールの例には、「スリープ処理」があります。たとえば、Webサーバの機能を提供するHTTPデーモンのプロセスは、普段はとくにすることがありません。クライアントのWebブラウザからのアクセスがあってはじめて、HTMLのコンテンツを送り返すという仕事が発生します。そこで、HTTPデーモンは、「クライアントからデータが届くまでスリープする」というシステムコールを発行します。

このシステムコールを受けたカーネルは、該当プロセスをスリープ状態にして、このプロセスには処理時間を割り当てないようにします。そして、クライアントからデータが届いた際に、あらためて、プロセスの実行を再開します。この後でいくつかの例が出てきますが、プロセスは、このほかにもさまざまな処理をカーネルに依頼します。これらはすべて、システムコールを通じて行われます。

もう1つの「タイマ割り込み」は、システムコールで中断されることなく、所定の時間、特定プロセスの実行が続くと発生します。1つのプロセスの実行だけを続けるのは不公平ですので、いったん、強制的にカーネルに処理を切り替えます。この後、カーネルは、次に実行するプロセスを選択して、そちらに処理を切り替えます。

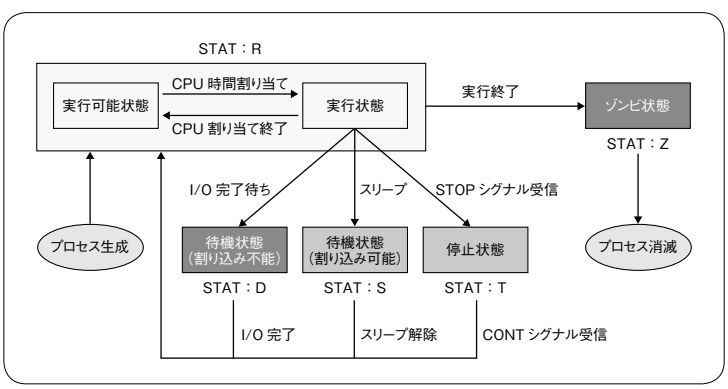
このように、実行するプロセスを順番に切り替える処理を「プロセススケジューリング」と言います。先ほど、複数のプロセスを擬似的に同時実行すると言いましたが、この切り替えをうまく行わないと、プロセスを細切れに実行していることがユーザにもわかってしまいます。Linuxカーネルのプロセススケジューリングには、多数のプロセスを効率的に切り替えながら、それぞれのプロセスをスムーズに実行する工夫がなされています。

プロセスの状態変化

Linux上でプロセスが起動すると、カーネルによってCPUコアの処理時間が割り当てられてプログラムの実行が進む中、その「状態」が変化していきます。プロセスの状態変化は、**図3**のようになります。また、これらの状態は、psコマンドのauxオプションで確認します。**図4**の実行例において、「STAT」列にある最初の文字(S、D、Rなど)が、**図3**の「STAT：R」などで示した文字に対応します。

それでは、これらの状態変化を少し詳しく説明します。まず、新しく生成されたプロセスは、実行状態「STAT：R」になり、プログラムの実行が進みます。厳密には、**図1**で見たように、CPUコアで実際に実行されている瞬間と、CPUコアの実行時間が割り当てられるのを待っている瞬間があります。しかしながら、これは「ミリ秒」単位の変化ですので、psコマンドの表示上は、どちらも同じ「R」の状態となります。

▼図3　プロセスの状態変化



続いて、先に説明したシステムコールで、ディスクアクセスなどのI/O処理をカーネルに依頼すると「STAT：D」の待機状態になります。その後、カーネルに依頼したI/O処理が完了すると、実行状態に戻ります。あるいは、先のHTTPデーモンのように、自発的にスリープ状態「STAT：S」に変化するプロセスもあります。この場合は、事前に指定した状況（Webブラウザからのアクセスが届くなど）になると、実行状態に戻ります。「sleep(60)」などの関数で、一定時間スリープする場合もこれと同じです。この場合は、「60秒経過したら実行状態に戻す」という条件を指定して、スリープ状態に入るシステムコールが実行されます。

最後に、実行中のプロセスに外部からシグナルを送って、動作を一時停止することも可能です。SSH端末でサーバにログインして、あるコマンドを実行したら、予想以上に時間がかかって困ったことはないでしょうか？　次のコマンドプロンプトがなかなか表示されないので、あきらめてCtrl＋Cでコマンドを中断した経験があるかもしれません。

そのような場合は、Ctrl＋Zを押すと実行中のコマンドが一時停止して、コマンドプロンプトに戻ります。その後、「bg」コマンドを実行すると、先のコマンドがバックグラウンドで実行を再開します。端末上は、コマンドプロンプトが表示されて次のコマンド入力ができますが、先のコマンドは背後で実行を継続しています。やや作為的な例ですが、**図5**は、sleepコマンドを一時停止して、バックグラウンドでの実行に切り替える例になります。

このようなプロセスの一時停止／再開処理は、内部的には、「プロセスシグナル」によって行われます。シグナルの詳細はこの後で説明しますが、Ctrl＋Zを押すと、実行中のコマンドのプロセスに「STOPシグナル」が送られて、プロセスは停止状態「STAT：T」になります。次のbgコマンドでは、「CONTシグナル」が送られて、これで実行状態に戻ります。

図3には、このほかに、実行を終了したプロセスに対応する、ゾンビ状態「STAT：Z」という怪しげな状態があります。これについては、この後、プロセスのforkと併せて解説します。

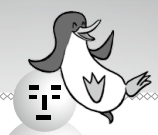
▼図4　プロセスの状態を確認

```
# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0  19400  1468 ?        Ss   2011   0:44 /sbin/init
root          2  0.0  0.0      0     0 ?        S    2011   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    2011   0:00 [migration/0]
root          4  0.0  0.0      0     0 ?        S    2011   0:01 [ksoftirqd/0]
...(略)...
root      31862  0.1  0.0  79580  4728 ?        S    20:47   0:00 /usr/sbin/packa
root      31872 30.8  1.8 470496 144464 ?      D    20:48   0:02 /usr/bin/python
enakai    31874  1.0  0.0 110292  1148 pts/2    R+   20:48   0:00 ps aux
```

プロセスID　　プロセスの状態

▼図5　プロセスを一時停止／再開する例

```
# sleep 60
^Z                                  ← Ctrl＋Zで一時停止
[1]+  停止        sleep 60
# bg                                ← バックグラウンドで実行再開
[1]+ sleep 60 &
#
[1]+  終了        sleep 60          ← バックグラウンドでの実行が完了
```



コラム

止められないプロセスの問題

図3において、スリープによる待機状態「STAT：S」とI/O完了待ちの待機状態「STAT：D」があることを説明しました。これら2つの待機状態は、カーネル内部では、「TASK_INTERRUPTIBLE」「TASK_UNINTERRUPTIBLE」と呼ばれており、それぞれ、「割り込み可能」「割り込み不可能」という意味があります。

これは、簡単に言うと、待機状態のプロセスを停止できるかどうか、という違いになります。スリープ状態のプロセスは、「TERMシグナル」や「KILLシグナル」を外部から送信することで、プロセスを終了することができます。一方、I/O完了待ちのプロセスは、外部からのシグナルを受け付けず、この状態のプロセスを停止することはできません。

これには、I/O処理に伴うデータを保護する役割があります。たとえば、プロセスからの依頼で、カーネルがディスクにデータを書き込んでいる途中でプロセスが停止すると、ディスク上のデータが中途半端な状態になる恐れがあります。そのため、I/O処理が完了してから停止するというわけです。

以前は、このしくみが裏目に出て、困った状況になることもありました。たとえば、デバイスドライバの不具合（バグ）で、I/O処理が完了しているにもかかわらず、カーネルがそれを認識しないことがありました。その結果、このプロセスはいつまでも待機状態「STAT：D」のままになります。しかたなくこのプロセスを停止しようとするのですが、割り込みを受け付けないため、何をしても停止できません。もはやサーバを再起動するしかなくなります。

最近では、デバイスドライバの品質が良くなったので、このような問題は見なくなりましたが、ほかには、NFSクライアントでも問題が発生します。NFSサーバの共有ディレクトリにネットワーク経由でアクセスしている際に、ネットワークの問題が発生してI/O処理が完了しなくなると、同様に待機状態のまま停止できないプロセスが発生します。

このような問題を改善するため、最近のカーネル（バージョン2.6.25以降）には、「TASK_KILLABLE」という新しい状態が用意されています。これは、一般的なシグナルは受け付けないものの、プロセスを強制停止するシグナルだけは受け付ける、という特別な状態です。NFSクライアントでは、「TASK_UNINTERRUPTIBLE」の代わりに、「TASK_KILLABLE」が使用されていますので、NFSで問題が発生した場合は、「KILLシグナル」でプロセスの強制停止が可能です。

シグナルでプロセスを操作

図3において、「STOPシグナル」と「CONTシグナル」を紹介しましたが、プロセスを操作するためのシグナルには、ほかにもいくつかの種類があります。参考として、**表1**に主要なプロセスシグナルをまとめてあります。

それぞれのシグナルには、シグナル名のほかにシグナル番号が割り当てられています。次のようにkillコマンドにシグナル名、もしくは、シグナル番号を指定して、プロセスにシグナルを送信することができます。

```
# kill -TERM 123
# kill -9 123
```

▼表1　Linuxの主なプロセスシグナル

シグナル名	シグナル番号	デフォルトの処理内容
HUP	1	プロセスの終了
INT	2	プロセスの終了
KILL	9	プロセスの終了（変更不可）
TERM	15	プロセスの終了
STOP	19	プロセスの一時停止（変更不可）
CONT	18	プロセスの一時停止からの復帰
CHLD	17	子プロセス終了通知の確認処理

それぞれ、プロセスIDが123のプロセスに対して、TERMシグナルとKILLシグナルを送信する例になります。シグナルを送信する対象となるプロセスのプロセスIDは、**図4**の「PID」の列から確認します。killコマンドは、内部的には、システムコールを利用し

て、カーネルにシグナルの送信を依頼します。

一方、シグナルを受け取ったプロセスの動作については、少し注意が必要です。それぞれのシグナルに対する反応は、プロセスとして実行中のプログラムによって異なります。正確に言うと、プログラムを作成するプログラマは、「シグナルハンドラ」と呼ばれる関数を用意することにより、シグナルを受信した際の動作を指定することができます。**表1**に記載の「デフォルトの処理内容」は、シグナルハンドラが存在しない場合の動作ですが、たとえば、「HUPシグナル」に対するハンドラを用意して、プロセス終了とは異なる動作をさせることもできます。

多くのプログラム(とくにLinuxのサービスとして起動するもの)では、HUPシグナルに対して、設定ファイルの再読み込み処理を割り当てています。あるいは、TERMシグナルについては、処理中のファイルの書き出しなど、プロセスの終了前に必要な処理を実施してから終了するシグナルハンドラを用意します。

一方、KILLシグナルとSTOPシグナルだけは特別で、これらのシグナルハンドラを用意することはできません。つまり、KILLシグナルを送信すると、プログラマが用意した終了処理は実施せず、その場で強制停止が行われます。したがって、シグナルでプロセスを停止する際は、TERMシグナルを使用することが推奨されます。KILLシグナルは、TERMシグナルで停止できないような異常が発生した際に使うと良いでしょう。

プロセスのforkとは?

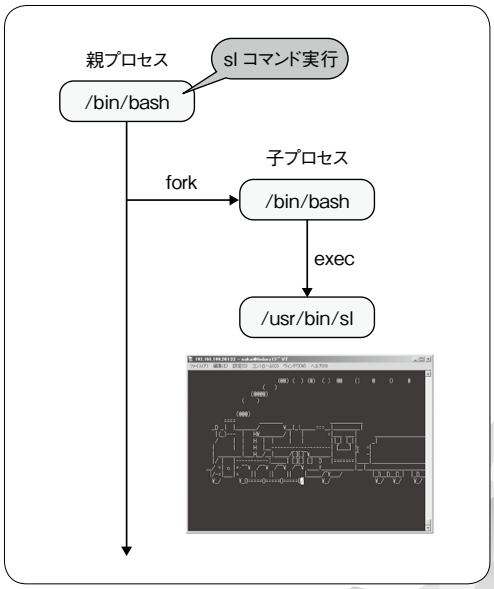
それではいよいよ、本特集のメインテーマとなる「fork」の解説に移ります。Linuxで新しいプロセスを起動する際は、必ず、その「親」となるプロセスが存在します。forkとは、親プロセスを複製して、同じプログラムを実行する「子プロセス」を用意するしくみです。つまり、あるプロセスは、システムコールを利用して、自分自身を複製した子プロセスの作成をカーネルに依頼します。これがforkです。

「これだと同じプログラムのプロセスが増えるだけで、新しいプログラムが実行できないのでは?」——そのとおりです。実は、新しいプログラムを実行する際は、forkだけではなく、もうひとつ、execというシステムコールを利用します。こちらは、同じプロセスのままで、このプロセスが実行するプログラムをそっくり新しいものに入れ替えます。この2つを組み合わせて、forkしてできた子プロセスのほうでexecを実施することで、新しいプログラムを実行する子プロセスが誕生します。

図6は、端末上のコマンドプロンプトに、「sl」コマンドを入力して実行する例です[注1]。コマンドプロンプトを表示して、コマンド入力を受け付ける処理は、シェル「/bin/bash」のプロセスが行います。slコマンドを受け付けたシェルは、forkシステムコールによって、同じシェルのプロセスを子プロセスとして生成します。この後、子プロセスのほうは、execシステムコールで、

注1) slコマンドは、lsと間違えて入力したユーザを驚かせるジョークコマンドです。導入手順は、この後の本文で説明します。

▼図6　プロセスのforkとexec



入力されたslコマンドのプログラム「/usr/bin/sl」にプロセスの中身を入れ替えます。

これだけを見ると、一度forkした後に、あらためてexecするのは二度手間のように感じるかもしれません。しかしながら、forkにはそれ自身の使いどころもあります。たとえば、**図7**は、Webサーバの機能を提供する、HTTPデーモンを起動している環境において、HTTPデーモンのプロセス「/usr/sbin/httpd」を確認した例です。多数のHTTPデーモンが起動していますが、これは、一番上のHTTPデーモンが最初に起動して、その後、forkによって多数の分身を子プロセスとして生成したものです。HTTPデーモンは、多数のWebブラウザからのアクセスに対応するために、これらの子プロセスが並行してアクセスを処理します(**図8**)。それでも処理が追いつかなくなると、さらにforkして、子プロセスを増やすようになっています。

このように、forkとexecは、複数プロセスで並列処理を行う「マルチタスクシステム」としてのLinuxにとって、その根幹となるしくみになります。

forkをプログラムする

forkとexecが理解できたところで、実際にこれらを使用する簡単なプログラムを見てみましょう。**リスト1**は、forkを使って同じプログラム

すべてのhttpdの親プロセス

▼図7　HTTPデーモンがforkで増殖している様子

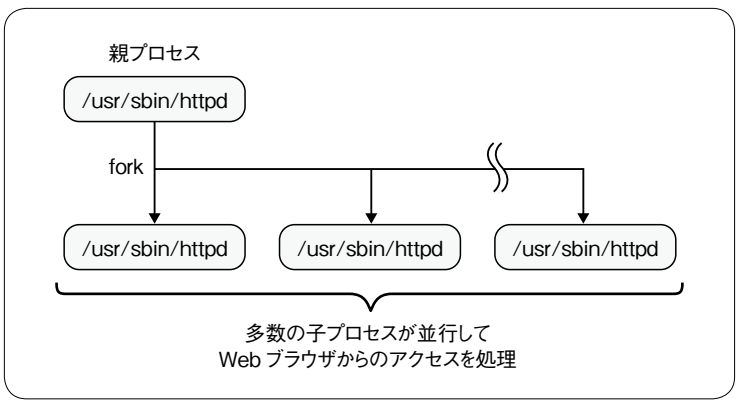
```
# ps -fC httpd
UID        PID  PPID  C STIME TTY          TIME CMD
root      9410     1  0 Apr15 ?        00:00:57 /usr/sbin/httpd
apache   32466  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
apache   32467  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
apache   32468  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
apache   32469  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
apache   32470  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
apache   32471  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
apache   32472  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
apache   32473  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
apache   32474  9410  0 May18 ?        00:00:00 /usr/sbin/httpd
```

子プロセス

プロセスID

親プロセスのプロセスID

▼図8　forkした多数の子プロセスで処理を分散



を実行するプロセスを複製する、C言語のプログラム例です。説明のために行番号を付けてあります。

このプログラムを実行すると、11行目～21行目のforループで、P_MAX(この例では3)個の子プロセスを生成します。12行目の関数fork()は、システムコールを用いて、自分自身を複製した子プロセスを生成するようにカーネルに依頼します。fork()の戻り値は、生成した子プロセスのプロセスIDになります。ここでは、配列pid[]にプロセスIDを保存しています。

一方、forkで生成した子プロセスは、親プロセスと同じプログラムコードの実行を継続します。同じプログラムを初めから実行するのではありません。今の場合は、12行目の関数fork()が終了したところから、子プロセスの実行が始まります。ただし、子プロセスのほうでは、関数fork()の戻り値は、0になります。そのため、子プロセスのほうでは、13行目のif文が成立して、14行目～17行目が実行されます。ここでは、しばらくスリープして終了するだけですが、子プロセスの開始／終了のメッセージを表示するようにしてあります。

他方、親プロセスのほうでは、13行目のif文は成立しませんので、19行目に飛んで、「子プロセスを開始した」というメッセージを表示します。そして、P_MAX個の子プロセスの生成が終わると、23行目～25行目のループで、すべての子プロセスが終了するのを待ちます。24行目の関数wait()は、子プロセスのいずれか1つが終了するまで待機します。子プロセスが1つ終了するごとに関数wait()を抜けて、次のforループがまわります。forループがP_MAX回まわれば、すべての子プロセスが終了したことになります。

このプログラムをコンパイルして実行する手順は、**図9**のとおりです。「Development Tools」のパッケージグループを導入した後、「fork.c」をカレントディレクトリにおいてgccコマンドでコンパイルしてください。

▼リスト1　forkのサンプルプログラム(fork.c)

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <unistd.h>
 4 #include <sys/wait.h>
 5 #define P_MAX 3
 6
 7 int main() {
 8     int pid[P_MAX];
 9     int status, i;
10
11     for (i = 0; i < P_MAX; i++) {
12         pid[i] = fork();
13         if (pid[i] == 0) {
14             printf("子プロセス No.%d: 開始\n", i);
15             sleep(i+1);
16             printf("子プロセス No.%d: 終了\n", i);
17             exit(0);
18         } else {
19             printf("親プロセス: 子プロセス No.%d(pid=%d)を開始\n", i, pid[i]);
20         }
21     }
22     printf("すべての子プロセスの終了を待ちます\n");
23     for (i = 0; i < P_MAX; i++) {
24         wait(&status);
25     }
26     return;
27 }
```

子プロセスは
この間を実行

親プロセスが子プロセスを生成した後に、子プロセスが開始／終了する様子がわかります。この例からわかるように、親プロセスと子プロセスは、実行するプログラムコード自体は同じですが、13行目のif文で処理を分けることで、それぞれ、異なる仕事をすることが可能です。

ここで、関数wait()のしくみについて補足しておきます。**図3**において、実行を終了したプロセスは、「ゾンビ状態(STAT：Z)」になると説明しました。このとき、Linuxカーネルは、終了したプロセスの親プロセスに対して、SIGCHLD(CHLD)シグナルを送信します。親プロセスが関数wait()で子プロセスの終了を待っている場合、このタイミングで関数wait()から抜けて、引数の変数(**リスト1**の例では「status」)に終了した子プロセスの情報が入ります。

親プロセスが関数wait()を実行していない場合、子プロセスはゾンビ状態のままでとどまります。その後、関数wait()が実行されたタイミングで、子プロセスの情報が親プロセスに受け渡されて、子プロセスは完全に消滅します。親プロセスに何らかの問題があって関数wait()を実行できない場合、子プロセスはゾンビ状態で残り続けます。「ps aux」コマンドで「STAT：Z」のプロセスがいつまでも残っている場合は、親プロセスに問題があると考えられます。

なお、子プロセスが終了するより先に親プロセスが終了した場合は、Linux起動時に最初に実行される、「プロセスID=1」のプロセス「/sbin/init」が代わりの親プロセスとなります。この新しい親プロセスが関数wait()の処理を実施するので、子プロセスは無事に終了、消滅することが可能です。

wait()に類似の関数として、特定プロセスIDの子プロセスについて、終了を待ち合わせる関数waitpid()もあります。

▼図9　fork.c(リスト1)を実行する

```
# yum -y groupinstall "Development Tools"
# gcc -o fork fork.c
# ./fork
親プロセス: 子プロセス No.0(pid=8672)を開始
子プロセス No.0: 開始
親プロセス: 子プロセス No.1(pid=8673)を開始
子プロセス No.1: 開始
親プロセス: 子プロセス No.2(pid=8674)を開始
すべての子プロセスの終了を待ちます
子プロセス No.2: 開始
子プロセス No.0: 終了
子プロセス No.1: 終了
子プロセス No.2: 終了
```

▼リスト2　fork＆execのサンプルプログラム(fork-exec.c)

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3 #include <sys/wait.h>
 4
 5 int main() {
 6     int pid, status;
 7
 8     pid = fork();
 9     if (pid == 0) {
10         printf("子プロセス: 5秒後にslが走ります\n", pid);
11         sleep(5);
12         execl("/usr/bin/sl","/usr/bin/sl", NULL);
13         printf("この行は実行されません\n");
14     } else {
15         printf("親プロセス: 子プロセス pid=%d の終了を待ちます\n", pid);
16         wait(&status);
17         printf("親プロセス: 終了\n");
18     }
19     return;
20 }
```

子プロセスはこの間を実行

fork＆execをプログラムする

続いて、forkで生成した子プロセスが、さらにexecで新たなプログラムを実行する例を紹介します。**リスト2**は、forkで生成した子プロセスがexecでslコマンドに変化する例です。**図10**のようにEPELリポジトリからslコマンドのRPMパッケージを導入した後、「fork-exec.c」をカレントディレクトリにおいてコンパイル、実行します。

リスト2の8行目にある関数fork()で子プロセスが生成されるところまでは、先のforkの例と同じですが、今回、子プロセス側では、12行目で関数execl()を実行しています。これは、システムコールによって、この子プロセスの中身を「/usr/bin/sl」に入れ替えます。この子プロセスは、**リスト2**のプログラムを捨てて、slコマンドのプログラムを開始するので、その後の13行目に戻ることはありません。slコマンドが終了した時点で子プロセスは終了します。

ただし、親プロセスとの親子関係は失われません。子プロセスが終了した時点で、親プロセスにSIGCHLDシグナルが送信されて、親プロセスは16行目の関数wait()の待機状態から戻ります。

最後に、プロセスのforkとexecに伴うメモリ割り当てについて補足しておきます。プロセスが実行中のプログラムコードは、当然ながら、サーバのメモリ上に読み込まれています。forkで子プロセスを生成する場合、子プロセスは同じプログラムの実行を続けますので、親プロセスのメモリの内容を子プロセス用にコピーして受け渡す必要があります。しかしながら、「fork-exec.c」の例のように、forkした子プロセスがexecでほかのプログラムに切りかわる場合、せっかくコピーした内容がすぐに不要になります。

そこで、Linuxカーネルは、forkの際にメモリの内容を丸ごとコピーするのではなく、物理メモリ上の同じ内容を親プロセスと子プロセスで共有するというテクニックを使用します。これにより、メモリコピーにかかる時間を削減して、forkの処理を高速化します。このあたりの詳細は、Part2で解説します。

▼図10　fork-exec.c（リスト2）を実行する

```
# yum -y install http://download.
fedoraproject.org/pub/epel/6/i386/epel-
release-6-8.noarch.rpm
# yum -y install sl
# gcc -o fork-exec fork-exec.c
# ./fork-exec
親プロセス: 子プロセス pid=8993 の終了を待ちます
子プロセス: 5秒後にslが走ります
← ここでslコマンドが実行される →
親プロセス: 終了
```

まとめ

Part1では、Linuxカーネルによって、複数のプロセスが実行されるしくみについて、その概要を説明しました。実行中のプロセスには、さまざまな状態があることがわかりましたが、システムの稼動状態を調査する際は、「ps aux」コマンドでプロセスの状態を確認することが大切です。「STAT：S」のプロセスは、何もすることがなくてスリープしていること、「STAT：D」のプロセスが多数ある場合は、I/O処理がボトルネックになっているかもしれないこと、そして、「STAT：Z」のプロセスがずっと残っている場合は、親プロセスに問題が発生していること、などは定番の確認ポイントです。

また、プロセスは、システムコールによって、カーネルにさまざまな処理を依頼します。forkシステムコールは、同じプログラムを実行する子プロセスを生成して、execシステムコールは、実行するプログラムを切り替えます。Part2では、カーネル内部のしくみに踏み込んで、これらシステムコールの舞台裏を徹底解説していきます。SD

Part 2 「fork」を通してカーネル内部を理解する

レッドハット株式会社　岩尾 はるか（いわお はるか）
Twitter @Yuryu

Part1では、Linuxのプロセス管理について、その概要を説明しました。Part2では、新しいプロセスを生み出す「fork」と「exec」が、カーネル内部でどのように実現されているのかを説明します。実行中のプログラムの依頼を受けて、カーネルがさまざまな処理を行う際は、「システムコール」と呼ばれるしくみが利用されます。ここでは、forkとexecを実現するシステムコールの舞台裏を徹底解説していきます。

forkとexecの全体像

Part1では、forkとexecの使用例として、シェル（bash）からslコマンドを実行する例を紹介しました（Part1の**図6**）。ここでは、bashでコマンドを入力したあとに、新しいプロセスが生成・実行される流れをステップごとに見ていきます。

1. bashでコマンドを実行

bashのコマンドラインにslコマンドを入力してEnterキーを押すと、bashはコマンドラインを解析して、実行するコマンドの名前を取得します。bashには、それ自身に内蔵されたコマンド（builtinコマンド）もありますが、「sl」は内蔵コマンドではありませんので、外部のプログラムを呼び出して実行する必要があります。bashは環境変数PATHを順に検索して、slコマンドが「/usr/bin/sl」に配置されていることを見つけます。

2. forkシステムコールの発行

新しいプログラムを実行する際は、forkのシステムコールを実行するfork()と、execのシステムコールを実行するexec()の2段階で処理が行われます。bashはまずfork()を実行して、自分自身を複製することで子プロセスを作成します。fork()を実行した親プロセスは、fork()から返ってきた子プロセスのプロセスIDに対して、その終了を待つ関数waitpid()を発行して、そのまま子プロセスが終了するのを待機します。

一方、fork()で複製された子プロセス側のbashは、exec()の実行に進みます。

3. execシステムコールの発行

子プロセス側ではexec()を呼び出して、「/usr/bin/sl」を実行します。なお、exec()には、引数の指定のしかたによっていくつかのバリエーションがあります（**図1**）。exec()というのはこれらの総称で、実際には、これらの1つを選んで使用します。

それぞれ微妙に引数が異なりますが、execのあとに続く文字には次のような規則があります。

▼図1　exec()のバリエーション

```
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execle(const char *path, const char *arg0, ... /*, (char *)0, char *const envp[]*/);
int execve(const char *path, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execvp(const char *file, char *const argv[]);
```

・l　プログラムへの引数を可変長引数として渡す
・v　プログラムへの引数を配列として渡す
・e　環境変数を配列として渡す
・p　PATH環境変数を検索する

Part1で登場したfork-exec.c（**リスト1**）中で登場したexecl()の場合は、PATHを検索せず、最初の引数「path」にフルパスまたは相対パスで実行するプログラムを指定します。その後の引数として、プログラムに引き渡すコマンドライン引数を指定します。**リスト1**の12行目の例では、「/usr/bin/sl」が2回並んでいますが、最初のものは実行するプログラムの指定で、2番目のものはプロセス名の指定です。プロセス名は、必ずしも実行するプログラムと同じ名前である必要はありません。

execファミリーの関数は、これを呼び出したプロセスを、指定されたプログラムに置き換えて、そのプログラムの先頭から実行を開始します。exec()の実行に成功した場合、exec()を呼び出したプロブラム自身はこのタイミングで解放されて、exec()の呼び出し元に実行が戻ることはありません（**リスト1**の13行目は実行されない）。

このあと、exec()で開始した新しいプログラムが終了すると、プロセスそのものが終了することになります。今回の例では、fork()で複製されたbashの子プロセスからexec()によって「/usr/bin/sl」が実行されて、これが終了すると、この時点で子プロセスは消滅します。

システムコールとヘルパー関数

それでは、システムコールが呼び出されたときの流れを詳しく見ていきます。**図2**は、その概略です（図に記載の用語は、このあとで説明していきます）。Linuxのforkやexecは、実際にはユーザ空間のC言語ライブラリ「libc」と、Linux内のシステムコールの合わせ技で実現されています。fork()やexec()という関数そのものがシステムコールというわけではありません。

たとえばforkの場合、関数fork()そのものはlibc内で定義されており、その中でLinuxのシステムコール本体を呼び出します。後述するように、カーネル内部のシステムコールを実行する関数は、CPUの特殊な命令により呼び出されます。そのため、C言語のプログラムからは通常の関数のように呼び出すことができず、そのギャップを埋めるためのヘルパーとしてlibcが

▼リスト1　fork＆execのサンプルプログラム（fork-exec.c）

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3 #include <sys/wait.h>
 4
 5 int main() {
 6     int pid, status;
 7
 8     pid = fork();
 9     if (pid == 0) {
10         printf("子プロセス: 5秒後にslが走ります\n", pid);
11         sleep(5);
12         execl("/usr/bin/sl","/usr/bin/sl", NULL);
13         printf("この行は実行されません\n");
14     } else {
15         printf("親プロセス: 子プロセス pid=%d の終了を待ちます\n", pid);
16         wait(&status);
17         printf("親プロセス: 終了\n");
18     }
19     return;
20 }
```

※ Part1のリスト2と同じものです

使用されます。また、エラーが発生したときにセットされるerrnoの値なども、このlibcに存在するヘルパー関数でセットされます。Linuxカーネルが、直接にユーザプログラムの変数を変更しているわけではありません。

さらに、今回例に挙がっているforkとexecは、システムコールの中でも特殊なもので、ユーザ空間から見える定義とカーネル内部の定義が1対1に対応していません。forkの場合、実際には、Linuxの「clone」というシステムコールを用いて定義されています。

一方execは、複数あるファミリー関数の中で、execve()のみがシステムコールとして定義されています。それ以外の関数が持つ、パスからの検索や可変長引数の処理はすべてライブラリ関数の機能として実装されており、これらの処理を行ったあと、内部的にexecve()を呼び出す流れになります。

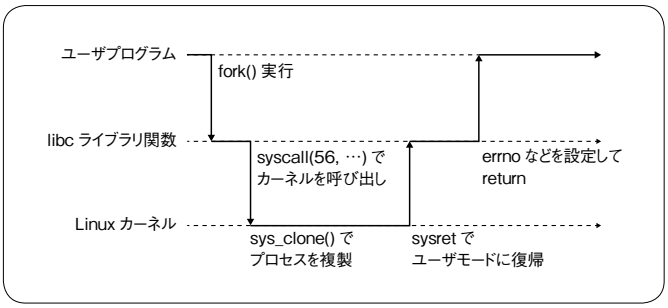
ちなみに、execファミリーの関数についてmanページを検索すると、execve()のみがシステムコールを集めた「Section 2」に掲載されており、その他の関数は、ライブラリ関数を集めた「Section 3」に掲載されています。システムコールを呼び出すプログラムを書く場合、「Section 2」に掲載されているものであれば、Linuxカーネルが処理する範囲と、libcが処理する範囲を意識する必要はありません。合わせ技としてシステムコールが実現されているというのは、あくまで内部的な話になります。一方、「Section 3」に掲載されている、execve()以外のexecファミリーの関数については、Linuxカーネルの機能とは別に、ライブラリとしての追加の処理が行われていることを知っておくと良いでしょう。

ところで、このmanページのセクションを表すために、fork(2)やexecl(3)というように、括弧の中に数字として書く表記がよく用いられます。このことを知っていると、fork(2)のように、括弧の中に数字の2が入っている場合は、それがシステムコールを指すとすぐに読み取れるようになります。

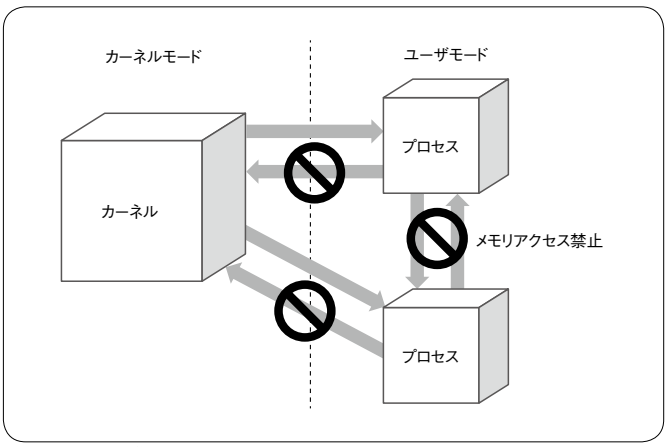
システムコールの呼び出し方

Linuxのシステムコールは、64bitアーキテクチャ(x86_64)の場合、libc内部から「syscall」という特殊なCPU命令によって呼び出されます。これは、通常の関数呼び出しとは何が違うのでしょうか？　この違いを理解するために、まずは、通常のプログラムとLinuxカーネルの違いについて説明します(32bitアーキテクチャでのシステムコールについては、本Part末尾の**「コラム：32bitアーキテクチャでのシステムコー**

▼図2　システムコールのおおまかな流れ



▼図3　カーネルモードとユーザモード



ル」を参考にしてください)。

Linuxカーネルは、すべてのユーザプロセスをまとめて管理する一方で、それぞれのプロセスはお互いに影響を与えず、独立して実行される必要があります。また、ユーザプロセスとして実行される通常のプログラムが、Linuxカーネルの処理を妨げたり、ほかのプロセスのメモリを破壊したりすると、システムが停止したり、セキュリティ上の問題が発生することにもなります。

そこで、Linuxは、CPUの実行モードを「カーネルモード」と「ユーザモード」に分けて、カーネルの実行とユーザプロセスの実行を分離します。また、メモリの内容についても「カーネル空間」と「ユーザ空間」の2種類に分けて管理します。

「カーネルモード」で実行中のカーネルは、「カーネル空間」と「ユーザ空間」のすべてのメモリ領域にアクセスできます。一方、「ユーザモード」で実行される通常のプロセスは、自分自身が使用する「ユーザ空間」のみにアクセスが可能で、「カーネル空間」やほかのプロセスが使用するメモリへのアクセスはできません。**図3**のように、カーネルから各プロセスのメモリへはアクセスできますが、各プロセスは、カーネルやほかのプロセスのメモリにはアクセスできないというわけです。このようなアクセス権限の分離機能を「特権レベル」と言います。この機能は、Linux以外のOSでも一般的に使用されています。

ここで、通常の関数呼び出しは、ユーザモードのプロセスが、自分自身のメモリ空間内部で、その関数を実行するのが前提となります。一方、カーネルの機能を呼び出すシステムコールにおいては、カーネルモードに権限を切り替えて、カーネル空間に用意された関数を実行する必要があります。しかし、ユーザプロセスからカーネル上の任意の関数を呼び出すことができてしまうと、セキュリティ上の問題や、カーネルが停止する致命的なバグを引き起こす恐れがあります。そのため、ユーザモードからカーネルモードへの切り替えは、CPUの機能として厳しく制限されており、この切り替えを行うための特別なCPU命令として、syscall命令が用意されています(**図4**)。

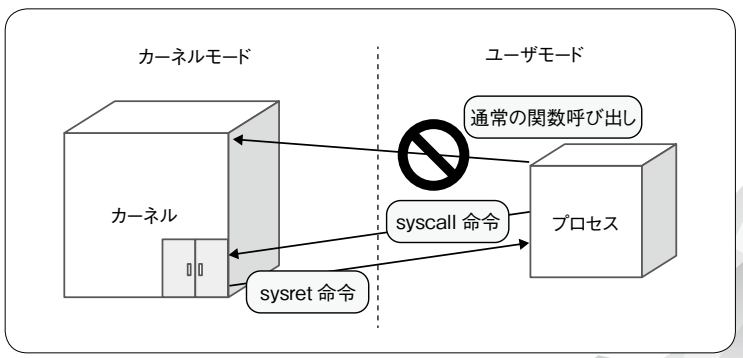
syscall命令は、カーネルが事前に設定したシステムコール専用の関数(システムコールハンドラ)を呼び出す命令で、このときに「カーネルモード」と「カーネル空間」への切り替えが行われます。システムコールを呼び出す際はシステムコールハンドラに引き渡す値として、システムコールごとに割り当てられた番号を引数として指定します。

このシステムコール番号は不変で、Linuxカーネルのバージョンが上がっても、増えることはあっても変更されることはありません。カーネルバージョン3.15の現在で、317個のシステムコールが定義されており、cloneは56番、execveは58番として定義されています。

一般のC言語のプログラムでは、関数syscall()を利用すると、指定のシステムコールを直接実行できます。**リスト2**は、fork()を使用する代わりに、直接にcloneシステムコールを呼び出すプログラムの例で、実行例は次のようになります。

```
$ ./a.out
Parent, child pid = 2455
Child, my pid = 2455
```

▼図4　syscall/sysret命令による特権モードの切り替え



コラム

マルチスレッドとは？

Linuxでは、1つのプロセスは逐次的に実行され、1つのCPUコアしか利用できません。並列処理を行いたいときは、基本的には複数のプロセスを起動することになります。ところが、異なるプロセス間では、メモリの内容が共有されないため、プロセス間でデータを共有するには一時ファイルやソケット、共有メモリなどを使う必要があります。また、プロセスを切り替えるとページテーブルの切り替えが発生し、大きなオーバーヘッドが発生してしまいます。

Linuxでは、この問題を解決するために、1つのプロセスの中で複数の実行単位を持てるように機能拡張が行われました。これを「スレッド」と呼びます(**図5**)。スレッドとプロセスの違いは、メモリ空間やファイルハンドラといったリソースを共有するかしないかです。あるスレッドがグローバル変数に変更を加えると、同じプロセス内のほかのスレッドからも、変更された値を読み取ることができます。また、オープン中のファイルや確保したメモリなどはプロセス単位で管理されており、1つのスレッドが変更を加えるとそのほかのスレッドにも影響がおよびます。プロセスが終了すると、そこに含まれるすべてのスレッドが終了します。

プロセスが分かれている場合、グローバル変数はお互いに読み書きできませんし、1つのプロセスでファイルを閉じても、ほかのプロセスでは開かれたままになります。このような点が、単純に複数プロセスを実行する場合と、複数スレッド(マルチスレッド)を実行する場合の違いになります。

プログラムからスレッドを扱う際は、POSIXスレッド(pthreads)というAPIを使用します。プログラムは、このAPIを通じてスレッドを生成したり、破棄したりできます。pthreadsの内部では、Linuxのcloneシステムコールを使って新しいスレッドを生成しています。

cloneシステムコールを呼び出す際に、フラグにCLONE_THREADを指定すると、プロセスの代わりにスレッドを生成します。この場合は、新しいプロセスIDを割り当てるのではなく、同じプロセスID内に異なる実行単位を作成し、新しいスレッドIDを割り当てます。また、メモリ空間やファイルを共有するために、CLONE_FILESやCLONE_VMといったフラグも指定されます。

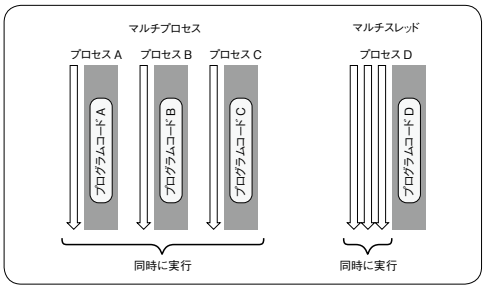
Linuxカーネル内部では、リソースを共有していることと、複数のスレッドが1つのプロセスにまとめて所属していること(プロセスが終了するとスレッドがすべて終了すること)を除けば、通常のプロセスとほぼ同じしくみで扱われています。これによって、Linuxの高度なプロセス管理のしくみが、ほぼそのままスレッドの管理にも役立てられます。マルチコアの環境下でもCPUコアの割り当てが、複数のスレッドに対してきちんと動作するようになっています。

Linuxでスレッドを確認するには、psコマンドに「-L」オプションを付けます(**図6**)。

通常のpsコマンドの出力と異なり、同じPIDの行が複数存在し、LWPとNLWPという列が増えていることがわかります。ここから、複数のスレッドが1つのプロセスに属することがわかります。この例では、プロセスID「608」のプロセスに、3つのスレッドがあります。NLWPはスレッド数を表しており、LWPは、それぞれのスレッドIDを示します。

ちなみに、「LWP」は、Light Weight Processの略で、軽量プロセスという意味です。過去のLinuxではスレッドが実装されておらず、軽量プロセスという別のしくみを使って擬似的にスレッドを実現していたことに由来します。「NLWP」はNumber of LWPで、軽量プロセス数という意味です。このように、列の名称は過去のままですが、現在ではそれぞれがスレッドIDとスレッド数を示しています。

▼図5　マルチスレッドのしくみ



▼図6　スレッドを確認する

```
$ ps axuw -L
USER       PID   LWP %CPU NLWP %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       608   608  0.0    3  0.2 326468  4500 ?        Ssl  21:10   0:00 /usr/sbin/ModemManager
root       608   641  0.0    3  0.2 326468  4500 ?        Ssl  21:10   0:00 /usr/sbin/ModemManager
root       608   702  0.0    3  0.2 326468  4500 ?        Ssl  21:10   0:00 /usr/sbin/ModemManager
(出力から抜粋)
```

syscall()は、ヘルパー関数を経由せずに、直接にシステムコールを呼び出す関数で、プロトタイプ宣言は次のようになっています。

```
int syscall(int number, ...);
```

1つめの引数にシステムコールの番号を指定し、その後ろにシステムコールに対する引数を必要な数だけ渡します。syscall命令と名前が同じで紛らわしいですが、関数syscall()の中で、CPUのsyscall命令を発行しています。

リスト2の8行目の例では、1つめの引数にある「SYS_clone」は、システムコールの番号を表すマクロです。ヘッダファイルの中で、cloneシステムコールに対応する「56」として定義されています。2つめの引数はフラグで、メモリ空間を共有したり、スレッドを生成したりなどのオプションが指定できます。ここでは、子プロセスが終了した際に、親プロセスにSIGCHLDシグナルを送ることだけ指定しています。3つめ以降の引数はスレッドを利用する際に指定するものですが、ここでは解説は割愛します。マルチスレッドについては、「**コラム：マルチスレッドとは？**」を参照してください。

厳密には、関数syscall()自体もヘルパー関数の一種ではありますが、syscall命令を発行するための必要最低限の処理に限っています。そのため、関数syscall()を利用することで、Linuxのシステムコールをよりダイレクトに呼び出せていることがわかります。

▼リスト2　clone.c

