Software Design
[ソフトウェア デザイン]
OSとネットワーク、
IT環境を支える
エンジニアの総合誌
7
2016
2016年7月18日発行
毎月1回18日発行
通巻375号
（発刊309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

》プログラマのためのTCP/IP　》正規表現できますか?

試して実感!

Special Feature 1

プログラマが知っておくべきTCP/IP

C | JavaScript | PHP | Python | Rubyでコーディング

Wiresharkでパケットキャプチャ



Special Feature 2

プログラミング／
エディタ作業の
効率大幅アップ

手を動かして学ぼう 正規表現入門

記事とWebツールで
トレーニング

? . + * \d {n} {n,m} [AB] [^AB] ^ $ ABC|DEF

新連載

DevOps脳を作る
基礎トレーニング

アプリエンジニアのための[インフラ]入門



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）

Special Feature 1 第1特集

プログラマが知っておくべきTCP/IP

試して実感！

C | JavaScript | PHP
Python | Rubyでコーディング、
Wiresharkでパケットキャプチャ

2016 7
［目次］
Software Design

contents

			017
序章	プログラマがTCP/IPを知っておくべき理由	及川 卓也	018
第1章	TCP/IPって何？～ネットワークの基礎の基礎	網野 衛二	023
第2章	実践ネットワークプログラミング～C言語編～	あきみち	035
第3章	実践ネットワークプログラミング～スクリプト言語編～	中島 雅弘、植西ゴロー由謹、國信 真吾	046
第4章	パケットキャプチャで学ぶTCP/IP	吉田 英二	059



7
2016
［目次］
Software Design

Special Feature 2 第2特集

プログラミング／エディタ作業の効率大幅アップ

手を動かして学ぼう 正規表現入門

記事とWebツールでトレーニング 伊藤 淳一 067

序章 正規表現は絶対にマスターしておけ! 068

第1章 さまざまな形式の電話番号を検索しよう 071

第2章 カタカナ語の揺れを許容して検索しよう 078

第3章 HTMLをCSVに置換しよう 081

第4章 スペースや改行などの空白文字を検索／置換しよう 087

第5章 正規表現のパフォーマンスを意識しよう 091

àla carte アラカルト

ITエンジニア必須の最新用語解説［91］SyntaxNet 杉山 貴章 ED-1

読者プレゼントのお知らせ 016

SD BOOK REVIEW 095

バックナンバーのお知らせ 135

SD NEWS & PRODUCTS 186

Readers' Voice 190



contents

Column

digital gadget [211]
変わり種のデジタルカメラ — 安藤 幸央 — 001

結城浩の再発見の発想法 [38]
ドッグフーディング — 結城 浩 — 004

[増井ラボノート]コロンブス日和 [9]
ExpandHelp — 増井 俊之 — 006

宮原徹のオープンソース放浪記 [5]
OSC群馬のドキュメンタリー — 宮原 徹 — 010

ツボイのなんでもネットにつなげちまえ道場 [13]
Why!? 技適 — 坪井 義浩 — 012

ひみつのLinux通信 [29]
デフォルト至上主義 — くつなりょうすけ — 169

Hack For Japan〜エンジニアだからこそできる復興への一歩 [55]
GAMATHON〜島ソン第3弾はクラウドファンディングを目指す — 小泉 勝志郎 — 180

温故知新 ITむかしばなし [56]
マイコンゲーム〜キャラクタ表示型のゲーム〜 — 速水 祐 — 184



Development

アプリエンジニアのための[インフラ]入門 【新連載】
データベース入門 — 出川 幾夫 — 096

使って考える仮想化技術 [2]
ホストシステムと仮想環境の構築(その1) — 笠野 英松 — 102

RDB性能トラブルバスターズ奮闘記 [5]
インデックスが動くしくみを理解しよう — 生島 勘富、開米 瑞浩 — 110

Androidで広がるエンジニアの愉しみ [7]
新しい3DグラフィクスAPI Vulkanを使ってみる — 高橋 憲一 — 116

Vimの細道 [9]
カーソルを自在に操る — mattn — 122

るびきち流Emacs超入門 [27]
シェルコマンドを活用しよう(後編) — るびきち — 128

書いて覚えるSwift入門 [16]
創立40周年を迎えたApple — 小飼 弾 — 132

Sphinxで始めるドキュメント作成術 [16]
Sphinxで運用ドキュメントを作る — 波田野 裕一 — 136

Mackerelではじめるサーバ管理 [17]
ロール編成の考え方 — 坪内 佑樹 — 144

OS/Network

Unixコマンドライン探検隊 [3]
プロセス、状態と制御 — 中島 雅弘 — 148

Be familiar with FreeBSD〜チャーリー・ルートからの手紙 [32]
使ってみようmtree(8)コマンド(その2) — 後藤 大地 — 156

Debian Hot Topics [37]
“Wheezy”LTS開始と、パッケージにまつわる悲喜こもごも — やまねひでき — 160

Ubuntu Monthly Report [75]
Ubuntu 16.04 LTS落ち穂拾い — あわしろいくや — 164

Linuxカーネル観光ガイド [52]
Linux 4.2から導入されたspinlockの進化形、qspinlock — 青田 直大 — 170

Monthly News from jus [57]
変わらない技術を求めて — 法林 浩之、りゅうちてつや — 178

[広告索引]
グレープシティ
http://www.grapecity.com/
裏表紙
システムワークス
http://www.systemworks.co.jp/
前付
日本コンピューティングシステム
http://www.jcsn.co.jp/
裏表紙の裏

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

GIHYO Digital Publishing

技術評論社の本が電子版で読める!

https://gihyo.jp/dp

イチオシの1冊!

[改訂新版]Spring入門
——Javaフレームワーク・より良い設計とアーキテクチャ

長谷川裕一，大野渉，土岐孝平　著
3,800円　PDF　EPUB

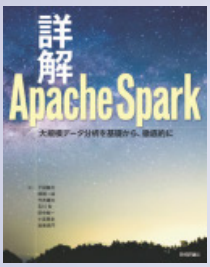
Javaを利用したシステム開発ではミドルウェアとしてSpring Frameworkが定番になった。バージョン4.0以降，急速に普及しているクラウド環境への対応が当たり前のものになり，さらにはマイクロサービスへの対応もすでに当然の機能になっている。そうしたWebアプリケーションの作り方の枠組みすべてを含んでいる体系がSpringと言えよう。

本書はSpringの使い方を基礎の基礎から，応用的・実践的な開発まで広く解説する。

https://gihyo.jp/dp/ebook/2016/978-4-7741-8247-6



あわせて読みたい



詳解 Apache Spark
EPUB　PDF



Firefox OS【決定版】開発ガイドブック
EPUB　PDF



C#プログラマーのための基礎からわかるLINQマジック!
EPUB　PDF



Webサーバを作りながら学ぶ
基礎からのWebアプリケーション開発入門
EPUB　PDF

他の電子書店でも好評発売中!

amazon kindle　honto　楽天kobo　ヨドバシカメラ www.yodobashi.com

お問い合わせ

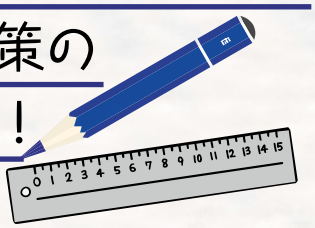
〒162-0846　新宿区市谷左内町21-13　株式会社技術評論社　クロスメディア事業部
TEL：03-3513-6180　メール：gdp@gihyo.co.jp
法人などまとめてのご購入については別途お問い合わせください。



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

あなたを
合格へと導く
一冊があります！

効率よく学習できる
試験対策の
大定番！





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



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



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



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



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



左門至峰・平田賀一・山内大史・幸田廣信　著
A5判／312ページ
定価（本体2380円+税）
ISBN978-4-7741-8067-0



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



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



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



角谷一成・イエローテールコンピュータ　著
A5判／512ページ
定価（本体1680円+税）
ISBN978-4-7741-7849-3

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



たった1秒で仕事が片づくExcel自動化の教科書

待望の新刊!!

●吉田拳 著

A5判／328ページ　定価（本体1980円+税）
978-4-7741-8087-8

「同じ作業を毎週繰り返しこなすだけで、仕事が終わってしまう……」
「大量のデータを扱うので、ミスが絶えない…」
といった悩みを解決するのは、Excel作業の効率化ではなく、Excelの作業そのものを自動化して、ゼロにしてしまうこと。そんな“究極の効率化”を実現するために必要なマクロ・VBAのポイントと、毎日の業務を瞬時に終わらせるしくみの作り方を教えます。



A5判／328ページ
定価（本体1780円+税）
978-4-7741-6808-1

たった1日で即戦力になるExcelの教科書

大ベストセラー
14万部
突破!!

●吉田拳 著

「Excelぐらい、まあなんとかなるよ」
…そういいつつ、作業に何時間もかかってイライラしたり、
いつもミスをして時間をムダにしていませんか？
そんなExcel地獄に陥らないための、今日から即役立つテクニックから、社会人10年目でも意外とわかっていない
「なぜ、それが必要なのか？」といった目的意識まで、実務直結の知識を最小限の時間でマスター。
50社以上、のべ2000名以上の指導実績に裏打ちされたノウハウが満載！

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

第91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。

SyntaxNet

自然言語認識フレームワーク「SyntaxNet」

「SyntaxNet」は、Googleがオープンソースで公開した自然言語認識システムを構築するためのフレームワークです。自然言語の認識は、コンピュータサイエンスの分野における極めて重要で、かつ難しい課題のひとつとされています。SyntaxNet自体は自然言語認識システムそのものではなく、自然言語認識を実現するための基盤となる学習機能を備えたフレームワークであり、これをベースとしてさまざまな自然言語用の認識システムを実装することができるようになります。

人間が使用する自然言語には多くの曖昧さが含まれており、このことがコンピュータによる構文解析を極めて困難なものにしています。数十個程度の単語からなる文章でも、該当する文法構造をすべて挙げると膨大な数にのぼる可能性があるからです。構文解析器は、その膨大な選択肢の中から文脈に合ったもっとも適切な候補を見つけ出す必要があります。

この問題に対して、機械学習を用いることで高精度の構文解析器の構築を可能にするのがSyntaxNetです。機械学習ライブラリとしては、同じくGoogle製の「TensorFlow」が使われています。つまりSyntaxNetは、TensorFlowをベースに構築された自然言語理解（Natural Language Understanding：NLU）のためのニューラルネットワーク・フレームワークというわけです。SyntaxNetには、新しいモデルのトレーニングに必要となるすべてのコードが含まれており、目的に合わせて任意の自然言語の解析用に学習を行うことができるとのことです。

SyntaxNetベースの英語パーサ「Parsey McParseface」

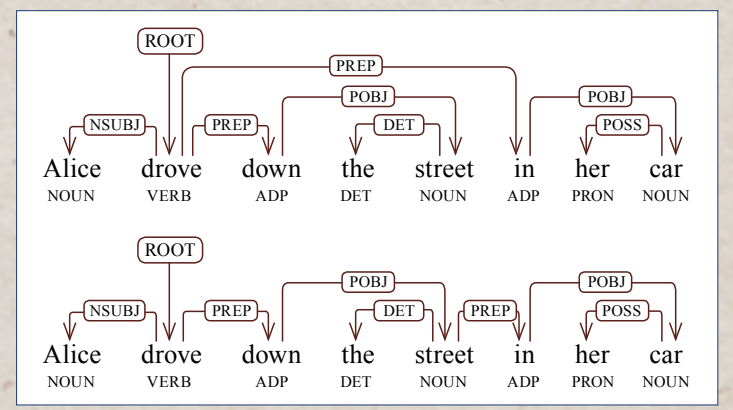
現在公開されているSyntaxNetには、英語の文章解析に利用できる「Parsey McParseface」が付属しています。これはGoogleの開発チームがSyntaxNetを用いた学習によって機能強化した英語パーサで、文章内の単語の機能的な役割（名詞、動詞など）を解析することができます。英語の文章の構文解析において、機械学習による学習結果を使用して正しい文脈を推測する点がParsey McParsefaceの特徴です。

図は8個の単語から成る英文を解析した例です。2つの解析結果は文法だけ見ればいずれも誤りではありませんが、下の例は「the street in her car」をひとまとまりとするおかしな文脈になってしまっています。このような不適切な候補を排除して解析の精度を上げるために機械学習が使われます。

Googleの発表によれば、ランダムに抽出された英語のニュース記事に対してParsey McParsefaceによる解析を実施した場合、94%強の正確さで正しい結果が得られたとのことです。そのほか、より解析が困難なWeb上のさまざまな英語テキストに適用した場合でも、90%強の正確さを示したそうです。

ユーザの指示に対してコンピュータが適切な処理を行うためには、入力された文章の構造を正しく解釈できる必要があります。GoogleではTensorFlowに続いてSyntaxNetをオープンソース化することによって、業界全体におけるこの分野の技術レベルの底上げを目指しているようです。SyntaxNetはGitHubのリポジトリより入手することができます。SD

▼図　8個の単語から成る英文の解析例（公式サイトより）



SyntaxNet
https://github.com/tensorflow/models/tree/master/syntaxnet

DIGITAL GADGET vol.211

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

変わり種のデジタルカメラ

デジタルカメラの台頭

スマートフォン用のカメラアプリであり、写真を中心としたSNSとして人気を博しているInstagramのアイコンが、最近カメラ風のものから、カメラをデフォルメしたようなシンプルなアイコンにリニューアルされました。Instagramの旧アイコンは、ポラロイドカメラの筐体を正面から見据えた風貌を模倣したものであり、アプリで撮影した写真も正方形のポラロイドを彷彿とさせるものでした。つまりは、今までの写真カルチャーの延長線上にある表現からスマートフォン時代の表現に変化したのです。近い将来、Instagramなどのカメラアプリが、なぜこのような風貌なのかを忘れてしまう時代が来るのかもしれません。

今でこそ、フィルムカメラ、デジタルカメラの延長線上としてスマートフォンのカメラアプリが存在しますが、近い将来、初めて触れるカメラがスマートフォンという人々も増えていくことが考えられます。今回は、そんな従来の「カメラ」という概念とは異なる、変わり種のカメラを紹介していきます。VR業界の発展や、Facebookなどの埋め込みによってパノラマカメラはますます勢いづいていますが、過去の記事で紹介したパノラマカメラは今回はあえて除外しています。

デジタルカメラの登場で、カメラという概念が大きく変わりました。現像というプロセスと待ち時間が無くなったことが、大きな変化の1つです。今でもデジタルカメラのRAW画像ファイルを加工編集するデジタル現像の作業は「現像」と言いますが、銀塩フィルムの現像をお願いして写真プリントができあがるまで、数時間から数日間待つという概念そのものがなくなっています。

その現像までの待ち時間をノスタルジックな事柄として楽しむため、「写ルンです」のようなフィルム一体型カメラがまたリバイバルで流行り始めているという現象もあります。プロカメラマンの仕事用機材もデジタルカメラが浸透しつつありますが、フィルムカメラを使い続けるために、一生分のフィルムを買い貯めているプロカメラマンもいるそうです。

フィルムがなくなり、デジタル化したことで、現像時間がなくなるメリットのほかに、解像度に関する弊害も存在します。たとえばハイビジョンで撮影した映像を考えてみてください。ハイビジョン映像は、既存の一般的に普及しているテレビで見る限りは、必要充分な解像度を持ちます。しかし、将来的に映像装置の解像度がより高精細になったときに困ってしまうのです。たとえそれが現在4Kで撮影して、高精細だと喜んでいたとしても、将来的に8KやHDRの表示装置が主流になったときに、それに追従できなくなるのです。

アップコンバートと呼ばれる、低い解像度の映像を一段階上の高解像度の映像に変換する技術や機材も進化しつつありますが、もともとない情報を補間だけで高解像度化するにはそもそも限界があります。フィルムならデジタルスキャンしなおせば、ある程度の高解像度であれば追従します。古い映画の上映の際に「リマスター」と呼ばれるのは、古いフィルム原盤を再度スキャンしなおしたり、フィルムの状態を修復したりして、その時々の最高の状態で公開することができるのです。ですから、予算に余裕のあるハリウッド大作映画の場合、フィルムで撮影し、デジタル化をしたとしても、遠い将来も満足に足りうる解像度で上映できることを想定しています。もちろんフィルムの乳剤の細かさや、保存状態にも影響しますが、長く保存し、将来のことも考えることは重要です。


旧Instagramのカメラ風アイコン


新Instagramのカメラを象徴として扱ったアイコン

※本記事で紹介しているものは
国内未発表・未発売のものを含んでいます。

カメラの自由度と活躍の場

カメラの本質は、ある特定の時間に目で見たものの、時間的瞬間と空間的瞬間を切り取るしくみです。画角は四角か横長、人物を撮影するポートレート写真などでは縦長。では新しいタイプのカメラとはどんなものでしょうか?

現在ではあらゆる携帯電話にカメラが搭載され、一人で何台もカメラを持つ人もいるでしょう。そこで、カメラの自由さを拡張しカメラの制限をなくしてみると、どうなるでしょう。今まで当然だと思ってきた概念を取り払うことで新しいカメラが考えられるのではないでしょうか。

デジタルカメラのこれから

現在の技術を駆使すれば、どんな形のカメラでも作れるはずなのに、なぜか依然として古い時代のフィルムカメラと同じような風貌のデジタルカメラが主流です。その理由は「メンタルモデル」と呼ばれる、人の中にある固定概念や、これはこういった形や使い方であるという経験が邪魔してしまうのです。

単なるモノとしてのカメラではなく、記憶の代替としてのカメラ、時間を切り取るカメラ、気持ちや雰囲気を残しておくためのカメラ、いっそ、残すことさえも必要なくて、目で見たものを一瞬共有することが目的のカメラ。目の

ライティングの制限をなくす



EF-M28mm F3.5 マクロ IS STM
マクロレンズの先端に
コントロール可能なLEDライトを搭載

カメラの性能制限をなくす



Nurugo Micro
スマートフォンに付けて使う
顕微鏡ツール



DxO ONE
スマートフォンに一眼レフ並みの
性能を持ったカメラ機構部を拡張



Scout TK
単眼鏡タイプの
赤外線カメラ

撮影場所の制限をなくす



Fishcam
釣りカメラ。釣り具の浮きに
つけて撮影できるカメラ



iBubble
赤外線コントロールでリモコン操
作し、水槽の中が撮影できるカメラ



Movi
遠隔操作前提の
ライブカメラ



podo
どこの壁にでも設置できる
リモートカメラ



Birdie
放り投げて使う
落下傘型カメラ

大きさの制限をなくす



Camera Obscura (San Francisco)
小屋型の超大型ピンホールカメラ



Phase One IQ180
10K超高解像度カメラ

操作の制限をなくす



GeckoEye
動体認識による屋外用防犯カメラ



Presence
赤外線検知による
監視カメラ



PICTAR
スマートフォンカメラの
グリップを改善

カメラマンが必要という制限をなくす



SOSPENDO
セルフィー用ネックレス型
カメラツール



Whiskas Catstacam
猫視点のカメラ



61N
1日中、位置情報付きで
写真を撮り続ける



Frodo
セルフィー専用腕
時計型カメラ



BULT X3
アクションカム搭載
ヘルメット

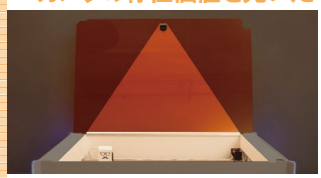


NatureView Cam
野生動物を自動で
撮影する

中に埋め込むコンタクトレンズ式のカメラで見たものすべてを記録するような、スパイ映画の中でしか実現していないようなものも特許が出されており、既存の技術の延長として考えたり、研究している人はいる模様です。

近い将来「カメラ」という概念は変わっているかもしれませんが、人々の記憶や風景を切り取り、皆で共有して楽しむことや、アート作品としての写真はなくなることはないでしょう。SD

カメラがなかったところにカメラの存在価値を見いだす



Mr Beam II
卓上レーザーカッターの内部にカメラを搭載。彫り方を詳細に検討できる

ピントの制限をなくす



Lytro
撮影後、あとからピントが合う場所を指定して調整することができるカメラ

特定のものを撮影する手間をなくす



Kaptivo
ホワイトボード専用カメラ

時間の制限をなくす



meMINI
5秒から5分さかのぼって記録できるカメラ



TLC200
タイムラプス撮影専用カメラ

Gadget 1

LG Action CAM LTE

http://www.lgnewsroom.com/2016/05/lg-introduces-active-lifestyle-camera-with-live-streaming-over-4g/

LTE通信搭載アクションカム

GoProをはじめ、アクションカムは多数の製品が存在しますが、このLG Action CAM LTEは携帯電話のLTE通信網に直接接続することのできるアクションカムです。つまり撮影しながら、その動画をアップロードし、配信できるのです。ライブストリーミング時は最大1280×720（60fps）品質の動画を30分程度配信可能。microSDへの記録も可能です。重さは95g。Snapdragon 650を搭載し、ほとんどスマートフォン並みの性能です。



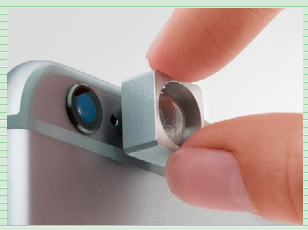
Gadget 2

F.lens

http://www.relio.it/flens

スマートフォン用フラッシュ拡張ツール

F.lensはスマートフォンにもともと搭載されているLEDフラッシュライトを拡張して、数メートル先でもフラッシュの光を収束させて、明るく照らせるようにするツールです。スマートフォンのLEDライト部分に丸いワッシャーを接着しておき、F.lensを使うときにはマグネットでくっつくようになっています。現在クラウドファンディングKickstarterで制作資金を集めているところです。



Gadget 3

写ルンです30周年アニバーサリーキット

http://fujifilm.jp/personal/filmandcamera/utsurundesu/promotion/lf30/kit.html

レンズ一体型カメラ復刻版

1986年に登場した「写ルンです」はフィルムにカメラ機能が一体化しており、そのまま写真店へ現像に出すことができるという画期的な製品でした。フラッシュ搭載のものもあり、手軽で必要十分なカメラとして爆発的に普及しました。スマートフォン、デジタルカメラ全盛の今、レトロ調の風貌と現像するまで待たないと何が写っているのかわからないというもどかしさを楽しむ人が増えているそうです。キットは現在発売中のレンズ一体型カメラに取り付けます。



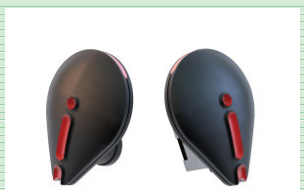
Gadget 4

Pinout

https://www.indiegogo.com/projects/pinout-a-smart-versatile-gadget-for-your-camera#/

スマホでカメラをリモコン操作

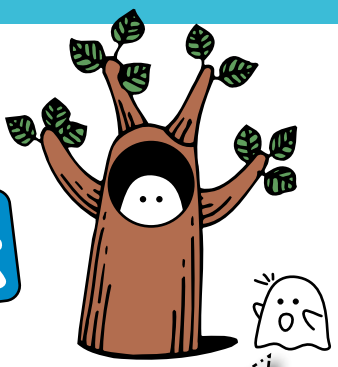
Pinoutはデジタルカメラ（現状ニコンのみ対応）に搭載されているアクセサリポートに装着し、リモコン機能をBluetooth+スマートフォンに置き換える装置です。電源はカメラから取得します。Pinoutを活用すると、一定時間ごとに撮影するタイムラプスや、撮影時の位置情報を記録しておくなど、スマートフォンを最大限に活用した利用方法が考えられます。装着したカメラと、スマートフォンが20メートル以上離れたら警告してくれます。



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

38

結城浩の再発見の発想法



ドッグフーディング

ドッグフーディング

ドッグフーディングとは

ドッグフーディング(dogfooding)とは、**開発するサービスを開発者自らが使用する**ことです。語源については諸説あり、「ドッグフードのコマーシャルに登場する有名人は自分の愛犬にそのドッグフードを食べさせているか？」という話が中心のようです(**図1**)。「ドッグフード」という名詞から変化して、現在では「開発するサービスや製品を開発者自らが使用する」という意味の動詞として使われています。ここには2つのポイントがあります。1つは「**品質保証**としてのドッグフーディング」で、もう1つは「**宣伝広報**としてのドッグフーディング」です。

良いサービスを開発するためには顧客のニーズや、既存のサービスへの不満を調査することが必須です。ドッグフーディングを行い、開発者が自分のサービスを使うなら、その調査は自動的に行われることになります。自分がサービスのユーザなので、ニーズに応え不満を解消する方向へ自然に向かうことになるでしょう。これは「品質保証としてのドッグフーディング」と言えます。

本当に良いサービスなら、多くの人が使いたいと思うでしょう。でも、もしも開発者自身がそのサービスを使っていないとしたら、「本当に良いサービス」と言えるのか、疑問符がつきますね。逆に、開発者自身がそのサービスを使っているなら、「ああ、これは本当に良いサービスなんだろうな」と他者へのアピールになります。これは「宣伝広報としてのドッグフーディング」と言えます。

IT業界ではドッグフーディングがよく行われます。それは、プログラミングをはじめとする開発活動ではもともと、自分が使うツールやサービスを自分で作ったりカスタマイズしたりすることがよくあるからです。また、開発者自身が必要に駆られて作り社内で使っていたサービスを、完成度が高くなったので社外に展開するというパターンもあるでしょう。

▼図1　宣伝マンは本当にそれを使ってる？



ドッグフーディングと当事者意識

会社の改善活動などで、よく**当事者意識**という単語が使われます。「トラブルを目にしたとき、他人事だと思わずに当事者意識を持って解決にあたりなさい」などと言われます。

ドッグフーディングは「当事者意識を持つ」よりも一歩進んで「当事者になってしまう」ことだ

と言えます。サービスが向上すれば開発者自身の役に立つし、サービスが悪ければ開発者自身が困る。これはまさに当事者になった状態と言えますので、うまくはまればサービスの改善に直結するでしょう。

ドッグフーディングの危険性

開発者にとっては、ドッグフーディングを行うことは品質を向上させる役に立つでしょうし、サービスを受ける側にとっても、ドッグフーディングを行っている会社のサービスを使うことには安心感があります。

けれど、どんなことでも万能ではありません。ドッグフーディングにも危険性はあります。1つは「自社サービスに慣れてしまう危険性」です。自社サービスを使い続けている開発者は当然ながらそのサービスに慣れることになります。ということはそのサービスを使い始めるユーザの戸惑いや混乱、サービスのわかりにくさを理解することは難しくなるかもしれません。

もう1つはその裏返しで「他社サービスの理解不足になる危険性」です。自社サービスを使うことが当たり前になってしまうと、他社サービスがどのようなものかわかりにくくなってしまいますね。それは、ドッグフーディングが目指している顧客の立場に立つという状態から離れてしまうことになります。顧客は多くのサービスを比較・検討して自分の使うサービスを選択するからです。

さらに「顧客と開発者にずれがあった場合の危険性」も考えられます。ドッグフーディングによってサービスが改善されるのは、本来の顧客が感じる不満点と、開発者が感じる不満点に共通部分が多いときです。しかし、もしも顧客と開発者の感覚にずれがあったなら、開発者が行うサービスの改善は見当外れのものになってしまうでしょう。その場合にはドッグフーディングは逆効果ということになります。

日常生活とドッグフーディング

先日、テレビの情報バラエティ番組で「**医者が自分で行っている健康法**」特集を放映していました。医者は患者の健康を守るのが仕事ですが、それでは自分自身の健康はどう守っているのか。医者が自分自身に対して行っている健康法には確かに興味がそそられます。それは、専門家が自分の健康を守るためにやっていることなので、効果に信頼性がありそうだと感じるからでしょう。これは「品質保証としてのドッグフーディング」に近いですね。

食料品の安全性が問題になる出来事があると、**政治家がその食料品を食べてみせる**映像が報道番組で流れることがあります。これは、食料品の安全性を直観的にアピールする良い方法だと思います。本当に安全だと確信できなければ、自分が食べてみせるなんてできないだろうと感じるからです。これは「宣伝広報としてのドッグフーディング」にあたります。

筆者はたまに講演を行うことがありますが、その際にはできるだけ**事前に聴衆席に座って音響効果やスクリーンの見え方を確かめる**ようにしています。聴衆席に座ることでサービスを受け取る立場に身を置き、問題がないかどうかを確認するためです。講演に限りません。自分が販売しているメールマガジンや電子書籍を自分で購読してみるのも有効です。そのドッグフーディングの中で、筆者はたくさんの不具合を見つけましたし、読者が勘違いしそうな点にも気づくことができました。これは「品質保証としてのドッグフーディング」そのものですね。

相手が受け取るものは自分も受け取ってみるし、相手が使うものは自分も使ってみるというドッグフーディングの精神は、どんな場面でも、どんな人にも適用できます。たとえばメールひとつにしても、自分が送ったメールをあらためて読み直してみることはメール作成を上達させるものです。

あなたの周りを見回して、ドッグフーディングしていると思われるサービスがあるか、探してみましょう。また、あなた自身は自分の提供しているサービスをドッグフーディングしているでしょうか。ぜひ、考えてみてください。SD

増井ラボノート

コロンブス日和

第9回　ExpandHelp

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

Author　増井 俊之（ますい としゆき）　慶應義塾大学

ヘルプの憂鬱

複雑なシステムにはオンラインヘルプやマニュアルが用意されているものですが、パソコンやアプリケーションの解説本が大量に売られているところを見ると、ヘルプやマニュアルは十分役にたってはいないようです。

ヘルプが利用されない最大の理由は**必要な項目が見つからない**ことでしょう。たとえばMacを米国時間に設定したいと思って「米国時間」で検索すると**図1**が表示されます。

これではわけがわからないので「時間」で検索すると、より多くの項目が表示されますが(**図2**)、米国時間の設定には関係なさそうです。

「時刻」や「時計」で検索すれば「日付と時刻」環境設定」という項目が見つかりますが、これで米国時間に設定できるのかはよくわかりませんし、そもそも「米国時間」や「時間」でも検索できるべきではないでしょうか。

必要な項目が見つかった場合でも、そこに記述されている方法は自分で実行する必要があります。ユーザはMacを米国時間に設定したいだけであり、時刻やタイムゾーンを設定する方法を知りたいわけではないのに、ヘルプシステムは一般的なやり方しか教えてくれませんから、ユーザは自分で環境設定パネルを開いて、米国時間に設定する必要があります。

このように、たいていのヘルプシステムは、

・ヘルプ項目が見つからない
・自分で操作が必要

という問題があることになります。一方で、最近は普通にWeb検索すればそれなりに有用情報が見つかることが多いため、ヘルプシステムへの期待が低くなってしまっているのだと思われます。

ExpandHelp

Web検索に苦労することが少なくなってきたのは、検索エンジンの性能が向上したこともさることながら、情報の量がすごい勢いで増えてきたことによる影響も大きいでしょう。どこかのブログに「Macの時計を米国時間に設定する方法」という記事があれば、普通に「Mac 米国時間」で検索すれば見つかる可能性が高いと思われますが、**さまざまな表現で情報が表現されていれば検索に成功する可能性が高くなる**と言えるでしょう。

Generate and Filter

パズルを解く方法として「Generate and

▼図1　米国時間で検索すると……



▼図2　さらに「時間」で検索すると……



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

Test」と呼ばれるアルゴリズムがあります。たとえば「数独」を解こうとする場合、理屈で推論して空きを埋めていくのではなく、あらゆる数字の組み合わせを**生成**して順番に**試して**みれば、時間はかかるかもしれませんが必ず最終的に答が見つかるはずです。これはひどく効率が悪いやり方に見えますが、明らかに間違っている組み合わせを早目に無視するような工夫をすれば、このような簡単なアルゴリズムでもうまくパズルを解くことができる場合があります。

ヘルプの検索でもこれに似た方法を使うことができます。ヘルプデータやWebをキーワード検索する場合、検索対象は固定であり、それを検索するためのキーワードをユーザが工夫するのが常識になっています。たとえば「時間」というキーワードを使って、米国時間に関連した情報を探すことに失敗した場合は「時刻」や「タイムゾーン」のような別のキーワードで再検索を試みるのが普通でしょう。これは結果的に「**(時間|時刻|タイムゾーン)**」のような正規表現で検索したのと同等ですから、ユーザにとっては面倒な話ですが、Web検索でもファイル検索でもこういうやり方があたりまえになっているので、これをとくに苦痛だと思っていない人も多いかもしれません。

しかし本当にユーザにとって便利なヘルプ情報を提供しようとするならば、ユーザに検索語を工夫させるのではなく、「米国時間にセット」でも「タイムゾーンを設定」でもヘルプエントリを検索できるように、あらゆる表現でデータを用意しておくほうが親切です。たとえばヘルプシステム側に「タイムゾーンを**(米国|英国)(時間|時刻)に(設定|セット|変更)**する」のような正規表現を用意しておき、これを使ってあらゆるヘルプ表現の組み合わせを**生成**し、ユーザの与えた検索語で**フィルタリング**することにすれば、「米国時間にセット」でも「タイムゾーンを設定」でもヘルプが見つかるようになるはずです。

Generate and Testであらゆる解の候補を生成してから解のチェックをするのと同じ要領で、あらゆる説明文字列のパターンを**生成**しつつ、その中でユーザの検索文字列とマッチするものでフィルタリングを行う「Generate and Filter」方式を使えば、ユーザはキーワードについて頭を使うことなく検索に成功する可能性が高まります。このような手法をExpandHelpと呼ぶことにします。

展開されたデータは巨大になってしまう可能性がありますが、最近のパソコンでは多少大きなデータを利用しても問題ありませんし、Generate and Testの場合と同じように実装を工夫すれば、実際にすべての文字列を展開せずに検索を行うこともできます。



ヘルプ項目の実行

ヘルプ項目の検索に成功した場合、記事を参照しながらユーザが自分で操作を行う必要があります。この手間をなくすには、**ユーザが指定した検索キーワードを基にして、ユーザの意図に合致する操作コマンドをシステムが生成して実行できる**ようにすれば良いでしょう。

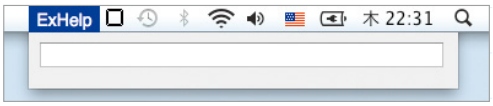
「米国時間」でヘルプ検索に成功してユーザの意図どおりの結果が出た場合、ユーザの意図は正しく検索システムに伝わっていることになりますから、そのままシステムに実行を指示できます。タイムゾーンを設定するtimezoneというコマンドが存在したとすると、「米国」のようなパラメータを使って、「timezone **米国**」を起動できます。



Mac版ExpandHelp

このような考えに基づいて、Mac版のExpandHelpを試作してみました。MacのExpandHelpはメニューバーに常駐するアプリケーションで、キーワードを入力するとそれに関連したヘルプが表示されます(**図3**)。たとえば「3:45」と入力すると、3:45という時刻に関連するコマンドが

▼図3　Mac版ExpandHelpの起動



メニューに表示されます(**図4**)。

ここで「時刻を3:45にセットする」を選ぶとdateコマンドが実行されて、システムの時刻が3:45にセットされます。

ExpandHelpのヘルプデータは**図5**のような配列として表現されています。

「**(時計|時間|時刻)を(0|1|2|3|4|5|6|7|8|9|10|11|12):(0|1|2|3|4|5)(0|1|2|3|4|5|6|7|8|9)(AM|PM)に(セットする|設定する|あわせる)**」を展開したパターンのうち3:45にマッチするものが表示され、「**時刻を3:45にセットする**」を選択するとsetdate(3,4,5)が実行されます。

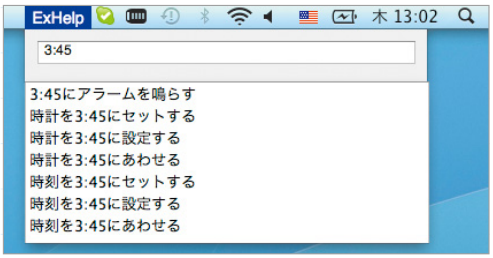
GitHelp

Mac版のExpandHelpは毎日使うようなものではないにもかかわらずヘルプデータの用意は面倒なので、結局あまり使わなくなってしまいました。万人向けの一般的なヘルプデータを用意するのはたいへんですが、自分が毎日のように使う複雑なシステムの利用を助けてくれるシステムならばもっとうれしいだろうと考え、Gitのヘルプシステム「GitHelp」を作ってみました。

▼図5　ExpandHelpのヘルプデータの構造

```
['(^{weather})の天気を調べる',
 `open http://tenki.jp/forecast/point-#{$1}.html`],
['(^{restaurant})のレストランを調べる',
 `open #{$1}`],
['(時計|時間|時刻)を(0|1|2|3|4|5|6|7|8|9|10|11|12)時に(セットする|設定する|あわせる)',
 '# date #{$2}:00',
 /[0-9]/], # 数字を入力したときだけ利用
['(0|1|2|3|4|5|6|7|8|9|10|11|12)時に(時計|時間|時刻)を(セットする|設定する|あわせる)',
 '# date #{$1}:00',
 /[0-9]/],
['(時計|時間|時刻)を(0|1|2|3|4|5|6|7|8|9|10|11|12):(0|1|2|3|4|5)(0|1|2|3|4|5|6|7|8|9)(AM|PM)に(セットする|設定する|あわせる)',
 'setdate(#{$2},#{$3},#{$4})',
 /[0-9]:[0-5][0-9]/],
['(時計|時刻)を(0|1|2|3|4|5|6|7|8|9):45に(セットする|設定する|あわせる)',
 'setdate(3,4,5)',
 /[0-9]:[0-5][0-9]/],
 '# delete mail from #{$1}'],
['(^{ps})というプロセス|アプリケーション|プログラム)を(消す|止める|停止する|殺す|落とす)',
 `kill -9 #{$1}`],
['(走って|動いて)いる(プロセス|アプリケーション|プログラム)をリストする',
 `ps -eaf`],
['現在のディレクトリのファイルをリストする',
 `terminal "ls -l"`],
['(#{station})駅から(#{station})駅までの電車の(時刻|時間)を調べる',
 `open "http://www.jorudan.co.jp/norikae/cgi/nori.cgi?rf-top&eok1-&eok2-&pg-0&eki1-#{$1}&eki2-#{$2}&Cway-0&S-検索&Csg-1"`],
['(^{twitteraccount})のtwitterを読む',
 `open http://twilier.com/#{$1}`],
['Firefox(ブラウザ)を(起動する|動かす|使う|走らせる)',
 `open -a firefox`],
['(#{ls})というファイルをFirefox(ブラウザ)で開く',
 `open "#{$1}" -a firefox`],
['(最新の|(最も|いちばん)新しい)(#{apps})ファイルを(開く|見る)',
 'file=`mdfind2 -c #{$3} -f 1970 | head -1`; `open \\#{file}`'],
['大きな(#{apps})ファイルをリストする',
 `terminal "mdfind2 -c #{$1} -b 10 | head -10"`],
```

▼図4　関連するコマンドが表示される



バージョン管理システム「Git」は、最近プログラマの間で広く使われていますが、基本概念もコマンド構成も複雑なのが難点です。Gitには100個以上のサブコマンドが存在するうえに各サブコマンドの仕様も複雑です。バージョン履歴のログを表示する「**git log**」コマンドのマニュアルページは2,000行近くもあり、これだけでも機能を理解するのは困難になっています。

Gitのコマンドに詳しいユーザでも、ちょっと標準と異なることをやりたい場合には、Gitの機能をどう組み合わせて使えば良いのか悩むことがあります。ユーザのさまざまな要求に対して良い解決手順を記述しておき、ExpandHelpの要領で検索できるようにしておけば便利です。GitHelpはGitに関するExpandHelpをコマンドラインから使えるようにしたシステムです。

たとえば「昨日のREADME.mdを見たい」と思っても、標準的なGitコマンドや引数でこれを実行することは簡単ではありません。`git log`コマンドでREADME.mdの編集履歴を参照し、昨日のコミットIDを見つけて「`git cat-file -p (コミットID):README.md`」のようなコマンドを起動すれば良いのですが、これはなかなか面倒な作業ですし操作を覚えるのはたいへんです。実はコミットIDを取得しなくても「`git cat-file -p "@{yesterday}":README.md`」というコマンドで実行できるのですが、こういうマイナーな機

能があることを知っているユーザは多くないでしょう。

GitHelpでは、「昨日の(#{pat})ファイルを表示する」という説明パターンと「git cat-file -p "@{yesterday}":#{$1}」というコマンドパターンのペアを用意しておくことにより、「昨日」「README」のようなキーワードを指定するだけで 必要なコマンドを検索してリストできます。

```
$ githelp 昨日 AD
[0] 昨日の「README.md」ファイルを表示する
   % git cat-file -p "@{yesterday}":README.md
[1] 昨日の「data/adddelete.rb」ファイルを表示する
   % git cat-file -p "@{yesterday}":data/
adddelete.rb
[2] 昨日の「data/afteradd.rb」ファイルを表示する
   % git cat-file -p "@{yesterday}":data/
afteradd.rb
$
```

ここではコマンドの候補がリストされているだけですが、-xオプションを指定すると候補を選択して実行させることができます。

```
$ githelp 昨日 AD -x0
# GitHelp

**Gitのコマンドの使いこなしを支援**

### 解決したい問題
...
```

GitHelpのヘルプデータは**リスト1**のような配列になっています。各配列要素の1行目がヘルプ文字列のパターンになっており、2行目がそれを実行するためのコマンド文字列になっています。

▼**リスト1** GitHelpのヘルプデータ

```
[
  [
    "昨日の「(#{files.join('|')})」ファイルを(表示する|見る)",
    'git cat-file -p "@{yesterday}":#{$1}'
  ],
  [
    "「(#{files.join('|')})」ファイルを(#{numbers.join('|')})日前の(もの|バージョン)と比較する",
    'git diff HEAD "@{#{$2} days ago}" #{$1}'
  ],
  [
    "「(#{files.join('|')})」ファイルをひとつ前のコミットと比較する",
    'git diff HEAD^ #{$1}'
  ],
  [
    "(#{numbers.join('|')})日前から(変化した|変わった|変更された)ファイルをリストする",
    'githelp-changed ¥'#{$1} days ago¥''
  ],
```

README.md、Rakefile、...のようなファイルがGitに管理されている環境において、ユーザが「githelp 昨日 AD」を実行すると、files()の結果が['README.md', 'Rakefile', ...]のような値になるので、最初のパターンは"**昨日の「(README.md|Rakefile|...)」ファイルを(表示する|見る)**"のようになります。この正規表現を展開した結果得られる文字列の中の「**昨日のREADME.mdファイルを表示する**」という文字列がユーザの与えたキーワード「**昨日**」「AD」とマッチするので、それに対応するコマンド「git cat-file -p "@{yesterday}":README」を実行できるようになります。

このようにGitHelpでは、やりたいことの一部を示すだけでヘルプ情報を検索し、実行まで行うことができることがわかります。

ExpandHelpの実装

今回はRubyで正規表現展開ライブラリre_expandをまず実装し、それを使ってExpandHelpを実装しています。

ExpandHelpはアイデア自体は単純なのですが、正規表現の展開が必要なのでre_expandも含めると、残念ながら実装はちょっと複雑になってしまいました。しかし、ヘルプなどの検索を行うときにGenerate and Testのような方法が有効だということに気づいたのはコロンブスの卵的だと思っています。GitHelpは実際毎日のGit活用に有用なので、ヘルプデータをもっと整備して毎日使っていこうと考えています。

GitHelpはhttps://github.com/masui/GitHelpで公開しており、「gem install githelp」で簡単にインストールできるので、ぜひご利用ください。SD



第5回 OSC群馬のドキュメンタリー

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

ドキュメンタリー? OSC群馬の3日間

これまで4回に渡って、「OSCとはなんぞや?」ということを説明してきましたが、具体的なOSCへの参加についてあまり触れていなかったように思います。ちょうど本稿締め切り直前の5月14日(土)にOSC群馬が高崎で開催されたので、筆者の行動を紹介して、OSCへ参加するとはどういうことなのか、雰囲気を味わっていただければと思います。

5月13日(金) OSC群馬開催前日

会場設営などのために開催前日に現地入りします。今回はオフィスのある渋谷から湘南新宿ラインに2時間乗って高崎に入りました。高崎駅で現地スタッフと合流して会場入り。今回は高崎駅から徒歩30秒のLABI1高崎にあるイベントホールが会場のため、過去のOSCの中でもダントツに移動が楽でした。

前日準備は次のことを行います。

・各種荷物の受け取り

事務局の荷物だけでなく、各出展者の展示物や配布物などを受け取ります。

・机、椅子などの設営

今回はセミナースペースに60脚ほど椅子を並べ、展示スペースに30台ほど机を並べました。事前に作成した図面に従って並べていきますが、現地合わせの余地も多い作業です。

・電源の配線

展示用机の塊ごとに基幹の電源を配線していきます。今回は天井からぶら下げる配線でしたが、脚立に上っての作業ですので事故のないよう慎重に作業します。

OSCでは電源タップは各出展者が持ち寄るルールです。口数などが不足した場合に、事務局から電源タップを出します。各出展者が分担することで開催コストを下げる工夫をしている部分です。

・来場者配布物の作成

来場者に渡す配布物を紙袋に詰めます。今回は100セットで済みましたが、東京の開催では1,200セット作成するたいへんな作業です。流れ作業を組んで、みんなでおしゃべりしながら楽しく実行できるように工夫しています。

そのほか、プロジェクタ出力やマイク出力のテスト、プログラムの掲出などを行って、予定どおり18時に前日準備は完了しました。

前夜祭は0次会から締めのつけ麺まで

前日準備が終わったら、運営スタッフを中心に当日運営が円滑に進むよう、スタッフの結束を高める前夜祭を行います(**写真1**)。

前夜祭が20時開始だったため、ホテルにチェックインをしたあと、純米酒専門バー「JapaneseBarKATSU」さんで0次会。いわゆる「プリインストール」という奴です(笑)。

美味しい群馬の地酒、「巖」と「流輝」を楽しんだ後、前夜祭会場に向かい「明日のOSC群馬を成功させるぞ〜!」と盛り上がったあと、「なかじゅう亭 WA-FU」で鶏清湯つけ麺で締めました。

開催当日は大忙し? でもないか

開催当日は、朝9時の会場入りからスタートです。運営スタッフの全体ミーティング、受付の準備などで10時の開場を待ちます(**写真2**)。

開場と同時に、出展者および来場者が続々と会場にやってきます。出展者はブースの準備に入りますが、やれ電源タップがない、貼り紙用の

▼写真1 前夜祭、地元組と県外組で乾杯。前の日から飲み過ぎないように!



▼写真2　OSC群馬当日運営スタッフ記念撮影。よい笑顔です



▼写真3　懇親会後、有志で2次会。群馬の地酒を飲みながら反省会



テープがほしいといったことへのサポートを行い、発表予定者のプロジェクタ接続テストなども行っていきます。

進行は、ほとんどが現地スタッフにお任せですので、13時ぐらいにはお昼休憩に出かけます。現地の方の教えに従い、「登利平」というお店の鳥めし竹重をいただきます。群馬のお弁当の定番中の定番だそうです。群馬のお弁当というと「だるま弁当」や「峠の釜飯」を思い浮かべますが、地元ならではの名物を知れるのも、OSCの楽しみの1つですね。

OSCはそのあとも順調に進行し、最後のライトニングトークが終わって無事に終了。今回は北関東で初めての開催ということもあり、185名の参加者がありました。これをきっかけに、群馬エリアでのコミュニティ活動が活発になるといいなあと思いつつ、会場の片付け、荷物を送り出して撤収完了となりました。

OSCは懇親会までが1セットです

大事なことなので何度も言いますが、OSCは懇親会までが1セットです。今回は66名が参加する大懇親会になりました。OSCの会場からエスカレーターで1フロア上がるだけ、という絶好のロケーションで、立食ビュッフェ形式・食べ放題飲み放題での懇親会です。

東京電機大学から学生スタッフがたくさん参加してくれていたこともあり、学生と社会人の交流が活発に行われたようです。途中からライトニングトークが始まったりしているうちに、あっという間に懇親会の2時間が過ぎていきました。

そのあと、有志での2次会に移動です。前夜祭で行った「Japanese BarKATSU」さんがとてもよかったので2日連続でお邪魔して、まだ飲んでいない地酒を堪能します(**写真3**)。締めは「環七ラーメン 周麺」さんで環七つけ麺をいただいて解散しました。

開催翌日は地元観光をして帰りましょう

翌日は10時にホテルをチェックアウトして、少しだけ観光をして帰ります。まず最初に向かったのは、高崎から小一時間の山中にある日本酒酒蔵「牧野酒造」さん(**写真4**)。日本酒をいくつか試飲して、4合瓶で7本ほど購入。冷蔵が必要な生酒はまとめて冷蔵便で送ってもらうことにしました。

次に倉渕温泉で温泉に浸かります。露天風呂で、野生の藤が満開なのを眺めながらの温泉は最高の贅沢です(入湯料600円)。

昼食は古民家で40年以上営業している「魚籠屋(びくや)」さんで岩魚(いわな)料理を堪能。とくに囲炉裏(いろり)で炙った岩魚を熱燗に丸ごと入れた「骨酒」は最高の味わいでした。

♣

いかがでしたでしょうか？　前夜祭から懇親会、終了後のプチ観光と、OSCとセットで楽しめるイベントが満載です。おもしろそうだな、と思った方は一緒に楽しみましょう。

SD

▼写真4　牧野酒造さんで「大盃」などを試飲。たくさん買ってしまいました



ツボイのなんでもネットにつなげちまえ道場

第13回

Why!? 技適

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

はじめに

日本で販売されている無線を使う装置には、たいてい「技適マーク」というマークと番号が表示されています。これは「技術基準適合証明等を受けた機器」に表示されているものですが、そうでない機器も売られています。そういった表示のない機器には、「本製品は技術基準適合証明等を受けていませんので、電波暗室や電波障害を起こさない十分広い敷地・建屋内で実験を行ってください。」といった注意書きが書かれていると思います。今回は、知られているようで知られていない技適、技術基準適合証明についていろいろと説明をしたいと思います。

電波法

そもそも、技適マークが存在するのは、「電波法」という法律があるからです。電波法は、電波を利用する際のルールを定めています。人混みで無線LANが快適に使えないのは、電波を発する機器が集まり過ぎて干渉を起こしているからです。ルールに従った機器が集まっても混雑して思うように機能しないのですから、ルールが必要であることは明らかです。

電波法では、電波を送受信する機器を操作する場合、原則として総務大臣の免許を受けなければならないということが定められています。「原則として」とあるとおり、たとえば無線LANやBluetoothといった無線通信は、電波法施行規則という総務省令に定められている「小電力データ通信システム」という無線設備であれば、免許を受けなくとも利用することができます。

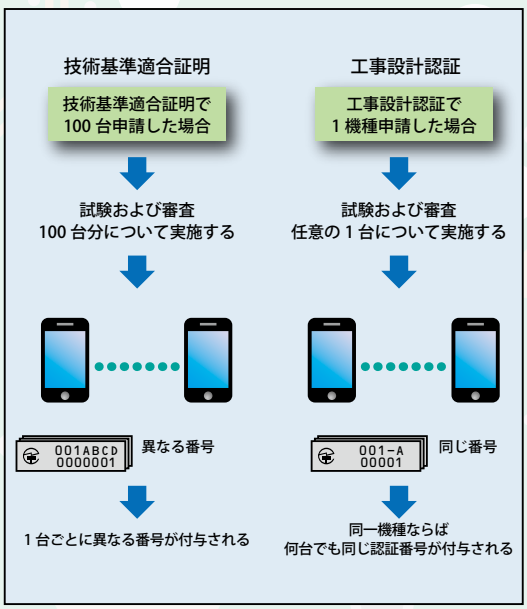
携帯電話機は、「特定無線局」という複数の無線局を包括して免許できる無線局です。この包括免許は、携帯電話事業者に対して与えられています。

技術基準適合証明と工事設計認証

電波を利用する端末を日本で利用するには、電波法などで定めている技術基準に適合している必要があります。この基準を満たしていることを証明する方法には、技術基準適合証明と工事設計認証の2種類があります。技術基準適合証明が機器1台ごとに証明するのに対し、工事設計認証は機種ごとに証明を行います（**図1**）。

技術基準適合証明は、1台ごとに異なる証明番号が付与されるのに対し、工事設計認証は同

▼図1 技術基準適合証明と工事設計認証



一機種であれば何台でも同じ証明番号が付与されます。スマートフォンなど大量生産される無線機は、通常は工事設計認証を受けます。つまり、スマホが受けている認証は、厳密には技適ではなく、工事設計認証です。総務省の文章を見ると、技術基準適合証明及び工事設計認証を併せ、「技適等」や「技術基準適合証明等」と表現されています。

個々の無線機が、電波法令で定めている技術基準に適合している無線機であることを証明するのが技適マークです。先ほど言及した「小電力データ通信システム」を例に挙げると、この技適マークが付いていることで技術基準に適合していることがわかり、無線局の免許を受けなくとも適法に使用できることがわかります。

電波法違反

では、電波法に違反するとどうなるのでしょうか。電波法では、懲役または罰金刑が定められています。ここで処罰を受けるものは、電波を使用した者であって、無線機の製造者や販売者ではないことに注意が必要です。ですので、冒頭の「本製品は技術基準適合証明等を受けていませんので、電波暗室や電波障害を起こさない十分広い敷地・建屋内で実験を行ってください。」といった注意書きを行うだけで、販売を行っている事業者が多数存在しているのが実情です。

電波法違反の取り締まりは実際に行われており、無許可でアマチュア無線を営業目的で使用した客引きが逮捕されたり、日本の技術基準に適合していない無線機を搭載したドローンの使用者が書類送検を受けたりと、事例は数多くあります。先ほど、処罰の対象者は電波を使用した者であると書きましたが、製造者や販売者が幇助（ほうじょ）の疑いで摘発を受けた事例もあります。

なお、「十分広い敷地・建屋内で実験」という点について、総務省の総合通信局に規定の有無を問い合わせてみましたが、担当者の知る限りそのような規定は存在しないという回答を得ました。日本の法律が適用される日本国内である限り、次に説明する試験設備を用意するなどの方法で技適マークのない無線設備は使用する必要がありそうです。

電波暗室

では、技適マークがない無線設備の実験を国内で行うにはどうすればよいのでしょう。平成18年3月28日に、「電波法施行規則第六条第一項第一号の規定に基づく総務大臣が別に告示する試験設備」という公示が出ています。

電波法施行規則（昭和二十五年電波監理委員会規則第十四号）第六条第一項第一号の規定に基づき、総務大臣が別に告示する試験設備を次のように定める。

電波に関する研究開発又は法及びこれに基づく命令に規定する技術基準等に対する適合性に関する試験等を行うための電波暗室その他の試験設備であって、金属遮へい体により収容され、その内部で使用される無線設備の使用周波数における漏えい電波の電界強度を四〇デシベル以上減衰させることが明らかであるもの。

電界強度を40dB（デシベル）減衰させるというのは、電波の強さを電力利得で10,000分の1にすることを意味します。dBは絶対的な量を表す単位ではなく、相対的な比を対数で表す単位です。利得（*Gain*）というのは入力と出力の比のことで、電力利得は電力[注1]の入力（P_{in}）と出力（P_{out}）の比のことです。dBは対数ですので*log*を使って表し、

$$Gain = 10\ log(\frac{P_{out}}{P_{in}})\ \mathrm{dB}$$

という計算式を使います。ですので、0dBのとき1倍、10dBは10倍、40dBは10,000倍といった具合に、10dB増えるごとに10倍になります。逆に、-20dBは100分の1と、-10dBごとに10分の1になります。対数であるdBを使うのは、

注1）電力は、電圧と電流の積で、W（ワット）という単位を使って表します。

計算に都合がよいからです。対数を使えば、実数の掛け算や割り算を、足し算や引き算で置き換えられますよね。

40dB以上減衰させるといっても、無線設備の出す電波の強さによっては十分に微弱な電波にまで減衰させることができません。ですので、「電波法施行規則」第一章第六条一に記されているように、定められている値以下でなければなりません。この定められた値は、総務省の「微弱無線局の規定」というページ[注2]にも図示されています(**図2**)。

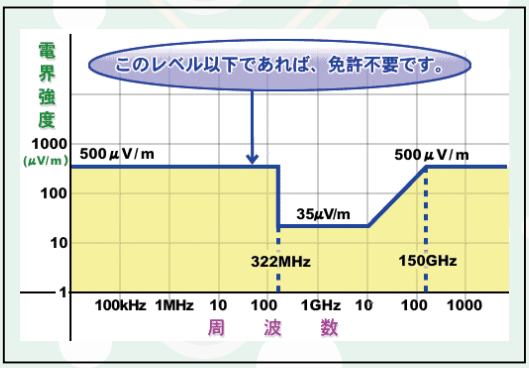
これらによると、無線LANやBluetoothで使われる2.4GHz帯の電界強度は、無線機器から3mの距離で測定したときに35 μ V/m以下であれば微弱無線局の許容値を満たしており、免許が不要になります。

電波暗室は、金属などで作られた部屋で、部屋の外からの電波の影響を受けたり、部屋の外に電波を漏らしたりしないように作られた部屋です(**写真1**)。

また、部屋の中に電波を反射しにくい電波吸収材を貼り付けて部屋の中で電波が反射しづらいようにも作られています。写真の青い四角錐が、この電波吸収材です。部屋をまるごと遮蔽された空間にするのではなく、机の上に置けるようなサイズの箱のシールドボックスというものも存在します(**写真2**)。

注2) http://www.tele.soumu.go.jp/j/ref/material/rule/

▼図2 微弱無線局の3mの距離における電界強度の許容値



※総務省「微弱無線局の規定」ページから引用

技適マークの表示

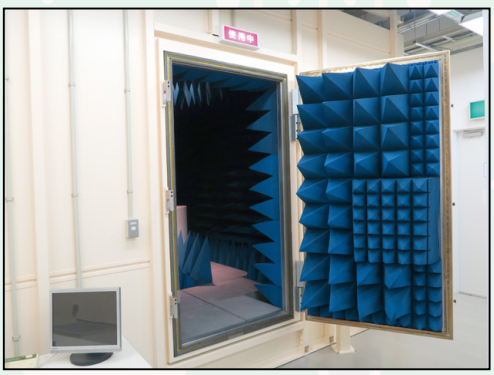
技術基準適合証明や工事設計認証に関わる試験や審査は、総務大臣の登録を受けた登録証明機関などが行います。登録証明機関には技術情報を含んだ各種書類と、テストパターンに合致する電波を出せるモードを用意した無線設備が必要になります。一般的には、メーカーや設計者でなければ、こういったものを用意することはできません。登録証明機関などは、技術基準を満たすことを試験して、問題がないと判断することで、番号を発行してくれます。この番号、工事設計認証の場合は「工事設計認証番号」は、技適マークとともに無線機に表示されます(**写真3**)。

こうした番号は、総務省のページ[注3]で検索をして確認できます。ただし、番号が発行されて直ぐに登録が行われるわけではないので、ここで検索をしても機器が見当たらない場合でも、登録が行われていないということを意味するものではありません。

技適マークの表示については、「特定無線設備の技術基準適合証明などに関する規則」の第八条などに定められています。ここでは、表示について大きさや、材料、色彩や表示する場所などについて、いくつか定めがあります。最近では、平成26年の総務省令によって、たとえばスマー

注3) http://www.tele.soumu.go.jp/giteki/SearchServlet?pageID=js01

▼写真1 電波暗室



トフォンの画面に表示する方法でもよいことになりました(**図3**)。

また、技適マークなどの適合表示を付すことが困難または不合理である特定無線設備、表示を付す面積が確保できない端末機器にあっては、当該特定無線設備(当該端末機器)に付属する「取扱説明書 及び 包装 又は 容器の見やすい箇所」に付すことが可能となりました。工事設計認証の場合、こういった表示については、製造者によって判断されるとのことです。個別具体に適法かどうかということを総務省に問い合わせをしたことがあるのですが、最終的にそれが適法かどうかは総務省が判断することではなく、裁判所が判断することであるという言われてみれば当たり前の回答をいただいたこともあります。

ローミング

従来より、電波法第百三条の五にあるように、技適マークのない携帯電話機でも海外通信事業者のSIMを使ってローミングする場合には、日本の携帯電話事業者の包括免許で適法に使用できます。ただし、同じ携帯電話機に搭載されているWi-FiやBluetoothなどは、適法に使用できないという問題がありました。

この5月21日から施行された「電気通信事業法等の一部を改正する法律」によって、入国から90日間は、指定されている認証マークが付いていれば、技適マークの付いていない携帯電話機のWi-FiやBluetoothを適法に使用できるようになりました。また、ローミングが前提だった技適マークの付いていない携帯電話機の使用も、国内の携帯電話事業者が総務大臣の許可を得ることで、国内の通信事業者のSIMを使って適法に使用できるようになりました。

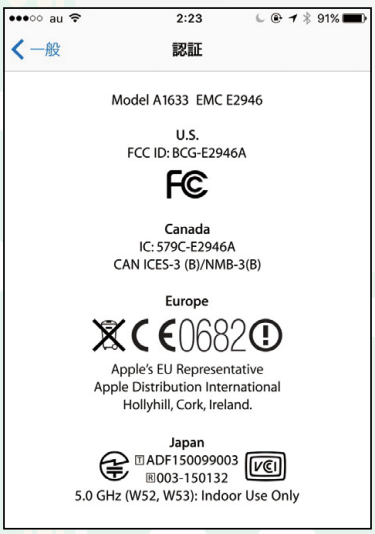
▼写真2　シールドボックス



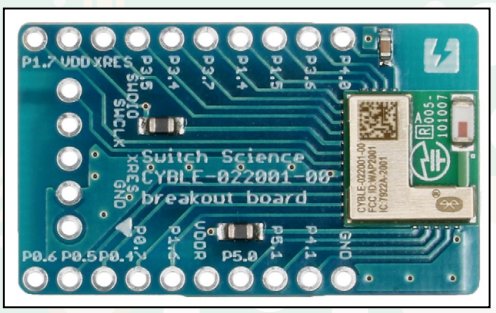
まとめ

意外と知られていない電波法令について筆者の知っていることをまとめてみました。法令の解釈には常に議論が伴い、適法か違法かについては裁判所が判断するものです。とはいえ、現実的には、総務省や総合通信局に相談するのがよいでしょう。総務省と聞くと身構えてしまいがちですが、筆者の経験の限りでは、とても親切に応対してくださいます。SD

▼図3　iPhoneで技適マークを表示してみた



▼写真3　技適マークの表示例



読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2016年7月14日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

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

01 HHKB Professional BT（日本語配列）



HHKB（Happy Hacking KeyBoard）Professionalシリーズのキータッチはそのままに、Bluetooth（Ver3.0）に対応したキーボード。デスクトップ／ノートPC、タブレット、スマートフォンなど最大４台の機器との併用ができます。Bluetooth HIDに対応した機器とはレシーバなしですばやくペアリングできます。給電は単三乾電池２本、またはUSBコネクタ（micro-B）から。

提供元 PFU http://www.pfu.fujitsu.com

1名

02 USBアナログメーター



「内蔵HDDの温度」「メモリ（主記憶）使用量」「CPU使用量（平均）」がアナログ針でひとめにわかるガジェットです。USBで接続し、専用のソフトをPCにインストールして起動させておくことで動作します。対応OSはWindows10/8.1/8/7/VISTA/XP。

提供元 MARSHAL http://www.marshal-no1.jp

1名

03 Vivaldi Tシャツ&ステッカー



2015年に発表された、アイスランド生まれの新しいWebブラウザ「Vivaldi」。特徴的なUIと、細かいカスタマイズが可能なことが特徴です。今回はVivaldiのノベルティTシャツ（Mサイズ）とステッカー３種をプレゼントします。

提供元 Vivaldi Technologies https://vivaldi.com

1名

04 Unix考古学

藤田 昭人 著



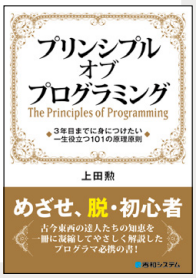
Unix開発の歴史を、当時の社会情勢や技術動向も確認しながら紐解いた１冊。SBクリエイティブの月刊誌『UNIX USER』の連載「Truth of the Legend - UNIX考古学」を再編集したものです。

提供元 アスキードワンゴ http://asciidwango.jp

2名

05 プリンシプル オブ プログラミング

上田 勲 著



なかなか「良いコード」を書けないと悩みを持つ若手エンジニアに向け、良いコードを書くうえで指針となる前提・原則・思想・哲学といった「プリンシプル」を解説する、プログラミングスキル改善書です。

提供元 秀和システム http://www.shuwasystem.co.jp

2名

06 Amazon Web Servicesクラウドネイティブ・アプリケーション開発技法

佐々木 拓郎 ほか 著



クラウドベースのアプリ開発において、AWSのサービスを有効活用するための実践的テクニックを解説。サーバ構築から一歩進んだ最新のAWS活用術を、わかりやすい手順と実例で身に付けられます。

提供元 SBクリエイティブ http://www.sbcr.jp

2名

07 基礎からのWebアプリケーション開発入門

前橋 和弥 著



Webサーバを作りつつ、実際に動かして結果を見ながら、HTTP・Webサーバ・サーブレット・JSP・Cookie・セッション・プロキシサーバ・TLS・認証・DOM操作・Ajaxといった技術を解説します。

提供元 技術評論社 http://gihyo.jp

2名

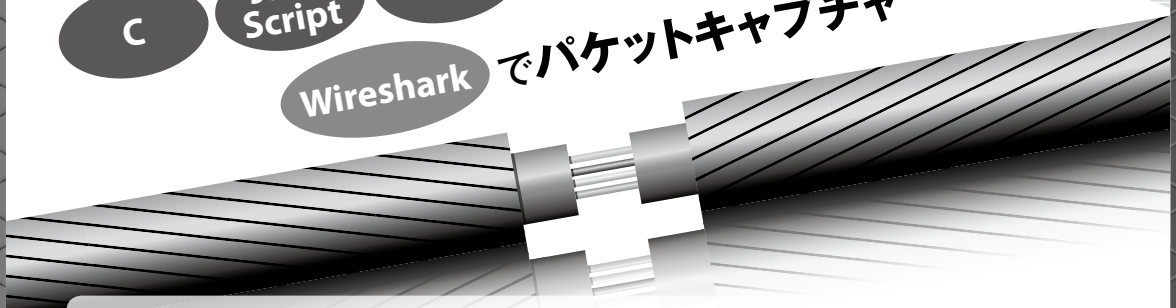
第1特集

試して実感！

プログラマが知っておくべき TCP/IP

C Java Script PHP Python Ruby でコーディング

Wireshark でパケットキャプチャ



アプリケーションを開発するとき、通信部分はどのように実装していますか？　ライブラリやフレームワークを使えば、ネットワークのしくみを意識せずに、しかも数行でその処理が書けてしまいます。ただ、裏で動いているデータ処理を理解せずに使っていると、より高度な通信処理を実現したいときやトラブルが発生したときに、何をすれば良いのか途方に暮れてしまうことでしょう。

本特集でネットワークの基礎を知り、TCP・UDP通信の流れをコードで追い、流れるパケットを目で見ることで、いざというときに使えるネットワークの知識を実感しながら身につけましょう。

序章　プログラマがTCP/IPを知っておくべき理由 …… P.18
Author 及川 卓也

第1章　TCP/IPって何?～ネットワークの基礎の基礎 …… P.23
Author 網野 衛二

第2章　実践ネットワークプログラミング～C言語編～ …… P.35
Author あきみち

第3章　実践ネットワークプログラミング～スクリプト言語編～ …… P.46
Author 中島 雅弘、植西ゴロー由謹、國信 真吾

第4章　パケットキャプチャで学ぶTCP/IP …… P.59
Author 吉田 英二

序章

プログラマがTCP/IPを知っておくべき理由

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

最近、Webとネットが同義語になったと感じることがあります。

「ネットがつながらない」と言う友人の環境を聞いてみたら、接続しようとしているサイトに対するDNSのAレコード[注1]も引けて、IPアドレスへのpingも通っているのにもかかわらず、Web接続ができなかっただけだったということがありました。また、このSoftware Designの読者のような技術者でも、Googleなどの Web検索サービスで検索することを「ネットで検索して」と言ったりすることもあるでしょう。

このように、ネット利用（厳密にはインターネット利用）の多くはWebとなっています。そしてそれに合わせるように、プログラミングの世界でも、HTMLやCSS、JavaScriptによるWebプログラミングの利用範囲が広がっています。

Webプログラミングはブラウザから利用されるアプリケーションだけではありません。スマートフォンアプリケーションの世界でもApache Cordovaなどのマルチプラットフォーム対応の開発環境やWebViewを使ったハイブリッドアプリケーションも普及しており、Webプログラミングの応用範囲はますます広がっています。

このようなWebプログラミングは、ともすれば上位レイヤだけで完結すると思われがちですが、実際にはWebプログラミングであっても、下位レイヤのTCP/IPの知識は不可欠です。むしろ、ますます必要になっていると言っても過言ではありません。今回の特集では、その理由とそのために必要な知識を解説します。

最初に、WebプログラミングにおいてTCP/IPの知識が必要な理由を考えてみたいと思います。



複雑化するWeb技術

まず、皆さんはWeb技術と聞いてどのようなものを思い浮かべるでしょうか？ HTMLやCSS？ それにJavaScript？ jQueryなどのJavaScriptライブラリ？ TypeScriptやCoffeeScriptなどのAltJS？ AngularやReactのようなフレームワーク技術？

フロントエンド技術としては、このようなものを思い浮かべる方が多いでしょうか。さらには、Webアプリケーション開発として、PHPやRuby on Railsなどのサーバサイドの言語やフレームワークを想像する方も多いでしょう。あなたがインフラエンジニアだったら、HTTP通信を行うWebサーバやその下回りの技術もWeb技術として考えるかもしれません。

このように、単にWeb技術と言っても、実際には多くの技術がそれに含まれ、1つ1つの技術も複雑化しています。



複数の技術で構成されるWeb技術

たとえば、Webのパフォーマンス向上について考えてみましょう。ブラウザのアドレスバーにURLを入力し、サイトのコンテンツがブラウザ内に表示されるという基本動作にも、多くのレイヤの処理が関与しています。

注1）Address record。DNSが持つリソースレコードのうち、ホスト名とIPアドレスの対応が記述されているもの。

まず、ドメイン名からIPアドレスを引くためにDNSを利用しますし、次にIPアドレスに対し、HTTP GETリクエストを投げ、リソースをレスポンスとして受け取ります。ここまでの最初の処理の部分だけでも、いくつかTCP/IPの知識があるかないかでパフォーマンスの最適化やトラブルシュートの際に差が出ます(**図1**)。

あるサイトに複数のサービスがある場合を考えましょう。ブランディングを考えると別ドメインや別サブドメインで展開したくなるかもしれません。www.google.comというWeb検索とは別に地図サービスをmaps.google.comで提供するケースなどです。この別ドメインでの提供という方法は独立したサービスのブランディングやユーザの識別のしやすさという意味では有効ですが、一方、異なるドメイン名になりますのでDNSへの問い合わせは別途発生することになり、パフォーマンスは悪化します。

したがって、あなたが今制作中のHTMLからこれらのサービスへのリンクを掲載する場合、それがあまり利用頻度の高くないサービスや単に情報を掲載しているページならば、別ドメインではなく、www.google.com/chromeのようにパスで提供したほうが少なくともパフォーマンスの面では有利です。

とても些細なことと思われるかもしれませんが、Webパフォーマンスはこのように小さな工夫の積み重ねで改良することが重要です。これなどはシンプルな例ですが、Webの通信という一連の流れが実際にはどのような処理により成り立っているかを知っているか知らないかで違いが出てきます。

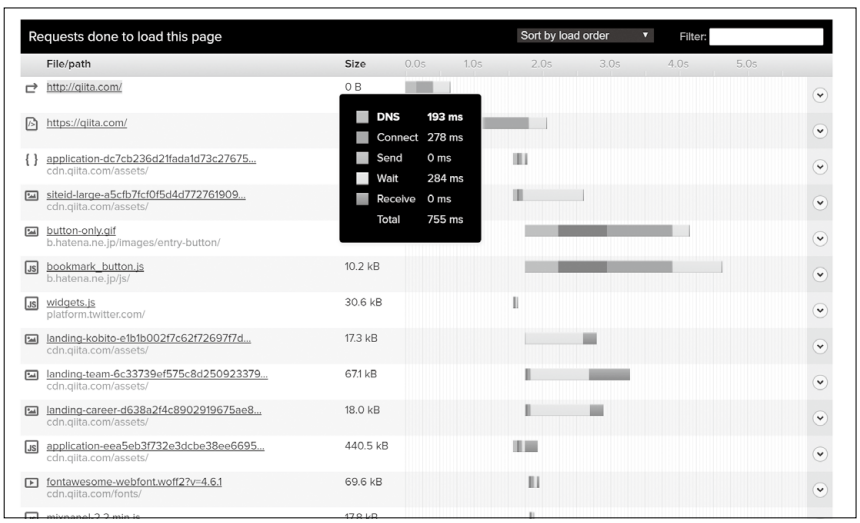


さまざまな通信技術に変容するWeb技術

また、Web技術がHTTP以外での通信を行うようになってきているという状況もあります。

たとえば、いわゆるHTML5と言われる技術の1つとして、WebSocketというものがあります。従来のWeb技術では、ブラウザとWebサーバ間で双方向通信は本当の意味ではできていませんでした。Cometと言われる、あまりエレガントといえない方法が開発され、それを利用することでチャットやドキュメントの共同編集、サーバ側での更新を即座にブラウザに反映させることなどが可能となっていました。しかし、効率は悪く、開発も複雑化するため、UNIXのソケット通信と同じように真の双方向通信を実現する方法としてWebSocketがWeb技術の標準化を行う場であるW3C(World Wide Web

▼図1　Pingdom Website Speed Testでhttp://qiita.comを計測した結果



Consortium)にて提唱され、ブラウザやWebサーバに実装されました(**図2**)。

WebSocketのAPIはデータの送信と受信を行うというシンプルな作りになっているため、単純なアプリケーションであれば開発は容易です。しかし、APIの簡易さとは裏腹に、通信において知らなければならないことがいくつかあります。

WebSocketを用いた通信は、最初はHTTPで接続され、クライアントであるブラウザとWebサーバ間でハンドシェイクされた後にWebSocketにアップグレードされます。

WebSocketにアップグレードされた後はHTTPとは異なるWebSocketとしての通信となりますので、ブラウザとWebサーバの間に介在するさまざまな通信機器がWebSocketの通信をブロックしてしまうことがありえます。そのために、SSL通信を使ったWebSocketを用いることが一般的なのですが、それでもHTTP(もしくはHTTPS)に比べて不安定なことは否めません。あなたのWebアプリケーションやユーザの通信環境によっては、HTTPへのフォールバックを考えなければいけないことがあります。

WebSocketの標準化の歴史を見ても、プロトコルの複雑さはわかります。標準化の最初はW3Cでのみ議論されていたのですが、上で説明したように、ほかの通信機器にも関係しますし、Web関係者が考える以上の用途があり得ることもわかったため、APIの標準化はW3Cで、通信プロトコルの標準化はIETF(Internet Engineering Task Force)で進められることになったのです。

このように、一昔前まではWebと言えば、その通信はHTTPしかありえなかったのですが、最近ではWebであっても、HTTP以外のプロトコルが使われるようになっています。

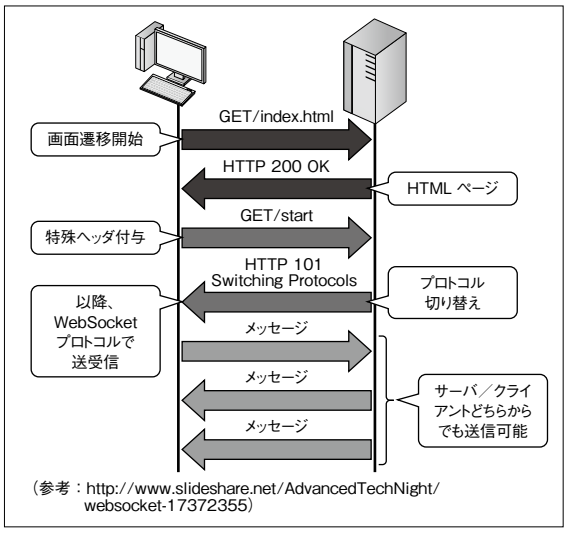
Webでリアルタイムコミュニケーションを実現するWebRTCなどもその典型例です。WebRTCはクライアント間でピア・ツー・ピア通信を行うのですが、これもHTTPとは異なる通信が用いられます。呼制御、つまりピア・ツー・ピアを確立するまでの部分などはむしろSIP(Session Initiation Protocol)やNAT Traversalと言われるNAT越えのためと同様の技術が使われています。

