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
6
2016
2016年6月18日発行
毎月1回18日発行
通巻374号
（発刊308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

》bash自由自在　》MySQLを始めよう!

Special Feature 1

速く堅実に
使いこなす
ための

bash 再入門

エンジニアの道具を磨こう

速報 Bash on Windows

Special Feature 2

RDBの学び方

MySQLを武器にSQLを始めよう!

ソフトウェア開発の
基礎の基礎

結城 浩
再発見の発想法

増井 俊之
コロンブス日和

宮原 徹
オープンソース
放浪記

すずきひろのぶ
セキュリティ実践の
基本定石

小飼 弾
書いて覚える
Swift入門

mattn
Vimの細道

好評連載

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）

6

2016

［目次］

Software Design

Special Feature 1 第1特集

速く堅実に使いこなすための

bash再入門

エンジニアの道具を磨こう

contents

6
2016
［目次］
Software Design

Special Feature 2 第2特集

RDBの学び方

MySQLを武器にSQLを始めよう! ソフトウェア開発の基礎の基礎

Extra Feature 一般記事

Catch up trend

àla carte アラカルト

contents

Column

Development

OS/Network

[ロゴデザイン]
デザイン集合ゼブラ+坂井 哲也
[表紙デザイン]
藤井 耕志(Re:D Co.)
[表紙写真]
Martin Barraud / 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

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。

BOOKS INFOMATION

ISBN978-4-7741-8080-9
A5判／256ページ
定価（本体1980円+税）

成果を出し続けるための 王道 SEO対策 実践講座

●鈴木良治 著

この1冊で、WebサイトすべてのSEO対策ができる！

本書は、サーバ選択やサイト設計から、Search Consoleによる運用管理・ペナルティ対策までを網羅した、総合的かつ実践的なSEO対策マニュアルです。現在主流となっている内部対策を中心に解説しつつ、伝統的なSEO対策の中からも、今でも効果がある方法だけを厳選して紹介します。

ISBN978-4-7741-8072-4
B5判／192ページ
定価（本体1980円+税）

Web文章の「書き方」入門教室

5つのステップで
「読まれる→伝わる」文章が書ける！

●志鎌真奈美 著

「うーん、商品紹介文、
どうやって書けばいいんだろう？」
そんなお悩みを持つ方はいらっしゃいませんか？

本書は、Webサイトに掲載する文章の書き方を、5つのステップでしっかり解説。自分の商品や会社を分析するところから、順を追ってわかりやすく解説します。経験豊富なプロの講師が、最終的な「集客」「販売」の目的に、最適化された文章を書く方法を伝授します。
Web文章を書きたいすべての人におすすめの1冊です！

ISBN978-4-7741-8094-6
A5判／352ページ
定価（本体2880円+税）

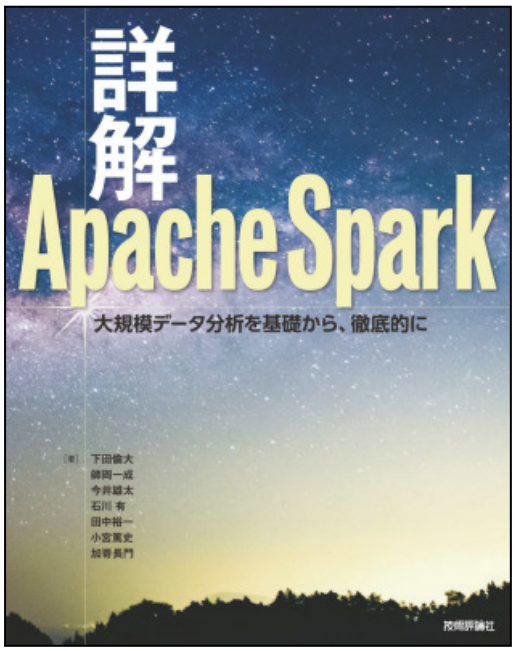
C#プログラマーのための

基礎からわかるLINQマジック！

●山本康彦 著

使わなければ損をする、実は超便利なC#の開発技！

なかなか現場にはそれを正しく教えてくれる人が見当たらないC#の重要技術「LINQ」ですが、実は、使わなかったことを後に後悔すること必至の、効率向上に寄与すること間違いなしの優れものなのです。LINQのエキスパートが、本書でその真髄を伝授いたします。



ISBN978-4-7741-8124-0
B5変形判／352ページ
定価（本体3600円+税）

詳解Apache Spark

●下田倫大、師岡一成、他 著

Apach Sparkはより高速にビッグデータを処理するための分散処理フレームワークです。 SQLインターフェースや機械学習などの機能が標準で組み込まれ、さまざまなシーンのデータ分析を強力にサポートします。

本書では、Sparkの分散処理の基礎であるRDDのしくみ、Sparkを構成する各コンポーネントの機能を理解するところからはじめます。 さらにSparkクラスタの構築と運用、構造化データを処理するためのDataFrame APIとSpark SQL、ストリーミング処理のためのSpark Streaming、機械学習ライブラリMLlib、グラフ処理のためのGraphXの各コンポーネントの実践的な利用方法を基礎から徹底的に解説します。

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

第90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。

サーバレスアーキテクチャ

クラウドをフル活用するサーバレスアーキテクチャ

ITシステムを構築するうえでクラウドという選択肢はすでに当たり前になりつつありますが、そのクラウドの能力を最大限に活用できるものとして「サーバレスアーキテクチャ」と呼ばれるコンセプトに注目が集まっています。サーバレスアーキテクチャとは、その言葉どおりアプリケーションサーバを省略した形のITシステムのアーキテクチャを指します。

従来のアーキテクチャは、アプリケーションサーバがクライアントからのリクエストを受け付け、データベースなどのバックエンドサービスと連携してサービスを提供するという3段階のレイヤで構成されるのが一般的でした。しかしこの方式はシステムの規模が大きくなるにしたがって複雑化しやすく、インフラの構築や運用管理といったビジネスに直接かかわらない部分のコストが増大するという問題を抱えていました。

これに対して、管理するべきアプリケーションサーバそのものを省略し、必要なときにだけ必要なプログラムを動かすようにしようというのがサーバレスアーキテクチャの考え方です。具体的には、クライアントや他のサービスからのリクエストなどをトリガーとして起動するイベント駆動型のプロセスを用意することで、アプリケーションサーバに代わって直接的にバックエンドのサービスに接続できるようにします。実行プロセスは非常駐で必要に応じて起動され、処理が完了すると終了するため、運用管理のための複雑なしくみが不要になるというのがこの方式の強みです。

サーバレスアーキテクチャを採用することで得られる主なメリットとしては、次のようなものが挙げられています。

- リソースの調達が容易
- サーバの運用管理コストを削減できる
- サーバのスケーラビリティやアベイラビリティの維持をクラウド事業者に任せることができる
- 自前のイベントを作成できるため、自由度が高い
- リクエスト単位の課金体系によって、コストを最適化できる
- 本来のビジネスに集中できる

主要なクラウドサービスがサポートを開始

ここ最近でサーバレスアーキテクチャへの期待が高まっているのは、パブリッククラウドサービスにおいてこのようなしくみを実現するための機能が提供されるようになったからです。これによって、前述のようなイベント駆動型のプロセスを、既存のクラウドサービス上に簡単に実装できるようになりました。本稿執筆時点で提供されているサーバレスアーキテクチャ向けの主要なサービスとしては次のようなものがあります（プレビュー段階のものも含みます）。

● AWS Lambda

Amazonの「Amazon Web Services（AWS）」で提供されるイベント駆動型のマネージドサービス。サーバレスアーキテクチャというトレンドを生み出すきっかけとなった。使用可能なプログラミング言語としてJavaScriptだけでなくPython、Javaをサポートしている。Amazon API Gatewayとの連携による独自APIの提供や、Amazon VPC（Virtual Private Cloud）への対応など、豊富な機能と高い利便性を備えている。

● Google Cloud Functions

Googleによる「Google Cloud Platform」に追加されたサービス。Node.jsベースのJavaScript実行環境として実装されており、Google Cloud StorageやGoogle Cloud Pub/Subからのイベントや、HTTPの呼び出しをトリガとして起動するサービスを開発できる。

● OpenWhisk

IBMの「Bluemix」で提供されるサービス。JavaScriptおよびSwiftを利用してイベントを作成できるほか、イベント駆動できる対象としてDockerコンテナ内のアプリケーションをサポートしている点が大きな特長と言える。

● Azure Functions

Microsoftの「Microsoft Azure」向けに提供されるサービス。JavaScript、C#、Python、PHPといった多数の言語をサポートしている。ランタイムやSDKなどがオープンソースで公開される予定で、他社のクラウドサービスやオンプレミスな環境でも実行可能になるとのこと。

◆ ◆ ◆

サーバレスアーキテクチャは、開発の内容をビジネスの本質に直結させることができるしくみと言えます。これはITシステムが市場の変化に追随していくための強力な武器になるでしょう。SD

DIGITAL GADGET vol.210

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

アプリの次はボットの時代

ボットの波が押し寄せる

機械学習や人工知能、自然言語処理の技術が進化してきたおかげで、メッセンジャーでなにか問いかけると、人と話すかのように会話形式で自動返信するチャットボットと呼ばれるシステムが台頭してきました。従来から、インターネットボット、Webボット、Slackボット、Twitterボットなど、ボット的なしくみは存在していましたが、最近になり、既存の人気プラットフォーム上で動き、平易に開発可能なボットが出てきました。LINEのBOT APIベータ版の発表、Facebook Messenger Platformの発表、MicrosoftのMicrosoft Bot Framework、Skype Botsの発表などが目白押しです。

わざわざインストールしなければいけないアプリの時代から、普段使っているメッセンジャーで気軽に人と会話する気分でサービスを享受できるチャットボットが、次にくるコミュニケーションスタイルだと注目を浴びています。

たとえば、ピザが頼める、フライト情報のチェックが行える、暇なときに会話の相手になってもらえる、などのほかにも多様な用途が期待されます。何か困ったときサポートに電話をかけますが、そっけない機械音声にしたがって「#1，#2」などを入力させられたあげく延々と待たされるサポート対応も、チャットボットにするとスムーズにやり取りができるようになるかもしれません。実際、メールや電話で問い合わせや質問をするよりも、チャットで気軽に問い合わせられることは、心理的障壁やそれに費やす時間の軽減などが見込まれ、注目されている分野でもあります。

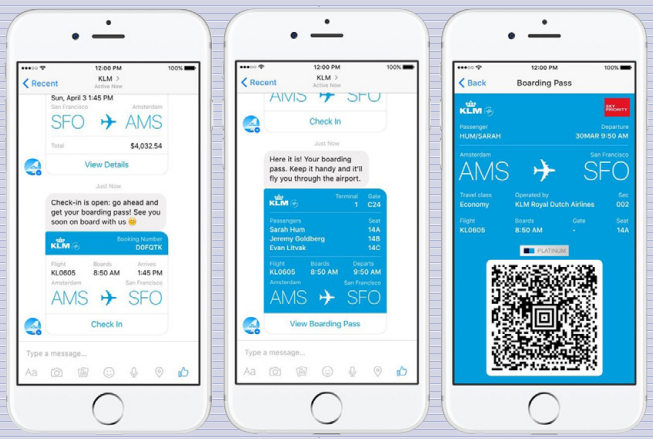
現在でも、チャットベースのコミュニケーションツールであるSlack上のチャットボットや、IntercomやZoho Chat、ChatCenterといったチャットを基軸としたユーザサポートのしくみなどが、これからはさらに一般的なメッセンジャーの中で使えるようになってくることが考えられます。

Facebook Messenger Platform
https://messengerplatform.fb.com/

LINE BOT API
https://developers.line.me/bot-api

Microsoft Bot Framework
https://dev.botframework.com/

これまでにもKik、Telegram、Slack、wit.ai、botland.ioなど、独自のシステムをもったチャットボットのフレームワークは存在しましたが、誰もが使っているメッセンジャーのプラットフォームがボットのしくみを用意したことで、放っておいても利用者が増え、ボット化するサービスの増加が期待されます。



航空会社KLMのMicrosoft Botを活用したサービス。搭乗にまつわるさまざまな事柄を会話形式で進めていくことができる。旧来もスマートフォンアプリでできたことではあるが、その場その場の状況に応じて、より気軽に平易になったと言える

※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。

ボットで何ができるのだろう?とまだまだ疑問視している方は、すでに先行しているKikの「bot shop」というサービスのポータルストアをご覧になるとイメージが沸いてくると思います。

また、botlist(https://botlist.co/)では、スマートフォン用のアプリストアのようなボット専用のストアとして対応プラットフォームを増やしつつあります。配信先として、メールやSMS、IFTTTなどにも対応しています。ボットのカテゴリも、カスタマーサポート、ニュース、ヘルスケア、マーケティング、セキュリティ、旅行、コミュニケーション、支払い、解析ツール、翻訳、タスクマネジメントなど、多岐にわたっています。そのうち、"たくさんあるボットを管理するボット"が重宝されるようになるのかもしれません。

ボットで考えなければいけないユーザ体験

ボットのサービスを考える際、従来のWebフォーム入力やメールによるコミュニケーションをそのまま持って来てもうまくいきません。また電話や人間同士の会話を参考にしても、まだまだ人間の持つ高度な状況判断能力を超えられるものでもありません。実際、すでにボット化されたサービスを営んでいる企業でも、定型文はボットにまかせ、複雑なやり取りが発生したら人が対応を引き継ぐなど、ボットと人のコミュニケーションをうまく組み合わせて実施しているところもあるようです。

それでは、ボットで何かサービスを代替したいと考えた場合、どのような観点で考えれば良いのでしょうか?

1 コンテキスト重視

ボットで重要視されるのは、できるだけコンテキスト(文脈、状態)を理解したうえで、適切にパーソナライズされたやり取りの部分です。たとえば直前の話題や、前回した話題などを引き継ぐと良いでしょう。また、位置情報や曜日、時間なども、そのときの状況を示す要素です。

2 実はプル型メディア

ボットはサービス側からユーザ宛に情報が発信されるプッシュ型メディアのように思われていますが、実際はユーザの意志で情報を取りに行く、プル型のメディアとして考えることが重要です。何らかの要求があったときに、適切なタイミングで適切な回答をしないと、単に迷惑なスパムだと思われてしまいます。

3 ボットは会話

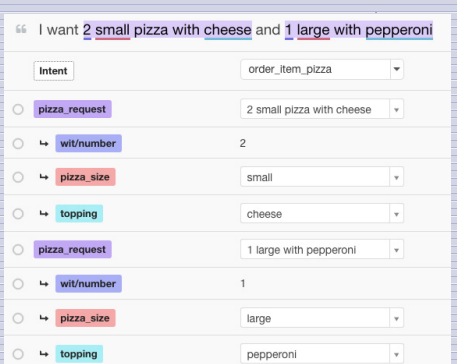
コミュニケーションの文面ひとつひとつを、広告や宣伝としてではなく、会話として有益な情報コンテンツになるように扱わなければ、親しみを持ってもらうことはできません。プッシュ通知以上に、タイミングや回答の内容、そして文面を吟味しなければいけません。最初のうちはマヌケなボットとして大目に見てもらえるかもしれませんが、使い続けてもらうには、スムーズなやり取りが重要です。この際、なんでもかんでも自由奔放な要求に答えるのではなく、人間側がある程度譲歩し、定型的なコミュニケーションの枠にハマるよう、体験をデザインしてしまうのも1つの工夫です。

4 ボットは距離感

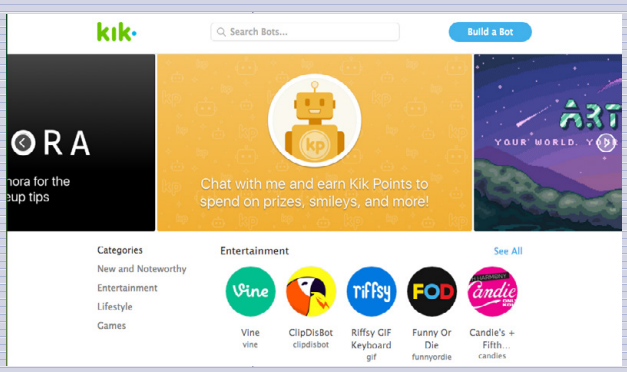
ボットとのコミュニケーションをシンプルな会話として設計することも大切です。メッセンジャーによるやりとりは、もともととても個人的なもので、本来信頼している親しい人同士としかやりとりしなかったものです。そういった意識が強いため、当然メッセンジャーを活用したサービスにも、信頼や親しみを期待することでしょう。サービスの頻度や文面によってコミュニケーションの距離感を適切に設計することも、利用し続けてもらうために重要な要素でしょう。距離感は近すぎても遠すぎてもいけません。

5 ボットでのパーソナライズ

個人情報を勝手に取得する気味の悪いものではなく、自分のことを良く知ってくれている行きつけの美容室や行きつけの居酒屋のように、好みを知り、その人の履歴を知ることが重要です。プライバシーを過度に心配する



wit.aiの設定画面。ピザを注文し、トッピングを選択できるような流れになっている



Kikのbot shopの画面。エンターテインメントや生活関係、ゲームなどが多数登録されている

ことなく、大切なのはその用途とバランスです。

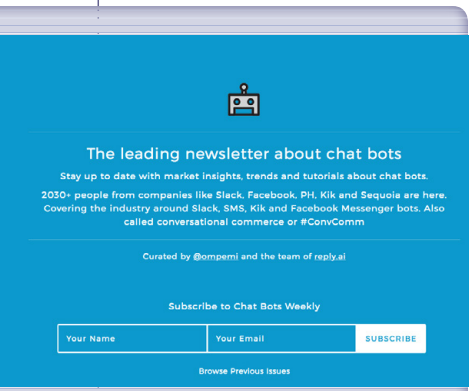
6 ボットはブランド

街にある店舗の店員の様相や態度、お客への対応がそのブランドを形作るように、ボットもその対応がブランドイメージを顕著に形作ることを忘れないようにしましょう。

ボットのこれから

長いあいだ連れ添った夫婦のように「おい。あれ」で通じたり、「いつものあそこに行きましょう」で通じたり、ボットの核となるのはコンテキストと呼ばれる状況や文脈の理解です。ある程度自分を理解してもらえるボットに育てあげることができれば、そのボットサービスから離れられなくなるのではないでしょうか?