```
 1 #include <unistd.h>
 2 #include <sys/syscall.h>
 3 #include <sys/signal.h>
 4 #include <stdio.h>
 5
 6 int main() {
 7     // equivalent to fork()
 8     int pid = syscall(SYS_clone, SIGCHLD, 0, 0, 0, 0);
 9
10     if(pid == 0) {
11         int cpid = getpid();
12         printf("Child, my pid = %d\n", cpid);
13     } else if (pid == -1) {
14         perror("SYS_clone: ");
15     } else {
16         printf("Parent, child pid = %d\n", pid);
17     }
18     return 0;
19 }
```

カーネル内でのfork処理

ユーザプログラムにおいて、fork()、あるいは、syscall()を実行すると、最終的にsyscall命令によって、カーネル内のシステムコールハンドラを呼び出すことがわかりました(**図4**)。このあと、カーネル内でどのような処理が行われるのかを見ていきます。

システムコールハンドラは、引数に渡されたシステムコール番号を基に、実際にシステムコールを処理する、カーネル内の関数を決定します。カーネル内ではそれぞれのシステムコールに対応する関数はsys_から始まる名前で定義されており、たとえば、clone()の場合はsys_clone()、execve()の場合はsys_execve()となります。システムコール番号と関数の対応付けはテーブルとして管理されており、今回はシステムコール番号としてSYS_clone(56番)が指定されていますので、sys_clone()が呼び出されます(**図7**)。

sys_clone()は、自分自身のプロセス定義をまるごとコピーし、新たなプロセスを生成します。まず、新しいプロセスIDを割り当てて、メモリ空間やオープン中のファイルなどの情報をすべてコピーします。このとき、新しく生成されたプロセスにおいて、次に実行するべきプログラムコードのアドレスとして、カーネル内のret_from_fork()という特別な関数を設定します。

新しくプロセスを作成した時点では、CPUは、まだそのプロセスの実行を開始しません。sys_clone()の処理の最後で、この新しいプロセスをプロセススケジューラに登録して実行可能にします。これで、子プロセスは親プロセス

の完全な複製となり、次にCPUが割り当てられた際には、前述の関数ret_from_fork()から実行が再開されます。

sys_clone()が終了すると、カーネルモードからユーザモードへの切り替えが行われ、呼び出し元のユーザプロセスに実行が戻ります。これを実現する専用の命令がsysret命令で、syscall命令とちょうど反対の役割を持ちます(**図4**)。ユーザプロセスでは、ヘルパー関数の続きが実行されて、変数errnoの変更などの必要な後始末を行い、戻り値を返して終了です。

最後に、生成された子プロセスの側の処理を見てみましょう。前述のとおり、子プロセスではカーネル内の関数ret_from_fork()から実行が再開されます。この関数は戻り値を0にセットして、sysret命令でカーネルモードからユーザモードに戻るだけの処理を行います。子プロセスでは、親子が別れる直前に呼び出したcloneシステムコールが、プロセス側に0を返してくるように見えます。Part1の**リスト2**のところで説明したように、子プロセスの側で関数fork()の戻り値が0になるのは、このためです。これでforkの処理は完了です。

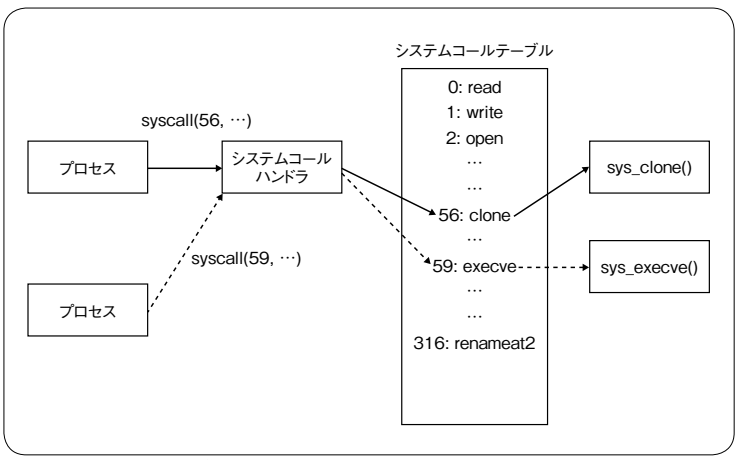
カーネル内でのexec処理

次に、execveシステムコールがどのように動作しているのかを見てみましょう。execveに対応して、カーネル内で実際の処理を行う関数は、sys_execve()として定義されています。その中での処理を順番に説明します。

①指定されたファイルが実際に存在するかを確認します。存在しない場合はエラーになります。
②指定されたファイルを開き、先頭部分をメモリにロードします。
③ファイルの先頭部分を見てファイルの種類を調べます。Linuxで実行可能なファイル形式には、最も一般的なELF(Executable and Linkable Format)のほかに、レガシーなa.outやスクリプトファイルなどがあります。「スクリプトファイルが実行可能形式？」と思うかもしれませんが、shebang(シェバン)と呼ばれる「#!」の2文字で始まるテキストファイルは、実行時には、ほかのバイナリファイルと同じく実行可能ファイルとして扱われます。
④それぞれのファイル形式に応じた、専用の読み込みルーチンを呼び出して、残りのファイルを読み込みます。スクリプトファイルの場合は、shebangの行で指定されたプログラムを実行します(指定のプログラムについて、再度、①からの処理を行います)。
⑤プログラムの先頭から実行を開始します。

sys_execve()は、メモリ上に展開したプログラムを先頭から実行しなおすだけで、それ以外の特別な処理は行いません。たとえば、execve()を呼び出したプロセスがオープン中のファイルハンドラは、そのまま実行されたプログラムの中からも使えます。リダイレクトやパイプといった処理は、親プロセス側でファイルハンドラを用意して、fork()したあとに適切につなぎ変え

▼図7　システムコールテーブル



てからexecve()することによって実現されています。

先に説明したように、sys_execve()では、スクリプトファイルもバイナリファイルと同じように扱い、必要に応じてshebangで指定されたプログラムを実行します。そのため、exec()ファミリーの関数では、スクリプトファイルについても、バイナリの実行ファイルと同様に扱うことができます。

物理メモリと仮想メモリの違い

先に説明したように、forkは、親プロセスをコピーすることで子プロセスを生成します。しかしながら、その直後に子プロセスがexecを実行する場合、すぐにコピーを破棄して、別プログラムのメモリ空間を展開することになります。これでは、最初のコピーの処理が無駄になり、効率がよくありません。そのためLinuxカーネルには、「Copy on Write(CoW)」というメモリの不必要なコピーを省略するしくみが導入されています。CoWを理解するために、まず、物理メモリと仮想メモリの違いを説明します。

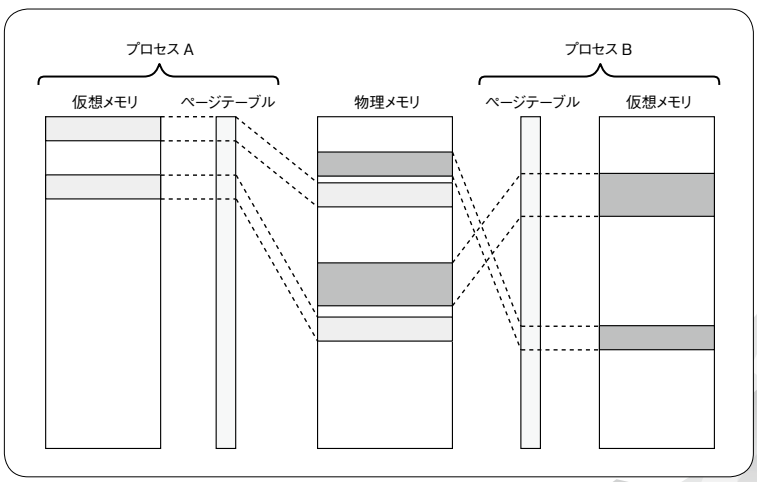
Linuxを含むモダンなOSでは、メモリを「物理メモリ」と「仮想メモリ」の2段階に分けて管理しています(**図8**)。物理メモリは、物理的にサーバに搭載されたメモリをそのまま表します。16GBのメモリが搭載されたサーバでは、16GB分の単一のメモリ空間になります。ただし、これをそのまま複数のプロセスから利用するのは、少し無理があります。どの部分のメモリを使用するかによって、プロセスからみたアドレスが変わってしまったり、複数のプロセスでメモリを取り合った結果、断片化されたアドレスを使用せざるを得なくなったりします。

このような問題を解決するために、仮想メモリのしくみが導入されました。仮想メモリは、すべてのプロセスに独立したメモリ空間を割り当てる機能で、物理メモリのサイズにかかわらず、32bitアーキテクチャであれば4GiB、64bitアーキテクチャであれば16EiB(2^{64}B)のアドレス空間が利用できます[注1]。ただし64bitの場合、現在のところ16EiBという広大な空間は必要としないため、Linuxでは、効率化のために256TiB(2^{48}B)に制限しています。

物理メモリと仮想メモリの対応付けは「ページ」という単位で行われます。x86_64アーキテクチャの場合、ページのサイズは4KiBです。このページを管理するのが「ページテーブル」で、どの仮想メモリがどの物理メモリに対応しているのかを記録します。仮想メモリはプロセスごとに独立していて、それぞれに専用のページテーブルが用意されます。これにより、それぞれのプロセスは、ほかに影響されない独立したメモリ空間を利用できます。

注1) KiB、GiBなどは、1KiB=1024B、1GiB=1024KiBのように、2進数に基づいた正確な単位を表す記号。

▼図8　物理メモリと仮想メモリの対応



forkに伴うメモリのコピー

それでは、forkに伴うメモリ管理の流れを説明していきます。forkが実行されると、始めに、プロセスが利用しているページテーブルの内容がコピーされます(**図9**)。つまり、親プロセスと子プロセスの仮想メモリは、同一の物理メモリにマッピングされて、両方のプロセスが同一のメモリ内容を共有する形になります。このタイミングで、共有する物理メモリのすべての領域を書き込み禁止にセットします。また、物理メモリの参照カウンタを増加させて、物理メモリを共有していることを記録しておきます。

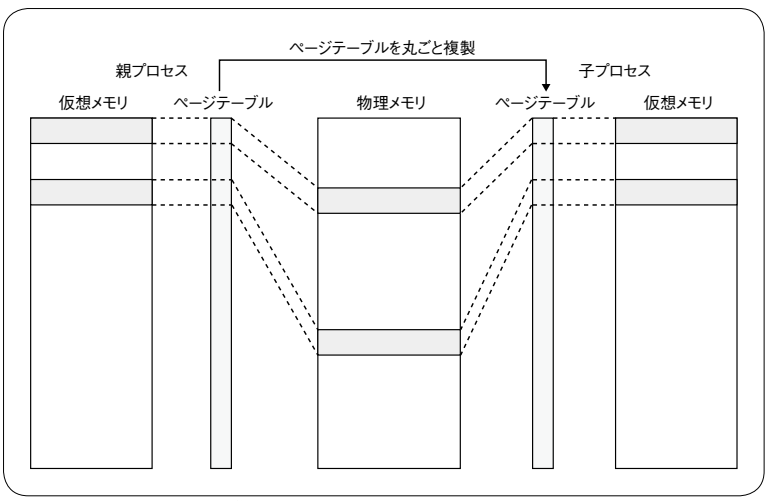
その後、子プロセスがexecve()を実行した場合は、子プロセスのページテーブルはすべて破棄して、新しいプログラムコードをメモリに読み込んで、対応するページテーブルを作成します。このとき、共有していた物理メモリの参照カウンタを減少します。参照カウンタが1になって共有がなくなった場合は、次の書き込みが発生した際に書き込み禁止が解除されます。このように、CoWのしくみでは、ページテーブルを複製することで、実際のメモリ内容をコピーせずに高速にメモリを複製します。これには、物理メモリの消費を抑える効果もあります。

それでは、forkした子プロセスがexecを実行せずに、共有状態のメモリに書き込みを行った場合はどうなるのでしょうか？　この場合、メモリへの書き込みが発生した瞬間に、ページフォルトと呼ばれる動作がCPUによって起動されます。これにより、プロセスの実行が一時的に中断され、カーネルへ動作が切り替わります。カーネルはページフォルトの発生を検知すると、メモリの状態を調べ、CoWによって物理メモリが共有されていることを知ります。その後、書き込み対象の物理メモリ(メモリページ)のコピーを作成して、ページテーブルを更新することで、対象となるメモリページの共有状態を解除します。また、コピーされた新しいメモリ領域は書き込み可能に設定されます。コピー元の領域の参照カウンタは減少されます。

これらの処理が完了すると、カーネルは一時停止していたプロセスに処理を戻して、プロセスによるメモリへの書き込み動作を再開します。Copy on Write(書き込み時複製)という名前は、実際に書き込みが発生した場合に必要最小限のコピーを行うという、このしくみに由来しています。

これで、forkとexecに関わるカーネルの動作説明は終わりです。forkとexecの2種類のシステムコールによって、プロセスの複製と新規プログラムの実行が実現されることがわかりました。

▼図9　CoWによるメモリの複製



システムコールにおけるvDSOの有用性

最後に、システムコールの呼び出しを高速化するしくみである「vDSO」について説明してお

きます。

forkやexecといった、システムの状態を変更するシステムコールは、syscall命令で呼び出されます。このしくみは、ユーザモードとカーネルモード、および、メモリ空間の切り替えが発生するために、CPUでの実行コストが高く、処理に時間がかかります。

一方、gettimeofdayという、日付と時刻を読み取るシステムコールがありますが、これはシステムの状態をまったく変更せず、状態を読み取るだけの処理になります。このようなシステムコールを高速化するために、Linuxカーネルには、vDSO（Virtual Dynamic Shared Object）というしくみが導入されています。

vDSOは、ユーザプロセスのメモリ空間の一部に、カーネルのメモリを読み取り専用でマップし、カーネルモードに切り替えることなくアクセスできるようにするしくみです（**図10**）。vDSOの領域は、Linuxカーネルがexecve()で新たなプログラムの実行を開始する際に、自動的にマップされるもので、仮想メモリ上のランダムなアドレスに配置されます。この部分に、gettimeofdayを実現するのに必要なコードとデータが読み込み専用でマップされており、これを利用することで、システムコールの処理がユーザモードだけで完結して、実行速度が向上します。vDSOの領域は、通常の共有ライブラリと同じELF形式になっているので、ユーザライブラリは、該当のシステムコール名からアドレスを検索して、通常の関数と同じ方法で呼び出すことができます。

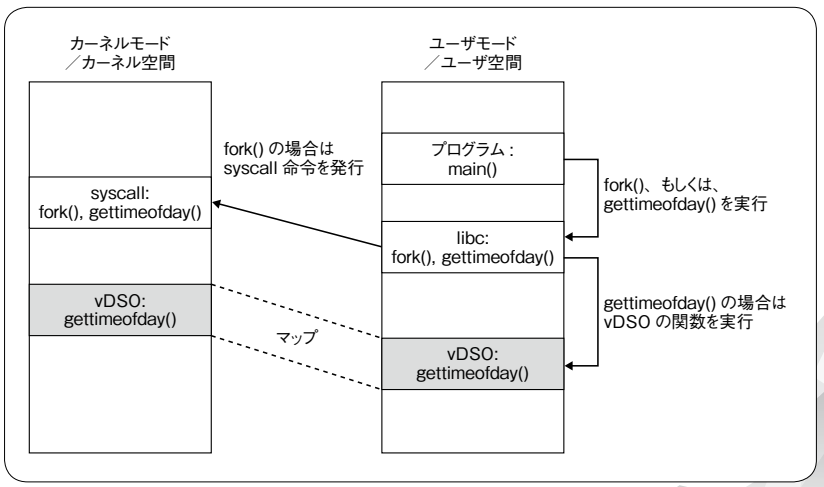
gettimeofdayを例にとって、具体的に説明します。まず、プログラムを起動した際に、プログラムにリンクされているlibcの初期化ルーチンが実行され、その中でvDSOがマップされているアドレスを取得します。さらに、vDSOで提供されるシステムコールの名前から、それぞれのルーチンの存在するアドレスを検索します。gettimeofdayの場合は__vdso_gettimeofday()という関数がvDSO領域内に存在するので、この関数へのポインタをlibc内の変数に保存します。その後、gettimeofdayシステムコールがユーザプログラムから呼ばれると、libcは、本物のシステムコールを実行する代わりにこの関数を呼び出します。

vDSOを利用するシステムコールは、アーキテクチャによって異なります。現在のところ、x86_64アーキテクチャにおいて、vDSO経由で呼び出されるシステムコールは、clock_gettime、getcpu、gettimeofday、timeの4つです。

まとめ

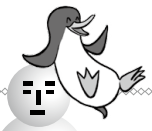
Part2では、forkとexecを実現するシステムコールについて徹底解説を行いました。システムコールは、ユーザプログラムからカーネル内部の関数を呼び出すという、おもしろい機能を

▼図10　通常のシステムコールとvDSOによるシステムコール



実現するしくみです。Linuxカーネルとユーザプロセスが連携する様子が垣間見えたのではないでしょうか。CPUのアーキテクチャによってシステムコールの実現方法が異なるなど、ハードウェアレベルのしくみにも興味がわくところです。

最後のPart3では、プロセス管理を含めた、Linuxカーネル全体の役割、そして、Gitを利用したソースコードの読み方を紹介します。SD



コラム

32bitアーキテクチャでのシステムコール

本文のシステムコールの解説では、64bitアーキテクチャのx86_64をベースにしました。それ以前の32bitアーキテクチャである「i386」では、システムコールの呼び出し方が少し異なります。

i386では、システムコールの呼び出しは、ソフトウェア割り込みの機能を使って行われます。i386のCPUには、ソフトウェアから呼び出せる割り込みが256個用意されており、Linuxでは、0x80番をシステムコールに利用することになっています。libcのヘルパー関数は、syscall命令の代わりに、ソフトウェア割り込みを発生させるint命令を利用します。

int命令による呼び出し

具体的には、0x80番目の割り込みを発生させる「int 80h」命令を実行します。すると、0x80番目の割り込みハンドラとして登録されている、カーネルのシステムコール実行関数が動き出します。その後のカーネル内の処理はsyscall命令で呼び出された場合と同じですが、システムコールの関数からユーザモードに戻るときが異なります。syscall命令で呼び出された場合は、対応するsysret命令でユーザモードに戻りますが、int命令で呼び出された場合は、割り込みから復帰するiret命令を使って戻ります。

しかしながら、このint命令には欠点がありました。もともとは、文字どおり「いつ発生するかわからない」割り込み処理を行うための命令だったため、システムコールの実行に使うにはオーバーヘッドが大きく、CPU内部での無駄な処理が含まれていたのです。そこで、システムコールを呼び出すための専用の命令として、Intel社はsysenter/sysexit命令、AMD社はsyscall/sysret命令を追加しました。この2種類の命令セットは、システムコールを呼び出すという目的に最適化されており、int命令よりも高速に実行できます。

CPUごとに自動的に最適化されるvDSO経由のシステムコール

ただし、これらの命令はIntel Pentium II、および、AMD K6以降のCPUで実装されたため、古いCPUでは使用できません。また、IntelのCPUとAMDのCPUで命令が異なるので、CPUの種類を判別して、実行する命令を変更する必要があります。そこで、i386版のLinuxカーネルでは、カーネル起動時にCPUの種類を判別して、vDSO内にCPUに適したシステムコールの呼び出し関数を用意することにしました。つまり、i386のlibcは、すべてのシステムコールについて、vDSOを利用して呼び出します。libcがvDSO内の関数を呼び出すと、そこに保存されている関数がCPUごとに最適な命令を利用してシステムコールを呼び出してくれます。カーネル起動時に関数が用意されるため、オーバーヘッドも最小限に抑えられています。

x86_64のvDSOは、ユーザモードで処理が完結するシステムコールだけを提供しますが、i386では、すべてのシステムコールに対するヘルパーとして、vDSOが利用されていることになります。また、互換性のために、vDSOを使わずに、int 80h命令を利用してシステムコールを呼び出すこともできます。

本文で解説したx86_64の場合は、x86_64に対応する、すべてのCPUでsyscall命令が用意されているので、通常のシステムコールの呼び出しにはvDSOは使用せず、syscall命令を直接に実行します。Intel社製のCPUでもsyscall命令が利用可能になっているため、CPUごとに命令を切り替える必要はありません。そのためi386に比べると、シンプルで統一されたコードになっています。

Part 3 ソースコードで見るカーネルの全体像

レッドハット株式会社　中井 悦司（なかい えつじ）
Twitter @enakai00

Part1とPart2で紹介したプロセス管理は、カーネルの役割の一部です。本パートでは、カーネルのソースツリーを見ながら、ほかにどんな役割を担っているのかを見てみます。また、カーネルのソースコードを読むときの手順や手がかりも示します。

Linuxカーネルの役割とは?

Part1～Part2では、Linuxにおけるプロセス管理、とくにforkとexecのしくみを通して、Linuxカーネルの動作を学びました。カーネルがどのようなしくみでプロセスの動作を支えているのか、その舞台裏を実感できたと思います。ただし、Linuxカーネルの役割は、プロセス管理だけではありません。プロセス管理を含め、主要なカーネルの機能には、次のようのものがあります。

・プロセス管理
・メモリ管理
・ファイルシステム管理
・物理ディスク管理
・ネットワーク管理

「プロセス管理」は、これまでに説明したプロセススケジューリングやfork/execなど、プロセスを生成／実行する機能です。「メモリ管理」は、プロセスに対する物理メモリの割り当てに加えて、ディスクキャッシュに使用するメモリの管理などが含まれます。「ファイルシステム管理」は、ext4などのファイルシステムの機能を提供して、「物理ディスク管理」は、デバイスドライバを介して物理ディスクにアクセスする機能を提供します。最後の「ネットワーク管理」は、その名のとおり、ネットワーク経由でプロセス同士が通信するための機能です。

これらの機能はすべて、物理ハードウェアを「抽象化」するものと考えることができます。わかりやすい例で言うと、サーバに搭載するハードディスクには、さまざまな規格や種類があり、ハードディスク自体が受け付ける命令はそれぞれに異なります。しかしながら、**Part1の図2**で見たように、ハードディスクそのものへの命令は、ハードディスクに合わせたデバイスドライバが発行します。そのため、Linuxを利用するユーザは、ハードディスクの種類を気にすることなく、「ls」「cat」などのコマンドでハードディスク上のファイルにアクセスが可能です。

物理メモリの割り当ても同様で、サーバ上の物理メモリをプロセスに配分する処理は、Linuxカーネルが行ってくれるため、それぞれのプロセスは、物理的にどの部分のメモリを使用しているかなど気にする必要はありません。アプリケーションプログラムを書くプログラマは、malloc()などの標準的なシステムコールで、メモリ領域を取得することが可能です。ほかのプロセスが使用しているメモリ領域を誤って使用することもありません。

つまり、Linuxカーネルの働きによって、Linuxを使用するユーザやアプリケーションプログラムは、単純化された「架空のコンピュータ」を与えられるのです（**図1**）。Linuxが稼働す

るハードウェアには、サーバ、PC、スマートフォン、ゲーム機器などさまざまな種類がありますが、すべて、同じ「架空のコンピュータ」として、共通のコマンドやシステムコールで操作することができるというわけです。

カーネルソースの入手方法

Linuxカーネルは、これらさまざまな機能を1つのバイナリコード(カーネルコード)で実現しています。/bootディレクトリ以下にある「vmlinuz」で始まるファイルがカーネル本体のバイナリコードです[注1]。ただし、このバイナリコードの大元となるソースコードは、「サブシステム」と呼ばれる機能単位に大きく分けられており、Linuxカーネルの開発コミュニティには、それぞれのサブシステムに対する責任者(メンテナ)が存在します。各サブシステムのメンテナは、世界中の開発者から送られてくるパッチ(修正コード)のレビューを行い、必要と判断したものについて、担当するサブシステムへの適用を行っていきます。

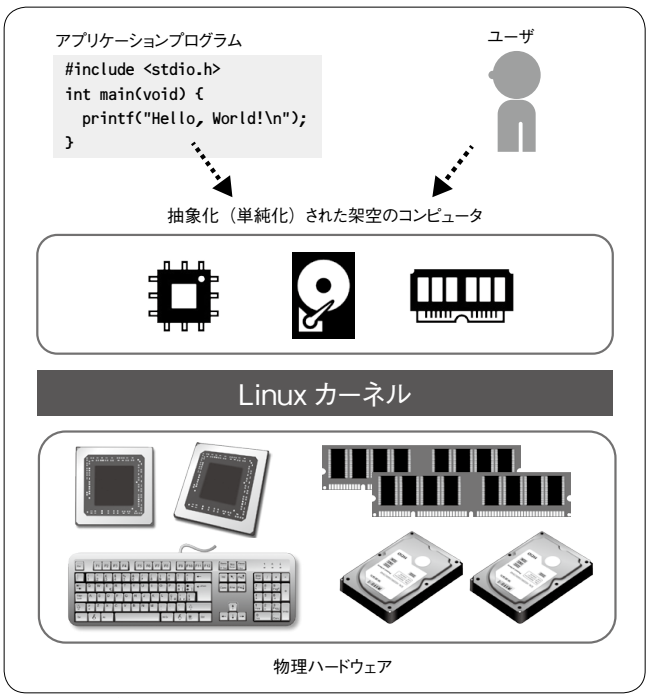
それでは、実際にLinuxカーネルのソースコード(カーネルソース)を入手して、その中身を覗いてみることにしましょう。ただし、カーネルソースにはいくつかの種類があります。まず、開発コミュニティで日々開発が続けられる「アップストリーム」と呼ばれるソースコードがすべての大元になります。アップストリームのカーネルは、2〜3ヵ月ごとにバージョンアップを続けています。

一方、Red Hat Enterprise Linux(RHEL)などのLinuxディストリビューションでは、アップストリームの特定バージョンのソースコードをベースに、独自の修正を施したものを使用しています。RHELの場合は、RHELのメジャーバージョンごとにカーネルのバージョンを固定することが目的です。「独自の修正」と言っても、RHELだけに固有の機能を追加するというわけではなく、アップストリームの新バージョンで追加されたバグ修正や機能拡張のパッチの中から、必要なものを選択的に適用しています。

RHELのカーネルソースは、SRPMというソースコード用のパッケージ形式で配布されており、これを元にして、自分でカーネルのRPMパッケージを作成することができます。『プロのためのLinuxシステム・10年効く技術』[1]では、RHELのカーネルソースを修正して、独自のカーネルを作成する方法が紹介されています。

一方、アップストリームのカーネルソースは、分散バージョン管理システムのGitで管理され

▼図1 Linuxカーネルによるハードウェアの抽象化



注1) 正確にはカーネル本体の起動後に、追加でメモリに読み込む「カーネルモジュール」もありますが、これも内部的にはカーネルの一部として動作します。

ており、gitコマンドを使用すると、インターネット上のリポジトリから、手元のLinuxマシンにダウンロードすることができます。RHEL6.5の環境であれば、**図2**の手順になります。

図2では、gitコマンドのパッケージを導入した後、Linusが管理する最新のカーネルソースをインターネットからダウンロードしています。ダウンロード時に作成されたディレクトリ「linux」の下は、**表1**のように、機能／役割ごとにディレクトリが分けられています。この表によると、プロセス管理に関わるソースコードは、カーネルの基本機能として「kernel」ディレクトリに含まれています。本特集のメインテーマであるforkについては、ずばり、「kernel/fork.c」というファイルがあります。

「kernel/fork.c」を開いてみると、冒頭に**リスト1**のようなコメントがあり、1991年にLinus自らがこのソースコードを作成したことがわかります。1991年といえば、まさにLinusがLinuxの開発をスタートした年であり、forkがLinuxの根幹となる機能であることがよくわかります。ちなみに、その後ろのコメントを翻訳すると次のようになります。

▼表1　カーネルのソースツリーに含まれるディレクトリ

ディレクトリ	説明
Documentation	カーネルソースのドキュメント類
arch	アーキテクチャに固有のソースコード
block	ブロックI/Oレイヤー
configs	カーネル・コンフィギュレーションのサンプル
crypto	暗号化API
drivers	デバイス・ドライバ
firmware	ドライバのコンパイルに必要なファームウェア
fs	VFSレイヤー、およびファイルシステム
include	カーネル・ヘッダファイル
init	カーネルの起動と初期化処理
ipc	IPC（Interprocess Communication Code：プロセス間通信）
kernel	カーネルの基本機能（プロセス管理、時間管理など）
lib	ライブラリ・モジュール
mm	メモリ管理サブシステム
net	ネットワーク・サブシステム
samples	サンプル・コード
scripts	カーネル・ビルドに使用するスクリプト
security	Linuxセキュリティ・モジュール
sound	サウンド・サブシステム
tools	開発用のツール類
usr	初期RAMディスク関連
virt	仮想化関連（Linux KVMなど）

▼図2　gitコマンドでカーネルソースをダウンロード

```
# yum -y install git
# git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
# cd linux
```

▼リスト1　fork.cの冒頭部分

```
/*
 *  linux/kernel/fork.c
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 */

/*
 *  'fork.c' contains the help-routines for the 'fork' system call
 * (see also entry.S and others).
 * Fork is rather simple, once you get the hang of it, but the memory
 * management can be a bitch. See 'mm/memory.c': 'copy_page_range()'
 */
```

「'fork.c' はforkシステムコールのヘルパー関数を含んでいる(entry.Sなども参照)。forkは、コツさえつかめば簡単だが、メモリ管理については難解かもしれない。'mm/memory.c' のcopy_page_range()を参照するように」

forkを開発中のLinusの気持ちがそのまま記載されており、歴史の1ページに触れたような気分になります。

Gitを駆使してソースコードを探索

先ほど、Gitのことを「分散バージョン管理システム」と紹介しました。実は、先ほどダウンロードしたディレクトリには、過去のソースコードの変更履歴が含まれており、gitコマンドを駆使すると、ソースコードの各行について、いつ、誰が、何のために、そのコードを書いたのかを調べることが可能です。

たとえば、先ほどの「kernel/fork.c」について、次のコマンドを実行すると、**図3**のような表示がなされます。

```
# git blame kernel/fork.c
```

これは、このソースコードの各行について、その行を書いた「コミッタ」の名前と、ソースコードに正式に取り込まれた「コミット日時」を表示しています。ここでは、例として、「がちゃぴん先生」として有名な、「KOSAKI Motohiro」氏がコミットした行を記載しています。

さらに、行頭の「コミットID」は、この行を追加したパッチを特定するためのID番号です。次のコマンドで、コミットIDに対応するパッチの内容が確認できます。

```
# git show c6a7f572
```

図4は上記コマンド出力の冒頭部分です。パッチの説明文に続いて、パッチの承認記録が記載されています。「Reviewed-by」は、このパッチをレビューして問題ないと確認した人物による署名で、「Signed-off-by」は、パッチの作成者、および、メンテナがこのパッチを正式採用することに同意したことを示す署名です。「Linusの右腕」と呼ばれるAndrew Mortonの後に、Linus

▼図3 「git blame kernel/fork.c」の出力(抜粋)

```
c6a7f572 (KOSAKI Motohiro      2009-09-21 17:01:32 -0700  204) static void account_kernel_stack(struct thread_info ↵
*ti, int account)
c6a7f572 (KOSAKI Motohiro      2009-09-21 17:01:32 -0700  205) {
c6a7f572 (KOSAKI Motohiro      2009-09-21 17:01:32 -0700  206)     struct zone *zone = page_zone(virt_to_page(ti));
c6a7f572 (KOSAKI Motohiro      2009-09-21 17:01:32 -0700  207)
c6a7f572 (KOSAKI Motohiro      2009-09-21 17:01:32 -0700  208)     mod_zone_page_state(zone, NR_KERNEL_STACK, account);
c6a7f572 (KOSAKI Motohiro      2009-09-21 17:01:32 -0700  209) }
c6a7f572 (KOSAKI Motohiro      2009-09-21 17:01:32 -0700  210)
^1da177e (Linus Torvalds       2005-04-16 15:20:36 -0700  211) void free_task(struct task_struct *tsk)
^1da177e (Linus Torvalds       2005-04-16 15:20:36 -0700  212) {
c6a7f572 (KOSAKI Motohiro      2009-09-21 17:01:32 -0700  213)     account_kernel_stack(tsk->stack, -1);
f19b9f74 (Akinobu Mita         2012-07-30 14:42:33 -0700  214)     arch_release_thread_info(tsk->stack);
f7e4217b (Roman Zippel         2007-05-09 02:35:17 -0700  215)     free_thread_info(tsk->stack);
23f78d4a (Ingo Molnar          2006-06-27 02:54:53 -0700  216)     rt_mutex_debug_task_free(tsk);
fb52607a (Frederic Weisbecker  2008-11-25 21:07:04 +0100  217)     ftrace_graph_exit_task(tsk);
e2cfabdf (Will Drewry          2012-04-12 16:47:57 -0500  218)     put_seccomp_filter(tsk);
f19b9f74 (Akinobu Mita         2012-07-30 14:42:33 -0700  219)     arch_release_task_struct(tsk);
^1da177e (Linus Torvalds       2005-04-16 15:20:36 -0700  220)     free_task_struct(tsk);
^1da177e (Linus Torvalds       2005-04-16 15:20:36 -0700  221) }
^1da177e (Linus Torvalds       2005-04-16 15:20:36 -0700  222) EXPORT_SYMBOL(free_task);
```

コミットID　コミッタ　コミット日時

本人が同意して、このパッチが正式採用されたという流れになっています。

ソースコードだけを見ていても、それが何をするためのコードかはなかなかわかりませんが、gitコマンドでパッチの説明を読むことで理解が進みます。Gitには、このほかにもカーネルソースを読むうえで便利な機能があります。Gitの参考書[2][3]にも、目を通しておくと良いでしょう。

まとめ

Part3では、Linuxカーネルの全体的な役割を説明したうえで、それぞれの機能／役割に対応するソースコードの「ありか」を紹介しました。紙幅の都合もあり、ソースコードの内容を読み解く部分までは踏み込みませんでしたが、まずは、本文で紹介したgitコマンドを使って、さまざまなパッチの内容を眺めていくと良いでしょう。

ちなみに、**図3**の「git blame」コマンドの出力全体を眺めると、「がちゃぴん先生」以外にも多数の日本人の名前が発見できます。日本人を含めて、世界中の開発者がLinuxカーネルの開発に貢献していることが実感できます。本特集を機会に、ぜひ、Linuxカーネルの世界へもう一歩、足を踏み出してください。ソースコードを具体的に読み解く方法については、先ほどの書籍[1]が参考になるでしょう。

◆ ◆ ◆

ちなみに、先月、筆者の新著『オープンソース・クラウド基盤 OpenStack入門』[4]が発売されました。Linuxカーネルにも増して急速に進化を続けるOpenStackですが、特定のバージョンに依存しない、OpenStackを支える技術の「本質」を伝えることを目指した内容となっています。何事も基礎を固めることが大切です。SD

●参考文献

[1] 中井悦司 著『プロのためのLinuxシステム・10年効く技術』技術評論社、2012年
[2] Travis Swicegood 著、でびあんぐる 監訳『入門git』オーム社、2009年
[3] 岡本隆史、武田健太郎、相良幸範 著『Gitポケットリファレンス』技術評論社、2012年
[4] 中井悦司、中島倫明 著『オープンソース・クラウド基盤OpenStack入門』KADOKAWA、2014年

▼図4 「git show c6a7f572」の出力（冒頭部分）

```
commit c6a7f5728a1db45d30df55a01adc130b4ab0327c
Author: KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com>
Date:   Mon Sep 21 17:01:32 2009 -0700

    mm: oom analysis: Show kernel stack usage in /proc/meminfo and OOM log output

    The amount of memory allocated to kernel stacks can become significant and
    cause OOM conditions.  However, we do not display the amount of memory
    consumed by stacks.

    Add code to display the amount of memory used for stacks in /proc/meminfo.

    Signed-off-by: KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com>
    Reviewed-by: Christoph Lameter <cl@linux-foundation.org>
    Reviewed-by: Minchan Kim <minchan.kim@gmail.com>
    Reviewed-by: Rik van Riel <riel@redhat.com>
    Cc: David Rientjes <rientjes@google.com>
    Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
    Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>

diff --git a/drivers/base/node.c b/drivers/base/node.c
index 91d4087..b560c17 100644
--- a/drivers/base/node.c
+++ b/drivers/base/node.c
(以下、パッチの内容が続く)
```

パッチの承認記録

セキュリティ実践の基本定石【特別編】

ITビジネスの足下を揺るがす大きなバグ

OpenSSLの脆弱性 “Heartbleed”の教訓 後編

前編では「Heartbleed」として知られることとなったOpenSSLのHeartbeat Buffer Overreadの脆弱性の経緯や、それをとりまく状況を説明しました。後編ではOpenSSLのソースコードの中に踏み込んで問題点の理解を試みます。

すずきひろのぶ
suzuki.hironobu@gmail.com

理解のための準備

OpenSSLの問題を理解するために、まずは、2つのことを確認したいと思います。具体的には、C言語の特性と、動的にメモリを確保するmalloc()関数の2つです。

C言語

まずはC言語の生い立ち[注1]を眺めてみましょう。C言語はUNIXをアセンブラから高級言語に書き換えるためにベル研のDennis Ritchie氏らによって設計され、実装された言語です。

それ以前のBCPLやBといった言語に影響を受けていますが、注目すべきは、C言語は研究のためではなく、具体的にUNIXというOSを書き直すために作られたということです。もともとは非常に柔軟で多様な表現ができ、コンパクトな言語仕様で、そのためコンパイラも（オプティマイズを考えなければ）シンプルで作りやすい利点があります。

「多様な表現」という言葉を使いましたが、いろいろな書き方を許している言語です。**リスト1**を見てみましょう。char型の配列とchar型へのポインタを使っています。bufferで8バイトの領域を確保し、bufferを配列として表現するだけではなく、ポインタにbufferの場所を代入し、その内容にアクセスするというプログラムです。このプログラムが最後の行に達したとき、bufferの中は**図1**のような並びになっています。「不定」とはコンパイラやOSの実装により、予期しない値が入っていることを意味します。同じ領域を配列とポインタを使って指し示せるという極めてシンプルでプリミティブな言語です。

C言語を例えるなら職人が使う切れ味の良い専用

▼図1　リスト1のbufferの中身

A	不定	X	不定	不定	不定	Y	Z

▼リスト1　配列とポインタ

```
main() {
  char buffer[8];
  char *a, *b;
  buffer[0]=0x41;
  buffer[7]=0x5A;
  a=buffer;
  b=buffer+4;
  *(a+2) = 'X';
  *(b+2) = 'Y';
}
```

注1）The Development of the C Language　Dennis M. Ritchie　http://cm.bell-labs.com/who/dmr/chist.html

切削工具、あるいは料理人が使う恐ろしく切れ味の良い包丁と言えます。その代わり使い方を間違えると大怪我をします。

今でこそISO/IEC 9899:2011として、がっちりした規格になっていますが、1989年に最初のANSI規格が出るまでは、1978年発行のDennis Ritchie氏とBrian Kernighan氏の共著『プログラミング言語C』が事実上の言語規格で、仕様としてあいまいな部分もありました。またC言語はUNIXのために作られているので、当然ながら標準ライブラリはUNIXの機能を前提としています。

malloc(3)

UNIXにはC言語から関数の形で呼び出せるライブラリインターフェースとして、システムコールとユーザライブラリの2つがあります。前者はOSの機能を呼び出すものであり、後者はユーザ権限としてプログラム内で実行されるライブラリ関数です。UNIXでは、システムコールとユーザライブラリではマニュアルのセクションが違い、システムコール(2)、ユーザライブラリ(3)という表現をして区別します。関数malloc()は、ユーザ権限で動作するライブラリですのでmalloc(3)と表現します。

malloc(3)は動的にメモリ空間を確保する関数です。**リスト2**では128バイトの領域を確保し、ポインタpに引き渡しています。

malloc(3)はGNU/Linux標準ではglibcのmalloc実装が使われます。こちらはデフォルトでは内部でmmap(2)を使ってメモリ空間を確保しています[注2]。

malloc(3)はglibcのmallocの実装だけではなく、これと互換性のあるGoogle提供のtcmallocや、FreeBSDで採用しているjemallocなど複数の実装があります。

malloc(3)は内部でシステムコールを呼び出し、ページサイズ(通常4KB)のバウンダリで、ある程度大きなサイズのメモリ空間をあらかじめ確保し、それを要求されたサイズのメモリ空間に分割して引き渡します(これを「アロケーション」と呼びます)。切り出すときは、malloc(3)の引数で指定したサイズに対応するエリアのほかに、頭の部分にメタデータ(制御情報)を入れます(**図2**)。

メタデータとはアロケーションしたメモリ領域の情報を保持する部分で、たとえばglibcのmallocの場合、**リスト3**のような構造体になっています[注3]。

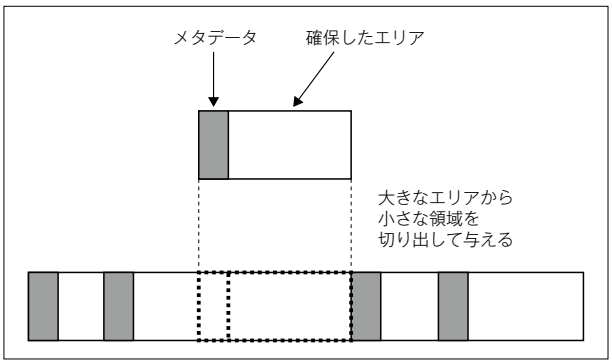
注2) メモリ空間は、古典的なUNIXではbrk(2)/sbrk(2)を使いましたが、POSIX.1-2001でbrk(2)/sbrk(2)はシステムコールから外されています。互換性を残すためLinuxカーネルでもbrk(2)/sbrk(2)のシステムコールを残していますが、その内部動作は古典的UNIXでのbrk(2)/sbrk(2)とは違います。

注3) mallocでアロケーションしたメモリ領域が不必要になったとき、free(3)を行うと、その領域は再利用するためのリストに保持されます。次に、mallocが呼ばれたときに、再利用が必要ならば、そのリストからアロケーションされます。このリストへの出入りをメタデータの情報を使って管理しています。

▼**リスト2　malloc(3)**

```
#include <stdlib.h>
main(){
  char *p;
  p = malloc(128);
}
```

▼**図2　malloc(3)内部での配置のモデル**



▼**リスト3　glibcのmallocが使うメタデータ(/usr/include/malloc.h)**

```
struct mallinfo
{
  int arena;    /* non-mmapped space allocated ➥
from system */
  int ordblks;  /* number of free chunks */
  int smblks;   /* number of fastbin blocks */
  int hblks;    /* number of mmapped regions */
  int hblkhd;   /* space in mmapped regions */
  int usmblks;  /* maximum total allocated ➥
space */
  int fsmblks;  /* space available in freed ➥
fastbin blocks */
  int uordblks; /* total allocated space */
  int fordblks; /* total free space */
  int keepcost; /* top-most, releasable (via ➥
malloc_trim) space */
};
```

さて、**リスト4**はmallocを呼び出し、データを書き込んだ状態でポインタの参照を先に進める（buffer overreadと呼ばれます）とどうなるかを実験するプログラムです。使っているmalloc(3)はUbuntu 12.04.4 LTSの標準で使われているglibcのものです。

allocdata()の中で、mallocしたエリアの先頭にA、B、Cと文字を書き込んでいます。このA、B、Cは秘密情報だとしましょう。ポインタは、ある領域を指しているだけですので、参照先を進めていくと、書いてある秘密の内容が読めてしまいます（**図3**）。

▼リスト4　buffer overreadの実験プログラム

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define LISTNUM 21
void allocdata (void)
{
  int i;
  char *d[LISTNUM];
  for(i = 0 ; i < LISTNUM ; i++) {
    d[i] = (char *) malloc (16);
    d[i][0]='A'+i;
  }
}
void main (void)
{
  int i;
  char *p;
  p = (char *)malloc(1); /* 読み出すための場所を取る */
  allocdata (); /* テストデータ生成中 */
  for (i = 0; i < 1024; i++,p++){ /* 1024バイト分進める */
    putchar (*p);
  }
}
```

図3でダンプしている内容はglibcのmallocを使った場合の例で、すべての実装で同じというわけではありません。tcmallocやjemallocでは、内部のデータ構造が当然違ってきます。

バグとmalloc()

malloc()で得たバッファ領域にオーバーライトすると、並びの次にある領域のメタデータを破壊し、次のデータ領域まで破壊します。そしてその先のデータも、さらにその先も……となります。

その場合、デバッキングがたいへんです。ある特定のモジュールでデータを破壊しているとすると、その副作用が現れるのは、破壊されたデータ領域を使っている全然別のモジュールであったりします。破壊しているモジュールを見つけるのは簡単ではありません。Purify[注4]のようなメモリのデバッキングツールが出てくるまでは、ずっとたいへんな状況でした。

注4）現在は、Rational Purifyという名称になってIBMが販売しています。 http://www-03.ibm.com/software/products/en/rational-purify-family

▼図3　リスト4の実行結果（glibcのmallocの場合）

```
% ./a.out | od -c
0000000  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0   !  \0  \0  \0  \0  \0  \0  \0
0000040   A  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000060  \0  \0  \0  \0  \0  \0  \0  \0   !  \0  \0  \0  \0  \0  \0  \0
0000100   B  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000120  \0  \0  \0  \0  \0  \0  \0  \0   !  \0  \0  \0  \0  \0  \0  \0
0000140   C  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000160  \0  \0  \0  \0  \0  \0  \0  \0   !  \0  \0  \0  \0  \0  \0  \0
0000200   D  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000220  \0  \0  \0  \0  \0  \0  \0  \0   !  \0  \0  \0  \0  \0  \0  \0
0000240   E  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000260  \0  \0  \0  \0  \0  \0  \0  \0   !  \0  \0  \0  \0  \0  \0  \0
0000300   F  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000320  \0  \0  \0  \0  \0  \0  \0  \0   !  \0  \0  \0  \0  \0  \0  \0
0000340   G  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
（.. 以下略 ..）
```

OpenSSLの脆弱性

いよいよ本題です。OpenSSLのバージョン1.0.1から1.0.1fまでは、Heartbeat Buffer Overreadのバグが入っていました。このバグは、相手から送られるHeartbeatのバッファの実際の大きさと、それを示すバッファ長の値が異なることで発生します。意図的にバッファ長の値を、実際のバッファサイズよりも大きくすることで、メモリアロケーションで使っていた領域をオーバーリードさせられます。

リスト5はRFC6520で定義されているHeartbeatの通信のメッセージのデータ構造です。

ペイロードの長さを表すpayload_lengthは2バイトの正の整数で表現されており、0～65,535までの値を入れられます。実際のペイロードであるpayloadのサイズは、本来ならpayload_lengthに入っている値と同じサイズです。つまり、最大で65,535バイトのサイズにできます。次のpaddingのサイズは任意のサイズです。

規格上ではHeartbeatMessage全体のサイズは、16,384（2^{14}）バイト、もしくはRFC6066で定義しているmax_fragment_length（最大の値は2^{12} = 4,096となる）ということになっています。よってペイロードのサイズは、RFC6520では2^{14} = 16,384 = 16KBです。