また、将来のWeb APIとして提案されている仕様には、UDPやTCPのソケット通信を実現するものさえあります。

WebSocketやWebRTCは新しいWebアプリケーションのための通信プロトコルと考えることもできますが、通常のWebアクセスであっても、HTTPの再定義やWeb通信を最適化するための新しいプロトコルも提唱されています。これに一番熱心なのが、Googleです。

HTTP/2として標準化されたHTTPの最新バージョンはGoogleがもともとSPDYという名前で提案していたものを基礎にしています。HTTPストリームという概念を導入し、その多重化によるセッションのパイプライン化や優先的なローディングを可能にしています。優先的なローディングなどは、今までWebプログラミング上の工夫で対処していたものを通信レ

▼図2 WebSocketの通信概略図



ベルで可能にする技術です。また、GoogleはTCPの置き換えを狙うQUICというプロトコルも現在実験中です。

このようにさまざまな通信技術が複雑に入り乱れる様相になってきたWeb技術を使いこなすには、WebプログラマであってもTCP/IPの知識は必要となってきています。



実務で必要となるTCP/IPの知識

ここまで変わりゆく技術の側面からTCP/IPの知識が必要となるケースを見てきました。ここからは現時点でTCP/IPが必要となる状況を考えてみましょう。



トラブルシューティングの際の需要

まず考えられるのが、トラブルシューティングにおける必要性です。すでに説明したように、Web技術は複雑化しています。そのため、それを使ったシステムの障害発生時には、各レイヤでの原因を疑う必要が出てきます。ブラウザに装備されている開発ツール(Chromeであればデベロッパーツール)だけでなく、ネットワーク診断ツールなどを使う必要も出てきます。パケットキャプチャでキャプチャしたログなどを読み解く能力も必要です。経験した方はわかると思いますが、このようなツールはキャプチャするタイミングや何をフィルタして絞り込んでいくかなど、ある種職人的な技が必要になります。日頃から、Webアクセスの配下で何が起きているかを知るために、たまにはこのようなツールを動作させてみると良いでしょう。

筆者がかかわったあるWebサイトでも、あるユーザからページのロードが遅いという連絡がありました。筆者の環境からは問題がなかったため、ユーザにWireshark(Windowsのネットワークキャプチャツール)を実行し、そのログを送ってもらいました。結果、ある特定のリソースにIPv6でアクセスし、それが重くなっていることがわかりました。そのリソースは外部のリソースだったため、担当者にIPv6アクセスの際の挙動を確認してもらい対応したという経験もあります。

また、皆さんも「技術者」だというだけで、知人や親戚から、彼らのコンピュータ環境に不具合が発生した場合にヘルプを求められることも多いのではないでしょうか。筆者は10年以上前にMicrosoftに勤務していたことがあるのですが、それだけでWindowsにトラブルが生じると、「あなた詳しいでしょ？」とその解決を求められることがあります。確かに、十数年前にはWindowsの開発を担当していましたが、その後のバージョンのことは詳しくありません。正直、「同じWindowsっていうだけで乱暴にくくるなぁ」と思わなくもないですが、ネットワーク周りのトラブルなどはWindowsのバージョンに依存するような障害よりも、むしろOSに限らず問題になりやすいところが問題になっていることのほうが多いように思います。そのような際にも重宝するのが、一般的なネットワークのトラブルシューティングの知識です。これなどは、Webプログラマとして必要というものでは必ずしもないですが、知人との関係を良好に保つためにも、広く技術は知っておくほうが良いという例かもしれません☺



フルスタックエンジニアとして

トラブルシューティング以外にも、TCP/IPの知識は、いわゆる「フルスタックエンジニア」と呼ばれるエンジニアになるならば必要不可欠です。フルスタックエンジニアは希望してなるものでなく、いろいろな仕事を押し付けられていたらいつのまにかフルスタック担当だったということもあるかもしれませんが、すべてのレイヤの作業を1人もしくは少人数で担うことが珍しくなくなってきている昨今、TCP/IPを知っているかどうかは個人や組織の生存戦略という意味でも重要になってきているでしょう。

フルスタックエンジニアのフルがどこからどこまでを指すかの正式な定義はありません。業

務がWebサイトやWebアプリケーションでしたら、サーバの構築からWebのフロントエンドからバックエンドまでを含むことが一般的でしょう。各社の求人情報を見てみると、このWeb側の人材でフルスタックを求めているところが多いようです。スマートフォンアプリケーションの開発では、あまりサーバ側の知識や経験が求められていないのを見ると、Webを専門にしてしまったことを後悔するかもしれませんが、諦めましょう☺むしろ業務を通じて幅広い領域の技術を学べる機会を得られたと思い、この際、ネットワーク部分の専門性を高めるのも良いでしょう。

フルスタックエンジニアとしても、とりあえず動けば良いというレベルから、システムに最適な技術を選択し、セキュリティやパフォーマンスに対する妥協を一切しないというレベルまで、さまざまなレベルが存在します。デモやプロトタイプならば動けばとりあえずは良いかもしれませんが、本格的なシステムを開発するならば、ぜひ後者を目指したいところです。



異分野への進出

最近、Web技術がさまざまな分野で使われます。Cordovaなどでスマートフォンのネイティブアプリケーションなどにも使われるようになったことはすでに紹介しましたが、もうひとつ注目されている分野として、IoTがあります。IoTの中でもWeb技術を用いるものをWoTと呼びます。IoTとWoTの違いについては、ここでは本題ではないので省きますが、要はWeb技術を用いてIoTを実現するものです。WoTよりもIoTのほうが馴染みがあると思いますので、ここでは違いについてはこだわらず、IoTと呼ぶことにします。

筆者の周りを見る限り、Web技術を用いてIoTを行うことは、従来のWeb技術者がIoTに進出するよりも、組み込み技術者がWeb技術を学んでいるケースのほうが多いように思います。筆者の知人にも、カーナビゲーションを開発していた技術者がいますが、彼はAndroid登場とともにスマートフォンアプリケーションの開発を学び、その関係の職を得て、その後、数社を経て、今は通信キャリア系の会社でIoT関連の開発をしています。彼は最初はCでの組み込み技術者だったのですが、Web系の技術も習得し、今ではIoT関連でのフルスタックエンジニアと呼ぶに相応しい技術者となっています。

同じように違う分野の技術を広げることはWeb技術者にも可能です。IoTシステムでは、デバイスがセンサーなどで取得したデータをクライアントとしてサーバ側に送信するという使われ方だけではありません。逆に、データが必要になった時点でほかのシステムがリアルタイムにデバイスからデータを取得することもありえます。この場合は、デバイスにWebでアクセスできるインターフェースを設け、そこからデータを吸い上げるというのも1つの方法です。具体的には、Webサーバをデバイスで動作させることになります。HTTPやWebSocketなどのプロトコルをサポートし、RESTfulなAPIを用意することで、Webで使われているのと同じ方法論でデバイスを制御できます。

同様のWeb技術と言っても、やはりIoTならではの注意事項があります。限られたシステムリソース(メモリやストレージ領域)を考え、消費電力を抑えなければいけないという場合、TCP/IPレベルでの最適化が必要であったり、トラブルシューティングやデバッグではTCP/IPの知識が必要になることも多いでしょう。

IoTのように注目されている領域を例にとりましたが、どのような領域でも、今後はWeb技術が使われることが多いでしょう。その場合もIoTと同じように通常のWebとは異なるので、TCP/IPの知識があるかないかで、Web技術の適用の成否は変わってきます。

◆ ◆ ◆

以上、WebプログラミングでもTCP/IPが必要な理由について、技術面およびその必要とされる場面の2つの観点から解説しました。SD

第1章

TCP/IPって何? ～ネットワークの基礎の基礎

Author 網野 衛二(あみの えいじ) http://www5e.biglobe.ne.jp/~aji
Twitter @a2amino

ネットワークには、データのやりとりのしくみを仕様として策定した「モデル」、モデルを形成する階層関係に分かれた「レイヤ」、各レイヤで動く「プロトコル」、各プロトコルで扱うデータ構造「PDU」という概念があります。本章ではそういったネットワークの全体像を、TCP/IPモデルの第2層・第3層を中心にみていきます。

OSI参照モデルとTCP/IPモデル

ネットワークモデルとOSI

ネットワークを理解するためにまず必要なこととして、アプリケーション間でデータを送信し、受信するためにはどのような段階を踏み、どのような手順で実行するかを知る必要があります。

それまで、これらのしくみはベンダごとに作られていました。それをベンダに依存しない仕様として策定しようとしたのがISO(International Organization for Standardization)で、策定したものがOSI(Open System Interconnection)参照モデルです。「オープン」、つまりベンダに依存しない形で、システムを「インターコネクション(相互接続)」するために「参照」するモデル、ということです。

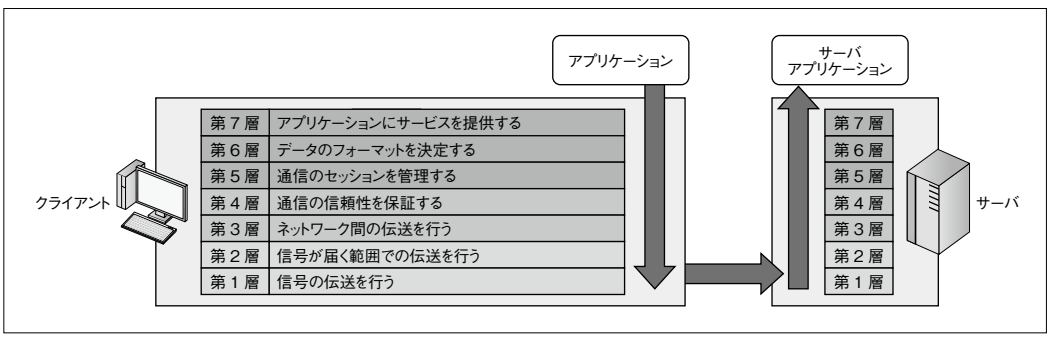
OSI参照モデルはネットワークを7つの段階に分解し、それぞれ「層(レイヤ:Layer)」と名付けています。このレイヤは1～7までの番号が付けられており、アプリケーションに近い側から7、通信のケーブル・信号の側が1と順に並んでいます。

送信側はアプリケーションから受け渡されたデータを、レイヤ7からレイヤ1の順番で段階を踏むことでデータを送信し、受信側はレイヤ1からレイヤ7の順番で段階を踏むことでアプリケーションにデータを渡します。これにより、データをアプリケーションからアプリケーションへ渡すことができるようになります(**図1**)。

OSI参照モデルの特徴

OSI参照モデルの各レイヤには、そのレイヤの役割を担うためにプロトコルや機器が割り当

▼図1 OSI参照モデル



てられます。それぞれのプロトコルや機器は、そのレイヤの役割のみを行うためにあります。また、それぞれのレイヤはそのレイヤ内で閉じており、ほかのレイヤに影響しません。

レイヤはアプリケーション側、つまり上位のレイヤに対して自らの役割を実現します。レイヤごとに自らの役割を果たすことを積み重ねることで通信全体が行われることになります。実際面で言えば、レイヤ1・2は通信インターフェース、つまりLANボード・LANカード、無線LAN子機などが担います。レイヤ3・4はOSの通信機能が、それ以上は通信アプリケーション側で実装します。

OSI参照モデルにより通信の手順・段階が分解されることで、トラブルシューティングやアプリケーション開発が容易になります。それぞれのレイヤの役割だけを実装することで、ネットワークアプリケーションが開発できたり、トラブルの切り分けが可能になったりします。

TCP/IPモデル

OSI参照モデルはISOが策定したものですが、実際の実装面ではあまり普及しませんでした。実際に大きく普及したのはTCP/IPプロトコルスイート(群)を持つ、TCP/IPモデルです。

TCP/IPモデルはOSI参照モデルとは別の組織により作られたモデルで、OSI参照モデルと直接的な関連はありません。ですがどちらも、「ネットワークの機能と段階をモデル化する」という目的で作られているため、とても似通ったモデルになっています。

TCP/IPモデルは4つのレイヤで階層化されており、OSI参照モデルとの対比で言えば、OSI参照モデルのレイヤ7・6・5とレイヤ2・1が、TCP/IPモデルではまとまったレイヤとして扱われている形になります(**図2**)。

我々が実際に触るものは、TCP/IPモデルとそのプロトコル・機器です。ですが、教育面ではOSI参照モデルがよく使われており、理論を学ぶ際はOSI参照モデルが多いです。理論と実装で使うモデルが食い違うというところがありますので注意が必要です。

パケットとカプセル化

パケットとPDU

ネットワークを使った通信により、データを送信側と受信側でやりとりします。この主体となるのは、要求や応答のデータそのものです。たとえば、WebならばURIを含むリソース要求やその応答であるファイルなどのリソース情報、メールならばメール送信のための情報や、その確認の応答といったものです。さらに、主体となるデータ以外にも「データを運ぶためのデータ」が必要となります。たとえば、宛先を特定するアドレスであったり、エラーチェック用のコードであったり、データの中身を指定する値であったりします。

よって、ネットワークで運ばれるデータは、「データ」と「データを運ぶためのデータ」から構成されることになります。この2つをまとめたものを表現する言葉として、パケット(packet)を使います。または、これらの「データ」と「データを運ぶためのデータ」はプロトコルによって決められますので、PDU(Protocol Data Unit)とも呼びます。一般的にパケットにはほかの用法もあるため、ここではPDUを使用します。

PDUとその中身は実際に運ばれるデータそ

▼図2　OSI参照モデルとTCP/IPモデル

OSI参照モデル		TCP/IPモデル	
第7層	アプリケーション層	第4層	アプリケーション層
第6層	プレゼンテーション層		
第5層	セッション層		
第4層	トランスポート層	第3層	トランスポート層
第3層	ネットワーク層	第2層	インターネット層
第2層	データリンク層	第1層	アクセス層※
第1層	物理層		

※ネットワークアクセス層、ネットワークインターフェース層とも呼ばれる

のものです。通信アプリケーションではこの中身を理解し、プロトコルで決められたとおりに構成する必要があります。

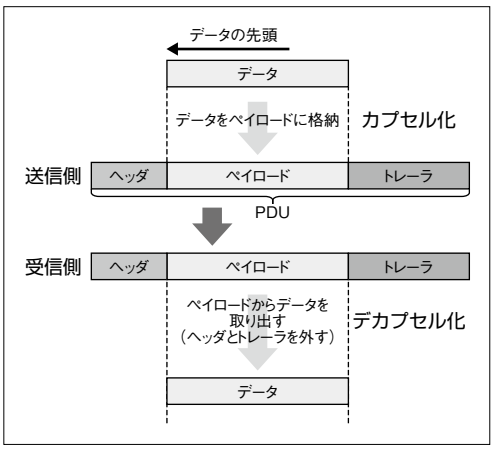
ヘッダとトレーラ

PDUはプロトコルによって決められています。PDUは大きく分けて3つの項目から構成されています。先頭から――つまりPDUを通信で運ぶ際に送信する順番が先の順から――ヘッダ(header)、ペイロード(payload)、トレーラ(trailer)になります。

まずペイロードは、送信されるデータそのものになります。一方、ヘッダとトレーラは「データを運ぶためのデータ」の部分になります。つまり、運ばれるデータであるペイロードをヘッダとトレーラではさむことになります。このことは、ペイロードをカプセルにつつむ形のようなので、カプセル化(encapsulation)と呼ばれます。

またカプセル化されたPDUですが、宛先に届くと「データを運ぶためのデータ」のヘッダとトレーラは必要なくなります。そのため、ヘッダとトレーラを削除して、カプセル化を解除して中身を取り出します。これはデカプセル化(decapsulation)と呼ばれます(**図3**)。なお、トレーラは使用するプロトコルによっては使用しないこともあり、その場合はヘッダとペイロードだけになります。

▼図3　カプセル化とデカプセル化

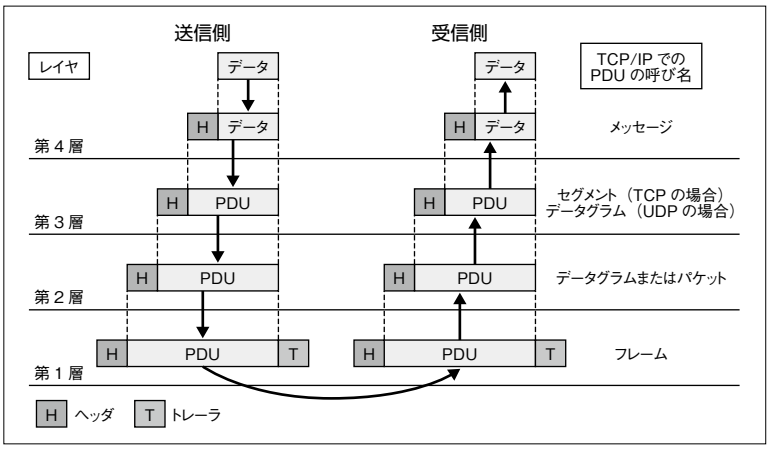


カプセル化とモデル

PDUとカプセル化は、モデルと関係が深い項目です。送信側のアプリケーションによって作成された送信データは、モデルの各レイヤを順番に下っていくことになります。

この際、レイヤごとに上位レイヤから渡されたデータがPDUのペイロードに格納されます。そしてそのレイヤで使用するプロトコルによってヘッダとトレーラを付けてカプセル化され、さらに下位のレイヤに渡されます。下位のレイヤはそのPDUをペイロードに格納し、さらにカプセル化します。

▼図4　TCP/IPモデルとカプセル化



一方受信側は、受信したPDUを下位から上位のレイヤを順に渡していきます。その際、下位のレイヤはデカプセル化し、ペイロードから上位レイヤのPDUを取り出して上位レイヤに渡すことになります(**図4**)。つまり、PDUはレイヤごとにその中身が異なります。区別するため、「レイヤ

3PDU(L3PDU)」のように頭にレイヤ番号を付けて呼ぶこともあります。また、TCP/IPの場合は、プロトコルごとにPDUに名前を付けています。



プロトコル



通信規約

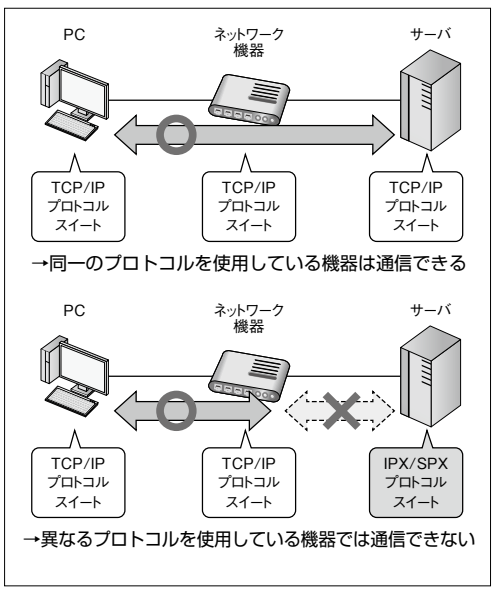
OSI参照モデルまたはTCP/IPモデルによって、通信の機能と手順はレイヤという形で分割されています。この各レイヤの機能を表現、実現するものがプロトコル(protocol)です。

プロトコルは日本語では「通信規約」などと訳されることが多いものです。その日本語訳のとおり、そのレイヤの通信機能のルールであり、このルールに従うことでレイヤの機能を実現します。

プロトコルはレイヤごとに存在します。つまり、1つのプロトコルだけで通信を行うわけではありません。複数のプロトコルを使うことになります。たとえるならば「手紙を相手に送る」ということに対して、「宛名の記述ルール」「ビジネス文書のルール」「封筒のルール」「便箋のルール」「郵便配達人のルール」のように、その手紙を送るための機能ごとにそれぞれのルールがあるのと同じようなものです。通信のルールとしてのプロトコルも、その機能ごとに存在しているのです。

また、異なるプロトコルを使用する機器同士では通信ができません。通信の間に入るルータやファイヤウォールなどの機器も、そのプロトコルを使用できないといけません。そのプロトコルを使用できることを「プロトコルを喋る」などと表現することがあります(**図5**)。

▼図5　プロトコル



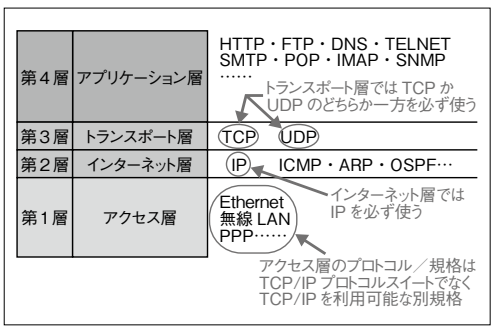


プロトコルスイート

プロトコルは闇雲に作られているわけではありません。また、複数のプロトコルをレイヤごとに使用しますが、適当なプロトコルを選んで使うわけではありません。

プロトコルは、モデルの設計思想により作られています。つまり、そのモデルを基に作られることになります。結果として、モデルのレイヤごとにその機能を実現するプロトコルが作られていくわけです。このように、同一のモデルから作られたプロトコルの集まりをプロトコルスイート(suite)と呼びます。たとえば、TCP/IPモデルを基に作られたプロトコルスイートはTCP/IPプロトコルスイートと呼ばれます。通信を行う際は、このプロトコルスイートからレイヤごとにプロトコルを選び、使用することになります(**図6**)。

▼図6　TCP/IPプロトコルスイート



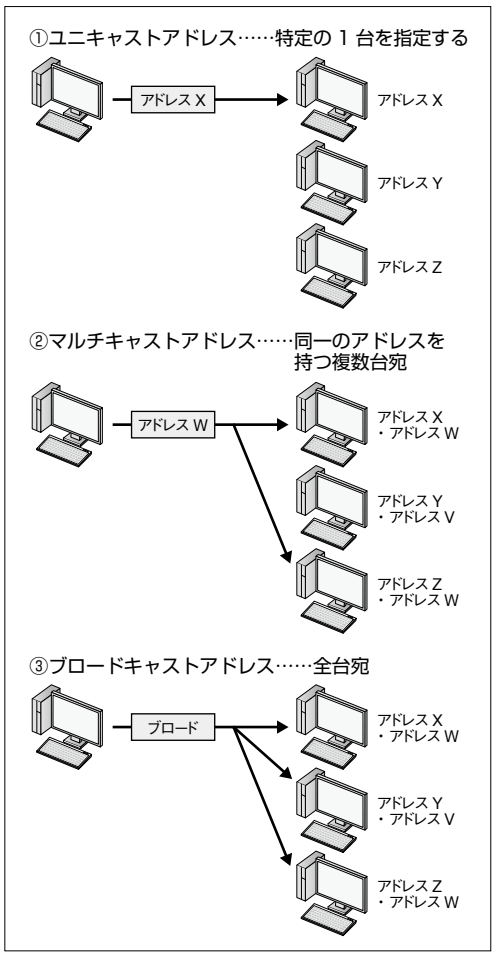
プロトコルの役割

プロトコルはそのレイヤの役割を果たすために、いくつかのルールが決められています。

まず、そのPDUの構成を決めています。ヘッダやトレーラの中身、サイズなどです。また、そのペイロードの中身が上位PDUではない場合、ペイロードに配置するデータの構造なども決めています。

さらに、送信する手順も決めています。これはたとえば、「データが含まれているPDUを送信した場合、受信側はそのPDUを受信したことを通知する応答を返す」「Aというデータの次はBを送信する」などのような手順です。

▼図7　アドレスの種類



3つのアドレス

アドレスとは

ネットワークでは宛先へデータを送るために、宛先を特定する何らかの情報が必要となります。これを「アドレス(address)」と呼びます。

一言にアドレスと言っても、ネットワークのレイヤによって、それぞれ指すべき「宛先」が異なります。つまり、「複数のアドレスを使用する」必要があるということです。

もう1つ、アドレスのユニーク(unique)性があります。同一のアドレスを持つ宛先が複数あった場合、どの宛先であるかの特定ができなくなります。よって例外を除き、アドレスはほかに同一の値がない必要があります。これを一意性・ユニーク性と呼びます。ただし、このユニーク性は「どの範囲でユニーク性を持たなければいけないか」というのがポイントです。同じネットワーク内のみでユニークなのか、インターネット上でユニークなのか、ということが重要になります。

そして現実の住所ではありえませんが、1つのアドレスで複数を指すことができるアドレスがあります。特定のグループを指すマルチキャスト(multicast)アドレス、範囲内の全機器を指すブロードキャスト(broadcast)アドレスがあります。また、特定の1台を宛先とするものをユニキャスト(unicast)アドレスと呼びます(**図7**)。

レイヤ2アドレス

レイヤ2の役割は、同一の信号が届く範囲内でのデータの伝送を制御する役割です。そこで使われるアドレスは、まずその「同一の信号が届く範囲」でユニークである必要があります。一般的に言えば、ハブ・スイッチングハブ・アクセスポイントによってつながれた、ルータで分断されない範囲、つまりネットワークでのユ

ニーク性があれば良いことになります。

Ethernet、無線LANではレイヤ2アドレスとしてMAC(Media Access Control)アドレスが使用されます。MACアドレスは機器のインターフェースに設定された48ビットの値です。

レイヤ3アドレス

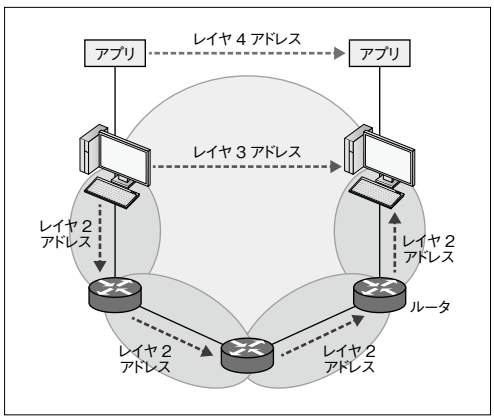
レイヤ3アドレス、TCP/IPでいえばIPアドレスです。レイヤ3は接続されているネットワーク間での伝送を担うため、そのアドレスは「接続されている全ネットワーク」でユニークである必要があります。それがインターネットなら「全インターネットでユニークなアドレス」です。

ただし、レイヤ3アドレスは「その機器が所属するネットワークのアドレス」と「その機器のアドレス」の2つを組み合わせた値であり、ネットワークのアドレスは接続されている全ネットワークでユニークな値を、機器のアドレスはそのネットワーク内でユニークなアドレスを使用します。

レイヤ4アドレス

レイヤ4のアドレスは「機器のアドレス」ではありません。機器ではなくその「機器内のアプリケーション」を識別するためのものになります。よって、レイヤ4アドレス——TCP/IPではポート番号になりますが——これは「その機器内でユニーク」であれば良い値になります(**図8**)。

▼図8　アドレスの対象と範囲



ARPとは

アドレスの紐づけ

前述のとおり、TCP/IPモデルで使われるアドレスのうち、レイヤ3アドレスがIPアドレスです。IPアドレスは全ネットワーク内で宛先を特定するためのアドレスです。

一方、Ethernetで使われるレイヤ2アドレスであるMACアドレスは、ネットワーク内部での宛先を特定するために必要となります。

送信する際に先に決定されるのはIPアドレスです。よってIPアドレスが決定されたあと、そのIPアドレスの宛先へ届けるために、ネットワーク内でのレイヤ2の宛先を決定する必要があります。つまり、宛先のIPアドレスとそれに対応するMACアドレスが必要となります。このアドレスの紐付けを決定するためのプロトコルがARP(Address Resolution Protocol)です。

ARPキャッシュ

レイヤ3PDUによりIPアドレスがセットされると、レイヤ3ヘッダの宛先IPアドレスが入手できます。そこで、そのPDUを送信するインターフェースのARPキャッシュを確認し、IPアドレスを検索します(**図9**)。

ARPキャッシュは、IPアドレスとそれに対応するMACアドレスを記載するメモリに格納された表です。ここにIPアドレスがあれば、その対応するMACアドレスを宛先MACアドレスとしてレイヤ2PDUのヘッダに入れることになります。

ARPキャッシュのIPアドレスとMACアドレスの対応(エントリ)は、一定時間で消去されます。これはMACアドレスがLANインターフェースに付属のアドレスであり、LANイン

ターフェースの障害などによって変更があり得るためです。変更があった場合、ARPキャッシュのエントリも変更されなければ古いMACアドレスが宛先として使用されてしまい、宛先へ届かないことになります。

ARPの動作

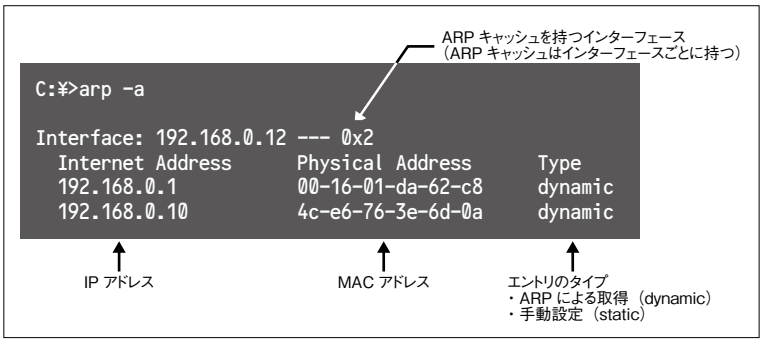
ARPキャッシュに宛先のIPアドレスのエントリがない場合、ARPリクエストを実行します。ARPリクエストは、ネットワーク内部へブロードキャストされる、MACアドレスを知りたいIPアドレスをデータとして入れたPDUです。

これを受け取った側は、IPアドレスが自身のIPアドレスでないならば無視し、自身のIPアドレスであるならばARPリクエストの送信元に対してARPリプライを返信します。

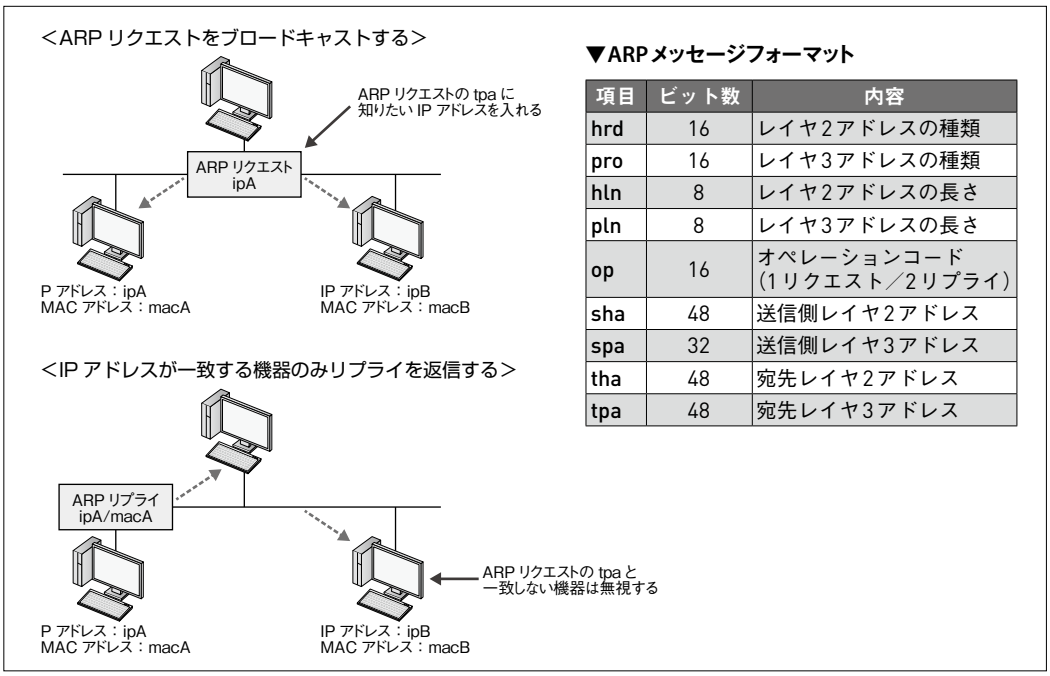
また、このARPリプライを送信する際には、ARPリクエストのIPアドレスとMACアドレスを自身のARPキャッシュに登録します。ARPリプライにはMACアドレスとIPアドレスが書かれており、ARPリプライを受け取ったARPリクエストの送信元はこれをARPキャッシュに登録します(**図10**)。

つまり、ARPキャッシュのエントリの書き換えは、「自身宛のARPリクエストの送信元のIPアドレスとMACアドレス」と「自身が送信したARPリクエストのリプライのIPアドレスと

▼図9　WindowsでのARPキャッシュの確認



▼図10　ARPの動作



▼ARPメッセージフォーマット

項目	ビット数	内容
hrd	16	レイヤ2アドレスの種類
pro	16	レイヤ3アドレスの種類
hln	8	レイヤ2アドレスの長さ
pln	8	レイヤ3アドレスの長さ
op	16	オペレーションコード (1リクエスト／2リプライ)
sha	48	送信側レイヤ2アドレス
spa	32	送信側レイヤ3アドレス
tha	48	宛先レイヤ2アドレス
tpa	48	宛先レイヤ3アドレス

MACアドレス」の2つの場合に行われるということになります。ARPキャッシュを手動で入力することもできますが、先ほど説明した障害などで対応できないという点もありますのであまり一般的ではありません。

IPとは

「internet」を担うプロトコル

IPはTCP/IPプロトコルスイートのレイヤ3プロトコルであり、プロトコルスイートの名前に持っているとおり、中核的なプロトコルです。

IPの役割としてはレイヤ3の役割である「ネットワーク間のデータ伝送」を行います。このネットワーク間のデータ伝送はinternetwork、または単にinternetと呼びます。IPはこのinternetのプロトコルです。ネットワーク間のデータ伝送を担うため、IPは大きく分けて2つの機能を持っています。

まず1つ目が前述したIPアドレス、つまり全ネットワークで各機器にユニーク性を保ったアドレスを付けることができるアドレス体系をIPは決めています。そしてこのアドレスはルータによる宛先の特定、つまりルーティング(routing)を可能にするアドレスです。

2つ目が転送中の制御です。具体的にはネットワークとネットワークをつなぐルータによってIPヘッダの値を使ったエラー制御や優先制御などになります。

転送制御

IPの転送制御として、ルーティングのミスなどで宛先に到達できなくなったPDUの破棄があります。IPv4ではTTL(Time To Live)、IPv6ではホップリミットと呼ばれる値を使います。IPv4で説明すると、送信側でIPヘッダに設定されたTTLはルータを中継するたびに1ずつ減少します。宛先に到達する前にゼロになってしまった場合、PDUは破棄されます(**図11**)。

TTLは送信側機器によって設定され、Windows系では128、Linux系では64に設定されることが多いです。

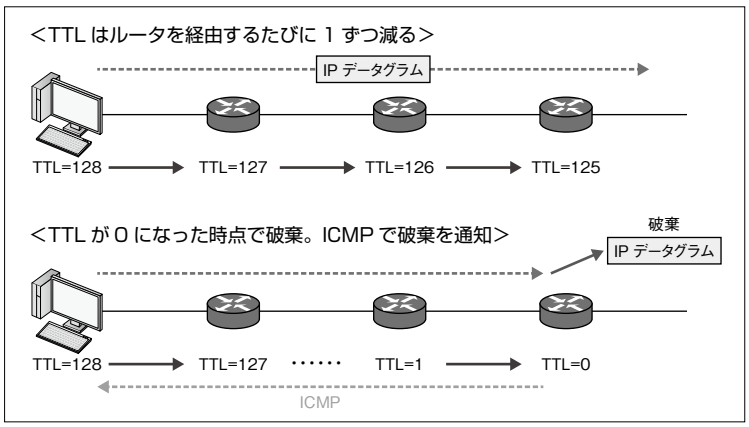
また、ルータがPDUを受け取りルーティングして送信する際に優先度を付けて送信するために、IPv4ではToS(Type of Service)、IPv6ではトラフィッククラスがあります。ただし、これらの値は組織内のルータではその処理に影響する場合がありますが、インターネットの基幹ルータでは無視されます。

フラグメント

通信で使用する回線により、その回線での最大転送サイズ(Max Transfer Unit Size：MTUサイズ)が決まっています。たとえば、Ethernetでは1,500オクテットになります。

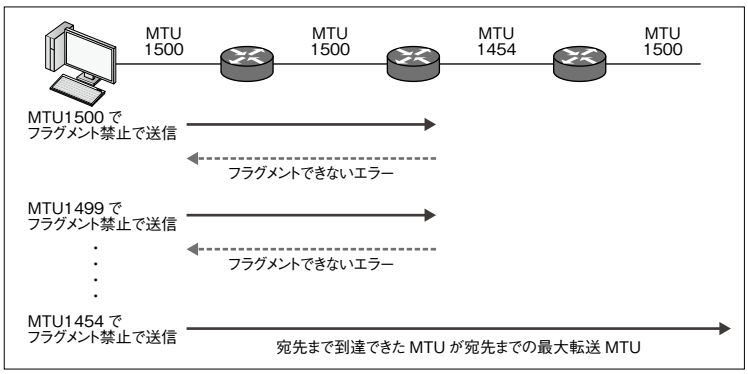
ですが、ルーティングしたルータがPDUを送信する際に、使用する回線のMTUがこのサイズを超えている場合があります。このとき、IPv4ではPDU

▼図11　TTLの動作



を分割するフラグメントを行います。ただし、このフラグメントにはルータの処理能力を使うため望ましくありません。よって、IPv6ではフラグメントそのものがなくなりました。そのため、IPv6では宛先までに使用する回線のMTUサイズを事前に調べるということが行われます。なお、IPv4でもフラグメントは望ましくないためこれが行われています。

▼図12　MTU探索



この作業はMTU探索と呼ばれており、pingを使用して宛先までの最小MTUを見つけ、上位プロトコルのPDUの時点でこのサイズに分割し、フラグメントを防ぎます(**図12**)。



TCPとは



TCPの利点

TCP/IPプロトコルスイートでは、レイヤ4プロトコルとして、TCP(Transmission Control Protocol)とUDP(User Datagram Protocol)の2つがあり、どちらか一方を排他で使用します。どちらを使用するかは上位プロトコルによって決まっており、TCPを使用する上位プロトコルが多くなっています。それはTCPの特性によります。

TCPは上位プロトコルに対し、大きく分けて2つの利点を提供します。それは「確実性」と「データの正しい順列」です。

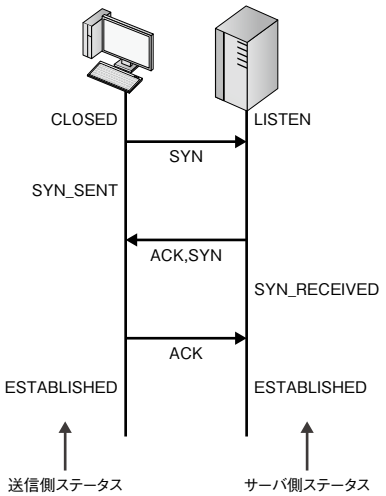
昔のネットワークの通信状況では遅延が大きく、また通信品質が低い(エラー率が高い)状況

▼図13　TCPフォーマットとステータス

▼TCPセグメントフォーマット

項目	ビット数	内容
送信元ポート	16	送信元アプリケーションのポート番号
宛先ポート	16	宛先アプリケーションのポート番号
シーケンス番号	32	送信するデータのシーケンス番号
確認応答番号	32	次に受信するデータのシーケンス番号
ヘッダ長	4	ヘッダ長
予約	6	将来に予約されたビット列
フラグ	6	SYN、ACKなどセグメント制御用フラグ
ウィンドウサイズ	16	受信用バッファサイズ
チェックサム	16	ヘッダのエラーチェックフィールド
緊急ポインタ	16	緊急用データの場所を示すポインタ
オプション		
ペイロード	可変	上位PDU

<スリーウェイハンドシェイクとステータス>



でした。そのため、「確実性」を持つTCPが多くの上位プロトコルで使われていました。現在では通信品質も上がりUDPを使う上位プロトコルも増えてきていますが、TCPは依然として多くの上位プロトコルで使われています。

「確実性」のためのコネクション

TCPはコネクションを確立するプロトコルです。コネクションとは、宛先との仮想的な通信路であるとよく説明されます。TCPの下位レイヤであるレイヤ3までの機能だけでは"届くであろう"ことがわかっているだけですので、実際に「相手に届き、相手が受け取る」ことを確認し、宛先との「通信路を作る」必要があるのです。これがコネクションです。

TCPではコネクションを開く(確立する)ために、スリーウェイハンドシェイク(3Way Handshake)を実行します。また、コネクションを維持する、つまり通信路が健在で、宛先が確実にデータを受信していることを確認するため、データの受信側は確認応答を送信側に返すことになっています。

また、コネクションは双方向の通信路です。スリーウェイハンドシェイクを始めた側からだけでなく、その相手側からも送信が可能です。そのため、お互いにデータをやりとりするアプリケーションであってもコネクションを1つ確立すれば良いことになります。

TCPのコネクションの状態はnetstatなどのコマンドで確認できます(**図13**)。

「データの正しい順列」のためのシーケンス番号

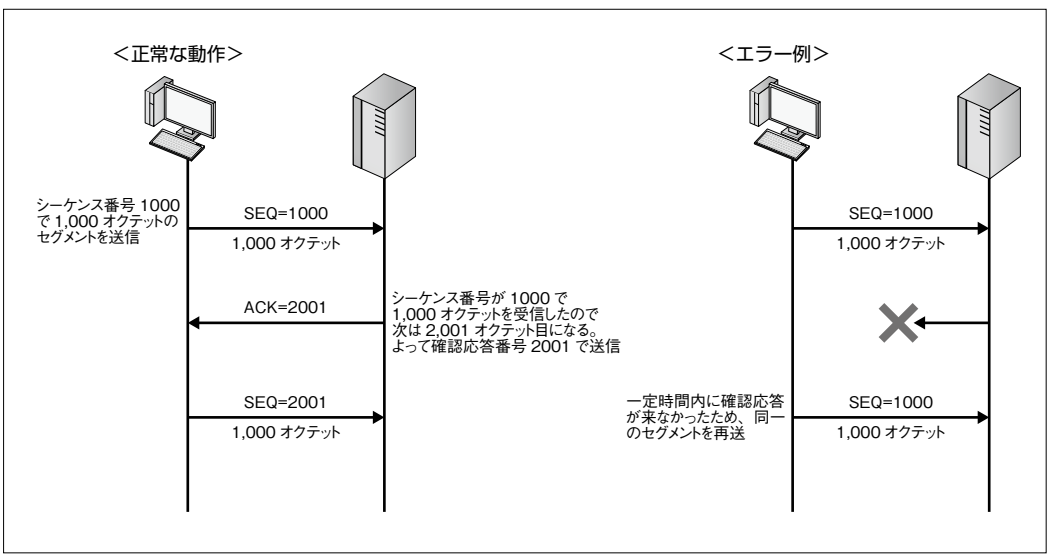
TCPでは送信する上位PDUを、MTUを基にしたサイズ(MSS：Max Segment Size)に分割します。これをセグメントと呼びます。

TCPではこのセグメントに番号を付けて、TCPヘッダに格納して送ります。これがシーケンス番号です。一方、受信側は受信したセグメントの次に受信すべき番号を、確認応答のTCPヘッダに格納します。これは確認応答番号と呼びます。

この2つの番号を使うことで、TCPは分割したデータを抜けなく送信でき、また抜けがあった場合には再送を行うことでエラーに対応します(**図14**)。

また、送信順と受信順に入れ替えがあり、順番どおりに届かなかったとしても、それを並べ替えて分割されたPDUを組み立てることを行

▼図14　シーケンス番号と確認応答番号



います。つまり、通信アプリケーションではTCPを使用することでデータは正しい順序で抜けなく送信・受信が行われることになります。

UDPとは

TCPの問題点

UDPはTCPと同じTCP/IPプロトコルスイートのレイヤ4プロトコルであり、TCPと排他で使われます。

TCPは確実性を備えたプロトコルで、TCP/IPプロトコルスイートの中核とも言えるプロトコルです。それと排他で使われるUDPは、TCPとは真逆の特性を備えています。つまり、TCPが行う確実性を担保する制御のいっさいを行いません。これだけ聞くと不確実で使いどころが難しいプロトコルですが、TCPではできないことを行うことができるプロトコルとして非常に重要です。

TCPは事前にコネクションを確立することが必須です。そのため、上位PDUがどんなに小さい場合であっても、コネクションの確立、確認応答、コネクションの切断にPDUをやりとりします。また、これらのためにTCPヘッダは大きいサイズが必要となっています。つまり、TCPはオーバーヘッドが大きいということになります。

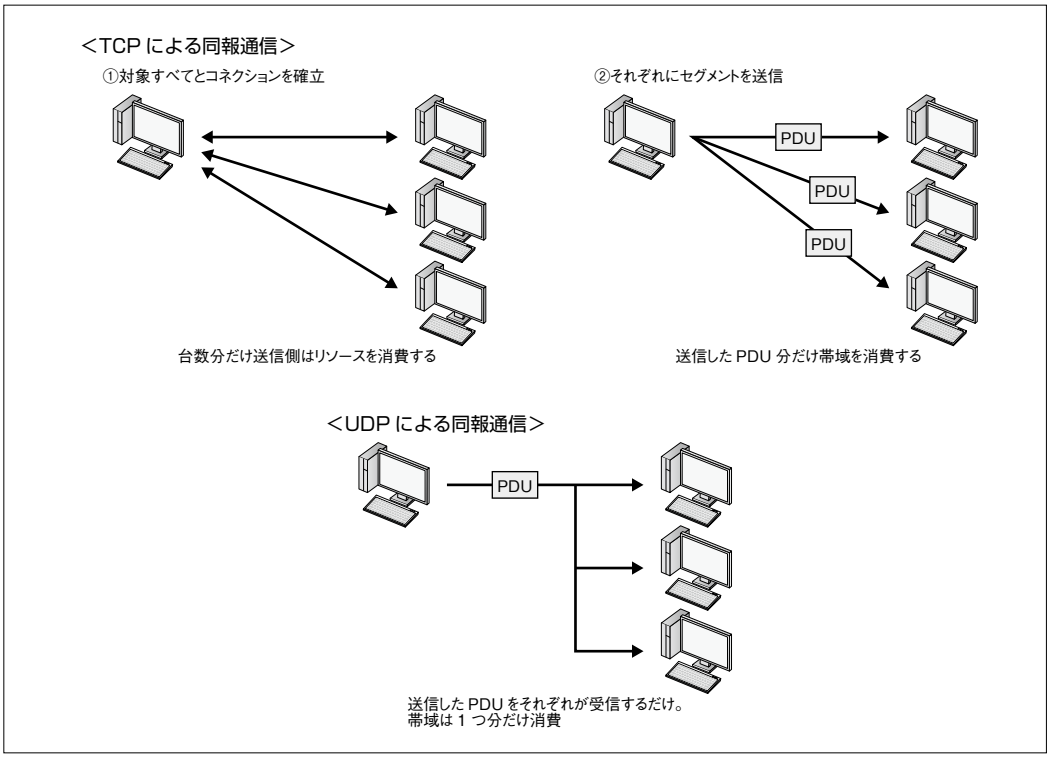
また宛先を特定してコネクションを確立するため、「存在するかしないかわからない相手」とは通信できませんし、「多数の相手に同報通信[注1]する」ことが苦手です。

UDPの利点

UDPはTCPとは真逆のプロトコルであり、TCPができない・苦手としている通信を行う

注1）同じ内容のメッセージを複数の相手に一斉に送ること。

▼図15　TCP/UDPと同報通信



ことができます。これは大きく分けて3つあります。「高速性が優先される通信」「小さいデータサイズの通信」「宛先が不明または同報通信」です。

とくに高速性が優先される通信、たとえばストリーミング再生や音声通信など、「確実性よりも高速性」の通信はUDPのほうが得意となります。

また、宛先が不明の通信はコネクションを必要とするTCPではできない通信です。代表的なプロトコルとしてはDHCPがあります。これはUDPの独壇場です。また、同報通信はTCPならば1台1台とコネクションを確立し、それぞれにセグメントを送信しなければなりませんが、UDPならば宛先不明に対する通信と同じように全体やグループに向けて送信し、必要な機器だけが受け取る、ということが可能となります。この方式ならば帯域を消費しなくて済むため、UDPがよく使われます(**図15**)。

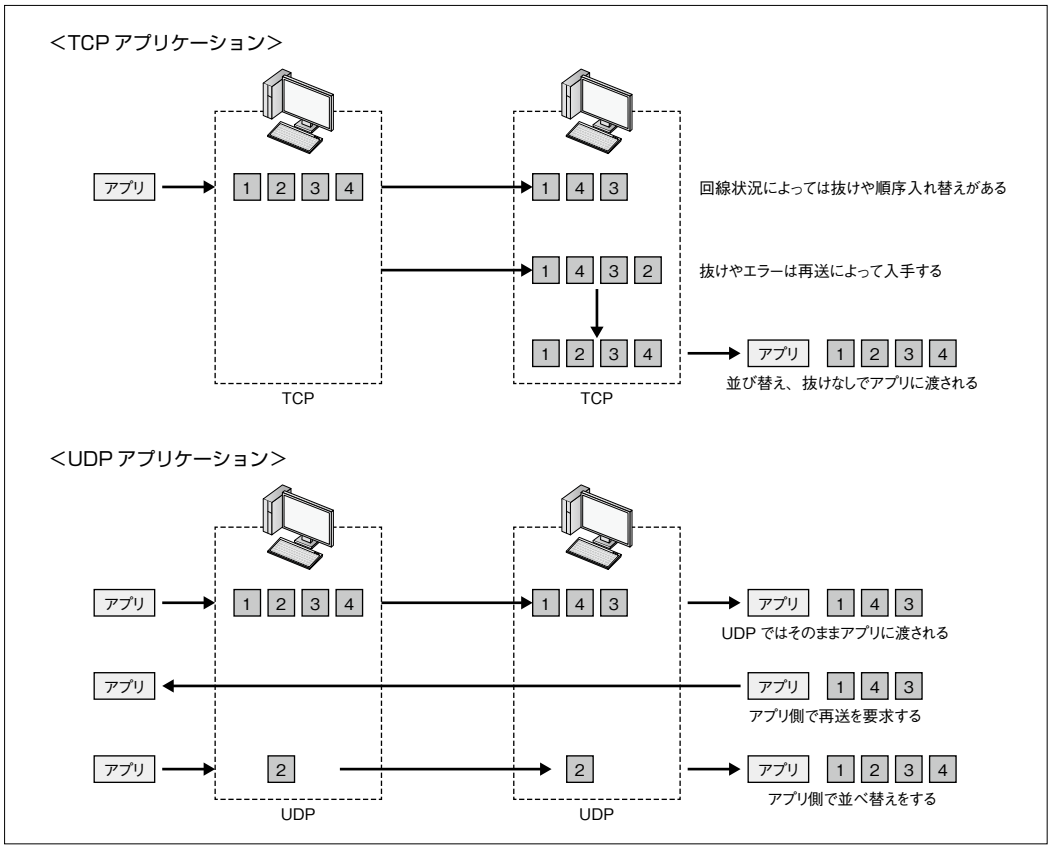


UDPアプリケーション

UDPを使用する通信アプリケーションでは、もしTCPを使用したならばTCPが行う確実性の担保や、データの分割とその並べ替えと組み立てを、アプリケーション側で実施する必要があります。

たとえば、確実性を担保するためアプリケーション側で受信応答を返すようにするなどが必要となります。また、データの分割と並べ替えと組み立てもアプリケーションで考え、間のデータが欠けた場合に再送するなどのしくみもアプリケーションで実装しなければなりません(**図16**)。SD

▼図16　TCP/UDPアプリケーション



第2章

実践ネットワークプログラミング～C言語編～

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

本章では、「ポート」と「ソケット」を使ったC言語によるネットワークプログラミングを紹介します。TCPのクライアント／サーバプログラム、UDPの受信側／送信側プログラム、そして名前解決を行うプログラムについて、サンプルコードを1行ずつ追いながらその動作を理解しましょう。



インターネットとC言語

インターネットの発明と普及に非常に重要な役割を果たしたUNIXですが、ベル研究所で開発されたUNIXは、C言語[注1]というプログラミング言語で作られました。C言語もまた、1960年代末から1970年代にベル研究所で開発されたものです。

C言語とUNIXとインターネットは同じ時期に誕生していることもあり、インターネットにはC言語とUNIXを前提とした考え方が各所に含まれています。

最近はC言語よりも便利なプログラミング言語が非常に多くあるので、C言語を必要とする人が減っていますが、そういった便利な環境が増えたために、逆にUNIXやインターネットが難解で勉強しづらいものになっているようにも感じています。インターネットが一般向けに普及し始めたころにインターネットを利用していた世代の人々は、今よりも不便な環境でコンピュータを利用していました[注2]。

しかし、その不便さのおかげで基本的な部分を一歩ずつ勉強できたのかもしれません。最初から便利になった環境で、「このボタンを押すといいよ」というような手順書に従っているだけでは、その手順書の裏側にあるものを理解しにくいというデメリットもありそうです。

そういった思想から、ここでは「ソケット」という裏側に少し近いテーマを扱いつつ、C言語を使ったサンプルを紹介していきます。



C言語を使える環境を用意しよう



インストール

C言語は、ほかのプログラミング言語同様に、まずはプログラミングを行うための環境を整備する必要があります。お店でパソコンを購入するだけでは、プログラミングの第一歩を踏み出せないことも多いのです。

C言語を利用したプログラミング環境としては、gcc(GNU Compiler Collection)を使います[注3]。

OS X(Mac OS X)では、AppStoreからXcodeをインストールしたうえで、Command Line Toolsをインストールします。OS XやXcodeのバージョンによって、Command Line Toolsのインストール方法は若干違いますが、OS X El CaptainではXcodeのXcodeメニューから

注1) 本来なら「プログラミング言語C」と言うべきかもしれませんが、ここでは「C言語」と表現してしまいます。

注2) 当時としては最先端であり、今となっては不便であったとしても、さらに昔と比べると便利にはなっているのでしょうが。

注3) ほかのものもありますが、ここではgccをインストールするものとします。

[Open Developer Tool]→[More Developer Tools]を選び、Command Line Toolsをダウンロードしてインストールします。

Linuxでは、利用しているパッケージ管理システムによってgccのインストール方法が異なりますが、基本的に「パッケージ管理システムコマンド + install + gcc」でインストールできます。

```
yumの場合
$ yum install gcc
aptの場合
$ apt-get install gcc
rpmの場合
$ rpm -ivh gcc
```

Windowsでは、MinGWをインストールするのがお勧めです。MinGWは、**http://www.mingw.org**からダウンロードできます。



コンパイル

gccがインストールできたら、今度はコンパイルを行います。C言語で書かれたプログラムは「ソースコード」と呼ばれますが、ソースコードそのままでは動作しません。ソースコードをコンピュータが実行可能なファイル形式に変換する「コンパイル」という作業が必要なのです。

gccを使ってコンパイルを行うには、ターミナル(LinuxやOS X)やコマンドプロンプト(Windows)などで、ソースコードが置いてあるディレクトリ(もしくはフォルダ)へと移動し、「gcc ファイル名.c」というコマンドを実行します。たとえば、ソースコードのファイル名が「test.c」であれば、「gcc test.c」となります[注4]。



通信の「入り口」となるソケット、何と通信するかを指定する「ポート」

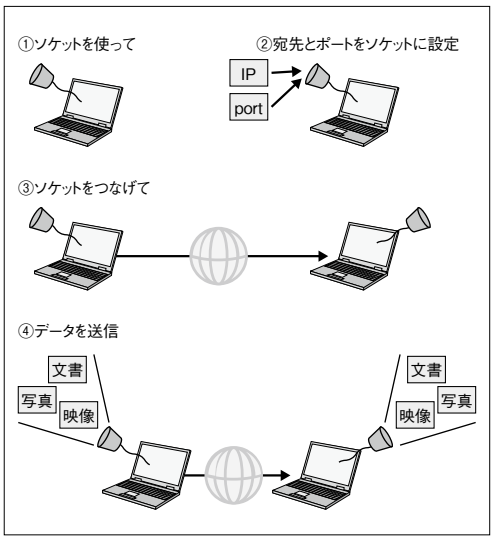
では、ソケットを使ってTCPでの通信を行う例を見てみましょう。

socketという英単語の日本語訳は「受け口」「コンセント」「差し込み口」などです。身近なものでは、電球を差し込む口やCPUやチップなどを差し込む部位が「ソケット」と呼ばれています。英単語としては、何かをつなぎ込むためのものが「ソケット」ですが、プログラミングにおける「ソケット」は、通信を行うための「口」です。ユーザは、ソケットにデータを書き込んだり、ソケットからデータを読み出したりすることで、インターネットを利用した通信を行います。

ソケットを使ってインターネットでの通信を行うプログラムは、**図1**のように、①ソケットを用意して、②通信したい相手のIPアドレスとポート番号を設定し、③相手と繋いで、④情報をやりとりするという手順になります。

①は、ソケットを用意するというものです。ユーザから見ると、パソコンを通じて通信を行うための「口」が、ニュウっと伸びて来ているような感じに見えるかもしれません。ソケットを用意するときに、そのソケットをどのように使いたいのかも指定します。たとえば、インターネットを利用してWebサーバと通信したいのであれば、TCPでの通信を行うためのソケットを用意します。

▼図1　ソケットとポートで通信する手順



注4) ここで紹介している方法のほかに、必要なソースコードなどをダウンロードしてgccをコンパイルするという方法もありますが、ここでは割愛します。

②は、通信相手と種類を指定するというものです。多少語弊はありますが、IPアドレスは「通信相手」を指定するために利用されるもので、ポート番号は「通信の種類」を指定するために利用されます。ポート番号というのは、通信の種類を番号で表したものです。たとえば、HTTPの一般的なTCPポート番号は80番ですし、メール配信の一般的なTCPポート番号は25番です。ここで重要なのが、通信する相手のIPアドレスを指定するということです。たとえば、「www.example.com」はIPアドレスではありません。www.example.comは「名前」であり、それをIPアドレスに変換したものが通信に利用されます。

③は、②で設定した相手と実際に「つながる」というものです。この作業が必要なタイプの通信と、そうでないものがあります。TCPでの通信では、「つながる」という作業が必要になります。

④は、つながったあとにデータをやりとりするというものです。一度つながってしまえば、あとはそのソケットを利用してデータを出し入れするだけです。ユーザから見ると、手元にあるファイルからデータを引き出すのも、ネットワークの向こう側にあるサーバからデータを引き出すのも同じような感覚で利用できます。

ソケットを利用してTCPで通信を行うと、こんな感じになります。ここまでの説明で、「クライアント」だの「TCP」だのという単語を説明なしに使ってしまいましたが、それらをこのあとから徐々に紹介していきます。



ソケットを利用したTCPプログラミング例

では、実際のTCPプログラミング例を見ていきましょう。

TCP接続が確立されるとき、「ポート○○につなぎたいです」と言う側と、それに対して「いいですよー」と言う側がいます。「つなぎたい」と言う――SYNパケットを送信する――側は「クライアント」、それを受け付ける側は「サーバ」と呼ばれます。

TCPの接続が確立したあとは、TCPそのものの機能としてはサーバとクライアントに差異はありませんが、TCP接続確立段階で動作が違うので、その部分はプログラムの書き方も違います。ソケットを利用したプログラミングを行うときの、TCPサーバとクライアントの違いを**図2**に示します。

TCPによる通信プログラムを書く場合、第1引数をAF_INET（IPv6の場合はAF_INET6）、第2引数をSOCK_STREAMにしたsocketシステムコールでソケットを作成します。この部分は、サーバとクライアントで同じです[注5]。

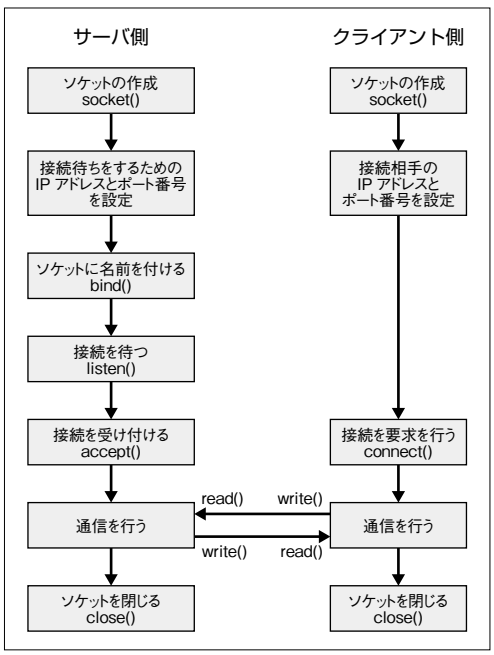


クライアント

では、まずはクライアント側のプログラムを見てみましょう（**リスト1**）。このサンプルでは、自分自身を示すIPアドレスである127.0.0.1

注5） 本稿のCサンプルはOS Xで書いています。Linuxの場合はリスト2・3・4において「socklen_t」部分が異なるので適宜修正してお試しください。エラー処理は割愛しています。

▼図2　TCPのサーバとクライアント



(localhost)の11111番ポートに対してTCP接続を確立したあとに、サーバからのデータを待ちます。サーバからのデータを受け取ると、標準出力へと受信データを表示して終了します。

socketシステムコールを使って作成したソケットを利用して、connectシステムコールで通信を行いたい相手とTCP接続を確立します。

▼リスト1　ソケットプログラミング(クライアント側)

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int
main()
{
 struct sockaddr_in server;
 int sock;
 char buf[32];
 int n;

 /* ソケットの作成 */
 sock = socket(AF_INET, SOCK_STREAM, 0);

 /* 接続先指定用構造体の準備 */
 server.sin_family = AF_INET;
 server.sin_port = htons(11111);
 server.sin_len = sizeof(server);
 /* ↑Linuxでは不要 */

 /* 127.0.0.1はlocalhost */
 inet_pton(AF_INET, "127.0.0.1", ➥
&server.sin_addr);

 /* sockaddr_in構造体の長さを設定 */
 /* (Linuxでは不要) */
 server.sin_len = sizeof(server);

 /* サーバに接続 */
 connect(sock, (struct sockaddr *)➥
&server, sizeof(server));

 /* サーバからデータを受信 */
 memset(buf, 0, sizeof(buf));
 n = read(sock, buf, sizeof(buf));

 printf("%d, %s\n", n, buf);

 /* socketの終了 */
 close(sock);

 return 0;
}
```

connectシステムコールの第2引数に接続相手情報を渡すことでTCP接続の相手を指定できます。TCP接続の確立に成功すると、connectシステムコールは成功しますが、そのソケットに対して読み書きを行うことでサーバとデータのやりとりが可能になります。

最後に、使い終わったソケットをcloseシステムコールで閉じて、一通り終了です。



サーバ

次に、サーバ側を見て行きましょう(**リスト2**)。

サーバ側は、ポート11111番で2本のTCP接続を受け付けてから終了します。1本目のTCP接続に対しては"HELLO"と書き込み、2本目には"HOGE"と書き込みます。

socketシステムコールを使って作成したソケットに対して、bindシステムコールを使って待ち受け用のIPアドレスとTCPポート番号を設定します。その後、listenシステムコールによってTCP SYNを受け付けるようになります。listen実行後に、待ち受けが行われているポート番号でTCP接続が確立されるようになりますが、確立し終わったTCP接続から新たなソケットを生成するのがacceptシステムコールです。サーバアプリケーションは、acceptシステムコールが生成したソケットに対して読み書きを行うことで、クライアントとの通信を行い



Linuxでの開発Tips

Linuxでは、sockaddr構造体(sockaddr_in、sockaddr_in6、sockaddr_storageなど)の中にサイズを示すためのメンバがありません。ここでのサンプルは、sockaddr構造体にサイズを示すメンバが入っているものになっていますが、Linuxで試す場合には、sockaddr構造体のサイズを示すメンバに値を代入している部分を削除してください。

ます。

サーバ側では、closeシステムコールをacceptで生成されたソケットすべてに対して個別に行う必要があります。また、listenを行ったソケットも使い終わったらcloseします。

◆ ◆ ◆

本稿では、TCPサーバ側サンプルで、あえて2本のTCP接続を受け付けるように書いてみました。これは、bindしたうえでlistenしている待ち受け用のソケットと、acceptによって新たに生成された通信を実際に行うためのソケットが違うものであることを理解していただくためです。このように、サーバ側は待ち受けしているポート番号に対してTCP接続を行って来るクライアントとのTCP接続を、次々と確立できるのです。



TCPの送信元ポート番号を設定する

bindの部分をもうちょっと掘り下げてみましょう。

bindは、「名前が付いていないソケットに名前を付ける」と解説されますが、TCPにおける「名前」とは送信元もしくは接続を受け付けるIPアドレスと、送信元ポート番号です。

listenの前にbindを行ったうえでacceptを行

▼リスト2　ソケットプログラミング（サーバ側）

左段下から続く↙

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int
main()
{
 int sock0;
 struct sockaddr_in addr;
 struct sockaddr_in client;
 socklen_t len;
 int sock1, sock2;

 /* ソケットの作成 */
 sock0 = socket(AF_INET, SOCK_STREAM, 0);


 /*** ソケットの設定 ***/

 /* IPv4を利用することを指定 */
 addr.sin_family = AF_INET;

 /* ポート番号は11111 */
 addr.sin_port = htons(11111);

 /* 任意のインターフェースを指定 */
 addr.sin_addr.s_addr = INADDR_ANY;

 /* sockaddr_in構造体のサイズを設定 */
/* (Linuxでは不要) */
 addr.sin_len = sizeof(addr);

 /* ソケットをインターフェースとポートに */
 /* 関連付け */
 bind(sock0, (struct sockaddr *)&addr, ➢
sizeof(addr));
```

右段へ続く↗

```
 /* TCPクライアントからの接続要求を */
/* 待てる状態にする */
 listen(sock0, 5);


 /* TCPクライアントからの接続要求を */
 /* 受け付ける(1回目) */
 len = sizeof(client);
 sock1 = accept(sock0, (struct sockaddr *)➢
&client, &len);

 /* 6文字送信 */
 /* ('H', 'E', 'L', 'L', 'O', '\0') */
 write(sock1, "HELLO", 6);

 /* TCPセッション1の終了 */
 close(sock1);


 /* TCPクライアントからの接続要求を */
 /* 受け付ける(2回目) */
 len = sizeof(client);
 sock2 = accept(sock0, (struct sockaddr *)➢
&client, &len);

 /* 5文字送信('H', 'O', 'G', 'E', '\0') */
 write(sock2, "HOGE", 5);

 /* TCPセッション2の終了 */
 close(sock2);


 /* listen するsocketの終了 */
 close(sock0);

 return 0;
}
```

えば、bindによって設定されたTCPポート番号でTCP接続要求を待つようになります。connectの前にbindを行えば、acceptを行うサーバ側が認識するクライアント側のTCPポート番号は、connect前に行ったbindによって設定されたTCPポート番号になります。

多少マニアックになってしまいますが、ソケットに対してTCPのポート番号が設定されるのは、bindを行ったときだけではないのでご注意ください。connectとlistenは、それらのシステムコールを利用する前にbindが行われていなければ、自動的にカーネルが送信元ポート番号を設定する機能を持っています。

たとえば、TCP SYNを送信するクライアント側にも送信元ポート番号はありますが、一般的なプログラミング手法では、bindを行わずにconnectが行われます。これにより、bindによる「名前付け」が行われずにTCP SYNが送信されるわけですが、TCP SYNを送信するには、何らかの送信元ポート番号が必要です。bindを行わずにconnectができるのは、connectを利用する前にbindが行われていなければ自動的にカーネルが設定を行ってくれているためです[注6]。

ここまでで、「あれ？　bindせずにlistenもできるのでは？」と考えたあなたは非常に鋭いです。多少マニアックになってしまいますが、

注6）bindしてからconnectすることもできますが、送信元ポート番号を明示したいような場合以外では、あまり行われない手法です。

▼リスト3　リスト2を、bindを行わずにgetsocknameを使うよう変更

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h> /* for memset */
#include <arpa/inet.h> /* for inet_ntop */

int
main()
{
 int sock0;
 struct sockaddr_in serv;
 struct sockaddr_in client;
 socklen_t len;
 int sock1;
 char serv_name[16];
 int n;

 /* ソケットの作成 */
 sock0 = socket(AF_INET, SOCK_STREAM, 0);

 /* TCPクライアントからの接続要求を */
 /* 待てる状態にする */
 n = listen(sock0, 5);
 if (n != 0) {
   perror("listen");
 }

 /* getsocknameを使うための準備 */
 memset(&serv, 0, sizeof(serv));
 len = sizeof(serv);

 /* getsocknameでソケットに付随する情報 */
```

右段へ続く↗

左段下から続く↙

```
/* を取得 */
 n = getsockname(sock0, (struct sockaddr *)
)&serv, &len);
 if (n != 0) {
   perror("listen");
 }

 /* ソケットに付けられた「名前」を */
 /* コンソールに表示 */
 memset(serv_name, 0, sizeof(serv_name));
 inet_ntop(serv.sin_family, &serv.sin_addr,
serv_name, sizeof(serv_name));

 printf("ipaddr=%s, port=%d\n", serv_name,
 ntohs(serv.sin_port));


 /* TCPクライアントからの接続要求を */
 /* 受け付ける（1回目） */
 len = sizeof(client);
 sock1 = accept(sock0, (struct sockaddr *)
&client, &len);

 /* 6文字送信 */
 /* ('H', 'E', 'L', 'L', 'O', '\0') */
 write(sock1, "HELLO", 6);

 /* TCPセッション1の終了 */
 close(sock1);

 /* listen するsocketの終了 */
 close(sock0);

 return 0;
}
```

TCP SYNを受け付けるサーバ側においても、bindを行わずにlistenを行うことで、カーネルが自動的に設定したポート番号でTCP SYNを待ち受けることができます。そういう書き方をしたときには、listen時点でどのようなポート番号になるのかを事前に予測するのが難しいため、listen後にgetsocknameというシステムコールを利用して、設定されたポート番号を調べたりします(**リスト3**)。

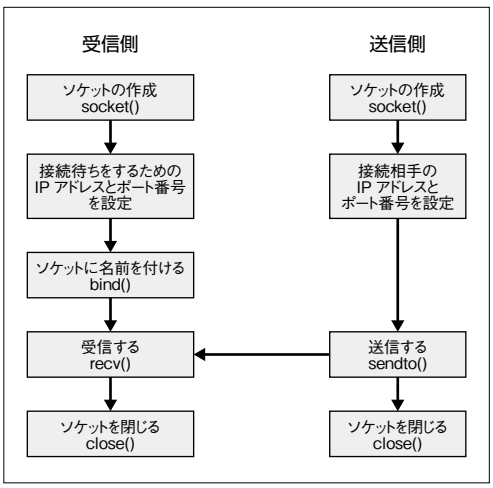
このように、bindが行っているのはポート番号のマニュアル設定であって、自動設定を使うことも可能だったりするのです。

UDPプログラミング例

次は、UDPパケットの送信や受信のプログラム例を見ていきましょう。

インターネットにおける通信の大半がTCPによるものですが、音声や動画の通信であったり、リアルタイム性が要求されたり、TCPほどの機能が不必要であったり、マルチキャストやブロードキャストによって同時に多数の相手と通信したいときなどに使われるプロトコルとして「UDP」があります。のちほど紹介するDNSに対する問い合わせにもUDPが使われています。

▼図3　UDPを利用した送信と受信



TCPでは、相手を指定したうえで接続をしてから実際の通信を行うという手順でしたが、UDPでは、**図3**のように相手を指定していきなり送信します。

送信側

まずは、UDPパケットを送信するプログラムの例です(**リスト4**)。

TCP用のソケットを作るときには、socketシステムコールの第1引数にAF_INETを、第

▼リスト4　UDPプログラミング(送信側プログラム)

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int
main()
{
 int sock;
 struct sockaddr_in s;
 int n;

 /* ソケットを生成 */
 sock = socket(AF_INET, SOCK_DGRAM, 0);

 s.sin_family = AF_INET; /* IPv4 */
 s.sin_port = htons(11111);
 /* ↑ポート番号は11111 */
 s.sin_len = sizeof(s);
 /* ↑Linuxでは不要 */

 /* 宛先IPアドレスとして 127.0.0.1 を設定 */
 n = inet_pton(AF_INET, "127.0.0.1", ➢
&s.sin_addr);
 if (n < 1) {
  perror("inet_pton");
  return 1;
 }

 /* UDPパケットを送信 */
 n = sendto(sock, "TEST", 5, 0, ➢
(structsockaddr *)&s, sizeof(s));
 if (n < 1) {
  perror("sendto");
  return 1;
 }

 /* ソケットをclose */
 close(sock);

 return 0;
}
```

2引数にSOCK_STREAMを指定していましたが、UDPでは第1引数がAF_INETで第2引数がSOCK_DGRAMになっています。AF_INETとSOCK_DGRAMの組み合わせは、IPv4のUDPになります。

▼リスト5　UDPプログラミング（受信側プログラム）

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int
main()
{
 int sock;
 struct sockaddr_in bindaddr;

 struct sockaddr_storage senderinfo;
 socklen_t addrlen;
 char buf[2048];
 int n;

 /* IPv4 UDPソケットの生成 */
 sock = socket(AF_INET, SOCK_DGRAM, 0);
 if (sock < 0) {
  perror("socket");
  return 1;
 }

 bindaddr.sin_family = AF_INET;
 /* ↑IPv4 */
 bindaddr.sin_port = htons(11111);
 /* ↑ポート番号は11111 */
 bindaddr.sin_addr.s_addr = INADDR_ANY;
 /* ↑すべてのローカルインターフェース */
 bindaddr.sin_len = sizeof(bindaddr);
 /* ↑Linuxでは不要 */

 /* ソケットに名前をつける */
 if (bind(sock, (struct sockaddr *)➥
&bindaddr, sizeof(bindaddr)) != 0) {
  perror("bind");
  return 1;
 }

 /* recvfromに渡すsockaddr構造体のサイズ */
 addrlen = sizeof(senderinfo);

 /* UDPパケットの受信 */
 n = recvfrom(sock, buf, sizeof(buf) - 1➥
, 0,
       (struct sockaddr *)&senderinfo, ➥
&addrlen);

 /* ターミナルに受信結果を出力 */
 write(fileno(stdout), buf, n);