これからものすごい数のボットが増え、皆がボットに依存していくであろう環境で、大量のボットからのメッセージを個々人がどう扱うかが、ひとつの課題になってくると思われます。我先にサービスを開始して最初のうちは物珍しがられるかもしれません。しかし、毎日、毎時間、興味のないメッセージでメッセンジャーが埋め尽くされることがないように、押し付けがましくない、適度で、適切で、対話したくなるボットとは何なのかを考える必要がありそうです。SD



チャットボットに関するニュースを毎週届けてくれるサービス
http://www.chatbotsweekly.com/

Gadget 1

PetBot

http://www.petbot.co/

ペット向けロボット

PetBotは犬や猫といったペット向けのボットです。内蔵するカメラでペットの状態を確認することや、遠隔で声をかけることもできます。エサやオヤツを蓄えておくことができ、専用アプリで遠隔操作でエサを与えることができます。家から離れているときも、いつもと同じしつけをしつつ、エサを与えることができるのが特徴です。今はまだ飼い主がリモートで応答しなければいけませんが、ペットとのコミュニケーションは、あらたなるボットの活躍領域かもしれません。

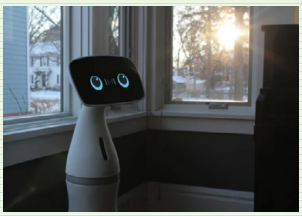
Gadget 2

Aido

http://www.aidorobot.com/

次世代ホームロボット

Aidoは音声認識で対応できる、顔がディスプレイ画面になっている家庭用ロボットです。バランスボールのような足で移動し、自宅のホームセキュリティ装置と連動したりすることもできます。背丈は約90cm、顔の役目をするディスプレイ装置でお知らせをしてくれたり、通知や警告などの役目を果たしてくれます。オプションでプロジェクタ搭載の機種もあり、まな板の上にレシピを投影する機能も予定しているそうです。お休み前に物語を読み上げてくれたり、アシスタントとしての役目をいろいろ果たすそうです。



Gadget 3

TuneBot

http://tune-bot.com/

ドラムチューニングロボット

TuneBotは、従来機械的には計りづらかったドラムの音程を調整するためのチューナーです。ドラムヘッドの張り具合を計測するのではなく、音程そのものを計測して表示することができ、素早くドラムセッティングすることが可能です。良い音が出たときのセッティングを記憶させておくこともできます。音程はヘルツ(Hz)と音階の両方で表示、調整することができ、クリップによってさまざまな打楽器に設置でき、設置したままでも演奏のじゃまにならないよう考えられているそうです。

Gadget 4

DoorBot

https://ring.com/

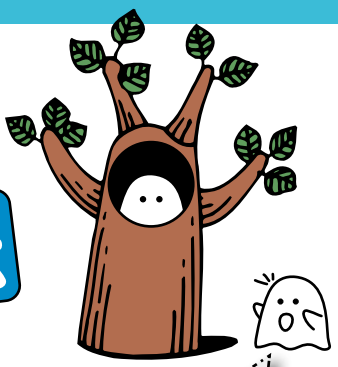
ドアベルロボット

DoorBotは登場当初の名前で、現在はringと名前を変更しています。デジタル技術のつまったドアベルで、家に不在のときにも専用アプリで訪問者と会話することが可能です。モーションセンサーとHD画質のカメラを搭載し、空き巣対策を考えた機能が用意されています。クラウドベースのビデオ監視サービスと連動していたり、モーションセンサーの検知領域を細かく設定できる機能を持つ上位機種も予定されています。

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

37

結城浩の再発見の発想法



カーソル

カーソル

カーソルとは

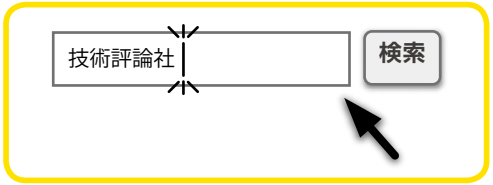
カーソルは、**現在位置を示す印**のことです。テキスト入力のときには文字が入力される位置を示し、GUIではマウスなどのポインティングデバイスの位置を示します。テキスト入力でのカーソルは「キャレット」と呼び、GUIでのカーソルは「マウスカーソル」と呼ぶことがあります（**図1**）。カーソルという言葉は、ラテン語で「走る」という意味の言葉から来ているそうです。ちなみに、英語のcurrent（現在）という言葉も同じ言葉に由来しています。

現在位置を示す印が「走る」という言葉に関わっているのはちょっと不思議な気もしますが、カーソルが画面を走っている様子を想像すれば納得がいきます。カーソルは、画面上を絶えず走り回って現在位置を教えてくれるからこそ、その役目を果たすことができるのですね。

カーソルという概念はキャレットやマウスカーソルのようなユーザインターフェースだけに出てくるものではありません。データベースを取り扱うSQLにもカーソルが登場します。クエリの結果得られた集合を1行ずつループ処理をするときにカーソルを使います。この場合も、カーソルは対象となっている行という「現在位置を示す印」として使われています。

カーソルは「現在位置を示す印」ですから、移動する手段が必ずあります。キャレットは矢印キーで移動できますし、マウスカーソルはマウスを動かせば移動できます。またSQLのカーソルも前後に移動させる命令があります。

▼図1　キャレットとマウスカーソル



カーソルの役割

コンピュータでは、カーソルは当たり前の存在ですが、あらためて「カーソルの役割」を考えてみましょう。もっとも重要なのは「たくさん並んだものの中から処理対象を指定する」役割です。指をさして《これ》を処理せよと指定したり、《ここ》で処理せよと指定したりするような役割ですね。だからこそ、直接画面にタッチして操作できるタブレットのようなデバイスの場合には、マウスカーソルを表示する必要がないのです。

カーソルは現在位置を示す印ですから、少ない指示で処理を表現できます。「位置(325, 221)にあるボタンを押下せよ」ではなく「カーソル位置にあるボタンを押下せよ」という指示になるという意味です。

カーソルは、「たくさんのものをまとめて処理」するときではなく、「たくさんのものを順番に

処理」するときに登場します。それは、問題解決の手法で言えば分割統治の道具として使えると言えます。まとめて処理できないほどの大きな問題があったなら、

- 現在位置で処理すべきことをせよ
- そして、次の位置まで移動せよ

ということです。この2ステップは、カーソル位置での処理と、カーソルの移動に相当しますね。

カーソルは出力装置でもある

カーソルは「現在位置を示す印」ですから、最大のトラブルは、位置が不明確になることです。ですからカーソルは、位置が明確になるような工夫を凝らします。キャレットなら点滅して、マウスカーソルなら軌跡を表示して位置をわかりやすく示します。

また、カーソルは位置以外の情報もユーザに伝えます。キャレットは、挿入モードなのか上書きモードなのかを幅の変化で示すことがありますし、マウスカーソルは、ボタンの上にきちんと乗ったことを矢印が指に変化して示すことがあります(**図2**)。

言い換えるなら、カーソルは自分の現在位置やモードをユーザに対して伝える出力装置であるとも言えます。それらの情報がきちんとユーザに伝わってこそ、カーソルが入力装置としての役割を果たすことができるのです。

日常生活とカーソル

日常生活にカーソルのようなものはあるでしょうか。すぐに思いつくのは**本のしおり**です。読んでいる現在位置を示す印として本のしおりを使います。しおりの位置は、自分が読み進むごとに移動していきます。

電話やSNSで誰かと話すときも、カーソルに似た概念が出てくることがあります。それは**あなた、今、どこにいるの？**という問いかけです。たとえば筆者は仕事から帰るとき「今から帰るよ」と妻にメールします。すると「あなた、今、どこにいるの？」という問いかけが返ってきます。もしも帰り道でスーパーマーケットを通り過ぎていなかったら、お豆腐を買ってきてほしいという買い物依頼が続いてやってきます。このとき筆者は、自分がカーソルになったような気持ちになります。幸い、マウスカーソルと違って、スーパーマーケットに引き返してという指示が出ることはありませんが。

もっと抽象的な意味でカーソルに似ているものがあります。それは**仕事の進捗状況**です。進捗会議で尋ねられる「現状はどうなっている？」という問いかけは、カーソル位置を聞かれているようなものですね。物理的な位置ではないけれど、1つのプロジェクトの中での現在位置が問われているのです。

人間は、大きな仕事を一気にまとめて処理することができませんから、現在の状況に合わせて順番に小さな仕事を処理していきます。現在の進捗状況というカーソル位置で直面している課題に取り組むのです。

マウスカーソルの位置を見失ったら、ユーザは適切なGUIの操作を行うことができません。それと同じように、メンバーが進捗状況を正しく報告できなければ、プロジェクトリーダーは判断を誤り、大きなトラブルになるでしょう。「現在位置を示す印」というのはとても大切な役割を果たしているのです。

あなたの周りを見回して、カーソルのように「現在位置を示す印」を探してみましょう。その印は、どんなときに変化するでしょうか。それが正しい位置を示さなくなったら、どんなトラブルが起きるでしょう。

ぜひ、考えてみてください。SD

▼図2　マウスカーソルの形が変わる

増井ラボノート

コロンブス日和

第8回　EpisoPass

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

パスワードの諸問題

さまざまなWebサービスでユーザ認証のためにパスワードが使われています。パスワードはいろいろ問題が多いシステムであり、パスワードにかわるさまざまな認証システムが提案されてはいるものの、すべての点でパスワード認証より優れたシステムは存在しないと言われているので、パスワードによる認証がほかの方法で置き換えられる時代はすぐには来ないでしょう。

パスワードの最も嫌なところは、**覚えておくのがとてもたいへん**なことだと思います。パスワードを忘れて困った経験がない人はいないでしょう。いろいろなサービスで同じパスワードを使い回すのは危険ですし、パスワードはときどき変更したほうが安全だと言われていますが、複雑で長いパスワードをたくさん覚えておくことは不可能ですから、仕方なく同じパスワードを使い回している人は多いと思います。

異なるパスワードをすべて記憶することが不可能なのであれば、紙やファイルに書いておけば良いかもしれませんが、パスワード文字列をそのまま記録するのは危険なので、複数のパスワードを暗号化して覚えておくためのさまざまな**パスワード管理システム**が利用されています。たいていのパスワード管理システムは1つの「マスターパスワード」を利用してほかのすべてのパスワードを管理するようになっていますが、マスターパスワードは覚えておかなければなりませんし、パスワード管理システムはどこでも使えるとは限りません。できれば特殊なシステムを使うことなく、複雑な多数のパスワードを利用できるほうがうれしいでしょう。

そもそも強力なパスワードを作成してからそれを覚えたり管理したりするというやり方が間違っているのではないでしょうか。新しく作ったパスワードを覚えたり管理したりするのではなく、すでに知っていて忘れようがないような秘密の記憶を基にして、複雑なパスワードを生成して使うことにすれば、パスワードを覚えられない問題は解決するはずです。

EpisoPass

子供のころちょっと怪我をしたとか、イジメられた嫌な経験とか、うっかりした失敗を隠していたとか、他人に話したことはないけれども忘れることがないような秘密の記憶というものが誰にもたくさんあると思います。強い体験に基づく記憶は、**エピソード記憶**と呼ばれ、時間が経っても消えることがありません。一方、数式や電話番号を記憶しようとしてもなかなか覚えられませんし、覚えていたつもりでも時間が経つとキレイに忘れてしまっていたりするものです。このようなものは**意味記憶**と呼ばれ、エピソード記憶に比べると長期的な記憶が困難です。

パスワードがエピソード記憶だったら良いのですが、体験的にパスワードを覚えるのは無理でしょう。であれば逆に、忘れることがない秘

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

密のエピソード記憶を基にして、パスワードを生成するようにすれば、秘密で複雑で忘れないパスワードを安心して使えるようになるはずです。

EpisoPassは、ユーザが忘れることがない個人的なエピソード記憶を、複雑な文字列に変換することによって安全なパスワードを生成するシステムです。

パスワード文字列は次の手順で生成されます。

- パスワード生成の「種」となる文字列（シード文字列）を用意する
- 忘れることがない個人的なエピソード記憶に基づく秘密の質問をいくつか作成し、それぞれについて1つの正答と複数の偽答を用意する
- 質問と回答の組に基づいてシード文字列に換字操作を行う。すべてに正しく回答したとき生成される文字列をパスワードとして利用する

問題文字列とユーザが選んだ回答文字列を結合した文字列を生成し、そのMD5値を基にしてシード文字列を換字することによりパスワードを生成しています。

ブラウザでの利用

図1は私がTwitterのパスワードを生成するために、ブラウザでEpisoPassを利用しているところです。

シード文字列として「Twitter123456」という文字列を指定しており、4個の秘密の質問に対する回答選択に応じて、「Mfveabn574923」のようなパスワード候補が生成されます。

異なる答えを選択するとまったく異なる文字列が生成されます。

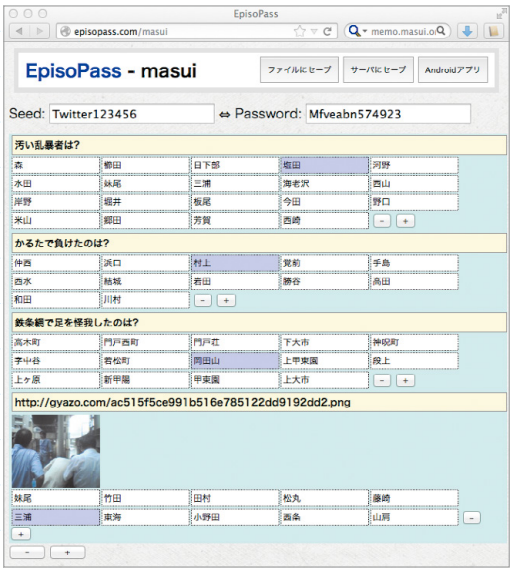
シード文字列を「Facebook123456」に変更すると、生成されるパスワードは**図2**のように変化します。このように、サービスごとに異なるシード文字列を利用することによってさまざまなパスワードを簡単に生成できます。

「いつのパスワードですか」のような質問を用意しておき、「2016/5」「2016/6」のような選択肢を用意しておけば、毎月異なるパスワードを生成できます。

シード文字列の8文字目が数字である場合はパスワードの8文字目も数字になるなど、シード文字列の文字種に対応したパスワード候補が生成されるようになっています。パスワードとして大文字／小文字／英数字／記号をすべて利用しなければならないサービスの場合は、シード文字列に「PassWord123!@」のような文字列を指定します。

最初の秘密の質問は私の小学校のときの体験に基づくもので、最後の質問は数年前の体験に関するものです。これらは古いエピソード記憶になっているので、私が将来答を忘れることはほとんど考えられませんが、私以外の人間がこのような質問に答えることは難しいので、正し

▼図1 EpisoPassの利用例



▼図2 EpisoPassによるパスワード生成

いパスワードを得ることはできません。

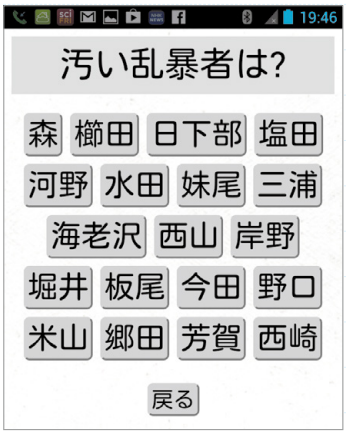
秘密の質問と答はブラウザで編集でき、右上の[サーバにセーブ]ボタンを押すことにより、シード文字列、秘密の問題、答のリストがサーバにセーブされます。[ファイルにセーブ]ボタンを押すとJSONデータをパソコンにダウンロードでき、パソコン上のJSONデータをブラウザにドラッグドロップすると、サーバにアップロードできます。ユーザはどれが正答かを指定するわけではないので、問題データを見てもユーザのパスワードはわかりません。

Android アプリケーション

ブラウザからWebサービスを利用する場合、ブラウザとサーバとの間の通信を盗み見されたり、パソコン上の操作を記録されたりする心配を完全になくすことはできません。ブラウザでEpisoPassを使う場合、パスワードはブラウザ内部でJavaScriptにより生成されるので、一度ページを表示したあとはネットワークを遮断してもパスワード計算できるのですが、ブラウザを使わずにパスワードを作成できるほうがより安心でしょう。このため、通信をまったく行わずにマシン単体でパスワード計算を行うためのAndroidアプリも用意しています。ページの右上の[Androidアプリ]ボタンを押すと、現在表示している秘密の問題と答を内蔵したAndroidアプリが、サーバ上でビルドされてダウンロードされます。Android端末でアプリを実行すると**図3**のような画面が表示されます。シード文字列を設定して[開始]ボタンを押すと**図3**のように質問が1つずつ表示され、ボタンを押してすべて回答するとパスワードが計算され**図4**のように表示されます。

回答の選択とパスワード計算はAndroid端末で実行されるため、端末を機内モードに設定するなどの方法でネットワーク接続を遮断した状態でもパスワードを計算できます。EpisoPassをインストールしたAndroid端末を持っていれば常に各種のパスワードを計算できるので、他人のマシンや公共の場所に設置されたパソコンなどでも、容易にTwitterなどのネットサービスを利用できます。

前述の方法でEpisoPassアプリをサーバからダウンロードする場合は、ブラウザ上で秘密の問題をサーバに登録する必要がありますが、秘密の問題をまったくネット上に露出することなくアプリを利用することもできます。秘密の問題を含まないEpisoPassアプリをGoogle Playで公開しているので、これを端末にインストールしたあと、ローカルマシンで作成した秘密の質問を端末に転送すれば、EpisoPass.comからダウンロードしたアプリと同様に利用できます。この手法を使うと秘密の質問が通信路を通ることがないので安全ですが、アプリのセットアップの手間は増えます。