ちなみに、padding_lengthは最低でも16バイト以上のランダムなサイズです。たぶんこのpaddingのエリアは暗号通信時に盗聴側のトラフィック・アナリシスを回避するために付けているのでしょう。

どうして問題が発生するのか

クライアントからサーバにHeartbeatMessageのメッセージを送ったときの動作を説明します。

▼**リスト5　Heartbeatメッセージの構造体**

```
struct {
  HeartbeatMessageType type;
  uint16 payload_length;
  opaque payload[HeartbeatMessage.payload_length];
  opaque padding[padding_length];
} HeartbeatMessage;
```

■ **Step 1：　クライアントからサーバにHeartbeat Messageを送る**

Step 1-1：payload_lengthを決める

Step 1-2：payload_lengthのサイズ分のpayloadのエリアを確保する

Step 1-3：paddingを確保する

Step 1-4：サーバにHeartbeatMessageを送る

■ **Step 2：　サーバがHeartbeatMessageを受け取る**

Step 2-1：先頭1バイトをHeartbeatMessageTypeとする

St2p 2-2：続く2バイトをpayload_lengthにする

Step 2-3：続くpayload_length分の長さをpayloadとしてバッファに保持する

■ **Step 3：　サーバからクライアントにHeartbeatMessageを戻す**

Step 3-1：戻すためのHeartbeatMessageにHeartbeatMessageTypeの値をセットする

Step 3-2：同じようにpayload_lengthの値をセットする

Step 3-3：payloadを保持しているバッファからpayload_length分をHeartbeatMessageにあるpayloadバッファにコピーする

Step 3-4：HeartbeatMessageのpaddingをセットする

Step 3-5：クライアントにHeartbeatMessageを送る

ここでクライアントからサーバに届いたHeartbeatMessageのpayloadの実際のバッファのサイズよりも、payload_lengthの値を大きくすることで、本来のpayloadのためのバッファの領域をこえて、メモリ領域をコピーしてしまいます。

もちろん、payload_lengthの長さのチェックを行い、不整合が発生しないかどうかを判断するコードを入れるべきなのですが、そのチェックが入っていませんでした。そのためにプログラム中で、使っているほかのメモリ内容までコピーして、外部に（クライアントに）送ってしまうことになりました。

バグの原因となったソースコード

では、具体的にOpenSSLのソースコードの該当個所を見てみましょう。ここではopenssl-1.0.1fを対象に説明します。OpenSSLの中の問題のコードは、ssl/d1_both.cとssl/t1_lib.cに存在しています。基本的にロジックは同じですので、ここではt1_lib.cの該当部分2552～2598行目に解説を加えたいと思います。**リスト6**がその問題の部分となります。ここではクライアントからサーバにHeartbeat Requestが送られた、という前提で説明しています。

OpenSSLの修正内容

では、**リスト6**のコードがどう修正されたかも見てみましょう。基本的には、先ほどのコードの次の2行の前後にペイロードのバッファ長が正しいかどうかのチェックを入れています。

```
2562 hbtype = *p++;
2563 n2s(p, payload);
```

リスト7が修正後のコードです。これでオーバーランはしなくなります。

最初のパッチが入っているopenssl-1.0.1gのコードはクイックハックだからなのかもしれませんが、

▼**リスト6　openssl-1.0.1gのssl/t1_lib.c（Heartbeat Buffer Overreadの該当個所）**

```
2552 #ifndef OPENSSL_NO_HEARTBEATS
2553 int
2554 tls1_process_heartbeat(SSL *s)
2555 {
2556 unsigned char *p = &s->s3->rrec.data[0], *pl;
2557 unsigned short hbtype;
2558 unsigned int payload;
2559 unsigned int padding = 16; /* Use minimum padding */
2560
2561 /* Read type and payload length first */
2562 hbtype = *p++;
2563 n2s(p, payload);
2564 pl = p;
(.. 略 ..)
2571 if (hbtype == TLS1_HB_REQUEST)
2572 {
2573 unsigned char *buffer, *bp;
2574 int r;
2575
2576 /* Allocate memory for the response, size is 1 bytes
2577 * message type, plus 2 bytes payload length, plus
2578 * payload, plus padding
2579 */
2580 buffer = OPENSSL_malloc(1 + 2 + payload + padding);
2581 bp = buffer;
2582
2583 /* Enter response type, length and copy payload */
2584 *bp++ = TLS1_HB_RESPONSE;
2585 s2n(payload, bp);
2586 memcpy(bp, pl, payload);
2587 bp += payload;
2588 /* Random padding */
2589 RAND_pseudo_bytes(bp, padding);
2590
2591 r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
2592
(.. 略 ..)
2597
2598 OPENSSL_free(buffer);
```

- pの値はクライアントから届いたHeartbeatMessageの内容となっている
- hbtypeはHeartbeatMessageTypeの値が入っている。具体的にはHeartbeatのrequestか、responseのどちらかとなる
- plにpayloadのバッファのポインタがセットされている
- n2sはpから2バイトを取って整数値にしてint payloadに値を入れる。つまり、クライアントに戻すpayloadバッファのサイズになっている。Heartbeat Buffer Overread攻撃では、この時点でpayloadが不正な値に設定されることになる
- クライアントに戻す内容を確保するためのバッファ領域をアロケーションしている。OPENSSL_mallocは独自実装ではなく、内部でmalloc(3)を呼んでいるだけのラッパーの役割を果たしている。OPENSSL_mallocに関しての詳細は後述
- payloadは実際のplのサイズよりも大きく、ほかの領域までクライアントに戻すバッファに書き込んでいる。つまり、ここがバッファのオーバーランしている実際の部分である
- padding部分を埋めている
- クライアントにHeartbeatのレスポンスを送っている。この時点で、情報が漏洩する
- bufferを開放する

筆者は、この修正は「本来の処理としては、まだ足りない」という気がしています。

RFC6520の記述では“The padding_length MUST be at least 16.”（最低でも16バイトでなければならない）とあり、16バイト以上で全体のメッセージのサイズと不整合が起こらなければ任意長であるように読めますので、固定的に16バイトで良いのかという疑問があります。

また、“The total length of a HeartbeatMessage MUST NOT exceed 2^14 or max_fragment_length when negotiated as defined in [RFC6066].”という記述があり、仕様では最大16KBまでしか許していないように読めるのですが、このコードでは最大約64KBまで許容するように見えます。暗黙のうちに、ここまでの前の段階で16KBに収まっている可能性はありますが、最終段階で明示的なチェックはされていません。

このような極めて単純なバグを出すことからもわかるように、（あとから付け加えられた）Heartbeat関連のコードはあまり質が高くないのは確かなようです。

OPENSSL_malloc関数

リスト6に出てくるOPENSSL_malloc()/OPENSSL_free()は、crypto/crypto.hの中で定義されているマクロで、CRYPTO_malloc()/CRYPTO_free()に展開されます。CRYPTO_malloc()とCRYPTO_free()はcrypto/mem.cで定義されています。

CRYPTO_malloc()とCRYPTO_free()の主な役目はデバッグです。どのモジュールファイルの何行目で呼び出されたかをダンプします。デフォルトでは、内部で標準ライブラリのmalloc、realloc、freeを呼び出しています。

これらのコードが用意されているsrc/crypto/mem.cやsrc/crypto/mem_dbg.cを見ると、デバッグ用のデータダンプのためのコードが、山ほど組み込まれています。Purifyといった開発ツールの利用などを前提とせず、オリジナルでデバッグ環境を組み入れてチェックしていたのでしょう。先ほどmalloc()関連でバグが発生するとたいへん面倒だと説明しましたが、そのためにこのようなデバッグ環境を組み込んだのは想像に難くありません。

実際に秘密情報は流出するのか

OpenSSLはSSL証明書や、実行中の暗号鍵（復号するための鍵）、あるいは暗号通信をしたあと、復号した内容を保持するために、大量のデータをmallocで確保したエリアに保持しています。

しかし、Heartbeatで送られてきたHeartbeat Messageをメモリ内のどこにアロケーションするかは、事前にはわかりません。動的にメモリをアロケーションしていますので、mallocがプールしている大きな領域のどこに割り当てられるかは運です。ですが、mallocの大きなメモリ領域には、確実にデータは残っていますし、その領域のどこかにHeartbeat Requestで送られてきたHeartbeat Messageはアロケーションされます。そして、そこから続く最大約64KBのメモリエリアをコピーしてHeatbeat Responseで戻します（外部に流出させます）。

そこで、openssl-1.0.1gに用意されているdemos/ssl/serv.cppとcli.cppの各ファイル名をserv.cとclip.cに変更して、そのmallocのエリアをダンプして、どういう情報が漏れるか実験してみます。

serv.cppとcli.cppに**図4、図5**の変更を行ったあと、serv.cppとcli.cppをコンパイルして、サーバ

▼**リスト7　openssl-1.0.1gのssl/t1_lib.c（修正後）**

```
2597 if (1 + 2 + 16 > s->s3->rrec.length)
2598 return 0; /* silently discard */
2599 hbtype = *p++;
2600 n2s(p, payload);
2601 if (1 + 2 + payload + 16 > s->s3->rrec.length)
2602 return 0; /* silently discard per RFC 6520 sec. 4 */
```

この条件が真なら、payloadのバッファ長の値が0より小さいこととなり整合性が取れないこととなる

この条件が真なら、payloadのバッファ長の値が実際に送られてきたデータサイズよりも大きいということになり整合性が取れないこととなる

▼図4　cli.cppの変更個所

```
% diff cli.cpp cli.c
38c38
<   SSL_METHOD *meth;
---
>   const SSL_METHOD *meth;
41c41
<   meth = SSLv2_client_method();
---
>   meth=TLSv1_2_client_method();
97c97
<   err = SSL_write (ssl, "Hello World!", ➥
strlen("Hello World!"));  CHK_SSL(err);
---
>   err = SSL_write (ssl, "SecretSecret", ➥
strlen("SecretSecret"));  CHK_SSL(err);
```

▼図5　serv.cppの変更個所

```
% diff serv.cpp serv.c
31,32c31,32
< #define CERTF  HOME "foo-cert.pem"
< #define KEYF  HOME  "foo-cert.pem"
---
> #define CERTF  HOME "test-cert.pem"
> #define KEYF  HOME  "test-cert.pem"
52c52,55
<   SSL_METHOD *meth;
---
>   const SSL_METHOD *meth;
>   char *dummy;
>
>   dummy=(char *)malloc(1);
56c59
<   SSL_load_error_strings();
---
>
58c61,62
<   meth = SSLv23_server_method();
---
>   SSL_load_error_strings();
>   meth=TLSv1_2_server_method();
150a155,159
>
>   while (dummy++)
>     fprintf(stderr,"%c",*dummy);
>
>
```

serv、クライアントcliを作ります。そして、serv、cliをそれぞれ実行します（**図6**）。なお、このserv.cのコードでは、可能な限りメモリ内容をアクセスしダンプしますので、最後は“Segmentation fault (core dumped)”で終了するのが、正しい終わり方になります。

クライアント側から送られている“SecretSecret”という文字列がdumpされているイメージ内にあれば情報が漏れてしまう、ということです。では見てみましょう（**図7**）。

実際にSecretSecretという文字列を見つけることができました。パスワードなど文字で入っているようなもの、あるいはセッションIDも含めてCookieに秘密情報を設定しているものも入手可能だということを、このダンプは意味しています。そのまま文字列で見えるので、ダンプをすれば簡単に目視できます。もちろん、サーバのSSL証明書、公開鍵、秘密鍵、また実行中の暗号鍵もすべてこのダンプしたデータに入っています。バイナリですが、データ構造がわかっているので、トライ・アンド・エラー的に探すことが可能です。

流出した情報を取れるかどうかは、別の問題

繰り返しになりますが、実行時に動的メモリでアロケーションに使われるメモリ領域がどうなるかは、そのときにならないとわかりません。またTLS/SSLは、httpsだけではなくVPNや、ほかの暗号通信のレイヤとしても使われています。メモリがどうマッピングされるかは、その通信でのメモリの使われ方やプログラムの作りに大きく左右されるので、どうなっているかは一概に言えません。ですから、秘密情報が入手できる確率は高いにしても、攻撃が成功するか否かは確定的ではないのです。

その逆も言えます。たとえば、SSL-GATWAY-PROXYのような、あまり内部的にメモリのアロケーションを必要としない構造をしているプログラムの場合、64KBもあればまるごと秘密鍵などの情報が入ってくる可能性も否定できません。

また、今回はglibcのmalloc()を前提にしていますが、サーバがtcmallocを使っていたり、jemallocを使っていたりすると、もっと複雑にメモリ領域に展開しているので作業はより複雑です。しかし、秘密情報が漏洩するという本質は変わりません。

まとめ

今回のHeartbeat Buffer Overreadを筆者なりに

まとめてみます。

①TLS/SSLは暗号技術を用いて情報を保護するためのものであるにもかかわらず、その実装が原因で情報が漏洩するという、あってはならない状況が発生した

②この漏洩は攻撃により確実に起こるものであるが、この攻撃が外部から行われていたか否かを知る術はOpenSSL側にはなく、また、ログにも残らないので監査が実質不可能である

③攻撃はタイミングに依存し、どんな情報が漏洩したかもOpenSSL側では察知することができない

④漏洩する可能性があるものは、サーバの動的にアロケーションされたメモリ上に存在しているものすべてである。具体的にはセッション中に使われたパスワードも含むユーザの情報など。セッションの暗号鍵も(たとえ使われたあとでも、まだ完全に破棄されていなければ)漏洩する危険性がある。またサーバのSSL証明書のようないったん漏洩するとサービスにとって致命的な結果をもたらすものもある

⑤コーディングミスと思われがちであるが、指定されたエリアのサイズと実際に確保しているサイズで不整合を起こさないか、あるいは仕様を満たした使い方がされているかを確認するという手順を怠った設計上の問題である

⑥今回のバグは、OpenSSLに限らずUNIX/C言語では古くから何度も繰り返されている、典型的な動的メモリアロケーションのバグである。結果として極めて重大なセキュリティ問題を発生するにもかかわらず、問題は極めて単純。しかし、プログラミングで見落としがちであり、デバッキングも容易ではない

どんなにソフトウェアの品質技術が向上しても、人間が作る以上、完全なソフトウェアというものは存在せず、今後も、単純なミスが大きな問題を引き起こすことはあるでしょう。だからこそ、前回言及したような脆弱性流通のしくみが作られており、それが上手に機能することが重要なのです。

TLS/SSLをお使いの方は、すでにOpenSSLの最新版に入れ替えたかと思いますが、早めにSSL証明書(公開鍵証明書)も新しいものに取り替えることをお勧めします。

今回の件で、ユーザのパスワードなど機密情報が漏れたという可能性は否定できません。それを十分に考慮する必要があります。

今回だけではなく、今後も新しい脆弱性が起因となって重要情報漏洩やパスワード漏洩が起こる事案が必ず持ち上がってくるでしょう。今回の問題をきっかけに、そのような場合のリスク管理も視野に入れて運用を考えていただければ幸いです。SD

▼図6　servとcliをそれぞれ実行

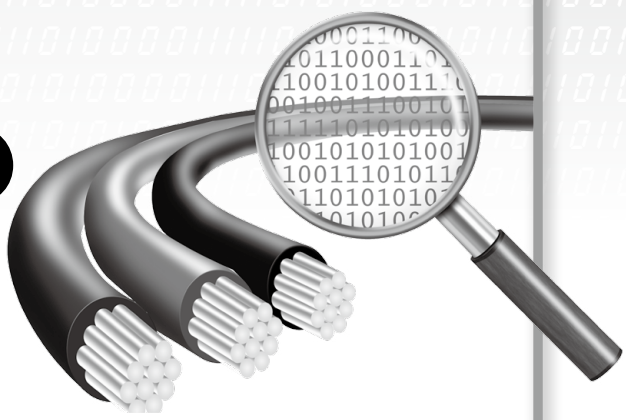
```
 サーバ側を実行
$ ./serv > /dev/null  2> dump
$ Segmentation fault (core dumped)
 これで dump の中に malloc の領域のイメージが入っているはず

 別シェルでクライアントを動かす
$ ./cli
```

▼図7　図6で取得したdumpの内容

```
$ od -c dump
0000000  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0 301  \0  \0  \0  \0  \0  \0  \0 260
0000040   1 257 001  \0  \0  \0  \0   z 261   C  \0  \0  \0  \0  \0   X
 (.. 略 ..)
0431000  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0 360
0431020   U  \0  \0  \0  \0  \0  \0   P   E  \0  \0  \0  \0  \0  \0  \0
0431040  \0  \0  \0  \0  \0  \0  \0   4 003   $ 227   *   e 332 273   S
0431060   e   c   r   e   t   S   e   c   r   e   t   g 207 264   w 344
0431100   F 212   % 247 177   ^ 355 303   I   k   X 201 352 177 336 342
0431120   x 261 254 250   c 303   0 317 325   s 265   F   d 314 336   ]
 (.. 以下略 ..)
```

ネットワークを眼で読む

使ってみよう! tcpdump



ネットワークを流れるデータはシステムのトラブル解決やセキュリティ認識にとって多くの情報を提供してくれます。誰でも簡単なコマンドで見ることができる便利さと怖さの両面を知っておいてほしい、ITエンジニア必修の基本中の基本のお話です。

後藤 大地(ごとう だいち)　(有)オングス　代表取締役

ネットワークを読む、tcpdump(1)

ネットワークを流れるデータは何もしなければ人間の目には見えませんので、ついほかの人にも見えないものとしてとらえてしまいがちです。しかし、ネットワークを流れているデータは物理的に接続することができれば簡単に覗き見することができます。これは良い面と悪い面の双方を抱えています。

良い面からいくと、ネットワークを流れているデータを把握することでネットワークの設定の確認、ソフトウェア動作の設定の確認、パフォーマンス向上のための調査データの取得、ネットワークトラブル発生時の問題の調査などを簡単に実施できることを意味しています。まともな仕様書もないような状態でデータセンターに放り込まれてシステムを組み上げるといった仕事はざらにありますので、こういうときにこうしたことができないと手も足もでない状況になります。

悪い面は、データが誰にでも読めてしまうのでプライバシーに関するデータやユーザ名、パスワードなどが簡単に盗聴されてしまう点にあります。しかし、実際にそうした現実に触れておくことで、プライバシーやセキュリティに対する認識が改まるという面はありますので、どのみち一度はネットワークを眼で読んでおくとよいでしょう。

そうした場合に使うツールがtcpdump(1)[注1]です。現在おもに使われているUNIX系オペレーティングシステム(以下、OS)ではどれでも使用できますので、本稿を機に使ってみてください。

tcpdump(1)初体験

現在ではtcpdump(1)の実装系はすべて同じです。tcpdumpコマンドに-hをつけてバージョン番号を表示させるとわかります。OSごとにバージョンは異なりますが、同じ出力を確認できます(**図1〜3**)。

まずは実行してみましょう。**図4**のように管理者権限でtcpdump(1)コマンドを実行して、「password」といったキーワードに関する出力だけを抜き出すようにしてみます。grep(1)コマンドの-iオプションは"大文字と小文字を区別せずにどちらも一致させよ"という指定です。

しばらく放置しておくと**図5**のようにずらずらと文字列が出力されます。この出力を見るだけでは意味がわからないと思いますけれども、自分は送信したつもりがなくても、デフォルトで動作しているサービスやソフトウェアは背後

注1) コマンドの後ろに付いている括弧書きの数字は、manコマンドで見られるマニュアルの章番号を表しています。

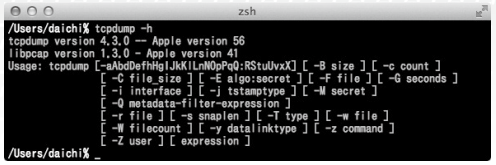
でネットワーク通信をおこなっており、その中には「password」というキーワードで絞り込んだだけでも結構なデータが流れていることがわかります。このように誰でも見られるものなんだという認識を持っておきましょう。

tcpdump（1）の使い方

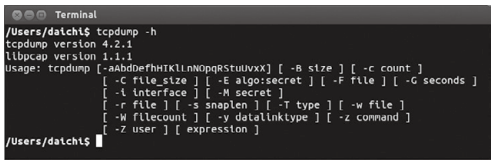
tcpdump（1）は**図6**のように使います。なにもオプションや条件を指定しないと、そのマシンに流れているすべてのパケットの通信情報を表示します。オプションを指定することで表示するデータの内容を変更したり、条件を指定することで表示するパケットを絞り込むことができます。

いくつもオプションがありますが、とくによく使われるオプションをまとめると**表1**のよう

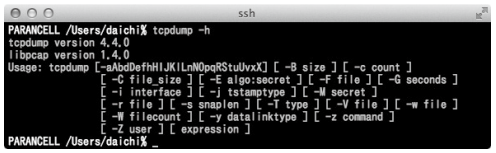
▼図1　tcpdump（1）／Mac OS X Mavericks



▼図2　tcpdump（1）／Ubuntu 12.04 LTS



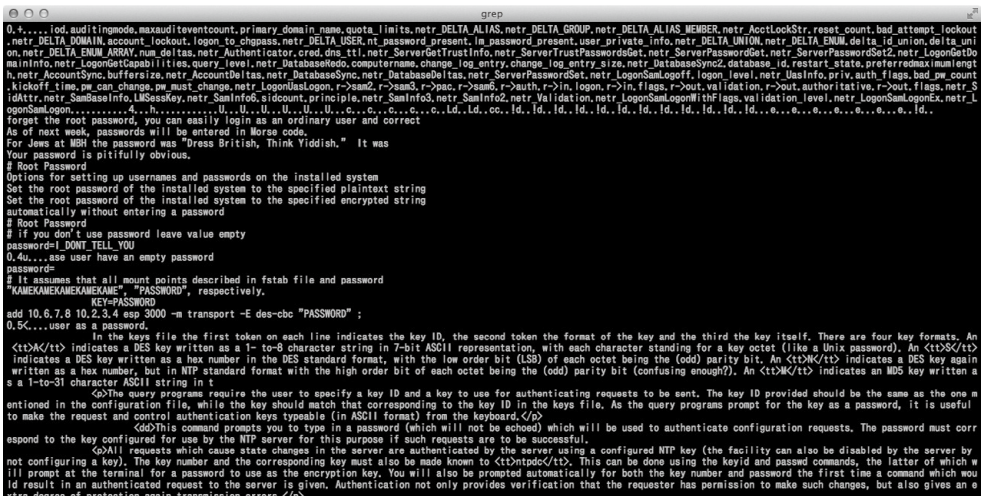
▼図3　tcpdump（1）／FreeBSD 10.0-RELEASE



▼図4　tcpdump（1）で指定したネットワークインターフェースに流れているデータを表示

```
sudo tcpdump -A | grep -i password
```

▼図5　コマンドの実行結果



▼図6　tcpdump（1）の基本的な使い方

```
tcpdump オプション［条件］
```

▼表1　tcpdump（1）でよく使われるオプション

オプション	意味
-i インターフェース	指定したネットワークインターフェースをモニタリングしてパケットをキャプチャ
-w ファイル	指定したファイルへキャプチャしたパケットデータを保存
-r ファイル	指定したファイルからキャプチャデータを読み込んで処理
-A	パケットをASCIIで表示
-X	パケットをHEXおよびASCIIで表示
-c 回数	指定した回数分だけパケットをキャプチャし、その後終了
-n	ホストアドレス、ポート番号などを名前へ変換しない
-p	プロミスキャスモードで動作しない
-s バイトサイズ	表示するパケットサイズを指定（デフォルトは64KB）

▼表2　tcpdumpの代表的な使い方

コマンド	内容
tcpdump -A -i インターフェース	指定したネットワークインターフェースに流れるデータをすべて表示
tcpdump -A -i インターフェース port 80	ポート番号80を指定しているパケットを表示
tcpdump -i インターフェース -w ファイル	指定したファイルにキャプチャしたパケットを書き出し

▼表3　条件で使用できるキーワード

修飾子の種類	指定できる値
type	host、net、portrange
dir	src、dst、src or dst、src and dst、ra、ta、addr1、addr2、addr3、addr4、inbound、outbound
proto	ether、fddi、tr、wlan、ip、ip6、arp、rarp、decnet、tcp、udp

▼表4　条件の指定例

条件	意味
dst host ホスト	IPパケットの行き先が指定されたホストになっているもの
src host ホスト	IPパケットの送信元が指定されたホストになっているもの
host ホスト	IPパケットの行き先または送信元が指定されたホストになっているもの
gateway ホスト	指定したホストをデフォルトゲートウェイとして使っているパケット
dst net ネット mask ネットマスク	IPパケットの行き先が指定されたネットワークに含まれているもの
src net ネット mask ネットマスク	IPパケットの送信元が指定されたネットワークに含まれているもの
net ネット mask ネットマスク	IPパケットの行き先または送信元が指定されたネットワークに含まれているもの
dst port ポート番号	パケットの行き先ポート番号が指定されたポート番号になっているもの
src port ポート番号	パケットの送信元ポート番号が指定されたポート番号になっているもの
port ポート番号	パケットの行き先ポート番号または送信元ポート番号が 指定されたポート番号になっているもの
dst portrange ポート番号1-ポート番号2	パケットの行き先ポート番号が指定されたポート番号範囲の中にあるもの
src portrange ポート番号1-ポート番号2	パケットの送信元ポート番号が指定されたポート番号範囲の中にあるもの
portrange ポート番号1-ポート番号2	パケットの行き先ポート番号または送信元ポート番号が 指定されたポート番号範囲の中にあるもの
ip proto プロトコル	パケットのプロトコルが指定されたプロトコルのもの （icmp、icmp6、igmp、pim、ah、esp、vrrp、udp、tcp）
ether broadcast	パケットがイーサネットブロードキャストパケットであるもの
ip boradcast	パケットがIPv4ブロードキャストパケットであるもの
ether multicast	パケットがイーサネットマルチキャストパケットであるもの
ip multicast	パケットがIPv4マルチキャストパケットであるもの
vlan VLANID	パケットが指定されたVLANIDを持ったIEEE 802.1Q VLANパケットであるもの
pppoed	パケットがPPP-over-Ethernetディスカバリパケットであるもの
pppoes	パケットがPPP-over-Ethernetセッションパケットであるもの

になります。なかでも-Aで人間でも読めるテキストでパケットの内容を表示、-iでモニタリングするネットワークインターフェースの指定、-wでキャプチャしたデータをファイルへ出力、あたりは覚えておきたいところです。本当に簡単にですが、よく使われるような書き方をすると**表2**のようになります。

また、条件で指定できるキーワードにはtype、dir、protoという3つの種類があります。指定できるキーワードを分類するとそれぞれ**表3**のとおりになります。どういったキーワードがあり、どのように指定するのかはpcap-filter(7)のマニュアルにまとまっていますので、基本的にはこのマニュアルを読みながら条件を書くことになります。とくによく使うような条件指定を**表4**にまとめておきます。

実践1 HTTPプロトコルを読み取ってみよう

Webサーバとのやりとりをtcpdump(1)で覗いてみましょう。たとえば**図7**のようなtcpdump(1)を実行すると、ネットワークインターフェースen0を流れるHTTP関連のパケットをキャプチャできます。

このコマンドの実行前にHTMLファイルを用意しましょう。Webサーバ側では/admin/index.htmlファイルにアクセスされると、ベーシック認証を経た後に**リスト1**のHTMLファイルを返すように設定しておきます。

ブラウザからアクセスすると**図8**のようにベーシック認証の後でページが表示されます(**図9**)。

この状態で**図7**を実行すると**図10**のようなテキストを出力します。図中に示したとおり、どのIPアドレスのどのポートから、どのIPアドレスのどのポートへデータが送信されているのかがわかります。パケットの種類も表示されていますし、HTTPリクエストでは「Authorization: Basic ZGFpY2hpOjEyMzQ1Njc4」のようにベーシック認証で使われるハッシュ値も確認できてしまいます。

実行するコマンドはとても単純ですけれども、これだけでネットワークインターフェースに到達するさまざまな種類のパケットの中身を見ることができます。問題発生時の調査からセキュリティ脆弱性の発見など、さまざまなことが可能です。

▼図7　HTTPのパケットをキャプチャするコマンド指定

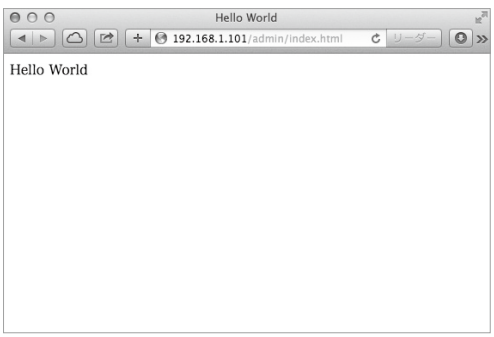
```
sudo tcpdump -A -i en0 port http
```

▼リスト1　用意するHTMLファイル

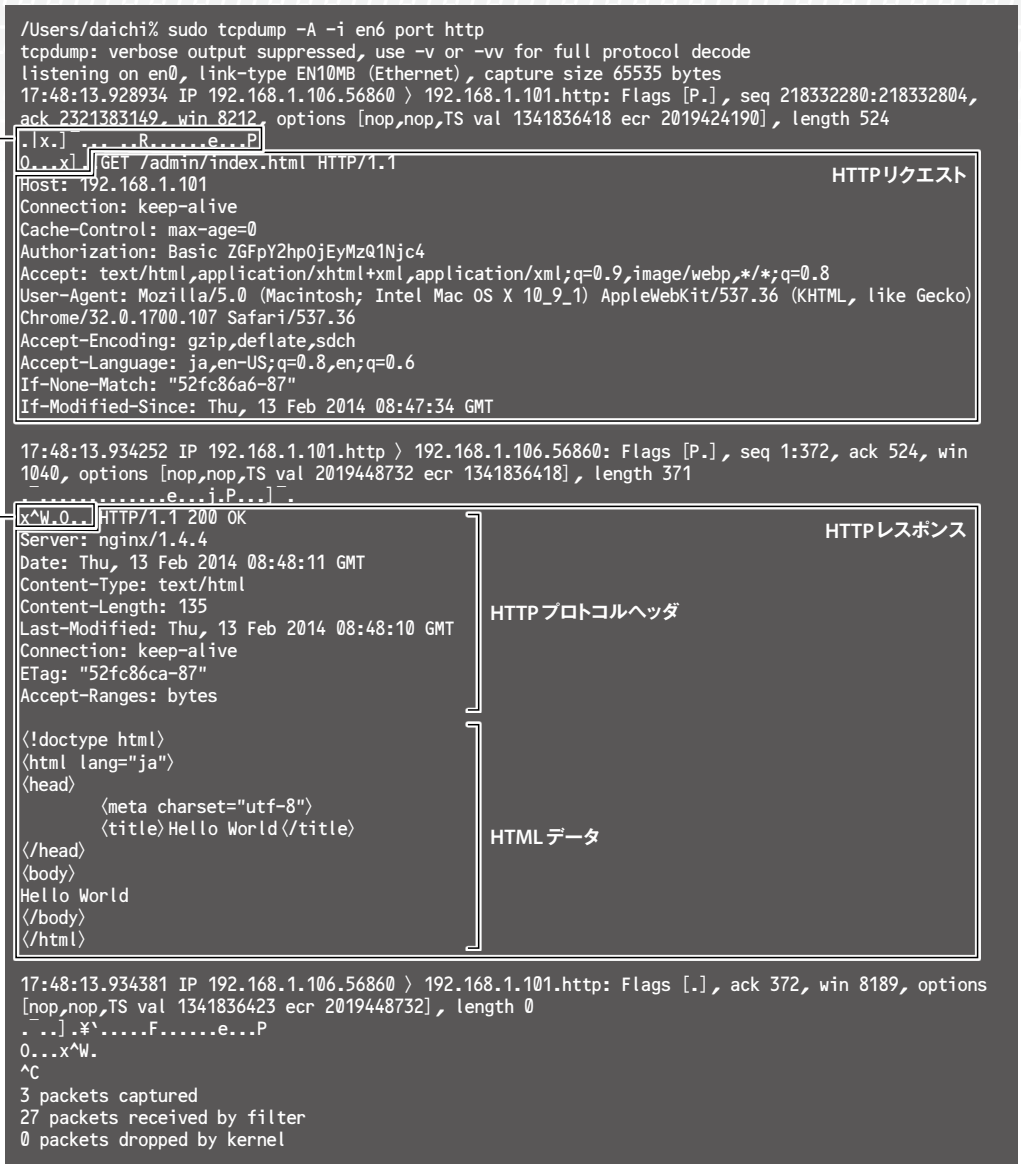
```
〈!doctype html〉
〈html lang="ja"〉
〈head〉
        〈meta charset="utf-8"〉
        〈title〉Hello World〈/title〉
〈/head〉
〈body〉
Hello World
〈/body〉
〈/html〉
```

▼図8　ブラウザからのアクセス

▼図9　表示されたページ



▼図10　キャプチャコマンドの実行結果



実践2　パケットをもっと詳しく分解してみよう

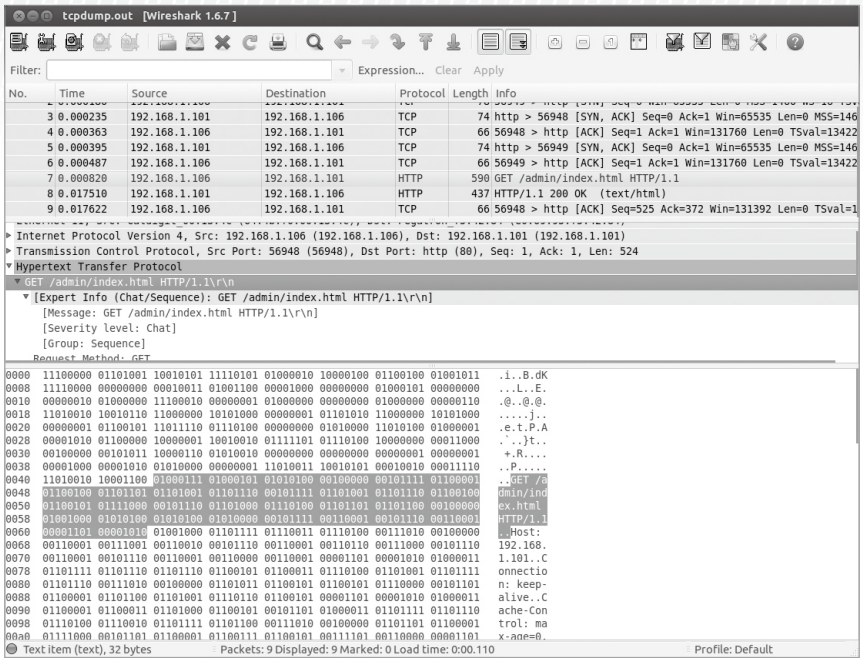
tcpdump(1)がキャプチャしたパケットデータは-wオプションを使うことでファイルに保存できます。いったんファイルに保存しておけば、あとからtcpdump(1)で解析したり、別のソフトウェアを使って分析するといったことができます。先ほどのHTTPパケットのやり取りを**図11**のようにいったんファイルに保存します。

▼図11　キャプチャしたデータをファイルへ出力

```
sudo tcpdump -A -i en0 -w tcpdump.out port http
```

tcpdump(1)のデータを解析するソフトウェアはいくつかありますが、視覚的に操作するとなると「Wireshark」が便利です。WiresharkはOSが提供している

▼図12　取得したキャプチャデータをWiresharkで解析



▼表5　キャプチャされたHTTPレスポンスを含むフレームの大枠の構造

	ヘッダやデータ	バイト数
1	Ethernet IIヘッダ	14バイト
2	IPv4ヘッダ	20バイト
3	TCPヘッダ	32バイト
4	HTTPプロトコル	236バイト
5	HTMLデータ	135バイト

パッケージ管理システム経由でインストールするか、Wiresharkのサイト[注2]からバイナリパッケージをダウンロードして利用できます。

図12のように、Wiresharkではペインが3つに分かれています。1番上のペインで調査したいパケット（イーサネットフレーム）を選択し、2つ目のペインでヘッダやフィールドを選択します。選択結果は3つ目のペインで強調表示されるしくみです。この機能を使うと、実際に送られてきたデータにどのようなデータが含まれているのかを詳しく知ることができます。

今回の例ですと、たとえばHTTPレスポンスが含まれているパケットを調べてみると、次のような構造になっていることがわかります。先頭から順にイーサネットパケット（フレーム）のヘッダ、IPv4のヘッダ、TCPのヘッダ、とヘッダが続いたあとでHTTPプロトコルがあり、最後にHTMLデータが送られていることがわかります（**表5**）。

こうした構造になっていることは仕様書などに掲載されていますのでよく見かけますが、実際に解析したデータで同じものが確認できると、なにかこう実感がわいてきませんか。フィールドまで含めてさらに詳しく分析していくと**表6**のような構造になっていることが確認できます。

OSのTCP/IPスタックはNICから得られるこうしたデータを加工して、最終的にソケットを経由して、ユーザランドで動作するソフトウェアがネットワーク通信できるようにデータを提供しているわけです。なかなか普段は気にしない部分ですが、内部ではこうした処理を繰り返し繰り返し実行しながら、ネットワーク通信を実現しているわけですね。

注2）http://www.wireshark.org/

▼表6　HTTPレスポンスを含むパケットのより細かな内容を整理

ヘッダやデータ	フィールド	値	サイズ
Ethernet IIヘッダ			14バイト
	行き先のMACアドレス	64:4b:f0:00:13:4c	6バイト
	送信元のMACアドレス	e0:69:95:f5:42:84	6バイト
	データの種類	0x0800(IP)	2バイト
IPv4ヘッダ			20バイト
	バージョン	4	1バイト
	ヘッダの長さ	20	1バイト
	DSフィールド	0(DSCP)	1バイト
	トータルの長さ	423	2バイト
	識別子	0xffb5	2バイト
	フラグ	0x02(フラグメントしていない)	3ビット
	フラグオフセット	0	13ビット
	TTL	64	1バイト
	プロトコル	6(TCP)	1バイト
	ヘッダチェックサム	0xb57b	2バイト
	送信元IPアドレス	192.168.1.101	4バイト
	行き先IPアドレス	192.168.1.106	4バイト
TCPヘッダ			32バイト
	送信元ポート番号	80	2バイト
	行き先ポート番号	56948	2バイト
	シーケンス番号	1	4バイト
	確認応答番号	525	4バイト
	ヘッダの長さとその他もろもろ	32	1バイト
	ウィンドウサイズ	1040	2バイト
	チェックサム	0x80a3	2バイト
	オプション	01:01:08:0a:12:1e:d2:9d:50:01:d3:95	12バイト
HTTPプロトコル		HTTP/1.1 200 OK¥r¥nServer: nginx/1.4.4¥r¥nDate: Thu, 13 Feb 2014 08:48:11 GMT¥r¥nContent-Type: text/html¥r¥nContent-Length: 135¥r¥nLast-Modified: Thu, 13 Feb 2014 08:48:10 GMT¥r¥nConnection: keep-alive¥r¥nETag: "52fc86ca-87"¥r¥nAccept-Ranges: bytes¥r¥n¥r¥n	236バイト
HTMLデータ		〈!doctype html〉¥n〈html lang="ja"〉¥n〈head〉¥n¥t〈meta charset="utf-8"〉¥n¥t〈title〉Hello World〈/title〉¥n〈/head〉¥n〈body〉¥nHello World¥n〈/body〉¥n〈/html〉¥n	135バイト

とにかく使ってみよう！

なにか問題でも発生しない限りtcpdump(1)は使うことがないかもしれませんが、このコマンドでどういったことができるのかを知っておくと、トラブルが発生したときなどに問題の切り分けが簡単になります。便利なコマンドですので、まだ使ったことがなければ使ってみてください。

また、データを視覚化することでセキュリティに対する意識も高まります。こうした通信を確認すると、ssh(1)やHTTPSで通信することの安全性などが身をもって体感できます。SD

BOOK no.1 オープンソース・クラウド基盤 OpenStack入門
構築・利用方法から内部構造の理解まで

中井 悦司、中島 倫明 【著】
B5変形判、208ページ／価格＝1,800円＋税／発行＝KADOKAWA
ISBN＝978-4-04-866067-9

OpenStackはクラウド基盤を構築するためのコンポーネント群。こう聞くだけでは、抽象的でわかりにくい製品だが、本書では実際の構築例を示しながら解説されるので、製品の機能や使い方が具体的に理解できる。前半は、物理マシン1台だけで実現できるIaaSを構築する。本書の手順に沿って設定作業を行えば、仮想マシン／仮想ネットワークの管理や、クラウド基盤の何たるかがわかってくる。後半では、複数の物理サーバを使った本格的なIaaS構築を試すが、実際にはKVMで1台の物理マシンに複数の仮想サーバを立てて実習する。このように、個人でも試せるよう例題が工夫されているので、実際に動かして学習してみてほしい。

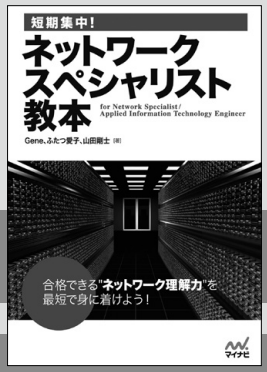
BOOK no.2 クラウド技術とクラウドインフラ
—黎明期から今後の発展へ—

黒川 利明 【著】
A5判、192ページ／価格＝2,300円＋税／発行＝共立出版
ISBN＝978-4-320-12374-8

クラウドが普及するまでの歴史や社会的背景、クラウドを支えるネットワーク技術・仮想化技術を幅広く解説した、クラウドの教科書。また、Amazon EC2やGoogle Appsなど、現存のさまざまな企業のサービス・製品が紹介されており、クラウド業界の全体像を概観できる。

「クラウドと紛らわしい言葉や概念」の節では、クラウド、グリッド、ユビキタスといった、似た概念を持つまぎらわしい言葉を紹介し、登場経緯や現状をふまえながら、違いを説明している。次々と新しい技術や専門用語が生まれるクラウド業界だが、言葉ひとつひとつの意味をとらえ、基礎知識を充実させることが、変化についていくうえで重要となるはずだ。

BOOK no.3 短期集中! ネットワークスペシャリスト教本



Gene、ふたつ愛子、山田 剛士 【著】
A5判、368ページ／価格＝2,800円＋税／発行＝マイナビ
ISBN＝978-4-8399-5081-1

ネットワークスペシャリスト試験の教本。各節で例題として取り上げられている「午後問題」は、企業の新技術導入といった実務的な事柄を扱うことが多いので、その節で扱われるネットワーク技術がイメージしやすい。

ネットワーク分野で肝心なTCP/IPプロトコルの章は、ひとつひとつの例題のボリュームが大きく、その解説も詳しいので、問題を解きながら重点的に学ぶことができる。最近よく話題に上る仮想化技術の章では、ホストOS型・ハイパーバイザ型の違いが、図を使ってわかりやすく解説されている。難易度は比較的やさしく、ネットワーク分野を概観する入門書として読むこともできるだろう。

BOOK no.4 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／価格＝2,880円＋税／発行＝技術評論社
ISBN＝978-4-7741-6516-5

Ruby on Rails（以下Rails）の初級～中級者を対象とした解説書。Railsのインストールから、基礎・応用まで、幅広くカバーしている。Railsを使ったWebアプリについては、その開発面だけでなく、VagrantやChef、Capistranoなどを使った「運用面」についてもページが多く割かれており、実践的な内容となっている。また、MVCの解説、DevOpsの概要など、Railsに限らない、Webアプリ開発全般に通じる知識も豊富だ。さらに仮想マシンの構築についても説明があるので、Railsを学びながら、Webアプリ開発のフローも習得・再確認できるだろう。第10章ではRailsの拡張を扱っており、上級者を目指す人にとっても充実した内容となっている。

新連載

サーバーワークスの瑞雲吉兆仕事術

第1回　クラウド前夜「スマートフォンの登場」

クラウドの登場によって、情報システムのありかた、そしてエンジニアのキャリアそのものが大きく変わろうとしています。本連載では、クラウドの登場とそれによって情報システムがどのように変わろうとしているのか、そしてエンジニアのキャリアがそれによってどのように変わろうとしているのか、すでに起こりつつある変化を読み取ることで、みなさんが自分のキャリアを考えるための材料を提供するものです。

Writer ㈱サーバーワークス　代表取締役　大石 良（おおいし りょう）／ http://blog.serverworks.co.jp/ceo/

エンジニア人生を主体的に生きるために

みなさんご存じのとおり、クラウドの台頭によってシステム開発の現場が大きく変わろうとしています。ビジネスの面では大きなメリットのあるクラウドですが、エンジニアの視点では「とりあえずJavaかPHPあたりができれば何とかなった」という牧歌的な時代が終わるという大変革を強いられることが間違いないという、いささか迷惑な状況でもあります。

こうした変革が訪れる中、これからエンジニアはどのようなキャリアプランを描けば良いのか、エンジニアに将来どのような可能性があるのか、読者のみなさんと一緒に考察することでより良いエンジニアライフを送る一助になりたいという想いで筆を執りました。