 /* ソケットのclose */
 close(sock);

 return 0;
}
```

IPアドレスは自分自身を示すlocalhost（127.0.0.1）、宛先ポート番号は11111番に送信しています。送信しているデータは「'T', 'E', 'S', 'T', '\0'」という5文字です。

TCPのように、相手に接続する（connectする）という段階が存在せず、いきなりsendtoを行っているのが大きな特徴です[注7]。このサンプルでは、UDPの送信元ポート番号はsendtoの時点で設定されています。sendtoを行う前にbindを行えば、UDP送信元ポート番号を明示的に指定できます。



受信側

次は、受信プログラムを見てみましょう（**リスト5**）。

こちらは、UDP用のソケットを作成したうえで、それに対してbindシステムコールでポート番号11111番という名前を付けています。

bindで受信用のポート番号を設定したあとに、recvfromシステムコールでUDPパケットの到着を待っています。パケットを受信すると、writeシステムコールを使用して標準出力に受信した内容をそのまま表示しています。

◆ ◆ ◆

このようにUDPプログラミング例をご覧いただくと、本稿前半で紹介したソケットプログラミングが、実際は「TCPソケットプログラミング」であることがわかると思います。

TCPやUDP以外にも、ソケットにはさまざまな種類があります。本稿では割愛しますが、IPパケットを直接作成できるようなソケットや、ホスト内のプロセス間で通信するためのソケットなど、実際にはさまざまな種類のソケットがあるので、興味がある方はぜひいろいろと調べてみてください。

注7）UDPソケットに対してconnectを使うこともできますが、本稿では割愛します。



ソケットプログラミングで名前解決を行う



名前解決とDNSのしくみ

最近の便利なライブラリを利用していると、アプリケーションが名前解決を行っていることを忘れがちです。しかし、ソケットを利用したプログラミングでは、名前解決部分は非常に重要な要素です。

これまで、IPアドレスを直接指定するという前提でソケットプログラミングを紹介してきました。インターネットにおける識別子はIPアドレスであり、インターネットでの通信はIPパケットのヘッダに記載されたIPアドレスを基に行われます。しかし、数値の羅列であるIPアドレスは人間にとってわかりにくく、覚えにくいものです。そのため「www.example.com」などの「名前」を使ってインターネットに接続された機器を表現しています。

このような「名前」を、IPパケットのヘッダに直接指定することはできません。IPパケットのヘッダに記載される送信元IPアドレスと宛先IPアドレスは、「名前」ではなくIPアドレスでなければならないのです。そのため、「名前解決」と呼ばれる、名前をIPアドレスに変換する作業が行われています。

名前解決は、インターネット初期から行われていましたが、その当時はコンピュータの数も多くなかったので、IPアドレスと「名前」を対応させて羅列した「HOSTS.TXT」というひとつのファイルに、「名前」とそれに対応するIPアドレスが記載されたうえで、それが管理されていました。中央管理された「HOSTS.TXT」というファイルをみんながダウンロードして使っていたのです。

しかしインターネットの普及とともに、そのような管理を行うことが難しくなったため、「名前」に階層構造を導入した「ドメイン名」と、それを実現するために開発されたDNS（Domain Name System）というしくみが使われるようになりました[注8]。



getaddrinfoとgethostbyname

ソケットを利用したプログラミングでの名前解決は、getaddrinfoを利用します。プログラマは、DNSサーバへの問い合わせを行うプログラムを直接作成するのではなく、getaddrinfoを通じて名前解決を行うことができます。

昔はgethostbynameというAPIが利用されていましたが、gethostbynameはIPv4しか扱えないため、現在はIPv4に限定されないgetaddrinfoを利用することが推奨されています[注9]。

たとえば、Appleは2016年6月からiOSアプリの審査基準としてIPv4に依存するコードの禁止を追加しました[注10]。プログラミング言語としては、gethostbynameはiOSアプリ開発用に提供されていますが、今後はgethostbynameなどの旧APIを含むプログラムはiOSアプリの審査を通らなくなります。



IPの枯渇問題

ここまでは、IPv4を前提に話を進めてきましたが、IPv6対応が求められることが徐々に増えつつあります。

IPv4はIPアドレスが32ビットですが、インターネットが普及し、1人で何個ものIPアドレスを利用するような使い方が当たり前になったので、IPv4のアドレスが足りなくなってしまいました。そこでIPアドレス空間が広いIPv6への移行が提案され、今までは単一であ

注8） 今ではHOSTS.TXTの中央管理は行われていませんが、各自が使うシステム内で個別に/etc/hosts（BSD系やLinux）、/private/etc/hosts（OS X）、C:¥Windows¥System32¥drivers¥etc¥hosts（Windows）という設定ファイルとDNSの両方が使われることも多いです。

注9） getaddrinfoはIPv6対応のための実装ではなく、特定のアドレスファミリーに非依存な実装です。

注10） WWDC15で発表されたときには2015年末からでしたが、その後2016年5月4日に、2016年6月からと変更されました。WWDC15の発表資料→URL http://devstreaming.apple.com/videos/wwdc/2015/719ui2k57m/719/719_your_app_and_next_generation_networks.pdf 2016年5月4日の発表→URL https://developer.apple.com/news/?id=05042016a

ることが前提であった「IP」が1つから2つへと変わろうとしています。

IPv6はIPアドレスが128ビットですので、32ビットであるIPv4アドレスで表現できるIPアドレスの2の76乗倍という広大な空間を持ちます。2011年に実際に枯渇が到来したことにより、IPv6対応が世界中で急速に行われるようになりました。しかし、IPv6はIPv4との直接的な互換性がないため、「インターネットプロトコル」という視点で見ると、これまで1つであったインターネットと平行して、IPv6による2つ目のインターネットが急速に成長しているようにも見えます。

その一方で、「インターネットは1つである」という考え方もあるために、IPv4とIPv6をあえて混ぜた状態で設計されている部分もあります(名前解決のしくみなど)。このように、2つなのだけど1つであり、別物であると同時に同じものであるという、非常にややこしい状態です。そういった「2つであるが1つである」部分を仮想的に実現しているのが名前解決であるとも言えるのですが、ソケットプログラミングという視点でみたとき、getaddrinfoがまさにそういった部分になるわけです。



getaddrinfoを使う

では、getaddrinfoを利用したサンプルをご覧ください(**リスト6**)。getaddrinfoに対して渡す引数は、名前解決を行う文字列だけではなく、そのあとにソケットに対して設定したいポート番号なども含まれています。このサンプルでは、getaddrinfoが成功したら、その結果に含まれるパラメータを使ってソケットを作成し、そのソケットに対してconnectを行っています。

connectに成功すればそのソケットを利用しますが、失敗すればソケットを閉じてgetaddrinfoが返した次の結果を試します。getaddrinfoのAPIがソケットの種類やポート番号等の情報を

▼リスト6　getaddrinfoを利用したサンプルプログラム

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int
main()
{
 char *hostname = "www.example.com";
 char *service = "http";
 struct addrinfo hints, *res0, *res;
 int err;
 int sock;

 memset(&hints, 0, sizeof(hints));
 hints.ai_socktype = SOCK_STREAM;
 hints.ai_family = PF_UNSPEC;
 /* ↑IPv4とIPv6両方を取得 */
 if ((err = getaddrinfo(hostname, service, ↵
&hints, &res0)) != 0) {
  printf("error %d\n", err);
  return 1;
 }

 for (res=res0; res!=NULL; res=res->ai_↵
next) {
  sock = socket(res->ai_family, res->ai_↵
```

右段へ続く↗

左段下から続く↙

```
socktype, res->ai_protocol);
  if (sock < 0) {
   continue;
  }

  if (connect(sock, res->ai_addr, res->ai_↵
addrlen) != 0) {
   close(sock);
   continue;
  }

  break;
 }

 if (res == NULL) {
  /* 有効な接続ができなかった */
  printf("failed\n");
  return 1;
 }

 freeaddrinfo(res0);

 /* ... */
 /* ここ以降にsockを使った通信を */
 /* 行うプログラムを書いてください */
 /* ... */

 return 0;
}
```

引数として渡すようにできており、getaddrinfoが返す結果にもそれらが含まれるので、その結果をそのまま利用してsocket、connect、bindなどのシステムコールを使えるようになっています。



IPv6とIPv4のどちらを使うべきか――Happy Eyeballs

前述のとおり、特定の名前に対して複数のIPアドレスが関連付られることもあるため、名前解決によって得られるIPアドレスは1つだけとは限りません。名前解決の結果、IPv4とIPv6の両方のIPアドレスが得られた場合に、どちらのアドレスファミリを利用するのかは、プログラマに任されています。

2015年の中旬に話題になったのが、Appleが実装した手法です。OS XのEl Captainでは、IPv4とIPv6の両方に対する名前解決を行いつつ、最初に届いた結果がIPv4であれば、25ms待ってからTCP接続確立を試みるなど、IPv6を優先的に使うような工夫が行われています[注11]。

2012年に発行されたRFC 6555では、Google ChromeとMozilla Firefoxで採用されているアルゴリズムとして次の手法を紹介しています。

①getaddrinfo()を利用して名前解決を行う
②getaddrinfo()から受け取ったIPアドレスリストの順番に接続を試みる
③接続の試みが短時間で確立しない場合（FirefoxやChromeでは300ms）、異なるアドレスファミリの最初のIPアドレスでの接続を試みる。この場合、直前に失敗したアドレスファミリがIPv6であれば、IPv4を試すことになる
④最初に確立した接続を利用する

このように、複数のTCP接続を並行して確立しようとしつつ、最初に成功したTCP接続を利用することで、IPv4とIPv6のどちらを利用するのが良いのかを自動的に判断する試みもあります。

◆ ◆ ◆

IPv6対応のためにも、今後はURLなどを入力として受け取るような便利なライブラリを利用して、できるだけソケットを直接いじらないようなAPIを利用することが今よりもさらに推奨されるようになるかもしれません。

とは言え、ソケットプログラミングがすべて不要になるわけでもありませんし、インターネットのしくみを学ぶうえでソケットプログラミングを経験したことがあるかないかで大きな差がでると、筆者は考えています。プログラミング環境としてはけっして便利とは言えませんが、その不便さがゆえに、実は理解がしやすいのかもしれません。SD

注11）URL https://www.ietf.org/mail-archive/web/v6ops/current/msg22455.html

Column gai.conf

getaddrinfoが返す順番によって、ユーザがどのようなIPアドレスで通信を行うのかが変わってきます。そのため、getaddrinfoが結果を返す順番というのは非常に重要な要素となるわけです。

DNSによって得られる名前解決結果が複数存在するときに、getaddrinfoがどのような順番で結果を返すのかに関しての標準がRFC 6724[注A]に記述されているので、興味がある方はぜひご覧ください。

環境によっては、IPv6ではなくIPv4を優先的に利用したい場合もあります[注B]。gai.confという設定ファイルを編集することで、そのホスト内でgetaddrinfoが返す結果の優先度を設定することもできます。ご興味がある方は「gai.conf」というキーワードでWeb検索をしてみてください。

注A）Default Address Selection for Internet Protocol Version 6 (IPv6), http://tools.ietf.org/html/rfc6724

注B）詳細は割愛しますが、6to4のIPv6アドレスを取得してしまうような場合など。

第3章 実践ネットワークプログラミング ～スクリプト言語編～

Author 中島 雅弘（なかじま まさひろ）、植西ゴロー由謹（うえにしごろーよしのり）、國信 真吾（くにのぶ しんご）
㈱アーヴァイン・システムズ

スクリプト言語でのネットワーク通信プログラムは、データ構造の柔軟さやライブラリの気配りから、C言語での実装よりも容易かもしれません。前半はJavaScript、Ruby、PHP、Pythonで実装例を紹介し、後半は実践的に、RubyとShellを組み合わせて通信性能の評価をしてみましょう。

※本章のサンプルコードの動作確認は，できるだけ閉じたLAN環境内でのみおこなうようご注意願います。
詳細はhttp://gihyo.jp/magazine/SD/archive/2016/201607/supportをご覧ください。

GitHubで自分のプログラムをパブリックにしておけば、誰かが見て評価してくれます。一部の企業の採用には、こうした活動をつぶさに観察している専任者がいるそうです。大学の1、2年次であっても活躍できそうな学生は青田刈りされます。実績を評価してもらえるのは学生にとってもいいことですし、企業はその人が本当に現場で活躍できるようになるか、より正確に判断できます。

最近は、ライブラリやミドルウェアの発展のおかげで、機能的にすぐれたプログラムを作りやすい環境が整っています。システムの基本を理解せずとも、フレームワークやミドルウェアのAPIの使い方さえ知っていれば、それこそコピー＆ペーストだけでも高機能なプログラムを実現できます。高度なライブラリやミドルウェアは、システムコールやライブラリファンクションをコツコツと組み上げて、煩雑なプロトコルに対応する手間や間違いを犯すリスクを軽減してくれます。

しかし、ベースとなるフレームワークやミドルウェアが廃れてしまうと、新しい環境に適応できなくなるかもしれません。また、非同期通信を駆使するより発展的なプログラムや、高いパフォーマンスが要求されるシステムの実現には、基本プロトコルをしっかりと理解していなければなりません。ネットワークプログラミングは、単独のコンピュータ上で動作するプログラムのように、一本道ではたどりつけない不確実な要素があります。そうした中、大学の情報科学教育の現場では、基礎を学ぼうとせず、企業にアピールして就職できればそれでよいという学生に、どのように基礎原理の大切さを伝えればよいかが課題になっているそうです。

Software Designを愛読する皆さんは、どのようなエンジニアを目指すべきかよく考えてみてください。新しい問題に出くわしても、高度な機能を素早く実現しなければならなくても、たとえ愛用しているミドルウェアに寿命がきても、基礎原理とハイレベルインターフェースの両方に熟練したネットワークプログラミング・グル[注1]なら、怖いものはありません。



スクリプト言語を使ったネットワークプログラミング

APIを組み合わせたWebアプリケーションで実現できることは、だいたい業務系・定型なものに限られます。インターネットプロトコルは、http(s)のほかにもたくさんあり、これらを活用しないのはもったいないことです。より発展的なWeb/Networkアプリケーション、より対話性の高いアプリケーションを実現するためには、TCPやUDPによる通信のしくみを理解し使いこなす必要があります。また、通信原

注1）「熟練者」とか「尊敬すべき人物」といった意味合いです。

理の理解は、性能面、セキュリティ面での問題解決にも役立ちます。



ソケットを使った通信プログラムとアプリケーションレベルの例

ここからは、複数のスクリプト言語でのソケット通信の実装例を見ていきましょう。紙幅の都合で掲載しきれないので、本誌Webサポートページ[注2]からすべてのサンプルソースコードをダウンロードして、各言語のソースコードを並べて比較しながら読み進めてください。文中のリスト番号とファイル名、収録フォルダの対応は**表1**になります。対象環境はLinuxとOS Xで、スクリプト言語環境が整っていることとします。また、本稿ではIPv4を対象としています。

スクリプト言語ごとに表記や処理、使うライブラリに多少の差異があります。とはいえ、いずれのスクリプト言語でソケット通信をする場合も、システムコールやライブラリファンクションが言語処理系に内包されたものを呼び出して使うので、次の基本手順はC言語での実装と大きな違いはありません。

注2）URL http://gihyo.jp/magazine/SD/archive/2016/201607/support

▼表1　サンプルソースコード対応表

記事中表記	ファイル名	収録フォルダ
リスト1	echo_server.js	nodejs
リスト2	echo_server.rb	ruby
リスト3	udp_receiver.rb	ruby
リスト4	udp_sender.rb	ruby
リスト5	tcp_client.py	python
リスト6	udp_client.py	python
リスト7	echo_server.php	php
リスト8	smtp_mail.rb	ruby
リスト9	tcp_receive_server.rb	ruby
リスト10	udp_receive_server.rb	ruby
リスト11	tcp_single_sender.rb	ruby
リスト12	udp_single_sender.rb	ruby
リスト13	getsockopt.rb	ruby
リスト14	tcp_multi_sender.rb	ruby
リスト15	udp_multi_sender.rb	ruby
リスト16	multi_sender.sh	ruby

※リスト9～16は記事中には掲載されていません。

［TCP通信での基本手順］

●Server
- TCPソケットを作る（socket）
- ソケットに接続待ちをするIPアドレスとポートを結びつける（bind）
- 接続待ちにする（listen）
- クライアントからの接続を受け付ける（accept）
- 通信を行う（read/write）

●Client
- TCPソケットを作る（socket）
- 接続先IPアドレスとポート番号を指定して接続する（connect）
- 通信を行う（read/write）

［UDP通信での基本手順］

●Receiver
- UDPソケットを作る（socket）
- ソケットに接続待ちをするIPアドレスとポートを結びつける（bind）
- データを受け取る（recv）

●Sender
- UDPソケットを作る（socket）
- 接続先IPアドレスとポート番号を指定して送信する（sendto）

JavaScriptでの実装

JavaScriptは、従来クライアント専用のプログラム言語でした。ネットワーク通信のしくみを駆使することで、Ajaxなどの非同期通信やプッシュ型通信が可能です。現在は、アプリケーションの表現や機能実現の幅が増え、ブラウザ上で従来のPCアプリケーションと同等のプログラム開発ができるなど、さまざまな応用が可能になりました。

ここではNode.jsを使い、標準的なソケットを用いたプログラムをJavaScriptで実装します。Node.jsは、従来のクライアント（＝ブラウザ）上での動作を前提としていたJavaScriptを、サーバサイドで動作させるインタープリタです。そしてそれ自体がWebサーバ＋アプリケーショ

ンサーバとして動作します。

JavaScriptとnode.jsを使った単純なサーバ・クライアント通信

Node.js[注3]をhttps://nodejs.org/から、使用しているプラットフォームに合ったものをダウンロードしてインストールしてください。

まずは適当なディレクトリを作り、その中に**リスト1**のecho_server.jsを配置します。準備ができたら次のようにサーバプログラムを起動します。

```
$ node echo_server.js
```

別の端末を開いて、クライアントを起動します。クライアントはtelnetコマンドを使い、3000番ポートに接続します。

```
$ telnet localhost 3000
```

接続ができたら何かキーボードから入力してみます。すると、入力した文字列がそのまま返ってくることが確認できます。

クライアントでexitを入力するとサーバとの接続が解除されます。サーバプログラムは、Ctrl + c などの割り込みで終了させます。

```
Server側:
Start Server - 127.0.0.1:3000
127.0.0.1:56397 >> abc

127.0.0.1:56397 >> def
```

注3) 使用したバージョンはnodejs:v5.9.1/npm:3.7.3

▼リスト1　echo_server.js (on Node.js)

```
 1 // エコーサーバ(TCP)
 2 //
 3 var net = require('net');
 4
 5 // TCPサーバを作成する
 6 var server = net.createServer();
 7 // 最大接続数を指定
 8 server.maxConnections = 10;
 9
10 // 接続イベントを定義する
11 server.on('connection',
12   function(socket) {
13     var data = '';
14     var newline = /\r\n|\n/;
15     // データ受信イベントを定義する
16     socket.on('data', function(chunk) {
17         string = chunk.toString()
18         if ( string == 'exit' + "\r\n"
19           || string == 'exit' + "\n") {
20             // chunkの中身がexitの場合は
接続終了とする
21           socket.end();
22         }
23         data += string;
24         var client =
socket.remoteAddress + ':' +
socket.remotePort;
25         // 改行コードが含まれているか？
26         if (newline.test(data)) {
27           console.log(client + " >> " +
```

右段へ続く↗

左段下から続く↙

```
+ data);
28           // 受け取ったデータを返送する
29           if (socket.writable) {
30             socket.write("RET: " + data);
31           }
32           data = '';
33       }
34     });
35     // 接続断イベントを定義する
36     socket.on('end', function() {
37       server.getConnections(
38         function(err, count) {
39           console.log('Connection
End()');
40         });
41     });
42   });
43 // クローズイベントを定義する
44 server.on('close',
45   function() {
46     console.log('Server Closed');
47   });
48 // 待受アドレスとポートを指定し、接続待ちを
おこなう
49 server.listen(3000, '127.0.0.1',
50   function() {
51     var addr = server.address();
52     console.log('Start Server - ' +
addr.address + ':' + addr.port);
53   });
```

```
Client側:
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
abc↵
RET: abc
def↵
RET: def
exit
```

TCP通信の基本パターンとちょっと異なるところは、**リスト1**のソースコード中49行目のlistenで、bindとlistenの処理をまとめて行っているところ。さらに、Node.jsはイベント駆動型の非同期通信モデルであることです。acceptの処理が明示的に記述されておらず、接続があると11行目からの接続イベント、通信があると16行目からの通信イベントが処理されます。socketを使う場合には、これら2つのイベントと、44行目からの接続断イベントを定義するようにします。

どうですか、簡単だったでしょう。続けてRubyを使ったネットワークプログラミングに挑戦しましょう。

Rubyの実装例

Rubyのように手軽にコーディングでき、書いたプログラムの可読性も高いスクリプト言語は、サーバ運用視点からも実践的で、活用する機会が多くあります。たとえば、Mailクライアントが導入されていないサーバで、Webアプリケーションからメールを送信するなど、簡単にネットワーククライアントプログラムを制作できます。Ruby on Rails(RoR)上でも、外部システムとの連携や広義な意味でのスクレイピングのための、(以前はbackgroundrb、最近はclockworkなどから定期的に起動される)バッチ型ネットワーク通信プログラムにも応用できます。

Rubyで単純なサーバ・クライアント通信

はじめは、TCPでNode.jsのときと同じようにクライアントからの入力をそのまま返すエコーサーバの例です。サンプルソースコードのecho_server.rb(**リスト2**)を参照してください。

サーバを起動します[注4]。

```
$ ruby echo_server.rb
```

JavaScriptのときと同様に、telnetで接続してください。まったく同じ動作を確認できると思います。

リスト2の3行目でsocketライブラリをrequireし、TCPServerクラスを使います。12行目のTCPServerのオブジェクトを生成するところでsocket + bindの処理がまとめて実行され、デフォルトではlisten[注5]のbacklog(接続待ちリクエストのキュー)が5に設定されます。18行目からは、accept以下をマルチスレッドを使って複数のリクエストを並列で処理しています。

続いてUDPの例を見てみましょう。こちらのソースコードはudp_receiver.rbとudp_sender.rb(それぞれ、**リスト3**、**リスト4**とします)です。

UDPではサーバ+クライアントという型で

NOTE

Node.jsを活用するときは、素のsocket通信より、WebSocketやAjaxを活用してブラウザとの非同期通信を実現したい場面が多いかもしれません。これらを実現するには、Socket.IOを導入します。

Socket.IOは、httpによる一方向の非同期通信の機能に対して、WebSocket、Ajaxロングポーリングなどの非同期双方向通信を抽象化した、リアルタイムWebアプリケーションのためのコミュニケーションレイヤです。これにより利用者は、ブラウザによるコンパチビリティを意識せずにメッセージ送受信するプログラムを開発できます。

注4) 使用したRubyのバージョンは2.2.3p173

注5) listenメソッドを個別に呼び出してbacklogの値を設定することもできます。

はなく、基本的に相手のアドレス＋ポートにめがけてメッセージを送り合うスタイルです。ここでは、受け手(**リスト3**)と送り手(**リスト4**)と、わかりやすく役割をつけています。送りつけるデータ(message.txt[注6])を用意して、それぞれのプログラムを別の端末で、受け手－送り手の順で起動してみてください。

```
Receiver側：
$ ruby udp_receiver.rb
Receiver Start - 127.0.0.1:3000
"abcdefghijklmnopqrstuvwxyz"

Sender側：
$ ruby udp_sender.rb
Sender Start - 127.0.0.1:3000
```

このプログラムでは、message.txtの内容を5文字ずつ分割して送受信しています。

注6) このファイルもサンプルに同梱されています。

▼リスト2　echo_server.rb

```
 1 #!/usr/bin/env ruby
 2
 3 require "socket"
 4
 5 # ホストアドレス
 6 host = "127.0.0.1"
 7 # ポート番号
 8 port = 3000
 9
10 # ソケットをオープンし待受アドレスとポートを指定する
11 #(関数内部でsocket/bindを同時に実行している)
12 server = TCPServer.open(host, port)
13
14 puts 'ServerStart:'+host+":"+port.to_s
15
16 while true
17   # スレッドを作成し、ソケットへの接続を待つ(関数内部でlisten/acceptを連続して実行している)
18   Thread.start(server.accept) do |socket|
```

右段へ続く↗

左段下から続く↙

```
19     # 接続がされたら処理を実行する
20     puts 'Connecttion Start'
21     # ソケットからデータを受信する
22     while buffer = socket.gets
23       if buffer.strip == "exit"
24         # データが"exit"なら処理終了する
25         break
26       end
27       puts socket.peeraddr[2]+":"+socket.peeraddr[1].to_s+">"+buffer.strip
28       # 受信したデータを返送する
29       socket.puts "RET: " + buffer
30     end
31     puts 'Connecttion Closed'
32     # ソケットをクローズする
33     socket.close
34   end
35 end
36 # メインソケットをクローズする
37 server.close
```

▼リスト3　udp_receiver.rb

```
 1 #!/usr/bin/env ruby
 2 require "socket"
 3
 4 host = "127.0.0.1"
 5 port = 3000
 6
 7 receiver = UDPSocket.open()
 8
 9 receiver.bind(host, port)
10
11 puts 'ReceiverStart:'+host+":"+port.to_s
12
13 # データ宣言兼初期化
14 n = 0
15 m = 0
16 datas = []
17
18 loop do
19   seq, data = receiver.recv(65535).split(':')
```

右段へ続く↗

左段下から続く↙

```
20   if seq.to_i == 0 then
21     # データ個数取得
22     m = data.to_i
23   else
24     # データ取得
25     #（受信データの順番が入れ替わる可能性があるのでその対応）
26     datas[seq.to_i]=data
27   end
28   if m == n then
29     break;
30   end
31   n += 1
32 end
33
34 p datas.join('')
35
36 receiver.close
```

Pythonの実装例

日本国内よりも海外のエンジニアに好まれる傾向のあるPythonですが、スクリプト言語のなかでも、比較的ハイパフォーマンスなプログラムが実現できる軽快さ、記述性と明快性のバランスが特徴です。最近は、ディープラーニングのプログラムにもよく使われていますね。ネットワーク関連ではセキュリティーツール、CTFやExploitでの活用事例が豊富です。また、世界的にヒットしたMMORPGのサーバサイド、クライアントサイドの両方に使われた実績もあります。

Pythonで単純なサーバ・クライアント通信

Pythonでの例は、クライアントサイドのプログラムです。サーバにはJavaScript版かRuby版のecho_serverを立ち上げておきます。準備ができたら次のように、クライアントを起動してください[注7]。

```
$ python tcp_client.py
```

ソースコードはtcp_client.py(**リスト5**)とudp_client.py(**リスト6**)です。socketライブラリをインポートして使います。Pythonのプログラムは標準型にそった形式です。

PHPの実装例

Webプログラミングの古典であり、ばりばり現役のPHPです。

PHPで単純なサーバ・クライアント通信

echo_serverをPHPで記述したのがecho_server.php(**リスト7**)です。phpのプログラムも標準型にそっていますね。

phpコマンド使ってサーバを起動します。

```
$ php echo_server.php
```

クライアントには、telnetかPythonのクライアントプログラムを使ってください。exitでの終了も同様です。



ハイレベルのライブラリを使う

次に、サービスと通信するRubyスクリプトを見てみましょう。ソケットの上位のレイヤは、Rubyに含まれているライブラリがサポートしてくれますので、とても短いプログラムで機能を実現できます。

注7) 使用したPythonのバージョンは2.7.10

▼リスト4　udp_sender.rb

```
 1 #!/usr/bin/env ruby
 2
 3 require "socket"
 4
 5 host = "127.0.0.1"
 6 port = 3000
 7
 8 # ファイルの読み込み(５文字ずつ分割をする)
 9 texts = File.read("hoge.txt",
10   :encoding => ➥
Encoding::UTF_8).scan(/.{1,5}/)
11
12 # 送信用ソケットアドレスを設定
13 sockaddr = ➥
Socket.pack_sockaddr_in(port, host)
14 sender = UDPSocket.open()
```

右段へ続く↗

左段下から続く↙

```
15 puts 'SenderStart:'+host+":"+port.to_s
16
17 # ファイル転送(シーケンス番号を使った独自プ➥
ロトコル)
18 # 送信個数を送る(シーケンス番号は０番)
19 n = 0
20 sender.send(n.to_s+":"+texts.count.to_s,
21              0, sockaddr)
22 # 分割メッセージを送る(シーケンス番号は１番から)
23 texts.each{|text|
24   n += 1
25   sender.send(n.to_s + ":" + text,
26               0, sockaddr)
27 }
28 # 送信完了
29 sender.close
```

Mailクライアントを Rubyで

Rubyでは、ネットワーク階層に沿った形で、SMTPなどTCP/UDPの上位プロトコルを使うためのライブラリがあります。ここでは、Net::SMTPライブラリを使ってSMTPと通信してメールを送信してみます。サンプルソースコードはsmtp_mail.rb(**リスト8**)です。このプログラムの中には、socket、bindなどはありません。Net::SMTPがすべて担っているところに注目してください。ハイレベルのプロトコルを使う場合には、上位のライブラリがあれば下位のsocket処理などを気にする必要はありま

▼リスト5　tcp_client.py

```
 1 # -*- coding: utf-8 -*-
 2 #
 3 from __future__ import print_function
 4 import sys
 5 import socket
 6 from contextlib import closing
 7
 8 def main():
 9   argv = sys.argv
10   argc = len(argv)
11
12   # ホストアドレス
13   host = '127.0.0.1'
14   # ポート番号
15   port = 3000
16   # 受信バッファサイズ
17   bufsize = 4096
18
19   # 引数をチェック
20   if argc == 3:
21     port = int(argv[2])
22     host = argv[1]
23   elif argc == 2:
24     host = argv[1]
25
26   # ソケット作成
```

右段へ続く↗

左段下から続く↙

```
27   sock = socket.socket(socket.AF_INET,
28                        socket.SOCK_STREAM)
29   # ソケットがクローズされるまでループする
30   with closing(sock):
31     # サーバに接続する
32     sock.connect((host, port))
33
34     while True:
35       # キーボードからのタイプ待ち
36       sys.stdout.write("input-> ")
37       line = sys.stdin.readline()
38       # 空行だと終了させる
39       if len(line.rstrip()) == 0:
40         break
41       # キーボードからの入力を送信する
42       sock.send(line.encode('utf-8'))
43       # 入力文字列が'exit'だと終了させる
44       if line.rstrip()=='exit':
45         break;
46       # サーバからのデータを受信する
47       print(sock.recv(bufsize).rstrip())
48   return
49
50 if __name__ == '__main__':
51   main()
```

▼リスト6　udp_client.py

```
 1 # -*- coding: utf-8 -*-
 2 #
 3 from __future__ import print_function
 4 import sys
 5 import socket
 6 from contextlib import closing
 7
 8 def main():
 9   argv = sys.argv
10   argc = len(argv)
11
12   host = '127.0.0.1'
13   port = 3000
14
15   if argc == 3:
16     port = int(argv[2])
17     host = argv[1]
18   elif argc == 2:
19     host = argv[1]
```

右段へ続く↗

左段下から続く↙

```
20
21   bufsize = 4096
22
23   sock = socket.socket(socket.AF_INET,
24                        socket.SOCK_DGRAM)
25   sock.bind((host, port))
26
27   sys.stdout.write("input-> ")
28   line = sys.stdin.readline()
29
30   sock.sendto(line, (host, port))
31
32   print(str(sock.recv(bufsize)))
33
34   sock.close()
35
36   return
37
38 if __name__ == '__main__':
39   main()
```

せん。

このスクリプトがあれば、サーバにMailクライアントがインストールされていなくてもメールの送信ができます。バッチジョブの結果やWeb＋アプリケーションサーバのログをメールで送ることができて実用的です。

▼リスト7　echo_server.php

```
 1 <?php
 2 # ホストアドレス
 3 $host = "127.0.0.1"
 4 # ポート番号
 5 $port = 3000;
 6
 7 # ソケット(TCP)を作成する
 8 $socket = socket_create(AF_INET,
 9             SOCK_STREAM, SOL_TCP)
10 if (false == ($socket)) {
11   $errno = socket_last_error();
12   $msg = socket_strerror($errno);
13   echo "Create Error: " . $msg . "¥n";
14   exit(-1);
15 }
16
17 # 待受アドレスとポートを指定する
18 $err = socket_bind($socket, $host, $port)
19 if (false == $err) {
20   $errno = socket_last_error();
21   $msg = socket_strerror($errno);
22   echo "Bind Error: " . $msg . "¥n";
23   exit(-1);
24 }
25
26 # 接続待ちソケットを指定する
27 if (false == socket_listen($socket)) {
```

右段へ続く↗

左段下から続く↙

```
28   $errno = socket_last_error();
29   $msg = socket_strerror($errno);
30   echo "Listen Error: " . $msg . "¥n";
31   exit(-1);
32 }
33
34 echo "ServerStart:127.0.0.1:".$port."¥n";
35
36 # 接続待ちをする
37 while($connection = ➥
socket_accept($socket)) {
38   echo "Connecttion Start¥n";
39   # データを受信する
40   while(($input = ➥
socket_read($connection, 1024))) {
41     # 受信したデータが"exit"なら処理を終了する
42     if (trim($input) == "exit") {
43       break;
44     }
45     echo "Data: ".$input;
46     # データを返送する
47     socket_write($connection, ➥
"RET: ".$input);
48   }
49   echo "Connecttion Closed¥n";
50   # 接続を終了する
51   socket_close($connection);
52 }
53 ?>
```

▼リスト8　smtp_mail.rb

```
 1 #!/usr/local/bin/ruby
 2
 3 # -*- encoding: utf-8 -*-
 4
 5 require 'net/smtp'
 6 require 'time'
 7 require 'kconv'
 8
 9 # 設定
10 smtp_options = {
11   :host    => "mailserver.host",
12   :port    => 587,
13   :domain  => 'mailserver.host',
14   :user    => 'username',
15   :passwd  => 'P@ssword',
16   :from    => "from-user@mailaddress.jp",
17   :to      => "to-user@mailaddress.jp",
18   :subject => "Mail Subject",
19 }
20
21 # 送信するメッセージ(本文は標準入力から受け取る)
```

右段へ続く↗

左段下から続く↙

```
22 send_message = <<EOM
23 From: #{smtp_options[:from]}
24 To: #{smtp_options[:to]}
25 Subject: #{smtp_options[:subject]}
26 Date: #{Time.now.rfc822}
27 MIME-Version: 1.0
28 Content-Type: text/plain; ➥
charset=ISO-2022-JP
29 Content-Transfer-Encoding: 7bit
30
31 #{$stdin.read}
32 EOM
33
34 Net::SMTP.start(smtp_options[:host], ➥
smtp_options[:port], ➥
smtp_options[:domain], ➥
smtp_options[:user], smtp_options[:pass]) ➥
do |smtp    |
35     smtp.send_mail send_message, ➥
smtp_options[:from], smtp_options[:to]
36 end
```



スクリプト言語を使って、コネクション数と転送性能の関係を評価してみる

スクリプト言語なら、プログラムを修正してすぐに実行するといった具合に、機敏にトライ&エラーを実施できます。ここからは、前半部分で作成したプログラムをベースにして簡単な実験をしてみます。ネットワークプログラミングにおいて避けて通れないパフォーマンス測定とチューニングの雰囲気を味わってください。

クライアント(送り手)からサーバ(受け手)へデータを送信する単純な通信モデルで、少しずつ条件を変えて通信性能を確かめてみましょう。

1. 1つの接続で通信するよりも、複数の接続で通信するほうが速くなるのか
2. 環境(ネットワークのホップ数や場所、混雑状況)による差異はあるのか
3. 転送量による差異はあるのか
4. TCPとUDPの転送性能に違いがでるのか

これら複合的な条件下での性能を評価し、可能であればチューニングによってより高い性能が実現できるかを見ていきます。実験の前に、次のTCPとUDPの性質をおさらいしておきましょう。

・TCPは、A)パケットの到達について保証がある、B)輻輳(ふくそう)制御[注8]がある、という性質があります

・UDPは、A)ただ送るだけで届いたかどうかはプログラムの判断に任せられている、B)ネットワークの混雑状態に関係なく送りつけることができる、という性質があります

今回使用した環境

今回のベンチマークに使った環境の概略が**表2**になります。LAN接続は、サーバ、クライアントとも同じネットワークスイッチに接続しておきます。インターネット上のサーバとクライアントの接続状況は、tracerouteコマンドを使ってホップ数(途中のrouterの数)などを確認しておきます。スクリプトの制作はOS X上でおこない、ターゲット環境はLinuxです。一応OS Xでも動く程度のコンパチビリティは意識して設計します。

受け側プログラムは、TCP(tcp_receive_server.rb：**リスト9**[注9])とUDP(udp_receive_server.rb：**リスト10**[注9])をすべての測定で使います。単純化のために、バッファを解放する、エラー処理などはあえて入れていません。実際にサー

注8) ネットワークのトラフィック状況に応じ、一定のアルゴリズムで送出を調整するしくみ。

注9) 誌面非掲載。ファイル名とリスト番号の対応はp.47の表1参照。

▼表2　ベンチマークに使った環境

クライアント	
CPU	4core
OS	Ubuntu 16.04 LTS
Ruby	ruby 2.3.0p0(2015-12-25) [x86_64-linux-gnu]
サーバ LAN環境	
CPU	4core
OS	Ubuntu 16.04 LTS
Ruby	ruby 2.3.0p0(2015-12-25) [x86_64-linux-gnu]
サーバ Internet環境	
CPU	2core
OS	CentOS release 5.6(Final)
Ruby	ruby 1.8.6(2008-08-11 patchlevel 287) [x86_64-linux]
クライアントからのネットワーク距離	約14hops

ビスに組み込む場合には、適切な例外処理を施し、とくにUDPではアプリケーションレベルでのプロトコルを定義する必要があります。複数のポートを使って接続するマルチコネクションでは、最大16ポートで待ち受けるようにします。tcp_receive_server.rbの42行目で、listenを使ってbacklog数を16に設定しているところに注目してください。udp_receive_server.rbでは、マルチスレッドを使った多重化(マルチプレクス)処理をおこなっています。

シングルコネクションでの測定からはじめる

echo_server.rbをベースに簡単な送信プログラムを作成します。スクリプト言語の機敏さを活かして、はじめはマルチスレッド化やチューニングパラメータなどは意識せず、とにかくネットワーク通信ができるもっともシンプルなプログラムを作り、段階的に必要な機能やチューニング要素を加えていきます。サンプルソースコードはtcp_single_sender.rb(**リスト11**[注9])およびudp_single_sender.rb(**リスト12**[注9])です。

tcp_single_sender.rb、udp_single_sender.rbができあがったら、早速それぞれのプログラムを使って動作・性能を確認しましょう。

TCPのプログラムは問題ありませんが、UDPでは、Ubuntuでの送信サイズは65,507bytes、OS Xは9,187bytesを超えるとsendtoが失敗します。

```
message too long
```

大きなファイルを特定の数に分割して送信するロジックでしたが、sendtoへ一度に渡すデータサイズが大きすぎたためです。対策として、getsockopt関数で送信バッファサイズを取得し、必要であればsetsockoptでバッファサイズを広げておきます。

getsockoptで取得できる情報も短いスクリプトで簡単に確認できます。サンプルソースコードはgetsockopt.rb(**リスト13**[注9])です。SO_SNDBUF[注10]を確認すると、OS Xでは9,216byte、Ubuntuでは212,992byteです。manでsocket(7)を確認[注11]すると、setsockoptで設定した値に対して、管理用オーバーヘッドを考慮してカーネルはこの値を2倍にして扱うとあります。つまり、getsockoptでは2倍された値が返ってきます。

さらにUDPでの送信サイズは下位層のIPパケットの大きさに制約を受けますので、最大で65,507(65,535bytes − IPヘッダサイズ − UDPヘッダサイズ)なので、これを超えることはできません。途中の経路によっては、これよりもさらに小さなものしか送れないときもあります。

ここまではsendtoで一度に送るサイズをあえて65,507bytesに設定しましたが、現実的には大きすぎます。経路上のルータやサーバのMTU(Maximum Transfer Unit)を超えたサイズのパケットはフラグメント化されたり、途中の経路を通過できないこともあります。IPv4でのMTUは最大で1,500bytesでネットワークによって1,492bytes、1,454bytesなどと一定ではありません。実際には、sendtoで送るサイズを経路上の最小MTUよりも小さくするべきです。

UDPのプログラムは、この後のマルチコネクションも含めて、最大送出サイズに合わせて分割数を調整するようにプログラムを修正します。

最終的にTCP/UDPともsingle_sender.rbは使わなくなり、次項で考えるtcp_multi_sender.rb(**リスト14**[注9])とudp_multi_sender.rb(**リスト15**[注9])に集約することにし、指定できるチューニング・測定パラメータを次のようにしました。

- デバッグ用フラグ
- スレッド数(16でリミット)
- 1回で送るデータサイズ(UDPは <= 65535)
- ターゲットホスト
- ターゲットホストのポートの開始番号(UDP)

注10) このスクリプトを修正すればほかの値もすぐに確認できます。実際のチューニングやプログラミングのときには、きめ細かく使っているシステムの状況を把握しましょう。

注11) OS Xではsocket(7)のmanページはないようです。

・データファイル

再送回数、キューが取れなかった回数などを内部的にはカウントしていますが表示はしていません。本来であればこれらも調べるべきです。

マルチコネクションに対応する

Rubyのスレッドで複数のコネクションを処理してみましょう。

tcp_single_sender.rbとudp_single_sender.rbをもとに、マルチスレッド化したTCP(tcp_multi_sender.rb)、UDP(udp_multi_sender.rb)を作成します。

データを送信バッファサイズに合わせて分割し順次送信していた部分(single)を、マルチコネクション版では送信キューに順に入れていきます。送信キューの長さは、同時コネクション数と同じになるようにします。ここでも、できあがったらすぐに動作を確認します。

今度はUDPのsendtoで`too many open files`が発生しました。スレッドが作成される量が多すぎたため、ファイルディスクリプタ数が制限を超えたことが原因です。対策として、スレッド数を無制限に増やさず対向サーバのスレッド数(16)に合わせ、分割したメッセージをキューとみなして動作するよう修正します。併せてキューの数とスレッドの数を独立で調節できるようにしました。

まだ、問題は生じます。UDPのsendtoで`No buffer space available - sendto(2) for "xxx.yyy.zzz.aaa" port 3000`が、スレッド数が4を超えてから生じることがあります。スレッド処理をしていないときには起きなかった問題です。おそらく原因は、SO_SNDBUF(Ubuntuの場合212,992bytes)のサイズを超えてバッファを確保しようとしたためで、スレッドがこのバッファを獲得するときに前のスレッドの送信処理が完了せず、バッファに空きがない状態だと考えられます。このエラーが生じた場合には、例外処理でリトライをするように修正します。

マルチコネクション～シェルの機能を使う

さて次は、シングルコネクションのプログラムをRubyの外側、つまりシェルでマルチコネクションを実現しましょう。bashでは、マルチプロセスとして起動されるrubyプロセスが複数になり、それぞれのプロセスの終了を同期します。実行しているコンピュータにCPU(コア)が複数あれば、それぞれのプロセスに個別にCPUが張り付いて処理されることが期待できます。Ruby内部のスレッドで処理する場合と違いはあるでしょうか。

手っ取り早くやっつけで作成したシェルスクリプト(multi_sender.sh：**リスト16**[注9])を使ってベンチマークを測定します。ここまでに作ったtcp_multi_sender.rbとudp_multi_sender.rbをスレッド数＝1として並行に起動します。そこそこ正確で実用的なプログラムが、迅速に実現できるのもスクリプト言語ならではの便利さですね。

測定と問題・ボトルネックの検出

それぞれのスクリプトや、測定・チューニング箇所の設定と準備が整ったら実測です。

しかし、ここで思わぬ問題に突き当たりました。LAN環境で測定してもWAN環境で測定してもパフォーマンスがひどく悪いのです。実質8Mbpsぐらいでした。ps、topコマンドほかを駆使して、スレッド／プロセスは適切に動いているかなどいろいろと確認してみます。どうも、スレッド／プロセスは起動されているのですが、いずれもイベント待ちでスリープしているようです。さらに調べていると、測定用マシンが接続していた社内ラボ環境のネットワークスイッチ(いえスイッチではなくハブ！)、が10base-Tだったのです。UDPで全力送信して8Mbpsって、まるで10base-Tの性能だと思って調べたら、そうでした！　パケットモニタリング用にスイッチではなくハブを使っていた環境に接続したのが原因でした。ということで、こいつをFast Etherスイッチ(100Mbps)に置

き換えます。

ネットワークプログラム性能測定のヒント

一般的なプログラムは、アルゴリズムとデータ構造が決まればほぼ決定的(まぎれのない)に動き、同じ環境であればほぼ性能も一定でしょう。ネットワーク通信においてもLAN環境など、静的にルーティングされた環境では単独のマシンでの性能と大きな差は生じないかもしれません。しかし**動的にルーティング**されたインターネットなどの環境では、観測者からは一意に特定できない要素があり、測定のたびに結果が異なる非決定的(同じ条件であっても異なる結果となること)な振る舞いをします。

こうした環境では、一度測定した結果を鵜呑みにして常に同じように動作するだろうと結論づけては大火傷します。a)条件(時間帯など)をさまざま変更して複数回測定する、b)測定のタイミングなど同じ条件での測定でも複数回計測し、平均値／標準偏差などを確認する、といった物理学などの分野で実践されている統計的手法での分析アプローチが有効です。

これからのエンジニアには、解決法の定義重視の視点だけでなく、“モデル化－パラメータの調整－測定－評価”を反復的に繰り返し、妥当な帰結を導くデータサイエンティストなどに共通の技能が必須となってくることでしょう。さまざまな条件を素早く変更し、繰り返し測定することが求められることからスクリプト言語の活躍の場は、ますます広がっていきそうです。

結果と考察

今回の測定での結果を**図1～5**に示しました。当初挙げた5つの評価ポイントについて個々ベンチマークしています。

TCPでは同時コネクション数の調整と、クライアント環境でのThreadかマルチプロセスかがチューニング可能な要素でした。

UDPでは、転送量が多くなっても失敗はありませんでした。インターネット接続でも、比較的安定したルートだったようです。

LAN環境ではプロトコルやチューニング要素での大きな差異は現れませんでした。今回使ったWAN環境では、UDPのマルチコネクションが良い結果になりましたが、途中の経路によって結果は異なることがあります。

マルチコネクションは、インターネット環境のUDP通信では効果がありましたがTCPでの優位性は確認できませんでした。マルチスレッド化による恩恵はほとんどないのであれば、むしろCPU負荷が上がります。TCPでマルチコネクションを使う価値は、全体性能よりも、ユーザエクスペリエンス向上のために使うのがよさそうです。

測定の開始前は、TCPの輻輳制御アルゴリズム[注12]があるので、送りたいデータ量によって性能に変化が生じるかと予測しましたが、今回の測定範囲(データ量ほか)からは、TCPのウィンドウサイズの相関は確認できませんでした。

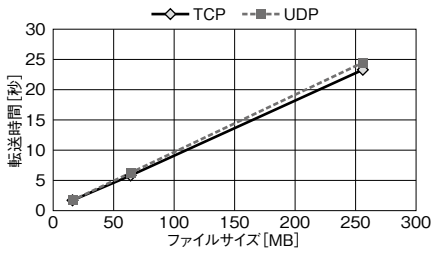
TCPでは、writeのサイズの制限を設けていません。内部で適切に処理していますので、アプリケーションレベルで分割するのはかえってオーバーヘッドが大きくなります。TCPは大きなデータをそのままwriteしてもよさそうです。

一方、UDPのプログラムでは最大送信サイズや、送信の失敗時のリトライ方式などチューニングできる余地が多くあります。基本形だけでは不十分で、エラー時のリトライ処理、各層でのバッファ管理、途中経路、マシン性能を考慮した適切なパケットサイズとスレッド／プロセス数[注13]のほか、目的に合わせて調整できる要素がたくさんあります。刻々と接続経路や環境が変わるインターネット接続でも、良い性能がでるようにTCPのような輻輳制御の考えを取り入れてみるのも面白いかもしれません。

注12) ウィンドウサイズを、スロースタートで、2倍にしながら、閾値を超えると1つずつ、輻輳が生じると振り出しに戻る、というアルゴリズム。

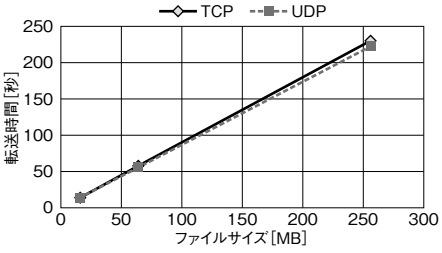
注13) スクリプト言語はC言語などで作成したネイティブプログラムに比べて、マルチプロセス時のコンテキストスイッチ負荷が高く、スレッドを使う場合の性能は低くなることにも留意しましょう。

▼図1　ネットワークロケーションによる影響 —ローカル接続



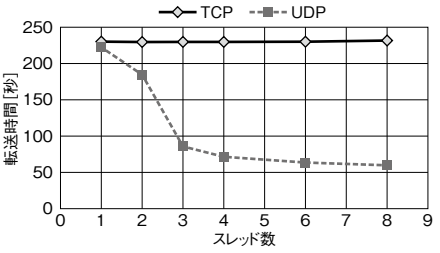
①ローカルネットワークにおける転送時間は、送信するファイルサイズに比例して遅くなる
②ローカルネットワークにおけるプロトコルによる転送時間に差異は見受けられなかった
③ローカルネットワークにおいては、分割数による影響はなかった

▼図2　ネットワークロケーションによる影響 —インターネット接続



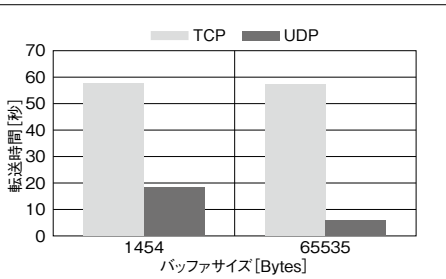
①インターネット環境においては分割送信しない場合、送信時間はファイルサイズに依存する
②ネットワーク転送コストがかかる分、ローカルネットワークに比べて転送時間は遅くなる

▼図3　分割数による効果



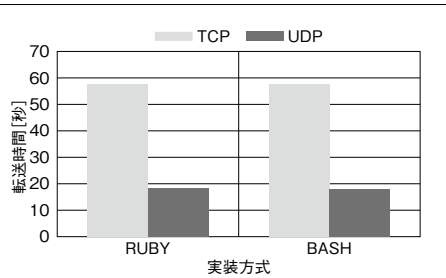
①インターネット環境において、分割送信をおこなった場合、UDP は分割数の増加にともない転送時間は短くなる
②ただし、転送時間については、分割数を増やしていっても特定の値に収束する
考察：分割数が増えていった場合、送信元、送信先双方のオーバーヘッドが高くなるため頭打ちになると考えられる

▼図4　バッファサイズによる影響



①TCP による送信では、バッファサイズを大きくしても転送時間への影響は見られない
②UDP による送信では、バッファサイズを大きくすることで転送時間を短縮することができる

▼図5　分割（スレッド）方式による影響



①スレッドの実装方式による違いは見受けられなかった
考察：Ruby1.9 以上で使用されるスレッドはネイティブスレッド、Linux の bash のプロセスと system time（カーネルモードでの負荷）などの値を測定し、システム負荷を比較するなどが必要か



まとめ

いかがでしたか。前半でさまざまなスクリプト言語によるsocket通信プログラムを見てきました。いずれの言語も基本形をベースにした使い方ですが、一部の言語では冗長な操作をしないでよい、イベント駆動に適した形の呼び出し方にするなど、インターフェースの工夫が見られました。

後半はほんの少しですが、スクリプト環境の機敏さを活かしたネットワークプログラム／環境のベンチマーキング＆チューニングの雰囲気を体験しました。ネットワークプログラミング／管理では、（とくにインターネット環境は流動的ですので）さまざまな視点で性能の測定－検討－施策を繰り返すことが重要です。ここから先は、あなた自身が測定とチューニングに挑戦してみてください。SD

第4章

パケットキャプチャで学ぶTCP/IP

Author 吉田 英二（Eiji James Yoshida）
合同会社セキュリティ・プロフェッショナルズ・ネットワーク（http://www.sec-pro.net/）

ここまでの章で、ネットワーク上でパケットがどのようにやりとりされているのかを把握できたのではないでしょうか。最後の章ではそのパケットに注目します。「ICMPパケット」「TCPパケット」「UDPパケット」をフリーソフトのWiresharkでキャプチャし、中身をのぞいてみましょう。

はじめに

みなさん、こんにちは！ Eiji James Yoshidaです。ネットワークについて勉強しようとTCP/IPの書籍を読んでみたけど、説明が複雑で何を言っているのかわからなかったり、書いてあるようなパケットの送受信を本当にやっているのかと疑問に思ったりしたことはないでしょうか。そこで今回は実際の通信で送受信されるパケットをキャプチャして、その解析結果からTCP/IPを学びましょう。

環境説明

本稿を書く際に使用した環境はWindows 10でWireshark 1.12.11です。Wiresharkは下記の公式サイトからダウンロードしてください。Wireshark 1.x系ですのでOld Stable Releaseになりますが、今のところ多くのユーザや書籍は1.x系だったりしますので、現時点ではこちらをお勧めします。

- **Wireshark**
  https://www.wireshark.org/download.html

今回の執筆時に作成したキャプチャファイルは、筆者のブログからダウンロードできます。

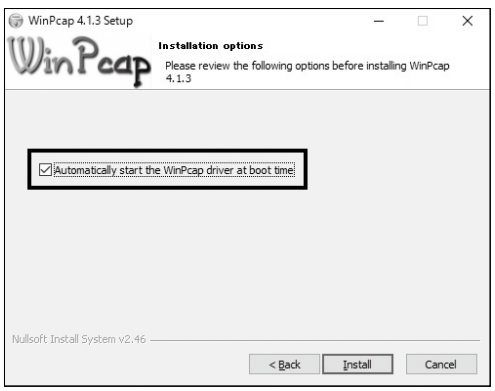
- **Software Design「パケットキャプチャで学ぶTCP/IP」用キャプチャファイル**
  http://d.hatena.ne.jp/EijiYoshida/20160520/1463728540

Wiresharkのインストール

Wiresharkのインストールは簡単で、とくにこだわりがなければ内容を確認しながら[Next]ボタンをクリックして進めるだけです。

途中、WinPcapのインストールで[Automatically start the WinPcap driver at boot time]チェックボックス（**図1**）が表示されますが、これはWinPcapドライバ(NetGroup Packet Filter Driverサービス)を自動起動させるためのものです。

▼図1　WinPcapのセットアップ・ウインドウ



このチェックを外してしまうと、パケットキャプチャをする際にWinPcapドライバを手動で起動させないと失敗するので、個人的にはこのチェックボックスにはチェックを入れた状態(自動起動)でインストールすることをお勧めします。

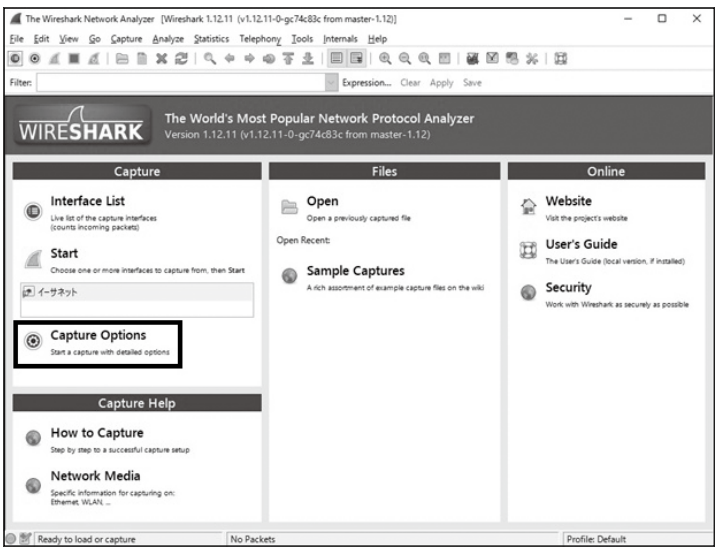
パケットをキャプチャしてみよう

Wiresharkのインストールが完了したら、さっそくパケットをキャプチャしてみましょう。

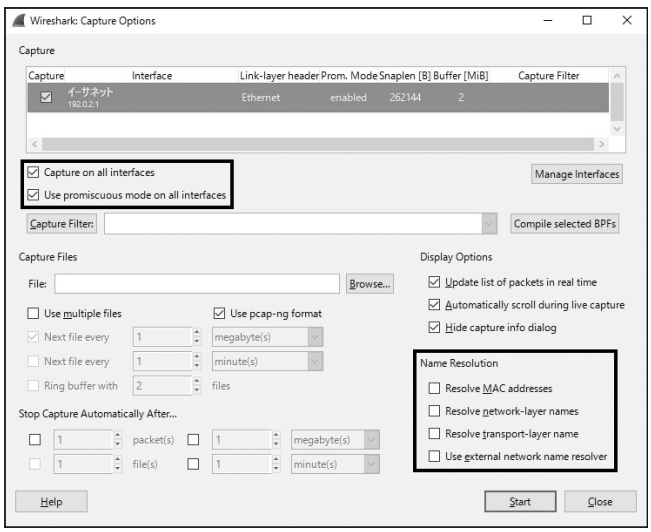
Wiresharkを起動するとウェルカム・スクリーン(**図2**)が表示されるので、左上の[Capture]の枠内にある[Capture Options]をクリックします。[Capture Options]ウインドウ(**図3**)が表示されたら、まずは[Capture on all interfaces]と[Use promiscuous mode on all interfaces]のチェックボックスにチェックが入っていることを確認します。もしチェックが外れている場合は入れてください。promiscuous modeとは無差別モードのことで、自分宛ではないパケットを受信してもインターフェースで捨ててしまわないようにする設定です。このチェックを外してしまうと、せっかく自分宛ではないパケットを受信してもインターフェースで捨ててしまうため、Wiresharkで解析できなくなります。あとは右下にある[Name Resolution]ですが、今回は名前解決しないので4つすべてのチェックを外してください。とりあえずここまで設定したら、ほかのオプションは気にせず[Start]ボタンをクリックしてパケットキャプチャを開始しましょう。パケットがインターフェースに届くと、promiscuous modeが有効であれば自分宛ではないパケットでも捨てずにWiresharkに渡され、解析結果が画面に表示されます。

試しにWiresharkを動かしているWindowsでコマンドプロンプトを起動して、下記のpingコマンドでほかのコンピュータ宛にパケットを

▼図2 ウェルカム・スクリーン



▼図3 [Capture Options]ウインドウ



送信してみましょう。

```
> ping 宛先IPアドレス
```

筆者の環境でping 192.0.2.2を実行したところ、**図4**の結果が表示されました。このようにPacket Listペインの Protocol項目にICMP、Info項目にEcho（ping）requestと表示されたパケットが取得できていればパケットのキャプチャに成功しているので、[Capture]メニューで[Stop]をクリックしてパケットキャプチャを停止してください。

もし何もパケットが表示されていなければ、[Capture on all interfaces]のチェックが外れている可能性が高いので、同様に[Capture]メニューで[Stop]をクリックしてパケットキャプチャを停止してから、再度[Capture]メニューで[Options...]をクリックして[Capture Options]ウインドウを表示、[Capture on all interfaces]のチェックボックスにチェックを入れてください。あとは[Start]ボタンをクリックしてパケットキャプチャを開始してからpingコマンドで再度パケットを送受信すれば、Wiresharkに解析結果が表示されるはずです。

Wiresharkの画面構成

Wiresharkの画面は下記3つのペイン（**図4**）で構成されています。

- Packet Listペイン
- Packet Detailsペイン
- Packet Bytesペイン

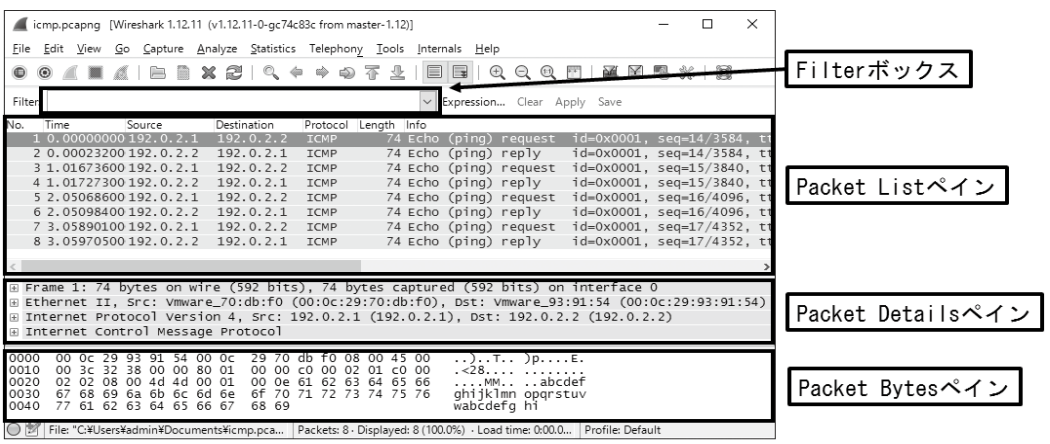
調べたいパケットをPacket Listペインでクリックすると、解析結果がPacket Detailsペインに表示され、パケットの16進数ダンプがPacket Bytesペインに表示されます。

Filterボックスには、条件式で指定されたパケットを画面に表示するディスプレイフィルタを入力します。ディスプレイフィルタの詳しい使い方は本誌2015年10月号の「Jamesのセキュリティレッスン第5回 ディスプレイフィルタの使い方を覚えよう！」をご覧ください。

ICMPパケットの解析結果を見てみよう

まずはpingコマンドで送受信されたICMPパケットだけを画面に表示したいので、Filterボックスにicmpと入力して Enter を押してください。

▼図4　メインウインドウの構成（ping 192.0.2.2を実行）



筆者と同じパケットを見ながら読み進めたい場合は、環境説明に書いてある筆者のブログからキャプチャファイル「icmp.pcapng」をダウンロードしてWiresharkで開いてください。Packet ListペインのProtocol項目にICMP、Info項目にEcho (ping) requestとEcho (ping) replyというパケットが表示されていると思います。Echo (ping) requestパケットの詳細な内容をPacket Detailsペインで確認してみましょう。

[Frame]と書かれた行を展開すると、ここにはパケットをキャプチャした時間や使用したインターフェースの識別子といった情報が表示されます。[Ethernet]と書かれた行や[Internet Protocol]と書かれた行を展開すると、ここにはEthernetヘッダやIPヘッダの解析結果が表示されます。[Internet Control Message Protocol]と書かれた行を展開すると、ここにはICMPパケットの解析結果が表示されます。

そもそもpingコマンドが何をしているのかというと、ICMP Echo Requestパケットを相手のコンピュータに送信して、そのパケットに対して返信されたICMP Echo Replyパケットを受信して結果を表示しています。"Echo"とあるように「やまびこ」と同じです。実際にEcho (ping) requestパケットとEcho (ping) replyパケットのPacket Detailsペインにある[Internet Control Message Protocol]をそれぞれ展開して内容を確認すると、Echo (ping) requestパケットのIdentifier、Sequence number、Dataの値が、そのままEcho (ping) replyパケットの同じ項目にコピーされていることがわかります。

簡単に言えば、pingコマンドで「ヤッホー！」というデータが格納されたICMP Echo Requestパケットを送信して、相手から「ヤッホー！」というデータが格納されたICMP Echo Replyパケットが返信されていれば、自分が送信したデータがそのまま相手に届いて、さらに相手から返信されたデータもそのまま自分に届いていることが確認できるということです。つまり、このようなしくみによって簡単な疎通確認ができるのです。ちなみにICMPは、TypeとCodeの値の組み合わせで何のメッセージなのか相手に伝えています。Typeの値が8でCodeの値が0だとEcho Requestのメッセージ、TypeとCodeの値が0だとEcho Replyのメッセージとして扱われます。



TCPパケットをキャプチャして解析結果を見てみよう

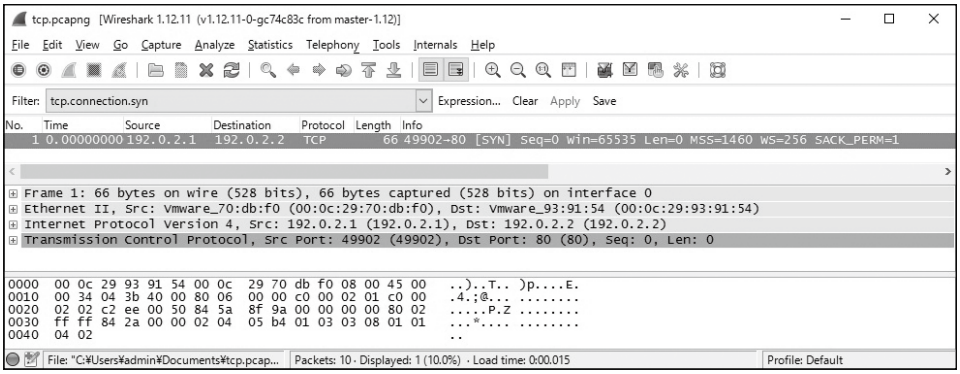
今度はTCPパケットをキャプチャして解析結果を見てみましょう。先ほどの手順でパケットキャプチャを開始します。その後、Wiresharkを動かしているWindowsでブラウザを起動してWebページを表示してください。あまり大きなWebページだと大量のパケットが送受信されて解析結果を確認したいパケットを探すのがたいへんになるので、表示するなら「http://www.example.com/」といったシンプルなWebページがお勧めです。

筆者の環境では「http://192.0.2.2/」を実行したところ、Wiresharkに**図5**の結果が表示されました。このようにPacket ListペインのProtocol項目にTCPまたはHTTPと表示されたパケットが取得できていればパケットのキャプチャに成功しているので、[Capture]メニューで[Stop]をクリックしてパケットキャプチャを停止してください。筆者と同じパケットを見ながら読み進めたい場合は、キャプチャファイル「tcp.pcapng」をダウンロードしてWiresharkで開いてください。

Filterボックスにtcp.connection.synと入力して Enter を押すと、Packet ListペインのInfo項目に[SYN]とあるパケットだけが表示されたかと思います。これがSYNパケット(TCPヘッダの制御フラグにSYNビットが設定されたパケット)になります。「tcp.pcapng」ではNo.1パケットに該当します。

Packet Detailsペインの[Frame][Ethernet]

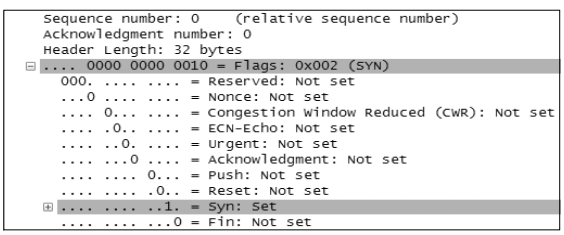
▼図5　ブラウザが送信したSYNパケット



[Internet Protocol]については先ほど説明したので省略しますが、その下の[Transmission Control Protocol]にはTCPパケットの解析結果が表示されます。展開して内容を確認すると、TCPヘッダのフォーマットと同じで[Source Port]から始まり、1行ずつ確認しながら下に進むと[Header Length]の次に[.... 0000 0000 0010 = Flags: 0x002 (SYN)]という行が見つかります。ここには制御フラグの設定状況が表示されていて、**図6**のように展開すると、SYNはSetになっていますが、ほかはNot setですのでSYNパケットということが確認できます。

続いてSequence numberとAcknowledgment numberの値を確認してみましょう。先ほどの[Header Length]の上に両方ともあるので値を確認すると、Sequence number: 0 (relative sequence number)とAcknowledgment number: 0と表示されています。これだけみると両方の値に0が入っているように思いますが、Wiresharkはデフォルトで相対的(relative)な表示をするため最初の値は0と表示されます。実際の値を表示するには[Sequence number]の行の上で右クリックをして、ポップアップメニューから[Protocol preferences]を選び[Relative sequence numbers](**図7**)をクリックしてチェックを外します。再度[Sequence number]を確認すると0ではない値が入っていると思いますが、それが実際の値になります。[Acknowledgment number]の値は0のままですが、それは実際に0が入っているので正しいです。ちなみにPacket Listペインの Info項目にあるSeq=とAck=にも、Sequence numberとAcknowledgment numberの値が表示されています(**図8**)。

▼図6　TCPヘッダの制御フラグフィールド

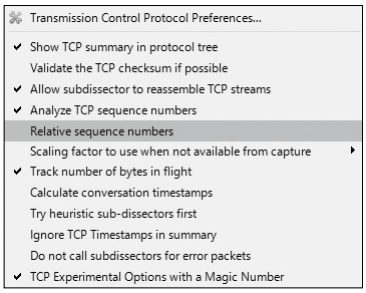




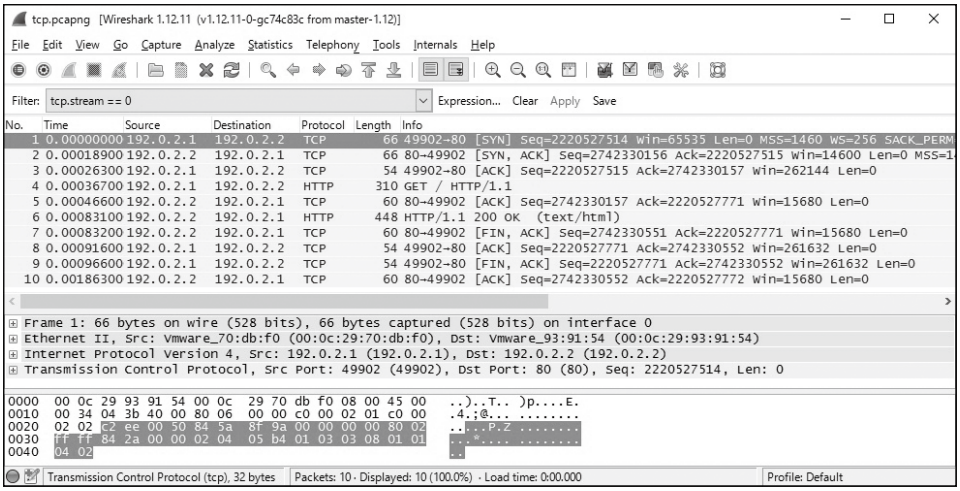
3way handshakeのパケットを見てみよう

Sequence numberとAcknowledgment number、制御フラグを確認する方法がわかったと思いますので、TCP接続で行われる3way handshakeのパケットを見てみましょう。

▼図7　[Relative sequence numbers]の無効化



▼図8　3way handshakeで送受信されたパケット



Filterボックスに`tcp.stream == 0`と入力して Enter を押してください。ここでは、Wiresharkが各TCP通信に通し番号を付けているので、その通し番号で指定したTCP通信のパケットだけをディスプレイフィルタで表示させています(**図8**)。Packet ListペインのInfo項目を見ると`[SYN]`、`[SYN, ACK]`、`[ACK]`と連続した3つのパケットが見つかると思いますが、これが3way handshakeで送受信されるパケットです。「tcp.pcapng」ではNo.1からNo.3までが3way handshakeで送受信されたパケット(**図9**)です。それぞれのパケットのSequence numberとAcknowledgment numberを確認すると、SYNパケットのSequence number(2220527514)に1加算した値(2220527514+1=2220527515)がSYN+ACKパケットのAcknowledgment numberに入っていて、さらにSYN+ACKパケットのSequence number(2742330156)に1加算した値(2742330156+1=2742330157)がACKパケットのAcknowledgment numberに入っていることがわかります。

このように3way handshakeや、のちほど説明する接続終了では、受信したパケットのSequence numberに1加算した値をAcknowledgment numberに入れて返信することで、相手にどのパケットに対して返信されたパケットなのかがわかるようになっています。

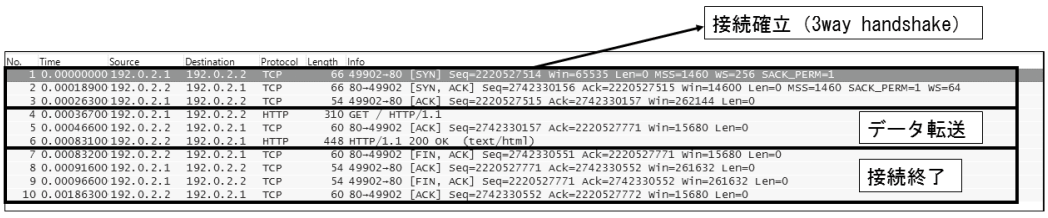
TCPは3way handshakeが完了すると接続確立済み状態(ESTABLISHED)になって、データ転送が始まります。



データ転送のパケットを見てみよう

3way handshakeが完了するとデータを含んだPSH+ACKパケットやACKパケットを送受信しているのがわかると思います。TCPでデータを転送するには基本的にACKパケット(ACK

▼図9　TCP通信の接続確立から終了までの流れ



ビットを設定したパケット)を使います。PSHビットも設定していたりしますが、最近はPSHビットを設定しても何も変わらないことが多いので気にしないでください。

「tcp.pcapng」ではNo.4からNo.6までがデータ転送で送受信されたパケット(**図9**)です。Packet ListペインでNo.4のPSH+ACKパケットをクリックしてPacket Detailsペインを見てください。[Transmission Control Protocol]の下に[Hypertext Transfer Protocol]があるので展開すると、**図10**のようにGETから始まる複数行が表示されます。これが、192.0.2.1の49902/tcpから192.0.2.2の80/tcpへ送信されたデータで、256バイトあります。このNo.4のPSH+ACKパケットに対して返信されたのがNo.5のACKパケットで、データは含まれていませんがPSH+ACKパケットのSequence number(2220527515)にデータサイズ(256)を加算した値(2220527515+256=2220527771)がAcknowledgment numberに入っています。

データ転送ではこのように、データが格納されていたACKパケットのSequence numberにデータサイズを加算した値をAcknowledgment numberに入れてACKパケットを返信することで、データを何バイト受け取ったのかを相手に伝えています。実際の通信だとデータ転送の効率を良くするために連続してパケットを送信したりするので、パケットの流れはもっと複雑だったりします。



接続終了のパケットを見てみよう

データ転送が終わってTCP接続が不要になると、接続終了のためにFIN+ACKパケットとACKパケットを送受信しているのがわかると思います。「tcp.pcapng」ではNo.7からNo.10までが接続終了のために送受信されたパケット(**図9**)です。Packet ListペインでNo.7からNo.10までのInfo項目にあるSeq=とAck=の値を見てみましょう。

No.7は、192.0.2.2の80/tcpから192.0.2.1の49902/tcpにFIN+ACKパケットが送信されていることから、192.0.2.2から192.0.2.1に接続終了を伝えていることがわかります。

No.8は、先ほどのFIN+ACKパケットのSequence number(2742330551)に1加算した値(2742330551+1=2742330552)がAcknowledgment numberに設定されたACKパケットですので、192.0.2.1がNo.7のFIN+ACKパケットを受信したことを192.0.2.2に伝えているのがわかります。

さらにNo.9では、192.0.2.1も接続終了のために192.0.2.2の80/tcpにFIN+ACKパケットを送信して接続終了を伝えています。No.10は先ほどのFIN+ACKパケットのSequence number(2220527771)に1加算した値(2220527771+1=2220527772)がAcknowledgment numberに設定されたACKパケットですので、192.0.2.2がNo.9のFIN+ACKパケットを受信したことを192.0.2.1に伝えているのがわかります。



UDPパケットをキャプチャして解析結果を見てみよう

最後に、UDPパケットをキャプチャして解析結果を見てみましょう。先ほどの手順でパケットキャプチャを開始します。その後、Wireshark

▼図10　192.0.2.2の80/tcpへ送信されたデータ

```
⊞ Transmission Control Protocol, Src Port: 49902 (49902), Dst Port: 80 (80), Seq: 2220527515, Ack: 2742330157, Len: 256
⊟ Hypertext Transfer Protocol
  ⊞ GET / HTTP/1.1\r\n
    Accept: text/html, application/xhtml+xml, image/jxr, */*\r\n
    Accept-Language: ja-JP\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: 192.0.2.2\r\n
    Connection: Keep-Alive\r\n
    \r\n
    [Full request URI: http://192.0.2.2/]
```

を動かしているWindowsでコマンドプロンプトを起動して、下記のnslookupコマンドでDNSの名前解決のパケットを送受信してみましょう。

```
> nslookup ホスト名
```

筆者の環境でnslookup localhostを実行したところ、Wiresharkに**図11**の結果が表示されました。このように、Packet Listペインの Protocol項目にUDPまたはDNSと表示されたパケットが取得できていればパケットのキャプチャに成功しているので、[Capture]メニューで[Stop]をクリックしてパケットキャプチャを停止してください。

続いてUDP通信のパケットだけを画面に表示したいので、Filterボックスにudp.stream == 1と入力して Enter を押してください。筆者と同じパケットを見ながら読み進めたい場合は、キャプチャファイル「udp.pcapng」をダウンロードしてWiresharkで開いてください。Packet Detailsペインで[User Datagram Protocol]を展開すると、UDPヘッダにはSource Port、Destination Port、Length、Checksumの4つのフィールドしかないことがわかります。

このように、UDPは非常にシンプルな設計のため、TCPの3way handshakeのような接続確立や接続終了の段取りはないです。今回キャプチャしたのはDNSの名前解決のUDP通信ですが、TCPと違って最初からデータ転送を始めているのがわかると思います。



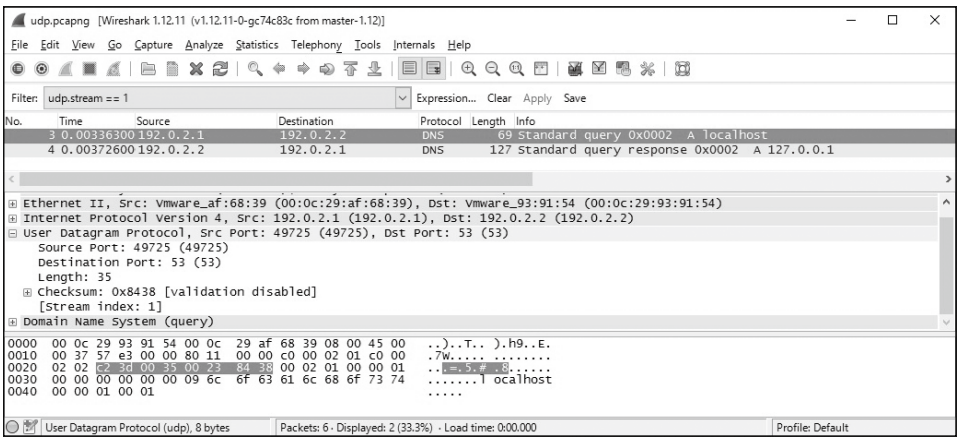
おわりに

パケットキャプチャで学ぶTCP/IPということで、実際の通信で送受信されるパケットをキャプチャして解析結果を見てみました。

pingコマンドで送受信されるICMPパケットや、ブラウザで送受信されるTCPパケット、nslookupコマンドで送受信されるUDPパケットといった身近なパケットを実際に見ることで、TCP/IPの理解がより進んだのではないでしょうか。筆者も同じように、ブラウザやポートスキャナが送受信したパケットをキャプチャして解析結果を読みながらTCP/IPを勉強しました。おかげで一般的なTCP/IPの書籍には書かれていないポートスキャンやOS推測の知識も身に付けることができました。みなさんもしくみが気になる通信があれば、パケットをキャプチャして解析結果を読んでみてください。

本稿の内容が少しでもみなさんのお役に立てば幸いです。SD

▼図11　nslookupコマンドが送受信したUDPパケット



第2特集

プログラミング／エディタ作業の効率大幅アップ

手を動かして学ぼう 正規表現入門

記事とWebツールでトレーニング

Author 伊藤 淳一　㈱ソニックガーデン

文字列の検索・置換が効果的に行え、エディタでのテキスト編集やプログラミングに役立つ正規表現。マスターすれば多くの作業で効率アップが期待できます。

正規表現の地味なメタ文字を、いち早く覚えるには実践が一番。本特集ではWeb上で正規表現を試せる「Rubular」というツールを使って、ハンズオン形式で解説します。みなさんも本誌を片手にRubularで同じことを試してみてください。本特集の内容をすべて実践して、正規表現の基本を自分のものにしましょう。

序章 正規表現は絶対にマスターしておけ！ …… p.68

第1章 さまざまな形式の電話番号を検索しよう …… p.71

第2章 カタカナ語の揺れを許容して検索しよう …… p.78

第3章 HTMLをCSVに置換しよう …… p.81

第4章 スペースや改行などの空白文字を検索／置換しよう …… p.87

第5章 正規表現のパフォーマンスを意識しよう …… p.91

序章 正規表現は絶対にマスターしておけ!