◀ **図3** Androidアプリでの使用例

◀ **図4** 生成されたパスワード

EpisoPassの安全性

EpisoPassで選択枝が20個の質問を10個使

用する場合、総当たりでパスワードを生成するには約10兆(＝20^{10})通りの試行が必要になり、大小英文字からランダムに8文字を並べた約50兆(＝52^8)通りの文字列からパスワードを選ぶ場合と同程度の強度になります。総当たり攻撃が可能なオフライン運用ではこのような強度は重要ですが、オンラインサービスでは、パスワード入力を何度か間違えるとサービスがブロックされるのが普通なので、それほど長いパスワードを用意する必要はないでしょう。

秘密の質問を利用する認証は脆弱だと言われることがあります。パスワードをリセットするために「母親の旧姓は？」「最初に飼ったペットの名前は？」のような質問に対してユーザに答えを登録させるサービスがありますが、このような問題は他人が調べたり推測したりすることが簡単ですし、秘密の質問の数は一般的に少ないので、パスワードよりも脆弱なのはたしかです。EpisoPassでは、他人には解くことが難しく自分では忘れないような秘密の質問をたくさん登録しておけば安全です。

秘密の質問の選択

EpisoPassでは、他人が推測することが難しく、自分は決して忘れないようなエピソード記憶を秘密の質問として利用します。忘れないエピソード記憶であっても、次のような性質を持つものは秘密の質問として不適当です。

- 自慢になるもの(何かの機会にうっかり他人に自慢しまう可能性があるので)
- ネット上に記録が残っているもの
- 他人と情報を共有しているもの
- 趣味や嗜好に関連するもの(他人に推測されやすいうえに嗜好が変化する可能性があるので)

このようなものではなく、「わざわざ人に話すことはないが自分の記憶に強く残っているような無難なエピソード記憶」を秘密の質問として利用するのが良いでしょう。例に挙げた「鉄条網で怪我した場所は？」という問題の場合、私はこの経験について他人に話したことはありませんし、今後自慢することがあるとは思えませんが、痛い思いをしたことは忘れませんから、問題として適切だと言えるでしょう。

偽答の作成方法

問題の種類によっては偽答の生成が難しかったり、答えが予測できたりしてしまう場合があります。たとえば「好きなスポーツは？」のような質問の場合、たくさんの偽答を用意することが難しいですし、本人を知っていたら想像がつくかもしれませんので問題としては適切でありません。

一方、答えが人名や地名の場合、正答に似た人名や地名を並べることは簡単です。たとえば「世田谷」が正答であるとき、「目黒」「杉並」のような偽答を用意するのは簡単です。例に挙げた「鉄条網で怪我した場所は？」という問題の場合、似たような地名をたくさん並べることが簡単なのでEpisoPassの問題として適切だと言えるでしょう。

ある単語と同じカテゴリに属する単語を探す「同位語検索」と呼ばれる手法がいろいろ提案されているので、これを利用して正答と同じカテゴリに属する単語を自動的にリストできれば、簡単に偽答リストを作ることができるでしょう。

私は数年前にEpisoPassを開発してから、FacebookやTwitterなどさまざまなWebサービスのパスワードの管理にEpisoPassを使っており、パスワードに関する悩みが完全に解消されました。パスワードで消耗している方はぜひご利用ください。SD

コラム 「ソースコードも公開中」

EpisoPassはhttp://EpisoPass.comで運用中です。ソースコードも公開しています(http://github.com/masui/EpisoPass)。

第4回 OSCは、なぜ全国をさすらうのか

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

全国でOSCを開催するまでの経緯

3月下旬からGW明けの5月中旬ぐらいまでの間はOSCはほとんど開催されないので、今月はOSCで全国を渡り歩いている話をしておこうと思います。

そもそも、全国各地を放浪するようになったのは、15年前（2001年）に会社を設立したときに「全国縦断オープンソースセミナー」と題して、北海道から福岡（沖縄ではない）まで全国4ヵ所で開催したことに遡（さかのぼ）ります。「びぎねっと 社史」で検索すると、10周年の際に作成した記録が確認できます。

- 2001年3月23日 大阪開催
- 2001年3月30日 福岡開催
- 2001年4月13日 札幌開催
- 2001年4月23日 東京開催

（以上サーバ構築）

- 2002年1月18日 大阪開催
- 2002年1月21日 福岡開催
- 2002年1月25日 東京開催

（以上Samba構築）

- 2002年12月 福岡開催
- 2003年1月17日 東京開催
- 2003年1月24日 札幌開催

（以上Postfix）

各開催の結果からみると、頑張って開催した割には集客に苦戦しました。場合によっては数名しか参加しないこともありました。しかし、このときにセミナーや終了後の懇親会に参加してくれた各地域の人たちが、2004年以降に全国でOSCを開催するときに大きな力となってくれたので、全国行脚をした成果はあったと言えるでしょう。

なぜ全国をさすらうのか

2年間、各地に赴いて現地のエンジニアのみなさんと交流してわかったことは、「東京一極集中」の現状です。筆者は神奈川県出身なので、あまり深く考えずに東京で就職して仕事をしてきていましたが、セミナーなどの情報提供も東京とせいぜい大阪ぐらいです。一方、東京だけで見ても、OSSコミュニティとしての情報発信力は企業のそれに比べると残念ながら弱いというのが当時の状況でした。

このような両面の課題を解決するために、「OSCを全国で開催しよう！」と思い立ちました。そして開催2年目となる2005年には、東京だけでなく北海道と沖縄で開催することにしました。「OSCは、北は北海道から、南は沖縄まで開催しています。」と言っていましたが、嘘はついてません。

北海道は、地元コミュニティの方々と飲みながら「OSCを北海道で開催しよう！」という話になったことがきっかけです（「石鍋会談」。札幌・すすきのにある石鍋亭は6月のOSC北海道開催時に紹介予定）。

そして沖縄で開催したのは、北海道も同様ですが、講師となる人が「[沖縄|北海道]だったら行ってみたい！」と思わせるのが狙いです。北海道や沖縄から大挙して東京のセミナーに参加するより、東京から講師が北海道や沖縄に赴いてもらうほうがトータルのコストが安く済む、というわけです。

そういえば、初の北海道開催のときは、Rubyの開発者であるまつもとゆきひろ氏に島根から来てもらい、講演していただいたのでした（**写真1**）。前夜祭の際に、北海道大学の学生がRubyの仕様についてアレコレ提案しているのに真摯に耳を傾けていたのが印象的でした。

▼写真1 OSC2005北海道の前夜祭の後、まつもとゆきひろ氏を囲んで

地域コミュニティのボトムアップな盛り上がりを支援

OSCを全国各地で開催する一番の理由はその地域のコミュニティの活性化ですが、その次に来るのは地域

間の交流です。最初は東京と北海道、沖縄の2点間で始まりましたが、その後、徐々に開催地を増やしています。2004年から2009年までの間だけみても、次のように開催地を少しずつ増やしていきました。

・2004年1月　東京のみ
・2005年4月　北海道､沖縄を追加
・2006年6月　新潟を追加
・2007年8月　京都、福岡を追加
・2008年11月　大分、長岡、名古屋、島根を追加
・2009年12月　仙台、高知を追加

京都や福岡より先に新潟(**写真2**)で開催されたり、名古屋はさらにその後だったりしました。ビジネスであれば大都市圏から攻めていくのが定石ですが、OSCは現地からの立候補があってはじめて開催することにしているので、このような不思議な順番になっています。

地域コミュニティの活動はボトムアップであるべきで、トップダウン、東京から一方的に出向いていくのでは地域の自主的なコミュニティ活動にはなりません。OSCはあくまで地域コミュニティの人たちが自主的に開催するもので、事務局はあくまでそのお手伝いをさせてもらっている、というのが基本的なスタンスとなっています。

▼写真2　2006年OSC新潟。川越を中心に活動する小江戸らぐの皆さんも出展

OSCは開催地域間でのメッシュ型トポロジー

OSCでは東京以外の地域での開催に、そのほかの地域のコミュニティの人たちが出向くことは珍しくありません(**写真3**)。東京対そのほかの地域という一極集中型のトポロジーではなく、すべての開催地域間でのメッシュ型のトポロジーが自然とできあがっていったところが、OSCの魅力の1つかもしれません。

▼写真3　「石鍋会談」にも参加していたサンビットシステム佐々木さん。北海道からOSC2015京都にバイクで参戦

前回、北陸・金沢に出かけて行ったのも、福岡での開催に金沢の大学の先生が講師として参加しており、懇親会の際に金沢でもOSCを開催したいというお話をいただいたので、まずは現地を視察に行ったという背景があります。現在、プレイベントとして「8月末にアンカンファレンス形式の集まりを開こう」という動きになっているので、開催の暁にはぜひ、ご報告したいと思います。SD

Report

飲み会開催のコツ教えます！

OSCといえば懇親会ですが、勉強会や職場で飲み会の幹事をするときのために、コツをいくつか伝授します。キッチリやろうとエンジニア精神を出すのではなく、「ゆるふわ」で適当にやるのが幹事のコツです。

・ドタキャン率を見込む

IT業界はドタキャンが多いので、ドタキャン率を見込んでおきましょう。OSCだと大体1割程度を見込んでおり、減った分を当日飛び入りで埋めるようにしています。

・参加費は多めに取る

お店に3,500円払うなら、キリよく4,000円徴収しましょう。OSCでは差額残金を学生の割引の原資などに活用しています。

・人数調整に融通が利く店を選ぶ

ドタキャン、飛び入りで増減した人数を午後3時ぐらいまでに伝えれば調整してくれるお店が望ましいでしょう。また、人数が増えた場合、飲み放題のみ差額追加でOKの店も使いやすいでしょう。

▲OSC沖縄の懇親会。幹事初心者はこれぐらいの人数から始めるといいでしょう

▲OSC2015新潟の懇親会。参加費を少し多めに徴収して、美味しいお酒の用意も素敵です

ツボイのなんでもネットにつなげちまえ道場

第12回

携帯電話通信網でつなげてみよう

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

本誌2016年5月号に、「SORACOM Air SD Special Version」が特別付録としてつきましたが、今回はSORACOM Airと携帯電話通信網に接続できるmbedを使って、インターネットに接続してみましょう。

u-blox C027

u-blox C027[注1]は、スイスのu-blox（ユー・ブロックス）社が製造している、LISA-U2という3G（UMTS）通信モジュールを搭載したmbedプラットフォームの基板です（**写真1**）。

この連載で題材に選択してきたmbed LPC1768と同じ、NXPのLPC1768というマイコンが、LISA-U2とUARTとUSBで接続されています。C027に搭載されているLISA-U2やGPS受信モジュールは、C027_Supportというライブラリ[注2]が提供されています。このライブラリのソースコードを読むと、LISA-U2とUARTを通じてLPC1768と通信をしています。

C027にはマイコンに加えて3G通信モジュールが搭載されているため、一般的にUSBから給電できる500mAよりも多くの電力を消費します。ですので、12Vで900mA以上を出力できるACアダプタを用意して給電してください。プラグの外形が5.5mm、内径2.1mmでセンタープラスのACアダプタを使うことができます。筆者は手元にあった12V2AのACアダプタを接続して使いました。

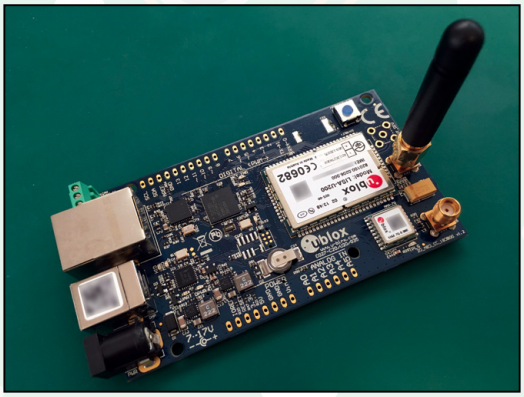
u-blox C027のドラッグ＆ドロップでプログラムを書き込む機能を提供するファームウェアは少々古く、筆者の手元のOS X 10.11.4ではうまく書き込みができませんでした。このため、今回はWindows 7を使って試してみています。

では、さっそくC027にSORACOM Airの

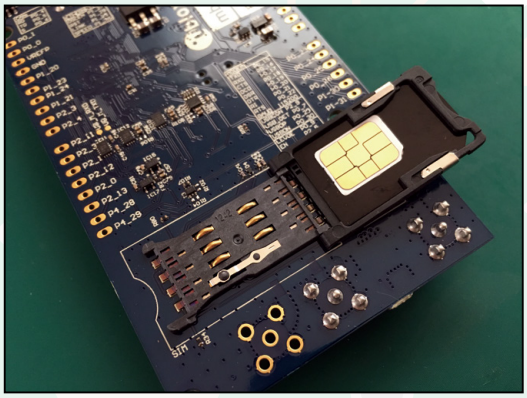
注1） http://ssci.to/2342 43,200円（税込み）

注2） https://developer.mbed.org/teams/ublox/code/C027_Support/

▼写真1 u-blox C027



▼写真2 SIMカードスロットの蓋を開けたところ



SIMカードを差し込んで使ってみたいと思います。本誌に付録していたSIMカードは、Nano-SIMサイズのものでした。一方でC027のSIMカードスロットは、Mini-SIMサイズ[注3]のものです。筆者は手元にあったアダプタを使って取り付けてみました。C027のSIMソケットはちょっと変わっていて、ロックする金具をスライドさせSIMカードスロットの蓋を開け、**写真2**のように蓋側にSIMカードを挿し込み、**写真3**のように蓋を閉じたところで金具をスライドさせて固定します。

HelloWorld

まず、C027で携帯電話通信網を使ってHTTPリクエストを出すサンプルプログラム、HTTPClient_Cellular_HelloWorld[注4]を試しに実行してみたいと思います。まず、このサンプルプログラムを自分のオンラインコンパイラ環境にインポートしてください。

次に、mcin.cppの冒頭にあるAPNやユーザ名の設定を、SORACOM Airに合わせて次のように書き換えます。

```
#define SIMPIN      NULL
#define APN         "soracom.io"
#define USERNAME    "sora"
#define PASSWORD    "sora"
```

また、main.cppの34行目にGETリクエストを発行するURLが書かれているのですが、これは古いmbed.orgのURLですので、筆者が自分で立てているWebサーバのURLに書き換えました。

```
int ret = http.get("http://www.ytsuboi.org/test.txt", str, 128);
```

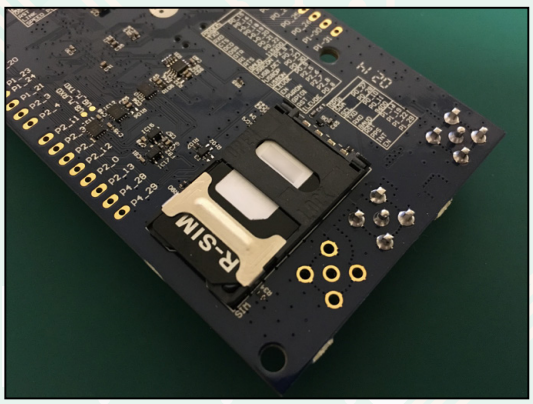
このようにソースコードを書き換え、コンパイルしたバイナリを書き込みます。Teratermなどのシリアルターミナルを使いmbedのシリアルポートを9,600bpsで開いた状態でC027のリセットボタンを押すと、**図1**のようにデバッグメッセージが出て動くことが確認できました。

サンプルプログラムは、通信を終えると、接続を切ったうえにLISA-U2の電源も切るように作られています。ですので、C027が通信をし

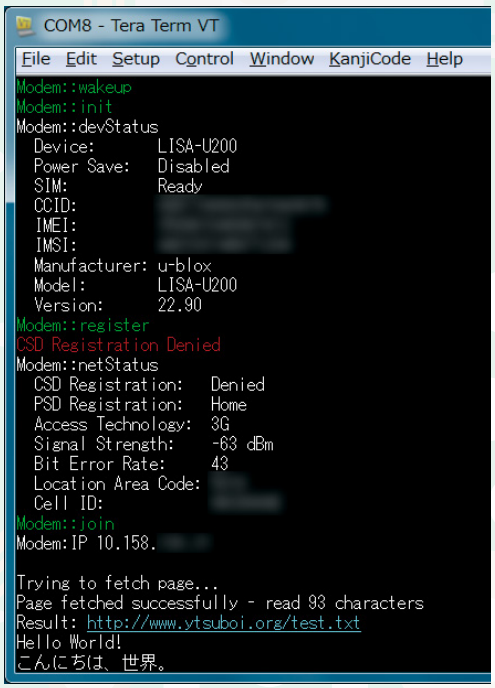
注3）余談ですが、実はSIMカードのサイズは、もともとはSIMが付いている台紙のサイズで、我々がよく「標準SIM」と呼んでいるのはMini-SIMサイズです。
https://upload.wikimedia.org/wikipedia/commons/thumb/8/8d/GSM_Micro_SIM_Card_vs._GSM_Mini_Sim_Card_-_Break_Apart.svg/2000px-GSM_Micro_SIM_Card_vs._GSM_Mini_Sim_Card_-_Break_Apart.svg.png

注4）https://developer.mbed.org/teams/ublox/code/HTTPClient_Cellular_HelloWorld/

▼写真3　蓋を閉じたところ



▼図1　サンプルプログラムを動かしてみた



▼図2　SIM管理画面

ている間にSORACOMのユーザコンソール側で接続を確認してみました。筆者はSORACOM Airを3回線登録しており、図2の一番下に表示されているSIMを使って実験してみています。

Milkcocoa

HTTPでGETするだけではおもしろくないので、クラウドプラットフォームにデータを送ってみましょう。今回は、日本の会社が提供しているサービスで、手軽に使い始めることができるMilkcocoa[注5]を使ってみたいと思います。MilkcocoaはBaaS[注6]ということで、Webアプリケーションや、IoT機器などの間のリアルタイムなデータのやりとりや、保存や取り出しといったバックエンドの機能を提供するサービスです。MQTT(MQ Telemetry Transport)というシンプルなメッセージ配信プロトコルをベースにしたプロトコルを採用しているとのことです。