まずは、クラウドに至るまでのITの流れを振り返ることで、過去数年に登場した技術や製品が、筆者たちの生活、エンジニアの存在にどのようなインパクトを与えてきたのかをおさらいしたいと思います。

モバイルの始まり

「モバイルの始まり」などと大げさな見出しを付けましたが、みなさんにとってモバイルコンピューティングの始まりとはどのあたりでしょうか？　Jornada？　ウルトラマンPC？　ご存じのない方はぜひこの機会にググってみてください。こんな時代もあったんです（**写真1**）。

筆者のモバイル生活はPalm Pilotが始まりでしたが、当時の端末はネットにつなぐことも一苦労で、今のモバイルコンピューティングのスタイルとはかけ離れたものでした。「通信機能とデバイス本体が高度に融合された、今のスタイルのモバイルコンピューティング」という意味では2005年にSHARPから発売された

▼写真1　HP Jornada680（ジョルナダ、上）、IBM Palm Top PC 110（ウルトラマンPC 下）写真提供：PDA博物館

▼写真2　W-ZERO3（縦、横でも使用できる。しかもミニキーボードも展開する）写真提供：シャープ株式会社

W-ZERO3がスタートだと思います（**写真2**）。

著者の会社でも、インフラのお守りをするためにW-ZERO3をエンジニアに貸与して、いつでもどこでもSSHできるようにしていました。今思うとありがた迷惑な話ですが、当時は「どこでも仕事をしなければいけない」というプレッシャーよりも「W-ZERO3のようなユニークな端末を会社が貸し出してくれる」というメリットのほうが大きくとらえられていたようで、おおむね好評だったようです。

意外な共通点

自分たちでモバイル端末を持つことで、その後使うことになる「クラウド」と「モバイル」との意外な共通点がわかってきました。

それは、

「**コンピュータのある場所に行かなくて良い**」

ということです。

当たり前のように聞こえますが、これは2006～2008年くらいの時点では非常に画期的なことでした。2006年頃ではまだまだ「何かあったらデータセンターにかけつける」ことが半ば当たり前とも言える状況で、当社のエンジニアが取っていた記録でも1年間にデータセンターに駆けつけて作業する時間は、年間90時間を超えていました。

ところが、モバイル端末を持つようになってデータセンターに駆けつける、作業のために会社に行くということが激減します。そして実は、クラウドもまったく同じように「駆けつけ作業」をなくすことができたのです。

知り合いのコンサルタントの方が、こんなことを仰っていました。

「モバイルとクラウドというのは、実に人間的なテクノロジだ。これまでのコンピューティングは**コンピュータの場所に人間が行く**ことが前提だったが、モバイルもクラウドも**人間の場所でコンピュータを操作できる**初めてのテクノロジだ」。

筆者たちは、これをのちに痛感することになりますが、そのエピソードは次稿に譲ります。

スマートフォンによる変革

W-ZERO3などのスマートフォンで、モバイルコンピューティングの可能性に大きな期待を寄せていたところに、モバイルに大変革をもたらすあの端末がやってきます。そう、iPhoneです。

2008年にiPhone 3Gが発売されたとき、まだまだ一部の人が持つオタク向けガジェットという趣でしたが、筆者たちはW-ZERO3ですでにモバイルの予習は終わっていましたので、すぐにそのパワーを理解できました。会社から貸与する端末も徐々にiPhoneに切り替えていき、iPhoneありきの業務に少しずつ変更していったのです。

Google Appsへ

こうしてiPhoneが業務で使われるようにな

ると、「メールをGoogle Appsへ換えよう」という話になっていったのです。

それまではずっと「管理がしやすい」などの理由でExchangeなどのメールシステムを選んでいたのですが、今回は違います。初めてユーザ視点、つまり「iPhoneとの相性が良いものは何か？」という視点で、クラウド上のメールサービスに置き換えようという話になったのです。こうした動きは、今でこそ「コンシューマライゼーション」という言葉で表現されていますが、当時は筆者たちもこれが良い選択なのかどうか、半信半疑という状態での選択でした。——ですが、結果からいうと筆者たちの想定以上にGoogle AppsとiPhoneの相性がよく、それによってカレンダーの共有などiPhoneを業務で使う場面が多くなり、この選択は結果的に良かったと考えています。

クラウド

そして、Google Appsの体験は筆者たちの事業そのものにも大きな影響を与えることになります。筆者たちは2007年からAWS（Amazon Web Services）を試験的に使い始めており、2008年1月には「社内サーバ購入禁止令」を出して、より踏み込んでAWS（当時は「EC2」と呼ぶことがほとんどでした）を使っていましたが、Google Appsの体験から「どうもGoogleやAmazonなど大規模事業者に運用まで任せたほうが良さそうだ」と感じるようになってきたのです。こうして、モバイルデバイスが起点となり、少しずつGoogle、Amazonなどのクラウド利用を増やしていくことになるのです。

蓄積の消失

さて、このようにモバイルデバイスがクラウドの導入を後押しし、ITのサービスを「使う」ようになってくると、筆者たちにも次のような変化が訪れることになりました。

- **メールサーバを立ち上げなくなった**
- **物理サーバを買わなくなることで、インストールや設置などの業務が激減した**

2008年頃は、メールサーバの立ち上げも一苦労でした。qmail + vpopmailという組み合わせが多かったのですが、2002年ごろから6年くらいかけてさまざまな環境でqmailをインストールしていた筆者たちは、シェルスクリプトなどを作ってqmailをスピーディにインストールしたり、迷惑メールが大量に飛んできたときの対応策を共有したりするなど、実にさまざまなノウハウを蓄積してきました。ところが、Google Appsによってそれらが一瞬で必要なくなってしまったのです。

これは筆者たちにとって最初の衝撃でした。**「技術を蓄積しても、一瞬でそれが必要なくなるかもしれない」**。

今までも同じようなことが言語の世界ではあったかもしれませんが、少なくとも書いたコードのメンテナンスという業務は残り続けることが約束されており、「運用の必要性が皆無になる」ということは起きていなかったように思います。それが、クラウドでは「構築業務も、運用ノウハウも、どちらも必要ない」ということが起きる。クラウドの登場によって、技術の選別が決定的に大切になる、と思い知らされた最初の出来事だったのです。このことは後からもう少し詳しく触れます。

エンジニアの姿

さて、ここまでで2005年から2009年くらいの間に起きた大きな変化を振り返ってみました。この時点で本誌を購読している読者の中で「エンジニア」というと、おおよそ次に挙げる3パターンの、いずれかに属しているのではないかと思います（あくまで「想定」です）。

◈(1)SI企業のエンジニア

ユーザにシステムを納入する立場のエンジニア。上流の方は設計やプロジェクトの進行、ベンダーコントロールに関する業務が中心で、下請けのSIerの方はJavaのコーディングが中心。下流の方は「コードも書けないやつが設計やるなんて」などとグチをこぼすのがお約束。

◈(2)ユーザ企業のIT部門

自社にシステム、IT環境を提供する立場のエンジニア。というと聞こえは良いが実際には過去のシステムの保守・運用、サーバやネットワークのお守りだけで手一杯。しかもベンダーをコントロールするだけなので技術は身につかない。SIer側にいた人が酷い扱いに嫌気がさして転職するが、自分が同じことをやってしまうという悪しき循環。

◈(3)ベンチャー・Web系のサービス開発エンジニア

自社で提供するサービスを開発する立場のエンジニア。何やら最先端の技術に触れられそうだと意気込んで転職してみたものの、思ったより分業が進んでおり技術を学ぶ機会が限定的。技術より課金。昔いたCTOとか呼ばれる人が作ったオレオレフレームワークでできたゲームをコピー&ペーストしてパラメータをちょっと変えるだけの簡単なお仕事。

ここではエンジニア像をわざとネガティブなイメージで描いています。

なぜなら、現状に満足していて将来も楽観している方は本稿に目を通すことはないだろうと思われるからです(笑)。

ですが、モバイルとクラウドという2つのトレンドは、上の3パターンのどのキャリアの方にとっても、将来をまったく変えてしまう大きなインパクトがあります。

しかし、それは決してネガティブなことばかりではないと思います。筆者たちがかつてそうだったように、なかなか楽観的なキャリアプランを描くことが難しい、不確実性の高い中でも、新しいサービス、モノに触れ、時代の流れをちゃんととらえることで、明るい未来を創ることは必ずできると信じています。

本連載で、IT業界がどのように変わりそうなのか、そして前述のイメージに当てはまるエンジニアの方々にどのような変革が訪れるのか、一緒に考えていきます。

次回予告

こうしてみると、思いのほか速いスピードで「モバイル」と「クラウド」という破壊的なテクノロジが筆者たちの身の回りに普及していったことがわかると思います。そしてすでにこの時点で、筆者たちの仕事に変化が訪れてきています。筆者は、すべてのエンジニアがクラウドとモバイルという2つの破壊的なテクノロジの時代でも幸せなキャリアプランを描くことができると確信していますが、それには正しい前提の認識や、これから必要とされる技術・能力の把握が不可欠です。そのためにも、筆者たちの例を挙げて過去を振り返り、「これから何が起きそうなのか」もっと言うと「すでに起こっていることで、未来に影響を及ぼすものは何なのか」を把握する必要があると考え、第1回目は「モバイル」と「クラウドの」のスタートをひもといてみることにしました。

さて、第2回目ではSalesforce、AWSというクラウド時代の本命サービスが立ち上がってくるにあたり、これらのサービスが世の中に与えたインパクトがどれくらい大きかったのか、具体的にどんなプロジェクトがあって、エンジニアの将来にどんな影響を及ぼそうとしているのか、具体的な事例と数字を基に考えていきます。SD

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://d.hatena.ne.jp/rubikitch/ http://www.rubyist.net/~rubikitch/

第4回 SKK＋AZIKで快適・効率的な日本語入力を!

Emacsによる文字入力一元化を実現するために、スムーズな日本語入力は必要不可欠なものです。本稿では、それを手助けするシンプルな日本語入力システム「SKK」、習得が簡単な拡張ローマ字入力システム「AZIK」を紹介しています。実際にインストール&入力してみて、その便利さ、普段の日本語入力との違いを実感してください。

SKKとは

ども、Emacs廃人のるびきちです。Emacsの最大の恩恵とは何か覚えていますか？　そう、「文字入力の一元化」でしたね！　つまり、文章の入力・編集をEmacsに一任すれば快適になるということです。それだけEmacsの編集能力はすごいものがあります。Emacsスキルが上がってくると、ブラウザのtextareaで文章を書いていた自分自身がバカらしくなってくるはずです。

ではその第一歩として、Emacs上での日本語入力システムの1つSKKを今回紹介します。SKK は Simple Kana to Kanji conversion programの略で、シンプルな操作性が持ち味です。筆者はEmacsを使うのと同時にSKKを使い始め、今でも手放せないツールとなっています。

SKKでの日本語入力方法はやや特殊ですが、少し練習すればすぐに慣れます。優れたチュートリアルがついているので御安心ください。

単語変換＋文節マニュアル指定

SKKは単語変換がベースとなっています。「え、今さら単語変換の日本語入力システム？」と思われるかもしれませんが、シンプルさを追求した結果、単語変換に行き着きました。一見非効率に思われそうな入力方式ですが、入力効率は悪くはありません。単語変換だからこそ実現できることもあります。

SKKの最大の特徴とも言えるのが「かなと漢字の境界を人間が指定すること」です。文節の区切りはコンピュータではなくて人間が指定するのです。最近は日本語の解析を高精度で行えるようになったものの、通常の連文節変換では文節区切りミスをゼロにはできません。ひらがなだけでは文節の区切りが不明な例すらあります。有名な文が「ここではきものをぬいでください」で、「ここでは着物を脱いでください」とも「ここで履物を脱いでください」とも変換できます。ひらがな文字列でこの文のみが与えられた場合、いかに精巧なアルゴリズムをもってしても、どちらの変換が正しいかは決定できないのです。これが日本語の宿命です。

そして、連文節変換における文節区切りミスの修正はかなり面倒でイライラするものです。文節を判断するアルゴリズムは難しいです。それならいっそのこと文節は人間が指定すればいいじゃないかというのがSKKの結論です。「単語変換＋文節マニュアル指定」こそがSKKであり、シンプルそのものです。面倒かもしれませんが、正しく入力する限り、文節区切りミスが根絶できるのはうれしいものです。SKKを使っていて文節を間違ったとき、それは自分が入力

を間違ったということですので自己責任です。ものすごく潔いスタンスです。

SKKでは、かな→漢字、漢字→かなに移るときに大文字を使います。SKKモードのとき、普通にローマ字入力していればひらがなが入力されます。そして、漢字が登場したらその読みのローマ字の先頭文字を大文字にします。また、送り仮名が登場したときも先頭文字を大文字にします。たとえば「動く」は「UgoKu」と入力します。

スムーズな単語登録

単語変換という一見時代遅れに思えるシステムですが、単語変換だからこそ実現できた機能が存在します。シンプルかつ強力な単語登録、そして特殊変換です。

連文節変換での単語登録は面倒なものです。読み(見出し語)と漢字だけでなく、品詞情報まで登録する必要があるのですから。たとえばATOKの顔文字辞書の1エントリはこうなっています。[TAB]がタブ文字で、前後にスペースは入っていません。「独立語*」というのが品詞情報です。

```
ぱんち [TAB] o -_-)＝○)゜O゜) [TAB] 独立語*
```

SKKは文節区切りという日本語の一番難しい部分を人間に任せているため、品詞という概念は存在しません。淡々と見出し語を変換するだけです。あるのは、見出し語と変換候補のテーブルのみです。変換できない場合は、すぐに単語登録モードに入り、ユーザは変換後の文字列を入力するだけで単語登録が終わります。単語登録は入力を妨げるのではなく、呼吸をするように自然な流れになっています。

言葉は生き物で、どんどん新語が登場してきます。アーチスト名、キャラクタ名などの固有名詞もどんどん増えてきます。アニメや漫画では長く難しい漢字名の必殺技だって登場してきますね。辞書のデータが古い場合、新語はもちろん登録されていないので、出てくるたびに単語登録することになります。連文節変換では品詞情報も指定する必要があるため、単語登録は面倒そのものです。SKKはあっさり単語登録が終わるので、新語の対応は簡単です。

SKKの単語登録は再帰的に行われます。たとえば、「再帰」と「的」と「頭字語」のみが変換できる場合に「再帰的頭字語」に変換しようとします。そのとき、「さいきてきとうじご」はそのまま変換できないので、単語登録モードに入ります。そこで「再帰的」+「頭字語」で変換しようとしても、今度は「さいきてき」が変換できないので、再び単語登録モードに入ります。このとき、単語登録モードの中で単語登録モードに入りました。落ち着いて「再帰」+「的」で「さいきてき」を単語登録します。すると、「再帰的」が入力されている状態で「さいきてきとうじご」の単語登録モードに戻るので、「頭字語」と入力すれば「さいきてきとうじご」が単語登録されます。

言葉で書くとすごく複雑そうですが、実際に再帰的単語登録をやってみればごくごく自然な流れだとわかります。品詞情報不要の単語変換だからこそスムーズにできるのです!

SKKのラージ辞書はかなり多くの単語が含まれていますが、複合語はあまり含まれていません。複合語を変換しようとすると、しばしば単語登録モードに入ります。たとえばブログなどでよく使われる表現の1つ「超絶便利」は入っていないので、そういう場合でも落ち着いて単語登録してください。

複合語を登録しておくと、しばしばタイプ数が削減できます。なぜなら、入力中に見出し語補完する機能が備わっているからです。「超絶便利」と変換したあとに「ちょ」と入力したら「ちょうぜつべんり」が見出し語候補に登ります。

特殊変換emacs

ほかにもアルファベットや記号を見出し語にして変換する機能が存在します。「computer→コンピュータ」、「skk→Simple Kana to Kanji conversion program」などです。さらに、skkの

変換を進めていくとSKKのバージョンが出てくるというおもしろい機能もあります。

応用例として、特殊変換を簡単なデータベースとして使えます。たとえば、「emacs→http://www.gnu.org/software/emacs/」のようにURLやメールアドレスを登録することもできます。

単語変換だからといって馬鹿にはできないでしょう？　シンプルながらも多機能で柔軟的なのがSKKなのです。

日本語入力についてはGoogle日本語入力が有名ですね。Google日本語入力はGoogleの高性能サーバを使って膨大なデータから最適かつ正確な変換をしてくれます。もちろん連文節変換もできます。サーバ経由ですので、自分で単語登録せずともすでに新語が登録されています。APIが提供されているので、SKKからGoogle日本語入力の機能を使うことすらできます。サーバ経由ですので一見遅いようですが、もちろんキャッシュが用意されているので問題ありません。SKKなのに連文節変換もできてしまいます。シンプルなSKKだからこそ、両者の強味を共存させられるのです。

いろいろなSKK実装

SKKはもともとEmacs Lispで実装されていますが、SKKファンはEmacsの外でもSKKを使いたくなるものです。そのため、いろいろな環境への移植版が作られています。Windows用の「SKK日本語入力FEP」、Mac OS X用の「AquaSKK」などです。

現在のEmacs用SKKはddskk（Daredevil SKK）です。SKKという名前ではとっくのとうに開発終了しており、ddskkという名前になって開発が続けられています。よって、Emacsの文脈でSKKといえば自動的にddskkとなるわけです。本稿でもSKK＝ddskkと認識してください。

SKKはEmacs Lispでのみで書かれているので、Emacsさえ動けばどのの OSであっても同じ操作性で日本語入力できます。英語版のOSであっても、問題なく日本語入力できます。新しいコンピュータを使うときに、EmacsとSKKさえインストールしてしまえばいいのですから。複数のOSを使う人にとって、日本語入力システムを使い分ける必要がないのは、とてもうれしいことではないでしょうか。

インストール

APTなどのOSのパッケージシステムでddskkが存在するのであれば、そこからインストールしてください。Debian系列のGNU/Linuxでは「sudo apt-get install ddskk skkdic」でおしまいです。Emacs初心者がGNU/Linuxで使う場合ならば、これが無難です。その時点で初期設定は済んでいます。

ddskkのアーカイブからインストールするのはちょっと手間がかかります。展開したら、まずdicディレクトリに移動し、辞書ファイルをダウンロードして置いてください[注1]。その後makeします。もし、うまくいかない場合はSKK-CFGファイルを編集してください[注2]。

```
$ cd dic
$ wget http://openlab.ring.gr.jp/skk/ ➡
skk/dic/SKK-JISYO.L
$ cd ..
$ make what-where
emacs -batch -q -no-site-file -l SKK-MK -f
SKK-MK-what-where Loading /home/rubikitch/
emacs/ddskk-15.1/SKK-CFG...

SKK modules:
  skk-viper, skk-jisx0213, ...(略)...
  -> /usr/local/share/emacs/24.3/site-lisp
/skk
...(略)...
SKK tutorials:
  SKK.tut, SKK.tut.E, NICOLA-SKK.tut, skk.
xpm
  -> /usr/local/share/skk
$ make install
```

そして、次の初期設定をすれば使えます。~/.emacs.d/init.elに書き加えてEmacsを再起動

注1）http://openlab.ring.gr.jp/skk/skk/dic/SKK-JISYO.L
注2）WindowsではREADMEs/README.w32.jaを参照してください。

してください。

```
;; make what-whereでSKK modulesで表示される
ディレクトリを指定
(add-to-list 'load-path "/usr/local/
share/emacs/24.3/site-lisp/skk")
;; M-x skk-tutorialでNo file found as ～と
エラーが出たときにskk-tut-fileを設定
;; make what-whereでSKK tutorialsで表示される
ディレクトリ上のSKK.tutを指定
(setq skk-tut-file "/tmp/share/skk/SKK.tut")
(require 'skk)
(global-set-key "¥C-x¥C-j" 'skk-mode)
```

Lispファイルの検索パスであるload-pathの設定は重要です。SKKをmakeでインストールした場合は、SKK用のLispディレクトリが作成されるので、load-pathに加えておく必要があります。さもなければskkを読み込んでくれません。

C-x C-jを押すとSKKモードになります。もし**C-x C-j**にdired-jumpが割り当てられている場合は、(require 'dired-x)をglobal-set-keyの前に書いてください。dired-xをロードした時点で**C-x C-j**にdired-jumpが割り当てられてしまうので、あらためてglobal-set-keyでskk-modeに再割り当てするためです。

チュートリアルから始めよう

SKKには優れたチュートリアルがあります。そのため、実際のSKKでの入力方法については本稿では触れません。はじめてSKKを使うときは、チュートリアルに従って手を動かして覚えていってください。チュートリアルは**M-x skk-tutorial**で実行できます。

チュートリアルで十分過ぎるほどの情報量ですので、日常的な日本語入力は一部の機能を使えば十分間に合います。無理に全部覚える必要はありません。ゆっくりでいいです。

拡張ローマ字入力AZIK

ここからはSKKの応用設定の話です。

現在の日本語入力方法の主流は当然ローマ字入力ですね。しかし、ローマ字入力というのは非効率的な入力方法なのです。ひらがな1文字入力するのに、ほぼ2ストローク必要になるのはかなり多くの打鍵数が必要といえます。おまけに漢字変換する必要があり、正しく変換されたかどうかを目視で確認する必要があります。

少しでも入力効率を上げるにはどうすれば良いのでしょうか？

漢直入力は……

その問題を解決する方法として漢直入力があります。漢直入力というのは、漢字変換なしで、直接漢字を入力する入力方式です。2ストロークの組み合わせに1文字を割り当てています。たとえば、T-Codeで使われるキーは40種類程度ですので、2ストロークでは40×40で1,600通りになります。その1,600個それぞれに文字を割り当てれば、2ストロークで漢字やかなを直接入力できるのです。

しかし漢直入力というのは、1文字1文字ストロークを覚える必要があるという重大なデメリットがあります。あなたが小中学生で漢字を1文字覚えるついでに漢直入力のストロークを覚えられればいいのですが、残念ながら日本の教育はそうなっていません。大人になってから覚えるのはとてもつらいもので、漢直入力に挑戦するも挫折した人はたくさんいます。脳科学的に、人間は14歳を過ぎれば丸暗記がしにくくなっているからです。

このように新たな入力方式を習得するには、習得コストが問題になります。漢直入力はローマ字入力といういつもの習慣を捨て去る必要があるうえに、せめてよく使う文字のストロークを覚えるまでは実用になりません。それでは日常業務に差支えてしまいますね。「T-Codeを覚えるために3ヵ月間仕事を休ませてください」なんて通るはずがありません。

ローコスト・ハイリターンのAZIK！

大人にとっては漢直入力は現実的な入力方式ではありません。かといってローマ字入力の非効率さはなんとかしたいものです。そこで、「ローマ字入力改」というべきAZIKという入力方式が考案されました。AZIKならばローマ字入力がベースですので、無意識で行っているローマ字入力という資産を捨てる必要はありません。

AZIKでの入力方法がわからないAZIK初心者であっても、ほとんどのケースで従来のローマ字入力が使えます。よって、AZIKは日常業務に影響することなく、段階的に習得できるのです。筆者もAZIKを愛用しています。

AZIKは実用的なアプローチをとっています。日本語の音韻を研究し、よくあるパターンの入力を簡潔化しています。既存の入力方式は(ローマ字入力もかな入力も漢直入力も)日本語という言語そのものの特性を考えているわけではありません。AZIKは日本語の文章を効率よく入力できるように、ローマ字→かな変換テーブルを拡張しています。

AZIKではおもに**表1**のように入力します。「っ」と「ー」が劇的に打ちやすくなったのは特筆すべき点です。これだけでもAZIKを習得する価値はあります。

母音+は、日本語のパターンから頻出のものを入力しやすくするための拡張です。たとえば母音+aというのは「か」などで、「かん」は「kz」で入力できるようになります。実際にやってみればわかりますが、これらは指の動きを少しずらすだけで効率的に入力できます。慣れないうちは「ん＝q」を使って「kaq」と入力しても良いです。

ほかにも互換キーや特殊拡張もありますが、まずは上記の基本をしっかりマスターしたうえでゆっくりと覚えれば良いです。実は筆者も全部は覚えきれてはいません。詳細はAZIKのWebページ[注3]を参照してください。

▼表1　AZIKでの入力方法

入力対象	キー	備考
ん	q	skk-azikでは@でかなカナ切り替え
っ	;	なんとホームポジションに！
ー	:	わざわざ小指を上に動かす必要なし！
しゃ	xa	しゅ、しょも同様
ちゃ	ca	ちゅ、ちょも同様
わ	lwa	skk-azikではxwa
ぁ	la	skk-azikではxxa
ゃ	lya	skk-azikではxya
母音+aん	z	aの下
母音+iん	k	iの下
母音+uん	j	uの下
母音+eん	d	eの下
母音+oん	l	oの下
母音+aい	q	aの上隣り
母音+uう	h	uの斜め下隣り
母音+eい	w	eの左隣り
母音+oう	p	oの右隣り

skk-azikを使う

SKKでもAZIKに対応しています。次の1行を~/.skkに加えるだけです。

```
(setq skk-use-azik t)
```

なお~/.skkはSKKローカルの設定ファイルで、**M-x skk-restart**を実行すれば再読み込みされます。それ以外は~/.emacs.d/init.elとの違いはありません。

SKKではローマ字に出てこないqやlをモードの切り替えに使っていますが、AZIKではこれらの文字も使っています。よって、SKKのAZIK拡張では一部異なる操作となっています。

小指を守ろう

SKKではかな漢字の区切りには大文字を使います。実際に使っていくとわかりますが、SKKを使い続けていくと Shift を押す小指が疲れて

注3）http://hp.vector.co.jp/authors/VA002116/azik/azikinfo.htm

しまいます。残念ながら、それがSKKの大きな欠点です。

でも御安心ください。Shiftを使わずにかな漢字の区切りを指定する方法があるのです。~/.skkにて変数skk-sticky-keyに区切りキーを設定すれば、Shiftを使わずに快適にSKKが使えるようになります。変換キーや無変換キーは親指で押せるのでお勧めです。たとえば「動く」は「UgoKu」ではなく「無変換ugo無変換ku」で入力できます。

```
;; Windows 環境だと [noconvert]
(setq skk-sticky-key [muhenkan])
```

muhenkanなどのキー名はどうやって取得するのかというと、<f1> cを使います。そのあとに無変換キーを押せば「<muhenkan> is undefined」と出てきます。

一緒にEmacs力を高めませんか？

筆者は毎週土曜日にEmacsのメルマガを発行しています。多くの解説ではその機能の説明に終始しており具体例に乏しいため、理解するのにとても時間がかかるうえ、今の自分に必要なのかどうかを見極めることも難しいです。メルマガではチュートリアル形式で手を動かして学ぶ形式になっているので、たった5分でその内容を習得できるようになっています。わかりにくい資料で長時間悪戦苦闘するか、月々527円で時間差を買うかはあなたしだいです。無制限メール相談権も付けています。初月無料ですので安心して登録してください[注4]。Happy Emacs ing！SD

注4）http://www.mag2.com/m/0001373131.html

シェルスクリプトではじめるAWS入門

――AWS APIの活用と実践

第5回　AWS利用環境の構築(後編)　MFA設定

Writer　波田野 裕一(はたの ひろかず)／運用設計ラボ　operation@office.operation-lab.co.jp

トピック

AWS Command Line Interface(以下「AWS CLI」)に、31番目のコマンドとして"emr"が追加されたことが6月5日に告知されました[注1]。

これは、Amazon Elastic MapReduce(以下「EMR」)を操作するためのコマンドで、すでにその詳細なマニュアルがAWS CLIリファレンスに追加されています[注2]。

2014年6月時点ではまだプレビュー版のため、その利用にはあらかじめ下記のコマンドを実行する必要があります。

・コマンド:

```
% aws configure set preview.emr true
```

このコマンドを実行すると~/.aws/configに下記の行が追加され、以後はemr関連の約20個のサブコマンドが利用できるようになります。

・~/.aws/config:

```
[preview]
emr = true
```

EMRの追加により、AWSマネジメントコンソール上のサービスメニューに表示されている29サービスのうち、AWS CLIが対応していないものは下記の4サービスとなりました。

・workspaces
・cloudfront
　(レビュー版、リファレンスへの記載なし)
・glacier
・appstream

AWS CLIの急ピッチな利便性向上を嬉しく思う一方で、個人的にCloudFrontの正式対応を期待したいところでもあります。

さて今月号の本題に入ります。

今回の流れ

前回は、AWSアカウントの作成後にやっておくと良い下記の4つの作業のうち、請求関連の設定について解説しました。

1. AWS CloudTrail(APIのロギング)の設定(第3回)
2. 作業用AWS Identity and Access Management(以下IAM)ユーザの作成(第3回)
3. 請求関連の設定(前回)
4. 多要素認証(MFA)の設定

注1) URL https://aws.amazon.com/jp/about-aws/whats-new/2014/06/05/announcing-a-preview-of-amazon-elastic-mapreduce-commands-on-the-aws-cli/

注2) URL http://docs.aws.amazon.com/cli/latest/reference/emr/index.html

今回は、4つ目の多要素認証(MFA)の設定について解説します。

> **Note**
> 第2回および前回の記事でも言及しましたが、AWSマネジメントコンソールの操作においては、Google Chromeの利用を推奨します。

MFA(多要素認証)の設定

前回までの設定で、AWSアカウントでなければできないことは請求情報の変更や請求レポートの閲覧以外ほぼ無くなり、日常運用については第3回に作成したIAMユーザで行うことができるようになりました。普段使わなくなったAWSアカウントについては、多要素認証(Multi-Factor Authentication)を導入して防御を固めておきましょう。

AWSアカウントにMFAを設定

MFAは、AWSのサービスにアクセスする際に、専用デバイス(ハードウェアMFAデバイス)もしくはスマートデバイスのソフトウェア(仮想MFAデバイス)などにより生成されるワンタイムの認証コードを要求する認証方式で、なりすまし防止効果の向上が期待できます。

今回は、AWSアカウントに対して、仮想MFAデバイス(iPhone + Google Authenticator)を利用した多要素認証の設定をし、なりすましによるAWSマネジメントコンソールへのサインインの防止を図ります。

▼図1　Google Authenticatorのインストール

手順①　Google Authenticatorのインストール

最初に、利用しているスマートデバイスのアプリケーションストア(iPhoneの場合はAppStore)にアクセスして、Google Authenticatorをインストールしてください(**図1**)。

> **Note**
> https://code.google.com/p/google-authenticator/に入手先へのリンクが掲載されています。

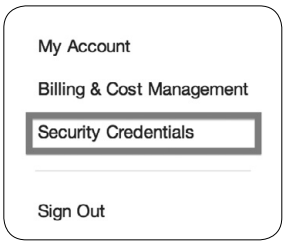
手順②　AWSマネジメントコンソールでの設定

MFAの設定は、Security Credentialsページ[注3]で行います。[マネジメントコンソールの右上のアカウント名]→[Security Credentials]の順にクリックするとアクセスできます(**図2**)。

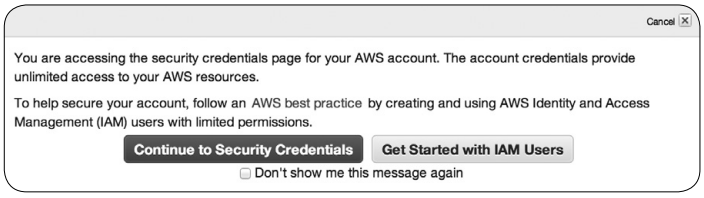
"AWSアカウントは権限が無制限なのでIAMユーザを使いましょう"という趣旨の注意が表示されます。今回はそのAWSアカウントへのアクセスに制約を追加する作業を行うので、[Continue to Security Credentials]ボタンをクリックします(**図3**)。

注3) URL https://console.aws.amazon.com/iam/home?#security_credential

▼図2　コンソールメニュー画面



▼図3　警告表示画面で[Continue to Security Credentials]を押下

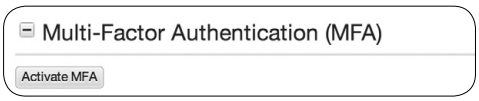


［Multi-Factor Authentication（MFA）］の左の［＋］ボタンをクリックすると［Activate MFA］ボタンが表示されるのでクリックします（**図4**）。

MFAデバイスを選択する画面が表示されるので、今回は［A virtual MFA device］（仮想MFAデバイス）を選択し、［Continue］ボタンをクリックします（**図5**）。

AWS MFA互換アプリケーションのインストールを促すメッセージが表示され、［Continue］ボタンをクリックするとQRコードが表示されます（**図6**）。

▼図4　［Activate MFA］ボタンの表示



▼図5　MFAデバイスの選択

▼図6　QRコードの表示

手順③　Google Authenticator と AWS アカウントの関連付け

QRコードが表示されたら、Google AuthenticatorとAWSアカウントの関連付けを行います。具体的には、Google AuthenticatorにAWSアカウント情報を登録し、表示される認証コードをAWSマネジメントコンソールに転記する作業を2回行います。

まず、スマートデバイスでGoogle Authenticatorを起動します。右上の鉛筆のアイコンをクリックすると、一番下に［＋］ボタンが表示されるのでクリックします。［入力を追加］画面が表示されるので［バーコードをスキャン］を選択します（**図7**）。

Google Authenticator内部でカメラが立ちあがるので、AWSマネジメントコンソール上に表示されているQRコードを読み取らせましょう。正常に読み取れればGoogle Authenticatorの［認証システム］画面に新しい欄が追加されているはずです（**図8**）。

▼図7　［入力を追加］画面から［バーコードをスキャン］を選択

▼図8　［認証システム］画面に新しい欄が追加される

ここに表示されている6桁の数字が認証に必要な認証コードで、30秒周期で更新されます。認証コードは通常は青で表示されていますが、更新5秒前から色が替わり点滅しはじめます。右下に残り時間が円グラフで表示されているので、残り時間が少ない場合は次の更新を待って利用するようにします。

また、数字の下にはラベル欄があり、ここにはアカウント情報が薄字で表示されています。このメモ欄は右上の鉛筆ボタンを押した後に編集可能になるので、忘れないうちに適切な名称に書き換えておくことをオススメします。

Google Authenticator上に認証コードが表示されるようになったら、その数値をAWSマネジメントコンソール上の[Authentication Code 1]欄に転記します。

Google Authenticator上で認証コードが更新されるのを待ち、更新されたらその数値をAWSマネジメントコンソール上の[Authentication Code 2]欄に転記します。この認証コードは、[Authentication Code 1]欄に転記した認証コードのすぐ次に表示された認証コードである必要があります(つまり、2つの認証コードは連続して表示されたものでなければダメです)。

[Continue]ボタンをクリックして"The MFA Adevice was successfully associated."と表示されればMFAの登録は成功です。[Finish]ボタンをクリックしてください(**図9**)。

[Multi-Factor Authentication (MFA)]欄に、今登録した仮想MFAデバイスが表示されていることが確認できます(**図10**)。

手順④　MFAを利用したサインイン

設定が完了すると、これ以降はMFA仮想デバイスなしではAWSマネジメントコンソールへのサインインができなくなります。ほかの端末もしくはブラウザで実際にサインインしてみましょう。

サインアップ画面から通常通りサインインし、パスワード認証が完了すると、[Amazon Web Services Sign In With Authentication Device]画面が表示されます(**図11**)。

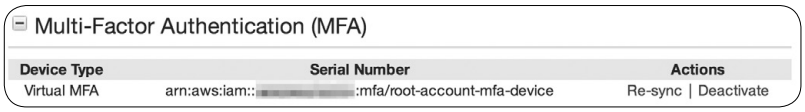
Google Authenticatorに表示される認証コードを[Authentication Code]欄に入力し、[Sign in using our secure server]ボタンをクリックすると、サインインが完了しています。

なお、MFAによるなりすまし防止は、AWSマネジメントコンソールに対しては有効ですが、アクセスキーを利用したAPIへのアクセスに対しては基本的に効果がありません。

MFAの設定の有無にかかわらず、アクセスキーの管理については引き続き注意を払うようにしましょう。

▼図9　MFAの登録の完了

▼図10　仮想MFAデバイスの表示



▼図11　Amazon Web Services Sign In With Authentication Device画面表示

Note

一部のAWSサービスではAPIへのアクセス制御にMFAを利用することが可能です。

MFAの利用を止めるには

MFAの利用をやめたいときは、Security Credentialsページにおいて[Multi-Factor Authentication(MFA)]欄に表示されている仮想MFAデバイスについて"Deactivate"を行えば、その仮想MFAデバイスが無効になり、通常のサインインができるようになります(前掲**図10**参照)。MFAを無効化したアカウントについては、Google Authenticatorからも忘れずに削除しておきましょう。ただし、MFAを無効化するよりも先にGoogle Authenticatorからそのアカウント情報を削除してしまうとサインインができなくなってしまうのでご注意ください。

万が一、MFAデバイスが故障したり紛失した場合は、公式ドキュメント[注4]に従ってMFAデバイスの無効化を行ってください。

次回は

次回からは、いよいよ実際にAWS APIをシェルスクリプトで利用する方法の解説に入っていきます。SD

注4) URL http://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/LostMFA.html

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2014年7月号

第1特集
［多機能］［高速処理］［高負荷対策］
そろそろNginxを考えているあなたへ

第2特集
知っているようで知らない
DHCPサーバの教科書

一般記事
・OpenSSLの脆弱性"Heartbleed"の教訓（前編）
・Webアプリのパフォーマンス改善（最終回）ほか
定価（本体1,220円+税）

2014年6月号

第1特集
設定ファイルの読み方・書き方でわかる
Linuxのしくみ

第2特集
Windows XPからの乗り換えにいかが？
Ubuntu 14.04 "Trusty Tahr"過酷な環境でも信頼できるLTSバージョン

一般記事
・Google Glassアプリ開発事情
・OpenTSDB（前編）ほか
定価（本体1,220円+税）

2014年5月号

第1特集
ネットワーク・ビギナー向け基礎講座
「ポート」と「ソケット」がわかればTCP/IPネットワークがわかる！

第2特集
UNIX必須コマンドトレーニング
rmコマンドからcadaverまで基本を押さえる

一般記事
・Rettyのサービス拡大を支えた「たたき上げ」DevOps
・Webアプリのパフォーマンス改善 ほか
定価（本体1,220円+税）

2014年4月号

第1特集
<Java/JavaScript/PHP>言語別で考える
なぜMVCモデルは誤解されるのか？

第2特集
ネットワークエンジニア養成
ロードバランサの教科書

一般記事
・さらに踏み込む、Mac OS Xと仮想デスクトップ #2
・SIMのしくみ
定価（本体1,219円+税）

2014年3月号

第1特集
データベースの諸問題
RDBとNoSQLどちらを選びますか？

第2特集
ネットワークエンジニアのための
プロキシサーバの教科書

一般記事
・さらに踏み込む、Mac OS Xと仮想デスクトップ #1
定価（本体1,219円+税）

2014年2月号

第1特集
λ式からはじめませんか？
関数型プログラミング再入門

第2特集
目利きによるトレンド予測
2014年IT業界はどうなるのか？

一般記事
・会社組織を活性化するスパイス「コンパ」

定価（本体1,219円+税）

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN & ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	渋谷区	紀伊國屋書店　新宿南店	03-5361-3315	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	書泉ブックタワー	03-5296-0051	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	千代田区	丸善　丸の内本店	03-5288-8881		広島市中区	丸善　広島店	082-504-6210
	中央区	八重洲ブックセンター本店	03-3281-1811	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322
	渋谷区	MARUZEN & ジュンク堂書店　渋谷店	03-5456-2111				

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。

ハイパーバイザの作り方

ちゃんと理解する仮想化技術

第21回

bhyveにおける仮想シリアルポートの実装（その1）

Writer　浅田 拓也（あさだ たくや）　Twitter @syuu1228

はじめに

前回（6月号）の記事では、bhyveにおける仮想ディスクの実装について解説しました。今回は、現在bhyveにおける唯一のコンソールデバイスである、仮想シリアルポートの実装について解説していきます。

物理PCにおけるシリアルポートの仕様

シリアルポートの仮想化について触れる前に、物理PCにおけるシリアルポートの仕様について簡単におさらいしましょう。

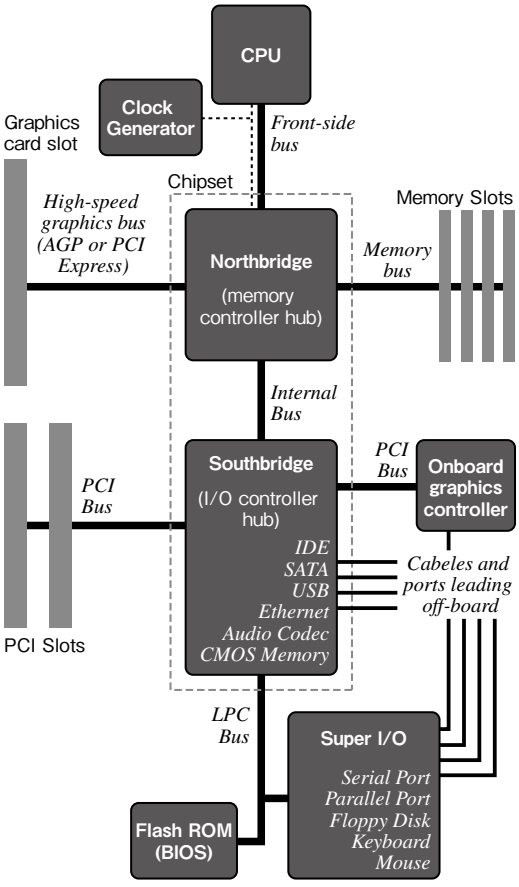
接続バス

ISAバス[注1]がまだ搭載されていた頃のPCでは、シリアルポートはフロッピーディスクコントローラやPS/2ポート、パラレルポートなどと共にISAバスに接続されていました。現在のPCでは古くなったISAバスは廃止されましたが、シリアルポートなどのレガシーデバイスは完全に廃止されておらず、代わりにLPCバス[注2]により接続されています。

LPCバスは低帯域なレガシーデバイスを接続するための専用バスで、I/O Controller Hub（ICH）[注3]に実装されています。LPCバスの物理的仕様はISAバスと互換性がありませんが、ソフトウェアからは従来のISAバスのように見えます。シリアルポートなどのレガシーデバイスはスーパーI/Oチップと呼ばれる単一のチップにまとめて実装されており、LPCバスの信号線からICHへ接続されています（**図1**）。

LPCバスはPCI-LPCブリッジを用いてPCIバスへ接続され、レガシーデバイスはLPCバスからPCI

▼図1　スーパーIOチップがLPCバスによってサウスブリッジに接続されていることを示すブロック図（CC-BY-SA 3.0 Moxfyre）



注1）ISA：Industrial Standard Architecture
注2）LPC：Low Pin Count
注3）略称ICH、サウスブリッジとも呼ばれる。

バスを経由してコンピュータと接続されます。

図2にlspciコマンドを用いてPCIデバイスリスト上にPCI-LPCブリッジが存在することを確認するコマンド例を示します。

このように、シリアルポートなどのレガシーデバイスは現在のPCにおいても互換性の維持のためにISAバスの時代と同様の方法でアクセスできるように設計されています。

割り込みとレジスタアクセス方法

ISAバス（実際にはLPCバス）ではPCIバスと異なりバス上のデバイスを検出する方法がありません。代わりに、デバイスごとに既定のIRQ番号、I/Oポート番号、メモリマップ先アドレスなどが決められており、ドライバは固定されたIRQ番号、I/Oポート番号などを使用します。シリアルポートに割り当てられているIRQ番号、I/Oポート番号を**表1**に示します。

シリアルポートはISAデバイスのままの仕様ですので、割り込みはPCIデバイスのようにレベルトリガではなくエッジトリガで行われます。また、割り込みはIO-APIC[注4]を経由しCPUへルーティングされます。ソフトウェアからみると、エッジトリガである点を除けば、MSI[注5]をサポートしないレガシーなPCIデバイスの割り込みに似ています。

16550A UART コントローラ

PCのシリアルポートには初期のPCから現在のPCまで16550A UART[注6]コントローラ互換のチップ[注7]が使用されています（現在のPCではUARTコントローラはスーパーI/Oチップ内に実装されています）。

16550A UARTコントローラは、より古いコントローラとの後方互換性を保つため1バイトずつの送受信を行うFIFO[注8]無効なモードと、数バイトのデータをバッファできるFIFO有効なモードを持ちます。

RS232C上のRTS/CTS、DSR/DTR各信号線を用いてFIFOのフロー制御ができます。これはハードウェアフロー制御と呼ばれており、これに対してソフト的なメッセージによってフロー制御を行う方式をソフトウェアフロー制御と呼びます。

表2に16550Aが持つハードウェアレジスタの一覧を示します。

各レジスタは使用するシリアルポートに割り当てられたI/Oポートの先頭アドレス＋offset（COM1のIERなら0x3F8＋1＝0x3F9）へin/out命令を行うことでアクセスできます。また、offset 0と1のレジスタはDLAB（LCRの7bit目）の値で切り替えて使用します。すべてのレジスタは1バイト幅になっています。