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

正規表現を学ぶわけ

「正規表現は絶対にマスターしておけ！」――これは筆者がプログラマとして働き始めて間もないころに、同じ職場の先輩プログラマ・Nさんから言われた言葉です(しかも、なぜかNさんが会社を退職する日に！)。当時はそもそもプログラミングの経験自体ほとんどなかったので、「正規表現」という言葉は初めて聞きました。当然、正規表現が何をするものなのかも知りませんでした。

あれから10年以上経ちました。あの日以来、筆者はNさんに会っていないので、彼が今どこで何をしているのかわかりません。しかし、Nさんの言葉を信じて勉強を始めた正規表現は今でも毎日のように役立っています。正規表現が使えるようになって本当に良かったと思いますし、正規表現が使えなかったら今のようなスピードで仕事をするのは絶対無理だと思います。

Nさんの言葉は、まさにそのとおりでした。なので、筆者からも読者のみなさんにこう言わせてください。「正規表現は絶対にマスターしておけ！」。

そもそも正規表現って何?

Wikipediaで正規表現を調べると次のように書いてあります。

> *正規表現(せいきひょうげん、英: regular expression)とは、文字列の集合を一つの文字列で表現する方法の一つである。*
>
> *(正規表現 - Wikipedia)*[注1]

うーん、これだけだとなんかよくわかりませんね……。

Wikipediaの説明を無視して、極めて個人的な視点で正規表現を説明するなら、

「パターンを指定して、文字列を効率よく検索・置換するためのミニ言語」

かな、と筆者は考えています。

まあどちらの説明を読んでも、初心者の方はピンと来ないでしょう。でも大丈夫です。とりあえず本特集を読み進めていけば、最後には正規表現が何なのか、どこがどう便利なのかわかってくるはずです。

なお、Wikipediaの説明にもあるとおり、正規表現は英語でRegular expressionと言います。プログラミング言語やエディタ上では略して、RegexやRegexpと書かれることが多いので、英語の略称も頭の片隅に置いておきましょう。

正規表現を覚えると何がうれしいの?

正規表現を覚えると、テキスト処理の効率が圧倒的に上がります。ここでいうテキスト処理とは、文字列の検索や置換のことです。

また、正規表現は環境によって使えるメタ文字(後述)が多少異なったり、方言レベルの差異があったりしますが、基本的な仕様は同じです。なので、一度覚えればプログラミング言語やテキストエディタが変わっても同じように活用できます。

逆に正規表現が使えないと、大量のテキストから目視で目的の文字列を探したり、何度も検索条件を変えながら検索したりする必要が出て

注1) URL https://ja.wikipedia.org/wiki/正規表現

きます。プログラム内でテキストを検索・置換するときも、正規表現を使わない場合はテキストを行単位や文字単位でループさせながら何度も条件分岐を重ねるようなコードになりがちです。

正規表現あり・なしでコードを比較してみる

とはいえ、「正規表現はすごい！　みんな覚えよう！」と文章だけで力説されても、正規表現を知らない人にとっては「ふーん、そうなのかなあ」で終わってしまうかもしれません。そこで簡単なRubyスクリプトを使って、正規表現を使う場合と、使わない場合のコードの違いを比較してみましょう。

ここではRubyコードや正規表現の細かい意味や仕様を理解する必要はありません。「正規表現を使うか使わないかでこれぐらいの違いが出るんだ」ということをざっくりと理解してもらえればOKです。

設問：テキストから電話番号だけを抜き出してください

次のようなテキスト(文字列)があります。

```
サンプルテキスト 0-1
今日もおいしいパンをたくさん焼いて、ご来店 ﾃﾞ
をお待ちしています！
お取り置きのお電話は0795-12-1234までどうぞ。
※来週の5/6-7はお休みです。
```

この中から電話番号を抜き出すプログラムをRubyで作ってください。つまり、"0795-12-1234"を抜き出すのがゴールです。

正規表現を使わない場合

正規表現を使わずに電話番号を抜き出そうとすると、**リスト1**のようになります。

正規表現を使う場合

一方、正規表現を使うと**リスト2**だけで済みます。\d{4}-\d{2}-\d{4}の部分が正規表現です。

▼リスト1　電話番号を抜き出すRubyプログラム（正規表現を使わない場合）

```
# 引数が0～9、またはハイフンであればtrueを返すメソッド
def tel_char?(c)
  %w(0 1 2 3 4 5 6 7 8 9 -).include?(c)
end

text = <<-TEXT
今日もおいしいパンをたくさん焼いて、ご来店をお待ちしています！
お取り置きのお電話は0795-12-1234までどうぞ。
※来週の5/6-7はお休みです。
TEXT

tels = []   # 電話番号の候補を格納する配列
prev = ''   # 1つ前の文字
buffer = '' # 電話番号文字列を作成するときに使うバッファ
# 1文字ずつループ
text.chars.each do |c|
  if tel_char?(c)
    if !tel_char?(prev)
      # 1つ前の文字が電話番号文字以外なら新しいバッファを追加
      buffer = ''
      tels << buffer
    end
    # バッファの末尾に文字を追加
    buffer.concat(c)
  end
  # 現在の文字を1つ前の文字として保存
  prev = c
end
# 見つかった候補の中から、12桁の文字列を選択して返却
tel = tels.find{|s| s.size == 12}

puts tel
# => "0795-12-1234"
```

▼リスト2　電話番号を抜き出すRubyプログラム（正規表現を使う場合）

```
text = <<-TEXT
今日もおいしいパンをたくさん焼いて、ご来店をお待ちしています！
お取り置きのお電話は0795-12-1234までどうぞ。
※来週の5/6-7はお休みです。
TEXT

tel = text[/\d{4}-\d{2}-\d{4}/]

puts tel
# => "0795-12-1234"
```

正規表現が使えるかどうかでこれだけ差が付く

いかがでしょうか？　正規表現を使うと実質1行で電話番号の抜き出しが完了しています。2つのプログラムの違いは長いか短いかだけではありません。正規表現を使わないプログラムには次のような欠点があります。

・複雑なのでデバッグやテストをするのがたいへん
・もろくて壊れやすい（予期せぬ入力や仕様変更に弱い）
・第三者がコードの内容を読み解くのがたいへん

一方、正規表現を使えばコードが短いぶん内容を理解するのも簡単ですし、入力値のバリエーションが増えた場合も正規表現を変更するだけで対応できます。

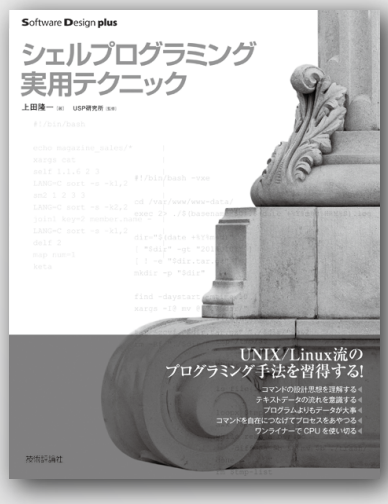
正規表現を「恐ろしい呪文」から「心強い武器」に変えよう

もちろん、正規表現がまったくわからない人にとっては**リスト2**のプログラムは意味不明で、**リスト1**のようなプログラムのほうが多少長くても精神的に落ち着く、と感じるかもしれません。しかし、もし読者のみなさんがそうだったとしても、そこで終わらないでください！

`\d{4}-\d{2}-\d{4}`が恐ろしい呪文にしか見えない人でも、本特集を一通り読めばきっと意味がわかるようになっているはずです。そして、実務で繰り返し使っていくうちに正規表現があなたの「心強い武器」に変わっていくはずです。

さあ、がんばって正規表現をマスターしましょう！SD

Software Design plus　技術評論社



シェルプログラミング実用テクニック

月刊誌『Software Design』の2012年1月号～2013年12月号で連載していた「開眼シェルスクリプト」の内容を大幅に加筆／修正し、書籍にまとめました。
LinuxやUNIXのコマンドは単独で使うよりも、複数のコマンドを組み合わせてこそ真価を発揮します。テキストデータの検索／置換／並べ替え、ファイルのバックアップや削除、数値や日付の計算など活用範囲は無限大。シェルは、端末にコマンドを入力してすぐに実行できるのも良いところ。その場かぎりの作業にこそ、ちょちょいとシェルプログラミングが使えると便利です。本書のいくつもの実例を順に見ていけば、コマンドを自在に組み合わせるために必要なシェルの機能と考え方が身につきます。

上田隆一 著 USP研究所 監修
B5変形判／416ページ
定価（本体2,980円+税）
ISBN 978-4-7741-7344-3

大好評発売中!

こんな方におすすめ
・Linux/UNIX利用者全般、プログラマ、インフラエンジニア
・コマンドを自在に組み合わせるコツを知りたい方
・大量のテキストデータの編集や集計を高速に行いたい方
・手作業でやっている作業を自動化したい方

第1章 さまざまな形式の電話番号を検索しよう

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

今回のお題・動作環境

さて、ここからが本編です。前章でも電話番号を抜き出すサンプルコードを紹介しましたが、本章でも同じようにテキストの中から電話番号を検索する処理を考えてみます。ただし、内容は若干変えて次のようなテキストを使います[注1]。

サンプルテキスト 1-1
名前：伊藤淳一
電話：0795-12-1234
住所：兵庫県西脇市なんとか町1-2-3

ここから“0795-12-1234”を抜き出すのがここでのゴールです。

説明に使用する言語とエディタについて

本記事では説明のため、プログラミング言語はRuby 2.3、テキストエディタはAtom 1.7を使っています。

とはいえ、Rubyがわからないとまったく読み進められない、というわけではありません。正規表現はそれ自体が独立したミニ言語みたいなものなので、プログラミング言語やテキストエディタを問わず同じように使えます。また、サンプルコードの内容はごく簡単なものばかりなので、ほかの言語で書き直すのも難しくないはずです。ただし、言語や環境によって微妙に使える機能や挙動が異なったり、言語仕様に合わせたエスケープ文字を入れる必要があったりするので、その点は注意してください。

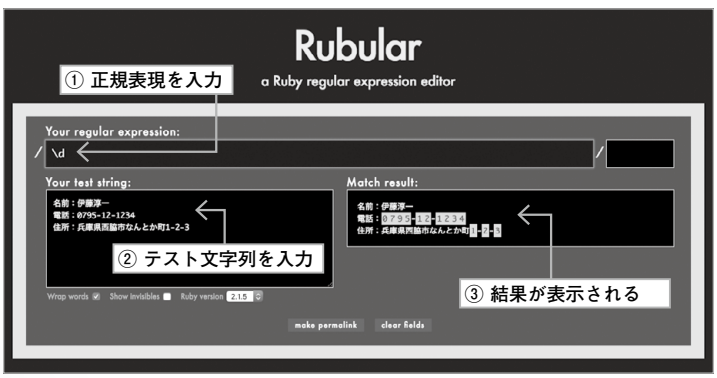
また、記事の中では正規表現の動きを視覚的に確認できるオンラインツールを使っています。このツールを使うと正規表現のしくみがよくわかるので、実際に自分の手で動かすことを強くお勧めします。

Rubularを使って視覚的に正規表現を確認する

正規表現を作るときは、一発で思いどおりの正規表現を作るのはなかなか難しいです。とくに、正規表現にあまり慣れていなかったり、複雑な正規表現を作ったりする場合は何度もトライアンドエラーを繰り返すことになると思います。そこで、今回は手軽に正規表現を試せて、なおかつその結果を視覚的に確認できるRubular[注2]というオンラインツールを使います。

Rubularを開いたら、とりあえず次のように入力してください(**図1**)。

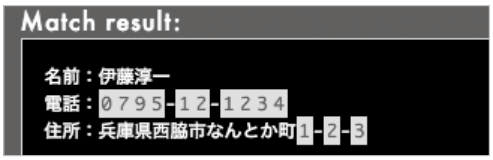
▼図1 Rubularで正規表現を確認する



注1) サンプルテキストは本誌サポートページ(http://gihyo.jp/magazine/SD/archive/2016/201607/support)よりダウンロードできます。
注2) http://rubular.com/

・Your regular expression（正規表現入力欄）
　＝\d
・Your test string（テスト文字列入力欄）
　＝冒頭のサンプルテキスト

入力したら、Match result（結果表示欄）に注目してください。次のような結果になるはずです。



背景色の付いている文字が正規表現にマッチした部分、すなわち検索して見つかった部分です。ここではすべての半角数字に背景色が付いています。このようにRubularを使うと、自分の作った正規表現の結果をすぐに確認できます。

\dは半角数字を表すメタ文字

さて、先ほど入力した\dの意味を説明しましょう。\dは**メタ文字**と呼ばれるもので、正規表現の中では特別な意味を持ちます。\dの意味は「1個の半角数字（0123456789）」です（文字の集合を表しているので、とくに**文字クラス**と呼ばれます）。

普通の検索であれば\dは“\d”という文字列そのものを検索しますが、正規表現の中で\dと書くと、「1個の半角数字」を検索することになります。なので、先ほどのRubularの結果表示欄では、すべての半角数字に背景色が付いていたわけです。

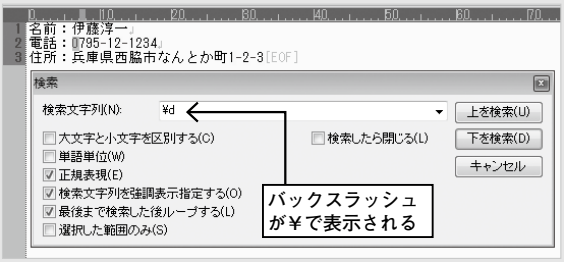
今後、正規表現中に\dが出てきたら、「あ、これは半角数字を意味してるんだな」と頭の中で意味を変換してください。

Ⓒ コラム　バックスラッシュ（\）はどうやって入力するの？

Windowsの場合、日本語フォントを使っているとバックスラッシュが¥で表示される場合があります（**図A**）。これはフォントの都合なので、そのまま¥を使っても構いません。この記事を読み進めるときは適宜バックスラッシュを¥に置き換えて読み進めてください。

また、MacでJISキーボードを使っている場合は、Optionキーを押しながら¥キーを押すとバックスラッシュを入力できます。

▼図A　Windowsのエディタで正規表現を入力した場合

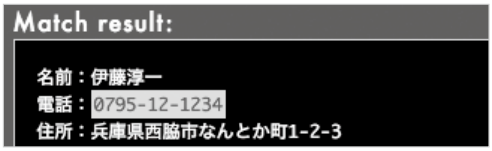


メタ文字をつなげて文字列を検索してみる

さて、このままだと、まだ電話番号だけをきれいに検索できていません。そこで次のような正規表現を入力してください。

```
\d\d\d\d-\d\d-\d\d\d\d
```

こうすると電話番号だけをきれいに検索できるはずです。



\dは「1個の半角数字」なので、\d\d\d\dは半角数字4個の意味になります。「-」はメタ文字ではないのでハイフン（“-”）そのものを表します（ただし、使い方によっては「-」も特別な意味

を持つので後述します)。よって、\d\d\d\d-\d\d-\d\d\d\d は「半角数字4個、ハイフン、半角数字2個、ハイフン、半角数字4個」が並んだ文字列を検索したことになります。

\d を使っている部分は半角数字であれば何でもマッチします。電話番号の“0795-12-1234”を何か別の番号、たとえば“0999-99-9999”に変えてみましょう。この電話番号でも「半角数字4個、ハイフン、半角数字2個、ハイフン、半角数字4個」というパターンに合致するので、ちゃんとマッチするはずです。

```
Match result:
名前：伊藤淳一
電話：0999-99-9999
住所：兵庫県西脇市なんとか町1-2-3
```

Rubyスクリプト内で正規表現を使ってみる

正規表現はRubularの中だけでなく、プログラミング言語内でも使用できます。**リスト1**はRubyで正規表現を使う例です。/ /で囲まれた部分が正規表現(Regexpクラスのインスタンス)になります。

▼リスト1　Rubyにおける正規表現の使用例

```
text = <<-TEXT
名前：伊藤淳一
電話：0795-12-1234
住所：兵庫県西脇市なんとか町1-2-3
TEXT
text.scan /\d\d\d-\d\d-\d\d\d\d/
# => ["0795-12-1234"]
```

リスト1ではscanメソッドを使いましたが、Rubyにはscanメソッド以外にも正規表現が使えるメソッドや演算子が数多くあります。

テキストエディタ(Atom)上で動かしてみる

正規表現はプログラミング言語だけでなく、テキストエディタでも使えます。プログラマ向けのテキストエディタであれば、大半のものは正規表現を使って検索や置換ができるはずです。ここでは参考にAtomでの実行結果を載せておきます(**図2**)。

ご覧のとおり、Atomでも正規表現を使って電話番号を検索できました。ちなみに正規表現を使う場合は画面右下の「.*」をオンにする必要があります。

このように、正規表現を覚えると言語やエディタを問わず、いろんな場面で活用できるようになります。

▼図2　Atomにおける正規表現の使用例



いろんな桁の市外局番に対応させる

ところで、電話番号はいつでも「半角数字4個、ハイフン、半角数字2個、ハイフン、半角数字4個」とは限りません。携帯番号であれば“090-1234-5678”のようになりますし、東京だと“03-1234-5678”のように市外局番が2桁です。また、地域によっては“04992-1-2345”のように5桁の電話番号もあるようです。

そこで、正規表現を変更してどの市外局番にも対応できるようにしてみましょう。次のテキストをRubularに入力してください。

```
サンプルテキスト 1-2
名前：伊藤淳一
電話：03-1234-5678
電話：090-1234-5678
電話：0795-12-3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

次のように、先ほどの正規表現だと市外局番が4桁の電話番号以外を正しく検索できません。

```
Match result:
名前：伊藤淳一
電話：03-1234-5678
電話：090-1234-5678
電話：0795-12-3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

これら4つの電話番号にはどんな法則があるでしょうか？　正規表現を使うときは、最初に検索対象文字列の**パターン**をうまく見つけだすのが重要です。今回であれば、次のようなパターンになっていると言えそうです。

「半角数字が2個〜5個、ハイフン、半角数字が1個〜4個、ハイフン、半角数字が4個」

あとはこれを正規表現として表すことができればOKです。

{n,m}や{n}は個数を限定するメタ文字

文字の個数を限定するときは{n,m}や{n}というメタ文字を使います(文字量を指定するので、**量指定子**と呼ばれます)。{n,m}は「直前の文字がn個以上、m個以下」の意味です。また、{n}とすれば「ちょうどn文字」の意味になります。

よって正規表現は次のようになります。

```
\d{2,5}-\d{1,4}-\d{4}
```

Rubularにこの正規表現を入力すると、正しく電話番号を検索できました！

```
Match result:
名前：伊藤淳一
電話：03-1234-5678
電話：090-1234-5678
電話：0795-12-3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

このように、正規表現では{と}も特別な意味を持っていることを覚えておいてください。

ハイフンだけでなく、括弧にも対応する

さて、もう少しだけ電話番号の正規表現を考えてみたいと思います。電話番号は"03(1234)5678"のように書くこともあります。つまり、ハイフンではなく括弧を使うケースです。ハイフンが来ても、括弧が来ても、どちらでも電話番号を検索できる正規表現はあるのでしょうか？　大丈夫です。そういう場合も正規表現で検索できます！　それではRubularに次のテキストを入力してください。

```
サンプルテキスト 1-3
名前：伊藤淳一
電話：03(1234)5678
電話：090-1234-5678
電話：0795(12)3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

先ほどの正規表現のままだと、括弧を使った電話番号は検索に失敗します。

```
Match result:
名前：伊藤淳一
電話：03(1234)5678
電話：090-1234-5678
電話：0795(12)3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

今回もやはりパターンを見つけることから始めます。この場合は次のようなパターンになっていると言えるでしょう。

「半角数字が2個〜5個、ハイフンまたは括弧(開き)、半角数字が1個〜4個、ハイフンまたは括弧(閉じ)、半角数字が4個」

新しく登場したパターンは「ハイフン**または**括弧(開き)」と、「ハイフン**または**括弧(閉じ)」という「AまたはB」のパターンです。

[ ]はいずれか1文字を表すメタ文字

「AまたはBのいずれか1文字」表す場合は[AB]と書きます(文字の集合を表すので、これも**文字クラス**の一種です)。[ ]の中の文字数に制限はありません。[ABC]と書けば、「AまたはBまたはCのいずれか1文字」の意味になります。

「ハイフンまたは括弧(開き)」と、「ハイフンまたは括弧(閉じ)」はそれぞれ正規表現の中では、[-(]、[-)]と表現します。記号だけが並んでい

てわかりづらいですが、よく見ると[AB]の形式になっていることを確認してください。これをふまえて正規表現を書き直すと次のようになります。

```
\d{2,5}[-(]\d{1,4}[-)]\d{4}
```

これをRubularに入力すると……ちゃんと電話番号を検索できました！

```
Match result:
名前：伊藤淳一
電話：03(1234)5678
電話：090-1234-5678
電話：0795(12)3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

メタ文字と普通の文字を区別できるようになろう

正規表現の知識がないと\d{2,5}[-(]\d{1,4}[-)]\d{4}は謎の暗号にしか見えませんが、この記事をここまで読んできた人であれば、メタ文字の意味と、メタ文字ではない普通の文字の区別がつくはずです。「これはメタ文字で、こういう意味」「これは普通の文字なので、そのまま検索」という区別ができれば、ほかの人が書いた正規表現もだんだん理解できるようになってきます。

ちなみに、今回出てきた半角の丸括弧((と))は、[ ]の外で使われると特別な意味を持ちます。それについてはこのあとの章で説明します。

やろうと思えば電話番号を使っていろんな正規表現を試せるのですが、どんどん章が長くなってしまうので、電話番号を使った正規表現はいったんここで終了します。

Cコラム [ ]の中のハイフンには要注意

[ ]の中に文字を書く場合、文字を並べる順番は自由です。通常、[AB]は[BA]と書いても同じ意味になります。

ただし、「-」だけは特別な意味を持つので注意が必要です。実は[a-z]と書くと、「aまたはbまたはcまたは……yまたはz」の意味になるのです。「aまたはハイフンまたはz」ではありません！

同様に、[a-zA-Z0-9]であれば「aまたはbまたは……z、AまたはBまたは……Z、0または1または……9」の意味になります(端的に表現するなら、これは「半角英数字のいずれか1文字」の意味です)。つまり、[a-z]におけるハイフンは「文字の範囲」を意味します。

ところが、[-az]や[az-]のように、ハイフンが[ ]の最初、または最後に置かれると「ハイフン1文字」の意味に変わります。なので、[-az]や[az-]は「aまたはzまたはハイフンのいずれか1文字」の意味になります。

▼リストA　サンプルテキスト1-4

```
名前：伊藤淳一
電話：03(1234)5678
電話：090-1234-5678
電話：0795(12)3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

実際に試してみましょう。Rubularにリストのテキストを入力してください。

[0-9]という正規表現を入力すると、半角数字のみにマッチします(図B)。つまり\dと同じです。

一方、[-09]と入力すると、「0または9またはハイフン」にマッチします(図C)。

このように、[ ]の中では使い方によってハイフンの意味が変わることを覚えておいてください。

▼図B　正規表現 [0-9] を入力した場合

```
Match result:
名前：伊藤淳一
電話：03(1234)5678
電話：090-1234-5678
電話：0795(12)3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

▼図C　正規表現 [-09] を入力した場合

```
Match result:
名前：伊藤淳一
電話：03(1234)5678
電話：090-1234-5678
電話：0795(12)3456
電話：04992-1-2345
住所：兵庫県西脇市なんとか町1-2-3
```

C コラム Rubular以外の正規表現テストツール

本特集で使用しているRubularは、Rubyの正規表現をテストするツールです。正規表現は環境ごとに微妙な方言があるため、環境に合わせてテストツールを使い分けたほうが無難です。類似の正規表現テストツールはいくつかあります。次のものはその一例です。

- JavaScript：Scriptular[注A]
- Java：Java regex tester[注B]
- PHP：PHP正規表現チェッカー[注C]

“java regex tester”のようなキーワードでネットを検索するといろいろ見つかるので、自分の環境に合ったテストツールを探してみてください。

なお、ブラウザ上で動作するテストツールは入力内容をサーバに送信するものが多いため、実在するパスワードやクレジットカード番号など、セキュリティ面で問題のある文字列は入力しないようにしましょう。

注A）http://scriptular.com/
注B）http://java-regex-tester.appspot.com/
注C）http://www.rider-n.sakura.ne.jp/regexp/regexp.php

C コラム 精度を取るか、シンプルさを取るか

今回は電話番号を検索するのに`\d{2,5}[-(]\d{1,4}[-)]\d{4}`という正規表現を使いました。しかし、これはあくまで「半角数字が2個～5個、ハイフンまたは括弧(開き)、半角数字が1個～4個、ハイフンまたは括弧(閉じ)、半角数字が4個」という文字列にマッチするだけです。電話番号だけを完璧に抜き出す正規表現ではけっしてありません。

この正規表現だと、検索対象の文字列によっては電話番号以外の文字列にマッチする場合もあります。たとえば、“9999-99-9999”は実在しない電話番号でしょうし、“03(1234-5678”のように、括弧とハイフンが混在しているおかしな文字列にもマッチしてしまいます(**図D**)。

こういったイレギュラーなケースをすべて除外しようと思うと、まず電話番号の仕様を完璧に理解し、その仕様を完全に満たす正規表現を作らなければなりません。そしておそらく、その正規表現はかなり複雑なものになるはずです。

▼図D　正規表現 \d{2,5}[-(]\d{1,4}[-)]\d{4} を入力した場合（サンプルテキスト1-5）

```
Match result:
名前：伊藤淳一
電話：03(1234)5678
？？：9999-99-9999
？？：03(1234-5678
住所：兵庫県西脇市なんとか町1-2-3
```

■ 99.99%正確なメールアドレスの正規表現？

筆者は電話番号の仕様を完璧に理解しているわけではないので、「100%正しい電話番号の正規表現」はちょっと作れません。そこで代わりにネットから「99.99%正確なメールアドレスの正規表現」[注D]を探してきました。なんと、**リストB**のような正規表現になるそうです。

ご覧のとおり、非常に複雑な正規表現になっています。毎回こんな複雑な正規表現を考えている時間はないでしょうし、もしうまく動かなかったときに正規表現を読み解いてデバッグするのもほぼ不可能ですね。

■ ときには妥協することも大事

というわけで、現実的には「100%完璧ではなく、**そこそこ正しい**レベルの正規表現」で妥協することがよくあります。そして、「そこそこ正しい」の度合いは要件によって変わってきます。

たとえば、大量のテキストから「電話番号っぽい文字列」を抽出して、人間が目視で判断するのであれば、今回作った`\d{2,5}[-(]\d{1,4}[-)]\d{4}`で

注D）http://emailregex.com/

▼リストB　99.99%正確なメールアドレスの正規表現

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-
\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-
9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[
0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-
\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

も十分でしょう。しかし、ユーザが入力するフォームの入力値チェックに正規表現を使うのであれば、もう少し精度を上げるべきかもしれません。たとえば、「電話番号の1文字目はゼロ、2文字目はゼロ以外の半角数字」とは言えそうなので、

```
0[1-9]\d{0,3}[-(]\d{1,4}[-)]\d{4}
```

にすれば、少し精度が上がります。**図E**のとおり、"9999-99-9999"はマッチしなくなりました。ですが、"03(1234-5678"にはまだマッチしています。

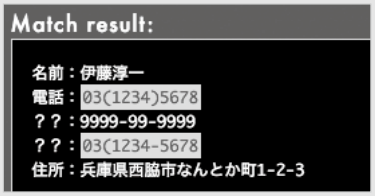
■ 正規表現とプログラムを併用するのも1つの手

RubyやJavaScriptといったプログラミング言語中で正規表現を使うのであれば、「正規表現を使ってそこそこの精度で文字列を抽出し、残りはプログラミング言語の機能を使って精度を上げる」というハイブリッドなアプローチを使うのも1つの手です。

たとえば、**リストC**のRubyスクリプトではgrepメソッドを使って"03(1234-5678"を除外しています。このとおり、正規表現とRubyのメソッドを併用することで、最終的に"03(1234)5678"だけを抽出できました。

ちなみに、**リストC**では、grepメソッドの引数にも正規表現が登場しています(`\(\d+\)|-\d+-`)。この正規表現にはこの章ではまだ説明していないメタ文字が含まれています。こちらに関してはこのあとの章で説明します。

▼ 図E　正規表現 0[1-9]\d{0,3}[-(]\d{1,4}[-)]\d{4} を入力した場合



▼ リストC　正規表現とRubyの機能の併用例

```
text = <<-TEXT
名前：伊藤淳一
電話：03(1234)5678
？？：9999-99-9999
？？：03(1234-5678
住所：兵庫県西脇市なんとか町1-2-3
TEXT
numbers = text.scan(/0[1-9]\d{0,3}[-(]\d{1,4}[-)]\d{4}/)
# => ["03(1234)5678", "03(1234-5678"]
numbers.grep(/\(\d+\)|-\d+-/)
# => ["03(1234)5678"]
```

まとめ

本章では正規表現について次のようなことを学びました。

- \dは「半角数字1文字」を表す
- {n,m}は「直前の文字がn文字以上、m文字以下」であることを表す
- {n}は「直前の文字がちょうどn文字」であることを表す
- [AB]は「AまたはBが1文字」であることを表す
- [a-z]と[-az]ではハイフンの意味が異なる
- 正規表現の正確さとシンプルさはトレードオフになることが多い

今回は「電話番号を検索する」というテーマだったので、まだまだバリエーションに乏しいです。実務で思いどおりに文字列を検索するには、もう少したくさん正規表現のルールを覚えなくてはなりません。ただ、初心者の方は本章では正規表現を使いこなすことよりもむしろ、次のようなことを感じ取ってもらえればまずはOKだと思います。

- 今まではテキストエディタで普通の文字列検索しか使ってこなかったけど、正規表現を使えば今まで目視で調べていたような文字列も検索できるじゃないか！
- 今まで文字列処理するプログラムでは1文字ずつループさせたり、特定の文字で分割したり、あれこれ面倒なことをやってきたけど、正規表現を使ったらそんなことせずに一発で処理できるのでは？
- 今まで正規表現を見ても意味不明な呪文にしか見えなかったけど、この記事を読んだら正規表現の見え方が変わってきたぞ!?

こんなふうに感じてもらえれば、このあとの章でどんどん新しい知識を吸収できるはずです！

SD

第2章 カタカナ語の揺れを許容して検索しよう

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

今回のお題

突然ですが、筆者の妻はCoupé Baguette(クープ バゲット)[注1]という小さなパン屋さんを経営しています。いちおう正式なカタカナ表記は「クープ␣バゲット(半角スペース区切り)」なのですが、ネットを見ていると「クープバゲット(区切りなし)」だったり、「クープ・バゲット(中黒区切り)」だったり、「クープ␣バゲット(全角スペース区切り)」だったり、人によってさまざまです[注2]。また、ときどき「バゲット」ではなく「バケット」と濁点なしの「ケ」で書いてる人もいます。「みんなちゃんと“クープ␣バゲット”って書いてよ！！」……と叫んだところで誰も聞いちゃいないので、こちら側で表記の揺れを許容するしかありません。

さて、前振りが長くなりましたが、本章では正規表現を使って、テキストの微妙な違いを許容しながら検索する方法を説明します。使用するのはこんなテキスト(架空のレビューテキスト)です。

サンプルテキスト 2-1

```
クープバゲットのパンは美味しかった。
今日はクープ␣バゲットさんに行きました。
クープ␣␣バゲットのパンは最高。
ジャムおじさんのパン、ジャムが入ってた。
また行きたいです。クープ・バゲット。
クープ・バケットのパン、売り切れだった(><)
```

「クープバゲット」やら「クープ・バケット」といった表記の揺れを許容しつつ、「クープ␣バゲット」について言及していると思われるレビューだけを抽出してみましょう。

まずは普通の文字列検索からスタート

今回もRubularを使っていきます。まず、Your test string欄に上記のテキストを入力してください。続いて、Your regular expression欄にクープ␣バゲット(半角スペース区切り)と入力してください。そうすると2行目にある「クープ␣バゲット」がマッチするはずです(**図1**)。

しかし、これはメタ文字をまったく使わないただの文字列検索です。ここから正規表現を使ってマッチする文字列を増やしていきます。

▼図1 Rubular画面(マッチ部分「クープ␣バゲット」が反転している)



注1) http://coupe-baguette.com/
注2) 本章では半角スペースを␣の記号で、全角スペースを␣の記号で明示的に表現します。

区切り文字の違い・濁点の有無を許容する

第1章では[AB]で「AまたはBのいずれか1文字を表す」というメタ文字を紹介しました。この知識を活用すると、次のように区切り文字を複数指定することができます。

```
クープ[␣␣]・ゴバゲット
```

誌面上では非常にわかりにくいですが、[ ]の中身は半角スペースと全角スペースと中黒の3文字が含まれています。これをRubularに入力してください。そうすると3行目と5行目のテキストにもマッチするはずです。

```
Match result:
クープバゲットのパンは美味しかった。
今日はクープ バゲットさんに行きました。
クープ　バゲットのパンは最高。
ジャムおじさんのパン、ジャムが入ってた。
また行きたいです。クープ・バゲット。
クープ・バケットのパン、売り切れだった（>＜）
```

濁点の有無を許容する

同じく正規表現を使って「バゲット」と「バケット」の両方にもマッチさせましょう。もうおわかりかと思いますが、ここでも[ ]を使います。

```
クープ[␣␣]・ゴバ[ゲケ]ット
```

```
Match result:
クープバゲットのパンは美味しかった。
今日はクープ バゲットさんに行きました。
クープ　バゲットのパンは最高。
ジャムおじさんのパン、ジャムが入ってた。
また行きたいです。クープ・バゲット。
クープ・バケットのパン、売り切れだった（>＜）
```

これで6行目のテキストにもマッチさせることができました。しかし、1行目にある「クープバゲット（区切りなし）」にはマッチしていません。これはどうやってマッチさせればよいでしょうか？

★★「?」は直前の文字が0個、または1個を表すメタ文字

まずは検索したい文字列のパターンを考えてみましょう。今回は次のようなパターンになります。

「クープ、区切り文字（半角スペースまたは全角スペースまたは中黒）が1文字、もしくは区切りなし、バゲ（またはケ）ット」

C コラム　最低限覚えるべきメタ文字の一覧

正規表現を読み書きする場合は、どれがメタ文字でどれがメタ文字でないかを区別することが重要です。**表A**に挙げる文字が正規表現中に出てきたら、何らかの特別な意味を持つと考えて間違いありません。正規表現を読み書きする際に参考にしてください。

ほかにも本特集で説明していないメタ文字として、(?:)や(?=)などがありますが、( )のバリエーションだと考えれば、「何か特別な意味を持っている」と予想できるはずです。

▼表A　最低限覚えるべきメタ文字

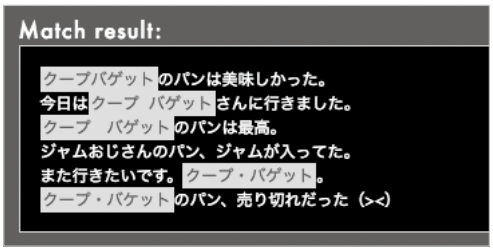
[]	いずれか1文字を表す文字クラスを作る（1章）
[^]	～以外の任意の1文字を表す文字クラスを作る（3章）
-	[]内で使われると文字の範囲を表す（1章）
.	任意の1文字を表す（2章）
()	内部でマッチした文字列をキャプチャもしくはグループ化する（3章、4章）
?	直前の文字やパターンが1回、もしくは0回現れる（2章）
*	直前の文字やパターンが0回以上連続する（3章）
+	直前の文字やパターンが1回以上連続する（3章）
{n,m}	直前の文字やパターンがn回以上、m回以下連続する（1章）
|	OR条件を作る（4章）
^	行頭を表す（4章）
$	行末を表す（4章）
\	メタ文字をエスケープする、\nや\wといった別のメタ文字の一部になる（1章、2章）

新しく登場したのは「区切り文字が1文字、もしくは区切りなし」というパターンです。

今回のように「～が1文字、またはなし」を表現するためには「?」というメタ文字を使います（文字量を指定するので**量指定子**の1つです）。「?」を使うと正規表現は次のようになります。

```
クープ[„„]・]?バ[ゲケ]ット
```

これをRubularに入力すると……1行目の文字にもマッチしました！



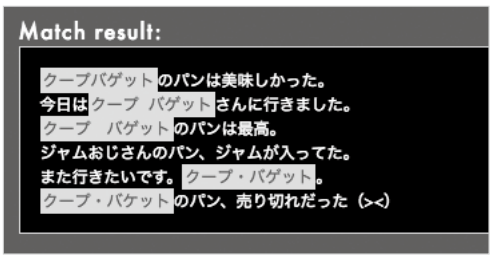
★「.」は任意の1文字を表すメタ文字

さて、この正規表現はこれで完成！……としてもいいのですが、もうひとつ新しい知識を学習しておきましょう。先ほどの正規表現の[„„]・]の部分ですが、「半角スペースまたは全角スペースまたは中黒」と厳密に文字種を指定しなくても、現実的には「任意の1文字」という考えで十分だったりします。

正規表現にはちょうど「任意の1文字」を表す「.」というメタ文字（文字クラス）があります。これを使うと先ほどの正規表現を次のように書き直せます。

```
クープ.?バ[ゲケ]ット
```

Rubular上でこの正規表現を入力しても、実行結果は変わりません。



もちろん、「.」は「任意の1文字」なので「クープ★バゲット」や「クープ@バゲット」にもマッチしてしまいますが、今回のような要件であればそういった文字列にマッチしても大きな問題はないでしょう。

Ⓒ コラム うっかり「.」をそのまま使わないように注意！

正規表現における「.」は「任意の1文字」を表すメタ文字になります……というのは先ほど説明したとおりなのですが、ドット（.）はファイルパスやURLによく登場する文字です。そのため、ついうっかり`sample.pdf`みたいな正規表現を書いてしまうことがあります。これだと“sample.pdf”だけでなく、“sample_pdf”や“samplexpdf”にもマッチしてしまいます（まあ現実的には“sample.pdf”以外の文字列がマッチするケースはまれだと思いますが……）。

こういう問題が起きないよう、ドットをそのままドットとして使う場合は、`sample\.pdf`のように必ずバックスラッシュをつけてメタ文字としての役割を打ち消してください（メタ文字のエスケープ）。

まとめ

本章では正規表現について次のようなことを学びました。

- 「?」は「直前の文字が1個、またはなし」を表す
- 「.」は「任意の1文字」を表す
- メタ文字をエスケープする場合は手前にバックスラッシュ（\）を付ける

次章では新しいメタ文字を学ぶだけでなく、正規表現を使った文字列置換も説明します。SD

第3章 HTMLをCSVに置換しよう

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

今回のお題

正規表現は検索だけでなく置換する場合にも非常に便利です。単純な文字列置換よりも柔軟なルールで文字列を置換することができます。本章では正規表現を使った置換処理を説明します。

HTMLをCSVに変換!?

ある日、あなたは上司からこんなお願いをされました。「このWebページに表示されているselectタグの中身をCSVにして渡してほしい」。

「なんでそんなことを頼むんだろう？」と疑問に思いつつ、上司が教えてくれたURLを開くと、こんなHTMLソースになっていました。

サンプルテキスト 3-1
```
<select name="game_console">
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
```

「selectタグの中身をCSVにする」というのは、どうやら次のようなテキストファイルを作る、ということのようです。

```
wii_u,Wii U
ps4,プレステ4
gb,ゲームボーイ
```

さて、あなただったらどうしますか？　3つぐらいなら手作業でも何とかなるかもしれませんが、実際は100個以上optionが並んでいる、と想定してください。

本格的にやるならHTMLパーサを使って変換プログラムを書くという手もあります。しかし、ある程度パターンが決まっているケースであれば、正規表現を使って置換してしまうほうが速いかもしれません。ちょっとやってみましょう。

対象のテキストを抜き出す手順を考える

というわけでまず、optionタグのvalueと、表示テキスト(“Wii U”や“プレステ4”など)を正規表現を使ってきれいに抜き出しましょう。

valueを抜き出す正規表現を考える

valueは次のようなパターンになっています。

「value=、ダブルクオート、英数字またはアンダースコアが1文字以上、ダブルクオート」

これを正規表現で表すとどうなるでしょうか？

「+」は「直前の文字が1文字以上」を表すメタ文字

「直前の文字が1文字以上」を表す場合は「+」というメタ文字(量指定子)を使います。また、「英数字またはアンダースコア」は[a-z0-9_]のように[ ]と「-」を使って書けます(第1章参照)。

よって、value="[a-z0-9_]+"のように書けばvalueの部分にマッチさせることができます([a-z0-9_]+以外はメタ文字でない、ただの文字です)。

Rubularを使って確認してみましょう。まず次のテキストを入力してください。

サンプルテキスト 3-1(再掲)
```
<select name="game_console">
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
```

続いてvalue="[a-z0-9_]+"という正規表現を入力します。これでvalueの部分がマッチするはずです。

```
Match result:
<select name="game_console">
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
```

表示テキストを抜き出す正規表現を考える

次に表示テキストを抜き出す正規表現を考えてみましょう。難しそうに思えるかもしれませんが、ここでは単純にこう考えることにします。

「>で始まり、何かしらの文字が続き、<で終わる」

厳密に言うと“>”と“<”はタグの一部ですので表示テキストではありませんが、今はとりあえず良しとしてください。

「何かしらの文字が続く」というのは、すでに紹介した「.」と「+」を使えば実現できます。つまり、表示テキストを抜き出す正規表現は次のようになります(“>”と“<”はメタ文字ではない、ただの文字です)。

```
>.+<
```

この正規表現をRubularに入力してみましょう。

```
Match result:
<select name="game_console">
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
```

これで表示テキスト部分もマッチしました。

置換する手順を考える

さて、valueと表示テキストを抜き出す正規表現はわかりましたが、問題はこれからどうやってCSV形式に置換するかです。今回のようなケースであれば、次のように処理するのが良いと思います。

①行全体にマッチする正規表現を作る
②valueと表示テキストの部分を( )で囲んでキャプチャする
③キャプチャを利用して新しい文字列を組み立てる

ややこしそうに感じるかもしれませんが、じっくり順を追って見ていきましょう。

①行全体にマッチする正規表現を作る

まず行全体にマッチする正規表現を作ります。「えっ、行全体ってどうするの？」と心配しなくても大丈夫です。難しく考えずに次のような正規表現を使えばOKです。

```
<option value="[a-z0-9_]+">.+<\/option>
```

[a-z0-9_]+と.+の部分がここまでに見てきた正規表現(メタ文字)です。それから<\/option>に入っているバックスラッシュ(\)はスラッシュ(/)をエスケープするためのエスケープ文字です。Rubyでは/abc/のようにスラッシュを使って正規表現オブジェクトを作るので、正規表現中に出てくるスラッシュはエスケープする必要があります。それ以外はメタ文字ではない、ただの文字列です。

この正規表現をRubularに入力すると、optionタグの各行が全部マッチするはずです。

```
Match result:
<select name="game_console">
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
```

②valueと表示テキストの部分を( )で囲んでキャプチャする

次にvalueと表示テキストをそれぞれ( )で囲みます。つまり次のような正規表現にします。

```
<option value="([a-z0-9_]+)">(.+)<\/option>
```

これをRubularに入力するとちょっとした変化が起き、画面の右下にMatch groupsという

欄が新たに登場するはずです(**図1**)。

1がvalueの値、2が表示テキストになっていますね。正規表現に( )を使うと、その部分がキャプチャ(捕捉)され、連番が付けられるのです。

③キャプチャを利用して新しい文字列を組み立てる

さて、残念ながらRubularには文字列を置換する機能がありません。ここまで来たら次はAtomにバトンを渡します(Atom以外のテキストエディタを使ってもOKです。ただし、置換で使用する記号などが異なる場合があります)。

Atomに先ほどのテキストと正規表現を入力します。Rubularと同様、optionタグの各行の全体がマッチしていることを確認してください(**図2**)。

ちなみに、AtomはRubyようにスラッシュを使った正規表現リテラルを持っているわけではないので、スラッシュをエスケープしなくても大丈夫です(<\/option>ではなく、</option>と書いても良い)。ただ、エスケープしていても今回は問題ないのでRubularで作った正規表現をそのまま使うことにします。

★☆ $1や$2でキャプチャ部分を置換で利用できる

続いて、Atomの Replace in current buffer 欄に次のような文字を入力します(**図3-①**)。

```
$1,$2
```

入力が終わったらReplaceボタンをクリックしてみてください(**図3-②**)。

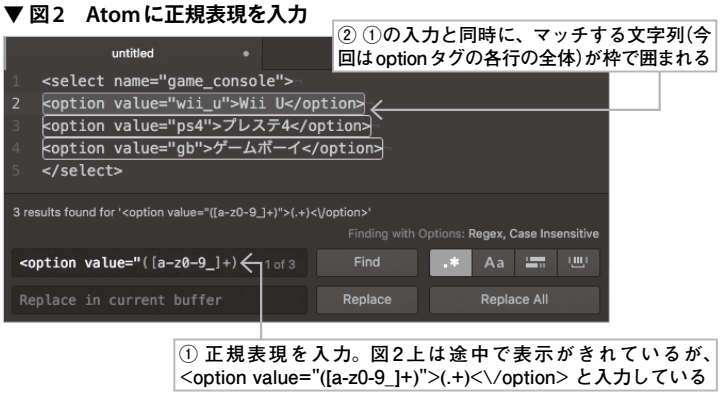
最初のoptionタグが"wii_u,Wii U"に置換されました(**図3-③**)!

なんとなく予想がついているかもしれません

▼図1 キャプチャされた値が表示される



▼図2 Atomに正規表現を入力



C コラム [a-z0-9_]+を\wに置き換える

正規表現には\wというメタ文字があります。これは「英単語を構成する文字」の意味です。Rubyでは「\w＝[a-zA-Z0-9_](半角英数字とアンダースコア1文字)」という仕様になっています。なので、[a-z0-9_]+は\w+と書き換えてもほぼ同じ意味になります(厳密には\wを使うと大文字のアルファベットも含まれる点が異なります)。

\wを使うと正規表現が短くなりますし、\wの意味を知っていれば何をやっているのかも理解しやすくなります。<option value="([a-z0-9_]+)">(.+)<\/option>の代わりに次の正規表現を使っても、検索結果に変わりがないことを、念のためにRubularを使って確かめてみてください。

```
<option value="(\w+)">(.+)<\/option>
```

が、$1と$2はそれぞれキャプチャされた1番目の文字列と2番目の文字列を表しています。つまり、$1,$2はマッチした文字列を「value,表示テキスト」の文字列に置換していることになるわけです。

このあと同じようにReplaceボタンを2回押すか、Replace Allボタンを1回押すとすべての文字列を置換できます(**図4**)。はい、ちゃんと全部置換されましたね！

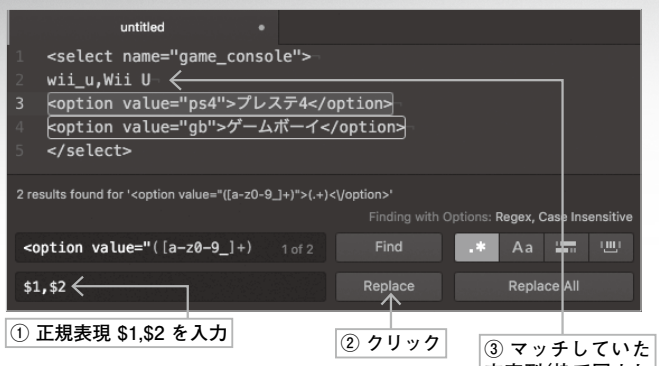
Rubyで置換してみる

Atomではなく、Rubyスクリプト内で文字列置換を実行してみましょう。**リスト1**がRubyを使った場合のサンプルコードです。

Atomとは異なり、キャプチャされた文字列を$1や$2ではなく、\1や\2で参照している点に注意してください。

このように、正規表現を使った文字列置換はテキストエディタでもプログラミング言語の中でも使うことができます。

▼図3　正規表現でマッチした部分を置換



▼図4　すべての文字列を置換

```
<select name="game_console">
wii_u,Wii U
ps4,プレステ4
gb,ゲームボーイ
</select>
```

▼リスト1　Rubyにおける正規表現を使った置換の例

```
html = <<-HTML
<select name="game_console">
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
HTML

replaced = html.gsub(/<option value="([a-z0-9_]+)">(.+)<\/option>/,
'\1,\2')

puts replaced
# <select name="game_console">
# wii_u,Wii U
# ps4,プレステ4
# gb,ゲームボーイ
# </select>
```

C コラム　「*」で空のvalueや空の表示テキストに対応する

先ほどのselectタグの中に、空のoptionが含まれていた場合はどうなるでしょうか？　たとえば、こんなHTMLだった場合です(2行目に空のoption)を追加)。

```
サンプルテキスト 3-2
<select name="game_console">
<option value=""></option>
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
```

<option value="(\w+)">(.+)<\/option>のままだと**図A**のように空のoptionが選択されません。

▼図A　正規表現 <option value="(\w+)">(.+)<\/option> を入力した場合

```
Match result:
<select name="game_console">
<option value=""></option>
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
```

ですが、これは先ほどの正規表現をちょっと修正すればうまくいきます。

たとえば、表示テキストを抽出する正規表現は.+でした。この正規表現は「任意の文字が1文字以上」の意味ですが、空の場合も含めるのであればこれを「任意の文字が**0文字以上**」に変えれば良いのです。

「直前の文字が0文字以上」を表す場合は「*」というメタ文字(量指定子)を使います。よって、.+を.*に変えればOKです。同様に、\w+も\w*に変えます。

```
<option value="(\w*)">(.*)<\/option>
```

▼図B　正規表現 <option value="(\w*)">(.*)<\/option> を入力した場合

Match result:

```
<select name="game_console">
<option value=""></option>
<option value="wii_u">Wii U</option>
<option value="ps4">プレステ4</option>
<option value="gb">ゲームボーイ</option>
</select>
```

▼図C　Match groups欄

Match groups:

```
Match 1
1.
2.
Match 2
1. wii_u
2. Wii U
Match 3
1. ps4
2. プレステ4
Match 4
1. gb
2. ゲームボーイ
```

Rubularで確認してみましょう(**図B**)。はい、各optionがちゃんと選択状態になりました。Match groups欄を見ると空のoptionも正しく選択されていますね(**図C**)。

Ⓒ コラム　「*」と「+」は「貪欲」なので使い方に注意する

ところで、先のコラムで紹介した「*」と「+」は扱い方を間違えると**ドツボにハマります**。ちょっと難しいかもしれませんが、今から説明する内容は重要なポイントなのでしっかり理解してください。

先ほど使った例ではoptionは毎回改行されていました。では、改行せずにoptionを並べるとどうなるでしょうか？　試しに**リストA**のようなテキストを使ってみましょう。

これをRubularに入力し、先ほどと同じ正規表現<option value="(\w*)">(.*)<\/option>を使って検索してみます。するとMatch groups欄の結果は**図D**のようになるはずです。なにかおかしいですね。なぜか表示テキストを正しく取得できていません。

問題は正規表現の>(.*)<の部分にあります。このパターンの意味を日本語で表現すると次のようになります。

「>で始まり、任意の文字が0個以上連続し(.*)、<で終わる」

これをふまえて、元のHTMLを見てください(**リストB**)。

「>(★)で始まり、任意の文字が0個以上連続し(.*)、<(☆)で終わる」

……どうでしょう？　パターンとしては間違ってないですよね？

.*は「任意の文字が0個以上」の意味なので、途中に出てくる「<」も「任意の文字」として扱われてしまうのです。これは.+に変えても同じ結果になります。

「+」や「*」はしばしば「**貪欲なマッチ**」を試みる量

▼図D　正規表現 <option value="(\w*)">(.*)<\/option> を入力した場合

Match groups:

```
1. ps4
2. プレステ4</option><option value="gb">ゲームボーイ
```

▼リストA　サンプルテキスト3-3

```
<option value="ps4">プレステ4</option><option value="gb">ゲームボーイ</option>
```

▼リストB　サンプルテキストのどこにマッチしているのか

```
<option value="ps4">プレステ4</option><option value="gb">ゲームボーイ</option>
                    ★                 .*                  ☆
```

指定子」と呼ばれます。パターンの結果として矛盾がなければ、(貪欲に)最長のマッチを結果として返すのです。

しかし、今回の場合、これは期待する結果ではありません。正規表現を工夫して、正しくvalueと表示テキストを抽出できるようにしましょう。

■解決策1　否定の文字クラス[^]を使う

1つの考え方は「任意の1文字」というゆるい条件を、少し厳しい条件に変えることです。具体的には「<以外の任意の文字」という条件に変えてあげるとうまくいきます。

「A以外の任意の文字」を表す場合は[^A]と書きます。[ ]の最初に「^」が入ると否定の意味に変わるのです。[^AB]であれば「AでもなくBでもない任意の1文字」の意味になります。よって、「<以外の任意の文字」を表す場合は[^<]と書きます。

では正規表現を次のように書き換えてみましょう。

```
<option value="(\w*)">([^<]*)<\/option>
```

この正規表現をRubularに入力してみると……無事にvalueと表示テキストを抽出できました！(**図E**)「<以外の任意の文字が0文字以上」というパターンに変わったので、正規表現は途中の<を突き破ってマッチすることはできなくなったわけです。

■解決策2　*?で「控えめに」マッチさせる

もう1つの方法は「*」や「+」の「貪欲さ」をなくすことです。>(.*)<の意味をもう少し厳密に書くと次のようになります。

「>で始まり、任意の文字が0個以上連続し(.*)、**最後に見つかった**<で終わる」

今回のケースはこれを次のように変えればうまくいくはずです。

「>で始まり、任意の文字が0個以上連続し(.*)、**最初に見つかった**<で終わる」

こうするためには>(.*?)<のように「*」の後ろに「?」を付けます。*?や+?にすると最長ではなく、最短のマッチを結果として返すようになるのです(これを**最小量指定子**と呼びます)。

というわけで次のような正規表現を作ってRubularに入力してみましょう。

```
<option value="(\w*)">(.*?)<\/option>
```

こちらも**図E**と同じ正しい結果になります。「*」が貪欲さを失ったので、最初に見つかった<でマッチが止まるようになっています。

ちなみに、「+」や「*」が「貪欲なマッチ」と呼ばれるのに対して、*?や+?は「**控えめなマッチ**」と呼ばれることがあります。

▼図E　正しくマッチした例



まとめ

本章では正規表現について次のようなことを学びました。

- 「+」は「直前の文字が1個以上」を表す
- 「*」は「直前の文字が0個以上」を表す
- ( )はマッチする部分をキャプチャ(捕捉)する
- キャプチャした部分は置換するときに$1や\1で参照できる
- \wは「英単語を構成する文字(半角英数字とアンダースコア)」を表す
- 正規表現中の特別な文字は「\」でエスケープする
- 「*」と「+」は「貪欲」で最長マッチを返すため、使い方を誤ると思いがけない結果が返る
- *?や+?にすると、最短マッチを返す

次の章では普通の文字列検索では扱いづらい空白文字を正規表現で検索・置換する方法を説明します。SD

第4章 スペースや改行などの空白文字を検索／置換しよう

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

お題① 無駄なスペースやタブ文字の入った空白行を見つける

空白文字というのは半角スペースやタブ文字、改行文字といった、通常画面には表示されない文字のことです。半角スペースはともかく、タブ文字や改行文字はテキストボックスに入力するのが難しいので検索したり置換したりするのが困難です。しかし、正規表現を使うとこういった文字も自由自在に操ることができます。

また、この章では空白文字に加えて、行頭や行末といったテキストの「位置」を表す正規表現についても説明します。

さて、今回は次のようなRubyスクリプトを用意しました。ただし、Rubyスクリプトといってもこれは実行するためのものではなく、あくまで正規表現の使い方を確認するための文字列として使います。

サンプルテキスト 4-1

```
1 def␣hello(name)
2 ␣␣puts␣"Hello,␣#{name}!"
3 end
4
5 hello('Alice')
6 ␣␣␣␣␣␣␣␣␣␣ ←半角スペースが10個含まれている注1
7 hello('Bob')
8 »···┤ ←タブ文字が1つ含まれている注1
9 hello('Carol')
```

一見、何も問題なさそうに見えますが、実は空白行の一部にスペースやタブ文字が含まれています。空白行にスペースやタブ文字が含まれていても意味がないので、おかしな行をあぶり出してやりましょう。

まず、上のテキストをRubularに入力してください……といっても、誌面上の␣や»···┤の記号を見て空白文字を手入力するのも手間でしょうから、本誌サポートページ注2よりサンプルテキストをダウンロードしてコピー&ペーストしてください。続いて、Your regular expression欄に␣+(半角スペースとプラス)と入力してください(ここまでの記事を読んだ方はわかると思いますが、これは「スペースが1文字以上続く」という意味の正規表現です)。

するとRubularは次のようにマッチした文字列を表示します。

Match result:

```
def hello(name)
  puts "Hello, #{name}!"
end

hello('Alice')

hello('Bob')

hello('Carol')
```

どうやらhello('Alice')とhello('Bob')の間の空白行にスペースが含まれるようですね。しかし、1行目や2行目にある正常なスペースまで検出してしまっています。

「^」と「$」は「行頭」と「行末」を表すメタ文字

そこで先ほどの正規表現を^␣+に変えてみましょう。すると次のようにマッチする部分がちょっと減るはずです。

Match result:

```
def hello(name)
  puts "Hello, #{name}!"
end

hello('Alice')

hello('Bob')

hello('Carol')
```

注1) 本章では半角スペースを␣の記号で、タブ文字を»···┤の記号で明示的に表現します。
注2) http://gihyo.jp/magazine/SD/archive/2016/201607/support

ここで使った「^」は「行頭」を表すメタ文字です。文字そのものではなく、マッチした「位置」を示すので、こういったメタ文字をとくに**アンカー**と呼びます。つまり、^␣+は「行頭からスペースが1文字以上続く」という意味になります。

ただ、このままだと2行目の␣␣puts␣"Hello,␣#{name}!"の行頭スペースもマッチしています。この行は除外すべきですね。なので、さらにメタ文字を加えて^␣+$という正規表現を使います。

「$」は「^」の反対で、「行末」を意味するメタ文字(アンカー)です。つまり、^␣+$は「行頭から行末までスペースが1文字以上続く」という意味になります。

では、Rubularに^␣+$と入力してみましょう。

```
Match result:
def hello(name)
  puts "Hello, #{name}!"
end

hello('Alice')

hello('Bob')

hello('Carol')
```

これで空白行に含まれるスペースだけを検出できました！

\tはタブ文字を表すメタ文字

さて、スペースが含まれる空白行は見つかりましたが、タブ文字が含まれる空白行はまだ見つかっていません。こちらも正規表現を使って検出してみましょう。

タブ文字は\tというメタ文字(文字クラス)を使って表現できます。そこでRubularに^[␣\t]+$という正規表現を入力してみましょう。おわかりだと思いますが、これは「行頭から行末までスペース、またはタブ文字が1つ以上連続する」という意味になります。

```
Match result:
def hello(name)
  puts "Hello, #{name}!"
end

hello('Alice')

hello('Bob')

hello('Carol')
```

おっ、hello('Bob')とhello('Carol')の間にタブ文字が隠れていたようですね！　これでスペースやタブの入った空白行を見つけ出すことができました。

お題②　ログから特定の文字列を含む行を削除する

さて、次のサンプルに移りましょう。今度はログファイルを扱います。**リスト1**はHeroku[注3]にデプロイしたRailsのログです(サンプル用にちょっと加工してあります)。

このアプリケーションはブラウザからWebページにアクセスされる場合と、定期的に起動するスケジューラーが動作する場合の2パターンあります。Webページへのアクセスは"app/web"や"heroku/router"と表示され(1行目と5〜7行目)、スケジューラーの実行ログは"heroku/api"や"heroku/scheduler"のように表示されます(2〜4行目)。

今はWebページへのアクセスだけを確認したいのですが、スケジューラーの実行ログが混在していると非常に読みづらいです。そこで、正規表現を使ってスケジューラーの実行ログを削除することにします。

削除したい行を選択する正規表現を別々に作る

まず、**リスト1**のテキストをRubularに入力

▼リスト1　デプロイしたRailsのログ

サンプルテキスト 4-2

```
1 Feb 14 07:33:02 app/web.1:  Completed 302 Found ...
2 Feb 14 07:36:46 heroku/api:  Starting process ...
3 Feb 14 07:36:50 heroku/scheduler.7625:  Starting ...
4 Feb 14 07:36:50 heroku/scheduler.7625:  State ...
5 Feb 14 07:36:54 heroku/router:  at=info method=...
6 Feb 14 07:36:54 app/web.1:  Started HEAD "/" ...
7 Feb 14 07:36:54 app/web.1:  Completed 200 ...
```

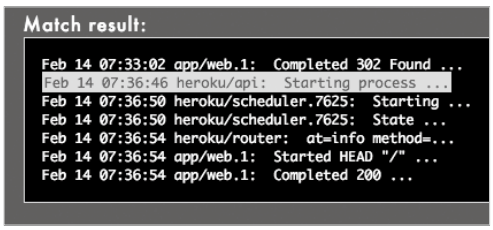
注3) https://www.heroku.com/

してください(このテキストも手入力するのはたいへんだと思うので、サンプルテキストをダウンロードしてコピー&ペーストしてください)。それから次のような正規表現を入力します。

```
^.+heroku\/api.+$
```

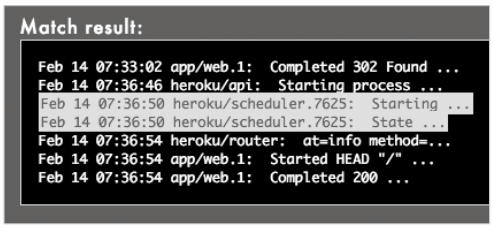
この正規表現の意味は「行頭からの何らかの文字が1文字以上続き(^.+)、"heroku/api"が現れ(heroku\/api)、その後行末まで何らかの文字が1文字以上続く(.+$)」という意味です。なお、\/のバックスラッシュ(\)はスラッシュ(/)をエスケープするためのエスケープ文字です(第3章を参照)。

この正規表現をRubularに入力すると"heroku/api"が含まれる行を選択できました。



同じ要領で次のような正規表現を作れば、"heroku/scheduler"が含まれる行を選択できます。

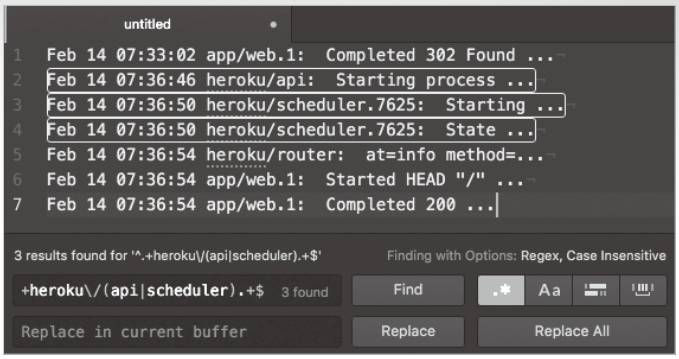
```
^.+heroku\/scheduler.+$
```



「|」は複数の条件をOR条件で連結するメタ文字

しかし、これらの行は別々に選択するのではなく、できれば"heroku/api"の行と"heroku/scheduler"の行をまとめて選択したいですね。

▼図1　Atomで表示してみる

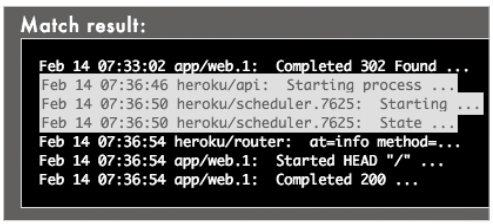


こういう場合は「|」というメタ文字を使います。ABC|DEFのように書くと、「文字列ABC、または文字列DEF」というOR条件の意味になります。実際にはOR条件の範囲を指定するため、(ABC|DEF)のようにグループ化の()と一緒に使われることが多いです(()はキャプチャのためだけでなく、パターンをグループ化する場合にも使われます)。

というわけで、「"heroku/api"または"heroku/scheduler"が含まれる行」を選択したい場合は次のような正規表現を使います。

```
^.+heroku\/(api|scheduler).+$
```

この正規表現をRubularに入力すると、期待したとおりに不要な行を検索できました!



置換を使って不要な行を削除する

さて、正規表現の動きを確認できたら置換処理を実行するためにAtomにバトンを渡しましょう。Rubularで使ったテキストと正規表現をAtomにコピー&ペーストしてみてください(**図1**)。

置換後の文字列(Replace in current buffer

欄）は何も入力しない状態でReplace Allボタンをクリックすれば、不要な行が削除されるはずです（**図2**）。やってみましょう！

うーん……これでいいと言えばいいかもしれませんが、空白行が残っちゃいましたね。できれば空白行を残さずに全部行を詰めてほしいところです。大丈夫です。正規表現をちょっと変えれば、ちゃんと空白行を残さずに削除できます。

改行文字（\n）も置換対象にすれば空白行をなくせる

行を詰めるということはすなわち「改行文字を削除すること」です。$は「行末」を表すメタ文字ですが、このままだと改行文字は含まれません。$の代わりに\nを指定してやると、検索結果に改行文字も含まれるようになります。

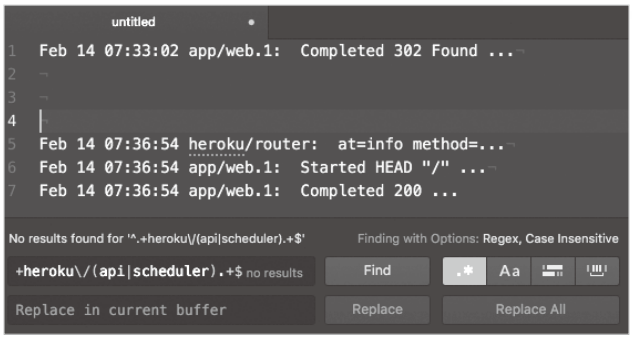
よって、先ほどの正規表現を次のように変更してみてください。

```
^.+heroku\/(api|scheduler).+\n
```

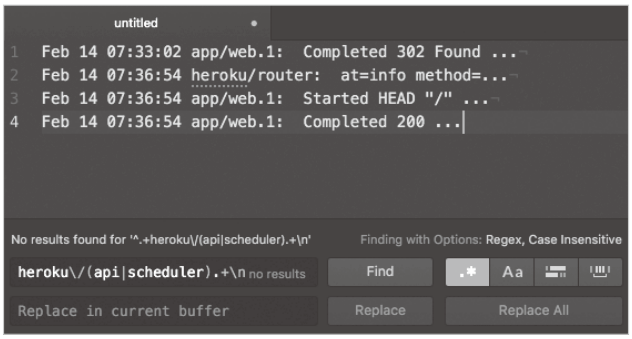
こうして置換を実行すると、空白行を残さずに削除できます。**図3**はAtomでの実行結果です。

なお、Windows環境の場合は改行コードがCRLF（復帰文字+改行文字）になるため、末尾の\nを\r\nに置き換えてください。

▼図2　行が削除された!



▼図3　空白行がすべてなくなった



まとめ

本章では正規表現について次のようなことを学びました。

- 「^」は行頭を表す
- 「$」は行末を表す
- \tはタブ文字を表す
- ABC|DEFは「文字列ABCまたは文字列DEF」のOR条件を表す
- \nは改行文字を表す
- \rは復帰文字を表す
- ()はキャプチャのためだけでなく、パターンをグループ化する場合にも使われる

空白文字の検索や置換の便利さがピンとこない場合は、正規表現なしで本章で説明した検索や置換ができるか試してみてください。おそらく無理なのではないでしょうか。

さて、最後の第5章では正規表現のパフォーマンスについて説明していきます。SD

第5章 正規表現のパフォーマンスを意識しよう

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

正規表現も書き方を誤ると遅くなる!?

さて、ここまではずっと「正規表現はすごいよ！便利だよ！」というスタンスで正規表現の便利機能を紹介してきました。しかし、残念ながら正規表現も無敵のツールではありません。実は書き方を間違えると、とんでもなく遅い正規表現を作ってしまうことがあります。たとえば……と、例を見せる前に注意事項です。

今から紹介する正規表現は必ずローカル環境で実行してください！！

Rubularのようなオンラインツールで実行すると、サーバの負荷を上げてしまいかねません！いいですか？　必ず約束を守ってくださいね。

とんでもなく遅い正規表現の例

はい、というわけで紹介します。こんなテキストと正規表現がとんでもなく遅くなるケースの一例です。

・テキスト：_a____
・正規表現：(_+|\w+)*a

あくまでパフォーマンスを調べるための正規表現なので、深い意味はありません。あえて(_+|\w+)*aの意味を説明するなら「アンダーバーが1つ以上、または英単語の構成文字が1つ以上続く部分が0個以上連続し、その後にaが現れる」という意味になります。さて、これをAtom上で実行するとどうなるでしょうか？　実際にみなさんのマシン上で試してみてください(**図1**)。

え？　「楽勝で動いた」って？　いいでしょう。じゃあ今度は_a_____みたいに末尾のアンダースコアを伸ばしてから「Find」ボタンをクリックしてください(**図2**くれぐれもRubularで実行しちゃダメですよ！！)。

まだ大丈夫？　じゃあもっと伸ばしてみたらどうなりますか？　たとえばこれぐらい！(**図3**)

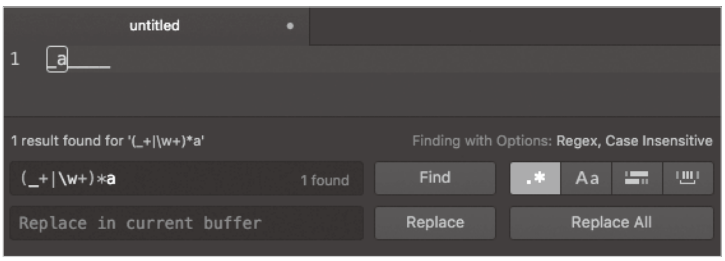
_a_____

このレベルになるとAtomがしばらく固まってしまうのではないでしょうか？　この正規表現はAtomだけでなく、Rubyで実行してもすごく遅くなるはずです。

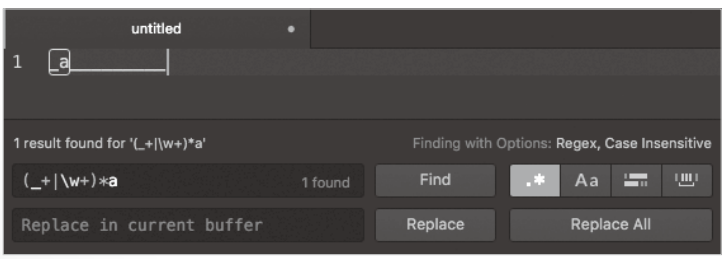
「+」や「*」を入れ子にしたり.*を多用すると危険

この正規表現が遅くなる理由は**バックトラッ**

▼図1　まずは小手調べ

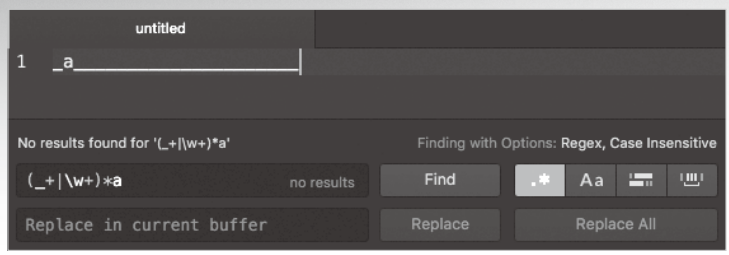


▼図2　少し時間がかかるかも



クと呼ばれる正規表現の動作原理が深く関連しています。ですが、この話はなかなかややこしく、丁寧に説明すると誌面が足りないため、ここでは割愛します。詳しく知りたい方は技術書やネットの情報を参照してください。

▼図3　止まったように見えますが、実行中です



「こう書くと遅くなる」「こう書けば絶対安全」というルールは簡単に示せませんが、少なくとも先ほど示した(_+|\w+)*aのように、「+」や「*」が()の中にも外にも出てくる正規表現は危険です。こういう正規表現は内部的な組み合わせの数が爆発的に増え、とんでもなく遅くなることが多いです。また、.*や.+のような「不特定多数の文字」がマッチする正規表現をなるべく避け、文字種を絞り込んだり、{n,m}の量指定子を使って繰り返しの回数を制限したりするとパフォーマンスの悪化を防げる場合があります。

ツールを使ってパフォーマンスを判定する

もし「この正規表現、ちょっと怪しいな」と思ったら、オンラインツールでパフォーマンスを確認するといいかもしれません。Online regex tester and debugger[注1]というオンラインツールを使うと、パフォーマンスの善し悪しをある程度判定できます。

基本的な使い方はRubularと同じです。正規表現とテキストを入力すると、その正規表現がテキストにマッチするかどうか判定してくれます。



このツールはそれだけでなく、フォームの右

C コラム　2種類の正規表現エンジン――NFAとDFA

正規表現エンジンの実装方式には大きく分けてNFA（Nondeterministic Finite Automaton、非決定性有限オートマトン）とDFA（Deterministic Finit Automaton、決定性有限オートマトン）という2つの種類があります。

RubyやAtomをはじめ、Javaや.NET、PHP、Perlなど、大半の正規表現エンジンはNFAを採用しています。一方、DFAを採用している実行環境にはawkやegrep、MySQLなどがあります。

NFAは柔軟で複雑な処理が可能になる利点がある一方、正規表現の書き方によっては速度が遅くなる、という欠点があります。DFAはその逆で安定して高速に動作しますが、NFAに比べると使える機能が少なくなります（DFAでは()でキャプチャしたり、*?を使ったりできません）。

本章で説明した「とんでもなく遅い正規表現」は正規表現エンジンがNFAの場合にのみ当てはまります（まさにNFAの欠点に該当する部分です）。正規表現のパフォーマンスには注意が必要ですが、必ずしもすべての実行環境で同じ問題が発生するわけではない、ということを知っておいてください。

注1）https://regex101.com/

上に「マッチするまで(またはマッチに失敗するまで)のステップ数」が表示されます(ステップ数は正規表現エンジンの内部的な処理ステップの数です)。たとえば次の例は、遅い正規表現として最初に示した`_a____`と`(_+|\w+)*a`の実行結果です。



右上に"491 STEPS"という表示が見えますね。この場合、マッチするまでに491ステップかかったようです。

ステップ数は入力するテキストによっても変わります。たとえば正規表現はそのままにして、`_a____________________`のようなテキストを入力してみましょう(このツールであれば、遅い正規表現でも途中で中断されるので大丈夫です)。



ご覧のとおり、赤いボックスで"ERROR"と表示されました。ERRORになったのはステップ数があまりにも多くなり過ぎたせいです。つまり、これはパフォーマンスの悪い正規表現だということがわかります。

こんな感じで正規表現やテキストをいろいろ変えていくと、パフォーマンスの善し悪しを判別できます。

C コラム 「|」や「*」にはメタ文字を組み合わせることもできる

本章の説明で使った`(_+|\w+)*a`という正規表現は、よく見るとこれまで説明してこなかったテクニックが登場しています。

1つは、「|」を使って正規表現をOR条件で連結している点です。「|」を使うと`ABC|DEF`のような単純な文字列だけでなく、`\d{4}|[A-Z]{6}`のように正規表現同士をOR条件で連結できます。

もう1つは`()`と「*」を組み合わせて使っている点です。「*」は`A*`のように文字の直後につけるだけでなく、グループ化の`()`の後ろにつけて「()の中のパターンが0回以上連続する」という意味にできます。`()`の中には単純な文字列だけでなく、`(\d{2}-\d{4})*`のようにメタ文字を含めることもできます。また、「*」だけでなく、「+」や「?」、`{n,m}`のようなほかの量指定子も同じように`()`と組み合わせて使うことができます。

このように、正規表現では正規表現同士を組み合わせて、より複雑な検索条件を構築できます。

C コラム 正規表現をもっと使いこなすための参考文献

本記事でカバーしきれなかった正規表現の機能や、バックトラック、NFA/DFAといった動作原理の詳しい内容を知りたい方は『詳説 正規表現[注A]』(オライリー・ジャパン)を読んでみることをお勧めします。この書籍は正規表現の基礎の基礎から高度な内容まで幅広くカバーしており、初心者にとっても上級者にとっても非常に役立つ一冊です。

実際、筆者も初心者のころにお世話になりましたし、本記事を執筆する際にも再度読み直して参考にしています。

注A) https://www.oreilly.co.jp/books/9784873113593/

まとめ

本章では次のようなことを学びました。

- 条件によっては正規表現がとんでもなく遅くなる場合がある
- 「+」や「*」を入れ子にしたり、.*を多用したりするのはなるべく避ける
- 「不特定多数の文字」がマッチしにくい正規表現のほうがパフォーマンス面で有利
- パフォーマンスの善し悪しを判定するのにツールを使うのも良い
- 「|」は正規表現同士をOR条件で連結することもできる
- 「*」や「+」、「?」、{n,m}のような量指定子は()と組み合わせて使うこともできる

さて、これで本特集はおしまいです。いくつかカバーできていないトピックもありますが、ここまでの内容が理解できれば「正規表現の基本」は十分マスターしていると言えるでしょう。\d{4}-\d{2}-\d{4}が「恐ろしい呪文」ではなく、「あ、これはハイフンで区切られた4桁、2桁、4桁の数字のことだね」とわかれば大丈夫です！

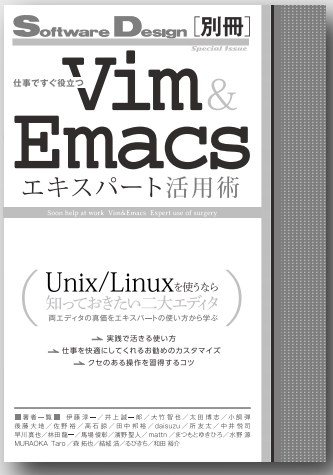
あとは実務で積極的に正規表現を使っていけば、しだいにすらすらと正規表現が書けるようになるはずです。本記事でカバーできていないトピックについては書籍やネットで知識を補強すれば、さらに正規表現力がアップします。

「正規表現は絶対にマスターしておけ！」

本特集がきっかけになって、読者のみなさんにもこのセリフを使うときがくれば幸いです。

SD

Software Design［別冊］ 技術評論社



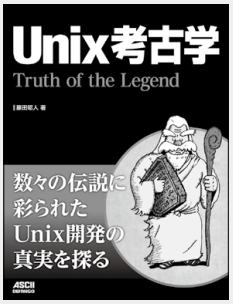
Software Design編集部 編
B5判／200ページ／
綴じ込み付録つき
定価（本体2,480円+税）
ISBN 978-4-7741-8007-6

大好評発売中！

仕事ですぐ役立つ Vim & Emacs エキスパート活用術

Unix/Linuxの使い始めのころ,慣れないコマンドライン上で設定ファイルを編集する際に使うテキストエディタがVim（Vi）あるいはEmacsでしょう。
本書はUnix/Linux初学者や、使えるけれど仕事でなかなか活かし切れていないVim/Emacsユーザを対象に,使い方マニュアルとは違う、仕事で実用的に使えるテクニックを集めました。
実務でそれぞれのエディタを長年愛用しているエキスパートユーザならではの知恵がつまっているので、気になったものから試してみれば、二大エディタの魅力が感じられること請け合いです。
VimとEmacsを仕事で積極的に使っていきましょう!