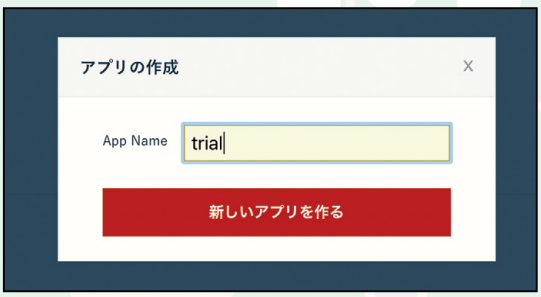
Milkcocoaは、同時接続数20まで、格納するデータ数が10万個までは無償で利用できます。さっそく、Webサイトの「無料登録」ボタンをクリックしてユーザ登録をしてみましょう。ユーザ登録を終えてログインをすると、ダッシュボードの「アプリリスト」が表示されます(図3)。登録した状態では何もアプリが登録されていないので、ここで「新しいアプリを作る」ボタンをクリックし、アプリを1つ作ってみてください(図4)。筆者はとりあえず「trial」と名付けて登録をしてみました。登録をすると、アプリリストに、今回作ったアプリが表示されます(図5)。名前の横に「app_id:」と表示されている値はデータを送るときに使いますので、これを控えておいてください。

次に、C027に書き込むアプリケーションを用意しましょう。サンプルプログラムは、Milkcocoa Sample_3G[注7]という名前で公開されています。

注5)　https://mlkcca.com
注6)　Backend as a Service：バックエンド機能を提供するサービス。
注7)　https://developer.mbed.org/users/jksoft/code/MilkcocoaSample_3G/

▼図3　ダッシュボード

▼図4　アプリの作成



これをインポートし、先ほどと同様にmain.cppでSORACOM AirのAPNなどを設定します。また、MILKCOCOA_APP_IDを定義している行で、先ほど控えたapp_idを指定してください。ソースコードの修正を終えたら、コンパイルして実行します。

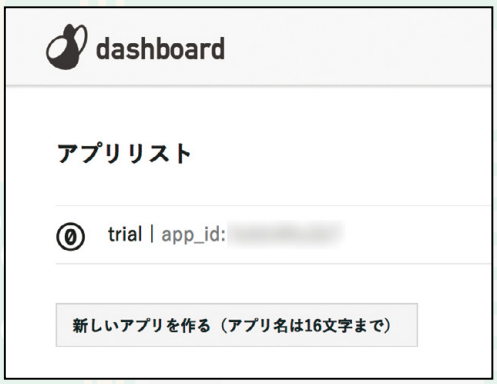
Milkcocoaのダッシュボードを開き、先ほど作ったtrialというアプリを開きます。そこで「データストア」を選択してデータストア名に「mbed」と入力して「リスト表示」ボタンをクリックしてください。すると、C027から送られたデータが表示されます(図6)。

このデータストアの名前「mbed」は、先ほど編集したC027のmain.cppに、MILKCOCOA_DATASTOREとして定義されていた名前です。main.cppに書いてあるように、値vと1を送った後7秒待つというプログラムが実行されています。ですので、データストアにも約7秒ずつ間隔を空けてデータが追加されていきます。センサから得た値をMilkcocoaに送っていけば、センサから収集したデータをMilkcocoaに保存できます。

まとめ

携帯電話通信網を使えば、EthernetやWi-Fiよりも自由な設置場所にノードを置くことができます。昨今では、SORACOMのようなセンサノードを設置するのに向いたMVNO(仮想移動体通信事業者)も増えてきました。3G(UMTS)通信の難点は、Wi-Fiと同様に通信時に比較的電力を必要とすること、モジュールの価格が高めであることくらいではないでしょうか。これも、LTEのカテゴリ0の登場によって変わっていくでしょう。SD

▼図5　アプリリスト



▼図6　データストアの表示

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6月16日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

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

01 無線LANルータ「WN-AX1167GR」 1名

IEEE802.11ac規格に対応し、867Mbpsの高速通信が可能な無線LANルータです。スマホやタブレットならQRコードを読み取るだけで設定でき、パソコンやゲーム機なら「WPSボタン」を押すだけで接続できます。全方向360度に電波の死角を作らない「360コネクト」、以前使用していたルータの無線設定をボタンを押すだけでコピーできる「Wi-Fi設定コピー」機能を搭載。

提供元 アイ・オー・データ機器 http://www.iodata.jp

02 Nimble Storage モバイルバッテリー 1名

次世代ハイブリッドストレージを提供する企業「Nimble Storage」の社名ロゴが入ったスマホ用モバイルバッテリー（2,200mAh容量）。Micro-USBで本体を充電、USBで給電します。電池残量が、デジタル表示で一目に確認できます。

提供元 Nimble Storage Japan http://www.nimblestorage.com

03 ESETファミリーセキュリティ1年版 3名

「動作の軽快さ」と「高い検出率」が特長のセキュリティソフトです。複数ユーザで5台分、1年間利用できます。最新版では新たに「インターネットバンキング保護」を搭載。対応OSはWindows XP以上／Mac OS X 10.6以上／Android 2.3以上となっています。

提供元 キヤノンITソリューションズ https://eset-info.canon-its.jp

04 ブラウザハック

Wade Alcorn ほか 著

毎日使っているFirefox、Chrome、Internet Explorerといった「Webブラウザ」をハッキングし、さらなる攻撃の足がかりとして利用する方法を学ぶことで、攻撃からの防御方法を考える1冊です。

提供元 翔泳社 http://www.shoeisha.co.jp 2名

05 Pythonチュートリアル第3版

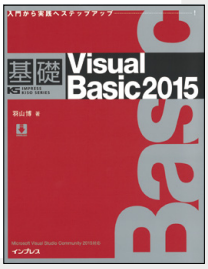
Guido van Rossum 著

プログラミング言語Pythonの作者Guido氏が書き下ろした、Python入門者のための手引き書。本書を読むことで、Pythonの雰囲気とスタイルをつかめるようになり、次の学習につながることでしょう。

提供元 オライリー・ジャパン http://www.oreilly.co.jp 2名

06 基礎 Visual Basic 2015

羽山 博 著



最新バージョン「Visual Studio Community 2015」上で、Visual Basicを使い、GUIの「Windowsデスクトップアプリ」を作るための方法を基礎から解説した入門書です。

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

07 ドキュメント作成システム構築ガイド

伊藤 敬彦、吉村 孝広 著

Gitを用いたバージョン管理、GitHubによる共同編集、RedPenによる品質チェック、CIツールによる継続的改善といったソフトウェア開発の技法に基づいた、ドキュメント作成支援システムを構築する1冊。

提供元 技術評論社 http://gihyo.jp 2名

第1特集

速く堅実に使いこなすための

bash再入門

エンジニアの道具を磨こう

速報 Bash on Windows

「毎日使うシェルだからこそ、その扱いに長けていたい」、そんな思いはありませんか？シェルを速く的確に使いこなすには、コマンドの知識以外に、効率的なキー操作、コマンドを組み合わせるノウハウ、状況に応じたシェルスクリプトの書き方も知る必要があります。

本特集ではbashを取り上げます。bashはほとんどのLinuxやMac OS Xの標準シェルです。多くの技術者がたしなみとして使える必要があるでしょう。Microsoft社もそれを意識して、Windows 10でbashを利用できるようにする計画を進めています。本特集で必要な知識を詰め込んだら、明日からはさっそく実作業で実践です！

※記事中の➡は本来1行のものが折り返されていることを、↵記号はEnterキーの入力をそれぞれ表しています。

bashとは何か ～古くて新しいシェル環境

Author くつなりょうすけ ㈱ネットワーク応用通信研究所 Twitter @ryosuke927

UNIXやUNIX互換OSを使うためには非GUI環境であるシェルでの操作は欠かせません。bashを使う前のウォーミングアップとして、シェルの役割とその歴史から触れていきます。そんなに身構えず、まずはのぞいてみましょう。

まだまだ主流なコマンドラインシェル

オフィスや研究室、家庭ではGUI(Graphical User Interface)が装備されたシステムの利用が一般的ですが、サーバ分野でそれなりにシェアのあるUNIXやUNIX互換OSが利用されるシステムでは非GUI環境がほとんどです。クラウドサービスやVPSを契約しても、まず最初に触れるのはGUIではなく非GUIのインターフェースですよね。

本特集では非GUIな、CUI(Character User InterfaceまたはCharacter-based User Interface)の代表的なソフトウェア、「bash」にフォーカスを当てます。

2016年夏に予定されているWindows 10のAnniversary Updateでは、Windows 10上でbashが動作するようになるとのアナウンスもありました。そんな今もチョベリグなインターフェースの、トレンディなbashをもう一度見つめなおして良さを確認しましょう。あまりbashを触ったことがない人は、このシェルの素晴らしさを共有し、楽しいbashライフを送れるようにしましょう。

そもそもシェルって何?

「Shell(シェル)」はコマンドライン・インタプリタという、入力されたコマンドを順次実行するソフトウェアです。大きく見ると、ユーザが指定したコマンドをカーネルに実行してもらい、結果をユーザに返す、ユーザとUNIXカーネルを橋渡しするソフトウェアです(**図1**)。

カーネルを貝殻(シェル)のように包み込んで、ユーザとのインターフェースになるシェルはおもに次の機能があります。

❶アプリケーションの起動と結果の出力
❷プロセスの「停止」・「再開」・「背後で実行」の切り替え
❸変数の保持
❹コマンド実行履歴の保持
❺まとまった処理をスクリプトとして読み込んで実行

機能❶:ユーザがシェルに指示したコマンドをシステムの中から探しだし、オプションや引数をコマンドに渡して実行、その結果の出力をします。ここで指定したコマンドのファイルが見つからなかったり、実行する権限がない場合は「コマンドが見つかりません」などのエラーメッセージを出力します。
機能❷:1つのシェル環境で、複数のプロセス(実行状態のコマンド)を起動することができます。実行しているプロセスを停止して他のプロセスを起動したり、停止してい

るプロセスを実行状態に復帰させる、もしくはユーザには見えないが背後で実行を継続させることができます。いくつものシェルを起動して画面を切り替えて動作確認する、といった煩わしさはありません。

機能❸：シェルの動作している環境で変数を保持することで、実行したプログラムが参照できたり、簡単な算術演算などに利用できます。

機能❹：一度入力した長いコマンドを再度入力しなくてもいいように、あるいは前に入力したコマンドを少しの修正で実行できるように、実行履歴の保持・編集ができます。

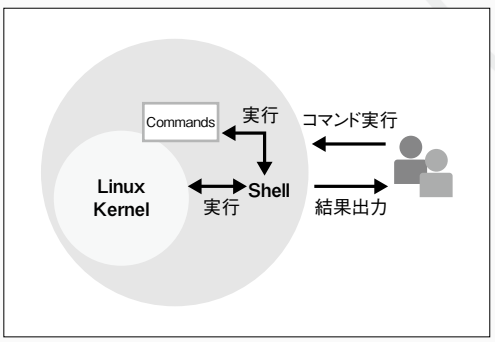
機能❺：あらかじめコマンドを記述したテキストファイルを読み込ませると、それを逐次実行することができます。これは、システム起動時に実行してシステム環境を整えるのに利用されていたり、簡単なコマンドを作るために利用します。シェルがifやforなどの条件分岐・繰り返し処理が利用でき、少し複雑なこともできます。このテキストファイルをシェルスクリプトと呼びます。

LinuxなどのUNIX系OSにおいて、シェルと言えばコマンドラインシェルというイメージが強くありますが、実はUnityやGNOMEデスクトップ、Windowsのデスクトップも「グラフィカルシェル」と呼ぶシェルの一種です。

ほら、カーネルを経由して「ファイルをオープンして表示する」を目的とした場合、Linuxでコマンド入力してテキストファイルをエディタで開くのと、Windowsでダブルクリックしてプレゼンテーションファイルを開くのは、あまり変わりないですよね。

この特集では、簡略化のために「シェル」といえば「コマンドラインシェル」のことを指すことにします。

▼図1　カーネルとシェルとユーザの位置関係



UNIXコマンドラインシェルの種類

ここでは今使える主なシェルの種類を挙げて説明します。それぞれのシェルの歴史も少し触れておきましょう。

最初のUNIX用コマンドラインシェルは、1971年にUNIXの最初のバージョンに搭載されたという「Thompson shell」です。これは後述するパイプやリダイレクトの機能は持っていましたが、スクリプトには不向きだったそうです。

その後1977年にAT&Tベル研究所というアメリカの電話会社の研究所にいたBourne氏が開発した「Bourne shell」、1978年に「csh」が開発され「Thompson shell」が置き換えられます。Bourne shellからの派生を「Bシェル系」、cshからの派生を「Cシェル系」と呼びました。これらの系譜を受け継ぐシェルにはおもに次のようなものがあります。

bash

bashは「Bourne Again SHell」の略で、GNUプロジェクトの1つとしてメンテナンスされています。それまでに存在したBourne shellに換わるシェルとして1987年に登場しました。ほとんどのLinuxディストリビューションと、Mac OS Xでも標準シェル(後述)に採用されています。

tcsh

Cシェルと呼ばれる流派のシェルです。BSD系OSでは標準シェルとして採用されています。

zsh

bashやksh(Bourne shellを機能強化したシェル)、tcshなどの機能を取り込んだシェルです。自分好みにカスタマイズしやすいので多くのファンがいます。

dash

Debian GNU/LinuxやUbuntuの/bin/shに採用されているシェルです。ashというBSD系OSでメンテナンスされているPOSIX sh(後述)に近い実装が基になっています。dashは「Debian版ash」という意味があります。

◆ ◆ ◆

UNIXには標準規格POSIXがあり、シェルについて記載されているPOSIX shは拡張版Bourne shellがそれにあたり、歴史的に/bin/shで利用できました。どのUNIX互換OSでも動作するシェルスクリプトを作成する場合は、POSIX shとして動作するよう心がける必要があります。

また、「標準シェル」にbashが採用されているからといって、/bin/shもbashとは限りません。「標準シェル」と言う場合は、「標準で設定されるログインシェル」のことを指します。Debian GNU/Linuxでは2009年のversion 5.0(lenny)、Ubuntuは2006年の6.10から/bin/shはdashへのシンボリックリンク[注1]になっています。/bin/shが指している実体のシェルは異なることがあると認識しておきましょう。

bashを使うメリット

zshやtcshも魅力的なシェルではありますが、あえてbashをオススメするのには理由があります。それはbashが現状Linuxの標準シェルであることです。自分の利用するPCでbash以外を利用することは自由ですが、ユーザのサーバや稼働系サーバであれば、インストールできるパッケージの制限がある場合もあり、その際は必然的にbashを使うことになるでしょう。パッケージを自由にインストールできる環境でも、ネットワークにつながっていない、インストールメディアを持っていない、といった場合もbashを利用することになります。

また、標準シェルなだけに情報があふれています。Webで検索すればbashの記事を簡単に見つけることができます。

そして、bashは今も進化を続けています。bashはもともと入力時にファイル名やディレクトリ名を途中まで入力すれば予測入力する「補完機能」がありましたが、zshで有名になった"コマンドのオプションも補完する機能"も最近(と、言っても2008年頃から)プラグインとして追加されています。他のシェル実装に対して遜色(そんしょく)ない機能を提供するbashは、なかなかできるやつです。

bashの情報

bashの最新情報は公式サイト[注2]を訪れるのがいいでしょう。最新のリファレンスマニュアルもあります。

Linuxディストリビューションで、manがインストールされているならば「man bash」でmanページを参照するのも手軽でいいです。収録されているbashの日本語manページの内容が少し古いと感じれば「LANG=C man bash」として英語manページも参照してみましょう。最新の情報が日本語版に反映されていないだけかもしれません。

bashのmanページは結構なボリュームです。ですが、一度流し読みして何がどのあたりに書いてあるか、求める機能の用語と自己認識に不整合がないかを確認しておくと、今後マニュアル内検索をする際に楽になります。SD

注1) 特定のファイル、ディレクトリに別名でアクセスするしくみ。Windowsのショートカットに似た機能。

注2) http://www.gnu.org/software/bash/

最初につまづかないための bashひとめぐり

Author くつなりょうすけ ㈱ネットワーク応用通信研究所 Twitter @ryosuke927

bashでの操作はキーボードから行いますが、ショートカットや履歴の使い方を覚えておくと長いコマンドを毎度ゼロから入力する必要がなくなり効率がよくなります。また、シェルの魅力の1つはシンプルな機能のコマンドを組み合わせることで煩雑な処理を一気に片付けられることです。本章で最初の一歩を踏み出しましょう。

シェルの起動方法

LinuxのようなUNIX系OSでは「login:」の文字が表示されている状態で、ユーザ名とパスワードを入力して認証が通ればシェルが起動します。GUI環境が目の前にある方は「端末」や「Gnome-terminal」のような名前のアプリケーションを起動しましょう。これらは「ターミナルエミュレータ」、「端末エミュレータ」もしくは簡単に「ターミナル」などと呼びます。ターミナルを起動すると、そのウィンドウの中でシェルを利用できます。あなたの目の前で、シェルはすでに起動しているかもしれません。

なお、本章執筆に用いた実行環境は、Debian/GNU Linux 8(jessie)です。また以後本章では、ユーザのアカウントを「ryosuke」として説明をします。

自分が使っているシェルは何だろう?