次に、**表3**から**表10**に各レジスタのフィールドと値の意味を示します。

OSからシリアルポートを初期化するには、**リスト1**のような手順で値をレジスタに書き込む必要があります[注9]。

OSからシリアルポートに届いたデータ1バイトを変数cへ受信するには、シリアルポートへ割り込みハンドラから**リスト2**のような手順でレジスタへのアクセスを行います。

注4） APIC：Advanced Programmable Interrupt Controller
注5） MSI：Message Signalled Interrupt
注6） UART：Universal Asynchronous Receiver Transmitter
注7） 厳密には初期のモデルのPCでは、これよりも古く機能の少ないコントローラが使用されており、現在では16550Aの上位互換なチップが使用されている。

注8） FIFO：First In, First Out：先入れ、先出し。
注9） リストではわかりやすくするためにビットフィールド単位で値を書き込んでいますが、実際にはレジスタ単位で読み書きを行います。

▼表1　シリアルポートのIRQとI/Oポート番号

	IRQ	I/O port
COM1	4	0x3F8-0x3FF
COM2	3	0x2F8-0x2FF

▼図2　PCI-LPCブリッジをlspciコマンドで確認

```
$ lspci | grep LPC
00:1f.0 ISA bridge: Intel Corporation 82801JIR (ICH10R) LPC Interface Controller
```

▼表2　16550Aのレジスター一覧

IO port offset	R/W	DLAB	名前	説明
0	R	0	RBR (Receive Buffer Register)	データの受信
	W	0	THR (Transmitter Holding Register)	データの送信
	RW	1	DLL (Divisor Latch LSB)	ボーレート(下位バイト)
1	RW	0	IER (Interrupt Enable Register)	割り込みの有効化・無効化
	RW	1	DLM (Divisor Latch MSB)	ボーレート(上位バイト)
2	R	-	IIR (Interrupt Indication Register)	現在の割り込み要因
	W	-	FCR (FIFO Control Register)	FIFOの設定
3	RW	-	LCR (Line Control Register)	通信方式(7bit目がDLAB)
4	RW	-	MCR (Modem Control Register)	制御信号の送信
5	R	-	LSR (Line Status Register)	通信状態
6	R	-	MSR (Modem Status Register)	制御信号の受信
7	RW	-	SCR (Scratch Register)	ソフトウェアが自由に使えるレジスタ

▼表3　ボーレートの設定(DLL、DLM)

bps	DLL	DLM
9600	0x0c	0x00
19200	0x06	0x00
38400	0x03	0x00
57600	0x02	0x00
115200	0x01	0x00

▼表4　有効化・無効化する割り込みの種類(IER)

bit	要因
0	データが着信(FIFO有効時はRX FIFOトリガ・タイムアウト)
1	データを送信可能
2	LSRの変更通知
3	MSRの変更通知

▼表5　現在の割り込み要因(IIR)

bit	要因
0	割り込みペンディング
1-3	000b: MSRの変更通知／MSR readでクリア 001b: データを送信可能(FIFO有効時はTX FIFOが空)／IIR read・THR writeでクリア 010b: データが着信(FIFO有効時はRX FIFOトリガ)／RBR readでクリア 011b: LSRの変更通知／LSR readでクリア 110b: RX FIFOタイムアウト／RBR readでクリア
6-7	00b: FIFO無効 11b: FIFO有効

▼表6　FIFOの設定(FCR)

bit	要因
0	FIFO有効化
1	RX FIFOのクリア
2	TX FIFOのクリア
3	0b: 1バイトごとに割り込み 1b: RX FIFOトリガ・RX FIFOタイムアウト・TX FIFOが空の時に割り込み
6-7	RX FIFOトリガ割り込みを行うバイト数 00b: 1byte 01b: 4byte 10b: 8byte 11b: 14byte

▼表7　通信方式の設定(LCR)

bit	要因
0-1	ワード長 10b: 7bit 11b: 8bit
2	0b: 1ストップビット 1b: 2ストップビット
3	パリティ有効
4	0b: 奇数パリティ 1b: 偶数パリティ
5	送信データのパリティ値を固定
6	ブレークシグナル有効
7	DLAB

▼表8　制御信号の送信(MCR)

bit	要因
0	Data Terminal Ready (DTR) をアサート
1	Request To Send (RTS) をアサート
3	割り込み有効

OSからシリアルポートへ変数cから1バイト送信するには、シリアルポートへ**リスト3**のような手段でレジスタへのアクセスを行います。

bhyveにおける仮想シリアルポートの実装

LinuxやFreeBSDなどのOSをシリアルコンソール上で動かすには、上述のレガシーなシリアルデバイスを用いる必要があります。

これらのOSでは、USB接続のUSB-RS232C変換ケーブルや準仮想化コンソールデバイスのような、非標準のコンソールデバイスを用いてブートメッセージを出力できないためです。

このため、bhyveではPCIデバイスの1つとして動くPCI-LPCブリッジのエミュレータとISAデバイス[注10]の1つとして動くシリアルデバイスのエミュレータを実装しています。また、シリアルデバイスのエミュレータは16550A UARTコントローラをエミュレートしています。

シリアルデバイスの入出力は、他のハイパーバイザと異なり単純に標準入出力に対して行われます。このため、仮想マシンをバックグラウンド動作させてコンソールをアタッチ・デタッチできるようにするには、tmuxのようなターミナルマルチプレクサを用いたり、nmdm(4)を用いてbhyveを実行している端末から標準入出力を切り離し、外部からアクセス可能にする必要があります。

まとめ

今回は物理PCにおけるシリアルポートの仕様を中心に解説しました。次回は、bhyveのPCI-LPCブリッジと仮想シリアルデバイスのソースコードを解説します。SD

注10) 前述のとおり、実際にはLPCバスに接続されたデバイスだがOSから見てISAデバイスと違いがない。

▼表9　通信状態（LSR）

bit	要因
0	データが着信
1	受信データ溢れ
2	パリティエラー
3	フレーミングエラー
4	ブレークシグナル受信
5	THRまたはTX FIFOが空
6	THRまたはTX FIFOが空で、かつ送信が完了している
7	RX FIFOにエラーのあるデータを受信

▼表10　制御信号の受信（MSR）

bit	要因
0	Clear To Send（CTS）が変化
1	Data Set Ready（DSR）が変化
2	Ring Indicator（RI）がネゲート
3	Data Carrier Detect（DCD）が変化
4	Clear To Send（CTS）がアサート
5	Data Set Ready（DSR）がアサート
6	Ring Indicator（RI）がアサート
7	Data Carrier Detect（DCD）がアサート

▼リスト1　シリアルポートの初期化手順

```
LCR[0:1] = 11b // ワード長 8bit
LCR[2] = 0 // 1 ストップビット
LCR[3] = 0 //パリティ無効
DLL = 0x01
DLM = 0x00 // 115,200 bps
FCR[0] = 1 // FIFO有効
FCR[1] = 1 // RX FIFOクリア
FCR[2] = 1 // TX FIFOクリア
FCR[3] = 1 // RX FIFO割り込み有効
FCR[6:7] = 11b // 14byteバッファしてから割り込み
MCR[0] = 1 // DTRをアサート
MCR[1] = 1 // RTSをアサート
MCR[3] = 1 // 割り込み有効
```

▼リスト2　シリアルポートの受信処理

```
if LSR[0] == 1 // データが着信している
  c = RBR
end
```

▼リスト3　シリアルポートへの送信処理

```
loop_until LSR[5] == 1 // TX FIFOに ➘
未送信データが残っている間、待つ
THR = c
```

第49回 Androidが生まれ変わる、活躍の場が広がる ～Google I/O 2014より

スマートフォン用のOSであったAndroidがいま変貌を遂げようとしています。この状況の変化を今年6月に行われたGoogle I/Oでの情報を含めながら、Androidの最新動向を紹介しましょう。

嶋 是一　SHIMA Yoshikazu
NPO日本Androidの会 理事長
Twitter @shimay

活躍の場が広がっている

2007年に発表され、翌年にオープンソースが公開されたAndroidも今年で7年目となりました。この間に全世界のスマートフォンの8割超に搭載され、事実上の標準プラットフォームとなるまでに普及しました。そのAndroidに、ここ一年新しい兆候が見え始めていました。それはスマートフォン以外の用途への活用です。それが、今年6月に米国サンフランシスコで行われたGoogleのイベント「Google I/O」にて、さらにその動きが公式な形で加速されました。

その1つがAndroidのクルマへの応用である「Android Auto[注1]」。そして、ウェアラブルデバイスへの応用である「Android Wear[注2]」、テレビへの応用である「Android TV」。そして、それらと連携することを前提としたスマートフォンAndroidである「Lリリース」です。スマートフォンのAndroidのときもそうでしたが、ワクワクするのは、今までプログラムできなかったところが自分たちの手でプログラムできるようになるところです。Androidが登場した当初、携帯電話の「待ち受け部分」も自分たちの手で自由にプログラムできると宣伝されていたように、携帯電話の機能を制限されることなく自由に開発することなど、従来の携帯電話（フィーチャーフォン、ガラケー）ではあり得なかったことです。この解放こそ、Android普及の原動力です。

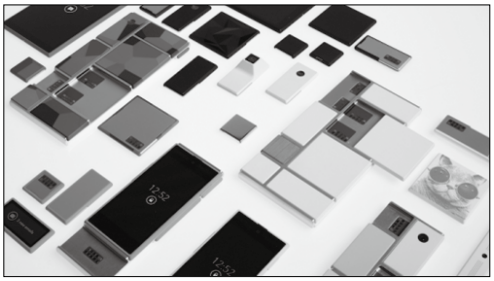
ここに来てAndroid Wearが発表されたことは、スマートウォッチなどのウェアラブルデバイスのプログラムを自分たちの手で行えることを意味し、開発環境の解放を意味します。これはAndroidがスマートウォッチの世界で普及する、きっかけとなるでしょう。同じく、クルマ関係のAndroid Autoも新しくワクワクする開発環境です。ただしこちらは機能の性格上、誰でも自由に開発できるようなオープンな運用はされないかもしれません。

このほかにも、Google I/Oでは発表されませんでしたが、興味深いAndroid関連の取り組みもあります。スマートフォンの機能をモジュール化し、自由に組み合わせて自分好みのスマートフォンを作り出す「Project Ara[注3]」（**写真1**）もAndroidを用いた新しい試みです。

さまざまなデバイスにAndroidが広がることは、Android開発者にとっては喜ばしいことです。スマートフォン以外のデバイスで、我々が慣れ親しんだAndroidと同じ方法でアプリケーション（以下、アプリ）が作成できるためです。ますます、Androidでプログラムを楽しむ私たちが活躍できる世界が加速して広がっています。

注1） http://developer.android.com/auto/index.html
注2） http://developer.android.com/wear/index.html
注3） http://www.projectara.com/

▼写真1　Project Ara



機能モジュールがブロック状になっており、それを組み合わせることでスマートフォンとして機能

Androidの躍進

冒頭にも書いたとおり、Androidは全世界の8割を超えるスマートフォンに搭載されているOSです。高機能スマートフォンが流通端末の中心である日本国内ではiOSのシェアが強いため、それほど高いシェアの実感はありません。しかし、海外の廉価版スマートフォンを含めるとAndroidのシェアは圧倒的となっています。

とくに今年のGoogle I/Oで報告された中で注目なのが、タブレットの世界でも6割を超えるシェアを獲得していること、そして1ヵ月で10億台のAndroid端末がアクティベートされている事実です。アクティベートとは、ユーザがAndroid端末の電源を入れ、Googleアカウントを入れて動かした数となります。昨年は5.4億台／月でしたので、2倍の増加率です。世界の人口は72億人ですから、その7分の1にあたります。もし自分の開発した1円アプリがすべての端末にダウンロードされるとしたら、1億円が儲かる計算です。もちろんあり得ませんが、これくらいの市場の可能性があります。

また、おもしろい数字として、人がどの程度Android端末を使っているかという点では、1日に200億のテキストメッセージが交換され、1日に自撮りカメラで9,300万枚の写真が撮られ、1日に1.5億歩が歩かれ、1日に1,000億回ロック解除して端末をのぞいています。このくらい、生活に密着した使われ方がされています。

▼写真2　Android L開発者向けプレビュー

▼表1　Androidバージョン

コードネーム	バージョン	SDKリリース日
	1.0	2008年9月23日
	1.1	2009年2月9日
Cupcake	1.5	2009年4月30日
Donut	1.6	2009年9月15日
Eclair	2.0, 2.1	2009年10月26日
Froyo	2.2	2010年5月21日
Gingerbread	2.3	2010年12月6日
Honeycomb	3.0, 3.1, 3.2	2011年2月22日
Ice Cream Sandwich	4.0	2011年10月18日
Jelly Bean	4.1, 4.2, 4.3	2012年6月27日
KitKat	4.4	2013年10月31日

Android Lリリース

Google I/Oで期待されたAndroidの新バージョンは、命名は決まらぬままその概要が発表され、現在はデベロッパプレビュー版が公開されています[注4](**写真2**)。Androidのバージョンは**表1**にあるように、お菓子の名前のコードネームがついています。このコードネームの先頭1文字はアルファベットの順番となっています。Cから始まって、最新の市場バージョンはKitKatの「K」です。そのため、今回は先頭に「L」が付与された名前になると期待されていましたが、Google I/Oではこのコードネームはお目見えされず「Lリリース」という名前で発表されました。その特徴的な機能を紹介します。

新しいUXによる操作の向上

WebにもAndroid端末にも共通する「マテリア

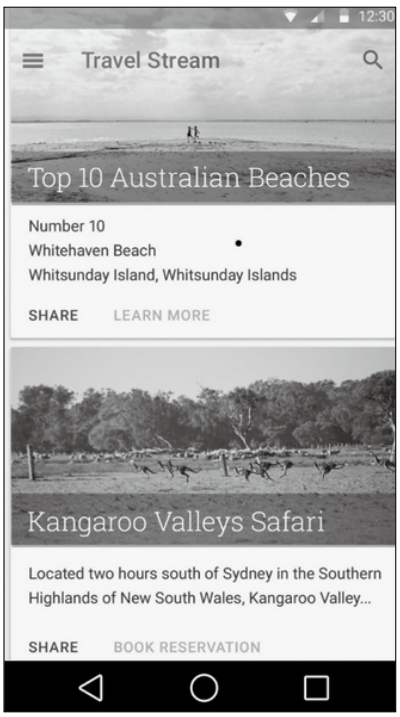
注4）http://developer.android.com/preview/index.html

ルデザイン」というデザインコンセプトが発表されました(**図1**)。

Androidのアプリでも、選択した途端に画面の全体が一気に「パッ」と切り替わってしまうと、利用者が面喰い、何が起こったのか認識できなくなることがあります。そのため、前の画面と次の画面の間で、画面の中の部品の関係の変化がわかるようにアニメーションを使って表現されます。

また、Google+でもよく用いられているようなカード型のUIベースとなり、カードを触ったときにはリップルという水の波紋が現れて、選択したことのフィードバックが表現されるようになっています。タッチパネルを触れても、正しく入力されたか、失敗されたかわからないことがよくあります。とくに端末の動作が遅いときに、触れたのが失敗したと思って二度目を触ると、最初に触れた動作が動いてしまい、二度目のタッチはそのあとの画面でタッチしたこととなり、誤操作を引き起こすことが多々あります。このあたりが改善されていることが期待されます。

▼図1　マテリアルデザイン



ART

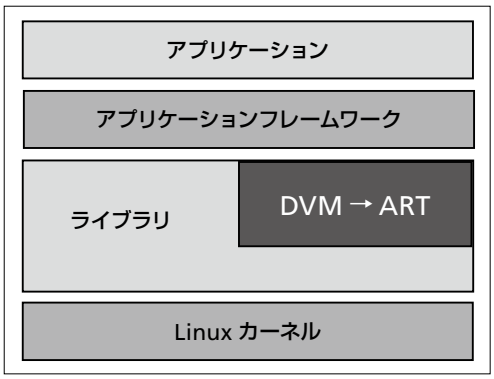
Androidは**図2**のように、Javaの実行環境としてDVM(Dalvik Vertual Machine)を利用していました。これはJIT(Just-In-Time)などを用いて高速化したJavaの仮想マシンです。Lリリースからはこれが変更となり、ARTが標準の仮想マシンとなります。

ARTはLLVMという機構を用いたインタープリタ型の仮想マシンで、高速化手法としてはAOT(Ahead-Of-Time)やJITも用いた仮想マシンです。実はバージョン4.4のKitKatからARTのしくみは搭載されており、設定により「DVM→ART」の変更が可能となっていました。デフォルトがDVMであるためにほとんど気にすることはありませんでしたが、今回はARTがメインとなります。また、64ビットを含めた、ARM、x86、MIPSの各CPUにも対応しており、クロスプラットフォームを実現しています。

端末連携

Lリリースは、スマートフォンだけでなく、車載端末、TV、スマートウォッチなどのディスプレイと連携するしくみが実装されています。とくにロック画面にも、カード型UIのノーティフィケーションが表示され、それらを操作することができるとともに、ほかのデバイスとも同

▼図2　Androidのアーキテクチャ図



▼▶写真3　Android One

期されます。

ロック解除機能としては、登録してあるBLE(Bluetooth Low Energy)機器(スマートウォッチなど)が近づくと、自動的にロックが解除されるような機能も搭載されています。これはBLEの周囲端末を探し出す機構を利用したものです。

バッテリー

Google内での「Project Volta」という取り組みにより、Lリリースに省電力のしくみが搭載されました。JobSchedulerというAPIが搭載されており、優先度が低い周期的なネットワーク通信などを遅延できるようにしています。たとえば、端末から無線通信する通信の回数を減らして省電力に貢献するようなしくみです。このスケジュールは通信に限らずに利用することができます。

Android One

Androidはこれまでソフトウェアプラットフォームでしたが、今回ハードウェアも含んだプラットフォームとして「Android One」が発表されました(**写真3**)。

iPhoneと同じような、ハードウェアとソフトウェアの一体化が実現できたという見方もできますが、本取り組みは新興市場に向けた廉価スマートフォンの取り組みとなります。価格も、100ドル程度でDualSIM、SDカード、4.5インチディスプレイ、FMラジオを搭載したスマートフォンが計画されています。

これまではハードウェアの部品についての規定はCDD[注5]の規定しかなかったので、実際には端末メーカーが数々の設計を行う必要がありました。その開発作業と検証費用により端末の開発費が高くなってしまい、安いAndroidスマートフォンの開発を難しくしていました。Android Oneによりこの費用が下がり、廉価端末の実現を可能としています。

Android Wear

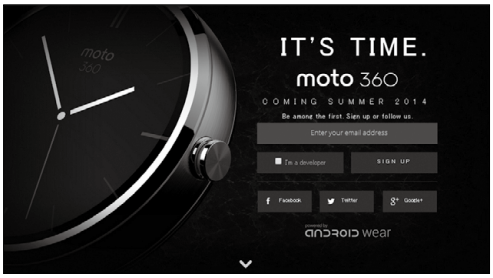
Google I/Oに合わせて、「Moto 360」(**図3**)「Gear Live」「LG G Watch」の3つのスマートウォッチの発表がありました[注6]。これらはAndroid Wearベースのスマートウォッチです。これまでもSDKは公開されていましたが[注7]、動かすためにはエミュレータの上でしか実行できず、試せる機能が限られており、実機の登場が待望されていました。このうちMoto 360については高級感ある仕上がりとなっており、話題を

注5) http://source.android.com/compatibility/

注6) 【Samsung Mobile Press】
http://www.samsungmobilepress.com/2014/06/26/Samsung-Expands-Gear-Portfolio-with-Android-Wear-1
【Samsung Mobile Press】
http://www.samsungmobilepress.com/2014/06/26/Gear-Live
【LG G Watch】
http://www.lg.com/global/gwatch/index.html
【Moto 360 by Motorola】
https://moto360.motorola.com/

注7) http://developer.android.com/wear/preview/start.html

▼図3　Moto 360のWebサイト



https://moto360.motorola.com/

呼んでいます。

動作としては、Androidスマートフォンに Wearアプリを入れておき、スマートフォンに届いた通知情報がWear端末に転送されて表示されるしくみとなります。

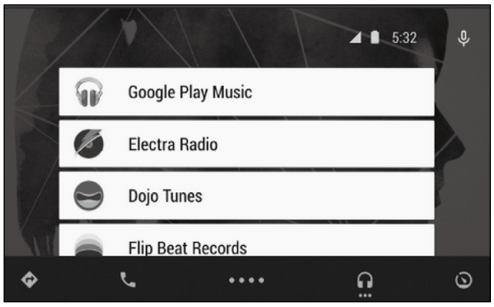
Android Auto

スマートフォン向けの現在のAndroidは、当初OHA(Open Handset Alliance)が設立され、ここからオープンソースのAndroidが公開されていました。これと同じような枠組みを行っているのが「Open Automotive Alliance」です。自動車にAndroidの搭載を進める団体であり、現在44社から構成されています。

Android Autoはスマートフォンやそのウェアラブルデバイスをクルマと接続することで、より良い車内IT環境を実現する取り組みです。AppleのCar Playと同じように、USBケーブルをクルマと接続して利用します。スマートフォンに搭載されているMediaアプリの画面を、クルマの中に入っているAutoアプリが受け取って、社内ディスプレイに表示を行います。

クルマの中で利用されるアプリであるため、Media UI(自動車に特化したUI。**図4**)、Notifications(スマートフォンの通知)、Voice Actions(音声コマンド)、Easy Development Workflow(開発の簡易さ)などの特徴があります。これらのアプリケーションを開発する「Android Auto SDK」が公開予定となっています。

▼図4　Android Autoのランチャーメニューの例



http://developer.android.com/auto/overview.html

Android TV

HDMIでテレビと接続するADT-1[注8]という開発キットが発表されました。今後Android TV[注9]対応のテレビが発売される予定です。STB用のAndroid OSであるAndroid TVを用いて、Androidのアプリをテレビのような大画面で動かすことができるようになります。

現在は開発キットがAndroid Lリリースのプレビュー版向けに提供されており、エミュレータで動作させることができるようになっています。しかしながらテレビである以上、放送を受けて番組を表示するためのチューナーやリモコンなどの、非スマートフォン関連の機能機構が必要となります。そのために今回はADT-1を発表しました。まさに、STB(セットトップボックス)のプラットフォームともいえます。"A Platform for the living room(リビングルームのプラットフォーム)"と知らされています。

Project Ara

Project AraはAndroidをソフトウェアプラットフォームとして流用した、モノづくりのためのプロジェクトです。

注8) https://developer.android.com/preview/tv/adt-1/index.html

注9) http://developer.android.com/tv/index.html

この成果物は、スマートフォンをレゴブロックのように自由に組み合わせることでスマートフォンを作り出すことができます。このデバイスをフレームに差し込み、磁石の力でスマートフォン本体と接点とを接続します。

しかしこのデバイスが出てくるのは、2015年の第一四半期となります。それまでは実機での確認はできません。その前にProjectAraの開発キットとして、「the Module Developers Kit (MDK)」がリリースされています。

そしてこのプロジェクトのポイントは「モノづくり」である点です。提供されているオープンソースのツールにて、回路設計から3Dプリンタで出力するための機構設計が行えるのです。単一のツールで、ハードウェアからソフトウェアまで統合されて開発できる環境を作り、開発の速度を上げる、これがこのプロジェクトの目的です。

まとめ

Androidをプラットフォームとして、さまざまなデバイスがサポートされるようになりました。スマートフォンの普及とは別に、新しいデバイスの領域にAndroidが生まれ変わろうとしています。また、これらを横断して、ノーティフィケーションの通知や、さまざまな連携機能がシームレスに行われる環境が提供されています。Androidのアプリ開発者としては、いろんな機器が連携したアプリやサービスが作れることとなり、Androidは、ますます夢を広げてくれるプラットフォームとなっています。SD

嶋 是一（しま よしかず） NPO日本Androidの会 理事長

日本Androidの会でAndroidに関する活動を行う傍ら、株式会社KDDIテクノロジーに所属し、モバイル関連の技術開発を行う。日本のモバイル関連技術の普及活動を継続して行っている。

Red Hat Enterprise Linuxを
極める・使いこなすヒント

.SPECs

ドット・スペックス

第4回 Red Hat Enterprise Linux 7に触れてみよう

前回はRed Hat Enterprise Linuxの開発フローについて理解し、サーバを可能な限り安定的に運用する方法について説明しました。今回と次回の2回にわたり、待望の新バージョン・RHEL7の新機能や「お試し方法」を中心に紹介します。

レッドハット(株)グローバルサービス本部プラットフォームソリューション統括部
ソリューションアーキテクト部長　藤田 稜 (ふじた りょう)

RHEL7の新機能・変更点

米国時間2014年6月10日にRed Hat Enterprise Linux 7(以降、RHEL7)がリリースされました[注1]。前バージョン6のリリースは2010年10月18日でしたので、1,331日・約3年8ヵ月ぶりの新バージョンということになります。kernelは3.10を採用し、システムに搭載できるメモリ容量の増大に対応するためサポートするCPUは64ビットのみ、x86_64/ppc64/s390xの3アーキテクチャ[注2]をサポートします。

RHEL 7のファイルシステム

デフォルトのファイルシステムは従来のext4に代わり、xfsが採用され最大で500TBをサポートする一方で、RHEL6では最大で16TBをサポートしていたext4はサポートの上限が50TBに設定されました。btrfsも利用できますが、Technology Previewという位置づけのためプロダクションシステムでの利用は推奨されません。

RHEL 7のインストールとサブスクリプション管理

RHELのインストーラであるanacondaはワークフローが大きく変更されモジュール&ハブ形式のデザインになりました。従来は対話型のウィザード形式でしたが、複数の項目をデフォルトのパラメータのままインストールする場合には途中のステップが煩わしく感じることも多く、今回の改良によりデフォルト以外の項目だけを選択して変更することが可能になっています(**図1**)。

インストール作業そのものが面倒だという人向きにはPXEブート[注3]によるkickstartインストール[注4]をお勧めします。さらにインストール作業をせずに利用したいということであれば、

注1) 同日付でopensslやkernelなどのエラータがリリースされており適用が強く推奨される。

注2) x86_64は米Intel社や米AMD社のいわゆる"Long Mode"をサポートするCPU、ppc64は米IBM社のCPU・POWER 7/7+/8、s390xは米IBM社のsystem zのCPUを表す「CPUアーキテクチャ・コード」。過去にはia64(Itanium/Itanium2)、ia32e(64ビットをサポートする初期の米AMD社のCPU)などのコードも存在した。

注3) RHEL7ではPXEの設定ファイル(/var/lib/tftpboot/linux-install/pxelinux.cfg/defaultなど)の"append"に"inst.repo"を追加する必要がある。たとえば、inst.repo=http://example.com/rhel7dvd/など。追加しない場合、Dracutによる"Warning: /dev/root does not exist"というエラーメッセージが表示されインストーラが停止する。インストール方法の詳細については、Red Hatのカスタマーポータル(https://access.redhat.com/)の"Installation Guide"を参照(サブスクリプションは不要)。

注4) kickstartインストールはLinuxの自動インストールのしくみ。kickstart設定ファイルを用意し、起動ディスクやWebサーバなどから取得することで自動的にLinuxをインストールすることができる。テスト環境などの構築にはプライベートクラウドや仮想化とともにテンプレートを用いるのが一般的になりつつあるが、kickstartでは最新のリポジトリに追随したクリーンインストールが短時間でできる特徴がある。

Red HatがKVMのゲスト用のイメージを配布(**図2**)していますし、米・Amazon社のAWS(Amazon Web Service)に用意されたRHEL7のAMI[注5]という選択肢もあります。

RHEL7ではサブスクリプションの管理方法として利用されてきたrhn_register(RHN Classicと呼ばれます)は廃止され[注6]、RHEL5.7/6.1から利用可能(RHN Classicと並行する形で)となっていた、Subscription Manager[注7]だけが提供されるようになりました。システムへのサポート権の付与(エンタイトルメントと呼ばれます)が完了したあとは、従来と同じyumコマンドによるパッケージ管理ができます。

RHEL7のGUI環境

デフォルトのGUI環境はGNOME 3.8の"Classic"となり(**図3**)、一足早くFedoraで利用可能となっていたGNOME Shell(**図4**)はログイン時(GNOMEセッションの開始時)に選択可能なオプションとなりました。一見するとわかるようにGNOME Shellは先進的なUI(User Interface)である一方で、おもに企業における利用が指向されるRHELにおいては敬遠される程度に大きな変更であるため、GNOME 3.8を採用しつつも従来のGUI環境と違和感なく利用できる"Classic"がデフォルトとして用意されたという経緯があります。

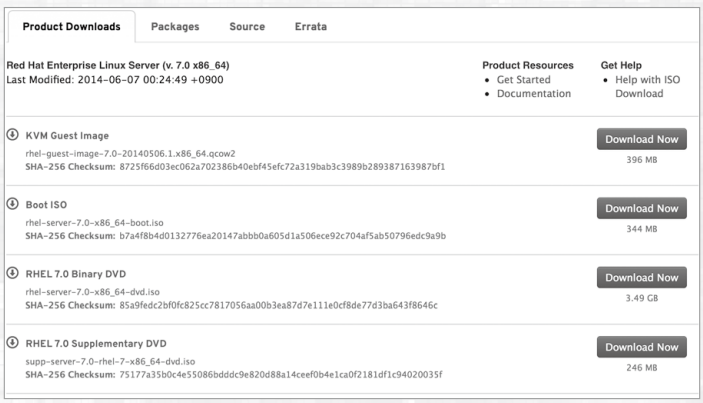
RHEL7のブートローダー

従来のGRUB Legacy(0.9x系)に代わり、GRUB 2が採用されています。これに伴い

▼図1 モジュール&ハブ形式のインストーラ

(ウィザード形式で順番に設定する従来方式と異なり、必要な個所だけ変更すればインストールを開始できる)

▼図2 Red Hatが配布するRHEL 7のイメージファイル



("KVM Guest Image"はRHELやRHEV(Red Hat Enterprise Virtualization)、RHEL-OSP(OpenStack)のNova Compute Nodeで利用可能)

注5) AWSのMarketplaceから利用可能(https://aws.amazon.com/marketplace/pp/B00KWBZVK6)。
注6) Red Hat Satellite(旧称・Red Hat Network Satellite)を別途購入している場合にはrhn_registerが利用可能。
注7) Subscription Managerの利用方法については、https://access.redhat.com/site/ja/node/69764

BIOSだけではなくUEFI(Unified Extensible Firmware Interface)への完全対応が含まれており、GRUB LegacyでUEFI採用システムへのインストール・起動時に発生していたトラブルが減少することが期待できます。一方で設定ファイル(/boot/grub2/grub.cfg)の書式が変更されており、エディタによる直接の編集は禁止されました。設定方法としてはgrub2-mkconfigコマンドが提供されます。

systemd, systemd, systemd!

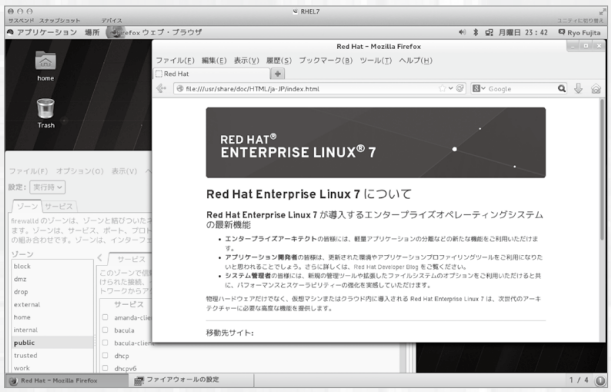
本誌でも数回にわたって紹介されていますが、従来のSysV Initに代わり、systemdがユーザスペースのさまざまな管理をすることになりました。RHEL7の最大の変更点はsystemdの採用であり、誤解を恐れずに言うならばLinuxの20年の歴史においても「RHEL7はまったく別のOS」といっても過言ではありません。

systemdの管理ツールとしてsystemctl、systemadmを用いることとなり、chkconfig、service、init、runlevelといった非常に多くのコマンドが廃止されていますが、これらのコマンドを実行するとsystemctlコマンドに「転送」することで従来との互換性を保っています。この互換性は将来的に廃止されていく可能性が高いため、RHEL7ではsystemctl、systemadmに習熟することをお勧めします。

ネットワーク関連の強化ポイント

systemdに次いで大きな変更の1つがNetworkManagerへの全面移行でしょう[注8]。長らく利用されてきたsystem-config-networkやifconfigなどのツールを含むnet-toolsパッケージはデフォルトではインストールされず、InfinibandやBonding、Teaming、VLANなどを含むネットワークの設定のいっさいは、nmcliコマンド、nmtui-edit、nm-connection-editorを利用して行います。パッケージグループとして最小インストールを選択した場合[注9]、従来どおりipコマンドを含むip-routeパッケージがインストールされているので、IPネットワークの設定や表示などについてはipコマンドを

▼図3　GNOME Classic



(デスクトップにゴミ箱等のアイコンがあり、ユーザのホームフォルダ以下にある[デスクトップ]フォルダ内のファイル等もデスクトップに表示される従来どおりのインターフェース)

▼図4　GNOME Shell

(デスクトップにゴミ箱などのアイコンがなく、操作の起点は左上の[アクティビティ]メニュー、あるいは【Windows】キーとなる。ユーザのホームフォルダ以下にある「デスクトップ」フォルダ内のファイルなどはデスクトップに表示されない)

注8) NetworkManagerそのものはRHEL4から提供されていた。
注9) x86_64用のRHEL7.0を最小インストールした場合のRPMパッケージは言語設定が英語の場合324、日本語の場合325。

利用することもできます。

NIC(Network Interface Card)のデフォルトの命名規則はPredictable Network Interface Namesと呼ばれるポリシーに変更されました。マザーボード上なのかPCIバス上なのか、何番目のポートなのかといったファームウェアが認識するNICの物理的な「位置」に基づくという点ではRHEL6で導入されたbiosdevnameと似ている[注10]のですが、udevのルールによって命名される点ではより「筋の良い」実装[注11]に変更されたと言えます。命名規則は次の順に適用されます。

①オンボード(例：eno1、"On-board")
②PCI Express(例：ens1、"Slot")
③NICのコネクタの物理的な位置
(例：enp2s0、"Physical"、"Slot")
④MACアドレス(例：enx78e7d1ea46da)
⑤従来どおりのカーネルによる命名方法
(例：eth1)

このため、たとえばオンボードのNICの名前は、"eth1"(従来の命名規則)→"em1"(biosdevname)→"eno1"(udev)と変わっているので注意が必要です。

ホスト名の変更についてもRHEL7では新しいツールが用意されました。従来は/etc/sysconfig/networkや/etc/hostsなどホスト名を設定するファイルが複数あり、これがトラブルの原因となることも少なくなかったのですが、RHEL7では設定ファイルは/etc/hostnameに統一されたうえ、hostnamectlコマンドを用いることになりました。

ファイアウォールは従来のiptablesからfirewalldがデフォルトになりました。firewalldは「動的ファイアウォール」と呼ばれる実装で、既存のネットワークコネクションを維持したままファイアウォールのポリシーを変更できます。管理にはfirewall-cmdコマンドやGUIツールのfirewall-configを利用します。

ご注文はdockerですか?

RHEL7に同梱されるのか、またサポート対象に含まれるのかが直前まで決まらなかったLinuxのコンテナ実装の1つであるdockerは、最終的に「ミッションクリティカル環境では用いないことが推奨される」という制限付き[注12]ながら、RHEL7に同梱されました。ただしRHEL7のインストーラDVDには含まれておらず、"Extras"というチャネル[注13]を追加登録すれば利用ができます。

次回は

まだまだRHEL7の新機能については紹介したいことが山ほどあるのですが、紙幅が尽きてしまいました。次回はRHEL7を試用する方法や、注意点について紹介する予定です。SD

注10) RHEL6では米・Dell社製サーバにおいてbiosdevnameによる命名がデフォルトで、マザーボード上のNICであれば"em1"(Embeddedの意)、PCIバス上のNICのポート1であれば"p1p1"といった名前になる。より正確にはbiosdevnameの命名ポリシーとして"physical"を指定しているために物理的な位置に基づく命名が行われる。biosdevnameを利用せずに従来どおりの命名規則を用いることも可能で、インストール時あるいは起動時のkernelパラメータ、もしくはgrub2のテンプレートファイルである/etc/default/grubの"GRUB_CMDLINE_LINUX"に、"biosdevname=0"を追加する。詳しくはman biosdevname。

注11) デバイスの命名規則全般を扱うべきudevとは別に、biosdevnameというツールが用意されるのは「筋が悪い」という意味。

注12) RHEL 7のリリースノート、"1. New Features"中の"7. Linux Containers with Docker Format"に次の記載がある。Docker is still in development and has not yet reached version 1.0. For this reason it is not recommended to use Docker in mission-critical production environments.

注13) RHELのRPMパッケージは「チャネル」と呼ばれるグループに分類されている。インストーラDVDに入っているのは「Baseチャネル」であり、開発に必要な*-develパッケージなどが含まれる「Optionalチャネル」、米・IBM社のJavaランタイムなどが含まれる「Supplementaryチャネル」などがあり、Subscription Managerでシステムにこれらのチャネルを追加することで利用可能となる。

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第10回 ◇ コンパイラ～GCCからLLVM Clangへ

◉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。

GCCからLLVM Clangへ、ベースシステムからGCCを排除

FreeBSD 10.0-RELEASEからはデフォルトのコンパイラがLLVM Clangへ変更されています。これまでFreeBSDはGCCをシステムのデフォルトコンパイラとして活用してきました。しかしこの数年をかけてGCCの依存状態から脱却し、LLVM Clangをデフォルトのコンパイラとする取り組みを進めてきました。現在のところamd64版とi386版が対応していますが、今後はほかのアーキテクチャもLLVM Clang化が進められる見通しです。

FreeBSDがデフォルトコンパイラをGCCからほかのコンパイラへ変更したのにはいくつかの理由があります。とくに重要な理由をまとめると次のようになります。

- FreeBSDベンダからの要望もあり、FreeBSDプロジェクトはGPLv3のソースコードをベースシステムに取り込まないとしている。GCCはGCC 4.2.1よりも後のバージョンでGPLv2からGPLv3へ移行したため、FreeBSDプロジェクトは古いGCC 4.2.1を使い続けるしかなかった。この状態を改善するために、ほかのコンパイラへ移行する必要があった

- LLVM Clangは後発のコンパイラインフラストラクチャだけあってよく設計されており、コンパイル時間が短く、生成されるバイナリも性能がよいという特徴がある。さらにBSDライセンスのもとで提供されているためFreeBSDへマージしやすい

- FreeBSDプロジェクトとLLVM Clangプロジェクトは関係性もよく、デフォルトコンパイラをLLVM Clangへ変更することは有益

10.0-RELEASEでは、amd64版とi386版でもブートコードまわりなどいくつかの点はLLVM Clangではカバーすることができずに、従来のツールチェーンが使われています。今後のリリースでツールチェーンのすべてをLLVMプロジェクトの提供するものへ置き換えていく予定です。

コンパイル時間大幅短縮 LLVM Clang

LLVM ClangとGCCのコンパイル時間をFirefoxとThunderbirdで比較すると、**表1、2、図1、2**のようになります。LLVM Clangのほうが1.7倍ほど早くビルドが完了していることがわかります。Ports Collectionからパッケージをビルドする作業などに直接影響を与える数値です。LLVM Clangへの移行は作業時間の短縮化という点でもプロジェクトにとって必要な作業でした。

カーネルおよびユーザランドのビルドは現在のところ簡単に切り替えられるようにはなっていません。LLVM Clang以外のコンパイラへの切り替えが

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。

簡単にできるようにする取り組みは、現在整理が進められている段階にあります。切り替えが簡単にできるようになったら、ビルド時間の比較結果などを紹介したいと思います。

エラーメッセージのわかりやすさ

LLVM Clangを開発に採用しているデベロッパであればすでに体感していることだと思いますが、LLVM Clangはコンパイルエラー発生時に出力してくれるエラーメッセージがとてもわかりやすいという特徴を持っています。これは開発時間にもデベロッパの精神的な負担にもかかわってくる大切なポイントです。

たとえば、**図3**のような**echo**(1)コマンドのビルドを考えます。問題がなければ**図3**のようにコンパイルが実行され、バイナリファイルとマニュアルファイルが生成されます。

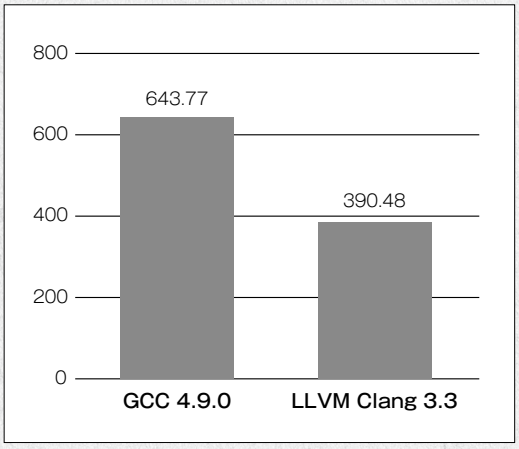
次に、ソースコードのどこかに記述誤りがあり、GCCがそれを検出したときのエラーメッセージを**図4**に示します。

「echo.c:114:5: error: request for member 'iov_base' in something not a structure or union」がエラーメッセージです。vp.iov_base = *argv;のように構造体のメンバに値を入れようとしていますが、この記述に問題があるようです。vp.iov_baseの「.」に矢印「^」がついていますので、この部分に問題があるように見えます。

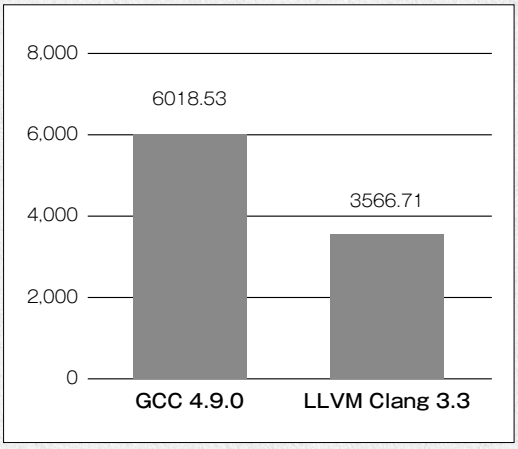
メッセージの内容はこのようになっています。「echo.cファイルの114行目、iov_baseメンバへアクセスしようとしていますが、構造体にもユニオンにもそのようなメンバは存在しません」。このエラーメッセージからはまず、たぶん「iov_base」という名前をタイポ(誤入力)しており、別の名前なんだろう、ということが予測できそうです。

同じソースコードをLLVM Clangでビルドする

▼**図1　Firefoxコンパイル時間比較(GCC 4.9.0 vs. LLVM Clang 3.3)**　※短いほど高速



▼**図2　Thunderbirdコンパイル時間比較(GCC 4.9.0 vs. LLVM Clang 3.3)**　※短いほど高速



▼**図3　echo(1)コマンドのビルド**

```
% ls
Makefile echo.1   echo.c
% make
Warning: Object directory not changed from original /Users/daichi/tmp/echo
cc -O2 -pipe  -fno-omit-frame-pointer -std=gnu99 -Qunused-arguments -fstack-protector  -c echo.c
cc -O2 -pipe  -fno-omit-frame-pointer -std=gnu99 -Qunused-arguments -fstack-protector   -o echo echo.o
gzip -cn echo.1 > echo.1.gz
%
```

と**図5**のようになります。

「echo.c:114:5: error: member reference type 'struct iovec *' is a pointer; maybe you meant to use '->'?」がLLVM Clangの出力したエラーメッセージです。こんなことが書いてあります。「echo.cファイルの114行目ですが、"struct iovec *"はポインタですので、(「.」ではなく)「->」ではありませんか？」

実際、変数vpは次のようにiovec構造体へのポインタとして宣言されています。

```
struct iovec *iov, *vp;
```

処理の途中で**リスト1**のようにメモリを確保して使われています。ですので、「vp.iov_base」ではなく「vp->iov_base」または「(*vp).iov_base」と書く必要があります。LLVM Clangのエラーメッセージはこのように丁寧な出力になっていることが多く、デベロッパの負担を減らしてくれるという特徴があります。ヘッダファイルの指定がなければ「このヘッダファイルの指定を忘れていませんか？」といったメッセージを出力してくれるなど便利です。

FreeBSD 11へ向けたIntelコンパイラへの対応

現在FreeBSDプロジェクトでは、コンパイラとしてIntel製のコンパイラを利用する方向についても模索しています。2014年5月にカナダの首都オタワで開催された開発者会議では、基本的にIntelの

▼図4　ビルドエラー：GCCの出力

```
% make CC=gcc49
Warning: Object directory not changed from original /Users/daichi/tmp/echo
gcc49 -O2 -pipe  -fno-omit-frame-pointer -std=gnu99 -fstack-protector  -c echo.c
echo.c: In function 'main':
echo.c:114:5: error: request for member 'iov_base' in something not a structure or union
   vp.iov_base = *argv;
     ^
*** Error code 1

Stop.
make: stopped in /Users/daichi/tmp/echo
%
```

▼図5　ビルドエラー：LLVM Clangの出力

```
% make
Warning: Object directory not changed from original /Users/daichi/tmp/echo
cc -O2 -pipe  -fno-omit-frame-pointer -std=gnu99 -Qunused-arguments -fstack-protector  -c echo.c
echo.c:114:5: error: member reference type 'struct iovec *' is a pointer; maybe you meant to use '->'?
                vp.iov_base = *argv;
                ~~^
                  ->
1 error generated.
*** Error code 1

Stop.
make: stopped in /Users/daichi/tmp/echo
%
```

▼リスト1　途中でメモリを確保して利用

```
if ((vp = iov = malloc((veclen + 1) * sizeof(struct iovec))) == NULL)
```

担当者の了解は取り付けてあるといった説明がありました。あとは開発を担当するコミッタがつけば、Intelコンパイラを使ったカーネルおよびユーザランドの構築ができるようになる可能性があります。

Intelコンパイラへ対応させる場合、現在よりも積極的にコンパイラの動的切り替えができるようにMakefile一式を整理する必要がありますが、これが完成すると適材適所で利用するコンパイラを切り替えることができるようになり、FreeBSDを適用できる幅が広がります。Intel x86系以外のアーキテクチャへの対応はGCCが優れた状況にありますので、LLVM Clang、GCC、Intelのコンパイラをそれぞれ切り替えてシステムを構築できることは、オペレーティングシステムを稼働させるマシンやアーキテクチャの種類の増加につながります。

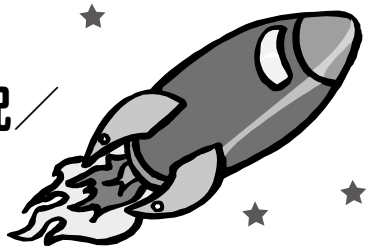
C/C++言語勉強のプラットフォームとして

FreeBSD 10.0-RELEASE以降はLLVM Clangがデフォルトのコンパイラとして同梱されていますので、C言語やC++を勉強しようという場合には手軽に環境を用意できるプラットフォームです。カーネルやコマンドのソースコードも含まれていますし、勉強用の教材として利用するには扱いやすい成果物です。一度このコンパイラで遊んでみてはいかがでしょうか :) SD

17

Debian Developer　やまねひでき　henrich@debian.org

Debian GNU/Hurdの状況／Squeeze LTSの使い方



Debian Hot Topics

開発の進捗どうですか

Hurdは順調

先月号ではsparcの進捗が思わしくない、という話をしたので逆に好調なDebian GNU/Hurdについて取り上げます。昨年にDebian GNU/Hurd 2013がリリースされ、その後も順調に進捗が進みパッケージのカバー率が80%に達しました(これは2012年と比べると10%の向上となります)。また、カバーされているパッケージのバージョンも98%が最新バージョンに追随するなど、良好な状況のようです。

この背景にはパッケージ側の修正もさることながら、ビルドマシンが安定して稼働するようになったことも大きく寄与しています。この調子でいけば、あと2、3年でほぼすべてのパッケージがHurdでもインストールできるようになるはずです(動くかどうかはまた別問題ではありますが)。うまく動作するようになった例としてIceweasel 29が挙げられており、libc側も修正を加えたとのこと。デスクトップ分野でHurdを使う人がどの程度いるのかは疑問ですが、何はともあれめでたいですね。

Initシステムについても、自前でこしらえていたシェルスクリプトから、SysVinitに移行してさまざまな動作が改善されました。これにより、halt/shutdownコマンドが動作するようになり、ようやく各種サービスが正常に終了するようになったとのことです(筆者はパッケージにおけるHurdでの問題を修正する際、QEMUで動作確認したあとで無理やり終了するしかなくて何度かイメージを壊していました……)。

Hurdが「Jessie」のリリースターゲットに含まれるかどうかは微妙ですが、いい感じで進んでいるようですので興味のある方はこれからもウォッチしてみてください。

試してみたいと思った方は、初期設定済みのQEMUイメージが用意されていますので[注1]、そちらをご利用ください。

eglibcからglibcに出戻り

これまで、Debianで使われている基本ライブラリ(libc)はglibcではなく、その派生のeglibc[注2]が使われていました。これは、glibcのメンテナであったRed Hat社のUlrich Dreppersさんが、エキセントリックな言動とともに、ARMなどのサポートに対して拒否的な姿勢を示していたことに起因します。当時のARMは今よりずっとマイナーなアーキテクチャだったからでしょう。そこで有志が「組込み向けのアーキテクチャにフレンドリーなglibcを作ろう」とeglibcを立ち上げ、多アーキテクチャサポートを進めるDebianが採用したのです。

しかしその後、状況は変化します。リポジトリがGitに移ったことで、開発体制がUlrich

注1) URL http://people.debian.org/~sthibault/hurd-i386/
注2) eglibc＝Embedded GLIBC

Dreppersさんの中央集権的状態から分散的なものになったのに加えて、UlrichさんがRed Hat社を離れたことにより、glibcはより自由度の高い開発が行われることになりました。

象徴的なのが、2012年にglibcの開発について中央委員会が解散し、開発者らが門戸が広くなっているのを訴えかけたことです。結果、eglibcの活動は停滞し、今回、Debianでは「glibcへマージして注力したほうが良い」との判断が下されました。なお、Debianとしてはすでにsid(unstable)ではglibcに移行していますが、実質的には「あまり変化はない」状態で、これといって問題や挙動の変化などは出ていません。今後は淡々と残っているパッチがマージされていくものと思われます。

Squeeze LTS

Debian 6"Squeeze"に対するLTS(Long Term Support、長期間サポート)の案内が出ました。とはいえ、当の作業の中心であるFreexian社のRaphaël Hertzogさんは自身のブログ[注3]で次のように言及しています。

「LTSが開始されたことはメディアで報じられているが、残念なことに作業の手助けが必要なことにはまったく触れられていない。まだLTSは満足のいく作業量をこなしているとは言えない状況だ」。

続けて「いつものように必要な作業をするのに協力者が足りていないのだが、今回は特例で簡単な方法がある。必要な作業をする人に支払いをすれば良い」と述べ、この作業を行うための資金として、Debianを利用している数千の企業のうち、サポートを購入する企業があと50社ほど必要であることが示唆されています。

- サポートのサブスクリプション費用は、255～24,480ユーロ(約3.5万～340万円)／年(台数は関係なし)
- サポートは支払額によって次のような権利が付与される
  - -支援企業として会社のロゴを掲載
  - -プライベートなメーリングリストへの参加
  - -LTS作業者への直接のコンタクト
  - -各企業のテストケース検証

日本のDebian利用企業も、Squeezeを使い続けたい場合は、上記サポート[注4]の購入を検討していただければと思います(専任の人を雇うよりはるかにお安いですし、宣伝にもなりますよ)。

LTSをシステムに適用!