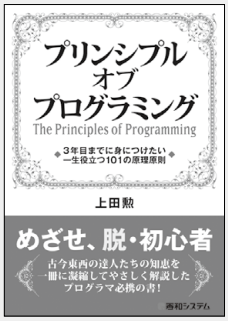
こんな方におすすめ
- Unix/Linux初学者（Vim/Emacs未経験者）
- VimとEmacsどちらを使ったら良いか悩んでいる方
- 使っているけれど仕事でなかなか活かし切れていないと感じているVim/Emacsユーザ



Unix考古学

藤田 昭人 著
B5変型判／272ページ
2,600円＋税
アスキードワンゴ
ISBN＝978-4-04-863050-5

本書はソフトバンククリエイティブ（当時）から発行されていた月刊誌『UNIX USER』での連載記事がベースになっている。当時愛読されていた方には待望の1冊ではないだろうか。日本語の資料としてたいへん希少価値が高く、Unixの祖、Ken ThompsonとDennis RitchieがかかわっていたMulticsからはじまり、PDP-7 Unix（1969年）から4BSD（1993年〜）まで解説。その当時の米国の情勢や関係した企業や団体の思惑などもあわせて考察されていてとても興味深い。本書を読めば、先駆者への尊敬とあこがれと、そして感謝の念があらためてわいてくるだろう。Unixにかかわるエンジニアは先人のパワーを連綿と引き継いでいる（人気コミックの“個性：One For All”か）。その思いを新たにして、良い仕事に向かおう。



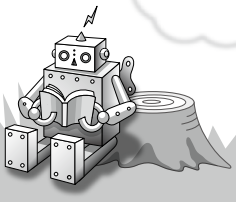
プリンシプル オブ プログラミング

上田 勲 著
A5判／304ページ
2,200円＋税
秀和システム
ISBN＝978-4-7980-4614-3

プログラミングの世界で有名な101の「プリンシプル（原理・原則）」を7つのカテゴリに分けて紹介し、そのひとつひとつについて、どういう意味なのか、どうして必要なのか、どうすれば実現できるのかを説明している。

［前提］の章の「プログラミングに銀の弾丸はない」、［原則］の章の「DRY（Don't repeat yourself）」などは、一度は聞いたことがあるがその背景や深い意味は知らないという方が多いのではないだろうか。もっと例を挙げると、「［思想］UNIX思想」「［視点］結合度」「［習慣］怠慢・短気・傲慢であれ」「［手法］ドッグフーディング」「［法則］セカンドシステム症候群」などなど。プログラミング言語の種類に囚われない、また、陳腐化しない知恵を手に入れられるだろう。

SD BOOK REVIEW





Amazon Web Servicesクラウドネイティブ・アプリケーション開発技法

佐々木 拓郎 ほか 著／佐々木 拓郎 監修
B5変型判／632ページ
3,680円＋税
SBクリエイティブ
ISBN＝978-4-7973-8631-8

クラウドネイティブとは、クラウドサービスを利用することを前提としたアーキテクチャ・設計のこと。既存のシステムにいかにクラウドを取り入れるか、でなはく、使用するクラウドに合わせてシステムを設計するのである。本書の前半ではおもにJavaScriptやモバイのアプリのフロントエンドから直接利用しやすいAWSのサービス群（S3、API Gateway、SNS、DynamoDB、Lambda、Cognito、Machine Learning、Kinesis、SQS、IoT、Mobile Hub）の導入と設定の方法を解説している。

後半では、それらサービスの使用を前提としたアプリの開発（Apple Watchを使った健康情報収集アプリ、iBeaconを使った勤怠管理アプリなど）について解説している。



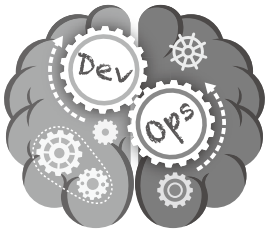
基礎からのWebアプリケーション開発入門

前橋 和弥 著
B5変型判／304ページ
2,680円＋税
技術評論社
ISBN＝978-4-7741-8188-2

本書は、Webサーバの自作を通じてWebアプリ開発の基礎を学ぼうという本。最初に作る「クライアントとサーバでテキストデータをやりとりするプログラム」では、TCP/IPで通信する方法を知る。次に、その通信プログラムを発展させてブラウザが要求したHTMLを返すWebサーバを作る。これはHTTPに従った手順で通信する必要があるため、その仕様を理解して実装する。さらに、Webアプリを作る際には、HTMLを動的に生成するしくみ（サーブレット）や、Cookieによるセッションの実現方法を学ぶ。最初は「学習のためにWebサーバを自作するとは大げさな」という印象でも、読み進めるうちに「必要な知識を段階的に無理なく学ぶには良い方法だ」と納得できてくる。題材が難し過ぎず、すぐに試せる例が多いのも技術者心をくすぐられそうだ。

新連載 DevOps脳を作る基礎トレーニング

アプリエンジニアのための［インフラ］入門



Author 出川 幾夫（でがわ いくお） レバレジーズ株式会社 teratail開発チーム Twitter @ikuwow

第1回 データベース入門

「現場でDevOpsを実現させるには、まずアプリエンジニアがインフラを知る必要がある」という前提に立ち、アプリの視点からインフラを広く学んでいく新連載。第1回ではデータベースを「作る」「使う」際にアプリエンジニアが気を付けること、知識として持っておくべきことをまとめます。

連載のはじめに

近年では、Amazon Web ServicesやMicrosoft Azure、Google Cloud Platformなどの大規模なクラウドを利用することは珍しいことではなく、深くインフラのしくみを知らなくても、アプリケーションやツールの動作検証からデプロイまでできるような環境が整っています。

しかし言うまでもなく、Webアプリケーションはサーバやミドルウェアなどの上に成り立っているため、その動作のしくみや限界などについては知っておく必要があります。

DevOpsという言葉はすでによく聞かれるようになり、知らないでは済まされない時代になっているというのが現状です。個別の技術や製品に関してすべてを細かく知る必要はありませんが、使ううえで理解しておくべき基礎的な部分はたくさんあります。

逆にインフラの知識を持っていると、インフラ担当者や運用担当者との連携を積極的にできるようになり、開発しやすく、パフォーマンスの高いアプリケーションを実現できます。

この連載では「インフラがよくわからない！」というWebやモバイルのアプリケーションエンジニアの人に向けて、全6回に渡ってアプリケーションのプログラミング以外にどういう分野や技術があるのか、何を学ぶ必要があるのか、またアプリケーションを考えるうえで大事なポイントについて紹介していきます。

今回は、アプリケーションともっとも密接に関わり、その土台とも言えるデータベースについて、アプリケーションから見たときの役割からまとめていきたいと思います。

データベースの役割

そもそもシステムやサービス、アプリケーションにおけるデータベースの役割とは何でしょうか。「データを永続的に保持しておくこと」というのが1つの答えですが、単にそれだけではありません。

データの整合性を保つことが最優先

Webアプリケーションでは、必ずと言っていいほど「リレーショナルデータベース（RDB）」と呼ばれる、テーブルとそれら同士の関係の形でデータを保持するデータベースを使います。Oracle Database や MySQL、PostgreSQL などもRDBです。これらはデータベース言語であるSQLを用いてデータを入出力したり、データ構造を定義したりします。

RDBはその構造上、理論的に整合性を保ちながら最適な状態を保つ構造を作れます。基本的に、不整合がなく常に正しいデータを持ち続けるこ

とを保証できるのがRDBの強みになっています。

とくに正規化理論というものが数学的にはっきりと定義される形で存在するため、機械的に正規化を行うだけでも、ある程度問題のない設計ができます。

また、トランザクションもRDBの持つ整合性を保つしくみの1つです。トランザクションはACID特性（Atomicity：原子性、Consistency：一貫性、Isolation：独立性、Durability：耐久性）を持ち、処理が中断してもロールバック・ロールフォワードなどによってデータの整合性を保ち続けられます。

もちろん、パフォーマンスの向上のために正規化を弱めたり、導出項目をあえてテーブルに保持することもありますが、それは正しいデータを正しく保持したうえでやることです。またRDBではなく、MongoDB、RedisなどのNoSQLデータベースを補助的に使うほうが良い場合も多くあります。

筆者の経験上、データの整合性が守られずアプリケーション側で強引に整合性を保とうとしているデータベースは多く存在します。こういうデータベースの上では、アプリケーションに本来必要ない処理を書いてしまったり、頻発するバグに追われたりしてたいへん苦労します。また、整合性が崩れるとデータの信頼性がなくなり、アプリケーションの根幹が崩れてしまいます。

情報はビジネスにおける経営資源の四要素（ヒト・モノ・カネ・情報）の1つに挙げられるぐらいですので、信頼性を落とさないように正しくデータを保持したいものです。



アプリケーションから独立させる

“データを正しく持つこと”がデータベースの第一の役割と考えると、なるべくすべてのデータをデータベースで持つことが良い設計であると言えます。

RDBは、トランザクションの途中で変更を安全に巻き戻したり、依存するレコードが存在しないデータの入力をはじいたりと、データを正しく持つことに関してはかなり多くの機能を持ちます。RDBは単体でデータの整合性を保つしくみが備わっているため、永続的に保持すべき情報はアプリケーションではなくデータベースに保持すべきです。

アプリケーション内で定数などの形でデータが保持されていたり、アプリケーションコードを見ないとデータの関連が把握できないといった状態は、データの在り処が分散しているためできる限り避けるべきです。

Ruby on Railsをはじめとする Webアプリケーションフレームワークにはデータベースマイグレーションという機能があり、データベースの構造を定義・変更できますが、アプリケーションと同じ感覚でデータベースに変更を加えていくと、データをソースコード内にも置いてしまいがちです。

データを独立させると、ほかのアプリケーションが同じデータにアクセスする場合にもデータの整合性を取る手間がなくなります。またインデックスで検索を高速に行ったり、高速に集計を行うなどの利点も活かせ、パフォーマンスの向上につながります。

このようにデータベースの役割をしっかりと認識し、住み分けをはっきりさせることが大切です。



データベースを作る

データはアプリケーションの土台となる部分です。RDBはその性質上、あとからのデータ構造の変更には弱いため、アプリケーションを作るもっともはじめの段階で、時間をかけて設計する必要があります。

RDBにはSQLという非常に柔軟な道具があるため、おおよそどんな設計でもデータを扱うことができてしまい、結果、設計が深く考えられていないまま運用されてしまっている場合も少なくないのではないでしょうか。

データベースが動かないとサービスが成り立ちませんし、その設計はサービスによって大き

く異なり最適な形が変わってきますので、けっして蔑ろにしてはいけません。

設計の視点と方法

データベースの設計は、管理するデータを抽象化して、それをデータモデルに落とし込むことを指します。RDBの場合はリレーショナルモデル(関係モデル)になります。

設計のプロセスを分類すると大きく分けて次の3つがあります。

- **概念設計**
  実体(エンティティ)と関連(リレーション)の形でデータの構造を定義する(ユーザと注文の関係は1:N、商品とタグの関係はN:Nなど)
- **論理設計**
  概念モデルをRDBに保持できる形に変換する(データ型や外部キーの設定、正規化など)。機械的に行える作業が多い
- **物理設計**
  パフォーマンスを考慮して設計を最適化する(インデックスを貼る、非正規化、その他RDBごとに依存するものなど)

概念設計の段階で設計の全体を俯瞰するためにはER図を使うと非常に便利です(**図1**)。ER図は実体(エンティティ)と関連(リレーション)を記述する図のことで、おもに概念設計でよく使われます。

論理設計の段階で行う正規化とは、「一事実一ヵ所」という原則に基づき一定の方法でテーブルの設計を加工していくことを言います。一般に、テーブルを分解して数が増えていくことになります。

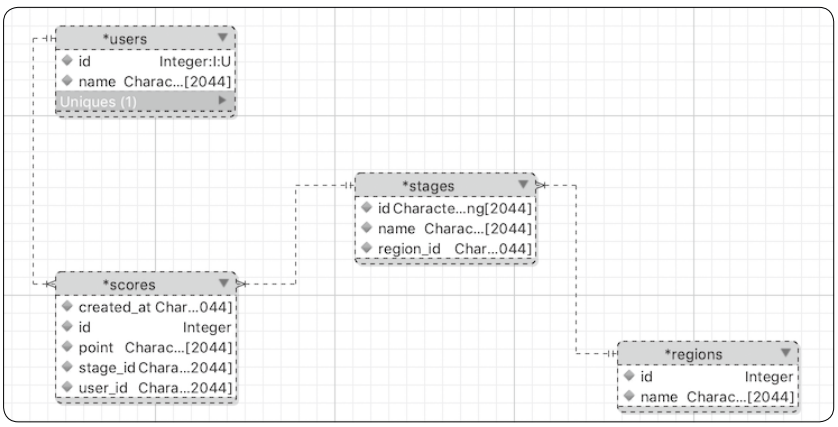
正規形には第一正規形から第五正規形(+ボイス・コッド正規形など)があります。「正規化は第三正規化までで良い」とよく言われるのは、第四正規形またはボイス・コッド正規形以降はレコード間の依存(関数従属性)が保存されないためです。逆に第三正規化までは情報の損失が起こらず、完全に機械的に設計を進めることができるため、必ず第三正規化までは行うというのが通例です(**リスト1**)。正規化については誌面の関係上詳細な説明は省きますが、書籍などを参照してみてください。

整合性を保つ方法

まずは、"正規化を正しく行う"のが大切です。正規化を正しく行うことでデータの冗長性が排除され、独立性が保たれた構造を作れます。これを徹底するだけでもかなり強い設計になります。

テーブルのカラムには外部キー制約を付けることができます。外部キー制約とはデータベースの機能の1つで、指定したカラムに格納するデータをほかのテーブルにある値に制限できます。たとえば、**リスト1**の『第三正規形』における「scoresテーブルのuser_idカラムの値は、usersテーブルのidカラムにある値に限る」など

▼図1　ER図の例



▼リスト1　とあるゲームのスコアを用いた正規化の例

↓非正規形

```
* scores
|username|stage1          |region1|point1|created_at1|stage2     |region2|point2|created_at2|
|--------|----------------|-------|------|-----------|-----------|-------|------|-----------|
|Andy    |Awesome stage   |Tokyo  |3     |2016-06-13 |Great stage|Osaka  |10    |2016-06-14 |
|Bob     |Great stage     |Osaka  |12    |2016-06-15 |           |       |      |           |
|Candy   |Incredible stage|Tokyo  |40    |2016-06-20 |           |       |      |           |
```

↓第一正規形：繰り返しがなく1つの列に1つの値のみを持つ状態（同時に主キー「id」も付加している）

```
* scores
| id | username | stage            | region | point | created_at |
|----|----------|------------------|--------|-------|------------|
| 1  | Andy     | Awesome stage    | Tokyo  | 3     | 2016-06-13 |
| 2  | Andy     | Great stage      | Osaka  | 10    | 2016-06-14 |
| 3  | Bob      | Great stage      | Osaka  | 12    | 2016-06-15 |
| 4  | Candy    | Incredible stage | Tokyo  | 40    | 2016-06-20 |
```

↓第二正規形：部分関数従属する項目を取り除き、非キー属性が主キーへ完全関数従属している状態

```
* scores                                          * users
| id | user_id | stage_id | point | created_at |   | id | name  |
|----|---------|----------|-------|------------|   |----|-------|
| 1  | 1       | 1        | 3     | 2016-06-13 |   | 1  | Andy  |
| 2  | 1       | 2        | 10    | 2016-06-14 |   | 2  | Bob   |
| 3  | 2       | 2        | 12    | 2016-06-15 |   | 3  | Candy |
| 4  | 3       | 3        | 40    | 2016-06-20 |

* stages
| id | name             | region |
|----|------------------|--------|
| 1  | Awesome stage    | Tokyo  |
| 2  | Great stage      | Osaka  |
| 3  | Incredible stage | Tokyo  |
```

↓第三正規形：主キー以外への関数従属を取り除いた状態

```
* scores                                          * users
| id | user_id | stage_id | point | created_at |   | id | name  |
|----|---------|----------|-------|------------|   |----|-------|
| 1  | 1       | 1        | 3     | 2016-06-13 |   | 1  | Andy  |
| 2  | 1       | 2        | 10    | 2016-06-14 |   | 2  | Bob   |
| 3  | 2       | 2        | 12    | 2016-06-15 |   | 3  | Candy |
| 4  | 3       | 3        | 40    | 2016-06-20 |

* stages                                 * regions
| id | name             | region_id |   | id | name  |
|----|------------------|-----------|   |----|-------|
| 1  | Awesome stage    | 1         |   | 1  | Tokyo |
| 2  | Great stage      | 2         |   | 2  | Osaka |
| 3  | Incredible stage | 1         |
```

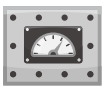
の制限を付けられます。これは整合性を保つためにも原則付けることをお勧めします。整合性の崩れたデータが混入する前にデータベースがエラーを出してそれを止めてくれるからです。

外部キー制約が付いていないデータベースには、親が存在しないと意味を成さないデータや、整合性が保たれていないゴミデータが残っていることが多く、クエリに本来必要ない条件を加えるなどして、筆者自身これまた非常に苦労した覚えがあります。

データ構造がほぼ同じでも、意味が異なるものは別テーブルとするのが原則です。たとえば会計のしくみにおいては、収入と支出のデータ構造はそれぞれ(id, 金額, 内訳, 日付)という同じ構造で表せるかもしれませんが、意味がまったく異なります。

必要十分な型定義も重要です。アプリケーションのバグによって起こる無意味なデータの混入を防ぐことができます。

RDBは正規化をはじめとして理論が整っているため、明確なアンチパターンが存在します。想定していないデータがあるとアプリケーション側で多くの苦労をすることになります。INSERTやUPDATEのたびにアプリケーションで整合性を保とうとするのは非常に手間で、バグの温床となります。面倒でも整合性を保つ設計にする努力を惜しまないようにしましょう。



データベースを使う

それでは、データベースをアプリケーションから利用するときには、何を心がけたら良いでしょうか。



データにアクセスする回数を少なくする

データベースはアプリケーションから独立に存在して別のプロセスやサーバで動くので、扱う際には通信が発生し、単にアプリケーションで処理を行う場合と比較して大きなコストを伴います。このため、そもそもデータベースへのアクセスの回数を減らすことがパフォーマンス向上のための1つの方針になります。

よくあるのが、アプリケーションの処理のループの中でデータベースアクセスをしてしまい、一覧取得(1回)＋個々のデータの取得(N回)のN+1回のアクセスが発生してしまう場合です。

多くのWebアプリケーションフレームワークでは、データベースへのアクセスがO/Rマッパーという形で使いやすくカプセル化されているため、実際に吐かれているクエリの内容や回数を意識せずにデータを扱ってしまいがちです。

アプリケーションのパフォーマンスが悪い場合は、まずはこのような処理が書かれていないかを疑ってみましょう。データベースに必要なパフォーマンスを実現するにはO/Rマッパーが生成するSQLでは不十分な場合も多々あります。



SQLを改善する

だいたいのRDBにはスロークエリと呼ばれる、一定時間以上かかったクエリをログに出力する機能が備わっています。これを見ることで、どのSQLに問題があるかを洗い出せます。初期設定では、1秒など致命的に遅いものを検出するのみとなっていますので、チューニングを進めるにつれてこの閾値(いきち)を下げていくと良いでしょう。数十ミリ秒以上かかっているクエリがあれば何かしらの問題を抱えていることが多いです。

個々のSQLのパフォーマンスの詳細を見るにはEXPLAINが便利です。任意のクエリの前にEXPLAINを付加することで、その実行計画を見られます(**図2**)。実行計画とは、RDBがSQLを実行する前に立てる内部の処理の計画のことで、どのような順番でデータが絞り込まれるのか、インデックスがどう使われているのかなどを確認できます。インデックスが利用されず、シーケンシャルスキャンとなっているクエリには要注意です。

SELECT文のパフォーマンスを向上するために、インデックスがよく使われます。インデックスにもB+Treeインデックス、ビットマップインデックスなど種類があり、データや検索の

しかたによって適切なインデックスが異なります。

ほかにも、集計処理もプログラミング言語で手続きを記述するよりもSQLを書いてデータベースに任せたほうが高速になる場合が多いなど、考える点が非常に多くあります。データベースが得意な処理はデータベースに任せるようにするのが、バグも少なく速いアプリケーションを作る近道です。



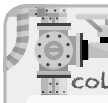
まとめ

データベースは正しくデータを持つものとして、アプリケーションにとってけっして軽視できない存在です。RDBという枯れた技術の基礎を知ることで、開発工数の削減やパフォーマンスの高いアプリケーションを実現できます。

アプリケーションエンジニアに比べてデータベースに強いエンジニアは非常に少なく、詳しくなれば希少な価値になれるでしょう。またとても奥が深い領域でもありますので、興味が出たらデータベース設計やSQLチューニングに関する書籍を読んでみると良いと思います。SD

▼図2　PostgreSQLにおけるEXPLAINの例

```
=# EXPLAIN SELECT u.id, u.name, q.id FROM users u LEFT JOIN questions q ON q.user_id = u.id WHERE
q.created > CURRENT_TIMESTAMP -  INTERVAL '2 days' ORDER BY u.id DESC LIMIT 100;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------
 Limit  (cost=3109.14..3109.14 rows=2 width=17)
   ->  Sort  (cost=3109.14..3109.14 rows=2 width=17)
         Sort Key: u.id
         ->  Nested Loop  (cost=0.29..3109.13 rows=2 width=17)
               ->  Seq Scan on questions q  (cost=0.00..3092.49 rows=2 width=8)
                     Filter: (created > (now() - '2 days'::interval))
               ->  Index Scan using users_pkey on users u  (cost=0.29..8.31 rows=1 width=13)
                     Index Cond: (id = q.user_id)
(8 行)
```



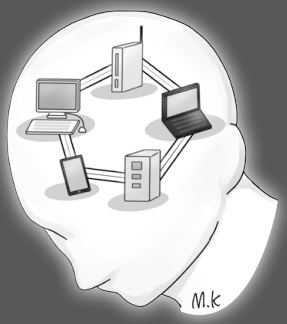
delete_flagは本当に必要か

データベースの設計でよく議論の話題に上がるものに、delete_flagと呼ばれる、いわゆる論理削除カラムがあります。「delete_flagがtrueの際には削除されているとみなす」ように使われ、デフォルトでfalse、削除されているとみなしたい場合はtrueとします。is_deletedや、deletedとして日付を入れるパターンもあるようです。

この論理削除カラムは、実際にデータを消さずに消したことにできる(?)ため一見便利なように見えますが、何も考えずにこのカラムを全テーブルに付けようとするのは非常に危険です。そのような設計にすると毎回のSQLで「delete_flag = FALSE」のように条件を指定する必要があり、データがあるかないかを常に判断しないといけない状態は開発者にとって非常に負担ですし、バグの温床にもなります。

削除したデータを簡単に戻せるというのもdelete_flagが正当化される1つの理由ですが、もし戻せるのであればそれは厳密には削除ではなく、単に状態が変化しているものです。たとえばユーザから見えなくするだけなら、is_visibleのような名前が適切なものになります。また、インデックスが適用されづらくパフォーマンスがでないなど多くの問題があります。

そもそも「特定のデータをアプリケーションでは扱いたくないけど記録としてデータは保持したい」という目的があるなら、何も考えずにdelete_flagを作るのではなく、削除したデータや履歴を保持するしくみを別に作るのが正しいと思います。



仮想化の知識、再点検しませんか?

使って考える仮想化技術

本連載は「仮想化を使う中で問題の解決を行いつつ、残される課題を整理する」ことをテーマに、小さな仮想化環境の構築・運用からはじめてそのしくみを学び、現実的なネットワーク環境への実践、そして問題点・課題を考えます。仮想化環境を扱うエンジニアに必要な知識を身につけてください。

Author
笠野 英松(Mat Kasano)
有限会社 ネットワーク・メンター
URL http://www.network-mentor.com/indexj.html

第2回 ホストシステムと仮想環境の構築(その1)

テスト用PC構築の前に

今回からハイパーバイザ型サーバ仮想化環境を実際に使っていきますが、まずはテスト環境を構築しなければ始まりません。CentOS 6.7上のKVMを使ってホストシステムや仮想環境の構築、仮想マシン/ゲストシステムの作成・利用までを解説していきますが、紙幅の都合上、数回にわたってしまうことをご承知おきください。まず最初に、「WARNING」の囲み記事に必ず目を通しておいてください。以降の本連載では記事検証用に用意した専用PCを使います。

連載の当初はできるだけデフォルトや簡単な選択操作で、かつファイル編集などを行わないファイル内容確認などを行います。仮想環境や仮想マシンとは実際にはどのようなものか、デフォルト最低限の仮想環境で単一仮想マシンシステムを使ってみることで、その不具合や不便などを認識し、どう対処するか、あるいは何が必要かなど仮想化の問題を考えていきましょう。

今回は専用PCのOS環境構築までを行い、以降、仮想マシンの作成とゲストシステムのインストール、仮想マシン・ゲストOSの利用、仮想マシンの管理へと進めて行きます。

なおKVM仮想化では、仮想化支援機能搭載プロセッサ、それを使用するKVMカーネルモジュール、KVM上で仮想マシンI/Oをエミュレートする QEMU-KVM(QEMUエミュレータのKVM版)を使用します。

使用するシステム環境

今回は初めてなので、できるだけデフォルトの最小設定・環境を使って行います。

- 最低限のパッケージの仮想化(「Virtual Host」)とデスクトップ(「Desktop」)
- 仮想マシン管理は「仮想マシンマネージャー」
- ゲストOSはDVDインストール
- 仮想ネットワーク環境はNAT接続
- インターネット接続にルータ使用

今回の進め方の要点

今回はOSインストールを行い、仮想環境構築や仮想マシン作成(次回以降)の準備・確認を行います。具体的には次のようになります。

①OSとKVMのインストール、ログイン
②システム/ネットワークなどの設定の確認
③仮想化やファイアウォールの設定の確認
④ハイパーバイザの起動確認

それぞれの段階でつまずきやすいポイントも挙げていきます。

WARNING!!! 本連載を通しての重要な注意「専用PCを使う」 WARNING!!!

本連載で使用するシステムでは、仮想化に特化したさまざまな処理を行い、さまざまな問題に対処するためにテストをします。逆に言うと「仮想化に関係がある・なしにかかわらず問題が発生する可能性がある」システムとなるので「PCのディスク全体を初期化や上書きしながら使用する」ことになります。したがって、Windowsなどと共用するマルチブート環境で行うとすべてのデータが消えてしまいます。本連載に関係する処理はマルチブート環境では絶対に行わないでください。

また、VMware Workstation PlayerやVirtualBoxなど既存OS上の仮想化ソフトウェアを使用して、仮想化ソフトウェア上での仮想化インフラの構築（Nested/入れ子仮想化※1）を行うためには、Intel VT-x/EPT※2またはAMD-V/RVI※3が必要になります（Intel VT-xやAMD-Vのみでは不可※4）。そのうえCPU速度やメモリ、ディスクなどに多大な要件・容量が必要になるなどがあります。

今回の連載を試すために、専用のPCを用意することを強く推奨します（不具合やトラブルなどを究明するときに切り分けもしやすくなります）。なお本記事の内容に基づいたいかなる行為も読者様ご自身の責任と判断によって行ってください。技術評論社および著者は責任を負わないものとします。ご注意ください。

なお当然のことですが、本連載とは関係ない仮想化で、かつ仮想化に慣れておられるならば、マルチブート環境やNested仮想化での仮想化の利用を妨げるものではありません。

※1 Nested……ページテーブルやセカンドレベルアドレス変換（SLAT：Second Level Address Translation）というCPUの仮想化支援機能で、Intel VT-xやAMD-Vに実装されている。
※2 Extended Page Tables：Core i7（コード名 Nehalem）から。
※3 Rapid Virtualization Indexing：K10（コード名 Barcelona）から。
※4 具体的には、Nested仮想マシンの作成ができないとか、Nested仮想マシン上のゲストOSのインストール時にメモリパニックになるなどの問題が生ずる。

OSインストールからログインまで

CentOS 6.7インストールディスクの準備

CentOS 6.7のisoファイルはCentOSミラーサイト注1から入手してDVD化します。なお、DVDインストーラを作るisoファイルは2つありますが「CentOS-6.7-x86_64-bin-DVD1.iso」のみを使用します。

[システム要件]
- システム：専用システム（必須）
- CPU：仮想化支援機能搭載プロセッサ Intel VT-x（やIntel VT-x/EPT）注2またはAMD-V（やAMD-V/RVI）
- RAM：4GB以上
- ディスク：40GB以上（80GB以上推奨）注3
- 有線LAN
- インターネット接続環境（ルータ／DNS）注4

設定値

今回のインストールでは**表1**のようにデータ設定をしました。なお、ハードディスク領域の設定で今回の設定とは異なる「カスタムレイアウトを作成する」を行うことでハードディスクを自由に設定可能ですが、本連載ではトラブルや手操作ミスをなくすために、「すべての領域を使用する」を選択設定します。

インストール手順

表2の手順（**表1**の設定値）でインストールからログインまで行います。ストレージデバイス

注1） http://archive.kernel.org/centos-vault/6.7/isos/x86_64/
注2） http://ark.intel.com/Products/VirtualizationTechnology/
注3） ディスク全体を使用。
注4） yumオンライン接続やゲストのインターネット接続テストのため。

▼表1　今回のインストール設定値

設定項目	設定値
ディスク／パーティション	搭載ハードディスク領域すべて使用
ストレージデバイスタイプ	デフォルト
IPv4ネットワーク設定：ホスト名	vm1.example.com、192.168.0.111/24
：ゲートウェイ／DNS	192.168.0.1（インターネット接続可能対応設定）
タイムゾーン	アジア／東京、UTCチェックオフ
パーティション作成	すべての領域を使用する
ソフトウェアセット選択	Virtual Host
カスタマイズ	今すぐカスタマイズ
	デスクトップ／デスクトップ
ユーザー	user1
日付と時刻	手動設定または自動

のフォーマット（初期化）やディスク全体への書き込みについて警告が表示されますが、「WARNING」で説明したとおりディスク全体の初期化・上書きを許可します。また、ソフトウェアセットの選択では仮想化「Virtual Host」と、GUIウィンドウ操作のためにソフトウェアカスタマイズで「デスクトップ」を選択します。

以降は、インストール、再起動、最終設定後、CentOSログイン画面になります。rootでログインし、コマンド入力用の「端末」ランチャ（アイコン）をデスクトップに設定しておきます。

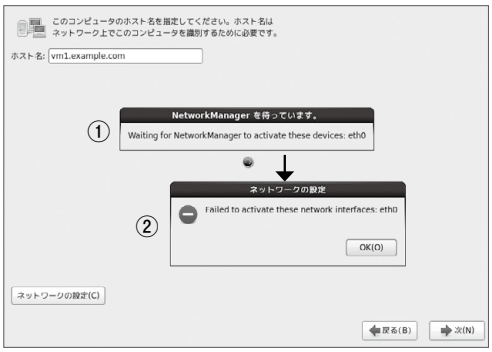
ログインまでに起こる問題と対処

システム選択からログインまでで発生する可能性があるトラブルを見てみましょう。

ネットワーク設定が終了できない

図1の①の表示後しばらくして②のダイアログが表示され、ネットワーク設定が終了できなくなることがあります。

▼図1　「ネットワークの設定」中の接続断絶によるエラーメッセージ



原因と対処

ネットワークを管理制御するNetworkManagerが、ネットワークインターフェースを有効にできないエラー表示で、ネットワーク接続が切れている可能性があります。ケーブルやコネクタなどを確認しましょう。

CUIログインになってしまう

インストール完了後の再起動時に、次のようなCUI黒画面のウェルカムメッセージ／ログインプロンプト表示になってしまう場合です。

```
CentOS release 6.7 (Final)
Kernel 2.6.32-573.el6.x86_64 on an x86_64

vm1 login:
```

原因と対処

表2インストール手順の12番「ソフトウェアカスタマイズ」でデスクトップを選択していない、もしくはインストールされていないためです。再インストール、またはrootでログイン後に次のコマンドでデスクトップをインストールしてから再起動します。

▼表2　CentOS（6.7）インストール手順

作業番号	画面／処理	設定値
	【CentOS 6.7 isoメディアをマウントして起動】	
	Welcome to CentOS 6.7!	
	Install or upgrade an existing system	⇒選択/ENTER
1	Disc Found（メディアテスト）	⇒[OK]/[Skip]
2	CentOS 6 開始画面	⇒Next
3	Install Language	⇒[Japanese（日本語）]⇒Next
4	キーボード選択	⇒[日本語]⇒次
5	ストレージデバイスタイプ選択	
	●基本ストレージデバイス	⇒次【注意】ディスク領域全体は初期化される
	…ストレージデバイスの警告…	
	「はい。含まれていません。どのようなデータであっても破棄してください。（Y）」	⇒選択
6	●新規インストール	⇒次
7	ホスト名設定	⇒ホスト名
	[ネットワークの設定]／ネットワーク接続／有線／System eth0	⇒編集
	■自動接続する	
	■すべてのユーザに利用可能	
	[IPv4のセッティング]	
	方式　手動	⇒[追加]
	アドレス／ネットマスク／ゲートウェイ	⇒設定
	DNSサーバ	⇒設定
		⇒適用⇒閉じる⇒次
8	タイムゾーンの選択	⇒東京，アジア／システムクロックでUTCを使用（オフ）⇒次
9	rootパスワード	⇒設定および確認⇒次
10	パーティション（領域）の作成方法選択	
	●すべての領域を使用する（既存ならば削除して作成）	⇒選択⇒次【注意】ディスク領域全体は初期化される（連載専用PC）
	…フォーマットの警告…	⇒[フォーマット]
	…ストレージ構成をディスクに書き込み中…	⇒[変更をディスクに書き込む]
	…作成／フォーマット中	
11	ソフトウェアセットの選択	
	●Virtual Host	⇒選択
	ソフトウェア選択のカスタマイズ方法	
	●今すぐカスタマイズ	⇒選択
12	ソフトウェアカスタマイズ	
	デスクトップ■デスクトップ	⇒選択⇒次
	……依存関係チェック……	
13	インストール開始	
14	（完了メッセージ）	⇒再起動
	【再起動】	
15	ようこそ	⇒進む
16	ライセンス情報	
	●はい、ライセンス同意書に同意します	⇒進む
17	ユーザの作成	⇒設定⇒進む
18	日付と時刻	
	ネットワーク上で日付と時刻を同期化します	⇒（オン：自動、または、オフ：手動で日時設定）⇒進む
19	kdump／kdumpを有効にしますか（オフ）	⇒終了
	【CentOS ログイン画面】	
20	ユーザ選択	⇒「その他」⇒ユーザ名：root／パスワード
	…root スーパーユーザで実行する警告メッセージ…	
	■再度表示しない（チェック）	⇒[閉じる]
	【ログイン後のウィンドウ】	
	「システムツール→端末」	⇒「このランチャをデスクトップへ追加」
	（以降、「端末」ランチャ（アイコン）を使用して管理作業を行う）	

```
# yum -y groupinstall Desktop "Japanese Support"
```

なおこの場合は、インストール後に/etc/inittabをviなどのエディタで開き、実質1行めの「id:3:initdefault:」の「3」を、「5」に変更してください。さもないと再起動後もCUIログインになり、毎回コマンド「startx」でGUI画面に入る必要があります。

ログイン後のシステム確認

作成した仮想化インフラシステムの設定を確認しておきます。基本設定(システム／ネットワーク／ディスク設定)と仮想環境設定(仮想化インフラKVMやファイアウォールの設定)、ハイパーバイザの起動確認です。

基本設定の確認

図2のように、ホストの物理システムと仮想環境側のIPアドレスを確認します。

ネットワーク接続の確認

図3のように、システムの物理／仮想アドレス、ルータ、そして、インターネット外部ホストへのネットワーク接続をpingで確認します。

ログイン後のシステム確認での問題

ネットワークがつながっていない

図3での確認の際、「connect: ネットワークに届きません」などのメッセージ表示はシステムの物理アドレスやルータにpingが通らない状態です(仮想アドレスにはpingが通っています)。また**図2**のifconfig確認で、正常な場合に表示される物理アドレス行やステータスの「UP BROADCAST RUNNING MULTICAST」中の「RUNNING」が表示されません。なお、DNSは適切な設定が前提です。

▼図2　システムの基本設定の確認

```
 確認用コマンド(root実行)
ifconfig -a

 表示確認事項：IPアドレス／マスク、接続ステータス
 eth0(抜粋)
inet addr:192.168.0.111  Bcast:192.168.0.255  Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
 lo(抜粋)
inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:65536  Metric:1
 virbr0(抜粋)
inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

▼図3　ネットワーク接続の確認

```
 確認用コマンド(root実行)
ping -c 3 192.168.0.111     自システム
ping -c 3 192.168.122.1     仮想側アドレス
ping -c 3 192.168.0.1       ルータ
ping -c 3 www.yahoo.co.jp   インターネットのシステム(ルータ/DNSがインターネット接続可能設定時)

 表示確認例(抜粋)
64 bytes from 宛先アドレス: icmp_seq=1 ttl=64 time=0.136 ms
```

▼図4　ネットワーク接続の自動接続設定（[システム]→[ネットワーク接続]→[System eth0]→[編集...]）



原因と対処1（ケーブル離脱）

この場合、ifconfigの表示中、ステータス行の「RUNNING」のみが表示されません。ケーブルが正しく接続されているか、ケーブルのコネクタが正しく装着されているか、などを確認します。

原因と対処2（eth0が停止）

この場合、ifconfigの表示中、物理アドレス192.168.0.111の行と、ステータス行の「RUNNING」の両方が表示されません。**図4**のように、「System eth0の編集」ダイアログで「自動接続する」にチェックを入れます。

設定後、ネットワークを「service network restart」で再起動してeth0を有効化します。

原因と対処3（ルータ接続失敗）

図4の[IPv4のセッティング]タブを開いてゲートウェイを確認します。「0.0.0.0」となっていて設定がされていなければ適切に設定します。

設定後、ネットワークを「service network restart」で再起動します。

ディスク設定の確認

図5のように、ディスクは論理ボリュームで/rootとswap領域、実ボリュームで/bootが作成されていることがわかります。

仮想環境設定の確認

図6のように、仮想環境設定を確認します。

▼図5　ディスク設定の確認

```
 ①ディスク割り当て確認用コマンド（root実行）
fdisk -l

 表示確認例（抜粋）
ディスク /dev/sda: 21.5 GB, 21474836480 バイト
デバイス    ブート  始点    終点     ブロック   Id  システム
/dev/sda1    *       1      64       512000    83  Linux   ← /boot
/dev/sda2           64    2611     20458496    8e  Linux LVM
ディスク /dev/mapper/vg_vm1-lv_root: 18.8 GB, 18798870528 バイト   ← /root
ディスク /dev/mapper/vg_vm1-lv_swap: 2147 MB, 2147483648 バイト    ← swap

 ②ファイルシステム設定確認用コマンド（root実行）
more /etc/fstab

 表示確認例（抜粋）
/dev/mapper/vg_vm1-lv_root /                       ext4    defaults   1 1
UUID=8e187e98-f253-40f3-8fb5-8d0dc6ea14ca /boot    ext4    defaults   1 2
/dev/mapper/vg_vm1-lv_swap swap                    swap    defaults   0 0

 ③領域の使用と空き確認用コマンド（root実行）
df -h

 表示確認例（抜粋）
Filesystem            Size  Used  Avail  Use%  Mounted on
/dev/mapper/vg_vm1-lv_root
                       18G  2.8G    14G   18%  /
/dev/sda1             477M   36M   416M    8%  /boot
```

▼図6　仮想環境の設定

```
 ①KVM設定ディレクトリ確認用コマンド(root実行)
ls -al /etc/libvirt/qemu

 表示確認例(抜粋)
drwx------. 3 root root 4096  4月 13 20:04 2016 networks

 ②仮想ネットワークディレクトリ確認用コマンド(root実行)
ls -al /etc/libvirt/qemu/networks/

 表示確認例(抜粋)
drwx------. 2 root root 4096  4月 13 20:04 2016 autostart   自動起動
-rw-r--r--. 1 root root  319  4月 13 20:04 2016 default.xml   仮想ネットワークデフォルト

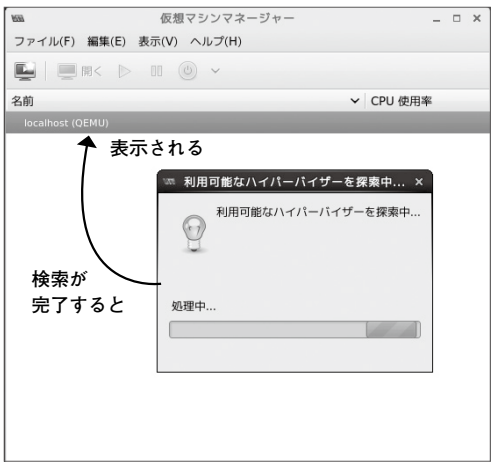
 ③デフォルト設定内容確認用コマンド(root実行)
more /etc/libvirt/qemu/networks/default.xml

 表示確認例
<network>
  <name>default</name>
  <uuid>e0a58970-3ef3-4b9d-af36-14b2a6820962</uuid>
  <bridge name="virbr0" />
  <mac address='52:54:00:7B:8F:80'/>
  <forward/>
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2" end="192.168.122.254" />
    </dhcp>
  </ip>
</network>

 ④ファイアウォール設定確認用コマンド(root実行)
iptables --list

 表示確認例(抜粋)
Chain FORWARD (policy ACCEPT)
target   prot opt source            destination
ACCEPT   all  --  anywhere          192.168.122.0/24   state RELATED,ESTABLISHED
ACCEPT   all  --  192.168.122.0/24  anywhere
REJECT   all  --  anywhere          anywhere           reject-with icmp-port-unreachable
```

▼図7　QEMU-KVMハイパーバイザの起動確認



KVM設定は/etc/libvirt/qemuにあり、仮想ネットワークのIPアドレス設定は/etc/libvirt/qemu/networks/default.xmlに記述されています。仮想マシンのアドレスは192.168.122.0/24で、DHCP(デフォルトで有効)で192.168.122.2から割り当てます。

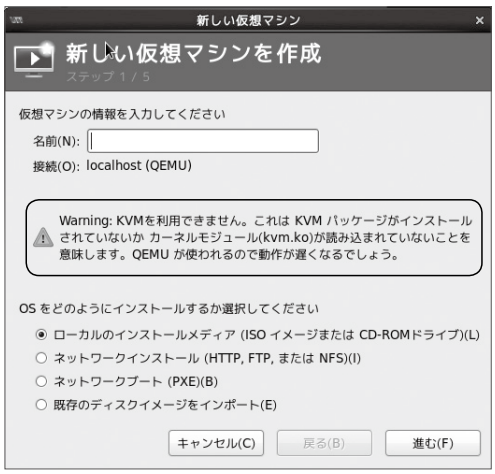
ファイアウォールiptablesのルールでのポイントは、FORWARDチェインの「192.168.122.0/24」に関する2行で、仮想マシン側からの発信接続が有効になっているものの、物理ホストネットワークから仮想マシンへの接続は拒否される設定です。

▼図8　ハードウェア仮想化支援機能の確認

```
 確認用コマンド（root実行）
egrep -e 'vmx|svm' /proc/cpuinfo

 表示確認例（抜粋。...部は省略）
flags  : ... pae ... mmx ... lm ... vmx（svm） ...
 pae=PAE、mmx=SIMD型拡張命令セット、lm=64ビットCPU、vmx=intel-VT（svm=amd-V）
```

▼図9　仮想マシンマネージャーで
KVMカーネルモジュールが有効でない場合



QEMUハイパーバイザの起動確認

メニューの[アプリケーション]→[システムツール]→[仮想マシンマネージャー]で仮想マシンマネージャーを開いて、ローカルのQEMUハイパーバイザが起動できるかを確認しておきます。起動していないならば以降の仮想化処理はできません。

図7のように「利用可能なハイパーバイザーを探索中...」が表示された後、仮想マシンマネージャー画面に「localhost（QEMU）」が現れ、ローカルQEMUハイパーバイザが利用可能になっていることがわかります。

KVMカーネルモジュールを確認

コマンド「lsmod | grep kvm」を実行して、LinuxのKVMカーネルモジュールがロードされているか確認します。「kvm」と「kvm_intel（または、kvm_amd）」の2行が表示されればOKです。何も表示されなければロードされていない（無効）ので、次の対処を行います。

原因と対処

図8のようにしてハードウェア仮想化支援機能が有効になっているか確認します。コマンド「egrep -e 'vmx|svm' /proc/cpuinfo」で何も表示されない場合、システムでハードウェア仮想化支援機能が装備されていないか、または装備されていてもBIOSで有効になっていないかのどちらかです。BIOSで無効になっているならば有効にします。装備されていないならば不可能です。

なお、KVMカーネルモジュールが有効でない場合、**図9**のように、仮想マシンマネージャーで新規仮想マシンを作成しようとするときも、KVMカーネルモジュールの警告が表示されます。

次回予告

今回はCentOSをインストールし、ハイパーバイザの起動とKVMカーネルモジュールが有効になっているかまでを確認しました。次回は仮想マシンを作って試してみましょう。SD

連載各回では、読者皆さんからの簡単な「ひとくち質問（QA）コーナー」や「何かこんなこともしてほしい要望（トライリクエスト）コーナー」を設けて「双方向連載」にできればと思っていますので、質問や要望をお寄せください。

宛先：sd@gihyo.co.jp
件名に、［仮想化連載］とつけてください

RDB性能トラブルバスターズ奮闘記



原案 生島 勘富（いくしま さだよし） info@g1sys.co.jp ㈱ジーワンシステム
構成・文 開米 瑞浩（かいまい みずひろ） イラスト フクモトミホ

第5回 インデックスが動くしくみを理解しよう

RDB（Relational Database）のインデックスは、検索を高速化するために有効な技術です。今回、大道君はインデックスのしくみを学びます。しくみがわかれば、実環境に適用する前にインデックスが有効なケースと、そうでないケースを予測できるようになります。

登場人物紹介



生島氏
DBコンサルタント。性能トラブルの応援のため大道君の会社に来た。



大道君
浪速システムズの新米エンジニア。素直さとヤル気が取り柄。



五代氏
大道君の上司。プロジェクトリーダーでもある。

自分が教える側になれば一番よく勉強できる

大阪を中心に20年間システム開発に携わったあと、現在は東京で仕事をしているSQLの伝道師ことジーワンシステムの生島です。それはジトジトした梅雨空が広がるある日のこと、大道君が真剣な表情で聞いてきました。

「生島さん、インデックスについて教えてください！」

「おお？　今回はなんや、また何か遅いコードが出てきたんか？」

「いえ、単に勉強したいんです」

今までは実システムでの性能トラブルがあって、私がその改善策の相談に乗る中で大道君にRDB技術の基本を教えるという形で彼にかかわってきましたが、今回はとくにトラブルがあったわけではないとのこと。

「今までのことを振り返ってみると、RDBのしくみをきちんと理解して使わないとこういうトラブルはなくならないな、と思えたので、時間があるときに勉強しておきたいんです！」

「おお！　そら、ええ心がけや！」

すばらしい！　この姿勢は本当に大事です。思わずナデナデしたくなりましたが、キモいのでやめておきました（笑）。とはいえ、一から十まで私が手取り足取り解説するのは、緊急のトラブル対応時ならともかく、平時には効率が悪過ぎます。幸い現代では市販の雑誌／書籍やネット上にさまざまな技術解説があるので、まずはそれらを利用することにしましょう。

「ほんじゃま、まず自分でインデックスについて調べてみて、それを自分の後輩に教えるつもりで発表してみよか？」

「えっ……ぼ、僕がですか？」

「自分が教える側になるのが一番いい勉強になるからな～」

「は、はい……ひえ～」

技術的知識の構造を意識して整理しよう

といっても完全に放っておくのもそれはそれで効率が悪いので、少し助け船を出しました。

「ついては、**図1**の『技術的知識の構造』を意識して整理するとええよ」

ある「状況」において、ある「問題」つまり何か困ったことが起きる、それを解決するために、何らかの「効果」を求めて、ある「用法」を適用する、その用法は「しくみ（動作原理）」に基づいて

いる、と、こんな構造に沿って技術的知識を整理すると、理解しやすく説明しやすいことが多いのです。

「状況というのは……」

「たとえば前回の話だったら、『顧客別に売上を集計する』というのが状況で、『遅い』というのが問題やね」

「じゃあ効果は、それを速くすることですか。そして用法はサブクエリを使わないようにすること？」

「そのとおり！」

「しくみは……」

「サブクエリを使うとインデックスが無効になるので、テーブルの全件アクセスが起きる、というのがしくみ」

「あ、なるほど、そうですね……」

「人に発表するときはとくに重要なんやけど、『こういう状況だとこういう問題が起きるから、困りますよね？』という話をまずするんよ。人間、困るなあと思ったことは人の話を聞こうとするから。そこでおもむろに、『その困った問題を解決するためには、こんな効果がほしいですよね、その効果はこの用法で得られます』と話を続ける」

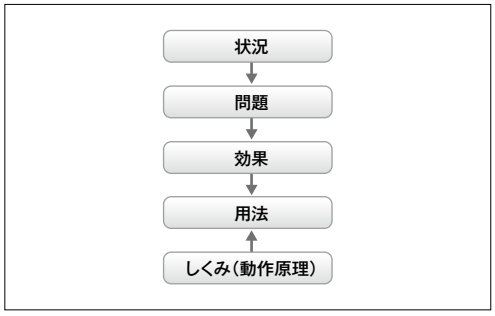
「おー、まさに私それやってますわ！」といきなり口を挟んできたかと思うと、あっという間に去って行ったのは五代さん。お客様に提案をするときの基本パターンの1つだそうです。

「で、じゃあなぜその用法で効果が出るのか？という理由を知るために必要なのがしくみ、動作原理というわけや」

「了解です！」

ということで1週間後に大道君がインデックス技術について発表する社内勉強会を開催することになりました。

▼図1　技術的知識の構造



インデックスがない検索はなぜ遅い？

そして早くも1週間後。浪速システムズの一室にて、私と五代さんほか2名の若いエンジニアを前に緊張した面持ちの大道君による発表会が始まりました。

「今回はRDBの検索を高速化するために有効なインデックス技術についてお話しします。まずはインデックスがない場合にどのようなことが起こるかを理解するために、ある状況をイメージしてください」

会話体で進めると長いので、ここから先は大道君の話を私がまとめた形でお伝えしましょう。

ある大きなテーブル、たとえば100万人分の会員マスタがあるとします。ここからたとえば「アオキ ユウコ」という名前の会員を検索したところ、非常に遅かったとしましょう。**図1**のフレームワークを使うと次のようになります。

- 状況：大きなテーブル（会員名簿100万件）から読み仮名項目で「アオキ ユウコ」を検索する
- 問題：レスポンスが遅い





- 効果：レスポンスを改善する
- 用法：読み仮名項目にインデックスを使用する

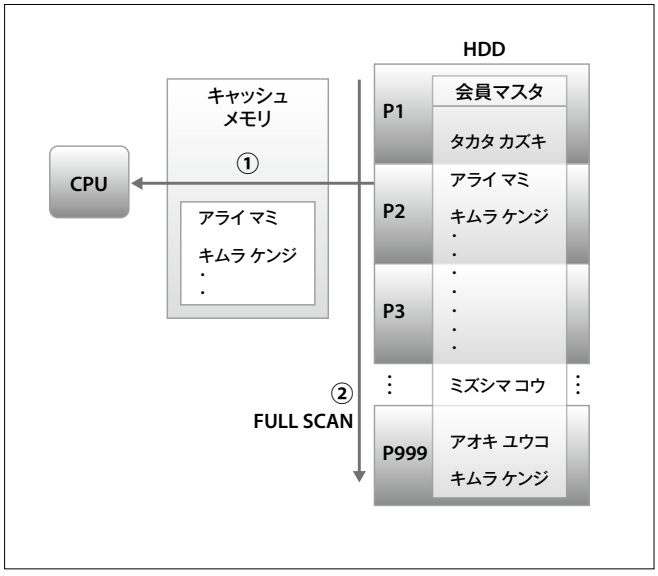
現実の問題解決をする場合には、「問題」と「効果」が上例のように裏返しにはならない場合も多いですが、本稿では扱いません。

さて、「読み仮名項目にインデックスを使用する」というのはこの問題を解決する有力な案ですが、なぜそれが効果を発揮するのでしょうか？ それを理解するためにはインデックスのしくみを知らなければなりません。そのために、まずインデックスがない場合の検索動作がなぜ遅いのかを確認しましょう。ということで**図2**です。

テーブルはハードディスク（以下、HDD）上に「ページ」単位で保存されています。ページというのはMySQLでHDDを読み出す単位を呼ぶ用語で、OracleやSQL Serverでは「ブロック」と呼びます。P1、P2……P999が「ページ」です。矢印①が示すように、HDD上のデータはページ単位で読み出され、いったんキャッシュメモリに載り、それをCPUが処理します。

図2はHDD上のP2のデータがキャッシュに載っている状態を示しています。この状態で検索をすると、P2の分のデータはキャッシュが使われますが、それ以外についてはHDDを読みに行くことになります。それぞれのデータはいったん読まれるとキャッシュに載りますが、キャッシュメモリの容量を超えると捨てられてしまうため、次の検索時にはまたHDDアクセスが発生します。HDDはメモリよりも格段に遅いため、高速化するためにはHDDアクセスを減らさなければなりません。しかし、テーブルのデータはソートされていないため、インデックスがないと目的のデータがどのHDDページに載っているかわかりません。結局、矢印②が示すようにP1から延々とHDDを999ページ（ただしキャッシュにある分を除く）読み続けなければいけないのです。

▼図2　大きなテーブルの検索（インデックスなし）



ユニーク性が保証されない項目の検索とは

と、そこまで聞いたところで質問してみました。

「同姓同名というか、読みが同じ名前があったらどうなるんや？」

「いい質問ですね！　たとえば『キムラ ケンジ』を検索したとき、P2のデータはキャッシュにあるのですぐ発見できますが、読み仮名はユニーク項目じゃありません。ほかにも同姓同名がいるかもしれないので、そこで検索を終われないんです。実際P999でもう1人見つかります。結局、インデックスなしで読み仮名検索すると、検索完了までに必ずテーブル全件の読み出し（FULL SCAN）が発生しますので非常に遅くなります！」

うーむ、完璧な回答。しかも「いい質問ですね！」で返すとは大道君恐るべし。

結局、たった1件のデータを探すためにHDDを999ページ（もちろん、データ量が多ければそれ以上）、FULL SCANしなければいけない、というのはあまりにも無

駄が多いわけです。なんとかこれを減らせれば高速化できます。多くのページを読まずに、目的のデータが存在するページを特定する方法があればいいのですが、でも、どうやって？

インデックスが効くと無駄なページを読まずに済む

ここで役に立つのがインデックスで、そのしくみのイメージが**図3**です。

読み仮名項目にB+tree（ビーツリー）という方式のインデックスを付けると、**図3**左側のようなデータ構造が作られます[注1]。ここでたとえば「キムラ ケンジ」を検索する場合、まずインデックス左端の「タヤ」というノードを読み、「キ」は「タ」の前にあるので次は①をたどって「カサタ」のノードを読みます。次いで「キ」は「カ」のあとにあるので今度は②をたどって「カキクケコ」のノードを読み、そこで得られる「キ」のリーフには、キから始まる名前へのポインタがすべて収納されているため、③、④のリンクをたどって「キムラ ケンジ」が存在する2ヵ所のHDDページだけを直接読み出すことができます。

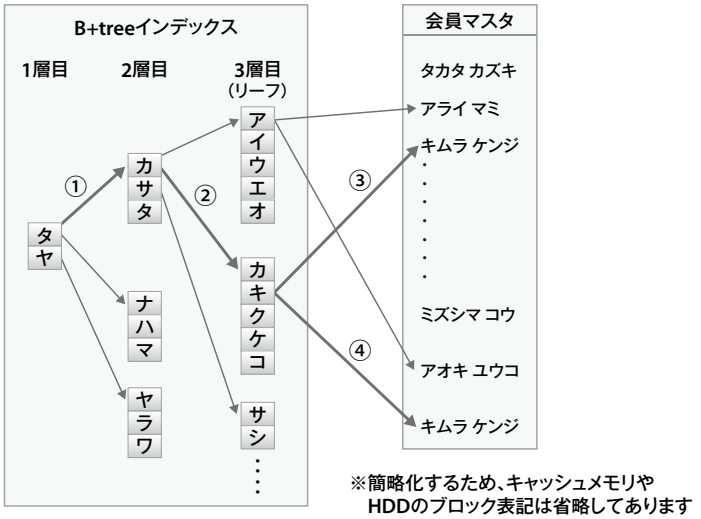
これがINDEX SCANで、FULL SCANと違って無駄なページをいっさい読まないので短時間で検索できるわけです。なお、「リーフ」というのはインデックスの末端ノード、**図3**で言えば3層目のことを言います。ただし、**図3**は簡略化しているため3層ですが、リーフを含むB+treeの階層数は実際には4層以上のことが多いです。

また、MySQL（InnoDB）ではプライマリーキー・インデックスのリーフ自体にテーブルの実データが載っているため、プライマリーキーでの検索やその順序での出力が速いです。それに対して、Oracleではプライマリーキーであってもリーフには実データではなくポインタが載っているため、どのキーでもバランスよく性能を発揮するといった細かい違いがあります。

「読み仮名項目のデータをこのように木構造化すれば、インデックスを3階層読むために3ページ、実データに1ページ、合計4ページだけHDD

注1）B+treeはインデックスの実装方式の主流。B-treeとも呼ばれるが、厳密にはB-treeを実用的に扱いやすく改良したものがB+tree。

▼図3　大きなテーブルの検索（インデックスあり）





を読めば検索が完了します。インデックスなしの場合は999ページ読まなければならなかったのに比べると圧倒的な差ですね！」と大道君。ここまでは完璧。

インデックスが役に立たないケースとは?

「じゃ、インデックスが役に立たないときっていうのはないんかな？」と質問してみました。

「えっ、それは……1つあるのは、テーブルがすごく小さいときは必要ない……ですよね？」

確かに、会員名簿が1,000件しかなくて全データが数ページ以下に収まってしまうようならインデックスによる検索速度改善はほとんど期待できず、逆に追加／更新時の負荷が増えるだけです。

「そのとおり！　で、ほかには？　検索の条件とかデータの分散具合によってはインデックスが役に立たないことがあるんだけど、どういうときやと思う？」

ここまでしくみを理解しているなら、よ〜く考えれば、インデックスが役に立たない条件に気がつくはずです。すぐに誰かに正解を聞こうとするのではなく、自分で考えるという習慣をつけてほしい。それがトラブルシューティングのときに生きてきます。

「検索の条件とか……」

「いつもいつもフルネームで検索できるとは限らんやろ？」

「……あ、そうか！　後方一致検索はできないんですね？」

「それや！」

図3のB+tree構造を見てもらうとおわかりのように、データを頭から照合していくため、たとえばlike "%ムラ"のように後方一致や中間一致で検索しようとするとインデックスは使われません。

「前方一致だったら使えるのか……あ、待てよ……like "サ%"みたいな検索をすると……」

おおっ、これはひょっとすると、と思っているうちに大道君、**図4**を書き出しました。

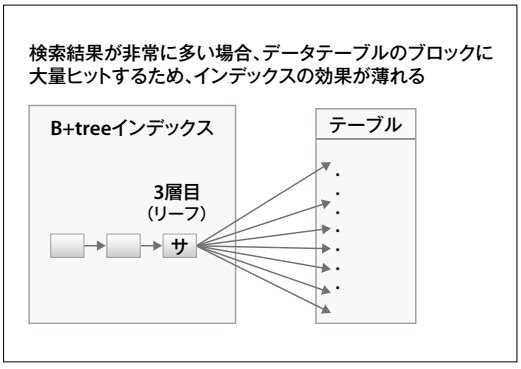
「こういうふうに検索結果がすごく多いと、結局どのページも読まなきゃいけなくなる、みたいなことに……」

「おお、そうなんよ。"佐藤"とか日本人にめちゃ多い苗字で前方一致検索とかやった場合やね。こういう場合は検索ヒット数が多くなるんで、読まなきゃいけないページが増えて、やっぱりインデックスの効果が薄れる。あとは、否定条件で検索した場合もインデックスは効かない」

「つまりあれですか、検索結果が少なければ少ないほど、インデックスの効果が大きいってことですよね？」

「ええとこに気いついた！　それや！」

▼図4　検索結果が非常に多い場合





カーディナリティの高いデータ項目に注目せよ

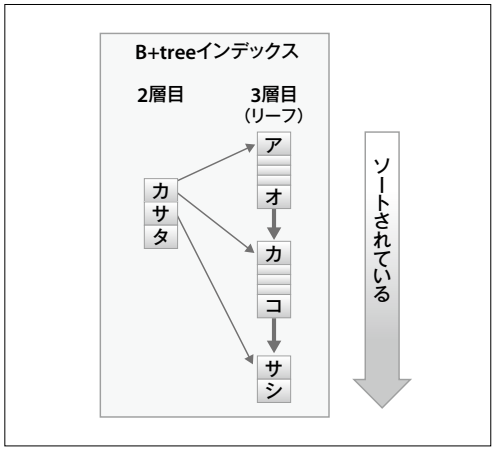
実はこれはインデックスを考える場合の非常に重要な観点で、「カーディナリティ」という用語で呼ばれています。この用語がインデックスについて使われる場合、簡単に言うとデータが分散していればいるほど「カーディナリティが高い」といって、インデックスを作るのに向いています。

たとえば「性別」は通常2種類しかなく、もし「不明」や「その他」を加えたとしても数種類しかありません。「都道府県」も47しかありませんので、カーディナリティは原理的に低くなります。一方、姓名、顧客番号、商品コード、受注日時のような項目はカーディナリティが高いため、インデックスによる検索性能向上効果が高くなります。

「あと1つ補足しておくと、インデックス末端のリーフはソートされてつながってるんよ」

図3からインデックスの2、3層目だけを簡略化して書いた**図5**にそのイメージを載せておきました。実際のRDBではリーフ同士が**図5**のオ→カ、コ→サの部分にある太い実線のようなポインタでつながっていて、リーフ部分だけをたどるとソートされた状態になっています。このためたとえば「先月1日の9:00から12:00までの注文を検索する」といった上下限範囲指定の検索も効率よくできます。

▼図5　インデックスの「リーフ」はソートされている



「しくみ」がわかっていないと真の応用は利かない

「ははあ……しくみがわかってると、いろいろ考えられるものなんですね」

「そう、そこが大事なんよ。しくみがわかってれば、実際のデータでどんなふうに動くかイメージわくやろ？　たとえば『性別』にインデックス張っても無駄やって、**図3**がイメージできれば一発でわかるし、部分一致検索したらどうなるかも類推できたやろ？」

「そうですね！　おもしろいです！」

本稿では最初に**図1**のような「状況→問題→効果→用法←しくみ」のフレームワークを紹介しました。実は、「こういう効果を上げるためには、この用法を使えば良い」という、用法レベルの知識を仕入れて満足してしまい、「しくみ」を理解しようとしないエンジニアが意外に多いことを私は危惧しています。

その用法がなぜ有効なのか、という「しくみ」レベルをわかっていないと、真の応用は利きません。しくみをわかっていれば、今回大道君が途中で「あ、待てよ……」と気がついたように、「しくみ」に合わない状況に対して「カンが働く」ようになります。「用法」を暗記しただけでは絶対に不可能なことです。

「わかりました！」

「おお！　でもな、まだまだこれからやで？　実は『削除フラグ』みたいな、普通に考えるとインデックスに向かない項目にもインデックスを張る手法もあるし、前方一致検索に向かないデータとか、文字数の多いテキストにインデックス張るときはどうするとか、細かい手法がまたいろいろあってな。インデックスの方式だってB+treeだけやないし」

「わかりました！　もっと勉強します！」

という大道君の返事はいつになく頼もしいものでした。SD

presented by
Japan Android Group
http://www.
android-group.jp/

コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

第7回 新しい3DグラフィクスAPI Vulkanを使ってみる

Androidは世界で出荷される8割のスマートフォンに搭載※される、事実上スマートフォンの標準OSです。このOS上で動くアプリケーションを開発することは、自分のアイデアを世界に発信し、最も多くのスマートフォン上で動かしてもらえる可能性がある行為です。このAndroidの最新情報と、それを取り巻くコミュニティを知って、魅力的な開発をはじめませんか？

※IDC Worldwide Mobile Phone Tracker, August 7, 2013

高橋 憲一
（たかはし けんいち）
㈱カブク
ソフトウェアエンジニア
Twitter @ken1_taka

Vulkanとは

Androidではその最初のバージョンから3Dグラフィクスを描画するためのAPIとしてOpenGL ESが使われてきました。そのOpenGL ESをはじめとした、特定のプラットフォームに依存しないAPIを策定してきたKhronosグループが新たな3Dグラフィクス APIとして出したものがVulkanです。2016年2月にVulkan 1.0の仕様がリリースされ、最近になってドライバの対応が進み、LinuxやWindows、そしてAndroidでも試すことができるようになってきました。

Vulkanが生まれた背景としては、AppleのMetal、AMDのMantle、MicrosoftのDirectX 12など、ここ数年のグラフィクスAPIの潮流としてよりローレベルなもの、ドライバ層が薄いものになっていっているというものがあります。その目的はダイレクトにGPUにアクセスするようにすることで、オーバーヘッドを下げてグラフィクス描画のパフォーマンスを上げていくためです。広く使われているOpenGLとOpenGL ES（以下、区別の必要がない場合はまとめてOpenGLとします）を策定してきたKhronosが満を持してそこに対応してきたのがVulkanというわけです。

AndroidではAndroid N Developer Preview 2から対応しており、本記事執筆時点でNexus 5X、Nexus 6P、Nexus Playerで試すことができます。まだ実行できる環境は限定的ではありますが、本記事ではこの新しいAPIを先取りして試してみたいと思います。

Vulkanの利点

統合されたAPI

PCなどデスクトップ向けであるOpenGLとモバイルなどの組込み向けであるOpenGL ESというような区別はなく、1つの統合されたAPIセットとなっています。

マルチプラットフォーム

特定のプラットフォーム専用ではなく、Androidのみならず LinuxやWindowsでも使用できます（同じ志向のMetalはAppleのみ、DirectX 12はWindowsのみとなっています）。

マルチスレッドに対応

複数のコマンドバッファを生成して別々のスレッドから制御することができます（OpenGLでは生成したコンテキスト上で動作する必要があり、単一のスレッドからしかそのコンテキストに命令を送れませんでした）。

細かいメモリ割り当ての制御

頂点座標やテクスチャなどに使うGPU側のメモリの割り当てをアプリケーションから細かく制御可能です。

GPUを活用した計算処理

グラフィクスだけでなく計算用途のパイプラインも生成できるようになっています。GPUを活用した高速計算処理にも使えます。

必要なもの

Vulkanに対応した端末と最新版のAndroid Studio(本稿執筆時点では2.1.1)、NDK r12以上が必要です。本稿執筆時点でVulkanに対応している端末は前述のとおりNexus 5X、Nexus 6P、Nexus Playerの3つ。まだAndroid N Developer Previewを入れていない場合はAndroid Beta Programのサイト[注1]から端末を登録してAndroidベータ版プログラムに参加します。自分でOSイメージを焼くことなく、OTA[注2]で取得してインストールできます(まだPreview段階のものですので、もしメインで使用している端末に入れる場合は自己責任でお願いします)。

NDK r12はまだベータ段階ですが、Vulkan Setupのサイト[注3]を参考にダウンロードしてセットアップします。

また、Googleから出されているチュートリアル[注4]とサンプル[注5]もあり、どちらもこの環境があれば試すことができます。サンプルを使ってVulkanを使ったコードがビルド、実行可能な状態になっているか確かめておきましょう。

注1) https://g.co/androidbeta
注2) Over The Airの略で、無線通信を通して取得することを言います。
注3) https://developer.android.com/ndk/guides/graphics/getting-started.html
注4) https://github.com/googlesamples/android-vulkan-tutorials
注5) https://github.com/googlesamples/vulkan-basic-samples

Vulkanを使ったコードの構成

現時点ではVulkanはJavaから使うことはできないので、NDKを使ってC、もしくはC++で書くことが必須となります。ここからの説明で使用するコード[注6]は前出のチュートリアルとサンプルを参考に、3Dグラフィクスではおなじみのteapot[注7]のモデルの描画、回転の操作ができるようにしたものです。実行すると**図1**のようになります。

Android Studioのプロジェクトでapp/jniの下を開くと、AndroidMainのcppとhpp、VulkanMainのcppとhpp、VulkanDeviceのcppとhppのファイルがそれぞれあります。AndroidMain.cppの中でWindow生成時にはInitVulkan()、Windowがクローズされるときには DeleteVulkan()、毎フレームの中ではVulkanDrawFrame()が呼び出されるようになっています。VulkanDevice.cppの中で具体的なVulkanのコードを記述しています。

大まかな手順は次のようになります。

注6) https://github.com/ktaka/VulkanTeapotにあります。
注7) 本誌2011年8月号で、NDKでOpenGL ESを使うことに関して書いたときもteapotの描画でした。

▼図1 サンプル実行画面



1. インスタンスの生成
2. デバイスの取得
3. スワップチェインの作成
4. コマンドの作成
5. 頂点バッファの作成
6. シェーダーのロード
7. シェーダーのユニフォームへ値を渡す
8. レンダーパスの作成
9. パイプラインの作成
10. レンダリング
11. タッチイベントからの値を回転に変換してユニフォームへ渡す

VulkanのAPIの呼び出しは**リスト1**でのdevice取得に見られるように、VkDeviceQueueCreateInfoとVkDeviceCreateInfoの構造体に必要な値をセットしてvkCreateDevice()に渡し、最初の引数では主体となるオブジェクト(レシーバに相当します)、最後の引数に格納先のアドレスを渡してそこに結果が入ってくることで必要なオブジェクトを得る、という手順が多いです。この先はここで得たdeviceオブジェクトを使っていきます。

シェーダーのロード

頂点ごとの処理をするためのバーテックス

▼リスト1　VulkanDevice.cpp(デバイスの取得)

```
float queue_priorities[1] = {0.0};
VkDeviceQueueCreateInfo queue_info{  // queue生成のための構造体
        .sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO,
        .pNext = NULL,
        .queueCount = 1,
        .pQueuePriorities = queue_priorities,
        .queueFamilyIndex = graphics_queue_family_index,
};
VkDeviceCreateInfo device_info{  // device生成のための構造体
        .sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO,
        .pNext = NULL,
        .queueCreateInfoCount = 1,  // queue生成のための情報定義数
        .pQueueCreateInfos = &queue_info,  // queue生成のための情報のアドレス
        .pEnabledFeatures = NULL,
};
device_info.enabledLayerCount = device_layer_names.size();
device_info.ppEnabledLayerNames =
        device_info.enabledLayerCount ? device_layer_names.data() : NULL;
device_info.enabledExtensionCount = device_extension_names.size();
device_info.ppEnabledExtensionNames =
        device_info.enabledExtensionCount ? device_extension_names.data() : NULL;
VkResult res;
res = vkCreateDevice(gpus[0], &device_info, NULL, &device_);
```

COLUMN　NDK

本誌の2011年6月号から8月号にかけて「Androidエンジニアからの招待状」という連載の中で「NDK＋GingerbreadでネイティブアプリNDK開発」というタイトルでNDKによるアプリ開発について書かせていただきました。当時は、端末の処理能力も上がり、Javaで書いたアプリの実行効率も良くなることで、いずれはNDKを使ってC/C++でネイティブなアプリを書くこともなくなるのではないか……ということも言われていました。ところが5年が経過した今もなくなる気配はありません。ハードウェアが速くなれば、たとえばグラフィクス処理であればよりたくさんのオブジェクトの描画、ライティングなどの表現力の向上を求めて、その速くなった分さらなる高みを求めるあくなきパフォーマンスの追求は留まることがないようです。筆者もまたこうしてNDKを使った記事を書くことができるとは予想しておりませんでした。

シェーダーと、ピクセルごとの処理をするためのフラグメントシェーダーは、OpenGLと同様にGLSLを使用して記述します。ただしOpenGLのようにテキストデータとしてアプリケーションで保持しておいて実行時にコンパイルするのではなく、中間言語であるSPIR-Vにコンパイルしてバイナリとして保持しておきます。中間言語であるためSPIR-Vもさらに実行形式に変換が必要ですが、これまでよりもサイズ、実行時コンパイルの観点で効率が良くなります[注8]。

シェーダーに値を渡す

C++で書いたコードからシェーダーに値を渡すには**リスト2**のようにします。ここでは**リスト3**のバーテックスシェーダー(頂点ごとに実行されるシェーダー)にmvpという名前で変換行列を渡しており、この行列で頂点座標を変換することによりモデル(teapot)回転や視点の変換を行います。このような変数をuniform(ユニフォーム)と言います。

リスト2でやっていることは、glm::mat4という4×4の行列を格納する型の変数に計算した変換行列を格納し、glm::mat4のサイズを指定してvkCreateBufferでバッファを生成、vkAllocateMemoryでGPU側のメモリを割り当てて、CPUから値を渡せるようにvkMapMemoryでメインメモリ側のアドレスにマップしています。そのアドレスにmvpの内容をコピーし、メインメモリへのマップを解除して最後にvkBindBufferMemoryでオフセット0(4番目の

注8) これまでも拡張機能としてコンパイル後の形式を持つことができる環境はありましたが、特定のGPU向けのバイナリとなり、多数の機種に対応する必要があるアプリケーションには適していませんでした。

▼リスト2 VulkanDevice.cpp(ユニフォームに値を渡す)