自身でインストールしたばかりのLinuxであればbashが起動していると思いますが、すでに稼働しているシステムにアカウントを追加してもらってログインしている場合は、利用しているシェルについて少なくとも2点把握しておく必要があります。

・ログインシェルは何か
・/bin/shの実体シェルはなにか

ログインシェルとは、ユーザのログイン直後に起動するシェルです(第1章参照)。/etc/passwdの各ユーザエントリの行末に記載されているのがそれです。

```
$ grep ryosuke /etc/passwd↵
ryosuke:x:1000:1000:ryosuke,,,:/home/
ryosuke:/bin/bash
```

fingerコマンドが利用できるようであれば、自身のアカウントを指定してShellの表示で確認することもできます。

```
$ finger ryosuke | grep Shell↵
Directory: /home/ryosuke  Shell: /bin/bash
```

第1章でも触れましたが、慣例上、シェルを指定する際に/bin/shを指定しますが、この実体がログインシェルと異なる場合があります。「`ls -l /bin/sh`」で確認しておきましょう。

別のシェルへの変更方法

LinuxのGUI環境で、GNOMEからUnityやKDEへデスクトップ環境を変更できるように、コマンドラインシェルもユーザが変更できます。

まず、使いたいシェルがシステムにインストー

ルされていることが前提なので、パッケージ管理ツールでインストールされているか、そして/etc/shellsに記載があるかを確認しましょう。

```
/etc/shellsを行番号付きで出力する
$ cat -n /etc/shells↵
     1  # /etc/shells: valid login shells
     2  /bin/sh
     3  /bin/dash
     4  /bin/bash
     5  /bin/rbash
     6  /usr/bin/tmux
     7  /usr/bin/screen
     8  /bin/zsh
     9  /usr/bin/zsh
```

シェルの変更はchshコマンドを利用するとコマンド一発で変更できて便利です。シェル変更後の確認は前述したようにfingerコマンドか、/etc/passwdを見るのが楽でしょう。

ここで試しにログインシェルを上記9行目に出力された/usr/bin/zshに変更してみましょう。**図1**の順に実行・確認して、ログインしなおせば変更したシェルが起動します。

ログインシェルを変更せずに他のシェルを試したい場合は、別のコマンドと同様に実行すれば起動できます(**図2**)。exitで戻れます。

コマンドプロンプト

使い始める前に、機能ではなく用語説明から。シェルを起動すると見えるチカチカと点滅するのは「カーソル」、その左隅に表示される文字列を「プロンプト」と呼びます。

図3では「ryosuke@mustain:~$」がプロンプトになります。とくに設定していない標準のbashでは「ユーザ名@ホスト名:作業ディレクトリパス$」を表示します。「~(チルダ)」はユーザのホームディレクトリを意味する文字です。

これからLinuxの入門書やマニュアルを読む場合にはプロンプトの最後の文字「$」に注意をしましょう。bashでは「$」は一般ユーザのプロンプトを意味します。もうひとつ「#」になる場合があります。**図4**ではユーザを切り替えるsuコマンドを使って管理者rootになったところです。プロンプトのユーザ名の部分がryosukeからrootに、プロンプトの最後の文字が「$」から「#」に変わっています。

書籍やマニュアルでは「$」は一般ユーザが実行できるコマンド、「#」は管理者権限で実行するコマンドと区別して書かれます。管理者権限で実行しなければならないコマンドを一般ユーザが実行してもエラーが出力されるだけで終わるかもしれませんが、一般ユーザで実行するコマンドを管理者権限で実行すると取り返しのつかないことが起こる場合があります。十分気をつけましょう。

コマンド実行

プロンプトの右側に、実行したいコマンドをタイプして↵キーを押すとコマンドが実行されます。コマンドの指定は次の形式になります。

```
$ コマンド名 [オプション]... [引数]...
```

「コマンド名」に指定するのは実行したい実行ファイル名です。実行ファイル名の後ろには「オプション」と「引数」を任意で指定します。オプションは実行ファイルの動作を変えるためにユーザが指定する文字列です。引数はプログラムの操作対象やプログラムに渡す文字列などを指定します。例を見てみましょう。

図5ではlsコマンドをオプションと引数を切り替えて実行しています。

①はオプション・引数なしで実行しています。結果は何も出力されず次のプロンプトが出ています。続けて②は「-a」オプションを、③では「-al」と指定しています。「-a」は「.」で始まるファイルを表示するオプションです。「.」で始まるファイルは設定ファイルなどに利用されるために隠しファイルとなっており、指定がないとリスト表示されません。「-l」オプションはファイルの詳細情報をリスト形式で表示するオプションです。このディレクトリには隠しファイルだけがあったようですね。④では「/var」という引数を

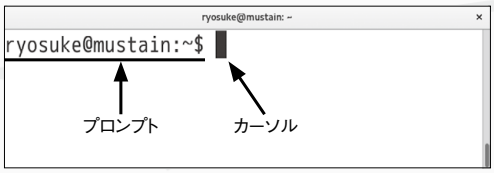
▼図1　ログインシェルの変更例

```
1：fingerで自身のシェルを調べる。「$USER」は自分のアカウント名が入った変数
$ finger $USER | grep Shell↵
Directory: /home/ryosuke                Shell: /bin/bash
2：/etc/passwdから自身のシェルを調べる
$ grep $USER /etc/passwd↵
ryosuke:x:1000:1000:ryosuke,,,:/home/ryosuke:/bin/bash
3：ログインシェルを/usr/bin/zshに変更する
$ chsh -s /usr/bin/zsh↵
パスワード:          ユーザのログインパスワードを入力する
4：fingerで再度自分のシェルを調べる
$ finger $USER | grep Shell↵
Directory: /home/ryosuke                Shell: /usr/bin/zsh
5：/etc/passwdから自身のシェルを再度調べる
$ grep $USER /etc/passwd↵
ryosuke:x:1000:1000:ryosuke,,,:/home/ryosuke:/usr/bin/zsh
```

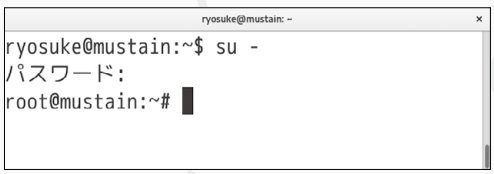
▼図2　一時的に別シェルを実行する

```
現在実行中のプロセスIDのプログラム名を表示する
$ ps $$↵   「$$」は実行中のシェルプロセスID
  PID TTY      STAT   TIME COMMAND
 6148 pts/1    Ss     0:00 -bash
/usr/bin/zshを実行する
$ /usr/bin/zsh↵
現在有効になっているシェルがzshであることが確認できる
% ps $$↵
  PID TTY      STAT   TIME COMMAND
 6550 pts/1    S      0:00 /usr/bin/zsh
```

▼図3　ターミナル起動画面



▼図4　suコマンドを実行してプロンプトが変わる例



渡して実行しています。これはlsコマンドに、/varを操作対象とするよう司令を出しています。結果、/var以下のディレクトリ・ファイル名がリスト出力されていますね。

lsコマンドのオプションは「-文字列」のように、「-」から始まる文字列をオプションと認識しますが、コマンドによっては「-」なしでもオプションとして解釈する場合があります。たとえば、tarコマンドでは圧縮ファイルを伸長する「-x」オプションを指定するのに「x」だけでも解釈してくれます。コマンドによって若干作法が変わるので、使い方がわからない場合は積極的にコマンドマニュアル（manコマンド）や『[改訂三版]Linuxコマンドポケットリファレンス』（技術評論社刊）などの書籍を参照しましょう。

▼図5　オプションや引数を入れたコマンド実行例（ls）

```
$ ls↵        ①
$            何も出力されずにプロンプトが表示される
$ ls -a↵     ②
.  ..  .dotfile
$ ls -al↵    ③
合計 20
drwxr-xr-x      2 ryosuke   ryosuke     4096  4月  7 01:02 .
drwxrwxrwt     18 root      root       12288  4月  7 01:02 ..
-rw-r--r--      1 ryosuke   ryosuke       17  4月  7 01:02 .dotfile
$ ls /var↵   ④
backups  cache  games  lib  local  lock  log  mail  opt  run  spool  tmp
```

▼図6　パスの確認

```
$ visudo↵
visudo: command not found
↑実行したらコマンドが見つからないとエラーになった
$ echo $PATH↵　変数名の前に「$」をつけてechoに渡すと代入されている値を参照できる
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
$ locate visudo | grep bin↵　実行ファイルはbinディレクトリにインストールされる、と当たりをつけて「grep bin」でフィルタする
/usr/sbin/visudo
↑環境変数PATHに入っていないディレクトリに配置されているので「not found」になったとわかる
```

▼図7　シェル変数を環境変数にするexportの使い方

```
$ cat var_test.sh↵　変数VALを出力するスクリプト「var_test.sh」の中身を表示
#!/bin/bash

echo VAL is ${VAL}.
$ echo $VAL↵
        ←echoコマンドで出力しても空行になることで変数VALが空っぽであることを確認
$ VAL=val↵              変数VALに「val」を代入する
$ echo $VAL↵            変数VALの中身を出力する
val                     代入した文字列が出力される
$ bash ./var_test.sh↵   先程作ったスクリプトを実行する
VAL is .                変数VALに「val」が入ってないのがわかる
$ export VAL↵           exportコマンドを使って変数VALを環境変数にする
$ bash ./var_test.sh↵   再度実行する
VAL is val.             「val」が表示される
```

実行ファイルのありか ～パスを探して三千里

シェルはコマンドとして指定された実行ファイル名を探します。実行ファイルが格納されている複数のディレクトリパスを環境変数PATHとして保持しており、そのパスのみを探します。たとえば/opt/bin/mkfugaという実行ファイルがインストールされているとして、環境変数PATHにこの/opt/binディレクトリが登録されていないとユーザはmkfugaだけでは実行できず、「/opt/bin/mkfuga」と指定しなければシェルはコマンドを実行できません。

一般ユーザと管理者権限で環境変数PATHが異なることがあるので、「あれ？　インストールされてるはずなのに実行できないな」という場合はPATHの値をechoコマンドで出力して確認し、その実行ファイルが本当にインストールされているかをlocateコマンドなどで調べましょう。**図6**では、visudoというコマンドのありかにパスが通っているかを確認しています。

ビルトインコマンド

シェルにはシステムにインストールされる実行ファイルのコマンドとは別に、「ビルトインコマンド」もしくは「組み込みコマンド」と呼ばれる、シェル自身が持っているコマンドがあります。代表的なビルトインコマンドにはcd(作業ディレクトリを変更する)、pwd(作業ディレクトリを表示する)、exit(シェルを終了する)などがあります。

pwdやkill(プロセスにシグナルを送る)などは、ビルトインコマンドと同名の実行ファイルがあります。両方存在するコマンドは、/bin/killのように指定して実行しない場合、ビルトインコマンドが優先されます。実行されるコマンドがビルトインか実行ファイルかは、ビルトインコマンドのtypeで確認できます。

```
$ type kill↵
 killだけ指定するとビルトインコマンドが使われる
kill はシェル組み込み関数です
$ type /bin/kill↵
 PATHを指定するとビルトインコマンドが使われない
/bin/kill は /bin/kill です
```

シェル変数・環境変数

実行中のシェルで変数を保持することができます。変数には数値や文字列を代入できます。変数に代入された値を参照する場合は変数名の前に「**$**」をつけます。

実行中のシェルで利用する変数は「シェル変

数」と言いますが、別のシェルやスクリプトを実行する際に、その変数を共有する場合は「環境変数」に変更する必要があります。シェル変数を環境変数にするには、ビルトインコマンドのexportコマンドを利用します。**図7**の例を見てみましょう。

変数値は**$VAL**でも**${VAL}**でも同じように参照できます。「**${}**」で囲うほうが変数名の区切りがわかりやすいので可読性も上がりますし、ほかの変数名に似たものがある場合、事故にならないのでオススメです。ここでは簡単に参照する用途として囲わない書式を利用しました。

データ入力、結果の出力、それをつなげるもの

「単純な機能のコマンドを組み合わせて目的を達成する」というUNIXの考え方があります。例として「ある時間、特定のIPからWebサーバへのアクセス数をカウントしてファイルに出したい」という目的を達成するとしましょう。これを簡単な機能で分割させると次になります。

・httpdサーバのアクセスログを開く
・探したい日時の行を探して抽出する
・アクセス元IPを選ぶ
・カウントする
・アクセス数をファイルに出力する

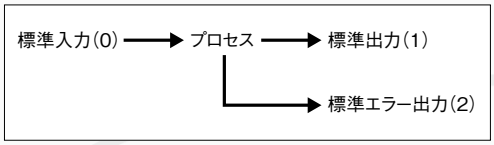
これから、探したい日時を2016年3月20日、アクセス元IPをAA.BB.CC.DDとすると、次のようなコマンドライン[1]が構築できます。

```
[1]$ cat /var/log/apache2/access.log | grep '20/Mar/2016' | grep AA.BB.CC.DD | sort | wc -l > /tmp/count.txt
```

上記コマンドからcatは単独で実行すると画面にファイルの内容や検索結果を出力しますが、grepとsort、wcを実行する場合は対象ファイル名を引数に指定する必要があります。ですが、ここではあえて指定しません。

これは各コマンドの出力を別のコマンドの入力として渡しているからです。1つのコマンドで簡単な結果を出力し、別のコマンドにその結果をさらに処理させることで目的を達成させます。これらのコマンドの組み合わせは自由で、次のコマンドライン[2]では[1]の最初のcatとgrepを1つにまとめていますが、同じ結果を得られるはずです。

▼図8　標準入力・標準出力・標準エラー出力




```
[2]$ grep '20/Mar/2016' /var/log/apache2/access.log | grep AA.BB.CC.DD | sort  | wc -l > /tmp/count.txt
```

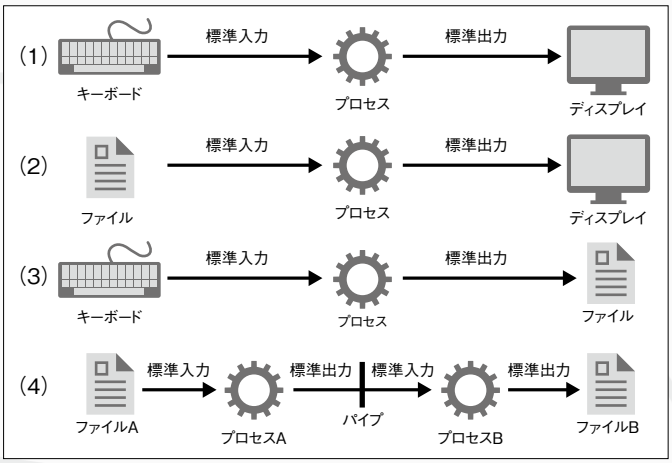
このような“コマンドの出力を別のコマンドの入力に連結させるしくみ”を「パイプライン」と言い、「|」をコマンドの間に挟むことでつなぐことができます。

コマンドは、実行してプロセスになった際に標準入力・標準エラー出力・標準出力の3つの入出力チャネルが用意されます。標準入力はプロセスへの入力、標準出力はプロセスの出力、標準エラー出力はプロセスのエラーメッセージをそれぞれ入出力します(**図8**)。

図9の(1)のように、標準入力はキーボード、標準出力はディスプレイになります。grepのようにテキストファイルからパターンの存在する行をディスプレイや端末に出力する場合は**図9の(2)**になります。**図9の(3)**はキーボードからの入力を受け、プロセスが出力用ファイルを作成し、そこに結果を書き出します。wgetのようにキーボードからURLを受け取り、URL末尾のファイル名と想像されるファイルを作成してそこに出力する場合がこれに当たります。**図9の(4)**はプロセスの出力と入力をつなぐ「パイプライン」を利用するイメージです。

コマンドライン[2]の最後、wcの後にある「**> /tmp/count.txt**」は標準出力をファイルに出力

▼図9　標準入出力とパイプ



した例です。プロセス同士をつなぐ場合はパイプライン「|」、プロセス以外のファイルやデバイスに入出力を変える場合はリダイレクト「<」「>」を使います。

シェルはコマンドラインにリダイレクトの文字があれば入出力チャネルを切り替えます。grepに渡すファイルを標準入力から流す例を見てみましょう。テキストファイルを用意し(例：stdin.txt)、「<」でコマンドにファイル名を引数に渡す指定をします。ファイルを左側に持ってくるとシェルはコマンドと解釈し実行するので注意しましょう。「>」は標準出力のリダイレクトです。

```
$ cat stdin.txt↵
愛知
岐阜
三重
$ grep 岐阜 < stdin.txt↵
岐阜
$ stdin.txt > grep 岐阜↵
-bash: stdin.txt: コマンドが見つかりません
```

grepの標準出力をテキストファイルに同時に出してみましょう。

```
$ grep 三重 < stdin.txt > stdout.txt↵
$ cat stdout.txt↵
三重
```

リダイレクトの記号「<」「>」が矢印に見えるので、コマンドラインだけみてると「はて？　stdin.txtがgrepにもstdout.txtにも行ってるようにみえるような……」と思うことがありますが、「コマンドラインは左から解釈される」ルールを覚えていれば怖くありませんよ。

標準エラー出力をリダイレクトする際は「2>」と、数字をつける必要があります。エラーが出るコマンドを実行(権限のないファイルを表示しようとしてみる)するとメッセージがでます。次のように「2>」でファイルに標準エラー出力をリダイレクトすると画面にエラーメッセージは表示されず、ファイルに書き込まれていることが確認できます。

```
$ less /var/log/syslog↵
/var/log/syslog: 許可がありません
$ less /var/log/syslog 2> stderr.txt↵
$ cat stderr.txt↵
/var/log/syslog: 許可がありません
```

「2」はファイルディスクリプタ番号です。ファイルディスクリプタは、プロセスがアクセスするファイル情報の番号です。標準入力は「0」、標準出力は「1」とそれぞれ割り当てられていますし、実は標準出力リダイレクトは「<0」、標準出力は「1>」と書くべきところを、ファイルディスクリプタ番号を省略できるだけです。

標準出力はプロセスの出力、標準エラー出力はエラーメッセージと出力を分けてあることは結果にノイズが入らないという意味ではありがたいのですが、デバッグ時など出力をそれぞれ分けて見るのは手間と感じることがあります。その際は、標準出力と標準エラー出力を同時に出力することで対応できます(**図10**)。

シェルはコマンドラインを左から解釈します。まずコマンドを見て(**図10**では**find /var/log/samba**)、「**> /tmp/find.txt**」から標準出力を/tmp/find.txtに換えます。次に、「**2>&1**」で標

▼図10　標準出力と標準エラー出力を同時に出力する

```
$ find /var/log/samba/↵                          標準出力、エラー出力をそのまま画面に出す
/var/log/samba/
find: `/var/log/samba/': 許可がありません
$ find /var/log/samba/ > /tmp/find.txt↵    「>」では標準出力だけが記録される
find: `/var/log/samba/': 許可がありません
$ cat /tmp/find.txt↵
/var/log/samba/
$ find /var/log/samba/ 2> /tmp/find.txt↵   「2>」では標準エラー出力だけが記録される
/var/log/samba/
$ cat /tmp/find.txt ↵
find: `/var/log/samba/': 許可がありません
$ find /var/log/samba/ > /tmp/find.txt 2>&1↵   これなら両方記録される
$ cat /tmp/find.txt↵
/var/log/samba/
find: `/var/log/samba/': 許可がありません
```