背景はこの程度にして、LTSを実際に適用する最小限の手順を説明します。Debian 6はこの手順を踏まないとLTSにはならないので、ご注意ください(勝手にLTSになるわけではありません。設定を追加する必要があります)。

①apt line(/etc/apt/sources.list、あるいは/etc/apt/sources.list.d/*.list)にSqueeze LTSのリポジトリを追加
②apt-get updateしてリポジトリの追加を反映
③apt-get upgradeしてパッケージの更新を反映
④debian-security-supportというチェック用のツールをインストールし、システム内のサポート対象外パッケージを確認する
⑤チェックの結果、必要に応じて自前で更新するなどの対応を検討する

実際のコマンドは**図1**のようになります。

注3) URL http://goo.gl/9hFhr1
注4) 詳細はURL http://www.freexian.com/services/debian-lts.htmlを参照。

▼図1　Squeeze LTSの適用手順

```
$ sudo sh -c "echo deb http://ftp.jp.debian.org/debian/ squeeze-lts
main contrib non-free >> /etc/apt/sources.list"        ←①
$ sudo apt-get update; sudo apt-get upgrade            ←②、③
$ sudo apt-get install debian-security-support         ←④
(図2の画面が表示される)
$ check-support-status      ←再度チェックをしたい場合に実行
```

④のインストール時に、現在のシステムに入っているサポート対象外パッケージが表示されます。**図2**はその一例で、webkit関連パッケージが対象外となっていることが表示されています。

check-support-statusによって判別されるサポート外パッケージについて、自前でのセキュリティ更新対応が難しそうな場合は、速やかにDebian 7“Wheezy”へのアップグレードを実施するなどして対応することを検討しましょう。

ビルドに必要なファイルを探す術

おもしろそうなソフトウェアを見つけ、ソースを持ってきてビルドをした[注5]が、「ファイルが足りなくてエラー！」という経験をした方は少なくないと思います。このような場合、どうやって対処すれば良いのでしょう？　Debianで同じソフトウェアのパッケージがある場合は非常に簡単で、apt-getコマンドのオプションを使えば解決できます。

```
$ sudo apt-get build-dep <package>
```

これだけで、パッケージの「Build-Depends」に定義されている、このパッケージのビルドに必要となるパッケージ群を一気に丸ごと持ってきてくれます[注6]。あとは適当にconfigureやmakeなどのビルドのコマンドを実施すれば解決です。

しかし、同じソフトウェアのパッケージがない場合は、この手は使えません。configureで行き詰まった場合、出力されたconfig.logなどのエラーの内容を見ると足りないファイルの名前はわかります。しかし、どうやって必要なファイルを持ってくればいいのでしょう？　Debianは豊富なパッケージ群を持っていますので、何かのパッケージを入れれば解決！……のはずですが、「さて、どれを入れていいのか」と見当がつかず途方に暮れる方もいるでしょう。これについては2つの方法があります。

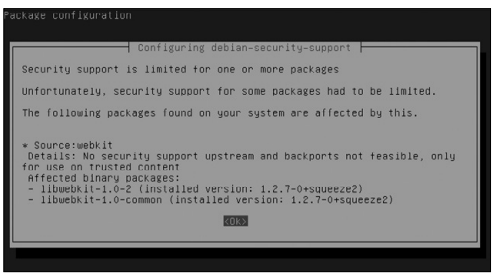
(1) Debianのサイトで検索をする
(2) apt-fileユーティリティを使う

まず、Debianのサイトに「パッケージ」のページが用意されています[注7]。お世辞にも見やすいとは言えませんが、このページの一番下のほうに「パッケージの内容を検索」という項目があり、「キーワード」にファイル名を入れて適当なオプションを選択すればそのファイルが含まれているパッケージ一覧が表示されます。

しかし、「サイトから検索すると、大量にヒットし過ぎて見づらい」「適当に絞り込みたい」「何回も繰り返して検索する場合に、いちいちブラウザを開くのは面倒」という要望が出てきます。この場合、apt-fileユーティリティがあなたの

注5）典型例は「./configure; make」などですね。

▼図2　debian-security-supportの画面



▼図3　apt-fileの使い方

```
$ sudo apt-get install apt-file      ←apt-fileをインストール
$ apt-file update      ←apt-fileのデータベースを最新化
$ apt-file search <file>      ←ファイルを検索
$ apt-file search <file> | grep foobar      ←検索出力が多い場合、grepでフィルタする
$ apt-file search --regexp <正規表現>      ←オプションを付ければ正規表現で検索も可能
```

注6）逆にapt-get build-depで取得したパッケージをまとめて一気に消す方法は、今のところないようです。
注7）URL https://www.debian.org/distrib/packages

助けになります。コンソールから呼び出せるので、リズムを崩さずに作業できます。また、単純な検索ツールですのでgrepなどと組み合わせて容易に絞り込みができます(**図3**)。

Debian上でソフトウェアの開発作業をする際、必要なファイルを探してGoogle検索して右往左往するよりも、上記のような解決策がすでに用意されているのを知っているとグッとQOLが上がることでしょう。ぜひ活用ください。

イベントの報告&お知らせ

6月14日に、Debian JP Projectとして「オープンソースカンファレンス2014 Hokkaido」(以下、OSC)に参加出展しました。イベントには数百名が参加し盛況の中、筆者は「Does Cowgirl Dream of Red Swirl ?」(カウガールは赤い渦巻きの夢を見るか)[注8]と題して、Debianの開発の流れや、次期リリースDebian 8"Jessie"の展望などを説明しました(**写真1**)。参加者のアンケート結果を見ると、幾人かからお褒めの言葉をいただけたのでうれしい限りです。

また、翌日に札幌市内で「Debian meetup Hokkaido 14.06」を開催し、前日のOSCの振り返りや各自のDebian関連作業を行いました。

筆者は、本誌連載「Ubuntu Monthly Report」の執筆などでお馴染みのUbuntu Japanese Teamの水野源さんがメンテしている「silversearcher-ag」パッケージ[注9]のレビューを手伝ったり、コミュニティ活動の事務作業的な事柄を淡々とこなしたりしていました。また来年などに機会があれば来訪したいと思っていますので、北海道のみなさまよろしくお願い致します。OSC北海道スタッフのみなさま、景品提供に協力いただいた編集担当さま、快くMeetupイベントの会場を貸していただいた㈱インフィニットループの方々[注10]、そして参加していただいたみなさまに感謝です。

次のイベントとしては、京都リサーチパーク(KRP)で開催されるOSC 2014 Kansai@Kyoto[注11]に関西Debian勉強会が参加出展を行います。近隣の方はぜひご参加ください。参加日は8月2日(土)で、セミナーは本連載でもRuby関連記事を寄稿いただいた佐々木洋平さんが担当する予定です。

なお、東京エリアDebian勉強会／関西Debian勉強会は毎月開催されていますので、折をみて参加いただければと思います。開催日程／場所については、Debian JP Projectのサイト[注12]に近日予定しているイベント一覧がありますので、そちらを参考にしてください(載っていない場合は筆者の怠慢ですのでツッコミを入れていただければ……)。

なお、勉強会は毎度ネタに飢えていますので、「こんな話を聞いてみたい」(あるいは「こんな話をしてみたい」)という提案も歓迎します。詳しくは参加時に相談してみると良いでしょう(おもしろい話ができると思いますよ)。SD

注8) タイトルの元ネタは「アンドロイドは電気羊の夢を見るか」です。

注9) grepを賢くしたようなソフトウェア「ag」のパッケージです。詳細についてはgihyo.jpのUbuntu Weekly Recipe第287回「Ubuntuで超高速grep『The Silver Searcher』を使う」URL http://gihyo.jp/admin/serial/01/ubuntu-recipe/0287をご覧ください。

注10) インフィニットループさんは今回のようなイベントに会議室の貸出を随時行っているそうですので、札幌近郊で勉強会などのイベント開催を検討されている方にお勧めです(会場には「うまい棒」の箱があって、自由に食べられるようになっているなど太っ腹)。詳しくはURL http://www.infiniteloop.co.jp/をご覧ください。

注11) URL http://www.ospn.jp/osc2014-kyoto/

注12) URL http://www.debian.or.jp/community/events/

▼写真1　OSC北海道での筆者の発表

OSSに求められる合理性

平 初
Hajime TAIRA
レッドハット（株）グローバルサービス本部
プラットフォームソリューション統括部
ソリューションアーキテクト

RHELとFedoraの関係

本連載第2回を執筆した平です。2年ぶりですが、今回も恵比寿にある「世界最大のオープンソースの会社」（の東京オフィス）のソリューションアーキテクトが日々、何をしているのか紹介します。第2回の恵比寿通信では「OSSで飯を食うということ」をテーマに書いてから2年経過しましたが、おかげさまで飯は食えています。

最近のホットトピックと言えば、米国時間の2014年6月10日に、待望のRed Hat Enterprise Linux 7（RHEL7）がリリースされました。前回のバージョン6から約3年の歳月を経てリリースされた最先端のOSです。RHEL7はFedora 18と19をベースとしたパッケージで構成されています。

製品ライフサイクルとして10年サポートを基本としており、ミッションクリティカル向けのサポートプログラムを適用すると13年間使えるLinuxディストリビューションです（**図1**）。13年間も同じカーネルを使うとか正直ぞっとしますね。

RHELの話は、ここまで。今回はRHELの開発ベースとなっているFedoraについて書きたいと思います。Fedoraはコミュニティベースで開発が行われているLinuxディストリビューションです。2003年にWarren Togami氏が前身となるFedora Linux Projectを立ちあげたのがきっかけです。そのころはOS本体ではなくRed Hat Linux用の追加ソフトウェアのyumとaptのリポジトリを提供していました。

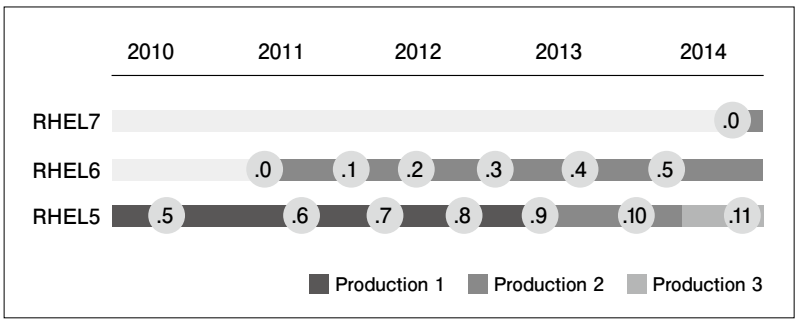
そして、そのころRed Hat Linuxの今後の方針を模索していたRed Hat社に対して、Fedora Linux Projectのリーダーであったwarren氏が話を持ちかけてできたのがThe Fedora ProjectとFedora Core 1（現在のFedora）というLinuxディストリビューションです。

Fedoraはリリースサイクルは約6ヵ月で、ライフサイクルは約12ヵ月です。最新のFedoraがリリースされたときに2つ前のバージョンがサポートから外れます。そんなサイクルで開発を続け、Fedora 20のリリースを迎えました。そして今はFedora 21の準備を行っています。

筆者とFedoraとの出会い

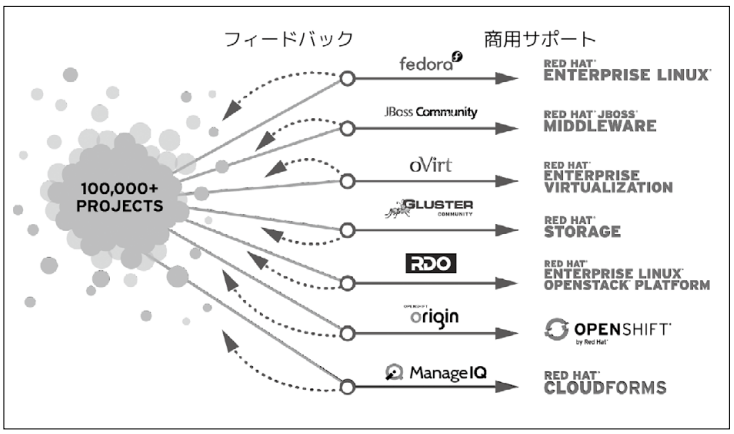
筆者がThe Fedora Projectに加わったのは2004年のことで、かれこれもう10年になります。

▼図1　現在サポート提供中のRHELのバージョンとライフサイクル



オフ会に参加してしまったのがきっかけで、今ではFedora L10N Japanese Teamのコーディネーターを務めています。当時はThe Fedora Projectのほかに、重松直樹氏が立ちあげたFedora JP Projectという日本人および日本語を母語とする人たちのためのローカルコミュニティがありました。

▼図2　Upstream first!



また、http://fedora.jp/というサイトが2008年ごろまでありました。Fedora JP Projectでは、アプリケーションの日本語ローカライゼーションを行ったり、独自の日本語インストレーションガイドを作成したり、2005年ぐらいから有志による勉強会なども開催していて割と活発に活動していました。今で言う勉強会ブームの先駆けだったと思います。

Fedora JP Projectを閉じた理由

Linuxディストリビューションとしての Fedoraとしても、EUC-JPからUTF-8への文字コード移行問題をはじめとする日本語環境の安定化が落ち着いてきたというのと同時に、そのころのFedora JP Project関係者の中には“Go upstream!”という思いがありました。日本のローカルコミュニティの中だけでタコ壺になっていないで、Fedoraに収録されるさまざまなソフトウェアのコミュニティのUpsteam(上流にある開発コミュニティ)で創造的な作業をしようという合い言葉です。

そこでFedora JP Projectは、すべての創造的活動をThe Fedora Projectで行うべきだという当時のコアメンバーの意向で2008年に発展的解散(The Fedora Projectへの移行)をしました。

Red Hatの開発では“Upstream first”という手法が行われます(**図2**)。これは開発コミュニティに対してバグ修正や機能改善を行ったうえで、自社のディストリビューションの中のパッケージに修正を取り込むというやり方です。オープンソースソフトウェアにおいて、ソースコードを公開しない独自拡張をメンテナンスするためにフォークするのは手間もコストもかかり合理的な選択肢ではありません。GPL以外の一部のライセンスではフォーク後のソースコード開示義務がないライセンスも存在しますが、どこかの会社がクローズドな独自拡張の部分実装を抱え込むのは合理的ではありません。

同じくFedoraのコミュニティもローカルでコソコソやるのではなく世界に1つのほうが合理的だという結論を導き出したのです。

有益な情報、どこに書いていますか？

時々、日本のエンジニアの方で、海外の有益な情報を独自に日本語化してブログに掲載される方がいますが、情報の価値を高めるのためにはUpstreamの開発コミュニティのサイトに掲載してもらうように調整するほうが合理的です。個人のブログはSEO的にもヒットする可能性が低いですし、鮮度が保てないとすぐにインターネットという広い宇宙の塵となってしまいます。“Go upstream!” SD

SoftEther VPNをUbuntuだけで使用する

Ubuntu Japanese Team
あわしろいくや　ikuya@fruitsbasket.info

今回はソフトウェアVPN（Virtual Private Network）サーバ／クライアントのSoftEther VPNを、Windowsを使わず簡単に使用する方法を紹介します。

SoftEther VPNとは

SoftEther VPNは、本誌の読者であればどこかで耳にしたことがあるのではないでしょうか。簡単に解説すると、ソフトイーサ社[注1]のPacketix VPN 4.0の一部の機能を除いて[注2]GPL2で公開されているソフトウェアVPNサーバ／クライアントです。とにかく難しいことが簡単にできてしまいます。

方針

実のところSoftEther VPNはUbuntuというか非Windows環境だけで使用するのはなかなかたいへんなのですが、Windows用のサーバ設定ツールはWine[注3]でも動作するので、今回はWineを使用します。ちなみにLAN内のどこか[注4]にWindowsがあれば設定ツールが使えるので、実運用の際にはWineを使う機会はあまりないでしょう。Wineは依存の関係上、大量にパッケージをインストールするのでインストールしたくないということもあると思います。その場合も、別途仮想マシンにWineをインストールしたUbuntuがあればいいということでもあります。GUIのないUbuntu ServerでSoftEther VPNを動作させ、設定自体は別のUbuntuから、というのが割と現実的な想定環境でしょう。

ただしWindows用のクライアント設定ツールはWineではうまく動作しないため、コマンドからすべての設定を行います。サーバの設定と比較すると簡単なので、順番さえ間違えなければ設定できるはずです。

ソースコードの中にはdebianフォルダがあり、Debパッケージが比較的簡単に作れるようになっています。make installするよりもパッケージ化してしまったほうがメンテナンスが楽なのはいうまでもないので、今回はそうします。

サーバのネットワーク環境は、ルータの設定ができない完全なNAT（Network Address Translation）の中にあります。クライアントはスマートフォン（au）のLTE回線を使用します。いわゆるテザリングでUbuntu[注5]がインストールしてあるノートPCにつないでいます。SoftEther VPNがあったら便利だと言える一般的な環境だと思います。

ソースコードのダウンロード

ソースコードはGitHub[注6]から取得してもいいので

注1）http://www.softether.jp/
注2）http://ja.softether.org/3-spec/cureent_limitations
注3）これも解説は不要かと思いますが、LinuxでWindowsのプログラムを動作させるソフトです。
注4）厳密にいえばIPアドレスがあればいいのでWANからでもいいかもしれません。
注5）実際にはUbuntu GNOMEです。
注6）https://github.com/SoftEtherVPN/SoftEtherVPN/

すが、今回はtarball（圧縮したtar形式のファイル）を取得します。ソースコードのダウンロードページ[注7]にアクセスし、［ダウンロードするソフトウェアを選択］を［SoftEther VPN（Freeware）］にします。［コンポーネントを選択］は［Source Code of SoftEther VPN］にします。［プラットフォームを選択］は［tar.gz package］にします。各バージョンが表示されますが、今回は執筆時点で最新のRTM版である［Ver 4.08, Build 9449, rtm］にしました。ソースコードのダウンロードが終わったら、［コンポーネントを選択］を［SoftEther VPN Server Manager for Windows］に変更し、［プラットフォームを選択］を［Windows（.zip package without installers）］にし、サーバ設定ツールもダウンロードします。

バイナリパッケージの作成とインストール

方針に従ってDebパッケージを作成しますが、詳しく解説すると、それだけで紙幅が尽きてしまうため、ごくごく簡単に解説します。

ホームフォルダにsoftether-vpnというフォルダを作成し、そこに先ほどダウンロードしたソースコードを移動したと仮定して話を進めます。

次のコマンドを実行してソースコードを伸張します。

```
$ cd ~/softether-vpn
$ tar xf softether-src-v4.08-9449-rtm.tar.gz
$ cd v4.08-9449
```

ビルドに必要なパッケージをインストールします。

```
$ sudo apt-get install dpkg-dev fakeroot debhelper libncurses-dev libssl-dev libreadline-dev
```

debianフォルダーにはchangelogファイルがあって、パッケージのバージョンはここで決定するのですが、添付されているスクリプトを使用してこのファイルを更新します。そのスクリプトとはdch-generate.shです。まずはこれを編集してください。

```
$ editor debian/dch-generate.sh
```

変更するのはこの3つの変数です。

```
status="trusty"
DEBFULLNAME="AWASHIRO Ikuya"
DEBEMAIL="ikuya@fruitsbasket.info"
```

見てのとおりですが、“status”はUbuntuのバージョンのコードネームを入れてください。例の“trusty”は14.04で、12.04だと“precise”です。“DEBFULLNAME”はご自分の名前を、“DEBEMAIL”はご自分のメールアドレスを入れてください。

編集が完了したら次のコマンドを実行してください。

```
$ debian/dch-generate.sh > debian/changelog
```

このままパッケージを作成してしまうと、サーバもクライアントも[注8]自動で起動しないので、いちいち使用するときに起動する必要があり、非常に不便です。debianフォルダにinit.dというフォルダがあり、ここにvpnserverというサーバ用の起動スクリプトがありますが、このままだと使われません。debianフォルダにパッケージ名.initというファイルがあれば（すなわちdebian/softether-vpnserver.initというファイルがあれば）パッケージに収録されますが、多少修正が必要なので差分を**リスト1**に掲載します。

同時にクライアントも常時起動しておくと便利なのでdebian/softether-vpnclient.initを作成します。これもinit.d/vpnserverへの差分として**リスト2**に掲載します。もちろん常時起動する必要がないというのであれば、この起動スクリプトをパッケージに含めなくてもけっこうです。注意点としては、SoftEther VPNの解説によっては起動する際に、

```
$ sudo /etc/init.d/vpnserver start
```

もしくは、

```
$ sudo service vpnserver start
```

というコマンドを紹介していることがあるかもしれませんが、これは、

注7） http://www.softether-download.com/ja.aspx

注8） SoftEther VPNはクライアントも常時起動しておくのが便利そうな作りです。

```
$ sudo /etc/init.d/softether-vpnserver start
```

もしくは、

```
$ sudo service softether-vpnserver start
```

と読み替えてください。あとは実際にパッケージを作成します。

```
$ dpkg-buildpacka  ge -r -uc -b
```

リスト1　debian/softether-vpnserver.initへの差分

```
--- debian/init.d/vpnserver
+++ debian/softether-vpnserver.init
@@ -8,7 +8,7 @@

 PATH=/usr/local/sbin:/usr/local/bin:/sbin:/
bin:/usr/sbin:/usr/bin:/usr/qmsys/bin
 DAEMON=/usr/bin/vpnserver
-NAME=vpnserver
+NAME=softether-vpnserver
 DESC="SoftEtherVPN Server"

 PIDFILE=/var/run/$NAME.pid
@@ -25,13 +25,13 @@

 case "$1" in
   start)
-  echo -n "Starting $DESC: $NAME"
-  $DAEMON -start
+  echo -n "Starting $DESC: "
+  $DAEMON start
   echo "."
   ;;
   stop)
-  echo -n "Stopping $DESC: $NAME"
-  $DAEMON -stop
+  echo -n "Stopping $DESC: "
+  $DAEMON stop
   echo "."
   ;;
   #reload)
@@ -53,10 +53,10 @@
   # option to the "reload" entry above. If
not, "force-reload" is
   # just the same as "restart".
   #
-  echo -n "Restarting $DESC: $NAME"
-  $DAEMON -stop
+  echo -n "Restarting $DESC: "
+  $DAEMON stop
   sleep 1
-  $DAEMON -start
+  $DAEMON start
   echo "."
   ;;
   *)
```

図1　インストール方法

```
$ cd ../
$ sudo dpkg -i softether-vpnserver_4.08.9449-rtm_amd64.deb softether-vpncmd_4.08.9449-rtm_amd64.deb
$ sudo dpkg -i softether-vpnclient_4.08.9449-rtm_amd64.deb softether-vpncmd_4.08.9449-rtm_amd64.deb
```

無事に完了すると、1つ上のフォルダに拡張子が.debの4つのパッケージが作成されます。インストール方法は**図1**のとおりです。サーバのインストールは1行目と2行目を、クライアントのインストールは1行目と3行目を実行してください。両方インストールする場合は3つのパッケージを同時にインストールしてください。softether-vpncmdパッケー

リスト2　debian/softether-vpnclient.initへの差分

```
--- debian/init.d/vpnserver
+++ debian/softether-vpnclient.init
@@ -7,9 +7,9 @@
 set -e

 PATH=/usr/local/sbin:/usr/local/bin:/sbin:/
bin:/usr/sbin:/usr/bin:/usr/qmsys/bin
-DAEMON=/usr/bin/vpnserver
-NAME=vpnserver
-DESC="SoftEtherVPN Server"
+DAEMON=/usr/bin/vpnclient
+NAME=softether-vpnclient
+DESC="SoftEtherVPN Client"

 PIDFILE=/var/run/$NAME.pid
 SCRIPTNAME=/etc/init.d/$NAME
@@ -25,13 +25,13 @@

 case "$1" in
   start)
-  echo -n "Starting $DESC: $NAME"
-  $DAEMON -start
+  echo -n "Starting $DESC: "
+  $DAEMON start
   echo "."
   ;;
   stop)
-  echo -n "Stopping $DESC: $NAME"
-  $DAEMON -stop
+  echo -n "Stopping $DESC: "
+  $DAEMON stop
   echo "."
   ;;
   #reload)
@@ -53,10 +53,10 @@
   # option to the "reload" entry above. If
not, "force-reload" is
   # just the same as "restart".
   #
-  echo -n "Restarting $DESC: $NAME"
-  $DAEMON -stop
+  echo -n "Restarting $DESC: "
+  $DAEMON stop
   sleep 1
-  $DAEMON -start
+  $DAEMON start
   echo "."
   ;;
   *)
```

ジのインストールは必須です。

サーバの設定と接続

サーバの設定は前述のとおりWindows用の設定ツールを使用します。ということはWineのインストールが必須です。次のコマンドでインストールしてください。

```
$ sudo apt-get install wine
```

先ほどダウンロードしたsoftether-vpn_admin_tools-(中略).zipを伸張し、vpnsmgr.exeを右クリックして[Wine Windowsプログラムローダー]をクリックしてください。すると[SoftEther VPN サーバー管理マネージャー]というウィンドウが表示されます。[新しい接続設定]をクリックしてください。

[新しい接続設定の作成]というウィンドウが表示されますので、[接続設定名]と[ホスト名]と[管理パスワード]を入力してください。ただし[ホスト名]以外はあまり真面目に入力しなくても大丈夫です。入力が終わったら[OK]をクリックしてください。すると前の[SoftEther VPN サーバー管理マネージャー]に戻ります。[接続]というボタンが押せるようになっているのでこれをクリックします。

今度はユーザ名とパスワードを入力するウィンドウが表示されますが、ここではパスワードを空欄のままにするのがポイントです。先ほど入力したパスワードは入力しないでください。そのまま[OK]をクリックするとあらためてパスワードを設定するダイアログが表示されるので、入力してください。あまり真面目に入力しなくても大丈夫です、というのは、ここであらためて入力するから、という意味です。

次に進むと[SoftEther VPN Server / Bridge 簡易セットアップ]というウィンドウが表示されます。一番上の[リモートアクセス VPN サーバー]にチェックを入れ、[次へ]をクリックしてください。するとダイアログが表示され、先に(不真面目に)設定した内容が初期化される旨の確認をされますので、[はい]をクリックしてください。

今度は[仮想HUB名]を入力するダイアログが表示されます。これは重要ですので真面目に入力してほしいのですが、デフォルトのままでもさしたる問題はありません。入力して[OK]をクリックすると[ダイナミックDNS機能]の設定ができます。覚えやすいものに変更しておくといいでしょう。その次は[VPN Azureサービス]を使用するかどうかですが、今回のような用途では必須です。これも必要であればドメインを変更してください。

いよいよユーザ名とパスワードの設定です。[ユーザーを作成する]をクリックして新規作成ウィンドウを表示します。今回はパスワード認証にしますので、[ユーザー名][本名][パスワード][パスワード確認入力]の4つに入力すればいいです。あとは[ローカルブリッジの設定]でEthernetデバイスを選択すれば設定はおしまいです。仮想マシンで使用している場合はさらにウィンドウが表示されますが、詳細はコラム「SoftEther VPNを仮想マシンのゲストOSで動作させる場合」をご覧ください。

クライアントの設定と接続

前述のとおりクライアントはauのLTE回線を経由したUbuntuから接続します。まずは次のコマンドを実行してください。

```
$ vpncmd
```

すると何を行うかの選択が表示されますので、今回は2を押して[VPN Client の管理]を選択します。次に接続先が表示されますが、今回はlocalhostですのでそのまま Enter キーを押してください。すると

```
VPN Client>
```

というプロンプトが表示されますので、「NicCreate」と入力して Enter キーを押してください。クライアントの仮想LANカード名を尋ねられますので、任意のものを入力してください。次ですぐに使用します。

続いて、「AccountCreate」を実行してください。ここで尋ねられるのは順番に接続設定の名前、接続先VPN Serverのホスト名とポート番号、接続先仮想HUB名、接続するユーザ名、使用する仮想LAN

カード名です。接続設定の名前は任意のもので、使用する仮想LANカード名は先ほど設定したものです。残りはサーバの設定を使用しますが、接続先VPN Serverはvpnazure.netドメインで、ポートは443としました。

まだパスワードは設定していないので、「AccountPasswordSet」を実行してください。接続設定の名前（先ほど任意で決定したもの）とパスワードを入力します。[standard または radius の指定]も聞かれますが、“standard”（引用符は不要）と入力してください。

いよいよ接続します。

```
AccountConnect（接続設定の名前）
```

で接続しますが、接続されているかどうかはわかりません。確認のために、

```
AccountStatusGet（接続設定の名前）
```

を入力し、[セッション接続状態]が[接続完了（セッション確立済み）]になっていれば接続されています。[再試行中]だと接続されていないので、「AccountDisconnect」で切断したあと、「AccountDelete」でアカウントを削除し、再設定してください。

無事接続完了になったら、今度は/etc/network/interfacesファイルに、

```
iface vpn_（仮想LANカード名）inet dhcp
```

を追記し、

```
$ sudo ifup vpn_（仮想LANカード名）
```

を実行すると接続先のIPアドレスを取得し、実際にアクセスできるようになります。仮想LANカード名がわからない場合は、ifconfigコマンドを実行するとわかります。SD

SoftEther VPNを仮想マシンのゲストOSで動作させる場合

今回の環境はVirtualBoxのゲストOSにXubuntu 14.04 LTSをインストールし、作成しました。仮想マシンのゲストOSでSoftEther VPNサーバを動作させる場合は、ネットワークアダプタをブリッジモードにする必要があります。あとは、忘れずにプロミスキャスモードを有効にしてください。**図2**はVirtualBoxでの設定例です。

図2　VirtualBoxのネットワーク設定には、そのものずばりプロミスキャスモードという設定項目がある

Software Design plus

Software Design plusシリーズは、OSとネットワーク、IT環境を支えるエンジニアの総合誌『Software Design』編集部が自信を持ってお届けする書籍シリーズです。

iOSアプリエンジニア養成読本
髙橋 俊光、諏訪 悠紀、湯村 翼、平屋 真吾、平井 祐樹 著
定価 1,980円+税　ISBN 978-4-7741-6385-7

[改訂新版]Linuxエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6377-2

Webアプリエンジニア養成読本
和田 裕介、石田 絢一(uzulla)、すがわら まさのり、斎藤 祐一郎 著
定価 1,880円+税　ISBN 978-4-7741-6367-3

Androidライブラリ実践活用
菊田 剛 著
定価 2,480円+税　ISBN 978-4-7741-6128-0

[改訂新版]Apache Solr入門
大谷 純、阿部 慎一朗、大須賀 稔、北野 太郎、鈴木 教嗣、平賀 一昭 著
定価 3,600円+税　ISBN 978-4-7741-6163-1

レベルアップObjective-C
沼田 哲史 著
定価 3,200円+税　ISBN 978-4-7741-6076-4

おいしいClojure入門
ニコラ・モドリック、安部 重成 著
定価 2,780円+税　ISBN 978-4-7741-5991-1

はじめての3Dプリンタ
水野 操、平本 知樹、神田 沙織、野村 毅 著
定価 2,480円+税　ISBN 978-4-7741-5973-7

PHPエンジニア養成読本
PHPエンジニア養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5971-3

独習Linux専科
中井 悦司 著
定価 2,980円+税　ISBN 978-4-7741-5937-9

データサイエンティスト養成読本
データサイエンティスト養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5896-9

Androidエンジニア養成読本Vol.2
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-5888-4

Raspberry Pi[実用]入門
Japanese Raspberry Pi Users Group 著
定価 2,380円+税　ISBN 978-4-7741-5855-6

Linuxシステム[実践]入門
沓名 亮典 著
定価 2,880円+税　ISBN 978-4-7741-5813-6

データベースエンジニア養成読本
データベースエンジニア養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5806-8

小飼弾のコードなエッセイ
小飼 弾 著
定価 2,080円+税　ISBN 978-4-7741-5664-4

JavaScriptライブラリ実践活用
WINGSプロジェクト 著
定価 2,580円+税　ISBN 978-4-7741-5611-8

〈改訂〉Trac入門
菅野 裕、今田 忠博、近藤 正裕、杉本 琢磨 著
定価 3,200円+税　ISBN 978-4-7741-5567-8

サウンドプログラミング入門
青木 直史 著
定価 2,980円+税　ISBN 978-4-7741-5522-7

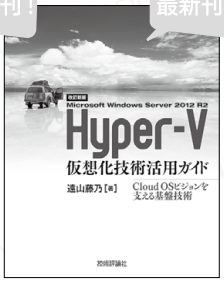
最新刊！

養成読本編集部 編
B5判・212ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6578-3

最新刊！

WINGSプロジェクト 著
B5判・256ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-6566-0

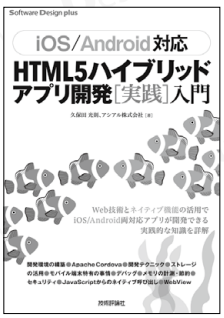
最新刊！



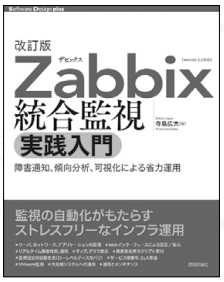
遠山 藤乃 著
B5変形判・392ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6571-4

森藤 大地、あんちべ 著
A5判・296ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-6326-0

㈱パイプドビッツ 著
A5判・224ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-6205-8



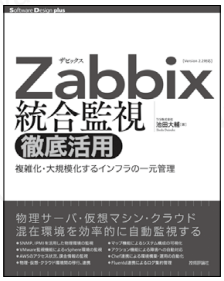
久保田 光則、アシアル㈱ 著
A5判・384ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-6211-9



寺島 広大 著
B5変形判・440ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6543-1

松本 直人、さくらインターネット研究所 (日本Vyattaユーザー会) 著
B5変形判・320ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6553-0

乾 正知 著
B5変形判・352ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-6304-8



TIS㈱ 池田 大輔 著
B5変形判・384ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6288-1

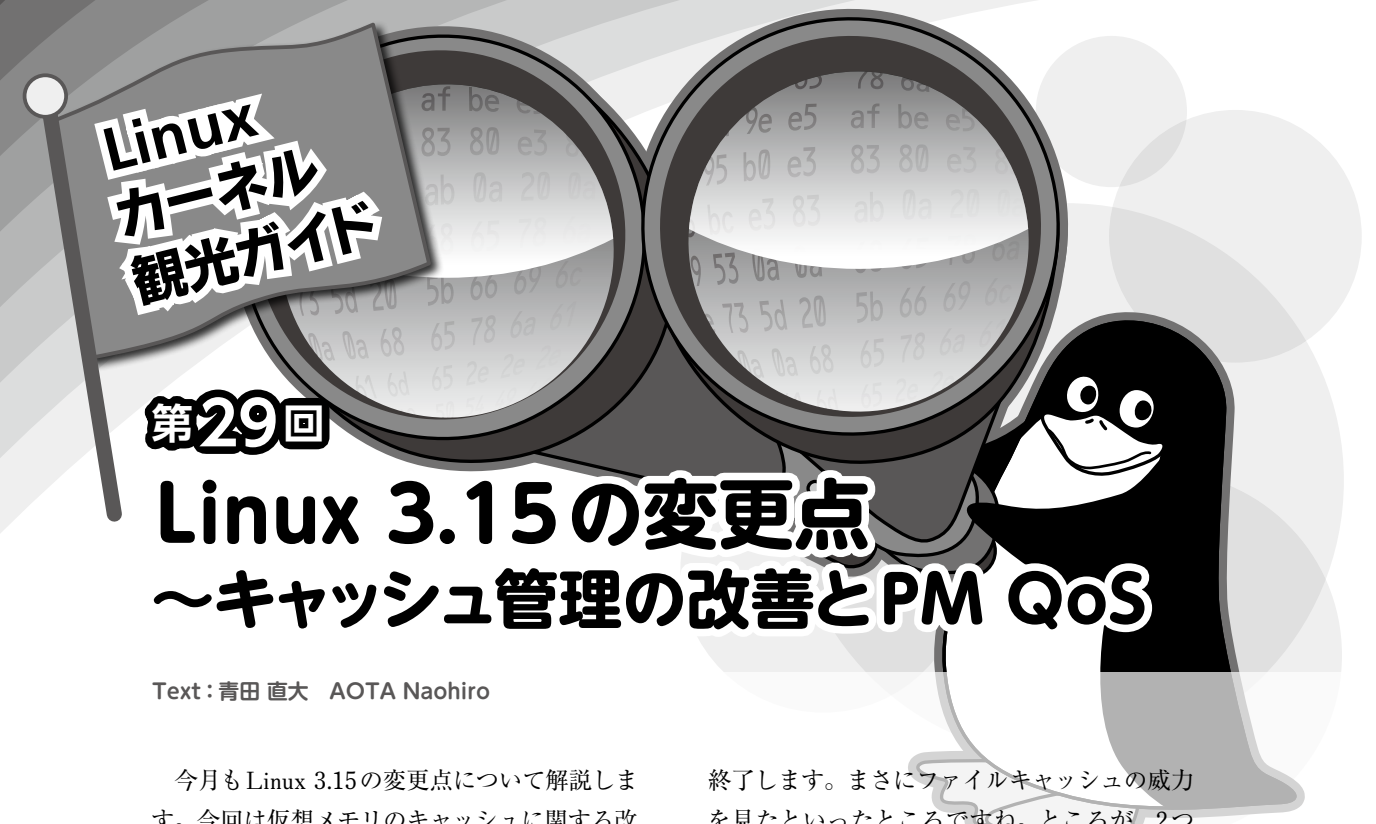
養成読本編集部 編
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6429-8

養成読本編集部 編
B5判・184ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6424-3

養成読本編集部 編
B5判・196ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6425-0

養成読本編集部 編
B5判・216ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6422-9

技術評論社　〒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29回

Linux 3.15の変更点 ～キャッシュ管理の改善とPM QoS

Text：青田 直大　AOTA Naohiro

今月もLinux 3.15の変更点について解説します。今回は仮想メモリのキャッシュに関する改良とスリープからの復帰レイテンシなどの制限を、さまざまな場所からシステムに設定する機能を提供する、PM QoSフレームワークの新機能について解説します。

Linuxのメモリ管理とビッグデータ

まず始めに去年の11月に“Linux memory manager and your big data”というタイトルのCitus Data社のブログ記事[注1]に書かれている例について紹介します。

そこでは2つの、いずれも42GBという大きなCSVデータを順番に“wc -l”で行数を数えるという簡単な処理を行っています。この処理が行われたマシンのメモリサイズは68GBで、これらのファイルサイズはメモリのサイズの60%ほどにはなりますが、個々のファイルは十分メモリに入るサイズです。