```
glm::mat4 mvp = clip * projection * view * model;  // 変換行列の計算

VkBufferCreateInfo buf_info{  // バッファ生成のための構造体
        .sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO,
        .pNext = NULL,
        .usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT,  // ユニフォームとして使うことを指定
        .size = sizeof(mvp),  // mvpのサイズ分の確保を指定
        .queueFamilyIndexCount = 0,
        .pQueueFamilyIndices = NULL,
        .sharingMode = VK_SHARING_MODE_EXCLUSIVE,
        .flags = 0,
};
res = vkCreateBuffer(device_, &buf_info, NULL, &uniform_data.buf);
VkMemoryRequirements mem_reqs;
vkGetBufferMemoryRequirements(device_, uniform_data.buf,
                              &mem_reqs);
VkMemoryAllocateInfo alloc_info{
        .sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,
        .pNext = NULL,
        .memoryTypeIndex = 0,
};
alloc_info.allocationSize = mem_reqs.size;
pass = memory_type_from_properties(mem_reqs.memoryTypeBits,
                                   VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT,
                                   &alloc_info.memoryTypeIndex);
res = vkAllocateMemory(device_, &alloc_info, NULL,  // GPU側のメモリを割り当てる
                       &(uniform_data.mem));
uint8_t *pData;
res = vkMapMemory(device_, uniform_data.mem, 0, mem_reqs.size, 0,
                  (void **)&pData);  // メインメモリ側のアドレスにマッピング
memcpy(pData, &mvp, sizeof(mvp));  // mvpの内容をコピー
vkUnmapMemory(device_, uniform_data.mem);
res = vkBindBufferMemory(device_, uniform_data.buf, uniform_data.mem, 0);
```

▼リスト3　shape.vert（バーテックスシェーダー）

```
#version 400
#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_shading_language_420pack : enable
layout (std140, binding = 0) uniform bufferVals {
    mat4 mvp;  // 変換行列（モデル * ビュー * プロジェクション）
} myBufferVals;
layout (location = 0) in vec3 pos;        // 入力：頂点座標
layout (location = 1) in vec3 inNormal;   // 入力：法線ベクトル
layout (location = 0) out vec3 outNormal; // 出力：法線ベクトル
layout (location = 1) out vec4 position;  // 出力：頂点座標
out gl_PerVertex {
    vec4 gl_Position;
};
void main() {
   outNormal = inNormal;
   gl_Position = myBufferVals.mvp * vec4(pos, 1); // 頂点座標を変換
   position = gl_Position;
}
```

引数）にバインドすることでシェーダーのユニフォームと関連付けています。

パイプライン

このシェーダーはパイプラインに結びつけることで実際の描画に使われるようになります（**リスト4**）。

リスト4でやっていることは、バーテックス、フラグメントの両シェーダーをVkPipelineShaderStageCreateInfo構造体（シェーダーステージ）の2つの配列で定義し、そのシェーダーステージをメンバに代入したVkGraphicsPipelineCreateInfo構造体の内容を渡してvkCreateGraphicsPipelinesを呼び出してグラフィクスパイプラインを生成しています。

最後に

いかがでしょうか。すべてのコードの解説は今回の誌面ではできませんでしたが、ここまで見ていただいたように、これまでOpenGLのドライバがやっていたことをアプリケーション側で実装していくようになっており、OpenGLと比較するとダイレクトにGPUを制御している感があります。反面、アプリケーションがやるべき内容は非常に多くなっていることがわかります。

「なぜ手間を増やしてまでやる必要があるのか」という疑問をお持ちの方もいるかもしれません。それに対する答えは「より高みのパフォーマンスを目指して」というものになります。Khronosのプレゼン資料などを見ても、すべての人がVulkanを使うわけではなく、OpenGL、OpenGL ESはなくならず継続して開発が続けられるとありますので、用途によって使い分けることができます。また、いずれUnityやUnreal Engineなどのエンジンが Vulkanに対応することで、多くの開発者もそのパフォーマンス向上の恩恵にあずかることができるようになるはずです。その一方で、こうした下回りの技術を知っておくこともエンジニアとして大切なことだと思います。SD

COLUMN

Android Bazzar and Conference 2016 Autumn

恒例のAndroid Bazzar and Conferenceが2016年10月15日に開催予定です。詳細乞うご期待！
http://abc.android-group.jp/2016a/

▼リスト4　パイプラインの生成とシェーダーとの関連付け

```
VkPipelineShaderStageCreateInfo shaderStages[2] {  // シェーダーステージ定義のための構造体
        {  // バーテックスシェーダー
                .sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO,
                .pNext = nullptr,
                .stage = VK_SHADER_STAGE_VERTEX_BIT,  // バーテックス処理のステージを指定
                .module = vertexShader,  // シェーダーをロードして得られたハンドル
                .pSpecializationInfo = nullptr,
                .flags = 0,
                .pName = "main",
        },
        { // フラグメントシェーダー
                .sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO,
                .pNext = nullptr,
                .stage = VK_SHADER_STAGE_FRAGMENT_BIT,  // フラグメント処理のステージを指定
                .module = fragmentShader,  // シェーダーをロードして得られたハンドル
                .pSpecializationInfo = nullptr,
                .flags = 0,
                .pName = "main",
        }
};
VkGraphicsPipelineCreateInfo pipelineInfo{
        .sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO,
        .pNext = NULL,
        .layout = pipelineLayout,
        .basePipelineHandle = VK_NULL_HANDLE,
        .basePipelineIndex = 0,
        .flags = 0,
        .pVertexInputState = &vi,
        .pInputAssemblyState = &ia,
        .pRasterizationState = &rs,
        .pColorBlendState = &cb,
        .pTessellationState = NULL,
        .pMultisampleState = &ms,
        .pDynamicState = &dynamicState,
        .pViewportState = &vp,
        .pDepthStencilState = &ds,
        .pStages = shaderStages, // シェーダーステージ
        .stageCount = 2,  // シェーダーステージの定義数（バーテックスとフラグメント）
        .renderPass = render_pass,
        .subpass = 0,
};
res = vkCreateGraphicsPipelines(device_, pipelineCache, 1,&pipelineInfo, NULL, &pipeline);
```

COLUMN コミュニティ活動

筆者の最近のコミュニティ活動の中心はAndroidアプリ開発の講師をすることです。2014年から2015年にかけて行われたTechInstituteや、本誌の2016年1月号でも紹介させていただいた東北TECH道場がその活動の場です。そういった場で実際に学んだ学生が日本Androidの会学生部でABCのイベントアプリを作るようになる、といった成長の姿を見るのは講師冥利につきるというものです。このようなことにかかわるようになったきっかけは日本Androidの会に参加したことがルーツとなっており、コミュニティに参加する、ただ参加するだけでなく発信していく側になっていくという一歩を踏み出したことに始まります。

一歩進んだ使い方のためのイロハ

Vimの細道

第9回

mattn
twitter:@mattn_jp

カーソルを自在に操る

テキストエディタにおいて、「カーソルの移動」は編集スピードに直結する重要な要素。今回は、そのカーソルの移動をスピードアップする方法を解説します。/検索による移動、fFtTによる移動、fFtTを強化する3つのプラグインを紹介します。

カーソル移動を考える

テキストエディタは画面にテキストをレンダリングし、カーソルという目印を基準にしてタイプした文字を挿入したり、その目印から文字を削除したりします。ほとんどの操作がカーソルという基準に対して行われます。

テキストを編集するためには、何らかの方法で目的の位置へカーソルを移動する必要があります。ですのでカーソルを目的の位置に移動するスピードが速ければ速いほど、テキスト編集の効率が上がるのです。一般的なテキストエディタで可能なカーソル移動方法は、おおよそ次のとおりです。

・上下左右
・行頭、行末
・ファイルの先頭、ファイルの末尾
・次の単語、前の単語
・検索でマッチした位置

一般的なテキストエディタでも、ある程度のカスタマイズ機能によって目的の位置へカーソルを移動する手数を減らせます。Vimには、それとは比べものにならないほどに特別なカーソルの移動方法があり、ユーザは用途によって使い分けています。

/によるカーソル移動

たとえば「/検索」。vi/Vimでの検索は/をタイプして検索を始め、検索にマッチするとカーソルが移動します。もちろんこれは通常のテキストエディタの検索でも同じなのですが、Vimではこの検索パターンを工夫することで、マッチしたどの部分にカーソルを移動させるかを操作できます。

```
/something\zs
```

\zsはマッチ開始位置を表します。つまり、somethingが先行する0文字めの検索となります。よってこの場合はsomethingを検索し、マッチした文字の最後の部分にカーソルが移動します。また、\zsを途中に入れることもできるので、

```
/some\zsthing
```

のように使うこともできます。ローテクですが意外と効きます。とくに検索語を入力し終わったあとで、「あ、この検索語の直後に移動したい」といった感じに後付けできるのが良いです。

fFtTとモーション

fFtTはほかの一般的なテキストエディタには

▼表1　fFtTの操作

モーション	説明
f{char}	右方向に{char}を検索
F{char}	左方向に{char}を検索
t{char}	右方向に{char}の1つ前を検索
T{char}	左方向に{char}の1つ前を検索

▼表2　clever-fの操作

キーシーケンス	方向	進む	戻る
f{char}	前	f	F
F{char}	後	f	F
t{char}	前	t	T
T{char}	後	t	T

ない、一風変わったカーソルの移動方法です。たとえばfでは、そのあとにタイプした1文字を検索してカーソルを移動します(**表1**)。

fFtTは単なる移動方法ではなく、モーションとして動作します。dfhとタイプすれば現在の位置からhという文字が見つかった位置までを削除します。

①dがタイプされるとオペレータペンディングとなる
②fFtTがタイプされると{char}の入力待ちとなる
③f{char}がdのモーションとして扱われ、現在の位置から検索した位置までが削除される

このオペレータペンディングをomapにてカスタマイズすることで、基本の動作と異なるモーションを生み出せます。そのモーションをうまくカスタマイズしているVimプラグインを3つ紹介します。



clever-f[注1]はfFtTをよりスマートに実行できるプラグインです。fFtTによるジャンプでよく起こるのが「目的の位置にジャンプするにはaをタイプすれば良いと思っていたのに、実はもっと手前にaがあったので足りなかった」といった目測ミスです。わざわざ2faとタイプしなおす人もいれば、再度faをタイプして前に進む人もいるでしょう。オペレータのモーションとして扱っている場合、再度入力をやりなおさなければなりません。こういうシーンを他人に見られるとちょっと恥ずかしかったりもします。動画でVimの操作を見せるなら完全にNGシーンです。

clever-fではfをタイプしたあとに検索したい文字をタイプし、「足りてない」と思ったら再度fをタイプすることで、もう1つ向こうの文字までジャンプできます。ですので、たとえばwという文字を検索してジャンプしたい場合、fwfffffのように目的の位置までfをタイプし続ければ良いのです。しかし時には、fをタイプし過ぎて通り過ぎてしまうこともあります。そんな場合はFをタイプすれば、1つ前の検索文字に戻れます。同様に、tで後方にジャンプしている最中に「行き過ぎた」あと思ったら、Tをタイプして戻せます(**表2**)。

さらにclever-fは、migemoを使うことにより日本語の検索ができます。migemoとはローマ字から日本語をインクリメンタル検索するためのプロダクトで、C言語の実装C/Migemoを、お馴染み香り屋のKoRoNさんが開発しています。C/Migemoのインストール方法は次のとおり。

```
$ git clone https://github.com/koron/cmigemo
$ cd cmigemo
$ make gcc-install
```

各OSおよびコンパイラにより、makeに渡す引数が異なります。詳しくはC/Migemoのdoc/README_j.txtを参照してください。

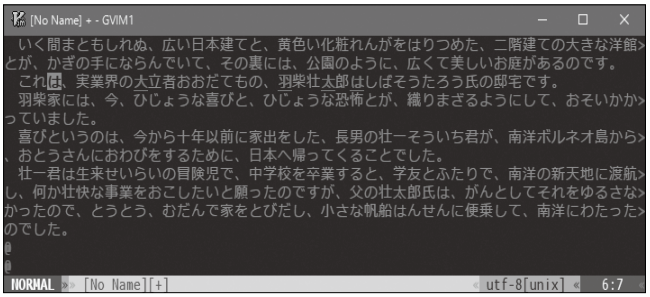
cmigemoコマンドをインストールして、

```
let g:clever_f_use_migemo = 1
```

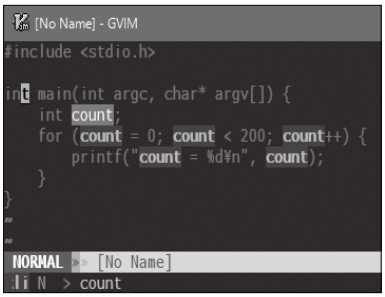
と設定を行うことで、clever-fにてマルチバイト文字に対してのfFtT拡張が行えます。この場合、タイプする検索文字は日本語にマッチさせるためのローマ字の先頭だけで良いので、たと

注1) URL https://github.com/rhysd/clever-f.vim

▼図1　fhでマルチバイト文字を検索



▼図2　smallsを起動、countとタイプ



えば「はひふへほ」の子音に対してはfhで検索できます(**図1**)。次の候補がハイライト表示されるので、とても見やすく便利です。



smalls[注2]はclever-fとは違ったカーソル移動方法を提供します。clever-fの場合はfに続く1文字を検索対象としていますが、smallsでは複数の文字を検索パターンとして扱えます。smallsはデフォルトではキーマッピングを提供しないので、**リスト1**の設定をvimrcに追加します。

smallsには2つのモードがあります。カーソルの移動を行うcli-modeと、モーションのあとに続けてオペレータを入力するexcursionモードです。

まずはcli-mode。cli-modeは次の2つのアクションで実行します。

❶パターンの入力
❷マッチの選択

まずパターンの入力を行います。**リスト1**のキーマッピングの例では、sをタイプするとプロンプトが表示されるので続けてパターンをタイプします(**図2**)。

パターンのあとに続けてTABをタイプするとインクリメンタルにマッチしたcountのハイライトが移動します。移動したい場所にハイライトが移ったら;をタイプして移動を確定します。

▼リスト1　smallsのキーマッピング

```
" Normal モードのsで発動
nmap s <Plug>(smalls)
" visual モードやoperatorモードからも
" 呼び出したい場合は以下も設定する。
omap s <Plug>(smalls)
xmap s <Plug>(smalls)
```

移動先がハイライトされるので、hlsearchを設定して/countの検索をnnnとしてたどっていくよりもわかりやすいです。またhlsearchは検索後も有効ですが、smallsの動作は検索時のみハイライトが有効になりますので、無駄にハイライトされて見づらくなることもありません。

また、TABでハイライトを動かさずに;をタイプすると移動先の先頭にマーカーが置かれ、EasyMotion(後述)と同じような見栄えになります。

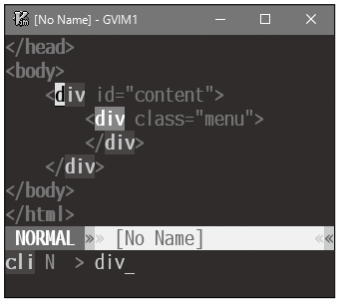
次にexcursionモード。cli-modeとパターンの指定までは同じで、そのあとオペレータをタイプします。つまり現在の位置からsmallsで移動したカーソル位置までの範囲に対して続くオペレータを適用します。Tabでインデントされた以下のHTMLがあるとします。

```
<body>
  <div id="content">
    <div class="menu">
    </div>
  </div>
</body>
```

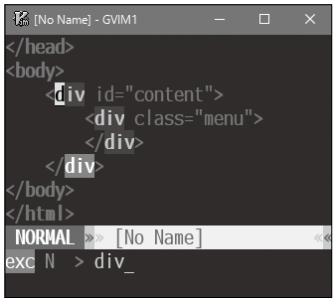
最初のdivにカーソルがある状態からsをタイプしてsmallsを起動します(**図3**)。続いてパターンとしてdevを指定し、TABを2回押すと**図4**

注2) URL https://github.com/t9md/vim-smalls

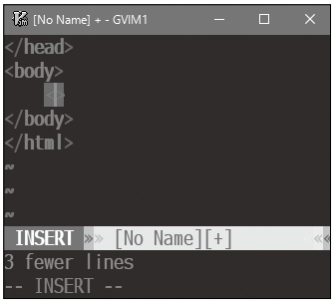
▼図3　smallsを起動



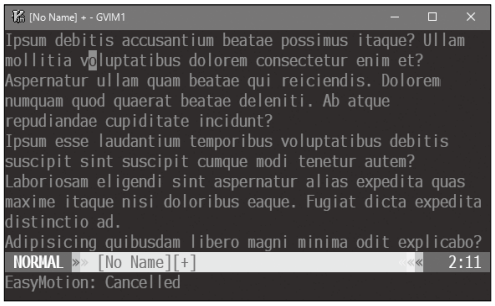
▼図4　TABを2回



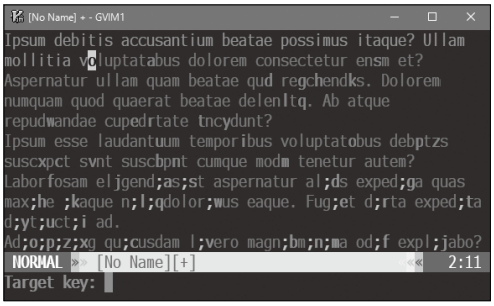
▼図5　dで削除



▼図6　EasyMotionを起動



▼図7　移動先の候補がハイライトされる



のようにカーソルが移動します。最後にdをタイプすると、その範囲が削除されます(**図5**)。

clever-fはfFtTによる移動の拡張、smallsは/検索の拡張と言えます。

EasyMotion

カーソル移動系プラグインの中で一番有名なのがEasyMotion[注3]です。EasyMotionは前者2つとは異なり、ユーザが期待するカーソルの移動先を、EasyMotionが提案する複数の英字で表現します。EasyMotionを起動する方法は**表3**のとおり。

<leader><leader>f(<leader>が未設定の場合\\f)をタイプするとプロンプトが表示され、検索する文字を聞かれます(**図6**)。この例ではiをタイプすると、近辺は1文字、ちょっと遠いところは2ストロークをタイプすることで目的の場所へ一瞬で移動できるようになります(**図7**)。またEasyMotionも、clever-fと同様に次の設定でmigemoを使えます。

```
let g:EasyMotion_use_migemo = 1
```

なんと言ってもEasyMotionのすごいところは、カスタマイズ性に富んでいるところです。

▼表3　EasyMotionの起動方法

マッピング	説明
<Leader>f{char}	右方向に{char}を検索
<Leader>F{char}	左方向に{char}を検索
<Leader>t{char}	右方向に{char}の1つ前を検索
<Leader>T{char}	左方向に{char}の1つ前を検索
<Leader>w	前方に単語(word)の始まりを検索
<Leader>W	前方に単語(WORD)の始まりを検索
<Leader>b	後方に単語(word)の始まりを検索
<Leader>B	後方に単語(WORD)の始まりを検索
<Leader>e	前方に単語(word)の終わりを検索
<Leader>E	前方に単語(WORD)の終わりを検索
<Leader>ge	後方に単語(word)の終わりを検索
<Leader>gE	後方に単語(word)の終わりを検索
<Leader>j	下方向に行を検索
<Leader>k	上方向に行を検索
<Leader>n	最後に行った検索文字で前方に検索
<Leader>N	最後に行った検索文字で後方に検索
<Leader>s	前方後方の両方向に{char}を検索

注3) URL https://github.com/easymotion/vim-easymotion

▼リスト2　テキスト内の単語を英文から日本語に変換するプラグイン

```
function s:Translate(mode) range abort
  if a:mode ==# 'v'
    let save_regcont = @"
    let save_regtype = getregtype('"')
    silent! normal! gvygv
    let text = substitute(@", '\n\+', ' ', 'g')
    call setreg('"', save_regcont, save_regtype)
  else
    let text = expand('<cword>')
  endif
  let r = substitute(system("excitetranslate", text), "[\r\n]", "", "g")
  if a:mode ==# 'n'
    " ノーマルモードならばciwで
    silent! exe "normal!" 'ciw' . r
  else
    " ビジュアルモードならばcで
    silent! exe "normal!" 'c' . r
  endif
endfunction

nnoremap <Plug>(excitetranslate) :<C-u>call <SID>Translate('n')<cr>
xnoremap <Plug>(excitetranslate) :<C-u>call <SID>Translate('v')<cr>
nmap ,E <Plug>(excitetranslate)
xmap ,E <Plug>(excitetranslate)
```

EasyMotionが提供している機能のほとんどをカスタマイズできますし、逆にVimが提供するモーションをオーバーライドするための設定をほとんど持ち合わせています。

たとえば/検索をEasyMotionの挙動に置き換えるには、次のように設定します。

```
set nohlsearch
map  / <Plug>(easymotion-sn)
omap / <Plug>(easymotion-tn)
map  n <Plug>(easymotion-next)
map  N <Plug>(easymotion-prev)
```

/検索でマッチした文字列が複数ある場合には、EasyMotionのストローク入力によって瞬時に目的の場所へ移動できます。

リピート対応

これらカーソル移動系プラグインのいくつかはomapをサポートしていますが、実は標準機能だけでは.によるリピートには対応できません。.によるリピートはキーの再生が行われるのですが、プラグイン自らがVimに対してカーソルの移動方法を教える手段がありません。

通常Vimのカーソル移動方法は次のようにVimが提供する移動方法のみとなります。

・単語を3回ジャンプしたのか
・それともhjklで移動したのか
・検索文字でジャンプしたのか

しかしプラグインでカーソル位置を移動する場合、Vimは.によるリピートをどう再生して良いのかわからないのです。そこで使われるのがvim-repeat[注4]というプラグインです。

vim-repeatはどちらかというと、プラグイン作者のためのプラグインで、.と<C-R>に対して特別な処理をさせるためのプラグインです。カーソル移動系プラグインだけでなく、多くのプラグインが採用しています。

vim-repeatのしくみは簡単です。.や<C-R>が押された際に実行してほしい処理を、プラグイン側がキーシーケンスや<Plug>マッピングとして登録するのです。

注4）URL https://github.com/tpope/vim-repeat

たとえばテキスト内の単語を英文から日本語に変換するプラグインを作ってみます。**リスト2**は、カーソル直下の単語(英単語)を翻訳して置き換えるプラグインです。このプラグインの動作には、excitetranslate[注5]というプログラムが必要になります。インストールしたら適当な英文のファイルを開き、単語の上にカーソルを移動したあと、,E(カンマのあとにE)をタイプしてみてください。すると単語(英語)が日本語に翻訳され置き換えられます。

これは通常の場合、とてもよく動きます。しかし別の単語に移動して . とタイプするとどうなるでしょう。当然みなさんは「別の単語で実行したのだから別の翻訳がされてほしい」と思うでしょう。しかし、実際は何も動作しません。プラグインによっては、最初に翻訳された内容に置き換えられてしまうかもしれません。

これをvim-repeatを使って直すには**リスト3**のように修正します。関数s:Translateにたった1行足すだけです(下線部)。. や<C-R>がタイプされたら何を実行するかをプラグイン側から登録しています。こうすることで . をタイプした際にvim-repeatが登録されたコマンドを実行してくれるので、別の単語で . をタイプしたとしても、正しく翻訳語に置き換えられるのです。

▼リスト3　リスト2をリピート再生できるよう修正

```
function s:Translate(mode) range abort
...略...
  if a:mode ==# 'n'
    " ノーマルモードならば ciw で
    silent! exe "normal!" 'ciw' . r
  else
    " ビジュアルモードならば c で
    silent! exe "normal!" 'c' . r
  endif
  " vim-repeat に登録する
  call repeat#set("¥<Plug>(excitetranslate)", v:count)
endfunction

...略...
```

まとめ

今回紹介した以外にもカーソル移動系プラグインがいくつかあり、それぞれユニークな視覚的効果を提供しています。いろいろ試してみるとおもしろいプラグインや便利なプラグインが見つかるかもしれません。また、誰も考えもしなかった別のアイデアが浮かぶかもしれません。ぜひ試してみてください。SD

注5) URL https://github.com/mattn/excitetranslate

Vim目線

True Color

Patch 7.4.1770によりVimがTrue Colorをサポートしました。これによりTrue Colorが出せる端末であればGUI版と同じ配色を出せるようになりました。Patch 7.4.1770時点では違うオプション名でしたが、最新版ではtermguicolorsというオプションを設定することで有効になります。

```
set termguicolors
```

ただし、このオプションはCUIにしか提供されないので、プラグインなどで参照する場合には、

```
if exists('&termguicolors') && &termguicolors
  " True Color の場合
endif
```

のように判定する必要があります。Vimではまれにこういったオプション名の変更がありますが、無理をして古いオプション名を参照せず、ある一定以下のバージョンにはそんなオプションは最初からなかったかのように扱ったほうが何かと便利です。

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer るびきち twitter@rubikitch http://rubikitch.com/

第27回 シェルコマンドを活用しよう(後編)

Emacsから使うシェルコマンド編、今回はコンパイルコマンドについて深掘りしていきます。M-x compileを実行したときに呼ばれるコマンドを設定するための、2つの方法と2つのパッケージを紹介します。

コンパイルをさらに便利にする

コンパイルコマンドたち

前回はシェルコマンドを扱うEmacsコマンドをたくさん紹介しました。M-x ifconfigやM-x run-pythonなどは特定のシェルコマンドをEmacs内で実行します。

一方で、ユーザがシェルコマンドを指定するタイプのコマンドもいくつか紹介しました。ベースとなっているのが、Emacs内でコンパイルを行うM-x compileです。このコマンドの重要な点は、コンパイルエラー行に簡単にジャンプできることです。*compilation*バッファでEnterを押しても良いですが、ウィンドウを切り替える手間がかかります。そこで、C-x `あるいはM-g M-n(next-error)を使うことで、ウィンドウを切り替えることなく次のエラーにジャンプできます。M-g M-p(previous-error)は前のエラーにジャンプします。

このエラー行ジャンプ機能は、スクリプトを実行するM-x executable-interpretでも使えます。Emacs内でgrepを実行するM-x grepでのマッチ行ジャンプも同様の操作ができます。それどころか、elispで書かれたバッファ内検索のM-x occurでさえも……。

M-x compileのデフォルト問題

M-x compileで実行するコンパイルコマンドは「make -k 」がデフォルトになっています。makeはメジャーですので多くの場合はこれで良いのですが、make以外のコンパイルプログラムが必要になることがあります。Rubyのrakeしかり、Pythonのsconsしかり、Javaのantしかり……。これらのプログラムによってコンパイルする場合、M-x compileを実行するときにわざわざ変更するのは面倒ですね。

もっとも、一度変更したあとにふたたびM-x compileを実行したときには変更後のコンパイルコマンドが使われるので、そこまで面倒には感じないかもしれません。

とはいえ、別のディレクトリに移動しても変更後のコンパイルコマンドが使われるので、ディレクトリ(プロジェクト)ごとにコンパイルコマンドを切り替える手間が生じます。複数のプロジェクトを抱えていて頻繁にコンパイルする人にとっては頭痛の種です。

ファイル・ディレクトリローカル変数を使う

ファイルローカル変数を使う

Emacs標準機能の範囲でのこの問題の解決方

▼リスト1　compile-commandの設定①

```
/* -*- compile-command: "gcc -o hello hello.c" -*- */
```

▼リスト2　compile-commandの設定②

```
/* Local Variables: */
/* compile-command: "gcc -o hello hello.c" */
/* End: */
```

法は、compile-commandをファイルローカル変数で設定することです。

たとえば、C言語で書かれた単一ソースコードhello.cをコンパイルする場合はmakeを使うまでもなく、直接gccで実行ファイルを生成すれば良いです。compile-commandを、"gcc -o hello hello.c"に設定すればいい話です。

ファイルローカル変数の設定方法は3つあります。

①ファイル先頭に-*-で囲んで宣言をする
②ファイル末尾にLocal variables:〜End:と囲んで宣言する
③M-x add-file-local-variableを使う

①、②の方法はプログラミング言語のコメントとして記述し、プログラミング言語側からは解釈されないようにします。プログラミング言語側からは無視されても、Emacs側には意味があります。

①では1行目に**リスト1**のように記述します。なお、先頭行に#!が含まれる場合は2行目でかまいません。

②ではファイルの末尾で**リスト2**のように記述します。

③のM-x add-file-local-variableでは、対話的に②の形式のコメントを作成します。実行すると「Add file-local variable: 」というプロンプトが出て変数名をきいてくるのでcompile-commandを指定します。

次に、「Add compile-command with value:」というプロンプトが出て値をきいてくるのでダブルクォーテーションありで、"gcc -o hello hello.c"を指定します。

どの方法もすぐにファイルローカル変数設定が反映されないので、C-x C-v RETと操作することで明示的に反映させる必要があります。そのあとにM-x compileを実行すると、設定が反映されていることがわかります。

ディレクトリローカル変数を使う

ファイルローカル変数だと、複数個のソースコードがある場合はすべてのファイルで設定する必要があります。それはとても面倒なうえ、変更時にはすべてのファイルに対して同じ変更を行う必要があります。DRY原則(Don't Repeat Yourself)――同じ情報を複数個所に書くな――に反し、保守性に問題があります。

そこで登場するのが、ファイルローカル変数の亜種であるディレクトリローカル変数です。ディレクトリローカル変数は、そのディレクトリおよびサブディレクトリ固有の変数を宣言するものです。メジャーモードごとにも設定できます。ディレクトリローカル変数は.dir-locals.elに記述するのですが、M-x add-dir-local-variableを使うと自動で設定されます。

たとえば、rakeでコンパイルするプロジェクトがある場合、そのプロジェクトのトップディレクトリにおいて、M-x add-dir-local-variableを実行します。すると、「Mode or subdirectory:」というプロンプトが出てメジャーモードかサブディレクトリをきいてくるのでnilを指定し、全ファイルを対象にします。次に、「Add directory-local variable:」というプロンプトが出て変数名をきいてくるのでcompile-commandを指定します。最後に、「Add compile-command with value:」というプロンプトが出て値をきいてくるので、ダブルクォーテーションありで"rake"を指定します。

設定後は.dir-locals.elが開かれて、**リスト3**の表示になるはずです。そのおかげで、正しくディレクトリローカル変数が設定されたことが

確認できます。もし、(compile-command . "rake")ではなくて(compile-command . rake)となっていたのならば、値の設定にダブルクォーテーションなしのrakeを入力したことになるので、M-x add-dir-local-variableを再実行してください。

ちなみに、M-x delete-dir-local-variableを実行すると、ディレクトリローカル変数を削除できます。

▼リスト3　M-x add-dir-local-variable設定後

```
;;; Directory Local Variables
;;; For more information see (info "(emacs) Directory Variables")

((nil
  (compile-command . "rake")))
```

ファイル名・メジャーモードに応じてコンパイルコマンドを自動決定する

M-x compileを拡張する

M-x compileは古くからあるので、それを拡張するパッケージも存在します。やはり一番の要求は、ファイル名やメジャーモードに応じてコンパイルコマンドを自動決定することではないでしょうか。ファイル・ディレクトリローカル変数でも細かく制御できますが、init.elに書きたい場合もあるでしょう。

smart-compile

そこで使いたいのが、昔から有名なsmart-compileパッケージです。M-x compileでのコンパイルコマンドを決定するcompile-commandの扱いを賢く(smart)します。次の機能があります。

- ファイル名(おもに拡張子)に応じてデフォルトのcompile-command変数を設定できる
- プロジェクトごとのcompile-commandを設定できる
- compile-commandをバッファローカルにする

実際には、次の処理をしてからM-x compileを実行します。

❶compile-commandがバッファローカルになっているときはそのまま
❷compile-commandをバッファローカル変数として設定する
❸Makefileが存在する場合はmakeを実行するか尋ねる
❹smart-compile-alist変数(後述)でファイル名・メジャーモードからcompile-commandを決定する
❺#!が含まれている場合はファイル名そのものをcompile-commandにする
❻C-uを付けて実行した場合は❸〜❺のルールでcompile-commandを再生成する

smart-compile.elの中核を成す変数であるsmart-compile-alistは**リスト4**のように定義されています。たとえば、Cソースコードについては「gcc -O2 hello.c -lm -o hello」が実行されます。PerlやRubyスクリプトは文法チェックが行われます(今ではflycheckというリアルタイムエラーチェッカーがあります)。

プロジェクトごとのコンパイルコマンドは**リスト5**のようにsmart-compile-alistにルールを追加して設定します。詳しくは、作者による日本語の解説ページ[注1]を見てください。

なお、smart-compile.el自体が古くからある平易な命令のみで記述されているので、elisp入門にうってつけの読みやすいソースコードとなっています。興味ある方はM-x find-library smart-compileでソースコードを読んでみてください。バッファローカル変数について深く理解できるようになるでしょう。

注1) URL http://th.nao.ac.jp/MEMBER/zenitani/elisp-j.html#smart-compile

▼リスト4　smart-compile-alist

```
(defcustom smart-compile-alist
  '(
    ;; emacs-lispは emacs-lisp-byte-compile 関数でバイトコンパイルする
    (emacs-lisp-mode    . (emacs-lisp-byte-compile))
    ;; 略
    ;; 拡張子を表す正規表現は "\\.拡張子\\'" と記述する
    ;; %fがディレクトリを除いたファイル名 / %nが拡張子を除いたもの
    ("\\.c\\'"          . "gcc -O2 %f -lm -o %n")
    ("\\.java\\'"       . "javac %f")
    ("\\.cron\\(tab\\)?\\'" . "crontab %f")
    ("\\.pl\\'"         . "perl -cw %f")
    ("\\.rb\\'"         . "ruby -cw %f")
    ;; 略
    ))
```

▼リスト5　smart-compile-alistに追記

```
(require 'smart-compile)
(add-to-list 'smart-compile-alist '("^/your/project/.*" . "make -C /your/project"))
```

multi-compile

smart-compileパッケージをより強化したものも存在します。multi-compileパッケージはファイル名・メジャーモードに応じて“複数の”コンパイルコマンドを指定できます。smart-compileでは1つだけでしたが、複数個指定できることでコンパイル以外にもテストやデプロイなども受け持てます。設定自体はsmart-compileに似ていますので、複数個のコンパイルコマンドが欲しくなったら移行を考えてみてください。筆者のサイト[注2]で紹介しています。

本稿は初中級者をターゲットとしているため、一気に高機能なパッケージを紹介するよりも、理解しやすいパッケージを優先的に取り上げました。高機能になればなるほど設定が複雑になっていきます。自分が何をやっているのか理解がゆき届かないと危険です。

おわりに

今回は前回に引き続きコンパイル関係のコマンドを紹介しました。今回は誌面の都合上コンパイルコマンドを設定する話にしぼりました。まだ書き足りない部分は次回に回すことにします。次回はコンパイル以外のシェルコマンド実行パッケージも紹介する予定です。

ファイル・ディレクトリローカル変数はコンパイル以外にも便利に使えます。今回はほんのさわり程度の紹介でしたが、詳しくは今後の連載の中で紹介する予定です。

筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指しています。手元でgrep検索できるよう全文をGitHubに置いています。またEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでもお答えします。「こんなパッケージ知らない？」「挙動がおかしいからなんとかしてよ！」はもちろんのこと、自作elispプログラムの添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

注2）URL http://rubikitch.com/2015/11/11/multi-compile

書いて覚える Swift 入門

第16回　創立40周年を迎えたApple



Writer 小飼 弾（こがい だん）　twitter @dankogai

不惑のApple

2016年4月、Apple Inc.は創立40周年[注1]を迎えました。自然人にとってはやっと人生の半分といったところですが、Dog Years換算で280年。その間に起きたコンピュータの世界の変革は、江戸時代全期（265年）以上に長く感じられます（**表1**）。

この40年、DECもSunも吸収合併で消滅し、IBMもPC事業を手放してしまったことを思えば、同社が継続していること自体が奇跡と言っても過言ではないでしょう。実際、Steve Jobs復帰前後は事業破綻一歩手前まで追い詰められ、同氏の復帰後の初仕事はMicrosoftの出資でそれを乗り切ったぐらいです。それが今や「世界で一番価値ある企業」。息を呑まざるを得ません。

しかし同社の40周年は、ちょっとほろ苦いものでもありました。同月26日（現地時間）に発表された16年第2四半期決算[注2]で、同社は13年ぶりに減収減益だったのです。時価総額世界一も、Googleあらため Alphabetがすぐ後ろに迫っています。「ピークオイル」ならぬ「ピークスマフォ」で、同社は今度こそ「オワコン」ならぬ「オワカン」（オワったカンパニー）となるのでしょうか？

筆者の意見は、99% No、1% Yesです。

▼表1　アップルとIT業界の歴史

Year	Apple	IT
1968		intel創立
1969		TCP/IP誕生
1975		Microsoft創立
1976	Apple創立	
1977	Apple][	
1980	Nasdaq上場	
1984	Macintosh	
1981		IBM-PC
1982		Sun Microsystems創立
1985	Steve Jobs 解任	
1990		Software Design創刊
1991	PowerBook	
1994	PowerPC Mac	Amazon創立
1995		Windows 95
1997	NeXT買収；Steve Jobs 復帰	
1998	iMac	Google創立；DEC、Compaqに吸収合併
2001	iPod, Mac OS X v10.0	
2002		Compaq、HPに吸収合併
2003	iTunes Store	
2004		Facebook創立
2006	Intel Mac	Twitter創立
2007	iPhone	
2008	App Store	
2010		Sun、Oracleに吸収合併
2011	Steve Jobs 逝去；Apple時価総額世界一	
2012		IBM、PC市場をLenovoに売却
2014	Swift 1.0	

99%

同社に対する最大の懸念は「Steve Jobsはもういない」というものでしょう。実際同氏が復帰しなければ、同社は21世紀を待たずして終わっていた可能性が高かったのは否定しがたい。

しかし“Jobsless”時代にも同社はIT世界の第一人者であったことは、それ以上に否定しがたいのです。現在のノートパソコンの形を事実上定義したPowerBookの登場は「Jobsless第1期」ですし、そしてSwiftの登場は「Jobsless第

注1）［創立40周年］http://www.macrumors.com/2016/04/01/apple-turns-40-history-timeline/
注2）［16年第2四半期決算］http://www.apple.com/pr/library/2016/04/26Apple-Reports-Second-Quarter-Results.html

2期」です。なぜ同社は40年の長きにわたり、「ただ生き残った」にとどまらず「デジタルデバイスのアイコン」であり続けたのでしょうか。

「過去の栄光は、自ら殺す」という同社の文化に、筆者はその答えを見いだしています。Apple ⅡをMacで殺し、Mac OSはOS Xで殺し、iPodをiPhoneで殺し、……同社の歴史は、同社製品自身の血で真っ赤に染まっています。

その点で見ると、同社はJobs逝去以来「殺し」がなくなってしまったように見えなくもありません。外部ポートのほとんどを「殺した」MacBook発売後もMacBook Airは売られていますし、iPhone SEにいたっては「筐体の黄泉返り」にすら踏み切ってます。

しかし同社史上最大の「殺し」が行われたのは、2012年なのです。何が殺されたのでしょう？

内部留保、です。

1995年以降、利益を100%内部留保していた同社は、2012年7月よりこれまでの姿勢を一転し、自社株買い、株主配当、そして社債を通じて株主に対する利益還元を開始しました。16年第2四半期決算によると、それから2016年5月現在までの通算で1,630億ドル(約18兆円)にも達しています(**図1**)。

ポストJobs時代のAppleは、同社そのものと言っても過言ではない同社株というその一点において別物になっていたのです。

かつてIT企業というのは「配当したら負け」という文化がありました。いや、おそらく今もそうでしょう。配当するぐらいだったら再投資すべきだし、配当せずともその分株価に反映されるなら株主も損をしないからです。Jobs第2期のAppleもそうでしたし、Alphabet(Google)もAmazonもFacebookも創業以来未配当です。

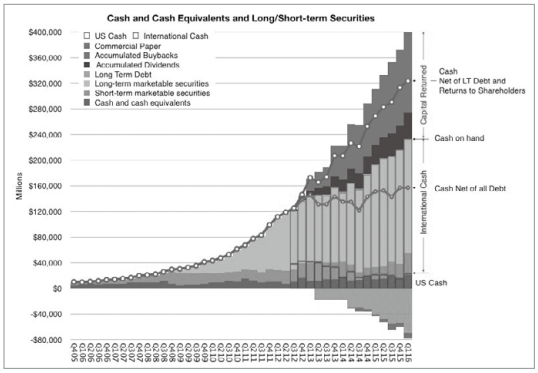
しかし、イノベーションというものは悲しいほど投資額に相関しないものでもあるのです。少し古いデータですが、2013年の「R&D Top 10[注3]」を見てみましょう(**表2**)。ITではSamsung、Intel、Microsoft、Googleとお馴染みの名前が並びますが、すでに時価総額世界第1位だったAppleの名はどこにもありません。「A look at Apple's R&D expenditures from 1995-2013[注4]」によると(**図2**)、同社は46位、投資額$4.5B、売上比率はわずか3%。

投資額とイノベーションが相関しないのは、Appleどころか ITを無視してさえ成り立ってしまうのでトヨタではなくVolkswagenが同リスト首位であることからもうかがえます。それだけお金をかけても「Dieselgate[注5]」を起こしてしまったのですから。

——だとしたら、使い切れないお金は株主に戻すのが正しいということになります。

それではなぜAmazonのように顧客に戻さな

▼図1　Apple's cash positions(https://twitter.com/asymco/status/727107455256240128)



▼表2　R&D Top 10

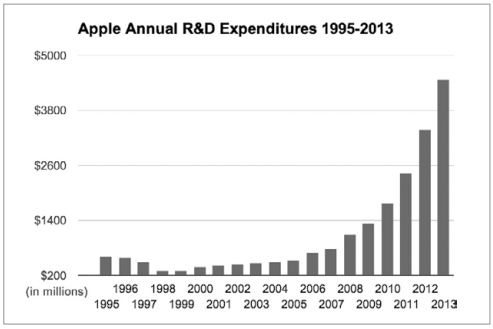
順位	企業名	投資額	売上比率
1	Volkswagen	$13.5B	5.2%
2	Samsung	$13.4B	6.4%
3	Intel	$10.6B	20.1%
4	Microsoft	$10.4B	13.4%
5	Roche	$10.0B	19.0%
6	Novaltis	$9.9B	16.8%
7	Toyota	$9.1B	3.5%
8	Johnson & Johnson	$8.2B	11.5%
9	Google	$8.0B	13.2%
10	Merck	$7.5B	17%

注3)「R&D Top10」http://fortune.com/2014/11/17/top-10-research-development/

注4)「A look at Apple's R&D expenditures from 1995-2013」http://www.engadget.com/2014/02/12/a-look-at-apples-randd-expenditures-from-1995-2013/

注5)「Dieselgate」https://en.wikipedia.org/wiki/Volkswagen_emissions_scandal

▼図2　A look at Apple's R&D expenditures from1995-2013（http://www.engadget.com/2014/02/12/a-look-at-apples-randd-expenditures-from-1995-2013/）



いのか。

それをやってしまうと、コモディティになってしまうからだ、というのが筆者の見立てです。Apple製品に課せられた役割は、それぞれのカテゴリの質で一番であることであって、量で一番であることではないのです。さしずめビールなら、エビス。発泡酒はほかに任せればいい。仮に量まで深追いしたら、今度は独禁法が控えているのはWindowsの歴史を見ればわかります。それよりさらに重要なデバイスとなったことが誰の目にも明らかになった現在、そうなる可能性はWindowsの時以上に高いのではないでしょうか。

「the firstでもthe mostでもなくthe best」という点において、悪くない意味で同社は「相変わらず」であるというのが筆者の意見の99%です。

1%

では残りの1%は何かといえば、「大失敗」の少なさ。

Apple Musicをめぐる混乱も、iPadやApple Watchの「冴えなさ」も、Google GlassやWindows 8と比較するとずいぶんと「控えめ」に感じます。そのクラスの「大失敗」はそれこそ「第一次Jobsless時代」のNewtonまでさかのぼってしまいそうです。「パチンコガンダム駅」が爆誕したiOS 6の地図アプリケーションのときには意地の悪い期待もしてしまったのですが……。

▼図3　Swiftが動く!（https://twitter.com/dankogai/status/730335242553200640）



ではSwiftは？

本連載の主題は、プログラミング言語Swiftです。そしてSwiftがオープンソース化された現在、Swiftの未来＝Appleの未来という構図は崩れています。仮にAppleがなくなってもSwiftがなくならないという未来は、SunがOracleになってもJavaがなくならない以上に確定的に思えます。どころか、Android Appsの開発言語がJavaからSwiftになる可能性すら喧伝されています。少なくともSwiftの「Androidポート[注6]」は、すでにマージされています。

Windowsでも、Swiftが使われるようになる可能性すら、LinuxがWindowsで直に動かせるようになった現在否定できなくなりつつあります。Visual Studio CodeはすでにSwiftをサポートしていますし、Windows Insider Preview上ではすでにLinux Subsystem上でSwiftが動くことは確認済みです（**図3**）。

それでもやはり、SwiftがAppleの子であることの重要性はいまだ薄れていません。Swiftはまずもってobjective-Cキラーとして産声を上げましたが、Swift自身もまたSwiftの餌食であることは、Swift 2がこれ以上ない形で示しました。

そのSwift 2も、Swift 3に殺される運命にあります。次回はそのSwift 3が主題となるはずです。SD

注6）「Androidポート」https://github.com/apple/swift/pull/1442

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。



2016年6月号

第1特集
速く堅実に使いこなすための
bash再入門

第2特集
RDBの学び方
MySQLを武器にSQLを始めよう!

巻頭特集
・Android Wearアプリ開発入門［特別編］
・フリーで始めるサーバのセキュリティチェック［後編］

定価（本体1,220円+税）



2016年5月号

第1特集
コード編集の高速化からGitHub連携まで
Vim［実戦］投入

第2特集
2年ぶりのLTS
安定のUbuntu 16.04の新機能

巻頭特集
・特別SIMで始めよう!　SORACOMでわかるIoT

特別付録
・SORACOM Air SD Special Version

定価（本体1,420円+税）



2016年4月号

第1特集
やればできる!　ワンランク上のプログラミング
今すぐ実践できる良いプログラムの書き方

第2特集
オブジェクトストレージの教科書

特別企画
・適切なLANケーブリングの教科書［番外編］
・春の嵐呼ぶ!　DevOps座談会

定価（本体1,220円+税）



2016年3月号

第1特集
チーム開発をまわす現場のアイデア

第2特集
あなたの知らないCOBOLの実力

一般記事
・iPad Proのさきに見えてくるもの
・Webサイトが改ざん!　サイトオーナーがとるべき行動と注意点

定価（本体1,220円+税）



2016年2月号

第1特集
［最新］MySQLとPostgreSQL徹底比較

第2特集
1Gbps超ネットワーク高速化時代の適切なLANケーブリングの教科書

一般記事
・Android Studioのスタイルで効率アップ!

定価（本体1,220円+税）



2016年1月号

第1特集
はじまっています。ChatOps
導入を決めた7社の成功パターン

第2特集
手軽さとコード化しやすさが人気!
Ansibleでサーバ構成管理を省力化

新連載
・Androidで広がるエンジニアの愉しみ

定価（本体1,220円+税）

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN & ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	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。



Sphinxで始めるドキュメント作成術



波田野 裕一　HATANO Hirokazu　Twitter @tcsh



第16回 Sphinxで運用ドキュメントを作る

運用ドキュメントとは

今回は、Sphinxで運用ドキュメントを書いていきます。

運用ドキュメントとは何でしょうか、と言い始めると長くなる[注1]ので、ここでは「運用業務に関連して、自分や他者に共有する目的で、書かれる文書」のこととします。

今回は、この運用ドキュメントのうち代表的なものとして作業一覧表と作業手順書をSphinxを使って書いていきます。

Sphinxで実現する「質の高い運用ドキュメント」

「質の高い運用ドキュメント」とは、何でしょうか。それは「運用業務に関連した情報を、自分や他者に共有する」という「目的」を実現できる文書のことです。とくに運用業務の継続性の観点から、「他者に共有する」ことを実現することが運用ドキュメントの最大の価値と言えます。

日本の運用現場においては、書き手都合で書かれた運用ドキュメントがとても多く、読み手にとってわかりづらいものが日常的に使われている、という現実があります。

読み手にとってわかりやすい、質の高い運用ドキュメントを実現するためには、Who(主語)／What(目的語)／When(時間)／Where(空間)／Why(根本的な意図)／How(動作)といった「5W1H」が明確になっている必要があります。運用ドキュメント作成に着手した時点では、少なくとも対象業務に関する5W1Hを明確にし、それをもとに大見出しを決定し、ドキュメントの骨組みを作り上げておくようにしましょう。

Sphinxの良いところは、ドキュメントの骨組みが明確になっていれば、余裕ができたときに優先度の高い記事から(各種ディレクティブを活用して)肉付けしていけること、それにより「ドキュメントがどんどん成長していく」ことです。

本記事では、ドキュメントの成長をさらに加速するための、「保守性を意識したドキュメント手法」について説明していきます。

運用ドキュメントの保守性を高めるSphinx

Sphinx利用時に少し工夫するだけで、運用ドキュメントの保守性を飛躍的に高めることができます。ここではSphinxが持つ保守性を3つの観点で見ていきます。

①ワンソース、マルチアウトプット

Sphinxでは、1つのソースから複数のフォーマット(HTML/PDF/man/EPUBなど)でドキュメントを生成できます。もともとHTML用に作成したコンテンツを、PDFやEPUBで配布することになった場合でも、必要なスタイルシートを用意すれば、もともとのソースをそのまま利用して、新しいフォーマットでドキュメント

注1) https://www.nic.ad.jp/ja/materials/iw/2011/proceedings/t3/t3-01.pdf

を作成することができます。

Sphinxは、ドキュメント自体の再利用性が高いツールと言えます。

②ドキュメントの部品化が可能

運用ドキュメントにおいては、似たような文章を書くことが多いため、ドキュメントを部品化しておき、再利用することでドキュメント作成の手間を減らせます。

Sphinxはインクルード機能が充実しており、ドキュメントの一部を外部ファイルとして切り出すことにより、ドキュメントを簡単に「部品化」し、複数のドキュメントから利用することができます。

Sphinxは、ドキュメントの一部を部品化し、再利用性を高めることができるツールと言えます。

③ドキュメントのライブラリ化が可能

上記②で説明した外部ファイルをGitやMercurialなどのバージョン管理システムで管理するリポジトリにまとめることにより、特定分野のドキュメントで利用できる「ライブラリ」を作成し、関係組織に配布できるようになります。

Sphinxでは、conf.pyとMakefile(もしくはmake.bat)とindex.rstがあるプロジェクト・ディレクトリでmakeコマンドを実行すると、そのディレクトリを頂点としたドキュメントを生成します。

このプロジェクト・ディレクトリ配下に、「ライブラリ」リポジトリを複数配置することで、ドキュメント本文から多様なドキュメントやソースコードをインクルードできるようになります。

Sphinxは、ドキュメントのライブラリ化を実現し、汎用性と再利用性を高めることができるツールと言えます。

◆ ◆ ◆

今回は、これらのうち②の「ドキュメントの部品化」を中心に解説していきます。

運用ドキュメントの部品化

運用ドキュメントの部品化は、次のように行います。

❶インクルード機能の活用

まず、部品化の前提として、部品化されたドキュメント(以下、部品ドキュメント)を取り込む方法が必要となる。Sphinxではインクルード関連ディレクティブが充実しており、ドキュメントの部品化が容易である

❷一覧表を部品化する

作業一覧などの一覧表については、テーブル系のディレクティブで表組みを作成し、単体のファイルとして保存する。各種運用ドキュメントはSphinxのインクルード機能を利用して、この一覧表を取り込むことができる

❸手順書を部品化する

手順書のうち共通利用できる部分を切り出して、単体のファイルとして保存する。各種手順書はSphinxのインクルード機能を利用して、この手順部分を取り込むことができる

❹一覧表と手順書を自動リンクする

クロスリファレンス機能を活用し、部品化された一覧表と手順書をリンクすることができる

❶部品を取り込むインクルード機能

Sphinxでは、外部ファイルをドキュメントに取り込む機能として、インクルード関連のディレクティブが提供されています。運用ドキュメントの部品化を行ううえで、次の2つのインクルードがとくに有用です。

■インクルード(include)

reSTructuredTextで記述されている部品ドキュメントを、このディレクティブを記述した位置に読み込むことができます(**リスト1**)。

読み込みの対象となるファイルの指定には、絶対パス指定と相対パス指定の2つの方法があ

ります。Sphinxは、絶対パス指定(/で始まる指定)の場合はプロジェクト・ディレクトリからの絶対パスで、相対パス指定(/で始まらない指定)の場合はincludeディレクティブを記述したファイルからの相対パスで、それぞれインクルード対象となるファイルを検索します。筆者は、部品ドキュメントを柔軟に組み込めるように、常に相対パス指定を利用しています。

さらに、部品ドキュメントでは、次のような工夫をしています。

- ドキュメントビルド時に、部品ドキュメント自体が単体ページとして作成される必要はないため、拡張子をrstではなくtxtとする[注2]
- 見出し(Headings)は、インクルード後にその文字種によってレベルが評価され、部品ドキュメントの活用に制約が生じるため、部品ドキュメント内部では利用しない(見出しを付ける必要があるときは、rubricディレクティブを利用する)

includeディレクティブの活用により、運用ドキュメントのうち共通利用できる記述の部分を、外部ファイルとして部品化することができます。

■リテラル・インクルード(literalinclude)

外部のソースコードを、このディレクティブを記述した位置に読み込むことができます。これにより、プログラムの一部を手でコピーしてくる必要がなく、動作する状態のソースコードをドキュメントの一部として組み込むことができます(**リスト2**)。

リスト2のように、languageオプションでプログラミング言語名を指定すると、その言語にあわせてハイライト表示されます。

literalincludeディレクティブにはstart-afterやend-beforeなどの便利なオプションが存在します。ソースコードの特定の部分だけをドキュメントに組み込みたい場合はこれらのオプションを使用すると良いでしょう。

❷一覧表を部品化する

運用ドキュメントに記載されている一覧表は、更新の重要度が高い割に保守されずに漏れることが多い部分です。Sphinxでは、テーブル系のディレクティブを活用することで、一覧表を効率良く保守できます。

■一覧表を自動作成する(csv-tableディレクティブ)

一覧表のマスタデータがデータベースなどで管理されている場合、最新情報を定期的にCSV形式でエクスポートし、Sphinxでインクルードするようにしておくと、ドキュメントをビルドするごとに最新の一覧表が作成されるようになります。

たとえば、定期的にデータベースから**リスト3**のCSVファイルをエクスポートしているとし

注2) 拡張子にrstを利用した場合、toctreeディレクティブの対象になっていないとwarningメッセージが表示されてしまい、toctreeディレクティブにglobオプションを指定すると、自動的に組み込まれてしまいます。

▼リスト1　インクルードの使用例

```
.. include:: ../../../resource-catalog/cli-sphinx/quickstart/usage-simple.txt
```

▼リスト2　リテラル・インクルードの使用例

```
.. literalinclude:: ../../../resource-catalog/cli-sphinx/quickstart/_command.sh
   :language: sh
```

▼リスト3　データベースからエクスポートしたCSVファイル(define/work-catalog/request/work-catalog-request.csv)

```
"op001","IAMグループ作成",:ref:`op001`,"甲野一郎"
"op002","IAMユーザ作成",:ref:`op002`,"乙野二郎"
```

ます。このCSVでは、「ドキュメント」フィールドにrefロール(:ref:)とドキュメントのラベル名(`op001`)を記載し、各作業ドキュメントにリンクしています(これについては後述します)。

申請作業の一覧表ページにこのCSVファイルを取り込むためには、csv-tableディレクティブを利用します(**リスト4**)。csv-tableディレクティブでは、fileオプションで取り込むCSVファイルを指定します。また、headerオプションでヘッダ行の記述内容を指定し、widthsオプションで各列の幅を指定しています。

すでにマスタデータが確立されている一覧は、このように自動的に一覧表が更新されるようなしくみ作りをすることが重要です。

■一覧表を手動作成する(list-tableディレクティブ)

一覧表を手動で管理する場合は、行単位でテーブルの内容を記述するlist-tableディレクティブが使いやすいでしょう。

今回はテーブルの保守性を意識して、外部ファイル(work-catalog-manual.txt)にlist-tableを記述します。まだ一覧表の内容が確定していないなどの事情によりデータベースで管理されていないものが存在する場合には、このファイルを手動で管理更新することになります。

リスト5では、「*」で行を、「-」で列を記述しています。また、header-rowsオプションでヘッダ行を指定し、widthsオプションで各列の幅を指定しています。

リスト4、5の2つのテーブルを閲覧用のページに反映して、一覧表を完成させます(**リスト6**)。出力結果は**図1**のようになります。

▼リスト4　csv-tableディレクティブの使用例

```
.. csv-table:: 作業カタログ（自動生成版）
   :header: "作業ID","作業名","ドキュメント","作業オーナー"
   :widths: 10, 15, 20, 15
   :file:   ./work-catalog-request.csv  ←リスト3のCSVファイルを指定
```

▼リスト5　list-tableディレクティブの使用例(define/work-catalog/request/work-catalog-manual.txt)

```
.. list-table:: 作業カタログ（手動版）
   :header-rows: 1
   :widths: 10, 15, 20, 15

   * - 作業ID
     - 作業名
     - ドキュメント
     - 作業オーナー

   * - op101
     - IAMロール作成
     - :ref:`op101`
     - 丙野三郎

   * - op102
     - クレデンシャル再発行
     - :ref:`op102`
     - 丁野四郎
```

▼リスト6　自動生成の一覧と手動作成の一覧を一度に閲覧できる一覧表(define/work-catalog/request/index.rst)

```
.. _work-catalog-request:

================================================================
申請作業
================================================================

.. csv-table:: 作業カタログ（自動生成版）
   :header: "作業ID","作業名","ドキュメント","作業オーナー"
   :widths: 10, 15, 20, 15
   :file:   ./work-catalog-request.csv

.. include:: ./work-catalog-manual.txt  ←リスト5をインクルード


----------------------------------------------------------------
作業ドキュメント一覧（申請作業）
----------------------------------------------------------------

.. toctree::
   :glob:
   :maxdepth: 1

   ./*/index
```

リスト4の記述

❸手順書を部品化する

手順書の部品化では、次のようにインクルードを2段階で利用するのが筆者のお勧めです。

■(1)手順のインプット／アウトプットを個別に部品化する

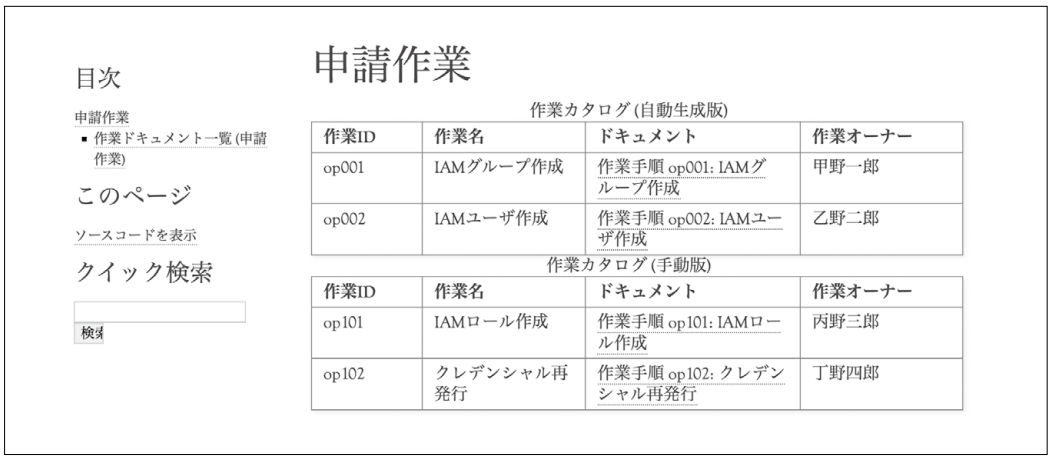
作業手順のインプット(作業で入力するコマンドの文字列)と、アウトプット(コマンドの実行結果)は、literalincludeディクレティブを活用し、コマンド文字列やその結果をそれぞれの個別ファイルに記述します。

コマンド文字列は、そのままスクリプトとしても動作可能なものを手順書に取り込むことが可能です。**リスト7**(_command.sh)はAWSでIAMユーザを作成するときのコマンドを記述したスクリプトです。

コマンド結果については、その出力を直接保存したものをそのまま手順書に取り込むことが可能です。**リスト8**では、**リスト7**(_command.sh)を実行したときの結果を保存したものを少し加工して、置換文字列(|IAM_USER_NAME|など)を埋め込んでいます(置換文字列については後述します)[注3]。

注3) parsed-literalディレクティブを利用すると、置換文字列の位置に置換後の文字列が埋め込まれます。

▼図1　リスト6のビルド後(申請作業の表部分のみ抜粋)



▼リスト7　インプット(コマンド文字列)の記載例(_command.sh)

```
aws iam create-user \
  --user-name ${IAM_USER_NAME}
```

▼リスト8　アウトプット(コマンド実行結果)の記載例(_result.template)

```
結果(例):

  .. parsed-literal::

    {
      "User": {
        "UserName": "\ |IAM_USER_NAME|\ ",
        "Path": "/",
        "CreateDate": "\ |today|\ T12:34:56.012Z",
        "UserId": "AIDAXXXXXXXXXXXXXXXX",
        "Arn": "arn:aws:iam::XXXXXXXXXXX:user/\ |IAM_USER_NAME|\ "
      }
    }
```

■(2)手順のインプットとアウトプットのセットを部品化する

作業手順の最小単位は、手順のインプットとアウトプットのセットです。**リスト9**は、その最小セットの実行前に「変数の確認」手順を組み込んだ例です。

外部のドキュメントからは、通常時はこの部品化されたセットを読み込むようにします。これにより、手順自体の標準化、手順作成工数の削減、保守工数の削減を実現しています。

Sphinxの置換機能を活用すると、さらに運用ドキュメントの保守性が向上します。置換機能を利用するには、まず**リスト10**のように置換ルールを定義します。両側にパイプ(|)を含む文字列が置換文字列で、この置換文字列を「何に置換するか」をreplaceディレクティブで定義します。

この例では|AWS_DEFAULT_REGION|をap-northeast-1に置換するように定義しています。置換ルールを定義したら、**リスト11**のように本文中に置換文字列を記述します。この置換文字列はビルド時に置換ルールに従って置き換えられます。

置換文字列の前後に半角空白が必要なことに注意してください。半角空白をドキュメントに表示したくない場合は、**リスト12**のように\で半角空白をエスケープします。ビルドすると置換が実施され、結果は**図2**のようになります。

この置換を活用することにより、運用現場で需要の多い「そのままコピー&ペーストできる手順書」の実現が容易となります。

たとえば、マスタとなる手順書ではホスト名やIPアドレス、ユーザ名の部分に置換文字列を記述しておき、実際の作業で当日必要となる各値については置換ルールで記述する、という使い方ができます(**図3**)。この場合、当日利用する手順書には、置換ルールと、マスタ手順書のインクルードを記述するだけで、必要なものが完成します。

置換を利用するうえでの注意点として、一度定義した置換ルールを同一rstファイル上で上書き定義できない、というものがあります。こ

▼リスト9　セットサンプル(usage-simple.txt)

```
変数の確認:

  .. code-block:: sh

    cat << ETX

      IAM_USER_NAME: ${IAM_USER_NAME}

    ETX

コマンド:

  .. literalinclude:: ../../../resource-catalog/cli-iam/create-user/_command.sh
     :language: sh

.. include:: ../../../resource-catalog/cli-iam/create-user/_result.template
```

←リスト7(コマンド)をインクルード

←リスト8(実行結果)をインクルード

▼リスト10　置換ルールの定義

```
.. |AWS_DEFAULT_REGION|  replace:: ap-northeast-1
```

▼リスト11　置換の利用

```
リージョンは' |AWS_DEFAULT_REGION| 'を利用します。
```

▼リスト12　置換の利用(半角空白を表示しない場合)

```
リージョンは'\ |AWS_DEFAULT_REGION|\ 'を利用します。
```

▼図2　ビルド後(置換された状態)

リージョンは' ap-northeast-1 'を利用します。

のような場合、rstファイルを分割する(手順自体を分割する)などの対応が必要となります。

❹一覧表と手順書を自動リンクする

部品化された一覧表と手順書をリンクするには、クロスリファレンス機能が便利です。一覧表から手順書にリンクを自動生成するには、見出しにラベルを記述する必要があります。たとえば、**リスト13**では、作業手順書の大見出し「作業手順 op002: IAMユーザ作成」に、ラベル「op002」を記述しています。

このラベルに対してリンクを作成するには、refロール(`:ref:`)を利用してラベルを指定します(**リスト14**)。

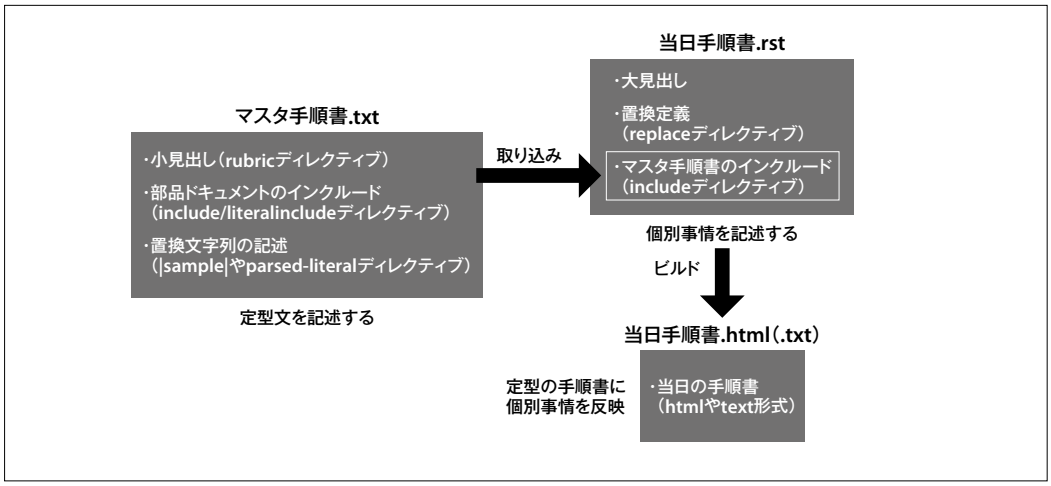
ビルドすると、refロールを記述したところに、アンカー表示として指定ラベルの見出し文字列が自動的に埋め込まれます(**図4**)。この見出しを修正した場合、ドキュメントを再ビルドすることにより、リンク元のアンカー表記も更新されます。

また、ドキュメントのファイル構成が変更になった場合でも、refロールはラベルを自動追跡してくれるため、リンクについて半ばメンテナンスフリーになるところもうれしい点です。

読み手に優しいドキュメントを書こう

運用ドキュメントをWikiで管理するケースは

▼図3　マスタ手順書のインクルード



▼リスト13　手順書にラベルを記述する(define/work-catalog/request/op002/indexの一部)

```
.. _op002:  ←ラベル

============================================================
作業手順 op002: IAMユーザ作成
============================================================
```

▼リスト14　refロールを使って手順書へのリンクを作る

```
IAMユーザの作成の手順については\ :ref:`op002`\ を参照してください。
```

▼図4　ビルド後(リンクが張られた状態)

IAMユーザの作成の手順については作業手順 op002: IAMユーザ作成を参照してください。

よくあると思います。筆者もかつては、作業手順書をWikiで管理し、手順のうち共通部分を別ページとして独立させる方法を多用していました。この方法は、ドキュメント作成者にとっては「効率的」であり、「共通化による恩恵」を得られる方法です。

しかし、あるとき、この手順書を読む作業者にとっては、全体像を把握しにくい、手順の途中でページを切り替える必要があるなど、読み落しによるミスを誘発しかねない「非効率」なものだと気づきました。運用ドキュメントは「他者に共有する目的」で書かれるものです。つまり、「読み手に優しくなければ価値がない」のです。

Sphinxでは、インクルードや置換などの機能を活用することで、作成者にとっての保守性を高めつつ、その作業で必要な手順を1枚の手順書に正確に反映することで作業者にとって優しいドキュメントの実現が可能です。Sphinxを活用して、作業者に優しいドキュメントを効率的に作成していきましょう。

次回予告

次回はSphinxが持つ機能の1つ、「ドメイン」について、その概要と使い方、メリットを紹介します。SD

COLUMN

外部リンクを短縮表記させるマークアップ

いくつかある拡張機能のうち、運用ドキュメントでも便利なextlinks拡張(sphinx.ext.extlinks)[注A]について解説します。

この拡張を利用すると、同一ドキュメント内で外部参照リンクを利用する際に、共通部分をパターン化することができます。ここでは、短縮表記の定義部分を部品化することを想定して、ややトリッキーな手法ですが、extlinksの定義を外部ファイルとした場合の利用方法を説明します。

まず、conf.pyの編集をします。extensionsに`sphinx.ext.extlinks`を追加し、拡張機能を有効にしましょう(**リストA**)。そして、conf.pyの末尾に**リストB**のように追記します。これにより、conf.pyの実行時にプロジェクト・ディレクトリに_extlinks.confが存在する場合は、同時に実行されることになります。

次に、プロジェクト・ディレクトリに「_extlinks.conf」というファイル名で**リストC**のようなファイルを作成します。

本文中に、次のように新しいロール「issue」とイシュー番号を記述します。

```
- Sphinx-User.jp :issue:`1234`
```

ドキュメントをビルドすると、上記の記述部分が「https://github.com/sphinx-doc/sphinx/issues/1234」のイシューチケットへのリンクに変換されます。

外部参照リンクは記述が長くなりがちですので、ドキュメントをシンプルに維持するためにも積極的に活用していきたい機能だと個人的に感じました。

注A) http://docs.sphinx-users.jp/ext/extlinks.html

▼リストA　conf.pyのextensionsに追記

```
extensions = [
  # 追記
  'sphinx.ext.extlinks',
]
```

▼リストB　conf.pyの末尾に追記

```
# sphinx.ext.extlinks
ext_link_conf = './_extlinks.conf'
if os.path.exists(ext_link_conf):
  execfile(ext_link_conf, globals())
```

▼リストC　_extlinks.confを作成