準エラー出力(2)に標準出力(1)の出力先をコピーする意味があります。ここではコマンドラインの行末までくると、2つの出力先が/tmp/find.txtになります。

順番を間違えると期待どおりにいかない場合があります。次の順番ではエラーメッセージがファイルに記録されていません。

```
$ find /var/log/samba/ 2>&1 > /tmp/find.txt↵
find: `/var/log/samba/': 許可がありません
```

シェルはコマンドラインを左から解釈する(2度目)ので、コマンドとオプションの後の「**2>&1**」がまず解釈されて標準出力先を標準エラー出力にコピーします。とくに標準出力を変更していないので2つの出力先は標準出力である「画面」になります。その後標準出力を「/tmp/find.txt」に変更するのですが、ここで標準エラー出力は、先ほどの標準出力のコピーのままなので画面に出力されます。

出力リダイレクトは「**>**」と別に「**>>**」も使えます。「**>**」は実行ごとにファイルを新しく作成する、つまり前に同名のファイルがあれば削除されますが、「**>>**」でリダイレクトすると既存ファイルを削除せずに追加で出力を記録していきます。

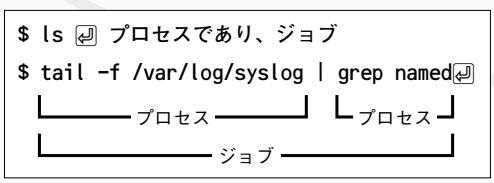
リダイレクトに似た機能で「ヒアドキュメント」というのも覚えておきましょう。コマンドラインから複数行をプロセスの入力にすることができます。「**<<文字列**」をコマンドに向けて記述し、入力する文字列を入力、最後に指定した「**文字列**」を記載すると入力終了の合図になります。次の例では、PREFを終了文字列に指定してcatコマンドにテキストを入力しています。catの-nオプションは行番号を含めて出力するので期待どおりの結果が出ていますね。

```
$ cat -n <<PREF↵
東京↵
神奈川↵
千葉↵
埼玉↵
PREF↵
     1  東京
     2  神奈川
     3  千葉
     4  埼玉
```

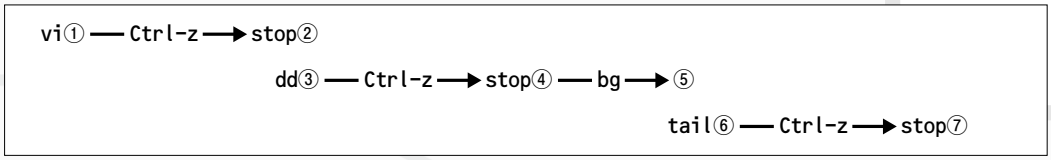
ジョブ制御して複数プロセスを同時実行する

コマンドを実行するとプロセスになります。このプロセスはシェルから見ると1つのジョブです。複数のコマンドを「|」などで組み合わせて実行した場合、それぞれはプロセスですが、まとまった1つのジョブとして扱われます(**図11**)。

▼図11　プロセスとジョブの関係



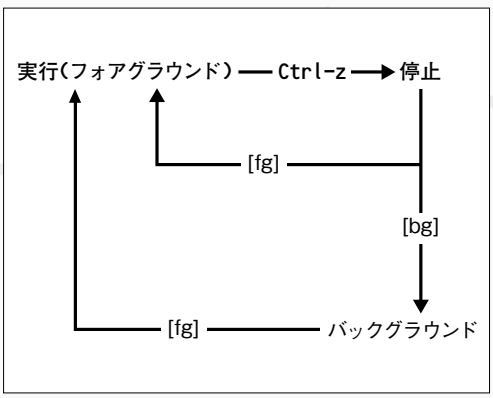
▼図12　プロセス起動と停止・復帰の例



▼図13　プロセス起動と停止・復帰のコマンドライン例

```
$ vi test.txt↵  viを起動①
[1]+  停止            vi test.txt  Ctrl-zを入力してviを停止②
$ dd if=/dev/zero of=/tmp/zerotest bs=1024 count=1024000↵  ③
^Z  ddコマンドを実行してすぐにCtrl-zして停止④
[2]+  停止            dd if=/dev/zero of=/tmp/zerotest bs=1024 count=1024000
$ bg 2↵  ddを止めたときの[2]でジョブIDが2とわかったのでバックグラウンドで再開⑤
[2]+ dd if=/dev/zero of=/tmp/zerotest bs=1024 count=1024000 &
$ sudo tail -f /var/log/syslog↵  tailコマンドを新たに実行⑥
Apr  9 03:54:07 mustain smartd[666]: Device: /dev/sda [SAT], SMART Usage Attribute: 190
Airflow_Temperature_Cel changed from 64 to 63
Apr  9 03:54:07 mustain smartd[666]: Device: /dev/sda [SAT], SMART Usage Attribute: 194
Temperature_Celsius changed from 111 to 110
^Z  Ctrl-zで停止⑦
[3]+  停止            sudo tail -f /var/log/syslog
$ jobs↵  ジョブ一覧を表示
[1]-  停止            vi test.txt
[2]   実行中          dd if=/dev/zero of=/tmp/zerotest bs=1024 count=10240000&
[3]+  停止            sudo tail -f /var/log/syslog
$ ls -lh /tmp/zerotest↵  バックグラウンドで動いているジョブID2の結果を確認
-rw-r--r-- 1 ryosuke ryosuke 2.3G  4月  9 05:41 /tmp/zerotest
$ ls -lh /tmp/zerotest↵  もう一度確認
-rw-r--r-- 1 ryosuke ryosuke 2.4G  4月  9 05:41 /tmp/zerotest
↑出力ファイルが大きくなっているのでバックグラウンドで動作が継続しているのを確認できる
```

▼図14　プロセス起動と停止・復帰のサイクル図



シェルはこのジョブを「停止」「フォアグラウンドで実行」「バックグラウンドで実行」することができます。

ユーザがシェルを使えるようにプロセスを止めるのは「停止」、ユーザから見える・ユーザからの入力を受け付けるジョブを「フォアグラウンドジョブ」、ユーザから見えない・入力も受け付けないが処理を継続するジョブを「バックグラウンドジョブ」と言います。

このジョブの切り替わりを**図12**で追ってみましょう。viエディタでファイルを編集しようと開いてみます①。viが動いてるところでCtrl-z(Ctrlキーを押しながらzキーを押す操作)を入力するとそのプロセスが一時停止します②。次に「**dd if=/dev/zero of=/tmp/zerotest count=1024000**」を実行してみましょう③。このコマンドは512byteを1,024,000回書き込んで5GBの/tmp/zerotestを作成します。これも実行中にCtrl-zを押して一時停止します④。今回は停止しただけではなくて、bgコマンドでバックグラウンドジョブとして処理を継続させましょう⑤。次に「**sudo tail -f /var/log/syslog**」を実行しましょう⑥。これもCtrl-zで止めます⑦。この

一連の動作はコマンドラインで**図13**のようになります。

このように、ジョブを実行(フォアグラウンド)→停止→バックグラウンド→実行(フォアグラウンド)を循環することで、1つのシェルで複数のプロセスに司令を出したり、出力を観察することができるのです(**図14**)。

便利に使おうbash

bashが読み込むファイル

シェルは2種類の起動モードがあります。システムのログイン画面でユーザ名とパスワードの認証を通過した後に起動する「ログインシェル」と、それ以外の「対話的シェル」があり、起動時に読み込むファイルが異なります。ログインシェルとしてbashが起動した場合、まず/etc/profileが読み込まれます。起動後に読み込まれるファイルは次の順番になります。

1. /etc/profile
2. ~/.bash_profile
3. ~/.bash_login
4. ~/.profile

一方、対話的シェルでは/etc/bash.bashrcか~/.bashrcだけを読み込みます。

コマンドラインを快適に動こう

シェルはシステムに向かって一番長く使うツールです。使う時間が長い分、快適に・ストレスなく利用することが健全な業務・趣味への第一歩になります(たぶん)。快適な環境を手に入れるためにシェルの操作方法を体に覚えこませましょう。

表1はシェル内のカーソル移動や編集に使える主なショートカット一覧です。移動や編集以外にも便利なショートカットキーがあるので章の最後で紹介します。こちらもあわせて覚えたいですね。

▼表1　シェル内移動・削除・カット&ペーストのショートカット

コマンド	操作
Ctrl-a	行頭へ移動
Ctrl-e	行末へ移動
Ctrl-b	一文字左へ移動
Ctrl-f	一文字右へ移動
Alt-b	一単語左へ移動
Alt-e	一単語右へ移動
Ctrl-d	カーソルの文字を消す
Ctrl-h	カーソルの左の1文字を削除
Ctrl-w	カーソルのある位置一単語カット。単語の中にいる場合は単語先頭まで削除
Ctrl-k	カーソルより右にある文字を全てカット
Ctrl-u	カーソルより左にある文字を全てカット
Ctrl-y	カットして保持している文字をペースト

※Ctrl-aはCtrlキーを押しながらaを押す操作

知ってると使いたくなる画面クリアと入力補完キー操作

移動や編集以外のショートカットキーもあります。使いこなせているとモテるかもしれないので覚えておきましょう(保証はしません)。

まずは画面をclearする機能です。コマンド入力や結果出力後の表示文字を、書き込んだ紙を破って捨てるようにキレイにする機能です。これをCtrl-lと入力するとサッパリできます。

bashの入力補完機能はぜひおさえておきたい機能です。入力補完とはユーザの入力を補助することですが、bashの入力補完機能はコマンドの途中まで入力して[Tab]を押すと、その後の入力を自動で補ってくれます。

```
$ geni[Tab]
      ↓
$ genisoimg
```

[Tab]を押したところで複数の候補がある場合は、予想変換のようにその後に続く候補を表示してくれます。

```
$ sudo mkfs.ext[Tab]
mkfs.ext2  mkfs.ext3  mkfs.ext4  mkfs.ext4dev
```

ファイルだけではなく、/etc/hostsに記述し

▼図15　bash-completionによるオプションの補完例

```
$ ls -[Tab][Tab]   オプション指定時の「-」だけ入力してTabを2回押せばオプション候補が表示される
--all               --directory          --ignore=                --show-control-chars
--almost-all        --dired              --indicator-style=       --si
(..略..)
```

ているホスト名について補完することもできます。/etc/hostsはホスト名とIPアドレスの関係を記述する設定ファイルの1つで、DNSへの名前解決前に参照するのでシステム起動時、ネットワーク未接続でDNSに問い合わせできない場合などに記述して利用します。

何も入力していない状態でAlt-@をタイプすると、/etc/hostsに記述されているエントリすべてを候補として表示します。

```
$ Alt-@を入力
::1         ff02::1       oakley
axl         ff02::2       oakley.example.com
(..略..)
```

途中まで入力した文字列にマッチするホスト名があれば、Alt-@を一度タイプするとホスト名が、2回タイプするとホスト名とFQDNが候補として表示されます。

```
$ oak [Alt-@]
oakley[Tab][Tab]
$ oak [Alt-@] [Alt-@]
oakley              oakley.example.com
```

インストールされていないコマンドを実行しようとすると、「そのコマンドはＸＸＸというパッケージにあるのでインストールしてください」と補完(？)してくれる機能もあります。ディストリビューションによってはパッケージ名を教えてくれたり、直接インストールしてくれる場合もありますが、DebianとUbuntuではcommand-not-foundという名前のパッケージをインストールすれば同様の機能が利用できます。command-not-foundパッケージをインストールする前は、不明なコマンドを入力すると冷たくあしらわれますが、

```
$ apg↵
-bash: apg: コマンドが見つかりません
```

インストール後は、ちょっと優しくなります。

```
$ apg↵
The program 'apg' is currently not
installed.  To run 'apg' please ask your
administrator to install the package 'apg'
apg: command not found
```

bash-completionパッケージをインストールすると、~/.ssh/configを見てsshコマンドの後に続くホスト名を補完してくれます。また、コマンドオプションの候補表示、補完もできます(**図15**)。著者の記憶が確かなら、bash-completionはzshの補完機能からポーティングされたと認識しています。違ってたらごめんなさい。でもこれも便利な機能ですよね。

コマンド履歴利用で、もう長いコマンドなんて打たないよ絶対

希望する処理によっては数多くの長いオプション、長い引数を指定することがあります。再度同じコマンドを実行すればすむ場合はコマンド履歴を利用しましょう。

直前のコマンドを再実行したい場合はたった2文字の「**!!**」で実行できます(**図16**)。

historyコマンドで履歴を出力すると行頭に履歴IDが表示されるので、これを使って再実行もできます(**図17**)。

ちょっとした間違いの後でもう一度そのコマンドラインを入力するかと思うと……二度とシェルを使いたくないですね。それなら履歴を使ってコマンドラインを呼び出し、カーソルを移動して間違ったところをなおせばいいのです。

単純に直前の実行履歴なら、上矢印キーか、Ctrl-pを打つことでコマンドラインに出てきま

▼図16　直前のコマンドを再実行

```
$ ssh -l ryosuke -i ~/.ssh/id_rsa-prime -p 10022 -6 prime↵   コマンド例
$ !!↵   再実行
ssh -l ryosuke -i ~/.ssh/id_rsa-prime -p 10022 -6 prime
```

す。数回押すことでさらに前の履歴にもさかのぼれます。さかのぼり過ぎたら、Ctrl-nか下矢印キーで新しい履歴のほうに戻ることができます。

矢印キーを押し続けるにはつらいぐらい昔の履歴の場合は検索が楽ですね。Ctrl-rを押して検索文字列を入力すると履歴から表示されます。さらにCtrl-rを続けて押せば次に古い検索結果にマッチする実行履歴が表示されます。目的の履歴までたどり着いたら↵を押せば実行できます。↵ではなくTabを押せば、そのコマンド履歴を編集して実行できるようになります(**図18**)。

検索結果をたどりすぎた場合は、Ctrl-sで新しいほうの検索結果に戻ることができます。ただし、Ctrl-sはターミナルのロックキーに割り当てられていることがあり、この機能を利用する場合はsttyなどでロックしないように設定変更する必要があります(**図19**)。

▼図17　historyコマンドの履歴から再実行

```
$ history 3↵
 2243  ssh oakley
 2244  ssh endeaver
 2245  history 3
$ !2244↵   履歴ID 2244は「ssh endeaver」。これを再実行
ssh endeaver
```

検索結果の履歴から新しいほうに戻る場合はプロンプトの表示が変わります(**図20**)。

直前のコマンドがわかっていて、少しだけ訂正して再利用する場合は「**^**」で置換して実行できます(**図21**)。ただし、この「**^**」を利用した履歴置換は、コマンドライン左側から最初に見つかった文字のみが置換対象になります。

行すべてを対象とする場合はちょっと入力数が増えるのですが**図22**のように行います。直前の履歴である「**!!**」で、「**:**」の後に行全体に置換を適用する「**g**」スイッチを指定し、置換オプション「**s/置換前/置換後/**」と指定します。最

▼図18　検索を使った履歴利用

```
$ Ctrl-r
(reverse-i-search)``:   検索の入力プロンプト
(reverse-i-search)`ssh': ssh -l ryosuke srv01.example.com
                  ↑「ssh」と入力して検索結果から一番新しいコマンドが表示される
(reverse-i-search)`ssh': ssh -l ryosuke prime   さらにCtrl-rを押すと、もうひとつ古い検索結果履歴が表示される
```

▼図19　Ctrl-sがロックキーになっていた場合の設定変更

```
$ stty -a
↑出力の中に「stop = ^S」があれば、ターミナルがCtrl-sでロックされる
$ stty stop undef   stopを未定義に設定する
$ stty stop ^S   stopを戻す場合は「^S」を定義する
```

▼図20　Ctrl-rからCtrl-sへの切り替え

```
$ Ctrl-r
(reverse-i-search)``:   Ctrl-rの検索の入力プロンプト
(reverse-i-search)`ssh': ssh prime
                ↑「ssh」と入力して検索結果を古いほうにたどる
(i-search)`ssh': ssh prime
                ↑Ctrl-sを押して新しいほうの検索結果をたどる
```

▼図21　^を使った置換

```
$ ssh srv01.example.com↵
$ ^1^2^↵
ssh srv02.example.com
$ scp arch.tgz ryosuke@192.168.1.22:/tmp/↵
$ ^.1.^.5.^↵
scp arch.tgz ryosuke@192.168.5.22:/tmp
```

▼図22　高度な置換

```
$ cp img_016.jpg img_016-01.jpg↵
$ !!:gs/16/17/↵
cp img_017.jpg img_017-01.jpg
```

▼表2　コマンド履歴操作

検索	ショートカットキー	備考
直前のコマンド実行	!!	―
直前の実行コマンド検索	Ctrl-pもしくは上矢印	さらに前の履歴に行く場合は同じキーを叩く
履歴を新しい方に戻る	Ctrl-nもしくは下矢印	Ctrl-pや上矢印キーでたどった履歴を戻る
最新実行履歴を複数表示	history［数値］	指定した数値だけ履歴を表示する
履歴IDから再実行	![履歴ID]	historyコマンドで調べた履歴IDを指定する
履歴の検索	Ctrl-r	続けて検索文字を入力する。求めるコマンドラインが表示されたらTABを押せばそれを編集できる
履歴検索を戻る	Ctrl-s	検索結果でたどった履歴を戻る。Ctrl-sがターミナルのロックに割り当てられている場合はsttyなどで解除する必要がある
直前のコマンドを置換して実行	^置換対象文字^変更文字列^	複数対象文字の左隅しか置換されない
直前のコマンドの文字列置換を行全体対象にして実行	![履歴ID]:gs/置換前/置換後/	―