1つめのファイルを2回“wc -l”してみると、1度目はディスクからの読み込みですので10分ほどかかりますが、2度目はメモリ上にデータがキャッシュされていることから20秒も経たずに終了します。まさにファイルキャッシュの威力を見たといったところですね。ところが、2つめのファイルも同様に“wc -l”にかけてみてもまったく所要時間が変わりません。本来ならば2つめのファイルもキャッシュされて高速になるはずです。しかし、何度やってもキャッシュを一度完全にクリアしてからやりなおさない限り、処理はまったく速くなりません。実はLinux 3.15ではこの問題を解決する変更がメモリ管理システムに入っています。では、いったい何が起きていて、Linux 3.15ではどのようにこの問題を解決したのかを見ていきましょう。

キャッシュリスト管理の改良

まずメモリキャッシュ管理の改良について解説します。一般にOSではメモリよりも格段に遅いディスクからデータ(ページ)を読み出したときに、あとでもう一度そのデータが必要になったときに備えて、メモリの中にそのデータをキャッシュしておきます。ですが、メモリの中にディスクのデータをすべてキャッシュしておくことは不可能ですので、何をキャッシュし、何を捨てれば良いのかが問題になります。そこで何を残し、何を捨てるかの判定のためにLRU(least recently used)リストというものを使用します。

注1）http://citusdata.com/blog/72-linux-memory-manager-and-your-big-data

このリストは、先頭が一番最近に使用されたページに、末尾がリストの中で使用された時間が最も古いページになるように整列されているリストです。新しく読み込んだページ、またはリストの中で使用されたページをリストの先頭に移し、キャッシュの削除は末尾から行われます。

2つのリスト

LRUリストが単純に1つだけという実装では、新しくファイルを読み込んだときに問題が出てきます。すなわち、新しいファイルを一度しか読み込まなくても、その一度の読み込みがLRUリストの中の本当によく使われるページを簡単に追い出してしまいます。そこでactive/inactiveと2つのリストを作ります。

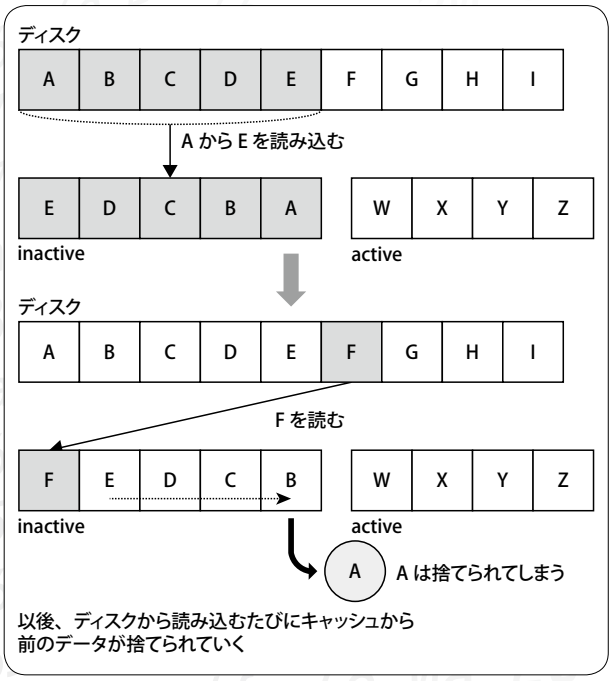
初めにページを読み込んだときは、そのページをinactiveリストの先頭に配置します。このinactiveリストの中のページが再度読み込まれると、このページはactiveリストに移動されます。また、メモリ逼迫(ひっぱく)時のページ回収はinactiveのリストからだけ行います。こうすることで、先ほどのような一度しか読み込まれないファイルが、よくアクセスされるページを追い出してしまう問題を避けることができます。

次に問題となるのはactiveリストとinactiveリストとのバランスです。ページの回収が、inactiveリストからだけ行われるので、activeからinactiveへのページの移動を行わないと、inactiveにいる間にもう一度アクセスされたページはいつまでもactiveリストに居座ったまま回収されないことになります。このactiveからinactiveへのページ移動の条件として、ページを回収するときに、activeリストがinactiveリストよりも長くなっていればactiveリストからいくつかのページをinactiveリストの先頭へと移動するというルールを使っています。こうすることでactiveリストとinactiveリストがキャッシュに使えるメモリの中で半々となり、よく使われるページをactiveリストで保護することと、なおかつ新しく頻繁に使われるようになってきたページを検出することとの両方をほどよく実現できます。

キャッシュが有効に働かない場合

さて、このようにキャッシュ管理はうまく考えられているわけですが、ある種のワークロードではうまく動かないことがあります。たとえばページAからページIまでの9つのページを何度も繰り返して読み込む作業を考えてみましょう(**図1**)。もしもinactiveのリストのサイズが9よりも小さい場合には、inactiveリストにページAからIのすべてをキャッシュしておくことはできないのでこの作業ではページを1つ読み込むと、毎回以前にキャッシュしていたページが捨てられ、必要になったページをディスクから読み込んでくることになります。もちろんキャッシュに使うことができるメモリのサイズが本当に9よりも小さいのであれば、これはしかたのないことです。しかし、キャッシュに使うことができるサイズには余裕があるのに以前にactiveリストに入って今はほとんど使われていないページの

▼図1　キャッシュからデータが捨てられてしまう



ために、今本当に必要なページをキャッシュに乗せておくことに失敗しているのであれば、これは改善すべき状況と言えるでしょう。

Linux 3.15での動作

では、どうやってこのような状況を改善できるでしょうか。もちろんどのページがどのようにアクセスされているかを完璧に追跡しておけば、より頻繁にアクセスされているページを優先し、activeリストに居座っている、以前には何度かアクセスされたものの今はアクセスされていないページを追い出すこともできるでしょう。しかし、今の実装がそうなってないことからもわかるように、すべてのページアクセスを追跡しておくことはとてつもなくコストの高い処理で現実的ではありません。

そこでLinux 3.15ではページがキャッシュから削除されてから、次にアクセスされてキャッシュに読み込まれるまでにどれぐらいの「距離」があったのかを追跡し、その「距離」に応じて読み込んだページをいきなりactiveリストに追加する変更が導入されました。

「ページアクセスの距離」について見ていく前に、前述のような例でのinactiveリスト内のページの動きについてよく見てみましょう。新しく読み込まれたページはまずinactiveリストの先頭に配置されます。このページはinactiveリストの中でどのように動いていくでしょうか。この注目されたページへのアクセスを除くと、ページへのアクセスは次の3つに分類できます。

❶activeリスト内のページへのアクセス
❷inactiveリスト内のページへのアクセス
❸active/inactiveどちらのリストにも入っていないページへのアクセス

❶の場合にはactiveリスト内で順番の変更があるだけで、inactiveのリストは変わることはなく、注目しているページは動くことはありません。

❷の場合には、アクセスされたinactiveリスト内のページがactiveリストへと移動されるので、注目しているページはinactiveリストの末尾のほうへと1つスライドします。

最後に❸の場合には、アクセスされたページは新しくinactiveリストへと追加され、inactiveリストの末尾のページがキャッシュから追い出されることになるので、この場合も注目しているページはinactiveリストの末尾へと1つスライドすることになります。

以上のことから、inactiveリストからページが追い出された数、およびページがactiveリストに移動された数とを合計しておき、ある2つの時点におけるその合計の差をとることによって、その2つの時点の間にいくつのinactiveなページがアクセスされたかの最低値を示すことになります。また、inactiveリストの中のあるページがN個末尾に近付くには、最低でもN回のinactiveページへのアクセスが起きていることがわかります。すなわち、あるページがinactiveリストから追い出されたときには、少なくともinactiveリストの長さの数と同じだけのinactiveリストへのアクセスが起きています。

inactive_ageの利用

また、あるページがinactiveリストから追い出されるときに、「**そのときまでのinactiveリストからページが追い出された数、およびページがactiveリストに移動された数の合計**」(以下、inactive_ageと呼びます)がいくつのときにそのページが追い出されたのかを記録しておき、その後そのページがふたたび読み込まれ、inactiveリストに入ってきたときのinactive_ageとの差をとると、それはそのページがキャッシュの外に置かれていた時間を示します。この差をページが追い出されてからまたアクセスされる(refaultする)間の距離と見て“refault distance”と呼びます。

すると、このページが一度アクセスされてから、(一度キャッシュから消えて)ふたたびアクセスされるまでには最低でも、“**<inactiveリストの長さ>＋<refault distance>**”の数だけのページ

がアクセスされていることがわかります。もしもこの「アクセス距離」がキャッシュ内に保持し得るページの数以下であれば、このページはキャッシュ内に残しておくほうが良かったということがわかります。

「キャッシュ内に保持し得るページの数」というのは“**<inactiveリストの長さ>＋<activeリストの長さ>**”であることから、整理すると次の式が成り立てばそのページはキャッシュの中に残しておいたほうが良かったということになります。

<refault distance> ≦ <activeリストの長さ> ……❶

結局のところ、このページがactiveリストに入ることができなかったのはinactiveリストの長さが不足しているからだということになります。かと言って、inactiveのリストを伸ばしてしまうのは、それだけ activeリストの長さを小さくすることを意味し、よくアクセスされているページのキャッシュ保持数を減らしてしまうことにつながるので好ましくはありません。そこで、❶式が成り立ったページは、通常キャッシュに読み込まれたページのようにinactiveリストに入れるのではなく、いきなりactiveリストへと登録します。こうすることで、activeリストに居座っている、もう今ではあまりアクセスされていないページをacitveからinactiveへと追い出すことができるようになります。もちろん本当によくアクセスされるページだけがactiveリストに残っていたときも、これらのページが再度activeになり、今いきなりactiveリストに入ることができたページはやがてはinactiveになり、キャッシュから追い出されていくのでこの場合も問題はありません。

動作例

最後に簡単な動作例を見てみましょう(**図2**)。activeリストにもうアクセスされていないものの過去にはアクセスされていたW、X、Y、Zという4つのページがあるという状況で、AからIのページに何度も繰返し(これらのページだけを)アクセスしていく状況を考えてみましょう。ここでキャッシュに使うことができるページの数は9とします。また、わかりやすい例にするため、最初のinactive_ageを0とします。

AからEを読み込むまではキャッシュに余裕があるので、キャッシュから追い出されるページもなく、inactive_ageは0のまま推移します。次にFを読み込もうとすると、それに伴ってAがキャッシュから追い出され、invalid_ageは1となります。このときに“A”がinvalid_age=1のときに追い出されたと記録をとっておきます。

このままG、H、Iを読み込むと、順次B、C、Dが追い出されそれぞれのinvalid_ageが記録されます。そしてもう一度Aが読み込まれるところまで来ると、まずEが追い出され invalid_ageがインクリメントされます。そして、ここでAの<refault distance> = 5-1 = 4が計算されます。これがactiveリストの長さ = 4以下なので、Aはacitveリストへと追加されます。この後、B、C、D、Eと読み込んでいくことでactiveリストに居座っていたW、X、Y、Zはすっかりactiveリストから追い出されてしまいます。

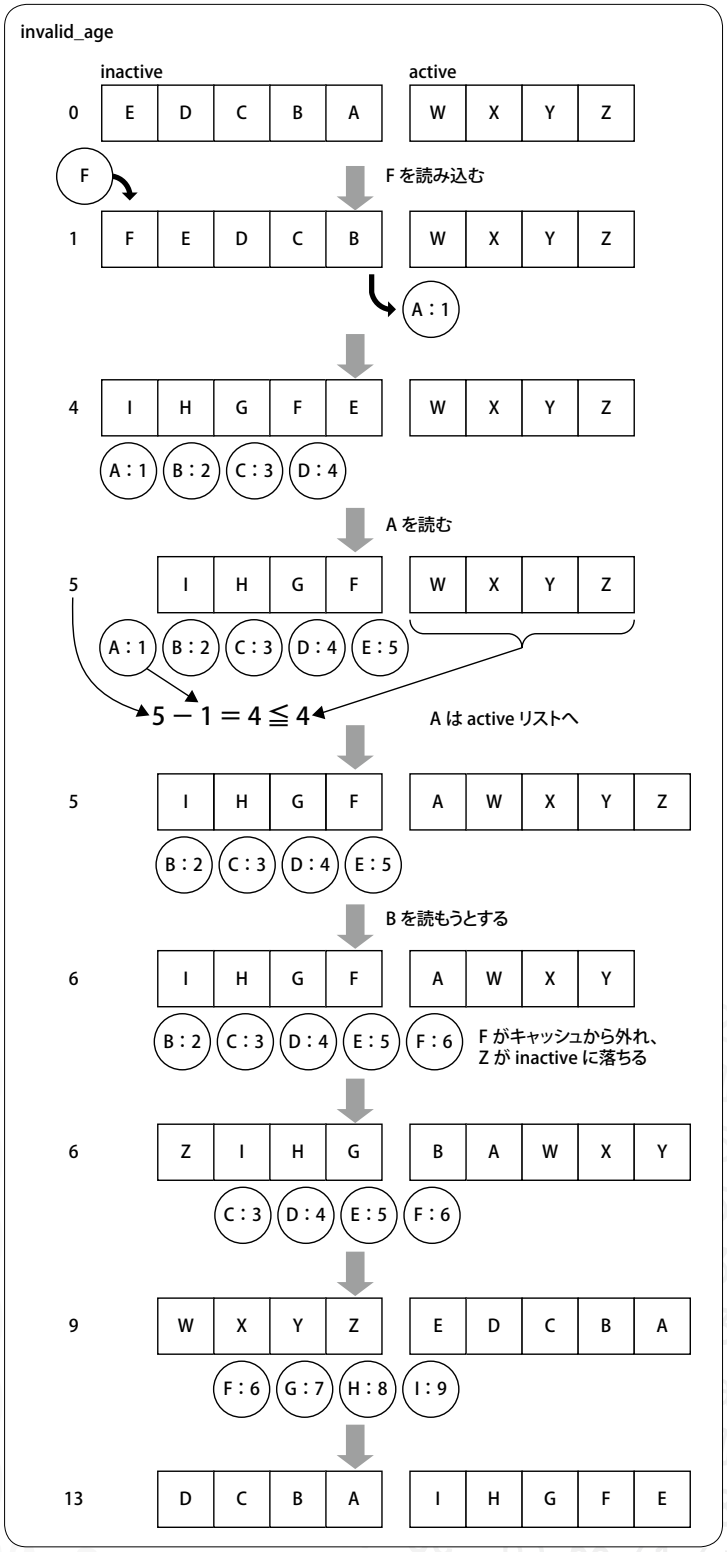
この次にFが読み込まれる時点で、Zがキャッシュからも追い出されていきます。最終的にAからIがすべてキャッシュに乗ってしまい、ディスクから読みにいかない高速な作業が可能となります。

PM QoS システムの制限

次にLinux 3.15で機能が拡張されたPM QoSフレームワークについて解説します。PM QoSはドライバなどが望むレイテンシ、スループットといったパフォーマンス上の制限をとりまとめて、制限内容をカーネルとユーザ空間とに見せるためのフレームワークです。

具体的な例としてCPUについて見てみましょう。以前にも紹介したように、現在のCPUには一口にアイドル状態といっても複数のアイドル

▼図2　動作例



状態が存在します。これは一般に、C0、C1、C2、C3というようにCに番号を付けた名前で呼ばれます。番号が大きくなるほどその状態において消費する電力は減っていきますが、同時に実行可能状態に戻るまでの時間(レイテンシ)も大きくなっていきます。そうすると、一度やることがなくなったからといってすぐに一番深い(番号の大きい)C-stateに入るのは得策ではありません。現在の稼働状況によって小さな消費電力と大きなレイテンシがつりあうC-stateを選択する必要があります。

稼働状況と同様にデバイスやアプリケーションから稼働状態に戻るまでのレイテンシに制限が付けられる場合もあります。たとえばdrivers/media/platform/via-camera.cでは**リスト1**のようにレイテンシに制限をかけています。C3以上深いC-stateに入るとでDMA転送がcorruptしてしまうので、アイドル状態になっても高々C2までになるようにレイテンシが50マイクロ秒(μsec)以上になるC-stateには入らないような設定をしています。

またユーザ空間でリアルタイムアプリケーションを動かしている場合、CPUがアイドル状態に入ってしまうと実行すべきイベントが起きたときからアプリケーションが動

作できるようになるまでに、C-stateから戻ってくる時間が余計にかかってしまいます。これを防止する1つの方法は、カーネルの起動パラメータにprocessor.max_cstates=1を付けることです。

しかし、この方法では常に(リアルタイムアプリケーションが動作していないときにも)深いC-stateに入れなくなってしまいます。ここでPM QoSフレームワークのユーザランドからのインターフェースを利用できます。/dev/cpu_dma_latency を開き32bitの数値(単位はマイクロ秒)を書くか、または"0x12345678"といった10文字の16進数文字列を書くことで、**ファイルを開いている間だけ**稼働状態に戻るまでのレイテンシが設定した値以上になるC-stateに入らないようになります。

さて、今度はこうして設定された制限がどのように使われるかについて見てみましょう。現在の制限値を取得するには"pm_qos_request()"関数に対応する引数を渡します。つまり、CPUのレイテンシの場合"pm_qos_request(PM_QOS_CPU_DMA_LATENCY)"と呼び出します。この関数は、現在登録されている制限のうち一番厳しいもの、すなわちレイテンシの場合には一番小さいものが返ってきます。

CPUのドライバはこれを使って移行するC-stateを決定します。たとえば、drivers/cpuidle/governors/ladder.c の ladder_select_state関数などで、この情報が使われています。また、制限の値が変わったときに、変更された通知を受けることもできます。たとえばdrivers/cpuidle/cpuidle.cではcpuidle_latency_notify()関数で通知を受け取り、残りのCPUをC-stateから起こす処理を行います。

また、ユーザランドからも/dev/cpu_dma_latencyを読み出すことで現在の制限値を取り出すことができます。例として/dev/cpu_dma_latencyの読み書きを見てみましょう。**図3**の実行例では、まず/dev/cpu_dma_latencyを読みこみんでいます。この例の場合レイテンシの制限は"0x000b71b0"(= 750000 μ sec = 0.75秒)と設定されています。デフォルトで設定されている場合はより大きく2,000秒となっています。筆者の例の場合では、音楽を流しているのでサウンドドライバがこの値を設定しています。ここで現在設定されている値よりも、より小さい値を書きこみ、書き込みに使ったファイルを3秒間開いておきます。同時に1秒後に、値を読み出すようにします。すると、0x00001234が読み出され確かに一番小さい値が使われているとわかります。最後にwaitして、書き込みに用いたファイルが閉じてから、再度読み出してみると、値が元に戻っていることも確認できます。

このようにPM QoSフレームワークの**1つの機能**は、ドライバやユーザランドのプログラムなどさまざまな場所から設定されるレイテンシなどの制限を整理し、その制限の情報を使いたいドライバに最も厳しい制限値を伝える機能となっています(**図4**)。今回の例ではCPUのレイテンシを取り上げましたが、ほかにもネットワークのレイテンシとネットワークのスループットの制限も設定できます。これらの制限値はユーザランドからは、それぞれ/dev/network_latencyと/dev/network_throughputからアクセスできます。ネットワークレイテンシの制限値は無線LANのドライバで実際に使われているようですが、ネットワークスループットのほうは制限の設定は行われているもののその値を使っている個所は今のところないようです。

▼リスト1　via-camera.c内のCPUレイテンシ制限例

```
/*
 * If the CPU goes into C3, the DMA transfer gets corrupted and
 * users start filing unsightly bug reports.  Put in a "latency"
 * requirement which will keep the CPU out of the deeper sleep
 * states.
 */
pm_qos_add_request(&cam->qos_request, PM_QOS_CPU_DMA_LATENCY, 50);
```

PM QoS デバイスの制限

ここまでに紹介したPM QoSは、CPUやネットワークなどシステム全体で1つの制限をかけてきました。最近の周辺機器の中には電源管理機能を持ち、実行時にサスペンドする機能を持ったものもあります。これらのデバイスをサスペンドにするかどうかをデバイスごとに制御できるようにするのがPM QoSのもう1つの機能です。

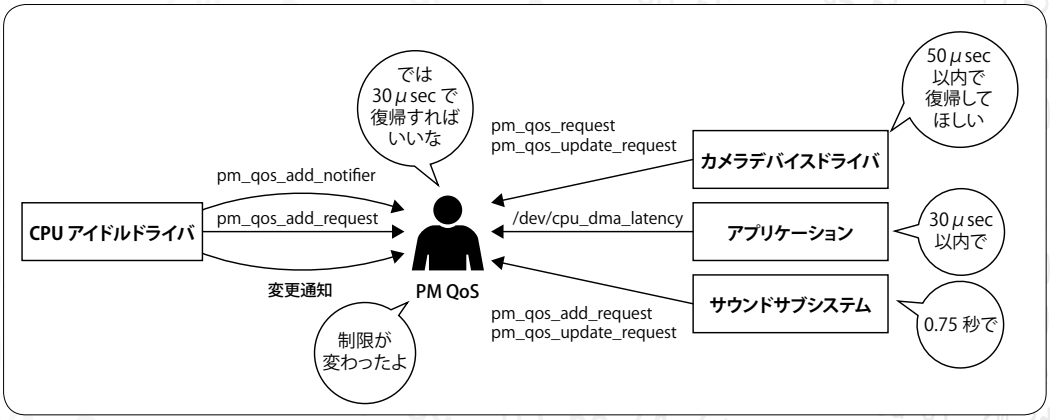
デバイスドライバに設定できる制限はCPUの場合と同じ復帰までのレイテンシと、フラグを設定できます。レイテンシのほうはCPUとまったく同じで、設定されたレイテンシよりも長い復帰時間になるサスペンドモード(D-state)に入らないようにドライバが動作します。フラグにはNO_POWER_OFFとREMOTE_WAKEUPのフラグとがあります。NO_POWER_OFFフラグを付けていると、カーネルドライバは、デバイスから完全に電源を取り去ってしまわないように動作します。

たとえば、drivers/acpi/device_pm.cのacpi_pm_device_sleep_state()関数では**リスト2**のようにNO_POWER_OFFフラグをチェックしています。デバイスの電源モードにはD0からD3まであり、さらにD3はD3hotとD3coldに分かれています。D3coldであれば、完全に電源が切れてしまうので、NO_POWER_OFFフラグでD3coldに入れるかどうかを制御しているということになります。もう1つのフラグ、REMOTE_WAKEUPフラグは、そのデバイスのremote wakeupを有効にするかどうかを設定します。remote wakeupは外部からのイベント(たとえばキーボードのキーを押すなど)に反応してデバイス自体のサスペンドを解除したり、あるいはマ

▼図3　/dev/cpu_dma_latencyの読み書き

```
$ sudo hexdump -C /dev/cpu_dma_latency
00000000  b0 71 0b 00                                       |.q..|
00000004
$ (echo 0x00001234; sleep 3)|sudo tee /dev/cpu_dma_latency& sleep 1; sudo hexdump -C /dev/
cpu_dma_latency
[1] 22120
0x00001234
00000000  34 12 00 00                                       |4...|
00000004
$ wait; sudo hexdump -C /dev/cpu_dma_latency
[1]+  Done                    ( echo 0x00001234; sleep 3 ) | sudo tee /dev/cpu_dma_latency
00000000  b0 71 0b 00                                       |.q..|
00000004
```

▼図4　PM QoSフレームワーク



シンのサスペンドから復帰することを可能にする機能です。

これらの制限は、ユーザ空間ではレイテンシはsysfsの/sys/devices以下のpower/pm_qos_resume_latency_usファイル、NO_POWER_OFFフラグは同ディレクトリのpm_qos_no_power_offファイル、REMOTE_WAKEUPフラグは同pm_qos_remote_wakeupファイルで制御できます。またカーネル内からも、dev_pm_qos_add_request()、dev_pm_qos_update_request()によって制御できます。たとえば、drivers/mtd/nand/sh_flctl.cのflctl_select_chip()関数では**リスト3**のように、レイテンシの制限を100マイクロ秒に制限しています。

PM QoS デバイス内での電源管理

さて、ここまでのPM QoSフレームワークはすべてカーネルドライバに制限内容を伝えるために使われるものでした。しかし、最近ではデバイスが自分自身で(ドライバの助けを借りずに)自身の電源状態を制御するような周辺機器も出てきています。これらのデバイスはアクセス状況によって自動的にスリープ状態に入ります。そうすると、ここにもまた「スリープしてほしくないときにスリープしてしまう」問題が出てきます。この問題に対処するために、Linux 3.15では、PM QoSを拡張し、デバイス側にレイテンシの制限を伝え、デバイスがそのレイテンシを考慮してスリープ状態への遷移を判断できるようにしました。

このレイテンシの設定が可能なデバイスに対応する/sys/devices以下のディレクトリのpower/下には、pm_qos_latency_tolerance_usファイルが作られます。このファイルに許容される最大のレイテンシの値、または"any"か"auto"を書きこみます。"any"の場合にはとくにレイテンシの制限は設定しませんが、ハードウェアが自動的に許容レイテンシを判断することは許可しません。一方で、"auto"の場合は完全にハードウェアが必要に応じて許容レイテンシを自動的に設定することを許可します。

この機能をサポートしているデバイスのドライバは、dev_pm_info構造体のset_latency_toleranceにコールバック関数を設定します。今のところACPILPSS(Low-Power Subsystem)ドライバであるdrivers/acpi/acpi_lpss.cのacpi_lpss_set_ltr()関数だけがこのコールバック関数として実装されています。SD

▼リスト2　NO_POWER_OFFのチェック

```
if (d_max_in > ACPI_STATE_D3_HOT) {
    enum pm_qos_flags_status stat;

    stat = dev_pm_qos_flags(dev, PM_QOS_FLAG_NO_POWER_OFF);
    if (stat == PM_QOS_FLAGS_ALL)
        d_max_in = ACPI_STATE_D3_HOT;
}
```

▼リスト3　カーネル内からデバイスの復帰レイテンシを設定

```
if (!flctl->qos_request) {
    ret = dev_pm_qos_add_request(&flctl->pdev->dev,
                    &flctl->pm_qos,
                    DEV_PM_QOS_RESUME_LATENCY,
                    100);
    if (ret < 0)
        dev_err(&flctl->pdev->dev,
            "PM QoS request failed: %d n", ret);
    flctl->qos_request = 1;
}
```

August 2014

NO.34

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

はじめてUNIXで仕事をする人に教えたいこと

6月にjusとしては久しぶりの勉強会を実施しました。その模様をお伝えします。

2014年6月jus勉強会

■はじめてUNIXで仕事をする人に教えたいこと
【講師】木本 雅彦（㈱創夢）、松山 直道（㈱創夢）、稲島 大輔（㈱創夢）、鈴木 嘉平（㈱KADOKAWA）、臼田 良寛（㈱KADOKAWA）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2014年6月4日（水）19:00～22:00
【会場】角川第3本社ビル 14F会議室ハワイ

今年の春に出版された『はじめてUNIXで仕事をする人が読む本』（**図1**）を題材に、これからUNIXで仕事をする人に何を勉強してもらいたいか、どんな教え方をすればいいのかを一緒に考えるという主旨のもとに開催されました。参加者は33人でした。

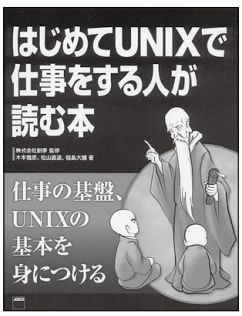


図1　木本雅彦、松山直道、稲島大輔 著、㈱創夢 監修『はじめてUNIXで仕事をする人が読む本』KADOKAWA、2014年

■本書の制作経緯

勉強会の第1部は「『はじめてUNIXで仕事をする人が読む本』ができるまで」と題し、本書の著者と編集者に制作過程や裏話を語っていただきました。

創夢の社内研修を担当していた木本さんが、ベテランと若者のUNIXユーザの間で世代の違いを感じるようになり、それを埋めるような書籍を作りたいということで臼田さんに企画書を送ったのが制作のきっかけです。創夢は社内研修でUNIXを教えているので、その資料をもとに執筆が始まりました。最初は20人ぐらいで書いていたのですが、人数が多過ぎてうまく進まず、仕切りなおしをする段階で今回の講師3人による執筆に切り替えたそうです。また、編集側も、企画段階では臼田さんが担当していたのですが途中で異動になってしまい、その後は鈴木さんが進行管理を務めました。

■話題の選択と分担

収録する話題の選択は、おもに木本さんの手で行われました。その際に重視したのは、長く使える普遍的な話題を選ぶことです。その結果、本書は、「コマンドラインの操作を中心に解説する」「カスタマイズについては触れない」「多くのUNIX系OSで共通に使えるテクニックを中心に説明する」といった特徴を持っています。なお、実行例などで使われているOSはFreeBSDとUbuntuです。創夢にはSolarisやNetBSDのユーザも多いのですが、世間のユーザ数や、これから勉強する人のための書籍であることも考慮して木本さんが選びました。

執筆の分担は、第1部・生活環境編を木本さん、第2部・プログラミング環境編を稲島さん、第3部・ネットワーク技術編を松山さんが担当しました。このような構成になっているのは、創夢では開発系の部署と運用系の部署があり、生活環境編は全員が受講しますが、その後は配属先に応じた内容を学ぶというカリキュラムに起因します。また、編集担当の裏話

として、ページ数の割に校正の回数が多く、原稿の確定に時間がかかったのですが、良い書籍を作るには必要なことと考えてアスキー[注1]社内の説得にあたったという話がありました。それから、本書では一貫して、「UNIX/Linux」などと書かずに「UNIX」で表記を統一しています。この点について著者陣からは「特定のOSの解説書ではなく、UNIX系OSで共通に使える技術の解説書であることを示したかったから」という説明がありました。

■内容の紹介

この後は本書の目次を見ながら内容を紹介していきました。まず仙人と弟子のイラストが描かれた表紙ですが、これはかつてアスキーから出版された『The Art of UNIX Programming』(**図2**)の表紙にインスパイアされたものです。それから、臼田さんから企画時の目次案が提示され、制作の過程で章立てが大きく変化したことが示されました。とくに生活環境編は仕切りなおしをするときに大部分を書き直しました。生活環境編で注目したいのはエディタの章で、vi/vimやEmacsも書いてありますが、最初にedの解説があり、またエディタが使えないときにechoやcatでファイルを操作する話も記述されています。プログラミング環境編は研修資料がかなり使えたのですが、研修ではSolarisを使っているところを本書ではFreeBSDやUbuntuで説明しているので、実行例をすべて取りなおすのがたいへんでした。ネットワーク技術編は、OSIモデルなど基礎的な概念の説明が研修資料にないので書き下ろしました。

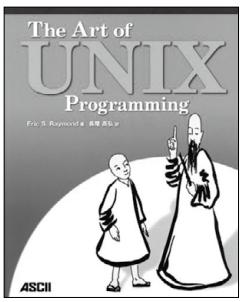


図2 Eric S.Raymond 著、長尾高弘 訳『The Art of UNIX Programming』アスキー、2007年

講師陣から、「新人が直接手に取るのでなく指導者経由でもいいから本書が広まってほしい」「創夢でやってきたことの成果を書籍という形で残せてよかった」「アスキーでは古くから『たのしいUNIX』[注2]などUNIXの入門書を出してきたが、久しぶりにその類の本を作ることができてうれしい」などの言葉をいただいて終了となりました。

■BoF

第2部は、ビールとピザをつまみながらBoF形式でトークを進めました。その中の一部を紹介します。

- 近年、若い開発者たちがMacBookなどを購入し、UNIXユーザになる例が急増しています。しかし彼らはこれまでGUIしか使ってこなかったせいか、CUIの操作をこわがっているようです。また、教科書の操作例をプロンプトも含めて入力したり、ヒストリー機能を知らないのでコマンドを毎回全部打ちなおしたりする例が散見され、UNIXの操作を体系的に教わっていない人が多いようです
- SIerでは、先輩もUNIXを知らないので、新人はUNIXを教えてもらえず、上達しないと言います。昔は同じマシンを使っている他人の設定ファイルを見て勉強できたのですが、今は各自でマシンを所有して設定ファイルを共有しないので、そういう勉強のしかたができません。やはり周囲にUNIXユーザの先輩がいるのが最高の教材のようです
- 最近はシェル芸勉強会など、シェルをテーマとする勉強会が行われ、ある程度の参加者を集めています。UNIXを使い始めた人たちの間でシェルに対する関心が高まっていることがうかがえます。本書ではコマンドを組み合わせて処理するようなシェルプログラミングの考え方には触れていませんが、そこを教えてほしいという声がありました

◆ ◆ ◆

このような話を笑いも交えて繰り広げるうちに、予定の時間を大幅にオーバーして終了となりました。jusならではのテーマで勉強会を行うことができて非常に有意義であったと思います。SD

注1) 本書はKADOKAWAから発行されていますが、企画／制作は㈱アスキー・メディアワークスのもとで始まりました。その後の2013年9月に、アスキー・メディアワークスは㈱角川書店などと合併し、KADOKAWAとなりました。

注2) 坂本文 著『たのしいUNIX——UNIXへの招待』アスキー、1990年

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
小泉　勝志郎 Katsushiro Koizumi
Twitter @koi_zoom1

第32回 島ソン!　電波も届かない離島でのハッカソン

“東日本大震災に対し、自分たちの開発スキルを役立てたい”というエンジニアの声をもとに発足された「Hack For Japan」。今回は宮城県の浦戸諸島で行われたハッカソンの報告です。

島ソンとは？

Code for Shiogamaは東日本大震災で被災した宮城県塩竈市の地域活性をITの力で行っていこうという団体です[注1]。震災復興団体とIT技術者がお互いに意見を出し合うミーティングの開催をしています。今回はCode for Shiogama/Hack For Japanの主催で2014年5月24～25日に行った「第1回　島ソン～浦戸諸島ハッカソン～[注2]」の模様について紹介します（**写真1**）。

開催のきっかけ

まだ震災から間もない2011年7月にHack For Japanでは、今回の島ソンの舞台である宮城県塩竈市の浦戸諸島を視察しました。離島であるため重機が入りづらく、震災の爪痕が強く残る中での視察[注3]でした（**写真2**）。その視察からほぼ3年が経過し、「うらとラウンジ菜の花」という人が集まるための場所が浦戸諸島開発総合センター（ブルーセンター）という研修センター内に作られました。

筆者は塩竈市の出身です。家が浦戸諸島にあるわけではありませんが、同じ塩竈で震災の被害の大きかった浦戸諸島にITの力でより貢献したいと以前から思っていました。そして、人が集まれる場所ができたということで今回の島ソンを企画しました。

島ソンのコンセプト

今まで筆者はHack For Japanが主催するものをはじめとしたHackathonに多く参加し、スタッフとして運営も多く行ってきました。そこで感じていたのが、①このプロダクトは本当に役に立つのか、②プログラムを組めない人が暇になる、③どんな良いプロダクトでもHackathonが終わるとそれで終息してしまう、ということです。

①を解消するために、島に住む方・島で仕事をしている方にイベントに参加してもらうことにしました。一緒にアイディアを出しあい、一緒にプロジェクトとして進めていきました。②に対しては、ガイドと一緒に島を歩いて見てもらい、島の実際を知ってもらうというイベントを入れました。

このような対策を取っているHackathonは多いです。しかし、③への対策には多くのHackathonが取り組みを見せつつも、あまりうまく機能しているように見えているものは個人的にはありませんでした。今回、ここを解決するために石巻専修大学の舛井研究室とタッグを組み、「学生のみなさんがこのイベントを通じて自分の名刺代わりとなるコンテンツを作る」という方法を取りました。舛井研究室のみなさんは経営学部なのでプログラム経験のある方はわずかです。しかし、「自分の名刺代わり」にするためにはHackathonが終わった後も活動を続けなければならない。そして、何よりこれが学生のみなさんにも就職で自分をアピールしやすい材料になります。この流れでHackathonの後も活動を続けようという試みです。

Hackは随所に！

Hackathonは野々島にある市の出先機関の浦戸諸島開発総合センターにて行われました。センターには宿泊ができる大広間があり、キッチンや体育館も

注1　https://www.facebook.com/CodeForShiogama
注2　http://tohoku-dev.jp/modules/eguide/event.php?eid=255
注3　2011年7月の浦戸諸島視察の模様　http://blog.hack4.jp/2011/07/hack-for-japan_13.html

第32回 島ソン！　電波も届かない離島でのハッカソン

備わっている施設で1泊2日のHackathonをするにはうってつけの場所でした。

が……離島だけあって電波の入りに少々難が。とはいえ、そこはハッカーのみなさん。なんとステンレスボウルを駆使して簡易パラボラアンテナを作成！（**写真3**）　Wi-Fiがぜんぜん電波を拾わなかったのが、このおかげで動画を見られるくらいにまでに大変化が！

また、今回のHackathonでは夕食・昼食は自炊！　ここでも魚捌きHackや夕食の残りを朝食のおかずにトランスフォームするなどのHackが繰り広げられました。

◆写真1　島ソンボード

◆写真2　震災当時の浦戸諸島

◆写真3　ステンレスボウルアンテナ

◆写真4　ディスカッションの様子

ディスカッションからスタート

島のみなさんから話を聞く

今回の島ソンでは、仙台はもちろんのこと、東京や福島、遠くは大阪からも参加が！　そして前述のとおり、石巻専修大学からの学生と島の方も含めて総参加者数はなんと47名。

参加者の全員が浦戸諸島の現状について詳しいわけではないため、島の方の話をもとにいくつかのワークショップを織り交ぜてHackathonに取り組む、というプログラムを運営スタッフの中西さんに構成いただきました。

島からの参加者は浦戸諸島で漁師をされている方、石浜区長、海産物販売をされている方などから島の現状について話をしてもらいます。区長というのは東京23区とは異なり、一般で言う町内会長に相当して行政区分の長ではありません。しかし、浦戸諸島には自治区という扱いで通常の町内会長とは異なる大きな影響力があります。

Hackathon初日は、まず島の方の話をうかがうことからスタート。このディスカッションでは「後継者問題」、「震災で海の生態バランスが崩れ漁がしづらくなった」、「島の年齢層が高く保守的なところがある」、「島と本島の塩竈を行き来する汽船の本数が少なく、島に住みながら塩竈や仙台で働くことができない」、「救急で人が倒れても救急船が来るまでに1、2時間かかる」といった話がされました（**写真4**）。

島の方から話を聞いた後、参加者同士で感想や疑問点を話し合い、さらに島の方への質問タイムを挟んで昼食へ。昼食は「島巡り弁当」という浦戸諸島の各島（桂島・寒風沢島・野々島・朴島）の味がすべて楽しめる豪華なお弁当です。

島のみなさんとアイディアを膨らませる

午後からは話し合いたいテーマを各自が発表。漁業体験、島を利用した脱出ゲーム、浦戸諸島で採れる健康にいい海藻アカモクのPRなど、たくさんのテーマが提案されました。

そこから似たテーマを出した者や連携できそうな者同士がチームを構成して、いくつかのグループ分けを行いました。それぞれのチームでマインドマップを使って、さらにテーマを関連キーワードで広げながら掘り下げていきます（**写真5**）。この中でも目立っていたのが、島の方がふと発言したほら穴についての発言が広がりを見せて生まれた、ほら穴グループ。なんと婚活グループと結びついて議論とい

う奇妙な光景も。

マインドマップを行った後、自分がHackathonでやってみたいアイデアを各々が用紙に書き込みプレゼン。気に入ったアイデアに参加者を募ってHackathonに取り組むグループ分けがなされました。

手を動かすだけではなく足も動かす！

いざ島あるきへ！

通常のHackathonでは、グループ分けの後は各グループの作業となりますが、今回は先に述べたように島の方に島内を案内いただいての島あるき！

複数の斑に分かれての島あるきでは話題になったほら穴を見て回ったり、アカモクを取る現場を実際に見たり、船に乗ったりとグループによってさまざまな体験を行えました。

実際に島について知る貴重な機会を得たあとは、お待ちかねの開発へ……と、すぐには行かずにまずは夕食へ。夕食は自炊なのでHackポイントです。港の街、塩竈の刺身をはじめとした美味しい海産物。そして、牡蠣やホタテのバーベキュー！　たっぷりおなかを満たして果たして開発はできるのか？

開発作業！

食事が終わってしばらくは満腹感に満たされていましたが、そこはHackerのみなさん。夜も更けるころにはみなさん真剣な表情での開発が始まりました。PCでの作業だけでなく、素材となるムービーや写真を撮りに行ったりするなど、現地にいながらにして開催されるHackathonならではのやり方にチャレンジしているチームも見られました。

そして成果発表へ

それぞれのチームのテーマと発表内容を紹介します。

❶「ほら穴」

野々島に多数存在している「ほら穴」に着眼。廃墟マニアなどが存在していることから、ほら穴マニアもきっといるという、ほら穴を観光資源として活用するアイデア。名付けて「ほら穴るるぶ」！　実際のほら穴を使って撮影したPRムービー、旅行者と島の方がほら穴画像の投稿を通じて交流が図れるSNSサイトを提案し、スマホ閲覧機能付きのサイトのデモも行われました。

❷「謎解きゲーム」

観光資源はあるのに訪れる人が少ないという課題を受けて、複数の島から成り立つ浦戸諸島そのものに注目。謎解きゲームを開催して、若い人に浦戸諸島を知ってもらうきっかけイベントとしてはどうかというアイデア。リアル脱出ゲームの波にうまく乗れるとおもしろくなりそうです。

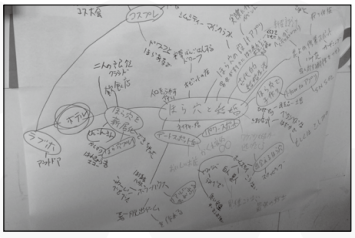
❸「写真」

島の紹介がされているガイドマップの情報量が少ない、見落とされているスポットが多いなどの課題があるので、それらを解決するアイデアを提案。住民の方から仕入れたネタを入れた解説付きのガイドアプリ、旅行者と島の方が写真を通じてSNSで交流できる写真アプリ、THETAを使った360度の風景スポットを見られるストリートビュー[注4]を発表しました。

❹「レシピ」

「島の食材＋マッシュアップ」という発想から

◆写真5　マインドマップ



◆写真6　ほら穴見学

◆写真7　船で島巡り

注4　野々島ストリートビュー　https://plus.google.com/photos/107760690310294960125/albums/6017705989741573009

「島っしゅ。」と名付けられたアプリを披露。思いついた食材を入れてスマホを振ると浦戸諸島で採れる食材と組み合わせたレシピが表示され、表示された浦戸諸島の食材が購入までできるようにサポートされています。

❺「養殖(育てる)」

島の方の「島の物産を身近なものとしていろいろな人に知ってほしい」という想いをもとに、養殖をイメージした育成ゲームを想定していたものを、ターゲットや内容のハードルの高さから方向を変えて、海産物が購入できる「海の子ネットオンラインショップ」を多くの人に広めていくアイデアを提案。おねだり機能を実装して、おねだりコメントがFacebookに投稿されるといった「一緒に食べよ」というサービスを発表しました。

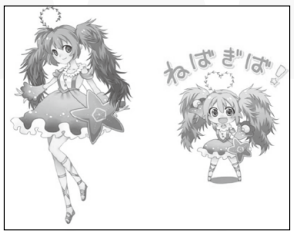
❻「アカモク」

Facebookページ、Twitterボット、NAVERまとめ、萌えキャラといったツールを使って浦戸諸島で採れる海藻アカモクをPRするアイデアをそれぞれ実際に作成して披露[注5]。萌えキャラは「渚の妖精ぎばさちゃん(ギバサはアカモクの別名)」としてクラウドソーシングサービスにてコンペでイラストを募集。イベント終了後の6月16日には総応募数31点とかなりの人気案件に。今回は**図1**のイラストを採用しました。

❼「新しい交流文化」

浦戸諸島について知らない人が多く、まずは興味を持ってもらうことから始めるために聖地化するというアイデアを提案。浦戸諸島をモチーフにした「Island Girls」というキャラを考えました。アドベンチャーゲーム、チャットといったキャラを使ったバーチャルなコミュニケーションを手始めに、浦戸諸島を訪れるとARデートができるプランを発表し、これらのイメージPVを作成しました[注6]。

◆図1 ぎばさちゃん



◆ ◆ ◆

上記7つのアイデアを発表して島ソンは終了となりました。今回の島ソンでは島の方に話題提供者となって参加いただくだけでなく、マインドマップ作成や島見学などに協力してもらったことで、一緒に取り組んでいくHackathonとなりました。

イベントを終えて

島ソンは筆者からの次の言葉で締めました。

「こうやって地方でハッカソンなどイベントを起こしても、イベントが終わったらそれで終わりというケースが多いです。ここでハッカソンをしたことにもっと意味を持たせるためにも、今回ここで出たアイディアや作ったものを、これからも継続していきましょう。」

そう。イベントが終わったら終わりではなくここからが始まりなのです!