```
extlinks = {
  'issue': ('https://github.com/sphinx-doc/sphinx/issues/%s',
                    'issue '),
}
```

Mackerelではじめるサーバ管理

Writer 坪内 佑樹(つぼうち ゆうき) ㈱はてな
Twitter @y_uuk1

第17回 ロール編成の考え方

監視対象のシステムに対して「良いロール」を付けることができれば、システム構成変更時の対応のしやすさや、グラフの可読性向上につながります。良いロールの付け方の心がけを紹介したあとは、はてなにおけるWebシステムの3層構成を例に、実際のロールの振り分け方をみていきます。

前回は、Mackerelのチェックプラグインの仕様と作り方を解説しました。今回は、Mackerelのユーザに尋ねられることの多いロール編成についての考え方を紹介します。

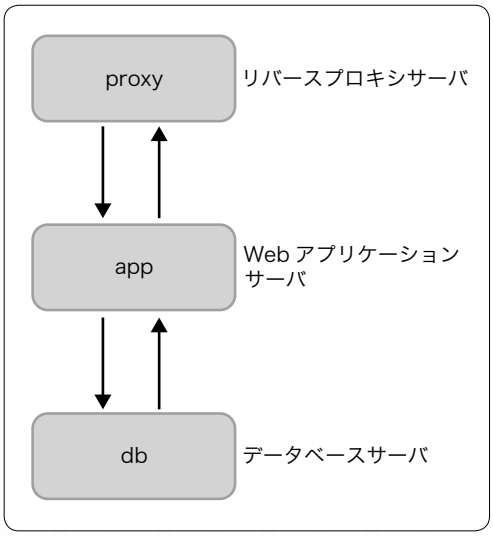
Mackerelのロールとは

「ロールってどうやって付ければいいですか？」とよく質問されます。ロールに限らず、何かに名前を付けてグループ化することはなかなか難しいものです。

そもそもMackerelにおけるロールとは何でしょうか。

ロールは直訳すれば「役割」を指し、Mackerelにおけるロールはホストの「役割」になります。Webシステムであれば、ホストの「役割」は、**図1**のようにリバースプロキシサーバ、Webアプリケーションサーバ、データベースサーバなどが代表的です。

▼図1 Webシステムにおけるホストの役割



Mackerelにおけるロールは、基本的にサーバのソフトウェア構成に合わせて付けることを想定しています。ロールが同じということは、サーバにインストールされているパッケージや設定内容が同じであるということです。

同じリバースプロキシであっても、サービスAのリバースプロキシとサービスBのリバースプロキシではソフトウェア構成が異なるでしょう。Mackerelには「サービス」と「ロール」という2つの単位のグルーピングがあります。たとえば、サービスAのリバースプロキシとサービスBのリバースプロキシは、それぞれ、`ServiceA:proxy`と`ServiceB:proxy`のように表現できます。

「サービス」は「システム」と読み替えても良いでしょう。1つのサービスが複数のサブシステムで構成されているならば、サブシステム単位でサービスを作成します。

マルチロールの活用

本来は1つのホストのロールは1つであることが望ましいのですが、どうしても複数のロールを兼任しなければならないケースがあります。そのようなケースに対応するために、Mackerel

では1つのホストに複数のロールを付与できます。

とくに、開発環境のサーバは本番環境と比べ負荷が小さいため、1台に複数のサーバソフトウェアをインストールすることが多いです。アプリケーションサーバとデータベースサーバが同居しているなら、appとdbロールを1つのホストに付けます。マルチロールにしておけば、あとからの分離もしやすいでしょう。

このように複数のロールを活用して、柔軟にホストを管理できます。

監視を意識したロールの付与

Mackerelでは、ロール単位で監視条件を設定できます。これにより、ホストを追加したときに毎回監視条件を設定せずに済むようになっています。ロールを分けておきたい一方で、監視条件は分けたくないということもあります。監視条件の管理は面倒なため、なるべく1つの設定を使いまわしたいものです。

そこで、同一の条件で監視するために、監視用のロールを用意しておくという工夫があります。基本のロールに加えて、監視用ロールを別途付けます。

はてなの場合、MySQLを、相互レプリケーションによるマスタ・マスタ構成で運用しており、マスタサーバのペア2台に対して、db-masterロールとdb-slaveロールを付けています。マスタサーバなのにdb-slaveロールを付けているのは、スレーブの監視条件の1つであるレプリケーション遅延を監視するためです。db-masterロールにスレーブの監視条件を付けても良いのですが、db-masterロールとdb-slaveロールで監視条件が重複するため、監視条件の管理の手間が増えてしまいます。

APIを意識したロールの付与

Mackerelは、APIとそのほかのツールの組み合わせでその真価を発揮します。とくにロールとその配下のホスト情報を取得し、構成管理ツールなどと組み合わせるといった場合です。

その一例を、本連載の第13回「MackerelとServerspecを組み合わせたインフラテスト」(本誌2016年3月号)で紹介しています。ロール単位でServerspecのspecを書くため、もし同じロールなのにソフトウェア構成が異なっていると、非常にテストがしづらくなります。

そのほかにはたとえば、JenkinsをEC2で複数クラスタ運用しているとします。夜間にJenkinsサーバを稼働させるのはもったいないため、バッチで自動停止させるときに、停止させたいJenkinsのホスト一覧を取得するでしょう。このとき、夜間停止させるJenkinsホストに夜間停止用ロールを付けておき、そのロール配下のホスト一覧を取得するようにしておけば、停止させるJenkinsの管理が楽になります。

ツールと組み合わせるときに、今のロール編成で問題ないか確認しましょう。APIをコールする側で複雑なAPIコールが必要であれば、ロール編成を調整して解決できないかを考えてみてください。

Webシステムにおけるロール編成

Webシステムにおけるロール編成の例を見てみましょう。はてなにおける代表的なロールは**表1**のとおりです。Webアプリケーションの構成については、筆者のブログ記事「はてなで大規模サービスのインフラを学んだ」[注1]を参考にしてください。

MySQLのロール

MySQLはマスタ・スレーブ構成で運用するのが一般的です。**図2**に構成図を示します。図中の点線はレプリケーションの流れを示しています。

マスタをdb-master、参照用スレーブをdb-slave、本番参照のないバックアップ用スレー

注1) URL http://yuuki.hatenablog.com/entry/large-scale-infrastructure

▼表1　はてなのWebシステムにおける代表的なロール

ロール名	説明
proxy	リバースプロキシサーバ
app	Webアプリケーションサーバ
db-master	データベースのマスタサーバ
db-slave	データベースのスレーブサーバ
db-slave-backup	データベースのバックアップ用スレーブ。本番参照なし
batch	バッチサーバ。cronでキックされるジョブが動く
worker	非同期ワーカー
db-worker	ジョブデータベースサーバ
memcached	Memcachedサーバ
redis-master	Redisのマスタサーバ
redis-slave	Redisのスレーブサーバ
redis-slave-backup	Redisのバックアップ用スレーブ
es-master	Elasticsearchのマスタノード
es-data	Elasticsearchのデータノード
es-router	Elasticsearchのルータノード

ブをdb-slave-backupとします。先に述べたように、はてなではマスタの冗長化のために、マスタ・マスタ構成をとっています。

参照用スレーブとバックアップ用スレーブのソフトウェア構成は同じです。しかし、これらのロールを分けずにすべてdb-slaveとすると、どのスレーブサーバが本番参照があるものなのかがわからないため、バックアップ用スレーブを分けています。

Redisのロール

MySQL同様、Redisもマスタ・スレーブ構成で運用するのが一般的です。**図3**に構成図を示します。ただし、MySQLのように相互レプリケーションできないため、フェイルオーバー時にスレーブがマスタに昇格します。つまり、マスタとスレーブが入れ替わります。

このとき、redis-masterロールとredis-slaveロールも入れ替える必要があります。Keepalived[注2]のような高可用性クラスタリングソフトウェアはフェイルオーバー時に任意のスクリプトを実行できるため、そのスクリプト内でMackerelのAPIをコールし、ロールを入れ替えられます。

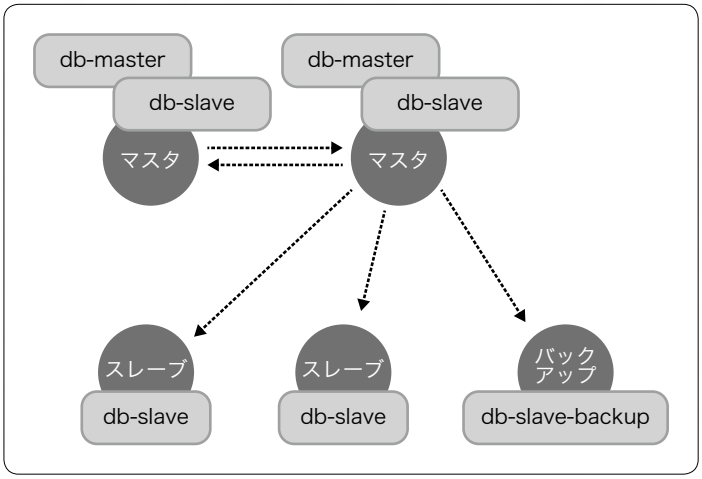
Elasticsearchのロール

Elasticsearchはマスタ・スレーブ構成とは異なる構成のクラスタを構築します。はてなでは、比較的標準的なルータノード、マスタノード、データノードの3種類のノードを組み合わせた構成になります(**図4**)。

ルータノードはアプリケーションからのリクエストを受け付け、適切なデータノードにリクエストを振り分けます。マスタノードはクラスタの状態などを管理します。データを保持せず、データを処理することもありません。データノードはその名のとおりデータを保持します。参照、更新、削除の要求を受け付け、検索処理やアグリゲーション処理をします。

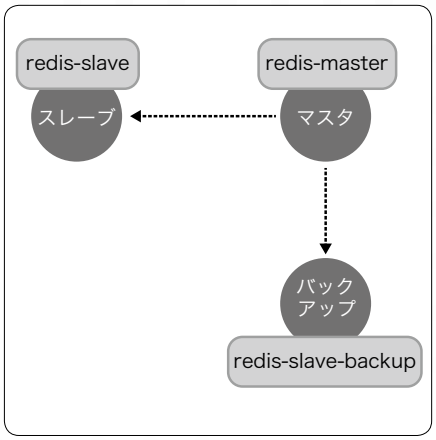
ロール名はそれぞれes-router、es-master、es-dataとなります。単一のホストが複数のノードを兼任することもあります。そのときはそのホストに複数の該当するロールを付けると良いでしょう。

▼図2　MySQLの構成

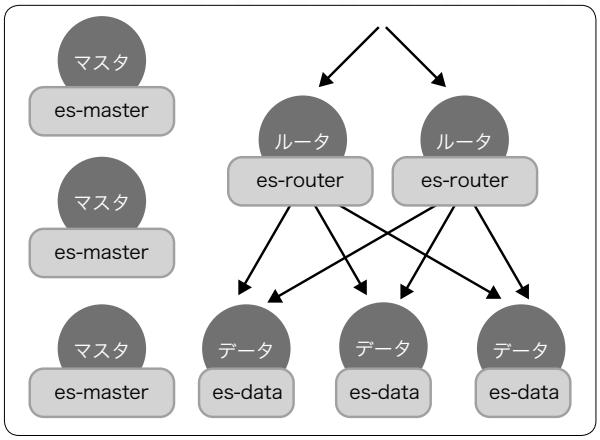


注2）URL http://www.keepalived.org

▼図3　Redisの構成



▼図4　Elasticsearchの構成



Elasticsearchのクラスタ構成について詳しくは、Elasticsearchのノードに関する公式ドキュメント[注3]を参照してください。

データベース分割とロール

データベースはパフォーマンス対策のために、垂直分割や水平分割をすることがあります。

垂直分割は機能単位でデータベースを分割するため、ロール名に機能名をいれます。たとえば、db-function-master、db-function-slaveなどです。

水平分割はシャーディングとも呼ばれることがあり、1つの分割単位を「シャード1」などというように番号で呼ぶことが多いです。ロール名は、db1-masterとdb2-masterのようになります。水平分割と垂直分割を組み合わせている場合は、db-function1-master、db-function2-masterのような名前になるでしょう。

Elasticsearchのようにミドルウェアの実装にシャーディングが組み込まれていて、ミドルウェア側でよしなにシャードを管理してくれる場合は、ロールを分けなくても良いこともあります。

アプリケーションサーバ分割とロール

障害発生時の影響範囲を限定したり、ワークロードに合わせてチューニングするために、アプリケーションサーバを分離することがあります。アプリケーションサーバの分割は本連載の第9回「Mackerelのアーキテクチャを知る」(本誌2015年11月号)で紹介しました。

Mackerelのシステムでは、リバースプロキシにより、ユーザとmackerel-agentのアクセスをそれぞれ別のアプリケーションサーバ群に振り分けます。それぞれ、app-userとapp-agentのようにロールを付けています。これらのロールのソフトウェア構成は同じであっても、ワークロードが異なるのであれば、ロールを分けることによってロールグラフの可読性が向上します。

◆ ◆ ◆

Mackerelにおけるロール編成の考え方とWebシステムにおけるロール編成事例を紹介しました。まとめると次のようになります。

- サーバのソフトウェア構成ごとにロールを用意する
- マルチロールを活用する
- 監視を意識する
- APIを意識する

基本はシンプルなロール編成を心がけておき、必要に応じてロールを分けて、マルチロールを活用するのが良いと思います。ぜひロール編成の参考にしてください。SD

注3) URL https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-node.html

Unixコマンドライン探検隊
Shellを知ればOSを理解できる

Author 中島 雅弘(なかじま まさひろ) ㈱アーヴァイン・システムズ

UnixのShellの実践的な操作とさまざまなコマンドを通じて、OSへの理解を深めていきます。今回は、Unixのプロセスの制御方法、状態の確認についてのお話です。



第3回 プロセス、状態と制御

「男性脳、女性脳」「A型、B型、O型、AB型」など、ラベル付けしてステレオタイプ化するのは人の性なのでしょうか。コンピュータの仕事をしていると、外の人たちからよく「数学ができる人ばかりなんでしょう」、「理系の仕事ですね」などとよく言われます。

理系、文系などの分類は、各国の教育制度によってずいぶんと事情が違い、著者が8年ほど駐在していた米国では「おまえサイエンス系だな」などと言うと首を傾げられることがありました。

実際、著者の周りにも英仏西語などを専門にしていた方々が、大変優れたソフトウェアエンジニアとして活躍されています。確かに論理的思考はプログラミングに不可欠な要素ですが、それがイコール数学の能力ではありません。それよりも、背景やボキャブラリの異なる、ユーザ、外部仕様設計者、ソフトウェアエンジニア、アーキテクト、そしてコンピュータなどがかかわるプロジェクトでは、コミュニケーション能力や言語力が重要な能力だと実感しています。

偉人ノーム・チョムスキーは、言語・哲学など幅広い分野で多くの業績がありますが、彼の言語研究がプログラミング言語理論の発展に不可欠でした。サイエンスの語源は、知ること、分類することなのだそうですが、チョムスキーのように理の探求(ことわり)をすることが理系の生業と前向きに受けとめています。著者にとって理系に対峙する語は、いわゆる文化系学問を修めた人のことではなく、理屈が立たない人のことで「無理系」と呼んでいます。「無理を通せば道理引っ込む」:-)



Unixのプロセス

また今回も新人エンジニアに質問してみました。「プロセスとプログラムの違いは？」

身近？なものでは、「DNA、RNAがプログラム」、「生きているものがプロセス」です。細菌は自身が細胞を持って、宿主に寄生しなくても生きている状態なのに対して、ウィルスは自身が細胞を持たずに、感染先の遺伝情報を書き換えて自己を複製することができる構造物です。ウィルスは宿主なしでは生きていないため、一般には生物とはみなさないのですね。コンピュータウィルスも、感染する前はプログラムの状態で、OSやアプリケーションに感染して動き出しますので、生物世界のウィルスのようなのでこの名前がついたのでしょう。

プログラムは設計図・計画図であるのに対して、プロセスはコンテキスト[注1]を持ったコンピュータ上で動作している、もしくは再開可能

注1) Context：プログラムの実行中の状態・リソース(CPU内部のレジスタやメモリ、オープンしているファイルやデバイスなど)を含むすべてのことです。

な一時停中のプログラムです。今回の質問には、事前に予習していた新人さんはきちんと答えることができました。👍

プロセスは木構造で管理されている

Unixは、**マルチユーザ・マルチプロセス**なオペレーティングシステム(OS)です。複数のユーザが複数のプロセスを同時に利用しても、混乱がないようにプロセスを管理しています。Unixは、プロセスから子プロセスを生成することができ、**図1**のような親子関係のある木構造として管理しています。



プロセスを操る～ジョブコントロール

システムを適切に動作させるには、プロセスの状態を理解してプログラムを書いたり実行することが大切です。ここからは実際にプログラムを実行・停止させて、プロセスへの理解を深めていきましょう。

どのコマンドでも、実行すればプロセスになります。例として、後半で説明するtopコマンド(ここでは使い方は気にしなくてかまいません)を起動してみましょう。

```
$ top↵
```

図2はLinuxで実行した表示例、**図3**はOS Xで実行した表示例です。topは対話型プログラムなので、実行している状態でさまざまなコマンドを受け付けます。でも、今は Ctrl + z (Ctrlキーを押しながらzキーを押す)をキーボードから入力してください。

サスペンド

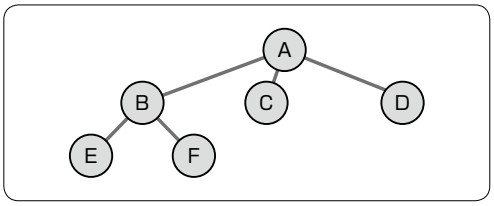
すると、次のような表示が出てシェルの入力待ちになったはずです。

```
[1]+  Stopped                 top
$
```

あなたは、実行中のtopというプロセスに対して、Ctrl + z によってサスペンド(プロセスを一時停止)しろというシグナル[注2]を送りました。その結果、topは停止状態になって親プロセスのシェル(bash)に制御が移りました。

注2) プロセス間通信の手法の1つで、実行中のプロセスに対して外部から割り込み要求をすることです。

▼図1　プロセス木



▼図2　top - Linux

ヘッダ部分

```
top - 15:49:06 up 6 days, 20:41,  3 users,  load average: 0.04, 0.09, 0.12
Tasks: 199 total,   1 running, 196 sleeping,   2 stopped,   0 zombie
%Cpu(s):  2.3 us,  0.7 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   8094256 total,  2655004 used,  5439252 free,   224528 buffers
KiB Swap:  8306684 total,        0 used,  8306684 free.  1682548 cached Mem
```

プロセス部分

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1659 masa      20   0  584732  26596  20176 S   4.0  0.3 408:42.08 indicator-multi
 1478 masa      20   0 1244380 101864  51292 S   3.6  1.3 402:56.49 compiz
 1484 masa      20   0  582452  63220  23184 S   1.7  0.8 175:49.35 unity-panel-ser
 1580 masa      20   0  395588  12480  10888 S   1.7  0.2 130:23.97 indicator-appli
 1381 masa      20   0   43816   4300   2836 S   1.0  0.1  87:26.10 dbus-daemon
  812 root      20   0  529024  71944  61568 S   0.7  0.9  63:57.62 Xorg
16042 masa      20   0   30136   3492   2952 R   0.7  0.0   0:00.08 top
    7 root      20   0       0      0      0 S   0.3  0.0  15:21.81 rcu_sched
(..略..)
```

jobs—JOBS—停止、実行中のプロセス(=ジョブ)を確認する

現在の端末で実行・停止しているプロセスは、jobsコマンドで確認することができます。

```
$ jobs↵
[1]+  Stopped                 top
```

[1]がジョブ番号で、+が今注目しているカレントジョブです。-が付いているジョブは、1つ前のジョブです。jobsは-lオプションを指定すれば、プロセスIDを表示させることができます。

fg—ForeGround—サスペンドされているジョブをフォアグラウンドで実行する

停止状態のジョブを再開させるには、fgコマンドを使います。

fgは引数を取ります。引数を指定しない場合は、+で注目されているジョブがフォアグラウンドで実行されます。フォアグラウンドとは、キーボードからの入力や画面出力が優先されている状態です。

複数のプロセスが停止しているなら、fgの引数にコマンド名を識別できる十分な長さの文字列(この場合、t, to, top)か、ジョブ番号(この場合%1もしくは1)と指定します。実際にfgを実行してみてください。topが再開されたことと思います。

bg—BackGround—サスペンドされているジョブをバックグラウンドで実行する

ここで、もう一度topの実行をサスペンド(Ctrl + z)してください。

bgは、バックグラウンドで実行するというコマンドです。バックグラウンドとは、キーボード入力や画面出力がない限り、停止状態のプロセスを実行状態にするということです。bgもfgと同様の引数を取ります。

実際にbgを実行してみましょう。これでtopがバックグランドで実行中になるはず……リターンキーを入力すると……!?

```
$ bg↵
[1]+ top &

[1]+  Stopped                 top
```

停止状態ですね。topが画面への出力とキーボード入力が必要なため、入力待ち状態で停止

▼図3　top - OS X

ヘッダ部分

```
Processes: 311 total, 3 running, 7 stuck, 301 sleeping, 1690 threads                 15:52:08
Load Avg: 1.43, 1.58, 1.57  CPU usage: 4.69% user, 6.26% sys, 89.3% idle
SharedLibs: 130M resident, 20M data, 10M linkedit.
MemRegions: 71269 total, 3835M resident, 97M private, 906M shared. PhysMem: 8138M used (2034M wired), 52M
unused.
VM: 825G vsize, 535M framework vsize, 969454(0) swapins, 1374933(0) swapouts.
Networks: packets: 6485925/4421M in, 5389325/1010M out. Disks: 13791682/485G read, 6590622/413G written.
```

```
PID    COMMAND      %CPU TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP  PPID  STATE    BOOSTS  %CPU_ME
47400  top          10.5 00:00.74 1/1   0    21    2920K  0B     0B     47400 11995 running  *0[1]   0.00000
47398- mdworker32   0.0  00:00.34 3     0    52    5108K  0B     0B     47398 1     sleeping *0[1]   0.00000
47396  ocspd        0.0  00:00.02 2     0    30    1264K  0B     0B     47396 1     sleeping *0[1]   0.00000
47390  vim          0.0  00:00.49 1     0    17    1472K  0B     0B     47390 11995 sleeping *0[1]   0.00000
47382  mdworker     0.0  00:00.06 3     0    49    3040K  0B     0B     47382 1     sleeping *0[1]   0.00000
47355  mdworker     0.0  00:00.05 3     0    40    3000K  0B     0B     47355 1     sleeping *0[1]   0.00000
47302  Google Chrom 0.0  00:08.59 9     0    120   106M   0B     0B     46979 46979 sleeping *0[2]   0.00000
47298  Google Chrom 0.0  00:07.83 12    1    125   73M+   0B     0B     46979 46979 stuck    *0[2]   0.00000
(..略..)
```

プロセス部分

してしまったと推察できます。画面出力は、バックグラウンドになっても画面に出力されます。

kill—KILL—プロセスにシグナルを送る

killコマンドは、プロセスにシグナルを送るコマンドです。killなんて、なんだか恐ろしい名前がついていますが、プロセスに送るシグナルの大半は、外部からプロセスを終了させるためのものです。killを使うときはだいたいプロセスを停止させたいときです。

```
kill［-シグナル］<プロセスid>
```

と使います。シグナルを省略すると15のSIGTERMが送られます。プロセスidの部分は、先のjobsで確認したジョブ番号(%1など)を指定することもできます。シグナル番号を指定するとき"-"を、またジョブ番号には"%"を付けます。忘れるとプロセスIDを指定したことになりますので注意してください。

スーパーユーザ以外は、自分のプロセスに対してしかシグナルを送ることはできません。勝手に他人があなたのプロセスを止めたら腹がたちますよね。

シグナルの一覧は、kill -lで表示させることができます(**表1**)。

STEP UP!

プロセスが終了しない場合は、次の順番でkillしてみましょう(シグナル名のSIGは省略可)。慣習に従って行儀よく書かれたプログラムなら、それぞれのシグナルに対して個々適切な処理をしているはずです。

kill -9つまり-KILLは、プログラム終了時の後処理(プログラム終了時のファイルへの書き出し、確保した共有領域の解放など)もせずにプロセスが終了しますので、できるだけ使わないようにします。

```
kill -TERM <pid>
kill -HUP <pid>
kill -KILL <pid>   最終手段。これでダメならシステムリブートか、ほおっておくか
```

ログアウトしても処理をバックグラウンドで実行し続けるには

ログアウトすると、ログイン端末での元のシェルが終了して、シェルから起動された終了していない処理は中断されてしまいます。夜間のメンテナンスなどで、処理をバックグラウンドで実行したままログアウトしても処理を続行させるには、nohupコマンドを使いましょう。

```
$ nohup yakan_batch hikisu &↵
appending output to nohup.out
```

この処理の出力は、nohup.outというファイルに書き加えられるというメッセージのとおり、結果はnohup.outに書き込まれます。



プロセスの状態を見る

さて次は、プロセスの状態を見てみましょう。プロセスの状態を確認したいときは、1)実行したはずのプログラムが動いているのか 2)プロセスが止まっている理由を知りたい 3)CPUを異常に使ったりしていないか 4)メモリを異常に使っていないか 5)システムの物理メモリが不足してスワップ[注3]が生じていないか、など

注3) 物理メモリが不足するとHDDなどの2次記憶装置にアクティブでないメモリを退避すること。

▼表1 重要なシグナル

#	シグナル名	デフォルトアクション	説明(一般的に起こりうる状態※1)	キーボードから送信時(標準的なキーの割当)
1	SIGHUP	プロセスの終了	端末の回線が切断※2	DEL
2	SIGINT	プロセスの終了	割り込み要求	−
3	SIGQUIT	core※3を出力してプロセスを終了する	プロセスをぬけ出す	Ctrl + \
9	SIGKILL	プロセスの終了	プロセスを殺す	−
15	SIGTERM	プロセスの終了	ソフトウェアリクエストによる終了	Ctrl + c
18	SIGTSTP	プロセスの停止	キーボードからのサスペンド要求	Ctrl + z
19	SIGCONT	停止要求の解除	サスペンド状態からの復帰	bgコマンドと同じ

※1 Unix創生のころからなので、今では起こりえないような状況も多くあります。そのため、オリジナルの意図とは異なったソフトウェア仕様上の都合のよい使い方がされます。
※2 昔は電話回線やシリアルでつながっていた。
※3 デバッグのためにプロセスのメモリ状態をすべてファイルに出力すること。

と多様な状況です。マルチユーザ・マルチプロセスのOSでは、リソースに対しての注意をはらうことは重要です。さらに、6)システムのパフォーマンスが異常 7)システムが不安定なときは、この後紹介するpsやtopで確認すると原因がわかるかもしれません。

ps—Process Status—プロセスの状態の“スナップショット”をレポートする

Unix上でプロセスの状態を確認するコマンドはpsです。実行してみると、次のような結果が得られます。

```
OS X
$ ps
  PID TTY           TIME CMD
11995 ttys002    0:01.21 -bash
47515 ttys002    0:02.35 vim unix_03.txt
44433 ttys003    0:00.07 -bash
47568 ttys003    0:00.46 ssh ubuntu0

Ubuntu
$ ps
  PID TTY          TIME CMD
16203 pts/18   00:00:00 bash
16295 pts/18   00:00:00 man
16308 pts/18   00:00:00 pager
16318 pts/18   00:00:00 ps
```

psがデフォルトでレポートするのは、ログインしているユーザが実行しているプロセスに関連するプロセスID、実行している端末(TTY)、稼働時間とコマンド名だけです。詳細な情報が必要ならオプションを指定します。psのオプションとレポートする内容はとても豊富ですが、次のオプションによるBSD系Unix型のレポートは重要な情報をおおむね含んでいて便利です。

a：自分のプロセスだけでなく、他人のプロセスも表示する。xオプションと一緒に使わないとターミナルを持たないプロセスは表示しない
u：ユーザIDを表示する
x：ターミナルを持たないプロセスも表示する

a、u、xのオプションは、OS XでもLinuxでも共通であることも複数のシステムを操る技術者にとっては助かります。**図4**の例はaとuのオプションのみ指定していますが、システムで動作しているプロセスをすべて見たい場合は、xを付け加えます。

プロセスの状態遷移

OSにとってのプロセスは、実行中や停止の状態を遷移するステートマシンです。psのaオプションで表示される“STAT”のシンボルが、プロセスの状態を示しています。

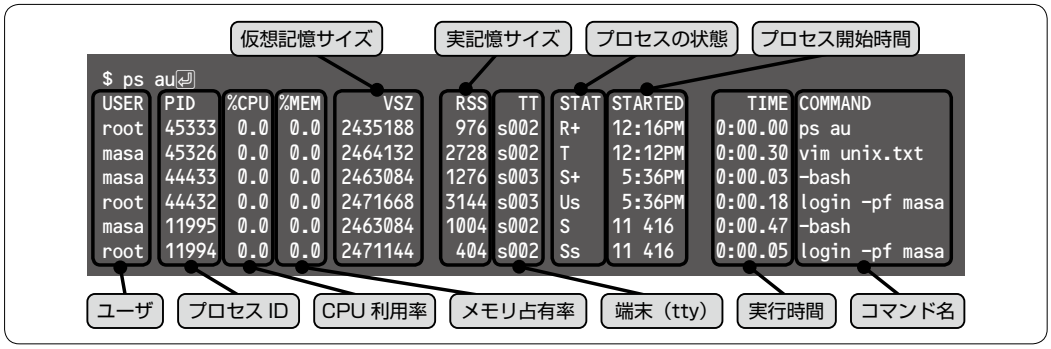
STAT[注4]がどのように遷移するかイメージを持っておきましょう(**図5、6**)。

STEP UP! ゾンビプロセスの話

通常親プロセスに作られた(forkシステムコール)子プロセスは、終了後親プロセスによるwaitシステムコー

注4) これらの主ステータスの後ろに修飾的な情報がつきます。これによって、たとえば同じスリープ状態でも何が理由でスリープしているかを確認できます。

▼図4 ps auの情報



ルによってプロセステーブルから消し去られます。
しかし、子プロセスの終了状態、親プロセスからのwait以前に親プロセスが先に終了してしまうと、子プロセスの情報はプロセステーブルに残ってしまいます。この状態をゾンビ(もしくはdefunct)と呼びます。
親プロセスに先立たれた子プロセスは孤児にならず、すべてのプロセスの先祖、プロセスID 1のinitと養子縁組します。ゾンビプロセスは、いずれinitからのwaitでプロセステーブル中の情報を解放することになります(Linuxなど一般的Unix系のOSでは最初の起動されるプロセスは"init"。pidは1です。OS X 10.11.4では"launchd"です)。

ppid、pidを表示する

jオプションは、プロセスID(PID)に加えて、親プロセスのID(PPID)を表示するオプションです[注5]。**図7**の出力結果を見ると、psの親プロセスがbashであるなど、それぞれのプロセスの親子関係がわかります。

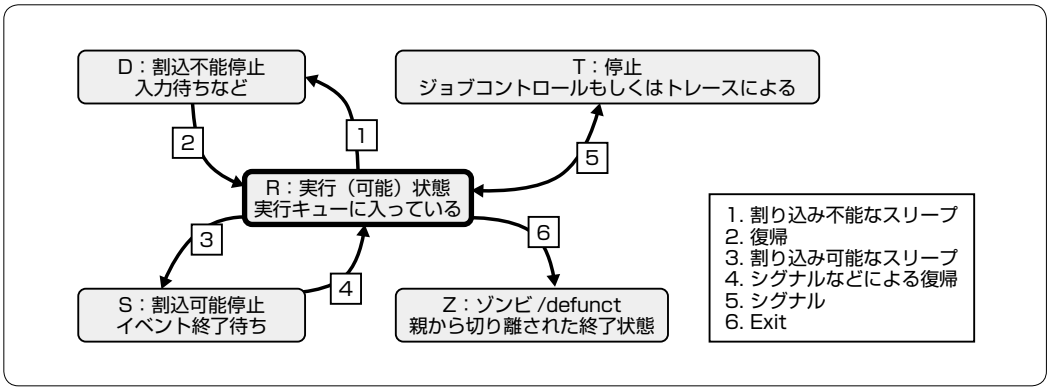
top—TOP—忙しいプロセスを順番に並べてみる

冒頭で実行した**top**コマンドは、システムやプロセスの状態をCPU、メモリなどの消費が多い順など、指定したキーで並べ替え、上位の件数だけプロセス表示します。

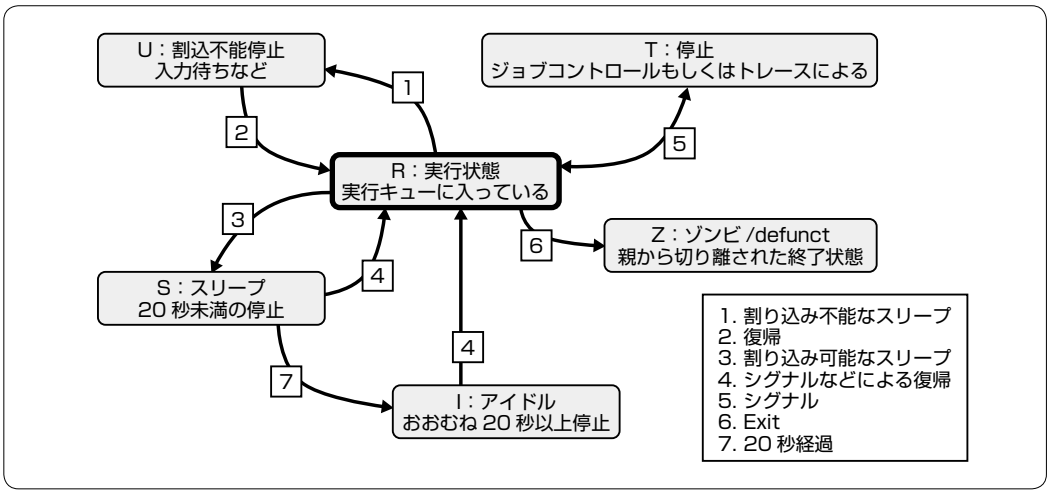
前出の**図2**、**3**をもう一度確認しましょう。実線枠でくくってあるところがヘッダと呼ばれるパートで、CPUやメモリなどシステム全体

注5) Linuxではuとjオプションは一緒に指定できません。

▼図5 Linuxプロセスの状態遷移



▼図6 OS X(FreeBSD)プロセスの状態遷移



▼図7　Linuxでのppid、pid表示

```
$ ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 1968  1974  1974  1974 pts/19    1974 Ss+   1000   0:00 bash
 1968  2070  2070  2070 pts/1     2096 Ss    1000   0:00 bash
 2070  2096  2096  2070 pts/1     2096 S+    1000   0:35 top -p 1379 -p 1848
 3018  3019  3019  3019 pts/21    3054 Ss    1000   0:00 -bash
 3019  3054  3054  3019 pts/21    3054 R+    1000   0:00 ps j
```

の現在状態が表示されています。メモリが不足してスワップしてしまっていないか、CPU稼働率が異常に高くなっていないかをひと目で見られてとても便利です。点線でくくってあるパートには、プロセスの状態が指定されたキーでソートされています。デフォルトではCPU使用率が高い順に表示されます。

topを実行するとデフォルトの対話モードになります。対話モードでは、システムやプロセスの情報が一定間隔[注6]で自動更新されます。対話モードでキー入力をすると、topの表示情報を変更したり、プロセスにシグナルを送ったりできます。また、端末の画面サイズを変更すれば、サイズに合わせて表示項目が増減します。

一方記録を残したい場合などは、出力をファイルや別のコマンドにリダイレクトする、バッチモードで起動することができます。

LinuxとOS Xのtopは、表示情報、対話時の操作方法や起動時オプションも異なります。よく使うオプションと操作を**表2**、**3**にまとめておきます。

CPUやメモリを多く使っているプロセスを見つけるのに便利なtopですが、topが忙しいプロセスの上位に位置することがあります。更新頻度、サンプリングの正確さを高める指定をしていると、topがたくさんのCPUを消費し無用にシステムに負荷をかけるばかりか適切な測定ができなくなりますので注意しましょう。

注6）起動時／対話時オプションで変更可能。



今回のまとめと次回について

今回の確認リスト

【manで調べるもの（括弧内はセクション番号）】
bash(1)、ps(1)、top(1)、kill(1)、nohup(1)、fork(2)、wait(2)
OS X：signal(3)、builtins(1)
Linux：signal(2)、signal(7)、bash-builtins(1)
【以下はbashのhelpコマンドを使って確認】
jobs、fg、bg

プロセスとプログラムの違い、Unixプロセスの構造が木構造であること、ジョブ制御の仕方、シグナル、プロセス状態の見方について解説しました。プログラムのチューニング時、システムの運用時にとてもよく使うことになるコマンドとしくみですので、しっかりと理解して使いこなせるようになりましょう。次回は、基本ファイル操作を紹介します。SD

▼表2　対話的コマンド（代表的なもの）

アクション	OS X	Linux
CPUでソート	oの入力後 cpu	P
Memoryでソート	oの入力後 mem	M
木構造で表示	—	V
画面更新	Ctrl-L	←
killシグナル	S<signal><pid>	k pid↵ signal↵
CPU/coreごと表示	—	1
help	?	?
終了	q	q

▼表3　起動時オプション（代表的なもの）

指定内容	OS X	Linux
取得回数	-l #	-n #
表示するユーザ	-U <user name>	-[Uu] <user name/id>
表示するプロセスid	-pid #	-p # (,で区切って複数指定できる)
表示するプロセス数	-n #	—
セキュアモード	—	-s

※　#には数値を入れる。

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。

最新刊！



前橋和弥 著
B5変形判・304ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-8188-2



五味弘 著
B5変形判・272ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-8035-9



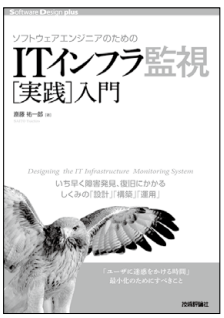
養成読本編集部 編
B5判・200ページ
定価2,080円（本体）+税
ISBN 978-4-7741-8034-2



高橋基信 著
A5判・256ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-8000-7



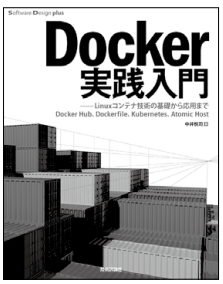
高宮安仁、鈴木一哉、松井暢之、村木暢哉、山崎泰宏 著
A5判・352ページ
定価 3,200円（本体）+税
ISBN 978-4-7741-7983-4



斎藤祐一郎 著
A5判・160ページ
定価 2,280円（本体）+税
ISBN 978-4-7741-7865-3



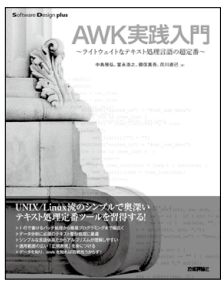
神原健一 著
B5変形判・192ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-7749-6



中井悦司 著
B5変形判・200ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-7654-3



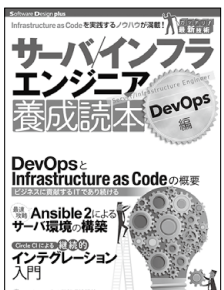
上田隆一 著
USP研究所 監修
B5変形判・416ページ
定価 2,980円（本体）+税
ISBN 978-4-7741-7344-3



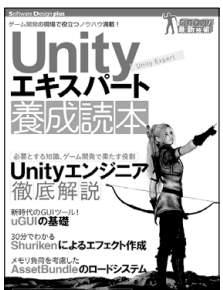
中島雅弘、富永浩之、國信真吾、花川直己 著
B5変形判・416ページ
定価 2,980円（本体）+税
ISBN 978-4-7741-7369-6



養成読本編集部 編
B5判・112ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7992-6



養成読本編集部 編
B5判・176ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7993-3



養成読本編集部 編
B5判・192ページ
定価2,480円（本体）+税
ISBN 978-4-7741-7858-5



養成読本編集部 編
B5判・168ページ
定価 2,280円（本体）+税
ISBN 978-4-7741-7962-9

データサイエンティスト養成読本 機械学習入門編
養成読本編集部　編
定価 2,280円+税　ISBN 978-4-7741-7631-4

C#エンジニア養成読本
岩永信之、山田祥寛、井上章、伊藤伸裕、熊家賢治、神原淳史 著
定価 1,980円+税　ISBN 978-4-7741-7607-9

Dockerエキスパート養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7441-9

サーバ／インフラエンジニア養成読本 基礎スキル編
福田和宏、中村文則、竹本浩、木本裕紀 著
定価 1,980円+税　ISBN 978-4-7741-7345-0

Laravelエキスパート養成読本
川瀬裕久、古川文生、松尾大、竹澤有貴、小山哲志、新原雅司 著
定価 1,980円+税　ISBN 978-4-7741-7313-9

Pythonエンジニア養成読本
鈴木たかのり、清原弘貴、嶋田健志、池内孝啓、関根裕紀、若山史郎 著
定価 1,980円+税　ISBN 978-4-7741-7320-7

事例から学ぶ情報セキュリティ
中村行宏、横田翔 著
定価 2,480円+税　ISBN 978-4-7741-7114-2

データサイエンティスト養成読本 R活用編
養成読本編集部　編
定価 1,980円+税　ISBN 978-4-7741-7057-2

Javaエンジニア養成読本
きしだなおき、のざきひろふみ、吉田真也、田洋一、渡辺修司、伊賀敏樹 著
定価 1,980円+税　ISBN 978-4-7741-6931-6

OpenSSH［実践］入門
川本安武 著
定価 2,980円+税　ISBN 978-4-7741-6807-4

JavaScriptエンジニア養成読本
吾郷協、山田順久、竹馬光太郎、智大二郎 著
定価 1,980円+税　ISBN 978-4-7741-6797-8

WordPress プロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6787-9

内部構造から学ぶPostgreSQL 設計・運用計画の鉄則
勝俣智成、佐伯昌樹、原田登志 著
定価 3,300円+税　ISBN 978-4-7741-6709-1

サーバ／インフラエンジニア養成読本 ログ収集〜可視化編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6983-5

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第32回 ❖使ってみようmtree(8)コマンド（その2）

◉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。

前回の振り返り

前回、ファイルやディレクトリの配置、またそれぞれのパーミッション情報などを表すフォーマットとしてmtree(5)という表記方法があることを紹介しました。これはファイルやディレクトリの配置データとパーミッションデータをまとめて保持しておけるフォーマットで、古くから使われているフォーマットです。

FreeBSDコマンドのいくつかは、このmtree(5)フォーマットを指定した処理ができるようになっています。mtree(5)はパーミッションなどの情報を保持する方法として便利なフォーマットですので、今回はその利用例の1つとしてbsdtar(1)コマンドを取り上げます。

アーカイブコマンド bsdtar(1)とmtree(5)

FreeBSD 10のtar(1)コマンドはlibarchive(3)ライブラリを使って実装されたbsdtar(1)が実体です。

```
% tar --version ↵
bsdtar 3.1.2 - libarchive 3.1.2
```

FreeBSD 1.0からGNU tarが採用されてきましたが（**図1**）、GPLv3のライセンスを採用したコードをベースシステムにマージしないという方針のため、GNU tarのデフォルト採用は停止となり、代わりにFreeBSD 5.4のタイミングで導入されたbsdtar(1)が、現在ではデフォルトのtar(1)コマンドとなっています。tar(1)コマンドは、bsdtar(1)コマンドへのシンボリックリンクになっています（**図2**）。

▼**図1 参考までにCentOS 7のtar(1)コマンド：GNU tar**

```
% tar --version ↵
tar (GNU tar) 1.26
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by John Gilmore and Jay Fenlason.
```

▼**図2 tar(1)はbsdtar(1)へのシンボリックリンク**

```
% ls -l /usr/bin/tar /usr/bin/bsdtar ↵
-r-xr-xr-x  1 root  wheel  58392 Apr  5 10:35 /usr/bin/bsdtar
lrwxr-xr-x  1 root  wheel      6 Nov 12  2014 /usr/bin/tar -> bsdtar
```

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。



bsdtarのマニュアルを見るとわかりますが、bsdtarにはGNU tarに存在しない機能として、アーカイブを作成するときにmtree(5)フォーマットのファイルを使用できるという機能があります(**図3**)。この機能を利用すると、ファイルシステム上のファイルの持ち主情報やパーミッションなどに関係なく、それぞれに持ち主情報やパーミッションなどを新しく指定したアーカイブデータを作成できるようになります。状況に応じたアーカイブデータを作成できますので、配布物やアーカイブデータをきっちり作成したい場合に重宝する機能です。



mtree(5)データを使ってアーカイブを作成しよう

前回でも使いましたが、次のような配置のファイルとディレクトリをアーカイブにまとめるケースを考えます。

```
% tree freebsd ↵
freebsd
├── Makefile
├── commands
│   ├── 001
│   ├── 002
│   └── 003
├── sources
│   ├── 001
│   ├── BSD.root.dist
│   └── BSD.sendmail.dist
└── typescript.xml

2 directories, 8 files
```

これらファイルとディレクトリのパーミッションがファイルシステム上では**図4**のようになっていたとしましょう。とくに設定をしなければ、ファイル

▼図3　bsdtarにはアーカイブ作成時にmtree(5)形式で任意のパーミッションなどを指定できる

```
% man tar ↵
 ...(略)...
An input file in mtree(5) format can be used to create an output archive
with arbitrary ownership, permissions, or names that differ from existing
data on disk:

       $ cat input.mtree
       #mtree
       usr/bin uid=0 gid=0 mode=0755 type=dir
       usr/bin/ls uid=0 gid=0 mode=0755 type=file content=myls
       $ tar -cvf output.tar @input.mtree
 ...(略)...
```

▼図4　現在のファイルとディレクトリのパーミッション

```
% ls -lR freebsd ↵
total 7
-rw-r--r--  1 daichi  daichi   628  5月 10 17:48 Makefile
drwxr-xr-x  2 daichi  daichi     5  5月 10 17:48 commands
drwxr-xr-x  2 daichi  daichi     5  5月 10 17:48 sources
-rw-r--r--  1 daichi  daichi  3917  5月 10 17:48 typescript.xml

freebsd/commands:
total 3
-rw-r--r--  1 daichi  daichi  303  5月 10 17:48 001
-rw-r--r--  1 daichi  daichi  202  5月 10 17:48 002
-rw-r--r--  1 daichi  daichi  379  5月 10 17:48 003

freebsd/sources:
total 4
-rw-r--r--  1 daichi  daichi  1213  5月 10 17:48 001
-rw-r--r--  1 daichi  daichi   457  5月 10 17:48 BSD.root.dist
-rw-r--r--  1 daichi  daichi   401  5月 10 17:48 BSD.sendmail.dist
```

チャーリー・ルートからの手紙

とディレクトリはこうしたパーミッションになっていることが多いと思います。この状態でアーカイブを作成すれば、それを展開したものにも同じパーミッションが適用されます。

これを、アーカイブを展開したときに**図5**のようなパーミッションとなるよう、作成時にそれぞれパーミッションを指定することを考えます。ユーザ以外の読み書きを禁止するとともに、基本的にユーザも読み込みしかできないような設定にしてあります。

これを表現するmtree(5)フォーマットデータを作成するもっとも簡単な方法は、目標とするパーミッション状態にファイルシステム上のファイルを設定してから、**図6**のように、mtree(8)コマンドを使ってmtree(5)フォーマットのデータを作成することです。ここでは誌面の関係でmtree -Cを通してコンパクト形式のフォーマットに変えていますが(**図7**)、これを実施しなければ前回紹介した状態のmtree(5)フォーマットでデータを作成できます。

tar(1)でアーカイブを作成するときに、このmtree(5)フォーマットのファイルを次のように指定します。

```
% tar jcf freebsd.txz @input.mtree ↵
```

▼図5　目標とするパーミッションの状態

```
% ls -lR freebsd ↵
total 7
-r--------  1 daichi  daichi   628  5月 10 17:48 Makefile
drwx------  2 daichi  daichi     5  5月 10 17:48 commands
drwx------  2 daichi  daichi     5  5月 10 17:48 sources
-r--------  1 daichi  daichi  3917  5月 10 17:48 typescript.xml

freebsd/commands:
total 3
-r--------  1 daichi  daichi  303  5月 10 17:48 001
-r--------  1 daichi  daichi  202  5月 10 17:48 002
-r--------  1 daichi  daichi  379  5月 10 17:48 003

freebsd/sources:
total 4
-r--------  1 daichi  daichi  1213  5月 10 17:48 001
-r--------  1 daichi  daichi   457  5月 10 17:48 BSD.root.dist
-r--------  1 daichi  daichi   401  5月 10 17:48 BSD.sendmail.dist
```

▼図6　mtree(5)フォーマットのデータを作成

```
% mtree -c -p . -k type,mode | mtree -C > input.mtree ↵
```

▼図7　パーミッション情報を書いたmtree(5)フォーマットファイル(コンパクト形式)

```
% cat input.mtree ↵
freebsd type=dir mode=0755
freebsd/Makefile type=file mode=0400
freebsd/typescript.xml type=file mode=0400
freebsd/commands type=dir mode=0700
freebsd/commands/001 type=file mode=0400
freebsd/commands/002 type=file mode=0400
freebsd/commands/003 type=file mode=0400
freebsd/sources type=dir mode=0700
freebsd/sources/001 type=file mode=0400
freebsd/sources/BSD.root.dist type=file mode=0400
freebsd/sources/BSD.sendmail.dist type=file mode=0400
```



この処理で、作成されるアーカイブの中にmtree(5)形式で指定されたパーミッション情報が反映されるようになります。

mtree(5)フォーマットのファイルを指定して作成したアーカイブを展開すると、**図8**のように目的とするパーミッションでファイルやディレクトリが展開されていることを確認できます。配布先に合わせたり、用途に合わせて指定するmtree(5)ファイルを変更するといった使い方もできます。ファイルシステム上の持ち主やパーミッションを展開先の状況に合わせて設定する必要がなくmtree(5)データで指定できますので、そういった使い方でも便利です。



たまにはパーミッションを気にかけてみよう

個人で直接使うことは少ないかもしれませんが、デフォルトで用意されているコマンドにはこうした機能が用意されていますので、一度使ってみるというのはありではないかと思います。インストール系のスクリプトではしばしば使われている機能ですし、機能の内容を知っておくと、そうしたスクリプトファイルの理解も早くなります。SD

column

libmtreeの使い方

mtree(5)フォーマットのデータやファイルシステムからmtree(5)フォーマットのデータを作成するためのライブラリ「libmtree」が開発されています。libmtreeのソースコードはGitHub[注1]で取得できるほか、簡単な説明をFreeBSD Wiki[注2]で確認できます。

mtree(5)データファイルまたは実際のファイルシステムからmtree_spec構造体のデータを作成し、そこからmtree_entry構造体データを取得して各種操作を行うといった方法が、このライブラリの使い方の基本となります。

このライブラリはmtree(8)の実装系に含まれていたコードをベースに、ライブラリとして利用できるようにまとめあげた内容になっています。これまでは、フォーマットは定まっているものの操作のためのライブラリが存在しなかったため、それぞれのコマンドが個別に実装系を持っていました。libmtreeをライブラリとして利用することで、それぞれのコマンドが持っている重複したコードを削除でき、さらにmtree(5)を利用できるシーンが広がりやすくなります。

注1 https://github.com/mratajsky/libmtree
注2 https://wiki.freebsd.org/SummerOfCode2015/mtreeParsingLibrary

▼**図8 アーカイブを展開して中身を確認**

```
% tar jxf freebsd.txz ↵
% ls -lR freebsd ↵
total 7
-r--------  1 daichi  daichi   628  5月 10 17:48 Makefile
drwx------  2 daichi  daichi     5  5月 10 17:48 commands
drwx------  2 daichi  daichi     5  5月 10 17:48 sources
-r--------  1 daichi  daichi  3917  5月 10 17:48 typescript.xml

freebsd/commands:
total 3
-r--------  1 daichi  daichi  303  5月 10 17:48 001
-r--------  1 daichi  daichi  202  5月 10 17:48 002
-r--------  1 daichi  daichi  379  5月 10 17:48 003

freebsd/sources:
total 4
-r--------  1 daichi  daichi  1213  5月 10 17:48 001
-r--------  1 daichi  daichi   457  5月 10 17:48 BSD.root.dist
-r--------  1 daichi  daichi   401  5月 10 17:48 BSD.sendmail.dist
```

37

Debian Developer　やまねひでき　henrich@debian.org

“Wheezy” LTS開始と、パッケージにまつわる悲喜こもごも

Debian Hot Topics

Debian 7 “Wheezy” LTS開始

4月26日からDebian 7“Wheezy”のLTS(Long Term Support、長期サポート)が始まりました。あと2年間(2018年5月31日まで)は主要なパッケージに限ってLTSチームからセキュリティ修正が提供されることになります。Squeeze LTSと異なる点として、Wheezy LTSでは「設定を変更せずに」セキュリティ修正されたパッケージを受け取ることができます。

また、LTS開始直前に、ぷらっとホーム㈱が同社製品の「OpenBlocks」[注1]でのLTSサポートを希望してゴールドスポンサーに加わりました。ただ、LTSはもともとi386とamd64のみの提供だったので、OpenBlocksが動作しているarmel/armhfアーキテクチャのサポートを追加するかというところは若干の論議があったようです。しかし、それも無事に通過したようで、これでWheezyではi386/amd64/armel/armhfの4つのアーキテクチャがサポートされます。

その話し合いの中で、「特定のスポンサーを持たず無償のボランティアで構成／運用されているDebian Projectの資産の利用(ビルドマシンなどの提供)と、現在、Freexian社が主導し、スポンサーの意向に影響されるLTS作業の線引きはどうなるのか(一部の開発者の金銭的利益のために、無償で提供／運用されているプロジェクトの資産が使われるのは問題ないのか？という疑問)」などの議論も起こっています。こちらはこの夏に南アフリカで開かれる開発者会議「DebConf16」での継続審議となりました。

ZFSがunstableに

前回の本連載でも話題にしたZFSがDebian unstableに入りました。Debianの正式なリリース対象ではない「contrib」コンポーネントに区分されているため、パッケージインストール前に明示的に手動でcontribコンポーネントの追加設定が必要です。

/etc/apt/sources.listファイルを

```
deb http://ftp.jp.debian.org/debian sid main contrib
```

のように変更して、「apt update; apt install zfs-dkms」と実行することで、Linuxカーネルモジュールであるzfs-modulesパッケージのインストールが可能になっています。

ownCloudパッケージの削除

2012年6月にDebianのリポジトリに入ったownCloudですが、「Debianのリリーススタイルに合わないので削除するべき」とのバグレポート[注2]がメンテナのDavid Prevot氏から出され、

注1) URL http://openblocks.plathome.co.jp/products/obs_a/

注2) URL https://bugs.debian.org/816376

パッケージが削除されてしまいました。

彼のバグレポートには「数ヵ月前にFedoraのメンテナも同じことを言っていて、s/Fedora/Debian/とすれば[注3]、だいたいそのとおり」とあります。当時のFedoraメンテナ曰く「Upstreamが自分たちでOBS[注4]にパッケージを作っていて、SELinux対応やWebアプリケーションの規則などについてのディストリビューション側のルールを無視したことばかりする……」とのことです[注5]。

バグレポートには、ほかの開発者からのコメントもあり、そこには、「私が理解しているかぎり、削除の決定は技術的な側面(複雑なアップグレードパス)などもからむが、upstreamによるDebianパッケージへの敵対的な態度が根本原因のように見える」とあり、穏やかではありません。

このコメントに対するDavid氏の答えは、「upstreamのスタッフブログやissue trackerでDebianがどんなふうに言われているかをみて(リンクは出したくないけど、見つけるのは難しくないよ)」でした。調べてみると、Debianを名指しで非難しているコメントがいくつも見つかります。どういう流れでこうなってしまったのでしょう？　どうやら次のような経緯があったようです。

- upstreamがバージョンアップのたびに、さまざまなチェックや変更を加える
- Debianのパッケージメンテナが、Debianではバージョンアップのチェックを無視してスキップするパッチを追加してパッケージを作成する
- upstreamが「チェックをスキップするなんて危険過ぎる！」と噛みつく

ownCloudのコードを一番理解しているのは、upstreamです。そのupstreamが加えたアップグレード手順は、「ユーザのデータが危険にさらされないために」確かに必要なものなのでしょう。その手順をDebianパッケージでは無視するようなパッチを、相談もなしに追加したパッケージメンテナの行動は、upstreamから非難されてもしかたがない筋合いのように思えます。

ところが、よくよく確認してみると、この変更がDebianパッケージに加わったのは「experimental」リポジトリ向けです。これを声高に非難するupstreamの態度は褒められたものではありません。なぜかというと、Debianのexperimentalリポジトリは通常のユーザが使うことがない、まさに名前どおりの「実験用」として隔離されたリポジトリです。ここに入っているパッケージを利用したい場合は、unstableリポジトリをベースに、さらに明示的にリポジトリ追加とパッケージインストールが必要で、まさに「実験的にアップロードされたものを確認する」ためのものとなっています[注6]。

何が言いたいかというと、ここに危険な変更が加えられたとしても「それはそういうものでしょ(わかってて使うよね？)」の一言で済ませられる類のものです。「ユーザのデータを危険にさらす!!!」などと言われても、変更を行ったパッケージメンテナ側からすると、「そういう危険もあるからexperimentalに入れているんだけど……？　危険があるとしたら、少なくとも実験的変更を受け入れる意思を持った、ごくごく少数の人柱的な人に対して、だよな？」という感想になります。

たとえ、その変更がupstreamから見たら思慮に欠けるものであったとしても、「その変更はまずい」とだけ指摘して取り消しするように依頼すれば済むはずです……。ところが、その指摘のメール[注7]は、「You want users to lose

注3) 「FedoraをDebianに置き換えれば」という意味。

注4) Open Build Serviceの略。openSUSEが開発している「ソースパッケージをアップロードして自分のRPM/DEBパッケージリポジトリを作成できる」Webサービス。
URL http://openbuildservice.org/

注5) その後、Fedoraでは別の人物がメンテナを引き継いで再度アップロードをしているようです。

注6) experimentalリポジトリに入ったowncloudパッケージを導入するには、/etc/apt/sources.list以下に、「deb https://ftp.jp.debian.org/debian exerimental main」という行を追加したうえで、「sudo apt install owncloud/experimental」のように実行する必要があります。これを無意識にやる人は存在しないはずです。

注7) URL https://lists.alioth.debian.org/pipermail/pkg-owncloud-maintainers/2016-February/002881.html

data?!?!」(お前はユーザのデータを消失させたいのか?!?!)というタイトルで、「こんな危険なことをするなんて！」という調子に満ちたものでした。そして、upstreamがこの問題に対応するということで、ownCloudのGitHubに挙げたプルリクエスト[注8]には、その敵意に満ちた言葉遣い(とブランチ名[注9])が見て取れます。

再度、経緯をまとめると、

- upstreamがバージョンアップのたびに、さまざまなチェックや移行のための変更を追加
- Debianパッケージメンテナの David氏が、アップグレード時のチェックを無視するパッチを加えたパッケージを「experimental」リポジトリにアップロードする(これは複数のメジャーバージョンをスキップせざるを得ないDebianのリリースモデルが原因)
- ownCloudのupstreamが「これは問題だ！」と言い、Debianのリポジトリからowncloudパッケージが入っていたら警告が表示されるようにownCloudを変更
- そして、ownCloudのJos Poortvliet氏が、Debian開発者のメーリングリストに、「You want users to lose data?!?!」という非難のメールを送る
- David氏がownCloudパッケージをDebianから削除するように依頼し、削除される

増え続ける依存関係(現状で約70パッケージ)、必要以上に複雑なアップグレードパス、商標の取り扱い問題、ディストリビューションでのパッケージングに対する配慮のなさ(というよりも明確な「敵意」[注10])……など問題も多く、ownCloudからお金をもらってメンテナンスをしているわけではないDavid氏にしてみれば、敵意に満ちた言葉ばかりを集中砲火的に浴びせられる状況では、うんざりして作業を続けたくなくなるのは道理です。

本記事の執筆にあたって、筆者もownCloudの開発者にコンタクトを取ってみましたが、彼らとしては「ownCloudの名前を冠するなら、たとえexperimentalだろうが我々の要求するクオリティを担保すべきだ。『ownCloudはデータを失う危険がある』というような風評被害にあいたくはない」という立場のようです。

gitlab、Debian公式パッケージ化

一方、ownCloudと対象的なのはgitlabです。Pirate Praveen氏の投稿によれば[注11]、gitlabのDebianパッケージ化に際して、upstreamであるGitlab,Inc.から資金援助をしてもらい関連作業を行ったとのこと。その成果は次のように多岐に渡っています。

- 21個のgemを新規にパッケージ化
- 48個のgemパッケージを更新
- 7個のパッケージを修正
- Debianとgitlabで使っているgemのバージョンを同期
- 8.1.0から8.5.8までのリリースに対してパッケージ更新を追従、lintianでのエラーを解消
- テストインスタンスをgitlab.pxq.inに設置
- Debianにアップロード準備ができてないパッケージのステージング環境としてhttps://people.debian.org/~praveen/gitlab/にリポジトリを作成(現在、同リポジトリはDebian 8用のgitlabパッケージが取得できるようになっている)
- nginx、systemdで動作するように調整。httpsはletsencrypt[注12]が有効に
- experimentalを経てunstableにgitlabパッ

注8) URL https://github.com/owncloud/core/pull/22691

注9) 「distributions-are-gone-insane-nobody-ever-should-install-the-owncloud-debian-package」

注10) そもそもownCloudの開発者が、ディストリビューションでの配布モデルが「安全ではない」という主張を持っていて、リリースサイクルが長めのDebianに対しては好意的ではないという点も影響しているようです。
URL https://statuscode.ch/2016/02/distribution-packages-considered-insecure/

注11) URL https://lists.debian.org/debian-devel/2016/04/msg00019.html

注12) URL https://letsencrypt.org/ 無償でSSL証明書を発行できるしくみと取り組み。

ケージをアップロードし、現在testingに

- git.fosscommunity.inを実際にDebianのgitlabパッケージで動作させるように設定

このまま順調に進めば、Debian 9"Stretch"では「apt install gitlab」とすればすぐに動作するgitlab環境が手に入るはずです。

このようなgitlabの取り組みを見ていると、先のownCloud関係者が言う「Webアプリケーションは動きが速いので、ディストリビューションのリリースポリシーとは合わない」というのは、一理ありますが、彼らの言動は行き過ぎではないか、と思います[注13]。

rebootstrapによる移植作業改善

新規アーキテクチャに対してDebianを移植する作業(bootstrapping)は、試行錯誤を重ね一歩ずつ進んでいく作業です。先日のDebian勉強会でも、tilegxというアーキテクチャへのDebianパッケージ移植をがんばっていらっしゃる方の発表がありましたが、かなり苦難の道程となります。

この作業には手間がかかることもありますが、「いったいどういう手順を踏めば新規アーキテクチャを追加できるのかわからない」という再現性がまったくないという問題があります。これに対する1つの回答がHelmut Grohne氏の「rebootstrap」という取り組みです[注14]。

rebootstrapはシェルスクリプトで実装されており、初めの段階で必要となるパッケージ群に対してクロスビルド[注15]でバイナリを作り出す際に、gccやglibcを含む特定のパッケージに対してパッチを当てて(無理やり)ビルドが通るようなハックがされています。これによって、初期段階の試行錯誤作業をスムーズに終わらせることが可能となっています。

現在、rebootstrapを使って先述のtilegxアーキテクチャや、glibcとは別のlibc実装であるmusl[注16]を使った移植作業などもすでに実施されているようです。

さらに、rebootstrapを使うことで、現状でのクロスビルドができないバグも洗い出されており、パッケージの品質を高めることに役立っているようです。Debianに限らず移植作業に興味のある方は注目ですね。

i386アーキテクチャはi686必須に

以前から動きはあったものの、i386アーキテクチャで明示的にi586のサポートを落としてi686クラスのCPUが必要になったことがアナウンスされました[注17]。以前の記事でdebian-installerでも同様の案内をしましたが、全般的に適用されるということですね。

次に示すプロセッサは未サポートとなりますが、該当のユーザはDebian 8"Jessie"のLTSが終了する2020年までにほかのシステムへ移る算段を付けておくと良いでしょう。

- AMD K5, K6, K6-2(aka K6 3D), K6-3
- DM&P/SiS Vortex86, Vortex86SX
- Cyrix III, MediaGX, MediaGXm
- IDT Winchip C6, Winchip 2
- Intel Pentium, Pentium with MMX
- Rise mP6
- VIA C3 'Samuel 2', C3 'Ezra'

ちなみに、この案内のきっかけとなったバグ報告者の方は、実運用環境で10年以上もunstableで動作させていて、この問題に遭遇したとのこと。いや、「unstable」ですので運用環境には適さないのですけど……。SD

注13) 筆者はDebian寄りであることを差し引いていただくとしても。

注14) URL https://wiki.debian.org/HelmutGrohne/rebootstrap

注15) コンパイラが動作するマシンとは違うアーキテクチャ向けにバイナリをビルドする作業。

注16) URL https://www.musl-libc.org/

注17) URL https://lists.debian.org/debian-devel-announce/2016/05/msg00001.html

Ubuntu 16.04 LTS 落ち穂拾い

Ubuntu Japanese Team
あわしろいくや

今回は、本誌2016年5月号第2特集に掲載できなかったUbuntu 16.04 LTSの変更点をお知らせします。

リリース直前のかけ込み変更

本誌2016年5月号の発売日は4月18日、Ubuntu 16.04 LTSのリリース日は4月21日[注1]で、原稿の締め切りは本誌発売よりも1ヵ月以上前になります。その段階で可能な限りの変更点を盛り込んではいるのですが、Ubuntuはリリースの直前に変更点を追加するということをよくやりますし、今回はとくに多かったように思います。というわけで、今回はそのキャッチアップをすることにしました。

14.04からのアップグレードの前に

7月21日に初のポイントリリースである16.04.1がリリースされる予定となっています。14.04を使用している場合は、このポイントリリースをもってアップグレードの告知が出るようになります[注2]。アップグレードを実行する前に、必ずリリースノート[注3]をご一読ください。事前に注意点が明確になり、トラブルを回避できるかもしれません。

なお、アップグレードは非推奨ながらSSH経由でもできます。方法は簡単で、SSHでログイン後`sudo do-release-upgrade`コマンドを実行するだけです。SSHデーモンがもう一つ起動し、緊急時にはそちらへもログインができるので、さほど危険性はない、と筆者は考えています[注4]。

全般

mainに入る条件の緩和

UbuntuにはCanonicalによってサポートされるmainリポジトリと、コミュニティによるサポートのuniverseリポジトリがあります[注5]。当然mainにあるパッケージはある一定の条件を満たしたものになるわけですが、そのルールが一部緩和されました。

今まではビルドに必要なパッケージもすべてmainにある必要があり、そうではない場合は機能を制限するなどの措置が取られていました。しかしこのたび、ビルドに必要なパッケージがuniverseにあってもmainに入るようになりました。具体的には、Mozcのビルドにはuimが必要です。しかし、uimはuniverseにあるため、mainに入れるためにMozcのuimサポート機能を削るという対応が取られていま

注1）日本時間では4月22日でした。

注2）15.10を使用している場合は、16.04のリリース直後にアップグレードの告知が出ています。可及的速やかにアップグレードしてください。そもそもサポート期間が9ヵ月間ですので、6月にはサポートが終了してしまいます。

注3）https://Wiki.ubuntu.com/XenialXerus/ReleaseNotes/Ja

注4）いわゆるKVM切り替え器を使用している場合は、リモート経由のほうが安全のように思います。アップグレード作業中に切り替えると、戻してもUSBキーボードとマウスが認識せず困ったことが過去にありました。

注5）プロプライエタリなパッケージは、今回は考えません。

した。しかし、uimサポートを有効にしたままでもよくなったということです。とはいえ、この決定もリリース直前であり、uimサポート機能を有効にできるだけの時間的な余裕がなく、16.04では15.10と同じく削られたままです。

Noto Sans CJKフォントのサポート

Noto Sans CJKは、GoogleがAdobeと共同開発し、リリースしているフォントです[注6]。ゴシック体のみですが、CJKの名のとおり日本語・中国語・韓国語をサポートし、7種類のウェイトがあるうえに高品位です。16.04からはこれがデフォルトでインストールされるようになりました。少なくとも現状は、インストールイメージのライブセッションではこのフォントを使用しているものの、インストール後のUIフォントは従来どおりTakaoゴシックフォントであり、このフォントを使用したい場合は手動で変更する必要があります。

とはいえWebブラウザ用のフォントを変更したり、LibreOfficeで使用したりなど、システム全体で使用するフォントになっていなくてもいろいろと活用する機会はあるはずです。

なお、もともとはすべての言語とウェイトが1つになっているバージョンが配布されていましたが、その後アップデートでウェイトごとに分かれたものに変更されました。

Network Managerのアップデート

Network Managerが1.0.4から1.2の開発版である1.1.93にアップデートされました。本誌が発売されるころには1.2系列にアップデートしているものと思われます。

目立った変更は多くはありませんが、同時に複数のVPNに接続できるようになったことはメリットと言えるでしょう。また、サードパーティ製のプラグインを使用している場合はバージョンアップによる非互換が発生しているため、最新版にアップデートする必要がありますのでよく確認してください。

snapパッケージサポート

2016年5月号のUbuntu Monthly Report 第73回で触れられているsnapパッケージ[注7]は、Ubuntuやそのフレーバーでも使用できるようになりました。詳しくは後述します。

Ubuntu

Unity Launcherの下表示

これまでUnity Launcherは左にしか表示できませんでしたが、16.04からは下にも表示できるようになりました。変更方法は2種類あり、コマンドを実行する場合は次のとおりです。

```
$ gsettings set com.canonical.Unity.Launcher launcher-position Bottom
```

元に戻す場合は、次のとおりです。

```
$ gsettings set com.canonical.Unity.Launcher launcher-position Left
```

設定ツールを使用する場合は、**unity-tweak-tool**パッケージをインストールし、起動後[Launcher]タブの[位置]を[下]あるいは[bottom]にします(**図1**)。

GNOMEソフトウェアをUbuntuソフトウェアに変更

リリース直前になって、GNOMEソフトウェアのアイコンと名称が変更されました。機能的な違いはまったくありません。アイコンはUbuntuソフトウェアセンターと同じで、名称はUbuntuソフトウェアとなりました。これはあくまでUbuntuのみの変更であ

注6) おおむね同じフォントを、AdobeはSource Han Sansとしてリリースしています。また、等幅に改良したSource Han Code JPは、エディタや端末用のフォントにぴったりです。筆者もSource Han Code JPをエディタ(もちろんgedit)のフォントに使用しています。残念ながらUbuntuのリポジトリにはありません。

注7) 本文中ではsnappyパッケージとなっていますが、ここではsnapパッケージとします。調査した限りではどちらも使われています。

り、GNOMEソフトウェアセンターを採用するフレーバーには影響しません。

GNOME/Ubuntuソフトウェアも、リリース後ローカルにダウンロードしたパッケージのインストール[注8]ができるようになりました。逆に言えば、まだまだ解決しなくてはいけない問題が多く、Ubuntuソフトウェアセンターの完全な置き換えができるようになったわけではありません。

注8）具体的にはAtomやGoogle Chromeなど、Debパッケージ単体で配布されているソフトウェアのインストールができるようになったということです。

図1　Unity Launcherを下表示に変更したところ



図2　［英語（US）］がある場合は削除する



存在しない英語キーボードを認識する問題

存在しない英語キーボードを認識し、日本語入力の切り替えに支障をきたすことがある問題が発生しています。対処方法は英語キーボードを削除するか、日本語Remixからインストールするか、日本語Remixのパッケージにアップデートするかの3つですが、ここでは削除する方法を紹介します。［システム設定］-［テキスト入力］を開き、［英語（US）］を選択して左下の［-］ボタンをクリックするだけです（**図2**）。

Kubuntu

KDE SCのバージョン

KDE Plasmaが5.5.5に、KDE Applicationsが15.12.3にバージョンアップしました。KDE Plasmaはメジャーバージョンアップです。もちろんリリース直前にです。2016年5月号の第2特集では触れませんでしたが、Kubuntuはリリースマネージャーが変更になっているため、いくつかの混乱が見られます。これもその1つです。

日本語でインストールできない問題

Kubuntu 16.04 LTSでは前代未聞の問題が発生しました。インストール時に日本語を選択すると、インストール完了後ログインができません。英語でインストールすると問題は発生しません。ついでにいうと英語ではないすべての言語で発生します。どういうことなのか調査すると、ローカライズのパッケージ（kde-l10n-*）の最新版をアップロードするのを忘れていた、という想像を絶する結論でした。本稿の執筆時点では、まだこの問題は修正さ

れていませんが、さすがに本誌が店頭に並ぶころには修正されていると思われます。

というわけで、Kubuntu 16.04を使用したい場合は16.04.1からインストールするか、あるいはインストール時に[Kubuntuのインストール中にアップデートをダウンロードする]にチェックを入れれば問題なくインストールできるようになるはずです(**図3**)。

サポート期間

Kubuntuのサポート期間は、12.04と14.04では5年間でしたが、16.04ではほかの多くのフレーバーと同じく3年間に短縮されています。Ubuntuフレーバーのサポート期間はリリース直前に決定するため、どうやっても記事にすることはできませんでした。

Ubuntu GNOME

Waylandのセッション

WaylandというX Window Systemの代替となるディスプレイサーバがずっと開発されているのですが、Ubuntu GNOME 16.04 LTSからはこれがデフォルトで使用できるようになりました。

Waylandセッションでログインするには、デスクトップマネージャーであるGDMで、ログイン時に選択するだけです(**図4**)。現在は実機でプロプライエタリなドライバを使用していない場合のみWaylandのセッションが使用できるようになります。また、まだまだ実験的な段階ですので、何か問題がある場合は通常のセッションに戻してください。

snapパッケージを体験する

では実際にsnapパッケージを使用してみましょう。まず、次のコマンドでリポジトリにあるパッケージ一覧を表示します。

```
$ snap find
```

本記事の執筆段階では46個のパッケージが表示されました。今度はパッケージを検索してみます。結果は**図5**のとおりです。

図3 [Kubuntuのインストール中にアップデートをダウンロードする]にチェックを入れる



図4 [GNOME on Wayland]でWaylandセッションにログインできる

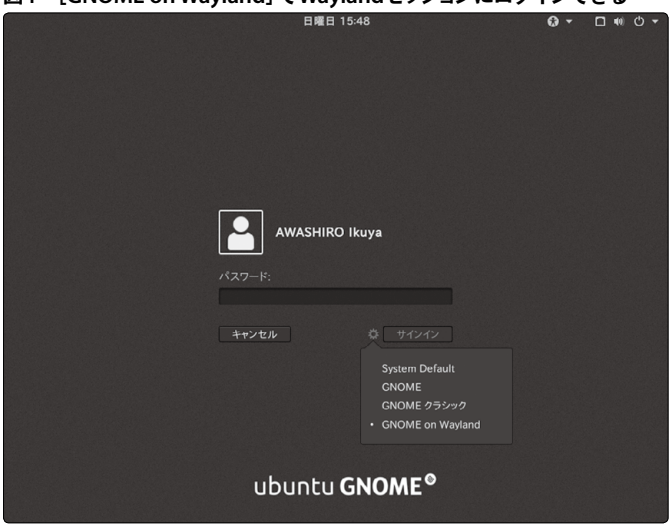


図5 パッケージの検索

```
$ snap find calc
Name                  Version    Summary
ubuntu-calculator-app 2.1+snap3  Ubuntu Calculator application for the Unity 7 desktop
```

これをインストールしてみます。

```
$ sudo snap install ubuntu-calculator-app
```

電卓アプリケーションであるにもかかわらず120MBものファイルをダウンロードするのは驚きです。起動するには、次のコマンドを実行します。

```
$ ubuntu-calculator-app.calculator
```

使い方の解説を表示したあと、関数電卓が表示されます。インストールされているsnapパッケージを表示する場合は、**図6**のコマンドを実行します。

新しいバージョンをインストールする場合は、次のコマンドを実行します。

```
$ sudo snap refresh ubuntu-calculator-app
```

snapパッケージを削除する場合は、次のコマンドを実行します。

```
$ sudo snap remove ubuntu-calculator-app
```

以上のとおりsnapパッケージもaptに近い感じで扱えますが、リポジトリにあるパッケージの数が圧倒的に少なく、またその中には正しく動作しないものもあり、まだまだプレビュー段階といったところです。

snapパッケージはどのようにインストールされるのかというと、/snap/フォルダが作られ、その下にアプリケーションごとのフォルダ、さらにその下にリビジョンごとのフォルダがあり、複数のリビジョンを同時にインストールできるようになっています。具体的には**図7**のようになっています。

実行ファイルは/snap/bin/以下にあり、ubuntu-calculator-appはubuntu-calculator-app.calculatorというシェルスクリプトになっています(**リスト1**)。

このように、すべてのファイルを1つのパッケージにまとめており、リビジョンで管理することによってロールバックも簡単にできることから、割と気軽かつ安全にバージョンアップもできるようになります。さらに重要なことに、既存のDebパッケージやコンテナとも併用できるため、適材適所で使い分ければ管理が大幅に楽になりそうです。今後の成熟が実に楽しみです。SD

リスト1　/snap/bin/ubuntu-calculator-app.calculatorの中身

```
#!/bin/sh
set -e

# snap info
export SNAP="/snap/ubuntu-calculator-app/5"
export SNAP_DATA="/var/snap/ubuntu-calculator-app/5"
export SNAP_NAME="ubuntu-calculator-app"
export SNAP_VERSION="2.1+snap3"
export SNAP_REVISION="5"
export SNAP_ARCH="amd64"
export SNAP_LIBRARY_PATH="/var/lib/snapd/lib/gl:"
export SNAP_USER_DATA="$HOME/snap/ubuntu-calculator
-app/5"

if [ ! -d "$SNAP_USER_DATA" ]; then
   mkdir -p "$SNAP_USER_DATA"
fi
export HOME="$SNAP_USER_DATA"

# Snap name is: ubuntu-calculator-app
# App name is: calculator

ubuntu-core-launcher snap.ubuntu-calculator-app.
calculator snap.ubuntu-calculator-app.calculator
/snap/ubuntu-calculator-app/5/command-calculator.
wrapper "$@"
```

図6　インストールされているsnapパッケージの表示

```
$ snap list
Name                Version                Rev  Developer
ubuntu-calculator-app  2.1+snap3           5    ubuntucoredev
ubuntu-core         16.04+20160419.20-55   109  canonical
```

図7　snapパッケージのフォルダ表示

```
$ ls -la /snap/ubuntu-calculator-app/
合計 8
drwxr-xr-x 3 root root 4096  5月 18 14:32 .
drwxr-xr-x 5 root root 4096  5月 18 14:46 ..
drwxr-xr-x 9 root root  141  4月 19 22:41 5
lrwxrwxrwx 1 root root    1  5月 18 14:32 current -> 5
```

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第29回 デフォルト至上主義

先月の特別付録のせいですっかりLinux通信を忘れていた邪悪な編集長に、デフォルトで原稿料の値下げを強要されてかわいそうなくつな先生の連載が読めるのは本誌だけ！



to be Continued

皆さんと同じで、著者もセットアップ大好きです。好きなセットアップは数年に一度のPC買い替え時、嫌いなセットアップは納期が近いときの作業端末の故障対応です。納期が近いときの緊急セットアップの頻度が高いのですが、著者は前世でよほど業の深いことをしたのでしょうか。すべてのツールをデフォルトで使えるようになれば楽じゃね？——と思ったのですが、カスタマイズしないで使うって結構難しいので、暇な人はぜひやってみて！　まず、Ctrlの位置を替えるの禁止、から。設定ファイルをGitHubに格納・利用端末で共有するのもアリですよね。ただし、アドレス帳を公開状態でpushしないように注意してね。コレを守らないと周囲の人へのSPAMが増えます。



第52回

Linux 4.2から導入されたspinlockの進化形、qspinlock

Text：青田 直大　AOTA Naohiro

ちょうど5月15日と、先月号発売の少し前にLinux 4.6がリリースされました。ほぼ5月いっぱいLinux 4.7のmerge windowということで、さまざまな新機能がコミットされています。今回はLinux 4.2から導入された新しいspinlockの実装であるqspinlockについて紹介します。

spinlock

spinlockは、Linuxカーネルで数多く使われる排他制御メカニズムです。あるCPUがspinlockで保護されたリソースにアクセスしている間、ほかのCPUがそのリソースにアクセスできないようにするものです。

その名前のとおり、スピンしてロックされていない状態になるまで待つというシンプルさから、ロック・アンロック処理のオーバーヘッドが少ないことが特徴です。ただし、スピンで待つという性質からロックの獲得に時間がかかる場合には、それだけスピンによりCPU時間が浪費されてしまいます。Linuxカーネルではその特徴を活かして、短い時間でロックが獲得できそうな場所にspinlockを使い、時間がかかりそうな場所ではmutexなどより複雑なロックを使っています。

spinlockの実装としてLinux 2.6.24までのシンプルなもの、2.6.25から導入されたticket spinlock、そして4.2で導入されたqspinlockについて紹介していきます。

spinlockの素朴な実装

まず、Linux 2.6.24まで使われていたシンプルな実装を見てみましょう。**リスト1**は素朴なロック・アンロックの実装で、__raw_spin_lock()がロック、__raw_spin_unlock()がアンロックを行います。“raw_spinlock_t *lock”の“lock->slock”がロックの状態を管理し、“1”であればロックされていない、そのほかであればロックされていることを示しています。