▼表3　パス名展開

特殊文字	意味	用途例
*	任意の文字列	IMG_*.jpgと指定すると先頭がIMG_で始まり、拡張子が.jpgで終わるファイル名に展開される
?	任意の1文字	IMG_0?.jpgと指定するとIMG_01.jpgからIMG_09、IMG_0a.jpgからIMG_0z.jpgなどに展開される
[ ]	囲まれた中の任意の文字	IMG_[0-9][0-9].jpgではIMG_00.jpgからIMG_99.jpgまで展開される

▼表4　チルダ展開

チルダ	意味	用途例
~	ホームディレクトリ	cd ~は引数なしのcdコマンド実行結果と同じ
~-	直前にいた作業ディレクトリ	cd ~-はcd -と同じ
~+	今の作業ディレクトリ	pwdの実行結果と同じ

▼表5　ブレース展開の例（1）

例	展開前	展開後
cpの例	cp script_01.txt{,.bak}	cp script_01.txt script_01.txt.bak
mkdirの例	mkdir -p rpm/{RPMS,SOURCES,SPECS,SRPMS}	mkdir -p rpm/RPMS rpm/SOURCES rpm/SPECS rpm/SRPMS
mvの例	mv example.txt{.bak,}	mv example.txt.bak expample.txt
touchの例	touch {1,11,111,1111}	touch 1 11 111 1111

初の「!!」は履歴IDさえわかれば「!2024:gs/16/20/」のように変更することもできます。

ここで紹介したコマンド履歴の操作一覧を**表2**にまとめます。

「展開」して入力数を減らして楽しよう

コマンドラインはどれだけ入力数を減らせるかが仕事を楽にすることにつながります。Tabでのファイルやコマンドの補完はそれを実践する例ですが、特別な文字を使って入力文字数を減らすこともできます。「覚えたもん勝ち」の「展開」を極めてみませんか？

とくに意識せずによく利用しているのは「*（アスタリスク）」を使ったパス名展開だと思います（**表3**）。「任意の文字列」に展開される「*」は「*.jpg」や「*.txt」など特定パターンと組み合わせて利用すると思います。パス名展開にはほかに任意の1文字に添加する「?」や、「[ ]」で囲んだ内の任意の1文字に展開するパターンがあります。

まず、すでに出てきた「~（チルダ）」はホーム

▼表6　ブレース展開の例（2）

例	展開前	展開後
touchの例（1）	touch hello_{1..5}	touch hello_1 hello_2 hello_3 hello_4 hello_5
touchの例（2）	touch hello_{a-f}	touch hello_a hello_b hello_c hello_d hello_e hello_f
touchの例（3）	touch hello_{{1..3},{a..c}}	touch hello_1 hello_2 hello_3 hello_a hello_b hello_c
forの例	for i in {1..3}; do echo $i done	for I in 1 2 3; do echo $i done

ディレクトリPATHに展開されます（**表4**）。「**~-**」とすると環境変数OLDPWDに入っている1つ前にいた作業ディレクトリ、「**~+**」とすると環境変数PWDに入っている今の作業ディレクトリPATHに展開できます。1～4文字も節約できるなんて儲けモンですね。

「**{**」「**,**」「**}**」でcpやmv、複数ディレクトリ作成などがすこぶるはかどるブレース展開も便利です。書き方は2通りあります。

```
[前置詞]{文字と「,」}[後置詞]
```

省略可能な「前置詞」と「後置詞」の間に「**{ }**」と文字例と「**,**（カンマ）」を使うと、前置詞＋文字列＋後置詞を展開します。この場合のブレース展開の例を**表5**に示します。

```
[前置詞]{始..終}[後置詞]
```

省略可能な「前置詞」と「後置詞」の間に「**{ }**」で指定された範囲の文字列で、前置詞＋文字列＋後置詞を展開します。この場合のブレース展開の例を**表6**に示します。

「展開」はちょっと使い慣れるとコマンドするのが楽しくなるスパイスだと思っています。

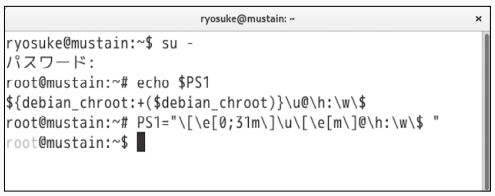
bashをカスタマイズしたい

bashの設定ファイル.bashrcを編集して使いやすい環境に変更してみましょう。

プロンプトのカスタマイズ

見た目が味気ない（と思う人もいる）プロンプトを変更してみましょう。ユーザ名、ホスト名、

▼図23　rootユーザの.bashrcにカラープロンプトを設定してわかりやすくする（グレー部分が赤字）



作業ディレクトリが表示されているプロンプトが出ていると思います。これは環境変数PS1に指定されている値を編集することで柔軟に変えることができます。色をつけたり、時刻を表示したり、複数行にすることもできます。

簡単な例として、自分の今のPS1の値を確認し、時計を表示するようにしてみましょう。

```
ryosuke@mustain:~$ echo $PS1↵
 現在のPS1を表示する
\u@\h:\w\$
ryosuke@mustain:~$ PS1="\u@\h:\w:\t\$ "↵
 ↑途中に「\t」を挿入する
ryosuke@mustain:~\:03:28:38$
 プロンプトに時刻が表示されるようになった
```

エスケープする文字が多かったりで非常に編集しにくいですが、一度凝り始めるとなかなか諦められなくなりますので時間があれば試してみましょう。

rootユーザを使っている場合はユーザへの警告にユーザ名部分を赤色にしてみましょう。rootユーザのbash設定ファイル/root/.bashrcにPS1を設定すればrootユーザになった際に反映されます。プロンプトの色はASCIIカラーコードを指定します。

```
PS1="\[\e[0;31m\]\u\[\e[m\]@\h:\w\$ "
```

最初と@前の「**\[**」と「**\]**」はプロンプト場所調

▼リスト1　aliasの設定例

```
alias printdate='date +%Y-%m-%d'
alias ytdlmp3="youtube-dl -x  --audio-format mp3"
```

▼リスト2　bash設定ファイルの一部抜粋

```
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
        PATH="$HOME/bin:$PATH"
fi
```

▼図24　一行で書いたwhile文の例

```
$ while [ "$i" -le 100 ]; do mv IMG_$i.jpg img_$(printf %03d $i).jpg; i=$((i+1)); done
```

▼図25　if文を使って3の倍数に「-AHO」を入れる

```
$ for i in {1..100}; do
  if [ $(( $i % 3)) -eq 0 ]; then
    mv IMG_$i.jpg img_$(printf %03d $i)-AHO.jpg
  else
    mv IMG_$i.jpg img_$(printf %03d $i).jpg
  fi
done
```

整用です。色は「\e[色指定\]」と「\[\e[m\]」で囲みます。\uはユーザ名、\hはホスト名、\wは作業ディレクトリを表示します(図23)。

Arch Linuxのbashプロンプト資料がとてもわかりやすくて有用だったので、参考にすると良いでしょう。

・Bash カラープロンプト(Arch Linux)
https://wiki.archlinuxjp.org/index.php/Bash_カラープロンプト

別名設定でオプション忘れるぐらい楽しよう

よく使うコマンドがある、しかもオプションもいつも変わらない。そんなときには別名を設定できるaliasを設定しましょう。たとえば著者はdateの日時出力フォーマットをいつも忘れるので、簡単aliasにして便利に使ってます。

いつも使うならば~/.bashrcに記述しましょう。一例をリスト1に挙げます。printdateは日付を「2016-02-29」形式で出力するので、「cp -a /etc ./backup-etc-$(printdate)」でバックアップする際に便利です。ytdlmp3はyoutube-dlという動画サイトから動画をダウンロードするコマンドを使いますが、音声だけをmp3で変換出力するaliasです。

~/.bashrcに記述した後は「source ~/.bashrc」を実行すると編集したaliasが有効になります。「alias -p」でalias一覧を表示して確認もしましょう。ちなみに、aliasを解除する場合は、unaliasコマンドを使います。

自作コマンドをインストールされたコマンドのように実行したい

bashの設定ファイルのPATHに特定のディレクトリを作り、そこに実行ファイルを配置しましょう。最近のLinuxディストリビューションの一部では、bashの設定ファイルにリスト2の数行が入っています。

もし入ってなければ追記しておきましょう。これは、環境変数HOMEで参照できるユーザのホームディレクトリ以下にbinディレクトリが存在すれば、環境変数PATHにそのbinディレクトリを追加する処理です。これにより、~/bin/ディレクトリに作成したスクリプトや実行ファイルを配置し、chmodコマンドで実行権を付加(例：chmod +x 実行ファイル)することで「/home/ryosuke/bin/実行ファイル」と絶対パスを指定しなくてもプログラムを実行できるようになります。

条件制御・繰り返し実行

forで繰り返す

シェルは条件制御や繰り返しもカジュアルに使えます。このような処理は、CやJava、Rubyのほうが効率が良いかもしれません。100個程

度のファイルの名前を変えるといった、プログラムを書くほどでもないただの繰り返し対応が可能な場合など、状況によってはシェルで処理できるのが便利ですよ。

たとえば、あるディレクトリにある画像ファイルIMG_1.jpgからIMG_100.jpgを全部小文字に、3桁数値のファイル名にしたいと思ったら次のように実行します。

```
$ for i in {1..100}; do
  mv IMG_$i.jpg img_$(printf %03d $i).jpg;
done
```

forは次の書式で繰り返し処理を指定できる構文です。

```
for 変数名 [ [ in 単語リスト ] ; ] do 〆
処理 ; done
```

上記の例では、変数名は「i」です。inの後に続くリストを展開し、それを処理で利用できます。ここでは「**{1..100}**」をリストとして指定します。これは1から100までをリストする書式です。処理はmvコマンドを実行します。「**printf %03d $i**」は、有効桁数3桁で前の空白は0で埋めて出力するprintfコマンドを利用して3桁数値を使います。これにより「**mv IMG_001.jpg img_001.jpg**」が順番に1から100まで繰り返されます。

forはC言語風にも書けるので次のものでも同じ処理になります。

```
$ for ((i=1; i<=100; i++)); do
  mv IMG_$i.jpg img_$(printf %03d $i).jpg
done
```

whileで繰り返す

forに続き、whileも繰り返しする構文です。

```
while リスト; do 処理; done
```

リストが、終了ステータス0を返せば「処理」を実行します。次のものはforで処理したのと同じ内容です。ここでリストは「**["$i" -le 100]**」です。変数iが100以下なら「処理」のmvコマンドを実行します。

```
$ i=1
$ while [ "$i" -le 100 ]; do
  mv IMG_$i.jpg img_$(printf %03d $i).jpg
  i=$((i+1))
done
```

この例では見やすいように改行していますが、各行を「**;**」で区切れば**図24**のように1行で実行することもできます。

ifで思いとどまってみる

ifで、条件にマッチした場合に処理を行うことができます。

```
if リスト1; then リスト2;  else リスト3; fi
```

リスト1を実行した結果が0ならばリスト2を処理、0以外ならばリスト2は処理せずにリスト3を実行します。本来ならばもう少し詳しく「elif」も説明しなければならないと思いますが、ここはbash再入門の場、気になった方はマニュアルを見てください。

さて、せっかく条件分岐ができるわけなので、先程から処理している画像ファイル名の変更で、3の倍数だけ「-AHO」の文字を入れることにしましょう(**図25**)。もうこのネタ知ってる人いないんじゃないかと不安ですが。

for文の中にifとelseの文が入っています。if直後の「**[$(($i % 3)) -eq 0]**」は、変数iを3で割った剰余が0とイコール(-eq)だった場合に続くリストを処理します。

小技集

「-」で始まるファイルを作ってしまったので削除したい

うわぁぁぁー。何かの手違いで記号が混じっ

たファイルを作ってしまった～。「#」や「$」などが混じったファイルができてしまっていますね(**図26**)。ホント、タイプミスかなんかでしょうけど、どうやってこんなの作ったの？と過去の自分を呪い殺したくなることが2日に1回ぐらいあります。そういうときも慌てないで調べることが大切です。

「#」や「$」、空白などが混じったファイルは比較的容易に対応でき、「\(バックスラッシュ)」でエスケープするか「'(シングルクォート)」でクォートすると対応できます。厄介なのは「-」で始まるファイル名です。

```
$ rm -help
```

とすれば当然rmコマンドのヘルプが出力されます。このケースは「\」も「'」も効きません。この場合は「--(ハイフン2つ)」を使います。次のように実行すると「--」以降はオプションではないので「--help」は引数として扱われます。

```
$ rm -- --help
```

引数文字数制限を超える大量のファイルを操作したい(どちらかというとcpやmvの小技)

うっかり、あるディレクトリにファイルをポンポン作っていたら大量になってしまった。こうなるとlsしても表示に時間がかかってしまいます。さすがに扱いにくくなったので整理整頓としていったん別のディレクトリに移動しようと思い、「**mv * ../bak**」とやろうとすると次のようにエラーが出ました。

```
$ mv * ../bak↵
-bash: /bin/mv: 引数リストが長すぎます
```

あらら。ファイルを移動できません。1つ1つ移動しないといけないでしょうか。

これはシェルの機能ではないのですが、シェルを使っているうえでぶつかりやすい障壁なのでここで押さえておきます。

mvを使う際、引数に移動する対象ファイルを指定しますが、「*」を利用するとシェルは実行前に実ファイル名に展開します。たとえば「**mv * ../bak**」は「**mv gihyo-001.txt gihyo-002.txt**（略）**../bak**」と展開されます。この引数がARG_MAXを超えるとエラーが出ます。

Linuxは引数制限値ARG_MAXを持っています。手元のLinuxでは2,097,152文字でした。これはgetconfコマンドで参照できます。

```
$ getconf ARG_MAX↵
2097152
```

この場合は対象ファイルをlsからxargsコマンドに預けて、mvコマンドの移動先ターゲットディレクトリ指定オプション「-t」を使うとエラーが出ません。

```
$ ls | xargs mv -t ../bak
```

lsコマンドでは「*」などの引数を指定していないので先ほどの引数展開のエラーは出ません。xargsは指定されたコマンド(ここではmvコマンド)を、入力された引数を適切に分割して実行するので、分割払いのイメージで処理が進められます。

mvコマンドは「mv␣元ファイル␣移動先ファイル」の書式で、移動先ファイルだけを指定することはできません。xargsから

▼図26　記号が混じったファイル例

```
$ ls -l↵
合計 4
-rw-r--r-- 1 ryosuke ryosuke  0  4月  7 15:41 #comments
-rw-r--r-- 1 ryosuke ryosuke  0  4月  7 15:40 $$fuga
-rw-r--r-- 1 ryosuke ryosuke  0  4月  7 15:39 --help
-rw-r--r-- 1 ryosuke ryosuke  0  4月  7 15:40 >hoge
-rw-r--r-- 1 ryosuke ryosuke  0  4月  7 15:41 Program Files
-rw-r--r-- 1 ryosuke ryosuke 33  4月  7 02:16 var_test.sh
```

▼図27　bashのディレクトリスタックに追加・削除して移動・処理する

```
$ pushd /tmp/↵          ディレクトリスタックに現在のディレクトリを入れて指定した場所に移動
/tmp /home/ryosuke/     ディレクトリスタックが表示される
$ pwd↵                  /tmpにいることを確認（処理）
/tmp
$ popd↵                 ディレクトリスタック先頭を削除してそこに移動（戻る）
/home/ryosuke
```

のファイル指定を受けるために「-t」オプションを使って移動先ディレクトリを指定することで「mv␣-t␣移動先␣元ファイル」の書式を利用できるようになります。これはcpコマンドでも利用できます。

ディレクトリを行ったり来たりするのにPATHを指定するのがつらい

直前にいた作業ディレクトリに戻りたいときは「cd -」を実行すると、環境変数OLDPWDを参照して戻ることができます。移動した後にはOLDPWDには「cd -」を実行していた作業ディレクトリが入っているので再度「cd -」を実行することで2つのディレクトリを行ったり来たりできますね。

ワンライナーでディレクトリ移動してプロセス起動して戻ってくることもできます。**図27**は、bashのディレクトリスタックに追加・削除して移動・処理する方法です。これならば戻るディレクトリを指定する必要なく、移動先で処理して戻ってこられますね。

pushd、popdはbash独自の内部コマンドなので、POSIX準拠を目指す場合は**図28**のサブシェルを利用する方法もあります。「(」「)」でコマンドを囲むとサブシェルという子プロセスを起動して処理されます。

プログラム実行後にプロンプトが帰ってこない

重い処理をさせるとプロンプトが戻るまでに時間がかかることがあります。実行中のプロセスを止めるにはCtrl-cを押すと、プロセスを終了（停止ではなくて）することができます。しかし、処理中の場合もありますので、Ctrl-zで停止して操作対象ファイルに変化があるか、ログに実行記録があるかなど、様子を見るのがいいでしょう。

▼図28　サブシェルを利用した図27と同等の処理

```
$ (cd /tmp; pwd ); pwd↵
/tmp          /tmpに移動した後の1回目のpwdの実行結果
/home/ryosuke （cd /tmp ）後のpwdの実行結果
```

ログインしていたリモートセッションが切れたがプロンプトが帰ってこない

リモートホストにSSHなどでログインして、しばらく放置しておくと、SSHサーバ側でタイムアウトして切れてしまったにもかかわらずプロンプトが帰ってこないことがあります。この場合、入力した文字が表示されないし、Ctrl-cでの終了も効かないことが多々あります。シェルの機能ではありませんが「~.」を入力することでプロンプトを「こちら」に持ってくることができるようになります。

突然入力を受け付けなくなった

なにかのタイミングでCtrl-sを押してしまい、ターミナルをロックした可能性があります。Ctrl-qでロックを解除しましょう。ターミナルをCtrl-sでロックしないよう設定する方法は、前述の「コマンド履歴利用で、もう長いコマンドなんて打たないよ絶対」で明かしたので試してみるのもいいでしょう。

bashで書いたスクリプトがPOSIX sh準拠か調べたい

bashに慣れてしまった体でシェルスクリプトを書いていると、bash専用の文法や機能を使ってしまうことがあります。うっかりそのスクリプトがほかのUNIX系OSで稼働することになっ