謎解きチームは実際にイベントの企画を始めPVを制作しました。新しい交流文化のチームはプロジェクトI.G.として毎週集まってゲームを作り始めています。アカモクチームもぎばさちゃんのデザインのほか、Facebookページにアカモクの販売促進のためのコネクトを作るなど精力的に活動が行われています。これからほかのチームでも新しい成果が続々出てくるでしょう。島ソンはHackathonだけでは終わらないHackathonなのです。

最後に今回のHackathonでも多くの方々に協力をいただきました。島の方や参加いただいた方はもちろんのこと、食事のサポート、事前準備に時間を割いてくださった方など、関係くださった皆さまにあらためて感謝いたします。

最後に

Code for Shiogamaでは、復興に携わっている方とIT技術者が議論することで生まれる化学反応を生み出したいと思っています。今後も定期的に開催していきますのでよろしくお願いいたします。SD

注5 アカモクbot https://twitter.com/akamokubot
注6 イメージPV https://www.youtube.com/watch?v=j_2VGTFmZlA

温故知新
IT むかしばなし

第36回

カセットテープとデータレコーダ

LINE株式会社　佐野 裕　SANO Yutaka　sanonosa@gmail.com

はじめに

今回は1980年前後に外部記録媒体として用いられていたカセットテープ[注1]とデータレコーダについてお話します。

カセットテープとは

カセットテープは音声や音楽を録音・再生するために使われているメディアです。オランダのフィリップス社が互換性厳守を条件に基本特許を無償公開したことにより標準化が進み、世界中で広く普及しました。

プラスチックのケースに2つの軸が入っていて、片側に巻かれた感じでテープが入り、もう片方に巻き取られるときに、外に露出したテープ部分に磁気ヘッドが当たり、録音と再生ができるしくみです。テープが長いほど長時間録音ができ、市販されているものとしては30分、46分、60分、90分、120分などの種類があります。カセットテープには表面と裏面があり、ひっくり返すことで両面に記録ができます。たとえば録音時間が60分のカセットテープの場合は各面30分ずつ録音できました[注2]。

データレコーダ

データレコーダとはカセットテープにデータを記録することのできる装置のことです。パソコンに既に内蔵されているものもあれば、メーカーから純正品として発売されたものもありました[注3]。また、市販の音楽用カセットレコーダ(俗称テレコ)を利用し、プラグをRECやPHONE端子に接続するようなタイプのものもありました。

アナログとデジタル

パソコンのデータを保存したカセットテープを音声として聴いてみると、「ピーーガラガラガラ」といったように聞こえるのは、実際にカセットテープにデータを保存したことがある人は誰しも思い浮かぶ光景[注4]でしょう。

カセットテープはアナログ(音声)で記録するメディアですが、データは0と1のデジタルデータであるため、録音したデータを読み書きする際は変復調というアナログデータとデジタルデータの変換を行う必要があります。

アナログで0と1を表現するために1,200Hzと2,400Hzの矩形波の並びで記録します。たとえば一定時間内に1,200Hzが2回続けば0、2,400Hzが4回続けば1といった具合[注5]です。

転送速度

カセットテープは非常に遅い外部記憶方法でした。1つのプログラムを読み込むのに短いプログラムで数分、長いプログラ

注1) カセットテープは、テープがケースに入った状態のメディアの総称ですが、今回は狭義のコンパクトカセットのことを指しています。

注2) テープの素材の違いで、クロムやメタルなどの高音質で価格の高いものもありました。また、長時間録音できるテープが長いものは、絡みやすく、ワカメ状になってしまうこともしばしば起こりました。

注3) 中には早送りや巻き戻し、頭出しを自動的に行う高機能なものもありました。

注4) FAXの送受信音に近い感じです。

注5) 記録方式にはいろいろな種類があり、当時はカンサスシティスタンダードと呼ばれる方式が主流でした。また日本の人が考えたサッポロシティスタンダードという方式もありました。

ムで数十分かかるといったことがざらにありました。

当時のデータレコーダは300～2,700baud[注6]程度の性能がありました。つまり当時データレコーダは1秒間に300～1,200回、さらに性能の良いものだと2,400～2,700回変復調を行えました。仮に1baudあたり1bit処理されていたとすると、転送速度は300～2,700bit/秒、すなわち37.5～337.5byte/秒となります。最近のSATAインターフェースは6Gbpsですので、単純には比較できませんが1,200bit/秒と6,000,000,000bit/秒ということは、当時よりも転送速度が500万倍にもなっているのですね。

保存可能容量

カセットテープの保存可能容量は、録音可能時間をボーレートで掛ければ算出できます。

たとえば30分テープに1,200baud(仮に1,200bpsとする)で記録する場合は、2.16Mbit(≒270Kbyte)となります。270Kbyteと言えば最近の写真データ1枚すら収まらない容量です。

ランダムアクセスできない

カセットテープはハードディスクなどとは違い、1本のテープを最初から最後まで順に読んでいくタイプのメディアで、シーケンシャルアクセスしかできません。1本のテープに複数のプログラムやデータがある場合、自分でテープを該当のデータがある位置まで早送りもしくは巻き戻しするような使い方をします[注3]。

そのようにランダムアクセスを手動で行うのは効率が悪いので、理想的にはプログラムごとに違うカセットテープを使う方法が一番わかりやすいのですが、当時著者は小学生で、お金がなかったので1本のテープにそれこそ何十個ものプログラムを詰め込んだのは懐かしい思い出です。

ダビング

ダブルデッキラジカセと呼ばれる、音楽録音再生用途で使われていたカセットテープが2本入るタイプの装置を使うと、音声の入ったカセットテープを空のカセットテープにダビング(コピー)を簡単に行うことができました。当時パソコン用市販ソフトもカセットテープに入って売られていたため、ダビングを行うことで簡単に不正コピーできるということが問題となっていました[注7]。

2倍モード

通常の半分の回転速度に落とすことで、音質を犠牲にして2倍の時間録音できる機能を搭載したカセットテープレコーダがよく見られました。これは会議など、とくに音質が重視されない環境において有効な機能でしたが、ことパソコンのデータ記録用に限って言えば、読み込みエラー発生率が明らかに上がるため、よほどの理由がない限り使われませんでした。

テレビでプログラムを配信

当時パソコン専門のテレビ番組がありました。そこでは番組で紹介したプログラムなどのデータを副音声に乗せて流し、それを視聴者が、自分のテレコやラジカセで録音してからパソコンに読み込ませるなんてこともありました。

ただ、どんなに神経を使って録音しても、アナログの世界ではどうしてもノイズが入っていまい、うまく録音したつもりでもパソコンにうまく読み込めないといったことが頻繁に発生していました。

終わりに

さすがに現在では、パソコンの外部記録装置としてカセットテープを使っている人はいないと思われますが、音楽用メディアとしてのカセットテープの需要はまだまだ健在だとのこと。高齢者のカラオケ練習用や発展途上国での音楽販売など、需要は当面なくならないようです。

SD

注6) 1秒間に変復調を行う単位のことをbaud(ボー)という単位で表します。

注7) ただしダビングを行うと、ダビングされた側は信号レベル低下とノイズ混入の影響で、マスターテープよりもリードエラーが発生しやすくなるため、繰り返しダビングを行うとパソコンから読めなくなります。

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

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

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

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

Hardware

シリコンパワージャパン、OTG対応USBメモリ「Mobile X20」発売

シリコンパワージャパン㈱は5月9日、USBフラッシュメモリ「Mobile X20」を発売した。

本製品は、パソコンを介することなくUSB機器を相互に接続するための規格OTG（USB ON-The-Go）に対応している。本体にはMicro USBと標準USB両方を備えており、パソコンだけでなく、スマートフォン（Android）やタブレットに直接接続してデータのやりとりができる。さらに、キャップが本体とつながっているので、キャップをなくしてしまう心配もない。インターフェースは、USB 2.0 / Micro USB（USB1.1互換）で、本製品を介した充電はサポートしていない。

▲Mobile X20

▼Mobile X20 仕様

容量	8/16/32GB
インターフェース	USB2.0/Micro USB
対応 OS	Windows 8/7/Vista/XP、Mac OS 10.3 以上、Linux 2.6 以上
モバイル端末	Android4.1 以上、Windows Phone 8
保証期間	永久保証

CONTACT シリコンパワージャパン㈱
URL http://www.silicon-power.com/

Software

キヤノンITソリューションズ、「ESET Cyber Security」の新バージョンV6.0を提供開始

キヤノンITソリューションズ㈱は、2種類のMac OS X用セキュリティ製品、総合セキュリティ対策プログラム「ESET Cyber Security Pro」および、ウィルス・スパイウェア対策プログラム「ESET Cyber Security」の新バージョンV6.0を、6月17日より提供開始した。

V6.0では、フィッシング対策機能が、個人向けWindows用総合セキュリティ対策プログラム「ESET Smart Security」と同等の性能に強化された。ログイン用ユーザ名・パスワード、クレジットカード情報などの機密情報を盗もうとする、悪意のあるWebサイトへのアクセスを未然に防ぎ、パソコンを機密情報漏えいの脅威から保護できる。また、SNS保護機能である「ESET Social Media Scanner」が搭載され、投稿されたコメントなどの情報の安全性のチェックや、オンラインスキャンによってパソコンの検査を実施する。

▼動作環境

OS	OS X Mavericks / Mountain Lion / Lion /Snow Leopard（※ サーバ OS を除く）
CPU	インテルプロセッサ（※PowerPC は対象外）
メモリ	512MB 以上
ハードディスク	150MB 以上の空き容量

CONTACT キヤノンITソリューションズ㈱
URL http://www.canon-its.co.jp/

Service

ハートビーツ、サーバのセキュリティスキャンサービスを提供する子会社「ウォルティ」を発足

㈱ハートビーツは7月1日、子会社として「㈱ウォルティ」を発足させた。ウォルティはサーバサイドのセキュリティスキャン「Walti.io」をSaaSで提供する。

Walti.ioの特徴として次の3つが挙げられる。

- OSSのセキュリティスキャナを利用し、FW・ミドルウェア・Webアプリケーションなどのセキュリティチェックを簡単・安価に利用できる
- スキャンを定期的に自動化し、ITシステムのセキュリティ管理を継続できる
- JenkinsなどのCIツールとAPIを通じて連携できる

現在、Walti.ioのクローズド β のテストユーザが募集されており、以下のフォームから応募できる。

応募フォーム：http://goo.gl/w7zQPj

▼ウィルティ 会社概要

社名	株式会社ウォルティ（Walti, Inc.）
代表取締役 CEO	藤崎 正範
取締役 CTO	Michael H. Oshita
所在地	東京都新宿区新宿 1-8-4 JESCO 新宿御苑ビル 6F
ティザーサイト URL	http://walti.io

CONTACT ㈱ハートビーツ
URL http://heartbeats.jp/

Event　オープンソースカンファレンス2014 Kansai@Kyoto開催

8月1日、8月2日の2日間、京都リサーチパーク（京都府左京区）において、「オープンソースカンファレンス2014 Kansai@Kyoto」が開催される。

オープンソースカンファレンスは2004年から全国各地で開かれている、オープンソース関連の開発者・コミュニティ・企業・団体が集う大型イベント。京都での開催は今年で8回目となっており、去年は2日間でのべ1,300名が来場した。

会場では、オープンソースコミュニティ・企業による展示が並び、最新技術に関する無料セミナーも開かれる。ローカル企画として、オープンソースの啓蒙と、若者・女性へのプログラミング教育の推進を目的とした「オープンソース入門塾」、オープンソースのハードウェアとソフトウェアを通して、コンピュータの原理が学べる「LilyPad Arduino の手芸作品展示」なども行われる。

また、広い展示スペースを、実行委員会のメンバが見学者と一緒に訪れる「展示ツアー」が実施されるので、初心者にもやさしいイベントとなっている。

会場では、スタンプラリーも開催される。受付で配布されるタイムテーブルには「スタンプラリー参加シート」が挟まれている。展示ブースに設置してあるスタンプを参加シートに集めることで、各種OSPN.JP特製グッズがもらえるとのこと（※無くなり次第終了）。

（スタンプ設置ブース）
Hinemos（NTTデータ）、Joe's クラウドコンピューティング、ほか

▼開催概要

日程	2014 年 8 月 1 日 (金) ・2 日 (土) 10:00 ～ 17:00 予定
会場	京都リサーチパーク（KRP）アトリウム・1号館4階 京都市下京区五条七本松通
費用	無料
内容	オープンソースに関する最新情報の提供 （展示） 　オープンソースコミュニティ、企業・団体による展示 （セミナー） 　オープンソースの最新情報を提供

CONTACT　SC2014 kansai@Kyoto 公式サイト
URL http://www.ospn.jp/osc2014-kyoto/

Report　Nginxユーザ会発足＆「NGINX Plus」発売

6月18日、オープンソースのWebサーバ「Nginx」のユーザ会が発足され、サイオステクノロジー㈱、ハートビーツ㈱の支援の下、第1回（#0）の会合がサイオステクノロジー東京オフィス（港区）で開催された。当日は、Nginx社のCEOであるGus Robertson氏、事業開発担当のAndrew Alexeev氏、そして開発者のIgor Sysoev氏が招かれた。

Gus Robertson氏は、Nginxの普及・発展に大きく寄与したというユーザコミュニティ活動の重要性を語り、日本のユーザ会発足に大きな期待を抱いていると述べた。Igor Sysoev氏は自身の来歴と、Nginxの誕生、今後の展開について語り、質疑応答の時間には来場者と技術的な意見を交わした。

サイオステクノロジー㈱は、6月17日に「Nginx Plus」の国内販売の代理店契約を締結している。ユーザ会の立ち上げによって、Nginxのすそ野を広げ、Nginxのエコシステムを発展させたいとのことだ。今後は、定期的な勉強会やエンジニアを招いたセミナーも展開していく予定である。

「NGINX Plus」はOSS版Nginxの機能を拡張し、Nginx社が2013年8月より米国で販売している商用製品。本製品はサブスクリプション契約によって提供され、エンタープライズ向けのサポートサービスが付加されている。おもな拡張機能は次のとおりである。

- **Health checks with NGINX Plus**
  管理者が広範囲の障害をチェックできる
- **Advanced cache control with NGINX Plus**
  キャッシュパージとキャッシュの状態を視覚化できる
- **Streaming media delivery with NGINX Plus**
  Apple HLSとAdobe HDSとVODアプリケーション用のアダプティブストリーミングをサポート

サイオステクノロジー㈱はNGINX Plusに日本語でのサポートを付加し、7月1日より販売している。契約は年間契約となり、価格は202,500円（税別）から。

CONTACT　日本Nginxユーザ会
URL http://nginx-ug.jp/
サイオステクノロジー㈱
URL http://www.sios.com/

Software

Infoblox、「効果的なSDNスイッチングを実現したLINCXを発表」

Infoblox㈱はDDIの世界市場の50%のシェアを持つ。DDIとは、DNS、DHCP、IPAM（IP Address Management）のことで、同社は独自技術（Gridテクノロジー）を実装しエンタープライズユーザ向けの製品として販売を行っている。日本では、ポッカサッポロフード&ビバレッジ㈱、青山学院大学などが汎用のDNS・DNSサーバでは対応できないインターネットからの攻撃に耐性が高いことから同社の製品を導入し、システム統合やセキュリティ面で効果を上げている。

同社創設者・CTOであるスチュワート・ベイリー氏と研究チームが、SDNコミュニティへの貢献として、関数型言語Erlangを使用したソフトウェアスイッチ「LINCX」を提供開始した。これはFlowFowarding.orgプロジェクトから無料ダウンロードできる。汎用的なPCとLinux、Xen HypervisorとLING（Xen上のErlang環境）があれば使用可能だ。

▲スチュワート・ベイリー氏（創設者・CTO）

CONTACT Infoblox㈱
URL http://www.infoblox.jp/

Service

データサルベージ、「ストレージ故障予測事前検知システム」サービスを立ち上げ

㈱データサルベージは、同社が開発した「MASAMUNE（マサムネ）」をフリーウェア化することで、ハードディスクなどの故障予測事前検知システムを構築し、新サービスとして提供することを6月24日に発表した。

「MASAMUNE」は、データ復旧やフォレンジックにおける証拠保全のための、ソフトウェア・デュプリケータ。ハードディスクの高速コピー、データ消去、ベンチマークの各機能などを持つ。使用時に対象ハードディスクの「S.M.A.R.T（Self-Monitoring, Analysis and Reporting Technolog）情報」が同社のサーバに送信される。集積するデータは、メーカ名、モデル名、不良セクタなどである。これらは匿名で管理され、障害予測分析のために利用される。ハードディスクの経年劣化による障害は避けられないが、サービスの利用者には有料で故障予測情報が提供される予定である。

▼「MASAMUNE」の詳細ページ
http://ja.masamune.com/

CONTACT ㈱データサルベージ
URL http://www.data-salvage.co.jp/

Software

日本マイクロソフト、プロダクション・アイジー、WebGLを活用した3D Webゲーム「翠星のガルガンティア～キミと届けるメッセージから」を無料公開

日本を代表するアニメーションスタジオ、㈱プロダクション・アイジーと日本マイクロソフト㈱が、Web標準技術であるHTML5のWebGL機能を活用した3DWebゲームを6月18日より無料公開した。

これは、アニメ「翠星のガルガンティア」をもとにしたスカイアクションゲームで、Webプログラマ向けにソースコードと3Dモデルが公開される（**http://fly.gargantia.jp/**）。オープンソースのTurbulenz Engine（**http://biz.turbulenz.com/**）を利用し、自分でゲームが開発できるようになる。ゲーム自体はInternet Explorer 11に最適化し、タブレット端末（Surface Pro2）でのタッチ操作にも対応。ソースコードもGitHubからダウンロードできる。これらはゲーム開発者育成サポートの一環として進めていくとのこと。


▲発表会には主人公役の声優・石川界人氏も参加

CONTACT 日本マイクロソフト㈱
URL http://www.microsoft.com/
㈱プロダクション・アイジー
URL http://www.production-ig.co.jp/

Service

Bitcasa、開発者とサービスプロバイダ向けのAPIサービス「CloudFS Platform」を発表

容量無制限クラウドストレージサービスを提供するBitcasa社は、6月23日に同社のクラウドストレージ技術を開発者が活用できるようにする新サービス「CloudFS Platform」をリリースし、同時に日本を拠点としてクラウドストレージを展開することを発表した。Bitcasaの特徴は、従来のネットワークストレージサービスに比べ、セキュリティの面とマルチデバイス対応で他のサービスと大きく差を広げていることにある。ストレージ中に500億のブロックに分散されたデータはユーザ側が持つセキュリティキーで管理され、その内容の秘密は完全に保護されている。本サービスは、Mac OS X、iPhone、Android、Windowsなど9つのプラットフォームに対応している。今回のAPIサービスCloudFS Platformは、さまざまなWebサービスに対応し、パブリッククラウドストレージ上のカスタムアプリケーション導入期間短縮を可能にするもの。

▲来日したブライアン・タペッチ氏（最高経営責任者・CEO）

CONTACT Bitcasa, Inc.
URL https://www.bitcasa.com/

Report

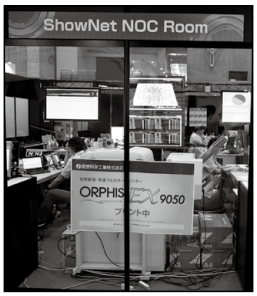
Interop Tokyo 2014開催

▲会場の様子

ネットワークインフラ・技術・製品を中心としたイベント「Interop Tokyo 2014」が開催された。今年は、カンファレンスが6月9日、10日にAP品川（東京都）で、展示会が6月11日～13日に幕張メッセ（千葉県）で行われた。21回目となる今年は、展示会に185の出展社が参加した。

Interop独自ネットワーク「Shownet」

「ShowNet」とはInterop会場で実際に構築される、相互接続検証ネットワーク。産業界、学会、研究機関から最新鋭の機器と技術、優秀なエンジニアが集まり構築される。このShowNetはイベント会期中、出展者や来場者がインターネットへ接続するためのISP、キャリアとして運用される。NOC（Network Operation Center）が会場の中心付近に設置され、ShowNetの運用を来場者が見学できるようになっている。センターの中では、ネットワークを監視するためのリアルトラフィック可視化ツール「NIRVANA改」が常時稼働し、センターの横にはインターネット接続やファイアウォールを実現するラックがずらっと並ぶ。ShowNetの解説ツアーでは、「今のインターネットはあと10年耐えられるだろうか!?」という疑問が強調され、ネットワークの仮想化やクラウド間イーサネット接続など、さまざまな新しい挑戦がなされていることが紹介された。今回のShowNetは、参加エンジニア数400名、提供機器・サービス総額約70億円の大規模なライブネットワークとなっている。



▲Shownet NOC

基調講演「モノと人をクラウドがつなげる」

Interopでは各界のエンジニアやエバンジェリストを招いた基調講演も行われる。

初日の6月11日に開かれた、アマゾンデータサービスジャパン㈱玉川憲氏による講演「モノと人をクラウドがつなげる」は、座席が満員、立ち見の聴講者も多く見られ盛況だった。講演の内容は、AWS（Amazon Web Service）の果たした役割、導入事例、今後の課題に関するものだった。玉川氏は「大規模なデータ解析やハイパフォーマンスコンピューティング、IOT（Internet of Things）など、一部の大企業しか実施できなかったサービスに、初期費用・運用コストが低いAWSを利用することで、中規模小規模のプレーヤが参入しやすくなった」と話した。また、異なる民族グループから1000人分の匿名ゲノムの配列決定を行う「1000人ゲノムプロジェクト」で得られた人間のゲノム情報を「AWS Public Data Set」で公開するといった導入事例も紹介された。最後に、AWSの今後の課題として、さらなる運用コストの低減を目指すと語った。

CONTACT Interop Tokyo 2014
URL http://www.interop.jp/2014/

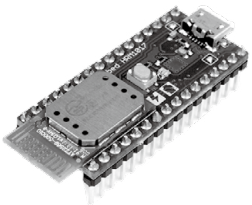
Hardware

スイッチサイエンス、ARM mbed対応開発ボード「mbed HRM1017」を発売

㈱スイッチサイエンスは、Bluetooth Low Energy（BLE）の通信が可能なARM mbed対応開発ボード「mbed HRM1017」を、7月に発売する。

本製品は、BLEによるBluetooth Smartデバイスを手軽に開発できるキット。ソフトウェアの開発はmbed環境で行う。パソコンからはUSBメモリとして認識されるので、ドラッグ＆ドロップするだけでマイコンにソフトウェアを書き込める。iBeaconのようなビーコン装置、無線センサ、IoTデバイスなどの開発に向いている。また、総務省の工事設計認証（いわゆる技適）を得たモジュールを搭載しており、日本国内でも合法的に使用できる。希望小売価格は5,400円（税込）となっている。

本製品は本誌連載「はんだづけカフェなう」（p.16～19）でも紹介しているので、試用例などはそちらを参照のこと。



▲mbed HRM1017

CONTACT ㈱スイッチサイエンス
URL http://www.switch-science.com/

Event

ユビキタスエンターテインメント、ハイパーテキスト絵本コンテスト開催　作品を募集中

㈱ユビキタスエンターテインメントは、同社製タブレット端末「enchantMOON S-II」を使ったハイパーテキスト絵本コンテストを開催する。作品の応募はすでに始まっており、締め切りは8月31日（日）の23時59分。

手書きでハイパーテキストコンテンツが簡単に作れるenchantMOON S-IIを使い、ビジュアルプログラミング機能などを活用したユニークな作品を募集している。作例はSkylab βで公開中（Webブラウザで閲覧可）。

▼作例『料理兵ソルトの大冒険』
http://skylab.enchantmoon.com/stickers/stack/4

▼コンテスト概要

応募期間	2014年6月23日～8月31日 23:59
審査日程	2014年9月中旬予定
結果発表	2014年10月上旬予定
賞金	10万円
応募方法	・enchantMOON 専用クラウドサービス Skylab β にコンテンツをアップロードする際に「ハイパー絵本コンテストに応募する」のチェックボックスをオンにする ・すでに投稿している作品を応募する際にも「Edit」画面で同チェックボックスをオンにすることでコンテスト参加可能 Skylab β　http://skylab.enchantmoon.com/

CONTACT ㈱ユビキタスエンターテインメント
URL http://www.uei.co.jp/

Service

サイボウズ、「脆弱性報奨金制度」を新設

サイボウズ㈱は、クラウドサービス「cybozu.com」上で動くサービスの脆弱性を発見・報告した人に報奨金を支払う「脆弱性報奨金制度」を6月19日に新設した。2014年2月から開始した常設の「脆弱性検証環境提供プログラム」でサービスの信頼性向上につながる報告が多数あったことを受け、今回の制度を新設したとのこと。また、同プログラムの開始以降、すでに報告があった37件の脆弱性に対しても、遡って報奨金を支払う。

▼詳細・申込
http://cybozu.co.jp/company/security/bug-bounty/

▼報奨金制度の詳細

対象	・cybozu.com で稼働する各サービス ・同社指定のパッケージ製品、API、Web ページ （詳細ページに記載あり）
期間	2014年6月19日～12月25日
報奨金	金額は共通脆弱性評価システム CVSS v2 の評価結果をもとに設定 CVSS v2 の基本値が 7.0 以上：CVSS v2 基本値 ×3 万円 CVSS v2 の基本値が 6.9 以下：CVSS v2 基本値 ×1 万円 （Web ページの問題は一律、1 件につき 1 万円）

CONTACT サイボウズ㈱
URL http://cybozu.co.jp/

Service

ブロケードコミュニケーションズシステムズ、Brocade Vyatta 5600 vRouter提供開始

ブロケードコミュニケーションズシステムズ㈱は、NFV（Network Functions Virtulization）を実現する仮想ルータ「Brocade Vyatta 5600 vRouter」を今春から提供開始した。

新しいIntel DPDK（Data Plane Development Kit）をベースにアーキテクチャを刷新したもので従来のものより10倍の性能向上を実現した。その特徴はコントロールプレーンとデータプレーンを分けたことが第一に挙げられる。これによりネットワークのコンポーネントを1つのVM（仮想マシン）に持たせるのではなく一元化し、データプレーンを複数コントロール可能となり、スケールアウトさせやすくなる。クラウド市場でのネットワーク機能提供に必須の機能である。

Vyattaの有償化については、市場ニーズへの迅速な対応のためだという。しかし、オープンソースコミュニティにはコードベースで積極的に貢献し公開していくとのこと。

▲ロバート・ベイズ氏（同社ソフトウェア・ネットワーキングR&D担当CTO）

CONTACT ブロケード コミュニケーションズ システムズ㈱
URL http://www.brocadejapan.com/

Report

アトラシアン×リックソフト プロジェクト管理ツール「JIRA」「JIRA Agile」活用セミナー

6月11日、アトラシアン社のプロジェクト管理ツール「JIRA」および「JIRA Agile」に関するセミナーが、販売パートナーであるリックソフト㈱の主催で行われた。

セッション1：これからのソフトウェア開発でのプロジェクト管理の展望～アトラシアン製品の価値

セッション1は、アトラシアン㈱エバンジェリストの長沢智治氏による講演。同氏は開発現場の効率化業務に長く携わってきた経験をふまえ、ソフトウェア開発の変遷とプロジェクト管理のあり方について解説した。

プロジェクトマネージャがすべてを把握して管理する従来の統制型管理では難しいプロジェクトが多くなってきていることから、これからはチームとして情報を共有し、各人ができる管理をそれぞれに行う自律型管理が望ましいとした。そのうえで、継続的デリバリーの開発手法を推奨する。

しかし、管理方法の切り替えは簡単に行えるものではなく、環境を整えることが重要であると指摘。アトラシアンでは、「企画・計画・開発・ビルド・デプロイ」の各段階に役立つツールを提供しており、しかも各段階相互の動きが追えるしくみが備わっていることが自律型管理をより行いやすくしているとアピールした。

最後に、「導入したとしても最初は汚い環境になってしまうことが多くありますが、最初はうまくいかなくて当然です。うまくいかないからといってやめてしまうのではなく、現状がこういうものだという認識に活かしてください。どこがうまくつながっていないかがわかれば、改善できます」と締めくくった。

▲アトラシアン㈱　長沢智治氏

▲リックソフト㈱　樋口氏

セッション2：「JIRA」「JIRA Agile」デモによる活用紹介

セッション2は、リックソフト㈱の樋口氏による「JIRA」および「JIRA Agile」の製品デモ。プロジェクト管理によく使われるツールとしてMicrosoft Excelを引き合いに出し、共有データのバージョン管理機能やコミュニケーション機能がないことによる共同作業の弱さを指摘。そのうえで、JIRAの優位点であるコミュニケーションをからめたアジャイルモデルでの開発の進め方をデモンストレーションで紹介した。

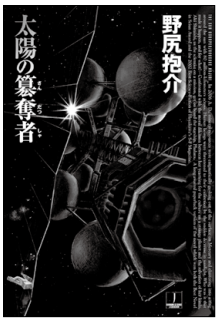
最後に、プロジェクト管理ツールを使うメリットを次のように示して締めくくった。

- プロジェクトメンバーのコミュニケーションツールになる
- プロジェクト状況をリストやグラフによって可視化できる
- プロジェクトのナレッジ・データベースになる
- ソースリポジトリ、ビルドツールとの連携により修正理由が明確になる

CONTACT リックソフト㈱
URL https://www.ricksoft.jp/
アトラシアン㈱
URL https://www.atlassian.com/ja/

SFで小飼弾

第8回



『太陽の簒奪者』
（野尻抱介／早川書房）

ある作品がSFなのか否かだという命題は、本誌の読者にとって「Emacsがエディタなのか Lisp仮想マシンなのか」ぐらいホットな話題だということは本連載でも何回か紹介しましたが、「これはSFに分類せざるを得ない」のが、異星人が登場する作品。もうウェルズの「宇宙戦争」以来の伝統といっても過言ではありません。いかにサイエンスの部分がビミョーでも、宇宙人さえ出てくればSFに分類されるのは“Star Wars”を見てのとおり。

そして異星人が出てくる作品では、彼らは人類の敵となったり味方となったりします。その傾向をざっくり見ると、Fに重きをおく作品では人類の敵となる場合が多く、そしてSに重きを置く作品では友好的な異星人が多いという傾向が明らかに見られます。後者のトレンドは年を追うごとに増しているようで。ウェルズの『宇宙戦争』のように（当時の）科学的に人類に敵対する異星人が最近ではなかなかお目にかかれず、前回紹介した『ディアスポラ』の「異宇宙人」も、本連載第2回で紹介した『竜の卵』のチーラもいたって人類に友好的でした。なぜサイエンスに重きを置くと異星人が友好的になるのかといえば、その方がリアルだというのが現代における定説で、『コンタクト』の作者でもある科学者、故カール・セーガンに言わせれば、好戦的な異星人は人類と戦争する前にとっくに自滅しているはずだ、と。

だとしたら、『未知との遭遇』は人類にとっていいことづくめなのでしょうか？　異星人に悪気はないのに人類存亡の危機が訪れるということはありえないのでしょうか？　『太陽の簒奪者』（野尻抱介）が指摘するのは、まさにその可能性です。

異星人のナノマシンが水星を削って築き上げたのは、太陽をとりまく直径8,000万kmのリング。そのリングがたまたま地球に影を落としたからさあ大変。地球は一挙に氷河期へ。人類がこの先生き残るには、まずこのリングを破壊しなければなりません。そのリングは異星人の宇宙船を止めるためのレーザー発振器らしいことが判明するのですが、だとしたらリングを壊したら異星人に対する宣戦布告になってしまうのでは？

よく考えると、我々は相手を敵に回すにせよ味方にするにせよ、その前に相手の意図を知り、相手に自分の意図を知らせようとします。しかし意志疎通を待っていては人類滅亡というのは、「敵か味方か」より一段とリアルな未知との遭遇ではないでしょうか。彼らは我々の敵にも味方にもならず、ただ[すれ違って（フライバイ）]しまうのか。ぜひご確認を。SD

題字・イラスト／aico

ひみつのLinux通信

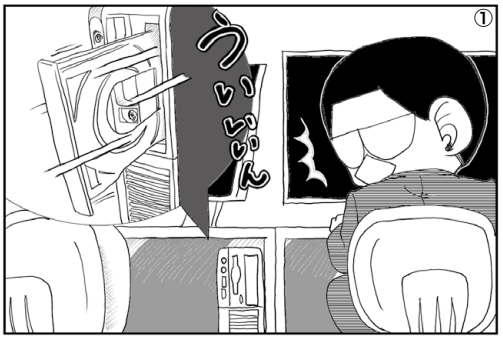
作)くつなりょうすけ
@ryosuke927

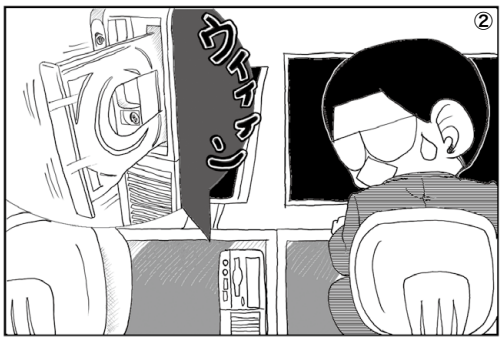
第8回 ejectコマンド

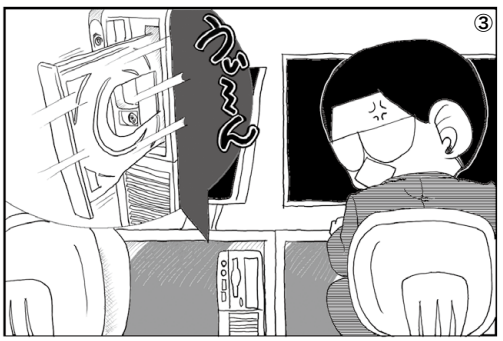
梅雨明けと海開きを心待ちにしているリア充野郎「くつな先生」に愛のムチを！

ツイート&メール

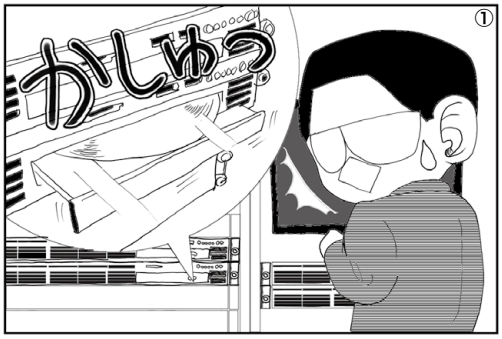
eject（1）







eject（2）



to be Continued

ejectコマンドは光学ドライブのトレイを排出するコマンドです。遠隔（リモート）から実行してPC近くにいる人を驚かせることもできます。「-t」がトレイを収納するオプションなので何度でも出し入れできますよ。1Uサーバに装備する薄型ドライブでは「-t」で収納できない場合があるので注意しましょう。筆者も誤って出してしまい、現場の人にトレイを収納してもらったことがあります。最近はデータを持ち帰るのも、OSインストールもUSBメモリで事足りるために光学ドライブを利用する機会が減っているのでejectコマンドの活躍の場も減りました。次のUSB規格で排出機能が提案されればejectで飛び出すUSBメモリを見られるかしら……。

Letters from Readers

Software Design読者層の若返り計画!?

6月に発表された政府の成長戦略素案に、「義務教育段階からのプログラミング教育を推進する」ということが書かれていました。来たる小学校でのプログラミング必修化に備えて、SDも今から子供向けプログラミング入門記事を扱うべきでしょうか。全国に小学校は22,000校あるそうです（2007年、文科省調べ）ので、全国の小学校でSDが定期購読されれば、ウハウハですよ!

2014年6月号について、たくさんのお便りをありがとうございました！

第1特集　設定ファイルの読み方・書き方でわかるLinuxのしくみ

Linuxを使っていると、設定ファイル変更などの作業は避けて通れません。本特集ではLinuxのユーザ管理、ネットワーク管理などの設定ファイルと、Apache/Sambaの設定ファイルについて、基本的な項目の意味や設定例を説明しました。

Linuxのしくみがわかりやすくまとまっており、参考になった。パートの分け方が良く、すいすい頭の中に入った。

大阪府／てんぷるさん

何をするにしても、まず設定ファイルを書き換えたり、作ったりしないといけないので、とてもよかったです。

愛知県／kmさん

systemdの話はよかった。

東京都／どら猫さん

本特集で紹介できたのは、一部の設定ファイル／項目だけですが、これを機にほかの設定ファイルについても、意味や役割を見直ししてみてはいかがでしょうか。

第2特集　Ubuntu 14.04 LTS "Trusty Tahr"

4月17日に最新のLTS（長期サポート版）である"Trusty Tahr"が公開されました。デスクトップとサーバ関連の機能の解説や、前LTSからのアップグレードにおける注意点などを取り上げました。

Ubuntu Touchもいつ実用に耐えられるようになるかわかりませんが、iOS、Androidに続くOSになってもらいたいです。

神奈川県／吉田さん

Ubuntuもいろいろと進化が楽しみです。遅れず試しながらついていきます。

富山県／Qkobさん

最近、サーバ用途としてもUbuntuが使われる事例を目にします。今後、そのような事例が増えてくると、ますますLTSの重要性が増してきますね。

一般記事　どうなってる?「Google Glass」のアプリケーション開発

一時、世間を賑わしたGoogle Glass。一般にはなかなか発売されないので、その後の状況がわかりにくいですが、アプリの開発環境は徐々に整えられてきています。その開発環境の概要、開発のイメージなどを紹介しました。

Google Glassについて日本での普及は不透明ですが、先進的なモノが大好きなエンジニアにはヒットしそうな印象です。開発にも興味がわいてきました。

東京都／ひよこ大佐さん

日本では利用に制限があることがわかっただけでもよかったです。

千葉県／Tayuさん

読者アンケートでは、「Google Glassは技適マークがないため、国内では無線の使用ができない」という情報に関するコメントが多かったです。最近は、海外発の新しいガジェットが出ても、この技適で制限を受けることが多いような気がします。もっと柔軟な運用を期待したいですね。

複雑化するサーバ環境の監視を変える「OpenTSDB」前編

ノード数が数万台といった大規模システムでの利用を想定して開発された監視ツール「OpenTSDB」の解説記事です。前編では、Vagrant上でOpenTSDB環境を構築し、リアルタイム分析を行う手順を紹介しました。

監視ツールはたくさんあるので、情報収集に役立ちました。

埼玉県／コーヒーブレイクさん

リアルタイム分析がどこまでできるのか

興味がある。

大阪府／オブジェクト脳192さん

OpenTSDBは大規模向けという特徴だけでなく、分析ツールも充実していますので、中小規模システムで監視を行っている方々も、今の監視ツールに不足を感じている方は、OpenTSDBを検討してみると良いかもしれません。

短期集中連載　Rettyのサービス拡大を支えた“たたき上げDevOps”

RettyのDevOpsの取り組みを紹介する記事の第2回目。セキュリティとシステム監視について、取り上げました。

「AWSあるある」がたくさん出てきて、とても興味深かったです。

東京都／n0tsさん

DevOpsの事例として参考になりました。

茨城県／サファイアさん

本記事は、システム運用時の悩みが生々しく書かれているのが特徴的ですが、その分、最終的なシステム構成に至った経緯もわかりやすいです。みなさんの現場で、課題を分析する際の参考になれば幸いです。

短期集中連載　Web標準技術で行うWebアプリのパフォーマンス改善

通信環境を改善することで、パフォーマンスを向上させる方法を取り上げました。具体的にはWebSocketなどのプロトコル、AppCacheなどのキャッシュ機能について解説しました。

サービス目線の場合、どうしても古いブラウザの対応も考慮しないといけないため、最新機能の導入が難しい。しかし、いずれ実践してみたいと思える内容だった。

東京都／もぐまぐさん

WebSocketを使った開発について参考になる。

徳島県／花岡さん

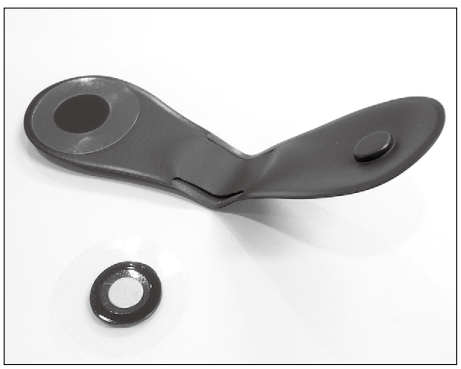
パフォーマンス改善にもさまざまなアプローチがあることが、わかっていただけたのではないでしょうか。アプリ側での改善、サーバ側での改善など、対策を施す場所もいくつかあるので、性能改善はシステム全体で考えないといけませんね。

エンジニアの能率を高める一品

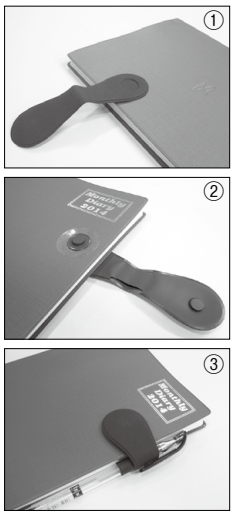
仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。

Belt Seal（マグネット）

480円（税別）／㈱デザインフィル　www.midori-japan.co.jp



ペンホルダー付きのノートや手帳はペンと一緒に持ち運べて便利です。だから、ホルダーの付いていないノートにもホルダーを取り付けたくなりませんか？　そんな場合は、このBelt Sealが便利です。ノートの裏表紙にシールでベルトを貼り付け（**写真①**）、表紙に留め金のマグネットを貼り付けると（**写真②**）、あっという間にペンホルダー付きノートに早変わり（**写真③**）。やや強引ですが、すでにペンホルダー付きの手帳に、Belt Sealでもう1本分のペンホルダーを増設することもできました。Belt Sealを貼り付けるときは、ホルダーにペンを挿した状態でやると、適切な位置に付けやすいですよ。



▲写真 Belt Seal取り付け手順

6月号のプレゼント当選者は、次の皆さまです

①クリップ専用プリンター「ココドリ」……神奈川県　西野豊陽様
②カバンの中身mini……愛知県　権田裕昭様

★その他のプレゼントの当選者の発表は、発送をもって代えさせていただきます。ご了承ください。

次号予告

Software Design
September 2014

2014年9月号
定価（本体1,320円+税）
176ページ

8月18日発売

［第1特集］パワーアップのチャンス!

この夏に克服したい2つの壁
「C言語のポインタとオブジェクト指向」

多くのプログラマ、エンジニアにとって避けて通れないテーマである「ポインタ」と「オブジェクト指向」。組込系開発だけでなく、最近流行のMake:系テクノロジを自在に操りたいときにC言語が見直されています。さらにゲーム開発ではC/C++が重要な役割にあることは皆さん承知のことでしょう。オブジェクト指向についても、Javaでの開発だけでなく、スクリプト言語でもまさに必須な技術です。

本特集では、これら2つの壁を乗り越える手がかりを多面的に紹介します。ITエンジニアのための「精神と時の部屋」で鍛えてみませんか!

［第2特集］基本技術を学ぶ

ネットワーク、データベースのための「クラスタリング」の教科書

■特別企画
・［実践+基礎］Serf・Consul超入門――オーケストレーションツール事始め
・IBMが放つIaaSプラットフォーム「SoftLayer」の使い方
■特別付録
より抜き「網野衛二の3分間ネットワーク基礎講座」

※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。

SD Staff Room

●今月の特集記事は特盛です。案外知られていない「ログ」を取り上げてみたのですが、そこにはきわめて人間的な世界が広がっていました。そもそもコンピュータの行動記録なわけですが、さらにもとを探れば人間の欲望の集積結果と言えます。見る人が分析すれば「全部お見通しよ！」のデータになるわけです。（本）

●以前はEmobileルータとiPad、iPod Touch、Nexus7、HTC J、コンデジ、モバイルバッテリという布陣だったが、最近はXperia Z1とiPad mini retina with IIJmioだけで済むようになった。しかし、緊急時対策として水1Lや食料、サバイバル系が増えて重量は変わらない。所詮、重力に魂を縛られているのだろうか。（感違幕）

●本誌特集で好評いただいた文書術が本になりました！『<文章嫌いではすまされない！>エンジニアのための伝わる書き方講座』という書名です。苦手意識を軽減する視点を身につけることで、自身の技術力を伝えるコミュニケーションツールの1つとして、文書を活用していただくための本です。（キ）

●前号でLinuxConのTシャツを今号のプレゼントにすると言いましたが、多くの企業からプレゼント品提供のお申し出があり、あぶれてしまいました。またの機会に提供します。同様に余っているノベルティなどがたくさんあります。いつか蔵出し大放出しようと思いつつ、たまっていく一方です。（よし）

●6月初め、宮古島へ旅行に行きました。気温は東京よりも5℃程高く、蒸し暑い気候でした。島で一番驚いたことは、きれいな海！　ではなく、海ぶどうの味！　甘酸っぱいデザートのようなものだと思い込んでいたのですが、むしろ塩っぱいんですよね。オリオンビールに良く合いました。（な）

●実家で家庭菜園をしたいという話をしたところ、きゅうりとオクラの苗を分けてもらいました。さっそく家に帰って植木鉢に植え、一晩たって見てみたら……きゅうりの苗がしおれ気味。どうやら水が足らなかったみたいで、吸い上げに失敗している模様。慌てて水をあげましたが、収穫までたどり着けるかな？（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2014年8月号

発行日
2014年8月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
松本涼子

●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。