図1のようにCPU#0からCPU#2がその順番にロックを獲得する状況を見ていきましょう。まず、CPU#0が__raw_spin_lock()を呼び出し、ロックの獲得を開始します。“decl %0”により、asm()に指定されていてるlock->slockの値がデクリメントされます。まだロックがとられていないので、slockは0になり、次の“jns 2f”で結果が負でないので“2:”にジャンプし、ロックの獲得に成功します。もし“decl”の部分でほかのCPU#1、#2がCPU#0と同時にslockをデクリ

メントしようとしても、LOCK_PREFIXがついていることからデクリメントがatomicになり、どれか1つのCPUだけがslockを0にデクリメントし、ロックの獲得に成功します。

CPU#1、CPU#2のようにdeclの結果が負になりロックの獲得に失敗するケースを見てみましょう。“cmpl $0, %0”からの3行はslockが0以下ならまだロックされているのでループの先頭である“3:”に戻り、slockが1ならロックが解除されているので“1:”に戻りロックの獲得を試みるというコードになっています。間の“rep; nop”はこの部分がスピン待ちのループであることをCPUに伝えるヒントとなり、パフォーマンスや電力消費を改善するために使われています。

CPU#0が__raw_spin_unlock()を呼び出すと、slockに1が書かれ、CPU#2または#3(もしくは両方)が“cmp1 $0,%0;～jmp 1b”で“1:”へとジャンプします。LOCK_PREFIXが付いているので、先ほどと同様に、CPU#2か#3のどちらかが“decl %0”でslockを0にして、ロックを獲得します。

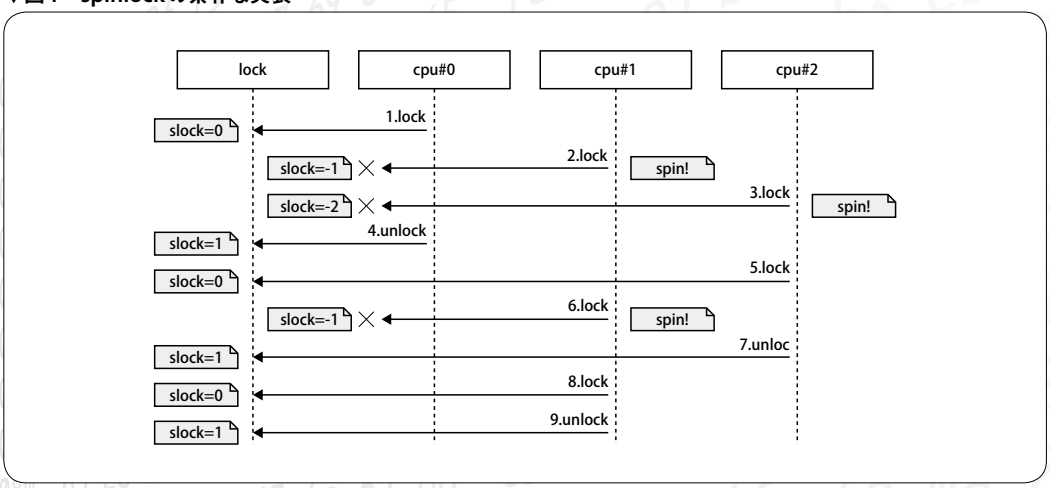
さて、**図1**にはこの実装における1つの問題点が現れています。ロックをとろうとした順番はCPU#1、#2の順番ですが、#2の方が#1よりも先にロックをとることができるという不公正さがあります。これにより、たとえば8コアのマシンでは場合によってはほかのスレッドより100

▼リスト1　素朴なロック・アンロックの実装

```
static inline void __raw_spin_lock(raw_spinlock_t *lock)
{
    asm volatile(
        "\n1:\t"
        LOCK_PREFIX " ; decl %0\n\t"
        "jns 2f\n"
        "3:\n"
        "rep;nop\n\t"
        "cmpl $0,%0\n\t"
        "jle 3b\n\t"
        "jmp 1b\n"
        "2:\t" : "=m" (lock->slock) : : "memory");
}

static inline void __raw_spin_unlock(raw_spinlock_t *lock)
{
    asm volatile("movl $1,%0" :"=m" (lock->slock) :: "memory");
}
```

▼図1　spinlockの素朴な実装



万倍もロックを獲得できないスレッドが出てくるケースもあります[注1]。こうした不公正に対応するように実装されたのがticket spinlockです。

ticket spinlock

ticket spinlockは銀行や市役所のカウンターにあるチケット発行機のようにロックを待っているCPUにチケットを発行し、各CPUにはカウンターの「受付番号」が自分の保持するチケットの番号になるまで待ってもらうような実装になっています。これによって先にロックを待ち始めたCPUが先にロックを獲得することを保障します。

実際のコードを見ていきましょう。**リスト2**がx86におけるticket spinlockの実装です。struct __raw_ticketsであるlock->ticketsはメンバとして"head"と"tail"を持ち、headが現在処理中のチケットを、tailが次に発行するチケットを示します。すなわち初期値はhead、tailともに0となっています。

では**図2**のようにCPU#0から#2が順番にロックを獲得する様子を見てみましょう。CPU#0は

注1）http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=314cdbefd1fd0a7acf3780e9628465b77ea6a836

▼リスト2　x86におけるticket spinlockの実装

```
static __always_inline void arch_spin_lock(arch_spinlock_t *lock)
{
    register struct __raw_tickets inc = { .tail = TICKET_LOCK_INC };

    inc = xadd(&lock->tickets, inc);
    if (likely(inc.head == inc.tail))
        goto out;

    for (;;) {
        unsigned count = SPIN_THRESHOLD;

        do {
            inc.head = READ_ONCE(lock->tickets.head);
            if (__tickets_equal(inc.head, inc.tail))
                goto clear_slowpath;
            cpu_relax();
        } while (--count);
        __ticket_lock_spinning(lock, inc.tail);
    }
clear_slowpath:
    __ticket_check_and_clear_slowpath(lock, inc.head);
out:
    barrier();  /* make sure nothing creeps before the lock is taken */
}

static __always_inline void arch_spin_unlock(arch_spinlock_t *lock)
{
    if (TICKET_SLOWPATH_FLAG &&
       static_key_false(&paravirt_ticketlocks_enabled)){
        __ticket_t head;

        head = xadd(&lock->tickets.head, TICKET_LOCK_INC);

        if (unlikely(head & TICKET_SLOWPATH_FLAG)) {
            head &= ~TICKET_SLOWPATH_FLAG;
            __ticket_unlock_kick(lock, (head + TICKET_LOCK_INC));
        }
    } else
        __add(&lock->tickets.head, TICKET_LOCK_INC, UNLOCK_LOCK_PREFIX);
}
```



“xadd(&lock->tickets, inc);”で、atomicにlock->ticketsのtail部分をTICKET_LOCK_INC(=1)だけ加算し、加算前の値を“inc”に取得します。ここで“inc.head == inc.tail”であれば、自分に発行されたチケットが処理中のチケットになるわけですので“out:”へ移り、待つことなくロックを獲得します。

CPU#1、#2は“for(;;)”の中で自分の順番になるまでスピンすることになります。“for(;;)”の中にもう1つループがありますが、これは後述するPV ticketspinlockのためのループで、実マシン上では関係ないので今は無視します。ループの中では“lock->tickets.head”から現在処理中のチケット番号を読み出し、“__tickets_equal(inc.head, inc.tail)”で自分のチケット番号と一致するかどうかをテストしています。一致すれば自分の順番ということで、ループを抜けてロックを獲得します。ループのあとの“__ticket_check_and_clear_slowpath()”もPV ticket lock用のコードです。

アンロック側のコードを見ていきましょう。if文の真の部分はこれまたPVticket lock用のコードですので無視します。すなわち、実機では単純に“lock->tickets.head”にTICKET_LOCK_INC(=1)が加算されます。するとループで待っていた次のCPUの“__tickets_equal(inc.head, inc.tail)”が真になり、ループを抜けるということになるわけです。

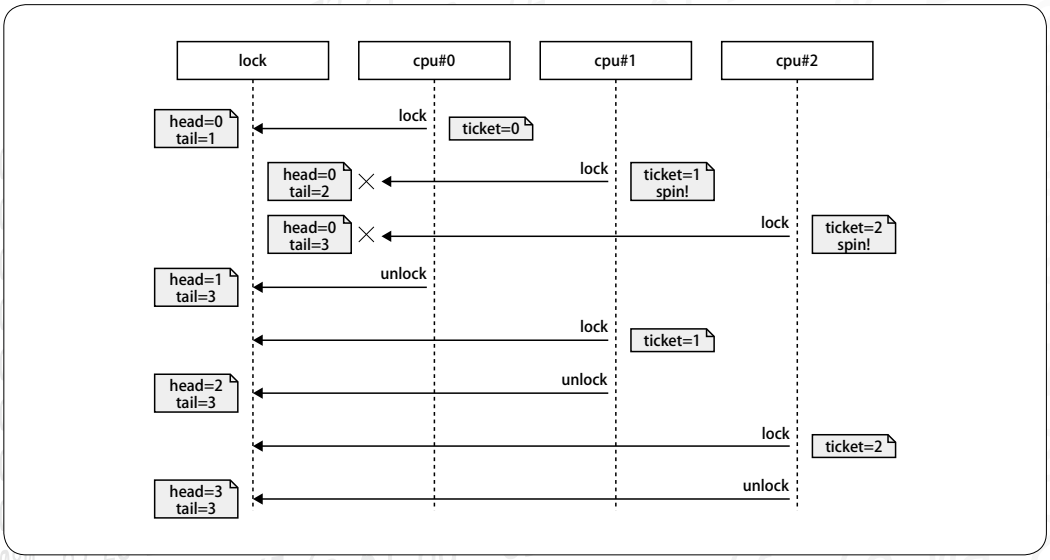


PV ticket spinlock

ここで少し道をそれますが、PV ticket spinlockについて見てみましょう。これは準仮想化(PV)環境用に効率化したspinlockになります。まず仮想環境上における普通のticket spinlockの問題点を見てみましょう。仮想環境においては、1つの物理CPUに複数の仮想CPU(VCPU)が割り当てられることがあります。ここではVCPU#0から#2がすべて同じ物理CPUにだけ割り当てられると想定します。すなわち、1つのVCPUが動作している間はほかのVCPUは動作しません。ここでVCPU#0、#1が同一のロックをとる作業を行い、VCPU#2がほかの作業を行っているとします。

もしVCPU#0がロックをとった直後に、VCPU#0から#1へと動作が移ると、#1はロックをとろうとしてスピンに入ってしまいます。物理マシンであれば、1つのCPUがスピンして

▼図2　CPU#0から#2が順番にロックを獲得する様子



いる間にほかのCPUも同時に動作し、ロックが解除されますが、今の状況ではVCPU#1がスピンしている間、#0は動作しないのでロックは解除されず、#1は無駄にスピンしてしまいます。もし、#1がロックをとるためにスピンしているということがわかれば、ほかのVCPUに物理CPUを割り当てて、たとえばVCPU#2上の作業を進めることができるはずです。

PV ticket spinlockのコードは前掲リストと同じものです。ただし、TICKET_LOCK_INC(=2)になり、最下位bitがspinlockのslow path、すなわちしばらくスピンしていたのにロックをとれなかったVCPUがいることを示すために予約されます。

SPIN_THRESHOLD回スピンしてもロックをとれなかった場合、PV ticketspinlockでは__ticket_lock_spinning()が呼び出されます。KVM guestにおける実装では、lockのアドレスと待ちうけているチケット番号を記録して、VCPUをhaltします。また、ticketのhead部分の最下位bitに1を立てて、ロック待ちでhaltしているVCPUがいることを示します。

アンロック側ではPV ticket spinlockが有効な場合、if文の真の部分に入り、最下位bitが立っていると__ticket_unlock_kick()が呼び出され、次のチケットを待っているVCPUを起こします。ここではまだ最下位bitはクリアされていません。起こされたVCPUはfor(;;)ループの先頭に戻ってチケットを確認し、ループを抜けます。

ループを抜けたところで**リスト3**の__ticket_check_and_clear_slowpath()で最下位bitをクリアします。ほかにロックを待っているVCPUがあるときにbitをクリアすると、次のアンロック時にそのVCPUが起こされなくなってしまうので、cmpxchg()を使って、自分が最後尾である場合にだけ最下位bitがクリアされるようにしています。

このようにPV ticket spinlockでは、ある程度スピンするとVCPUを止めてほかのVCPUをスケジューリングさせることで、スピンで無駄な時間を消費しないように作られています。



MCS lock

さて、一般のticket spinlockに話を戻しましょう。ticket spinlockを使うことで、素朴な実装と違ってFIFOでロックを獲得できるようになりました。しかし、よりコア数が増えてくるとまた別の問題、キャッシュラインの問題が浮上してきます。ticket spinlockではロックを待つすべてのCPUが同一のメモリアドレスを読み込み、その値は各CPUのキャッシュに載っています。この状況でロックを持ったCPUがアンロックを行うと、読み込んでいたアドレスの値が更新され、各CPUが持つキャッシュが無効にされます。ロックを待機していたCPUは一斉に

▼リスト3　最下位のspinlock slowpath bitのクリア

```
static inline void __ticket_check_and_clear_slowpath(arch_spinlock_t *lock,
                                                     __ticket_t head)
{
    if (head & TICKET_SLOWPATH_FLAG) {
        arch_spinlock_t old, new;

        old.tickets.head = head;
        new.tickets.head = head & ~TICKET_SLOWPATH_FLAG;
        old.tickets.tail = new.tickets.head + TICKET_LOCK_INC;
        new.tickets.tail = old.tickets.tail;

        /* try to clear slowpath flag when there are no contenders */
        cmpxchg(&lock->head_tail, old.head_tail, new.head_tail);
    }
}
```

新しい値を読み込みにいきます。この読み込み処理はシリアライズされるので、ロックを待つCPUが増えれば増えるほど次のチケットを持つCPUが新しい値を読み込み、スピンを抜けるのに時間は長くなります。すなわち、コア数が増えるほどに、アンロックから次のロックまでの時間がかかり、パフォーマンスが低下していきます。

そこでLinux 4.2から導入されたのがqspinlockです。qspinlockは前述したキャッシュラインの問題を解消するようなロックであるMCS lockをベースに作られています。ですので、まずはMCS lockから見ていきましょう。

MCS lock

MCS lockは、各CPUに固有のデータ領域(struct mcs_spinlock)を割り当てて、それぞれの領域をスピンすることで、キャッシュラインの問題を回避します.struct mcs_spinlockは次にロックを待っているCPUのデータ領域を示すnextポインタを持ち、struct mcs_spinlockでqueueを構成します。共通の領域であるlockの本体はqueueの末尾(またはqueueが空であればNULL)を指します。

具体的に**図3**のようにCPU#0から#2が順番にMCS lockをとる様子を見てみましょう。以後、CPU#n localのデータ構造nodeをnode[n]と書きます。

CPU#0は**リスト4**のmcs_lock()を呼び出してロックの獲得を試みます。まず、各CPU localのデータ構造であるnodeを初期化します。そして“xchg()”を呼んで現在ロックが指している値とnode[0]のアドレスを入れ換えます。ここではlockはqueueの末尾となったnode[0]を指し、prevにはNULLが入ります。prev == NULLということは、誰もロックをとっていないということですのでCPU#0がロックを獲得します。

次にCPU#1がmcs_lock()を呼び出し、同様に“xchg()”を呼びます。ここで、tailがnode[1]を指し、prevはnode[0]を指します。今回はqueueが空ではないので、node[1]をqueueにつなぎ、ロックの獲得を待つ必要があります。prev->nextをnode[1]を指すようにして、while()でnode[1]->lockedが1になるまでスピンします。CPU#2も同様にqueueに自分のnodeをつなぎ、node[2]->lockedが1になるまでスピンすることになります。

mcs_unlock

では、アンロック側を見ていきましょう。CPU #0はmcs_unlock()を呼び出します。node[0]はnode[1]につながっているので“node->next != NULL”ですのでnode[0]->next(= node[1])の“locked”を1にします。するとCPU#1はスピンを抜けてロックを獲得します。同様にCPU#1もnode[2]->lockedを1にして、CPU#2にロックを渡します。最後に、CPU#2は“node[2]->next == NULL”で、すなわちqueueの最後のメンバであるので“cmpxchg()”でlockをNULLへと書き換えます。“cmpxchg()”は、“lock == node”のときだけ値を書き換え、書き換え前の値を返します。“cmpxchg()”が成功した場合、queueを無事に空にできたということですのでreturnします。“cmpxchg()”に失敗するのはどういうときでしょうか。CPU#2がmcs_unlock()したと同時に、ほかのCPU#xがmcs_lock()を呼び、mcs_lock()側の“xchg()”が先に実行されると、“node[2]->next == NULL”かつ“lock != node”という状況が存在します。この場合、しばらく待つと“mcs_lock()”側がnode[2]->nextをnode[x]に設定してくるわけですので、“node[2]->next”が設定されるまで待ったうえで、next(= node[x])の“locked”を1に書き換え、CPU#xにロックを渡します。



qspinlock

MCS lockはキャッシュラインの問題を解決していますが、各CPUごとにデータ領域が必要となります。カーネル内で短期間のロックに使われるという性質からこの領域を動的には確

保できません。かといって、素直にロックごとにCPUの数だけのデータ領域を予約しておくとロックのデータ構造のサイズが巨大になります。spinlockのデータ構造はカーネルのさまざまなデータ構造に埋め込まれているので、サイズが増えてしまうのは好ましくありません。そこで、各CPUごとにグローバルにnode領域を確保しておきます。node領域が各CPUに1つでは、

▼リスト4　MCS lockの疑似コード

```
void mcs_lock(lock) {
    node->locked = 0;
    node->next = NULL;

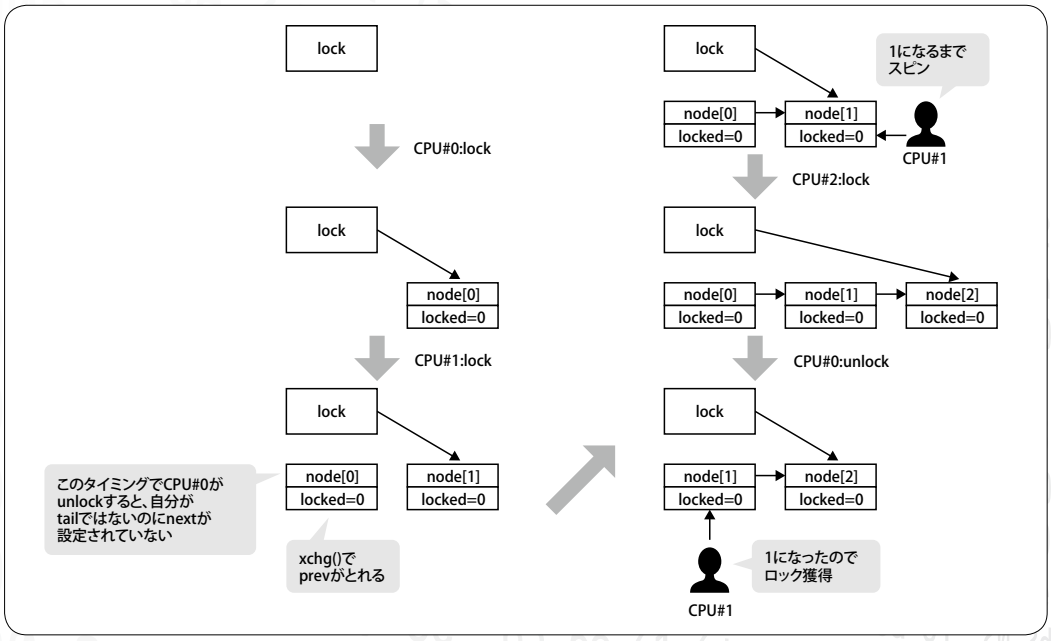
    prev = xchg(lock, node);

    if (prev != NULL) {
        WRITE_ONCE(prev->next, node);
        while (!(smp_load_acquire(&node->locked)))
            cpu_relax();
    }
}

void mcs_unlock(lock) {
    next = node->next;
    if (next == NULL) {
        old = cmpxchg(lock, node, NULL);
        if (old == node)
            return;
        while (!(next = READ_ONCE(node->next)))
            cpu_relax();
    }

    next->locked = 1;
}
```

▼図3　MCS lockをとる様子



spinlock中に割り込みが発生したときに新しくspinlockをとれなくなるのでそれぞれtask、softirq、hardirq、nmi用に4つのnode領域をCPUごとに確保します。

さらにqspinlockのlock構造体では、MCS lockのtailのほかに誰かがロックを獲得していることを示す“locked”と、少なくとも1人ロックを待っていることを示す“pending”のメンバを追加して、ロック待ちが1つのときまではMCS lockを使わない最適化を施しています(**図4**)。具体的に見てみましょう。ここではlockの値を“(tail, pending, locked)”と表現します。この値の初期値は(0,0,0)です。CPU#0はcmpxchg(lock, (0,0,0), (0,0,1))を実行します。このcmpxchgは成功し、CPU#0はロックを獲得します。さらにCPU#1がロックの獲得を試みると、先ほどの“cmpxchg”は失敗します。CPU#1は次にlockのtailとpendingを見てほかにロックを待っているCPUがあるかを確認します。ここではほかに待っているCPUはいないので、lockの“pending”を1に設定して、“lock”が0になるまでスピンします。

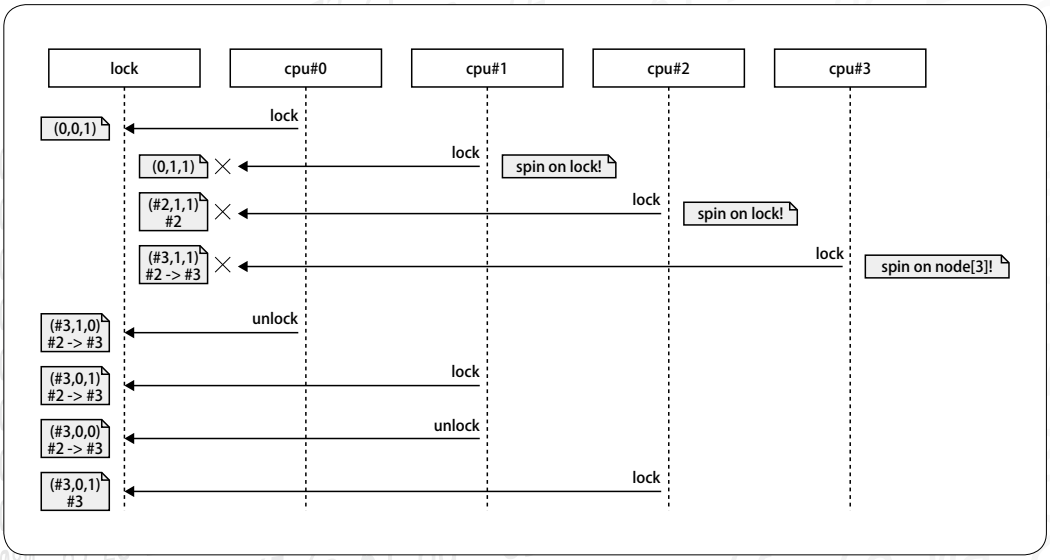
さらにCPU#2がロックをとろうとすると、最初の“cmpxchg()”が失敗し、ほかに待っているCPUがいるので、“tail”を使ってMCS lock同様に自分をqueueにつなぎます。MCS lockではqueueの先頭にくればロックを獲得していましたが、qspinlockでは“locked”を1にしたCPUや“pending”を1にしたCPUがいなくなり、lockの値が“(x,0,0)”になるまでスピンします。それ以降にロックをとろうとするCPUはMCS lockで見たように各自のデータ構造上でスピンすることになります。

アンロック側を見てみましょう。アンロックはlockedをデクリメントすることで実現されます。lockedが0になると、pendingで待っていたCPU#1がスピンを抜けます。CPU#1はpendingを0にして、lockedを1に設定し、ロックを獲得します。CPU#1がアンロックすると、queueの先頭で待っていたCPU#2はMCS lockでやったようにqueueの先頭を渡して、lockedを1に設定してロックを獲得します。

まとめ

今回は、Linuxカーネルにおける最も基本的な排他制御であるspinlockの実装の歴史と、Linux 4.2で導入されたqspinlockの実装について紹介しました。SD

▼図4　qspinlockの概要



July 2016

NO.57

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp
りゅうちてつや　RYUCHI Tetsuya　ryuchi@ryuchi.org

変わらない技術を求めて

今回は、2月末に東京で行った研究会と、4月に開催したシェル勉強会について報告します。

jus研究会 東京大会

■売れ筋書籍で振り返る
オライリー・ジャパンの20年
【講師】渡里 健司（㈱オライリー・ジャパン）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2016年2月27日（土）15:15～16:00
【会場】明星大学 日野キャンパス26号館202教室

2016年度としては7回目となる研究会を、オープンソースカンファレンス2016 Tokyo/Springにて行いました。今回は、IT系出版社として名高いオライリー・ジャパンから渡里さんをお招きし、昨年12月で20周年を迎えた同社の歴史を振り返るセッションを行いました。参加者は53人でした。

■創業期から1990年代

セッションは「オライリー・ジャパン20年の歩みと年度別売上ベスト3発表！」というスライドを見ながら進行しました。創業期である1990年代後半は日本にインターネットが普及し始めた時期です。売れ筋書籍にもそれが反映されていて、ファイアウォール、DNS、sendmailなどネットワーク管理系のものや、CGIでよく使われていたPerl関連のものが目立ちます。この傾向は2000年代前半まで続きます。

また、当時行われたイベントとして、1996年のオープニングパーティー、1998年のPerl Conference、2000年のPerl/Ruby Conferenceなどが紹介されました。

■2000年以降

これに対して、2000年代中盤からはブログなどの隆盛を受けて、Webサービス系の技術を解説した書籍が台頭してきます。書籍の編集形態にも変化があり、それまで主流だった体系的な解説本よりも、Tipsをまとめた体裁のHACKSシリーズがよく出るようになりました。

2000年代後半になるとMake:が創刊され、日本でもMake:イベントが継続的に開催されるようになります。その流れは売れ筋書籍にも影響し、Arduino、Android、iPhoneなどを扱った書籍が上位にランクインします。

そして2012年には近年最大のヒット作『リーダブルコード』が出版され、オライリー・ジャパン年度別売上でも2015年まで4年連続1位を継続中です。

Make:イベントも年々大規模化し、ついに2014年からは東京ビッグサイトで開催されるようになりました。

■ロングセラーはどれか

最後に質疑応答があり、参加者から「長く販売されている書籍はどれか」という質問が出ました。C++やmakeコマンド、viの本などが該当するそうです。Webなど最新技術の本は改訂されたり廃れて絶版になったりすることも多いですが、先述の書籍は内容が普遍的なため、長く売れ続けているようです。このほかにも渡里さんからは貴重な思い出話を多数

聞かせていただき、とても興味深い内容でした。

シェル勉強会

■jus・USP友の会共催シェル勉強会
【日時】2016年4月30日（土）10:00～19:00
【会場】㈱KDDIウェブコミュニケーションズ

本年2月に引き続き、USP友の会と共催で初心者向けにシェルスプリクト関連の勉強会を開催しました。連休のはじめにもかかわらず、東京の会場には37名の参加がありました。大阪と福岡にてネットワークで接続されたサテライト会場からの参加もありました。午前中は初心者向けのセミナーが2本、午後はシェル芸勉強会（全8問）が開催されました。

■午前の部

午前中は次のセミナーがありました。

- 「コマンドライン処理（仮）」
  鳥海秀一（USP友の会）
- 「/etc/rc を読んでみる」
  りゅうちてつや（日本UNIXユーザ会/USP友の会）

最初は鳥海さんのプレゼンです。実際に手を動かしながら実践してみることが大切であるとの説明がありました。今回の内容は前回の勉強会からの続きになるため、まず前回のおさらいをし、そのときの内容のフォローとして、IFS変数の扱い、環境変数とシェル変数の違いなどが説明されました。その後、シェルのコマンドの展開の種類やその動作についての解説が行われ、alias展開や子プロセスとしてシェルを起動したときの動作などを、実際の画面を使いながらひとつひとつ詳しく説明されました。evalコマンドを使ったコマンドラインの展開についても説明があり、これを使ったデモを懇親会のLTで行うことを予告されてセッションが終了しました。

続いて、筆者（りゅうち）より「/etc/rcを読んでみる」というタイトルでプレゼンを行いました。環境はVMwareにインストールされたFreeBSDの/etc/rcとしていましたが、今回は時間の都合もあり、/etc/rcを読むコツとして、ネットワークで公開されている日本語ハンドブックやマニュアルを紹介しました。また、日本では幸いなことに日本語のドキュメントや情報がネットワークにあり、検索したりコミュニティに参加して質問したりしながら勉強するのが良いだろうと紹介しました。最後に少しだけ、/etc/rcの中にあるスクリプトを見てみました。会場からはLinuxとの違いなどの質問がありました。

■午後の部

昼休みを挟んだあとは、上田隆一さんによるシェル芸勉強会が行われました。ここ数回のシェル芸勉強会では、午前中のスクール形式のままでグループ分けをせずに進行していましたが、今回はグループ分けを行い、各グループには詳しい人を配置するようにしました。提示された課題についてグループ内で解き方などについてディスカッションができるようにすることで、より理解を深められるようです。

問題は8問用意されていました。問題が出されると、各自で解答を考えたり、グループ内でのディスカッションも行われたりします。またサテライト会場とはTwitterなどを使って解答などを共有しながら進められました。今回の問題は、中央値を求める、クロスワードを作成する、素数を使った問題、足し算をして特定の値になる並びを見つけるもの、置換をするもの、プロトタイプ宣言を作るものなどパズル系の問題だったようです。

◆ ◆ ◆

サテライト会場との交流は午後の部までですが、そのあとは各会場での懇親会もあったようです。東京では、午後の部の終了後、有志により同じ会場でビアバッシュ形式による懇親会が開催されました。そこでは、自己紹介のほか、短いプレゼン（LT）も行われました。このLTでは、午前中のセミナー講師の鳥海さんが再帰を利用したデモを実行され、たいへん賑わいました。懇親会は、軽食を食べて、楽しく歓談しながら交流を深められます。これからも引き続き活動を行いたいと考えています。SD

Hack For Japan
エンジニアだからこそできる復興への一歩

● Hack For Japan スタッフ
小泉 勝志郎　KOIZUMI Katsushiro
Twitter @koi_zoom1

第55回

GAMATHON ～島ソン第３弾はクラウドファンディングを目指す

この連載の第32回（2014年8月号）でCode for Shiogama/Hack For Japanが2014年5月24～25日に行った「島ソン～浦戸諸島ハッカソン～」を紹介しました。 今回はその島ソンの第3弾となる、2016年3月12日に開催した「GAMATHON」を紹介します。

島ソンの成果物がもう１つ誕生！

「GAMATHON」を紹介する前に、島ソンから生まれた成果物のご報告から。2014年に開催した島ソンで生まれた「渚の妖精ぎばさちゃん（TwitterID：@gibasachan）」はこの連載の中でも何度か紹介してきましたが、このたび同じ島ソンで生まれたプロジェクト「Island Girls」（**図1、2**）がついにリリースされました！

「Island Girls」は宮城県塩竈（しおがま）市浦戸諸島を舞台にしたアドベンチャーゲームで、浦戸諸島の4つの島それぞれが舞台。そしてそれぞれの島ごとに1人ずつ、ヒロインとなる女の子がいます。アドベンチャーゲームの中では浦戸諸島の写真とともに歴史や観光スポットが出てくる作りです。

現在、Android版がGoogle Playにて無料でダウンロード可能です。この機会に「Island Girls」を通じて浦戸諸島のことをぜひ知ってもらえればと思います。

● ダウンロードリンク
http://bit.ly/projectIG

GAMATHON―島ソンの流れをくむアイデアソン

これまで行っていた島ソンは宮城県塩竈市の離島である浦戸諸島が舞台でした。塩竈市の中ではもっとも東日本大震災の爪痕が色濃く残る地域です。今回は島ではなく宮城県塩竈市の本土側での開催です。

実は「islands court」という浦戸諸島と塩竈本土をつなぐ新しい拠点が生まれました。ここはコワーキングスペーススタイルのカフェでネットワーク環境なども充実しています。島ソンのときは、離島だけあって電波の入りに難がありネットワークにつながらなかったので、電波ハックとしてステンレスボウルでパラボラアンテナを作ったりしていましたが今回はその心配はありません！ パラボラアンテナハックも、ないとちょっと寂しいですが（笑）。

GAMATHONは、復興庁の「新しい東北」先導モデル事業で

◆図1　Island Girls



◆図2　Island Girlsプレイ中の画面



第55回 GAMATHON ～島ソン第３弾はクラウドファンディングを目指す

ある「塩竈アイランズネットワーク協議会」の「浦戸サスティナビリティ プロジェクト」の一環として、2016年3月12日に行われました。前回の島ソン2015同様、日帰りのアイデアソンです。

島ソンでは「大学のゼミと連動することでプロジェクトの継続性」を作っていました。今回は別のところに着目しました。それはプロジェクトの実現可能性です。「アイデアソンのみのアイデアソン」と「ハッカソンがセットのアイデアソン」では、アイデアの質に実現可能性の点で差があることが多いと感じていました。何が差になるのか？　それはハッカソンだとプロトタイピングレベルであろうと一応の形をそこに作るのですがアイデアソンではそれがない。そこで今回はまた新しい工夫をしてみました。それは「クラウドファンディングのプロジェクト」として作ること、です。

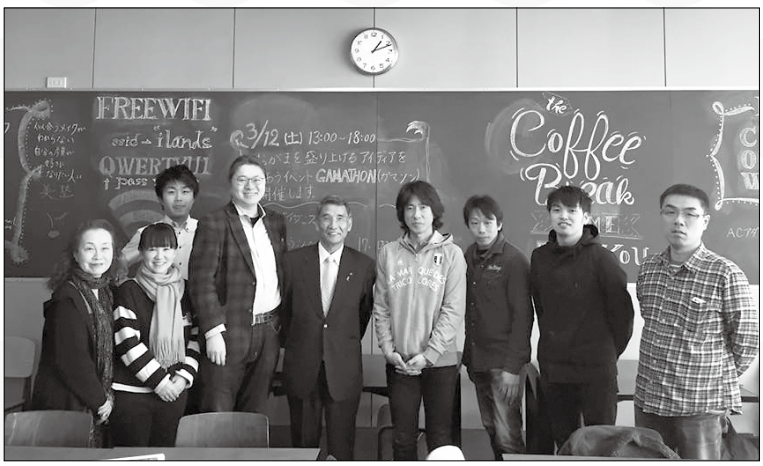
今回のテーマは「塩竈を盛り上げるアイデア」。初回のテーマは「島の課題を解決する」、前回は「島をより楽しむためのしくみを考える」でした。参加者のハードルをさらに下げるために、重くならないテーマで意見も活発に出てきました。

また、今回は島ではなく塩竈市内陸で開催したので、多くの方が見学にいらっしゃいました。塩竈市長佐藤昭さんもいらっしゃいました（**写真1**）。

GAMATHONの流れ

GAMATHONでは島ソンの肝となる「島あるき」はなくなっています。場所が島じゃありませんからね。開催日程が3月12日と震災から5年の節目の日の翌日だったこともあり、まずは震災当時の振り返りからスタート。そしてその中でアイデアワークを行いチーム分け、そして発表の流れです。今回の参加者は学生がいない代わりに、塩竈にお住いのさまざまな方達が参加しました。

◆写真１　集合写真



Column　Code for Shiogama紹介

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。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。Code for ShiogamaにはFacebookページもあります。

https://www.facebook.com/CodeForShiogama

アイデアソン！

アイデアソンではまず塩竈の好きなところを箇条書きしてもらい、次にその紙を隣の人に回して一言コメントを書いてもらうというのを繰り返してアイデアに肉付けをしていきます。

このあとがアイデアソンの肝となるスピードストーミングです。スピードストーミングは**写真2**のように対面し、持ち時間3分で対面しあった2人でのブレインストーミングを行います。持ち時間が終わったら片方の列が1人分ずれて、また別の人と2人でブレインストーミング。この手法の良いところは通常のブレストと異なり、しゃべらないで終わってしまう人が出ないことです。人によって見ているところが異なるので、このスピードストーミングでお互いの情報を交換しながらアイデアを膨らませます。

スピードストーミングが

数セット終わったら、次は自分なりのアイデアをまとめてもらう「アイデアスケッチ」と呼ばれる工程に行きます。ここで各人最低1つは自分のアイデアをまとめます。このアイデアスケッチをグルーピングし、ここでチーム分けを行います。

チームに別れてからはチーム内でディスカッションをして、その内容を模造紙にまとめて発表となります。

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。

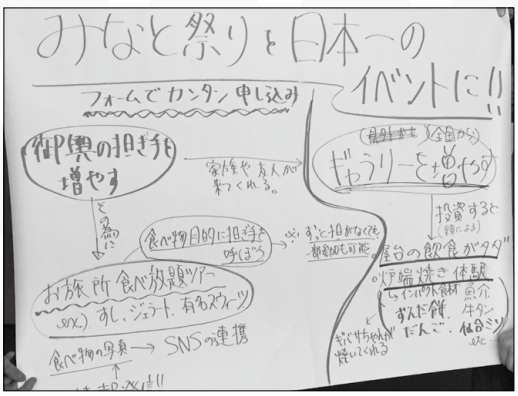
①「みなと祭りを日本一のイベントに!!」

みなと祭りは毎年数万人が訪れる塩竈でも最大のお祭り。しかし、そのみなと祭りでも神輿（みこし）の担ぎ手が年々減少しているため、担ぎ手になるためのハードルを下げる、そして担ぎたくなるためのインセンティブを用意するというものです。担ぎ手のハードルを減らすためにWeb登録を可能にする。実は現在は担ぐタイミングに関係なく、その日の朝に神社に集合するとなっており、昼からなら時間が空く人などに対応できていないんだとか。また、インセンティブには屋台食べ放題などの案が出ていました。これをクラウドファンディングで実現するためのリターンとしても屋台で使えるチケットを用意するという案も発表していました（**写真3**）。

◆**写真2　アイデアワーク**



◆**写真3　みなと祭りを日本一のイベントに!!**



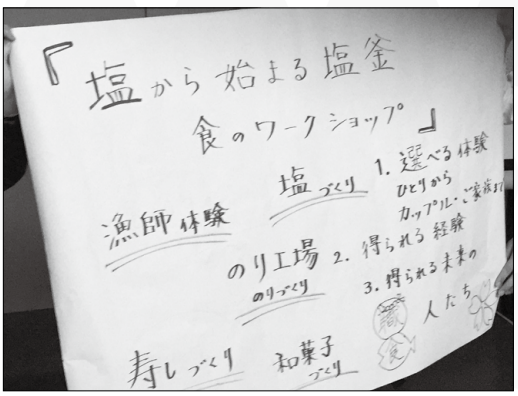
②「塩から始まる塩竈」

食のワークショップについてのアイデアです。塩作り、漁師体験、寿司作りなどを体験するワークショップが考えられていました。今回会場となったislands courtはキッチンスペースをレンタルすることができ、それを生かしたアイデアとなっています（**写真4**）。

③「塩竈から未来のクリエイター発掘」

映像マジックという、映像の編集で行う面白動画のコンテストおよびワークショップを行うというアイデア。塩竈を舞台にした映像マジックのコンテストで、ネット上で塩竈のネタが話題になることを目的としたものです。クラウドファンディングのリターンとしては、塩竈に来たときにガイドをつけて案内するという案が出ていました（**写真5**）。

◆**写真4　塩から始まる塩竈**



④「ふるさと塩竈」

現在塩竈でのふるさと納税が行われていますが、ここにオール塩竈でふるさと納税のリターンを作り、より税収をアップさせる。そして市民に必要なものをそこから作り出していこうというプロジェクト。これはクラウドファンディング相当のものをプロジェクトで作るという、ほかのプロジェクトとはまた別の視点となっています（**写真6**）。

以上4つのアイデアを発表して、GAMATHONは終了となりました。

イベントを終えて

前回は次の言葉で締めていました。「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アや作ったものを、これからも継続していきましょう」。実際にイベント終了後も継続していくプロジェクトが複数生まれました。

前回の島ソンですがアイデアソンに絞ったのもあるのかもしれませんが、実現可能性が昨年の島ソンに比べてちょっと難しいものが多いかなという印象でした。今回のGAMATHONではクラウドファンディングというある程度現実的なプロセスを前提に置いたためか、だいぶ実現可能性が高いものが出てきたと思っています。

最後に、今回のアイデアソンでも多くの方々に協力をいただきました。そして塩竈アイランズネットワーク協議会のみなさん。本当にありがとうございました。

Code for Shiogamaでは復興に携わっているかたと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行きますのでよろしくお願いいたします。SD

Column

islands court について

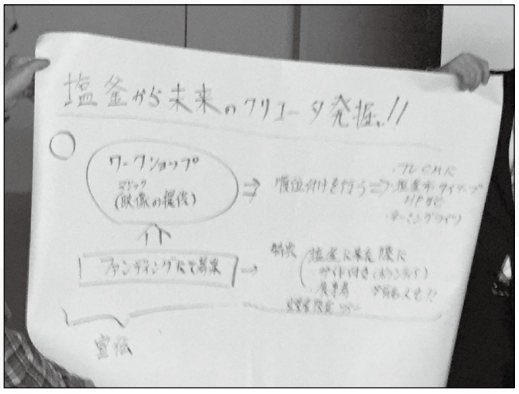
今回GAMATHONの会場となった「islands court」は本塩釜駅近くに新しく作られたコワーキングカフェ。復興団体の集合拠点として、そして浦戸諸島の本土でのアンテナショップとして作られました。

GAMATHONのようなイベントへの貸出はもちろんのこと、「コワーキング」が名に付くとおりコワーキングスペースとしても利用できます。電源・Wi-Fiも完備。300円のコーヒー一杯で2時間利用できるリーズナブルさ。しかも、コーヒーは水出しで作ったこだわりの逸品。

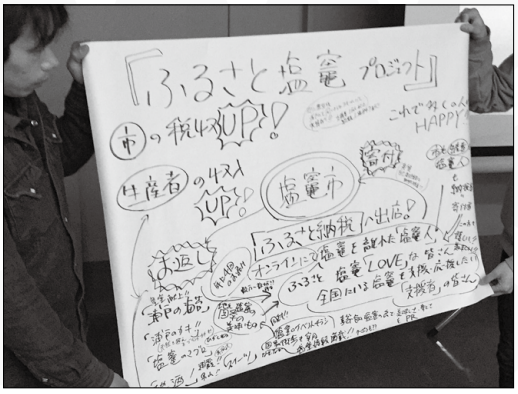
そして、「島のおばんざい」という名前で「浦戸諸島のお母さんたちが作った料理」も楽しめます。メニューは日によって異なるのですが、渚の妖精ぎばさちゃんの元となったギバサ（標準和名：アカモク）をはじめ、メロウドの天ぷら、ふぐの天ぷら、塩辛など。どれも200～500円の低価格。

復興関係者以外にも近所の奥様方が憩いの場として利用する姿も多く見られます。塩竈に来る機会があればぜひお立ち寄りください。

◆写真５　塩竈から未来のクリエイター発掘



◆写真６　ふるさと塩竈



温故知新
IT むかしばなし

第56回

マイコンゲーム
～キャラクタ表示型のゲーム～



速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami



はじめに

初期のマイコンではゲームをすることが1つの目標であり、低性能なマシンに鞭打ってアーケードゲームに近づけようと技術的なさまざまな努力がなされていました。今回は初期のマイコンゲームであるキャラクタ表示型のゲームについてお話ししましょう。



キャラクタ表示型ゲームについて

1970年代、コンソールやテレビモニタのキャラクタスクリーンに文字の組み合わせで表現したゲームキャラクタを使い動作するゲームが登場しました。現在の高度なグラフィック表現によるビジュアル豊かなゲームとは画面の表現力は比較になりませんがゲームとしてのおもしろさや工夫は優れたものがあり、そのアイデアは現在のゲームにも継承されています。

その代表的なものがSFテレビドラマを題材にしてBASICインタプリタ(以下BASIC)で作成されたスタートレックです。最初に表示されるゲームの表題に宇宙船エンタープライズが表示され、未来的なゲームを予感させますが、ゲーム自体はアルファベットと記号が表示されるだけです。とはいえ宇宙シミュレーションとしての想像を豊かにすれば熱中してクリンゴン帝国との戦いに臨むことができました。



最初のキャラクタを表示するマイコン

1978年にBASICが使えディスプレイにキャラクタを表示できるシャープMZ-80Kが発売されました。

MZ-80Kは、カセットテープドライブとディスプレイが一体化したオールイン設計で(**写真1**)、未来のパーソナルコンピュータを予感させる外観と機能を有していました。10型モノクロディスプレイには、横40文字×縦25行の文字を表示できましたが画面に絵を描くようなグラフィック機能はありません。

筆者は、MZ-80Kシリーズの中でも高級感のあるMZ-80Cを入手し、カセットドライブの回転まわりの修理、モニタの調整を行い完動させることに成功しました。



マイコンのゲームプログラミング

MZ-80Kシリーズでは、キャラクタの表示は簡単で描画スピードもある程度確保できるので、BASICでもなんとかリアルな動作が可能なゲームを作成できました。ただし、PRINT命令ではインタプリタの遅さが表れてしまうので、高速化のためにテキストVRAMに直接キャラクタデータを書き込むテクニックを使います。

BASICではメモリにデータを書き込むコマンドはPOKEで、

```
POKE メモリアドレス, ↵
書き込む8ビットデータ
```

のようになります。MZ-80Kの場合は、テキストVRAMの開始アドレスが0xD000(＝53,248)、書き込むデータはASCIIコードではなく、**図1**に示すようなディスプレイコードです。画面左上の座標を(0, 0)として、座標(X, Y)に立った人のキャラクタを表示するためには、次のようにディスプレイコード202(＝0xCA)を書き込みます。

```
POKE 53248+X+40*Y, 202
```

ディスプレイコード表には、一般的なASCIIキャラクタ以外にゲームに使えそうなキャラクタが多く存在しました。MZ-80のゲームはこれらのキャラクタをうまく利用したものが多くありました。

また、マイコン雑誌[注1]に掲載されたBASICソースリスト内の文字列にキーボードに割り振られていない特殊なキャラクタが含まれているとき、その行を入力して、特殊キャラクタが入る場所を決め、その部分に特殊キャラクタを表示させるようにPOKE文を実行するのです。表示されたらその行にカーソル移動してCRキー(今のEnterキー)をたたくことでBASICソース行として認識して入力するテクニックなどが使われました。



MZ-80Kの代表的なゲーム

MZ-80Kの特殊キャラクタを活用してゲームが作成しやすかったため、数多くのゲームが登場しました。「地底最大の作戦」は工学社のマイコン雑誌『I/O』1980年7月号に掲載されました。**写真1**のディスプレイに映っているのがこのゲーム画面です。人間のキャラクタ(コード0xCA～0xCD)が穴を掘り進み、蛇キャラクタ(コード0xDF)を穴に落とし捕まえてスコアを積み上げていきます。1983年に登場しApple IIで動作したロードランナーをシンプルにしたようなアクションパズルゲームで、時間が経ってから発生するWARNモードまで、どこまで準備するか考える戦略的な要素も併せ持っています。1行命令を並べたマルチステートメントを駆使していますが、プログラムソース行サイズはわずか119行で、マシン語部分は22byteだけです。

電波新聞社のマイコン誌『マイコン』に掲載され、ハドソン社(ゲーム作者が創業者です)からも発売された「アルデバランパート1」はパート3までつながる物語性が豊かなゲームで、そのパート1が掲載されました。宇宙船内でバクテリアが異常繁殖をして倒れている人を助け出して、バクテリアが蔓延しないように壁で塞ぐことが目標になります。時間制限がある「倉庫番」的なゲームで、増殖するバクテリアからいかに時間内で人を助け出すかの醍醐味とパズル性を兼ね備えたよく考えられたゲームです。アルデバランもBASICの行数が129行、7byteのマシン語部分でリアルなキー操作を実現しています。

▼写真1　MZ-80K





おわりに

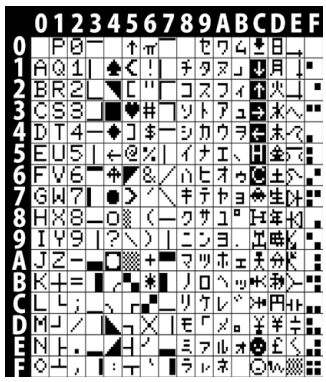
キャラクタ表示型のゲームは、その後登場した富士通FM-8やNECのPC-8801が備える精細なグラフィックを活用するゲームに代わっていき、一部ではPCG[注2]を活用して表現力を高めたゲームの作成が継続されました。

現在のゲームは、高度なグラフィックによるハリウッド映画のようなビジュアル感の高いもので溢れています。しかし、アイディア豊かな単純なキャラクタ表示型のゲームでも十分楽しめますし、基本はゲーム自体のアイディアです。プログラムの学びにはゲーム作成が有効なので、このようなシンプルなゲームの創造を指導することも効果的かもしれません。昔、マイコンを使いこなし上達する目的のひとつがゲームで遊ぶことであったのですから。SD

注1) 1980年前後、工学社「I/O」、アスキー出版社「月刊アスキー」、廣済堂「RAM」、電波新聞社「マイコン」が毎月18日に発売される月刊誌で、マイコン4大雑誌と呼ばれた。その流れでSoftwareDesign誌も18日発売なのかもしれませんね(笑)。

注2) プログラマブル・キャラクタ・ジェネレータ。キャラクタパターンを書き換えることで独自のキャラクタを生成してセミグラフィック処理を実現する。

▼図1　ディスプレイコード一覧





Topic

トレンドマイクロ、「2016年第1四半期セキュリティラウンドアップ」を公開

トレンドマイクロ㈱は5月25日、日本国内および海外のセキュリティ動向を分析した報告書「2016年第1四半期セキュリティラウンドアップ」を公開した（**http://www.go-tm.jp/sr2016q1**）。ハイライトは次のとおり。

・国内ランサムウェア被害報告件数8.7倍に

2015年に引き続きランサムウェアの感染被害が拡大、不正なメール経由で感染する手口が多くみられた。3月にはバックドア機能を備えた新型ランサムウェア「SAMAS」や「SAMSAM」が確認されている。

・オンライン銀行詐欺ツールの国内検出台数1.9倍に

2015年末より確認。既存のバックドア型不正プログラムを改造した「ROVNIX」と「BEBLOH」が急増し、1月～3月の検出台数の約8割を占めている。海外でも同型の「QAKBOT」が確認されている。

・送金指示メール攻撃「BEC」の新ツールを確認

2014年ごろより、「BEC（Business E-mail Compromise）」の被害が海外を中心に拡大。BECは攻撃者がキーロガーなどを介して標的企業内の情報を把握し、取引先とのメールに介入したり、企業の経営層になりすまして不正な送金を指示する詐欺の手口。

CONTACT
トレンドマイクロ㈱ URL http://www.trendmicro.co.jp

Report

「AWS Summit 2016」、イベントレポート

アマゾンウェブサービスジャパン㈱は、6月1日～3日の3日間、グランドプリンスホテル新高輪（東京都港区）で、同社の最新技術や導入事例、活用方法などを紹介するカンファレンスを開催した。

第1日目はおもにエンタープライズとパートナー企業を参加対象としたもの。第2日目以降は、一般向けと開発者向けで100を越えるセッションが開催された。2つのセッションをレポートする。

○攻めのITを実現するクラウド活用

1日目のEnterprise Dayにてアマゾンウェブサービスジャパン㈱技術統括本部技術統括本部長の岡嵜禎氏が行ったのは、企業におけるクラウド導入の真価と最新事情に関するセッション。

氏によると、最近ではコストの低減目的というより、他社との競争・差別化のためにクラウドを導入する企業が増えており、さらにそのコアとなるアプリは企業での内製化が進んでいるようだ。クラウドを導入することで得られる最大のものはビジネス展開のスピードとのこと。変化の速いWebの世界ではまず最小構成のMVP（Minimum Viable Product）をリリースし、ユーザのフィードバックにすばやく対応しながらアプリを成長させる「リーンスタートアップ」の考え方が非常に適しており、その実現にはクラウドが必要不可欠となるという。また最新の状況としては、クラウドサービスを使うことを前提としてアプリを設計する「クラウドネイティブアーキテクチャ」、サーバを意識せずに開発する「サーバレスアーキテクチャ」が浸透しつつあるそうだ。

○Day2 キーノート

第2日目のキーノートは同社代表取締役社長の長崎忠雄氏が行った。氏は5年前を振り返り、同社のテクノロジの進化を強調した。今や顧客はクラウド導入で悩むことはなくなり、自社のビジネスをいかに推進していくのか、その実装が問題になっている現状をアピールした。AWSのサービス進化の原動力は「お客様の声やフィードバックからイノベーションが生まれる」とし、その実例として、ゲストスピーカーに㈱ゲオホールディングスの末延寛和氏、日本電産㈱の佐藤年成氏、㈱freeeの佐々木大輔氏が壇上に上がり、各社での導入事例を紹介した。そのほかには、クラウドへの移行のための「スノーボール」システムなどエンジニアにとっても興味深い最新技術の数々をあらためて公開し、同社の目覚ましい躍進ぶりを強調した。



▲アマゾンウェブサービスジャパン㈱長崎忠雄氏がスノーボールを紹介

CONTACT
アマゾンウェブサービスジャパン㈱
URL https://aws.amazon.com/jp/

Service

KDDI、「au OPEN DEVICE DEVELOPER SITE」開設

KDDI㈱は4月22日、IoTデバイスメーカーや開発者を対象とした技術支援サイト「au OPEN DEVICE DEVELOPER SITE」を開設した（**http://open-dev.kddi.com/home**）。

本サイトではIoT市場の拡大に備え、KDDIネットワーク（現在はLTEのみが対象）との接続性検証（IOT：Inter-Operability Testing）の実施をはじめ、KDDIネットワークへの接続に必要な情報、評価環境、接続確認済み機器一覧、相互接続性試験に関するFAQ・手順などが公開される。また、検証体制がないデバイスメーカーや開発者向けに、新たな製品や技術開発に関する情報も提供していく。一部の情報の閲覧については、会員登録が必要となっている。



▲au OPEN DEVICE DEVELOPER SITEトップページ

CONTACT
KDDI㈱ URL http://www.kddi.com

Software

レッドハット、「Red Hat Enterprise Linux 6.8」提供開始

レッドハット㈱は5月27日、Red Hat Enterprise Linux（以下RHEL）6の最新バージョン、「6.8」を提供開始した。

RHEL 7が発表されてから2年経過している現在でも、約6年間の実績を誇るRHEL 6は多くの組織で採用されている。「6.8」の特徴は次のとおり。

- VPNのセキュリティを強化するため、openswanに代わりVPNプロトコルの実装にlibreswanを採用
- 認証サービス「System Security Services Daemon（SSSD）」を強化
- システムアーカイビングツール「Relax-and-Recover」を導入
- yumのパッケージ検索プロセスにインテリジェンス機能を追加
- 「dmstats」を通じてストレージ使用量とパフォーマンスに対する可視性を向上
- Scalable File System Add-onが、最大300TBのxfsファイルシステムに対応
- 従来のワークロードをコンテナベースのアプリ（RHEL 7/Atomic Hostへのデプロイに適したもの）に移行できる最新のRHEL 6.8ベースイメージの提供を開始

CONTACT
レッドハット㈱ URL https://www.redhat.com/ja/global/japan

Service

F5ネットワークスジャパン、「F5 SOCパートナープログラム」を発表

F5ネットワークスジャパン合同会社は6月2日、同社のWAF（Webアプリケーションファイアウォール）製品「F5 BIG-IP Application Security Manager」の提案、導入、運用を支援する「F5 SOCパートナープログラム」を発表した。

現在、Webアプリケーションに対する攻撃は増加を続けており、WAFの採用が必須となりつつある。しかしながら、エンドユーザにおいてはWAFシステム導入の判断やポイントがわからないなどの課題が、システムインテグレータ（以下SIer）にはWAFに関する知識や運用経験の不足、WAFの提案体制構築における膨大なコストの発生などの課題があった。さらに、エンドユーザおよびSIerともに、24時間365日の運用監視の体制確保が困難な状況だった。

本プログラムでは、SOC（Security Operation Center）パートナーはエンドユーザやSIerに向けて、WAFの提案、導入に関するノウハウや24時間365日のセキュリティ運用監視体制など、さまざまな選択肢を提供する。プログラム開始時でのSOCパートナーは、伊藤忠テクノソリューションズ㈱、SCSK㈱、NRIセキュアテクノロジーズ㈱、三井物産セキュアディレクション㈱の4社。

CONTACT
F5ネットワークスジャパン合同会社 URL https://f5.com/jp

Service

グローバルセキュリティエキスパート、「EC-Council セキュリティエンジニア養成講座」開講

グローバルセキュリティエキスパート㈱は5月19日、米EC-Council International社の「EC-Councilセキュリティエンジニア養成講座」を国内で独占販売することを発表した。

EC-Councilセキュリティエンジニア養成講座は、Certified Ethical Hacker（認定ホワイトハッカー、以降CEH）をはじめとした情報セキュリティエンジニア向け認定トレーニングおよび認定資格試験など、日本初上陸のカリキュラムとなっている。認定トレーニングは、企業や政府などへ行われている実際のサイバー攻撃手法を学ぶ「CEH」、これら攻撃手法を理解したうえでの対処を学ぶ「Computer Hacking Forensic Investigator（コンピューターフォレンジック調査員、以降CHFI）」、防御としてのプログラミング手法を解説する「EC-Council Certified Secure Programmer（認定セキュアプログラマ、以降ECSP）」などで構成され、これらをEC-Councilが提供する仮想環境で実践的に学ぶことができる。

CEH（5月より開講）：5日間／498,000円（予価）

最新のセキュリティ脅威、高度の攻撃ベクトルと、最新のハッキングの技術、手法、ツール、巧みな技、情報セキュリティ対策のリアルタイムでの実演／実用に重点を置いた、エシカル・ハッキング／情報システム・セキュリティ監査の統合型プログラム。

CHFI（7月より開講）：5日間／498,000円（予価）

高度なセキュリティ研修プログラムにより、侵入者の足跡を見つけ出し、告発に必要な証拠を正しく集めるスキルを習得できる。

ECSP（10月より開講）：3日間／298,000円（予価）

Javaのセキュリティ機能、セキュリティ・ポリシー、長所と短所をカバーした実践的トレーニングを行う統合型コース。開発者が安全で堅牢なJavaアプリのコード記載を理解する手助けをするとともに、Javaのセキュア開発のさまざまな側面に関する高度な知識を提供。

CONTACT
グローバルセキュリティエキスパート㈱
URL http://www.gsx.co.jp

Event

7月20日、「JAIPA Cloud Conference 2016」開催

7月20日、コクヨホール（東京都港区）にてクラウド事業者・クラウド関連事業者の経営者向けビジネスイベント「JAIPA Cloud Conference 2016」が開催される。

同イベントは一般社団法人日本インターネットプロバイダー協会（JAIPA）のクラウド部会が主催するもので、「JAIPA加盟および関連ビジネスの経営者・社員の気づきを提供する」ことと「クラウドビジネスのエコシステムに寄与する」ことを目的に毎年開催されている。今年で4回目で、毎回、多くのクラウドサービスプロバイダ（IaaS/PaaS/SaaS）、システムインテグレータ、ソリューションベンダーが参加している。

今回は次のようなセッションが予定されている。

- IoT、CPS時代のセキュリティガバナンス（㈱ディアイティ　河野省二氏）
- Googleスケールの機械学習テクノロジー（グーグル㈱　佐藤一憲氏）
- AIとフィジカルシステムを支えるクラウドと法（駒澤綜合法律事務所／㈱ITリサーチ・アート　高橋郁夫氏）
- 工場・製品・重要インフラの安全を守るIoTセキュリティ技術の開発に向けて（NTTコミュニケーションズ㈱　境野哲氏）
- パネルディスカッション：IoTによるビジネス変革

開催概要は次の表のとおり。5月23日より同イベントの公式サイトから参加申し込みが開始されている。

●開催概要

名称	JAIPA Cloud Conference 2016（略称：クラコン 2016）
URL	http://www.cloudconference.jaipa.or.jp
開催日時	2016年7月20日（水） 開会：9:50（開場：9:30） 閉会：17:55 懇親会：18:30～
会場	コクヨホール・多目的ホール 〒108-8710　東京都港区港南1丁目8番35号 http://www.kokuyo.co.jp/com/hall/
参加費用	無料・事前申し込み制

CONTACT
一般社団法人日本インターネットプロバイダー協会 クラウド部会
URL https://www.jaipa.or.jp/active/cloud/

6月25日、技術書オンリーイベント「技術書典」開催

6月25日、秋葉原通運会館（東京都千代田区）にて技術書限定の同人誌即売会「技術書典」が開催される。

本イベントで扱われる「技術書」は、ITや機械工作とその周辺領域について書いた本を指し、ソフトウェア、ハードウェア、開発環境、コンピュータサイエンスからその他科学・工学全般などのジャンルを対象としている。

参加予定のサークル数は個人が50で、アスキードワンゴ、オライリー・ジャパン、翔泳社、インプレスなどの法人が8。主催は同人誌サークル「TechBooster」と技術系電子書籍の制作・販売を行う「達人出版会」。入場開始は11時で、終了は17時。一般参加は無料となっている。



▲公式サイト

CONTACT
技術書典 URL https://techbookfest.org

7月29～30日、「オープンソースカンファレンス2016 Kyoto」開催

7月29～30日、京都リサーチパーク（京都府京都市下京区）にて「オープンソースカンファレンス2016 Kyoto」が開催される。

オープンソースカンファレンス（以下OSC）は、オープンソースに関する最新情報の提供を目的とし、企業・団体によるブース展示・セミナーが各種行われる。京都での開催は今年で10回目。去年までは「OSC Kansai@Kyoto」という名称だった本イベントだが、今年からは「OSC Kyoto」となる。

今年の基調講演は、「Unix考古学」の著者である藤田昭人氏が1994年リリースの4.4BSDの開発プロジェクトに参加した日本人プログラマとして、その活動の顛末を語る『平成生まれのための「Unix考古学」～GitHubなしでどうやって開発していたの？～』。2日目の14:00から行われる。

●開催概要

日程	7月29日（金）10:00～17:00／7月30日（土）10:00～17:30
会場	京都リサーチパーク（京都府京都市下京区）
参加費	無料
主催	オープンソースカンファレンス実行委員会
企画運営	㈱びぎねっと

CONTACT
オープンソースカンファレンス2016 Kyoto
URL http://www.ospn.jp/osc2016-kyoto

7月13～15日、「LinuxCon＋ContainerCon Japan 2016」開催

7月13～15日にかけて、ホテル椿山荘東京（東京都文京区）にて「LinuxCon＋ContainerCon Japan 2016」が開催される。

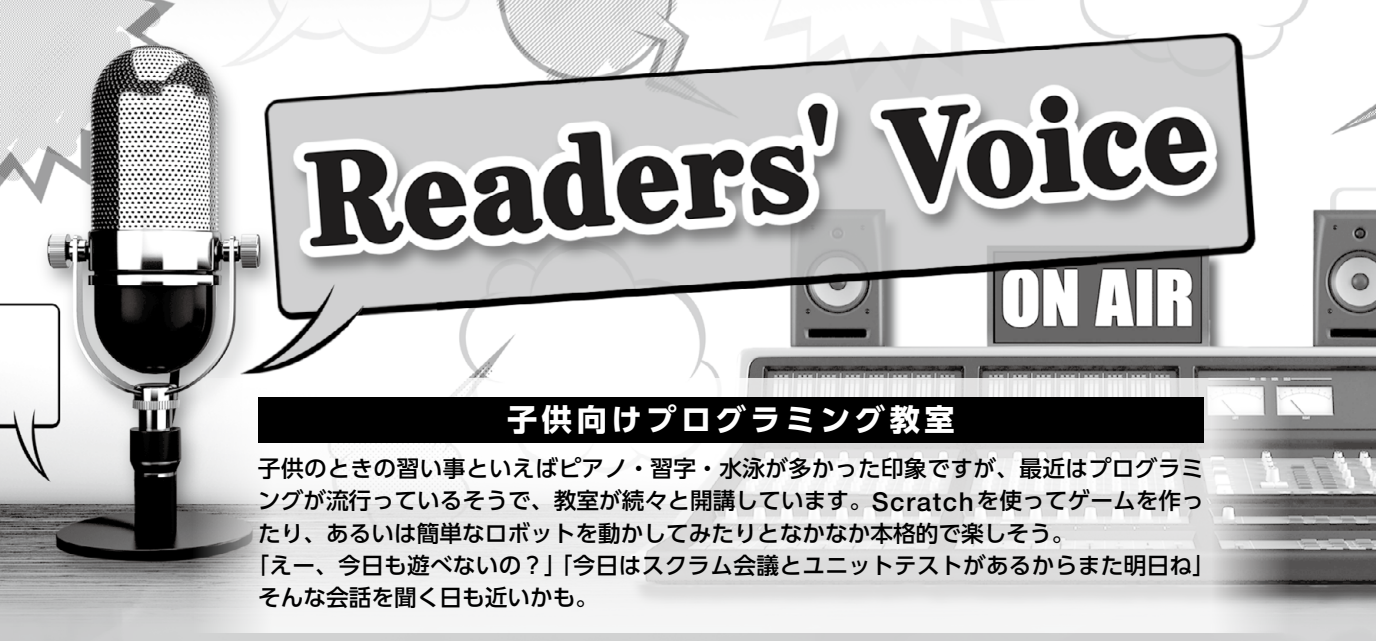
LinuxCon Japanは、Linuxのコア開発者、管理者、ユーザ、コミュニティマネージャ、業界専門家が一堂に会する。今年はLinus Torvalds氏が来日するほか、基調講演では新たにHyperledgerプロジェクトのエグゼクティブディレクターに就任したBrian Behlendorf氏などが登壇する。

また今年は、クラウドコンピューティングやLinuxコンテナ分野に特化した「ContainerCon」、自動車関連の専門家やオープンソースエキスパートが集う「Automotive Linux Summit」、世界的なSDN/NFVのエコシステムを対象としたミニサミット「Open Networking Mini-Summit」も同会場で併設される。参加登録は**http://events.linuxfoundation.jp/events/linuxcon-japan**から。

●登録料

準早期登録料金（6月1日～6月22日）	US$400
通常登録料金（6月23日以降）	US$475
学生登録料金（要学生証）	US$125

CONTACT
The Linux Foundation
URL http://www.linuxfoundation.jp



子供向けプログラミング教室

子供のときの習い事といえばピアノ・習字・水泳が多かった印象ですが、最近はプログラミングが流行っているそうで、教室が続々と開講しています。Scratchを使ってゲームを作ったり、あるいは簡単なロボットを動かしてみたりとなかなか本格的で楽しそう。
「えー、今日も遊べないの？」「今日はスクラム会議とユニットテストがあるからまた明日ね」そんな会話を聞く日も近いかも。

2016年5月号について、たくさんの声が届きました。

第1特集 Vim［実戦］入門

Vimのエキスパート5人が、Vim超入門から高速編集術、正規表現、GitHub連携、Vimの最新事情・Vim開発の舞台ウラまで解説しました。章末には新人向けにコマンドを厳選した「Vimチートシート」も付属した、「実戦」を意識した大特集でした。

基本機能だけでも使いきれていないのが現実です。　下平さん／東京都

導入後の応用的な使い方がまとまっていたので良かった。　谷口さん／埼玉県

意外と知らないことがあって役に立った。　kamocさん／静岡県

普段のテキスト編集はEmacsですが、サーバ上ではviを使っています。viは基本的な個所を編集するだけしか使用してなかったのですが、今回の特集で便利なしくみがたくさんあることに気づきました。とくにピリオドの繰り返しはかなり便利ですね。
今井さん／千葉県

使用歴だけは長い、万年初心者のVimmerにはうれしい記事でした。とくに正規表現は知らないことが多く、とても参考になりました。
犬棟梁さん／埼玉県

普段から使ってはいるが、こんな機能まであったのは知らなかったという声が多かったです。自分にとって必要な機能を知り、必要な分だけ拡張することで、手に馴染むエディタになっていくことでしょう。

第2特集 安定のUbuntu16.04の新機能

最新のLTS（安定版）であるUbuntu 16.04について、Ubuntuの基本から解説し、最新機能まで幅広く紹介しました。また、デスクトップ用途としてUbuntuの派生フレーバーを紹介し、サーバ用途としてUbuntu Serverの独自機能を深掘りしました。

自宅PCにUbuntu16.04をインストールしたばかりだったので、とても参考になりました。　山下さん／東京都

Xubuntuを利用しているので、Xfceの開発状況はとても気にはなりますが、現在でも基本的な機能には個人的に十分満足しているのでセキュリティさえ担保されるのであれば、開発自体の進捗はそれほど重要ではないのではないかと思っています。OSが使いたいわけではなく、その上で動作するアプリが快適に動けば十分なので。
出玉のタマさん／大阪府

ZFSが追加されていることを初めて知った。　psiさん／東京都

Ubuntuの動向が追いきれていなかったので参考になった。　時武さん／東京都

GNOMEに戻ることができて、やっとましなディストリビューションに戻った感じがします。LTSはほかのディストリビューションには少ないので貴重だと思います。　みふさん／神奈川県

Linuxの中では、やはりUbuntuを選ぶ読者の方が多いようです。ただ、オープンソースでの開発ということでセキュリティが気になる人も多いようで、LTSの発表は毎年注目を集めています。

特別付録　SORACOM Air SD Special Version

IoT向けのSIMカード「SORACOM Air」の、Software Design限定特別バージョンです。アクティベーション期間は1カ

5月号のプレゼント当選者は、次の皆さまです

①Raspberry Pi 3 Model B
友利奈緒様（愛知県）

②Power Plus 3 13400mAh DANBOARD version
鄭君様（埼玉県）、八百板博文様（埼玉県）

③microSDカード TS16GUSDHC10V
竹田寛子様（東京都）、tomo様（神奈川県）

④『RDB技術者のためのNoSQLガイド』
橿山公一様（埼玉県）、古角浩太郎様（岡山県）

⑤『アルゴリズムの基本』
高橋哲史様（神奈川県）、田代博嗣様（佐賀県）

⑥『Swiftポケットリファレンス』
くるるたん様（大阪府）、ももんが様（静岡県）

⑦『はじめてのLisp関数型プログラミング』
福田昌弘様（埼玉県）、野口菅子様（福岡県）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

月ですが、通常のSORACOM Airと同じ機能を利用できます。みなさんは、いったいどのように使われたのでしょうか。

いろいろやりたくなりました。Arduinoを引っ張り出してきます。
tomato360さん／東京都

白状します。SORACOM Air SD Special Version目当てで、久々（数年ぶり）に購読しました。
mkouheiさん／東京都

格安SIMが話題になっていますが、いざ購入となると種類も多く、どれにするか決めるのもたいへんです。付録になっていると、とりあえず使ってみてから考えられるのが助かりました。
中村さん／大阪府

発売前から大きく注目されていたSIMカードの特別付録。従量課金制ではあるものの初期費用が0円ということで、IoTをとりあえず始めてみるには絶好のものと言えるでしょう。

巻頭特集　特別SIMで始めよう！SORACOMでわかるIoT

特別付録に関連した、IoT入門者向けの特集です。SORACOM Airの使い方から、APIを使った通信のコントロール、Yahoo!の「myThings」との連携、最近始まった「さくらのIoT Platform」の紹介まで、最近のIoT事情を一望できる特集でした。

自販機や監視カメラなど活用が期待されますね！　さかなかなさん／大阪府

イマドキで、ほかのサービスと連携していろいろできるようになっている点が興味深かった。　ほまれさん／千葉県

eSIMは勉強になりました。
風間さん／新潟県

生活をちょっと便利にするガジェットを作ったり、あるいはビジネスとして展開するためのサービスを作ったり……、IoTの持つ可能性を感じられる記事でした。

一般記事　フリーで始めるサーバのセキュリティチェック【前編】

セキュリティベンダに頼むと予算やスケジュールの面でコストがかかる「脆弱性診断」。これをフリーソフトで行ってしまおう、というのが本短期連載。前編では、nmapを使ったポートスキャンについて解説しました。

初めて知る機能の紹介もあり参考になった。とくにバージョンスキャンは便利そうだが、ここまで調べられるのはちょっとこわい……。
にゃーんさん／愛知県

とても良い記事ですが、こういう情報はある種危険ではありますね。
Tayuさん／千葉県

nseスクリプトについては知らなかったため、たいへん参考になりました。ただ、個別に実行するのは敷居が高いので何か上側に面倒を見てくれる層がほしいと思いました。
sho-hさん／島根県

手元でできる脆弱性診断ということで、興味を持った読者が多いようです。逆に、「こんなことまでできてしまうのか」と不安になってしまった人もいるようで、使いようでは毒にも薬にもなってしまうのが怖いところですね。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
August 2016

2016年8月号
定価（本体1,220円+税）
192ページ

7月16日発売

［第1特集］プルリクしていますか?

GitHubさいしょの一歩

はじめてのPull Requestから、チーム導入へ

デザイナーさんも使いはじめているGit/GitHubですが、それでも初心者にとって少しハードルが高く、プルリク（Pull Request）はドキドキします。GitHubを使用するにあたり、まずは基本用語の概念をおさえ、安心してPull Requestを出せるようになるための必要最小限の使い方に的を絞って解説をします。そして開発の現場で、みんながどうGitHubを使っているのか、すでに導入+活用している企業の皆さんをレポートします。

［第2特集］OSを安全にアップグレードしていますか?

apt-getとyumを通して OSとアプリの運用・管理を考える

案外知らなかったパッケージの導入とそのしくみ

システム管理者にとってLinuxディストリビューションのパッケージ導入は基本スキルです。apt-get（Debian系）やyum（Red Hat系）を使うのは簡単ですが、トラブルが発生したときに、どう対処したらよいのか、そんな問題を解決する実践テクニックを解説します。

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「セキュリティ実践の基本定石」（第34回）は都合によりお休みさせていただきます。

SD Staff Room

●「ザ!鉄腕!DASH」が好きだ。0円食堂は、OSSのツールを探すような楽しさがある。Dash島はゼロからインフラを作る、それってインフラエンジニアじゃん。世界一うまいラーメンを作れるか?、まさソフトウェア設計と開発じゃないか。などと日曜の夜に本誌の企画を考えるのが習慣である。（本）

●自宅庭で鉢植えのミニトマトに支柱立てをしていたところ、すぐ隣の椿に大量発生した茶毒蛾の幼虫を発見。知らずに触れていたらと思い、鳥肌が立つ。遠くから高枝切り鋏で発生枝を切り、まとめてバーナーで焼却。残りの樹にもオルトランをスプレーした。ヤブ蚊、なめくじ、青虫、夜盗虫の季節。ご注意を。（幕）

●5月のGoogle I/Oでガジェット好きの自分が気になったのは、Chrome OSにPlay Storeが導入されてAndroidアプリが使えるようになるという発表。Chromebookがなかなか流行らないけど、Androidスマホユーザがノート PCっぽいものを欲しくなったときに候補にあがるようになるのかな?（キ）

●以前の本欄で、今年は継続して大河ドラマを見ていると書きましたが、ついに1回見逃してしまいました。ここで話についていけなくなると今後見る気がなくなります。そこで、その回も見ていた妻から話の要約を聞きました。次回見たときにちゃんと話がつながるのか、妻の要約の力量が問われます。（よし）

●帰省ついでに、去年開園した「エキスポシティ」に行ってきました。万博公園のすぐ近くにある大型施設で、ポケモンセンター、ガンダムスクエア、IMAX映画館までありました。おまけに日本最大級の観覧車まであるのですが、行ったときはまだ建造中……。でも作りかけの観覧車を見たのは初めてかも。（な）

●編集部から本社ビルの間を通る小路にある、とあるマンションの玄関アプローチで日向ぼっこをしているハチワレにゃんこ。見かけるたびに挨拶をしていたので顔を覚えてくれたのか、歩いていると向こうからニャ~と呼びかけてくれるようになりました。うふり。まだ2m以内には近寄らせてくれないけれど、いつかきっと!（ま）

本誌に記載の商品名などは、一般に各メーカーの登録商標または商標です。 © 2016 技術評論社

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2016年7月号

発行日
2016年7月18日

●発行人
片岡 巌

●編集人
池本公平

●編集
金田富士男
菊池 猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱



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



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