▼図29　POSIX準拠のスクリプトかを調べる

```
$ cat pushd-popd.sh
#!/bin/sh
pushd /tmp/
popd
$ checkbashisms pushd-popd.sh
possible bashism in pushd-popd.sh line 2 ((push|pop)d):
pushd /tmp/
possible bashism in pushd-popd.sh line 3 ((push|pop)d):
popd
```

▼表7　コマンドライン編集のショートカット

コマンド	操作
Alt-u	単語のカーソルより右を大文字にする
Alt-l	単語のカーソルより右を小文字にする
Ctrl-t	カーソルのある文字と左の文字を入れ替える
Alt-c	カーソルのある単語の先頭文字だけ大文字にする

た際、スクリプトの先頭に「**#!/bin/sh**」とあれば「ああ、動くかもな」と思ってしまいます。

書いたスクリプトがbashの機能を使わずにPOSIX準拠か調査してみましょう。調査方法はcheckbashismsを使います。Debian系Linuxではdevscriptsというパッケージに含まれます。

例として、ディレクトリスタックに入れる・出すを行うpushd・popdを使ったスクリプト「pushd-popd.sh」を用意します。pushd/popdはbashの機能なので、**図29**のとおりcheckbashismsにかけると、「((push|pop)d) はbash専用」との警告が出力されます。この警告がなくなるように編集するか、もうbashでしか動かしちゃダメ、と「**#!/bin/bash**」にスクリプトの先頭を変更するかは自由です。

ちょっと知ってると便利 コマンドライン編集ショートカットキー

カーソル移動以外の、ちょっと知っていると便利な編集ショートカットキーを**表7**にリストアップしておきます。

bashはreadlineライブラリを利用するので、気軽にあなただけのショートカットキーを作成できます。システム全体で有効にするショートカットは/etc/inputrcに記述、個人のショートカットキーはホームディレクトリに.inputrcを用意します。

ショートカットキーの記述形式は「キーシーケンス：機能名もしくはマクロ」です。キーシーケンスは「control-」や「\C」で Ctrl キー、「Meta-」や「\M」で Alt キーを指定することができます。マクロはreadlineで用意している機能を指定するか、入力する文字列を指定します。

たとえば個人用に、Ctrl-oをタイプしたら「**> /dev/null**」が表示されるショートカットを作ってみましょう。.inputrcファイルを用意したら新しいシェルを開くか、ログインしなおします。「bind -s」で確認することができます。

```
$ cat ~/.inputrc          個人用ファイル用意
control-o: " > /dev/null"
$ bind -s                 新しいシェルで確認
"\C-o": " > /dev/null"
$ ps ax   ここでCtrl-oを入力
$ ps ax > /dev/null   期待どおりの表示がされる
```

bashやreadlineのマニュアルを参照して、さらに凝ったショートカットを作ってみてはいかがでしょうか。

第1章、2章のまとめ

bashは現役でLinuxのフロントエンドです。Mac OS Xもターミナルを開けばbashが起動します。UNIX系OSを使っていれば、なんだかんだで使わないといけない道具なので、少しでも便利に使えるようになれば幸いです。SD

シーンに応じたシェルスクリプトの自在な書き方・使い方

Author 上田 隆一 （うえだ りゅういち） 千葉工業大学／USP友の会

ファイルに複数のコマンドを書き並べて保存しておくと、シェルスクリプトとして一度の指示でまとめて実行できます。同じ処理を繰り返すとき、自動化するときに便利です。多くの場面で使われるシェルスクリプトですが、自由度が高く、汎用性や即興性、実行時間などのトレードオフで書き方も変わります。実際のスクリプトを読んでその違いを考えましょう。

はじめに

ご無沙汰しております。シェル芸おじさんこと上田です。本章はシェルスクリプトの書き方について扱います。このテーマ、ベテランの人に話をさせると、それぞれ全然違うことを言います。何で違うことを言うのかというと、シェルスクリプトは用途が広すぎて、書き方が用途に応じて変わるからだと筆者は考えています。入門者には文法も大切ですが、最初に偏りすぎるのも防ぐ必要があります。そこで、本章では文法を紹介しつつも、用途に応じたシェルスクリプトの書き方について一緒に考えていきたいと思います。今回の特集はbashですが、shの話も出てきますので混乱しないように適宜説明します。

本章の内容は、初心者と言っても、ある程度UNIX系OSの知識を持つ人が対象となります。本章がギブアップだという方は、次章のめくるめくシェル芸の世界に飛んでいただければと。

シェルスクリプトって何?

まず、シェルスクリプトとは、シェル上でコマンドを呼び出していく手順を、まとめてファイルに書いて実行できるようにしたものです。次の例は、ファイル／ディレクトリの一覧からディレクトリを除いて、ファイルの容量だけを出力して、小さい順に並べるという処理です。

```
$ ls -la | grep ^- | awk '{print $5,$NF}' | sort -n
```

もっと簡単な方法があるかもしれませんが、とりあえず筆者はこれくらいしか思いつきませんでした。ちなみに環境はUbuntu 14.04 LTSのbash上です。

これが1回で済むならシェルで上記のように打ち込んで終わりですが、何かのディレクトリの監視をしていて頻繁に打たなければならない状況を考えてみましょう。この場合、**図1**のようにfilesizerankというファイルに今のコマンドを保存しておき、**図1**の②のようにbashというコマンドに読み込ませてやると、同じ処理が何度もできるようになります。このようなファ

▼図1 シェルスクリプトを作って実行する

```
↓①エディタなどで次のようなファイルを作っておく
$ cat filesizerank
ls -la | grep ^- | awk '{print $5,$NF}' | sort -n
↓②実行する
$ bash filesizerank
18 .gnuplot_history
26 a.bash
49 filesizerank
（..中略..）
50269 .bash_history
310911150 TESTDATA.gz
1157730515 imgs.tar.gz
```

イルは、「シェルで実行するスクリプト」であり、これを我々は「シェルスクリプト」と言っています。シェルはash、bash、csh、tcsh、dash、zsh、……といろいろありますが、これらを使って実行するスクリプトはすべてシェルスクリプトと呼んで良いでしょう。ただし、場合によってはbashのスクリプトを「bashスクリプト」と明示的に言う場合があります。

シェルスクリプトでのコマンドの呼び出し方はパイプでつなぐ方法だけではありません。たとえば**リスト1**は、あまり使い道はないかもしれませんが、2つのファイルの名前を交換するためのシェルスクリプトを雑に書いたものです。ファイルを移動するためのmvというコマンドを3回呼び出しています。**図2**はこのchangeというシェルスクリプトを使っている様子を示しています。②でchangeと書いたあとにa、bとファイル名を指定していますが、これが**リスト1**のシェルスクリプト内の**$1**、**$2**に置き換わります。

この2例でわかるように、シェルスクリプトとは「コマンドを使う手順をそのままファイルに書いたもの」と言えます。コンピュータに作業をさせる手順を書いたものなので、プログラムの一種ということになります。ただ、プログラムと言っても呼び出すのは関数ではなくコマンド(プロセスが違う別のプログラム)なので、普通のプログラミング言語とはしくみも役割も書き方も違います。「シェルの代わりに○○を使いたい」という発言をTwitterでよく見ますが、それでは済まない場合はまだ多いと言えます。

さまざまな使い方をされるシェルスクリプト

代わりのないものなのでシェルスクリプトはさまざまな用途で使われます。筆者の主観ですが、用途は次の3つに分かれるようです。

- システム用(OS起動時のサービスの立ち上げ、インストーラなど)
- ユーザ用1(よく行う作業の自動化)
- ユーザ用2(その場限りの作業の即興プログラミング)

本章では最初の2つについて、実例を読みながら解説します。3つめはワンライナーか、スクリプトにしてもすぐ捨てるようなもので、第4章で扱うような類のものです。

冒頭にも述べましたように、上記3用途のシェルスクリプトはいずれも「シェルスクリプト」なのですが、書き方や気をつけなければならないことが大きく違います。この違いは「汎用性」と「時間(スクリプトを書く時間と計算時間)」のどちらを重視するかで変わってきます。

▼リスト1　ファイルの名前を交換するシェルスクリプト(changeという名前のファイルに保存)

```
mv "$1" "$1.backup"
mv "$2" "$1"
mv "$1.backup" "$2"
```

▼図2　シェルスクリプトchangeを使う

```
↓①次の2つのファイルa、bを交換してみる
$ cat a
これはファイルa
$ cat b
これはファイルb
↓②実行
$ bash change a b
↓③名前が入れ替わっている
$ cat a
これはファイルb
$ cat b
これはファイルa
```

システム用スクリプト(rcスクリプト)を読む

まず、システム用のものの例として、OSが立ち上がるときに動作するシェルスクリプトを見てみましょう。OSの起動時には、Webサーバのようにコンピュータに常駐するプログラムがいくつも立ち上がりますが、UNIX系OSでは伝統的にこの処理にシェルスクリプトが使われており、今も使われています。

筆者が使っている某所のサーバ(Ubuntu

Server 14.04.1 LTS)を例にとると、/etc/init.dの下に**図3**のようなファイルがあります。これらはrcスクリプトと呼ばれるシェルスクリプトで、コンピュータが起動するときにあるルールに従って実行されます。

この中でも短い部類に入る/etc/init.d/sudoファイルを、**リスト2**にすべて示します(rcスクリプト特有のヘッダは除きました)。これを読むのは筆者でもたいへんで、初心者の方は必ずしもすべて理解することはありません。しかし、最初のうちに知っておくと良いことが散りばめられていますので、ちょっと読んでみましょう。

まず、1行目ですが、**#! /bin/sh**とあります。**#!**やこの行自体は「シバン(shbang)」と呼ばれます。**図1**では、**bash filesizerank**というようにbashにシェルにスクリプトfilesizerankを渡して実行していたのですが、シバンの行があると、**図4**のようにfilesizerank自体をプログラム扱いして実行できるようになります[注1]。

シバンを見るとわかりますが、/etc/下のシェルスクリプトはshのものが多いです。この手のスクリプトはさまざまな環境で動作する必要があり、bashがインストールされていない環境で使われる場合も考えるべきだからです。本特集で扱っているのはbashなので補足しておくと、shの文法はbashでも使えます。逆は成り立たないことがあるので、注意が必要です。

次に**リスト2**の3行目ですが、**. <ファイル(スクリプト)名>**という文になっています。これは、スクリプトを読み込んで実行するという意味です。端末から、

```
$ . filesizerank↵
```

▼図3　/etc/init.d下のシェルスクリプト

```
$ ls -1 /etc/init.d | head↵
README
apache2
apparmor
automaked
console-setup
(..略..)
```

▼図4　filesizerankにシバンを付けて実行

```
↓このようにシバンで/bin/bashを使うと明示
$ cat filesizerank↵
#!/bin/bash
ls -la | grep ^- | awk '{print $5,$NF}' | sort -n
↓ファイル自体を実行できるようにパーミッションを変える
$ chmod +x ./filesizerank↵
↓実行
$ ./filesizerank↵
(..図1と同じ結果..)
```

▼リスト2　/etc/init.d/sudo

```
01 #! /bin/sh
02
03 . /lib/lsb/init-functions
04
05 N=/etc/init.d/sudo
06
07 set -e
08
09 case "$1" in
10   start)
11         # make sure privileges don't persist across reboots
12         if [ -d /var/lib/sudo ]
13         then
14                 find /var/lib/sudo -exec touch -d @0 '{}' \;
15         fi
16         ;;
17   stop|reload|restart|force-reload|status)
18         ;;
19   *)
20         echo "Usage: $N {start|stop|restart|force-reload|status}" >&2
21         exit 1
22         ;;
23 esac
24
25 exit 0
```

※左の2桁の数字は、説明用の行番号(リスト3、4も同様)

注1) 実はシバンがなくても使っているシェルがbashならbashで動作します。ただ、別の環境だと別のシェルで動きますし、パッと見て何のスクリプトかわからなくなるのであまりシバンは省略されません。

とやってみるとfilesizerankが実行されるので試してみましょう。ところで/lib/lsb/init-functionsでは何を実行しているのか気になるところですが、これは何も実行していません。このスクリプトには関数がいくつか定義されていて、それを実行できるように読み込んでいるのがリスト2の3行目です。

リスト2の5行目はNという変数に文字列/etc/init.d/sudoを代入しています。シェルスクリプトの変数はすべて文字列です。変数Nは、$Nあるいは${N}というように、頭に$を付けると中の文字列を参照できます。20行目で使われていますね。$Nと${N}では、後者のほうが丁寧な書き方で、変数名に数字や記号が入っていると、このように書かなければならない場合もあります。

7行目のset -eはshに-eオプションをセットするという意味です。-eは、コマンドがエラーを返したらその場で止めるためのオプションです。Pythonなど、普通のスクリプト言語の場合、何か例外があるとその場で止まりますが、シェルスクリプトは基本的に止まりません。何が起きてもだらだらと実行されてしまうので、それを抑制しています。

コマンドのエラーは、コマンドが終了したときに残す「終了ステータス」で確認できます。図5に例を示します。このようにコマンドを実行した直後に$?を見ると、終了ステータスの値を確認できます。基本的に、何か異常がある場合は終了ステータスには0でない数が入ります。

ただし、-eを指定しても、パイプの途中のコマンドのエラーは捕捉できず、エラーが起きてもスクリプトは止まりません。bashの場合は、set -eに加えてset -o pipefailと書くと、その問題を解決できます。図6のようにbashで打つと挙動がわかります。pipefailを指定したあとに、パイプにfalse(常に非0の終了ステータスを返すコマンド)を混ぜて実行すると、bash自体が終了します。

シェルスクリプトで終了ステータスを指定したい場合は、リスト2の21行目のようにexit <終了ステータス>と書きます。この行に制御が来ると、シェルスクリプトは終わります。

さて、リスト2の9行目に行きましょう。制御構文のcase文が出てきました。case文は、

```
case <文字列> in
  パターンA) 処理 ;;
  パターンB) 処理 ;;
  (..中略..)
  *) デフォルト処理 ;;
esac
```

という構造になります。ほかの言語と比べておもしろいのは、シェルのcase文は文字列のパターンマッチングで条件分岐していることです。

リスト2のcase文では、文字列の$1($1の意味についてはリスト1を参照)について、10、17、19行目で調査をしています。10行目では$1がstartなら12〜15行目の処理をして、17行目ではstopあるいはreloadあるいは……というように|で区切られた単語のいずれかなら何もしないというコードになっています。19行目の*は10、17行目の条件をすべてすり抜けた場合すべてに当てはまり、この場合は20行目でこのスクリプトの使い方を出力して21

▼図5 lsコマンドの終了ステータスを観察

```
↓存在するファイルを指定してlsを実行
$ ls memo
memo
↓すぐに?という変数の値を確認(0は正常終了)
$ echo $?
0
↓存在しないファイルをlsしてエラーを出す
$ ls momo
ls: momo にアクセスできません: そのような
ファイルやディレクトリはありません
↓終了ステータスが0でない値となる
$ echo $?
2
```

▼図6 パイプの途中のエラーで処理を止める

```
$ set -e
$ false | false | true        ←Enterキーを押下し
                                てでも何も起こらない
$ set -o pipefail
$ false | false | true        ←Enterキーを押下し
                                た瞬間、シェルが終了
```

行目で終了しています。

端末でコマンドを使っていると、たまにyes/noや[Y/n]のように聞かれることがありますが、このような入力をシェルスクリプトでさばく場合にはcase文が便利です。パターンには、[Yy][Ee][Ss]のように、大文字小文字の区別をなくすような書き方も用意されています。

リスト2にはもう1つ制御構文があります。12～15行目のif文です。if文の構造は

```
if コマンド ; then
  処理
elif
  処理
else
  処理
fi
```

というものです。

if文は文字列のパターンでなく、実行したコマンドの終了ステータスで条件分岐をします。これも筆者は過去よく書いてきたことですが、**リスト2**の12行目で使われている[は実はコマンドで、これが引数である-dと/var/lib/sudoと]を読み込んで、終了ステータスを返します。-dというのは次の引数に指定した文字列がディレクトリを指しているかを判断するという意味です。このようなオプションはほかにもあり、man [で確認できます。]は単なる飾りです。

最後に**リスト2**の14行目について。findというのは、基本、指定したディレクトリ以下にあるファイルやディレクトリをひたすら列挙して出力するコマンドです。また、-exec <コマンド>という引数を渡すと、列挙したファイルに、指定したコマンドを実行させることができます。ですので、、最初の**find /var/lib/sudo -exec** で、/var/lib/sudoディレクトリのファイルに何かを実行すると読めます。次の**touch -d a0**が、その実行する何かです。これは筆者も何だろうと思ったのですが、「ファイルの更新時刻をUNIX時刻の0秒(1970年1月1日0時0分0秒)にする」という意味になります。次のように試してみるといいでしょう。

```
$ touch -d a0 a↵
$ ls -l a↵
-rw-r--r-- 1 ueda ueda 23  1月  1  1970 a
```

その後の**'{}' \;**は、findに渡す引数です。本当にfindに渡したいのは**{}**と**;**なのですが、**{};**はシェルが見つけると記号で別のものに置き換えてしまうので、**{}**はシングルクォート、**;**はバックスラッシュでエスケープして、置き換わりを防いでいます。**{}**はfindが見つけたファイル名に置き換わります。**;**はコマンドの終わりをfindに教える引数です。ほかには**+**という終わり方もありますが、**man find**に説明を譲ります。

最後に挙動を調べておきましょう。結局このシェルスクリプトは、sudoというコマンド(rootになるためのコマンド)が時刻を管理するために使うファイルの時刻を初期化するためのものだったようです(**図7**)。

さて、rcスクリプトの例を見てきましたが、このようなシステム用シェルスクリプトを書くときはかなり多くの制約を気にしなければなりません。シェルはshが基本ですが、shにはpipefailがないので、あまり調子に乗ってパイプを使うこともはばかられます。

さらに面倒なことに、「sh」が指すものが環境によってdashだったりbashだったりashだったりして挙動が変わってしまうので、場合によってはこの点も考慮して共通の書き方をしなければなりません。これについては第5章で今泉さ

▼図7　/etc/init.d/sudoを実行してみる

```
↓まずsudoを使ってrootになる
$ sudo -s↵
↓時刻が1970年の正月でないものを探す
# ls -l /var/lib/sudo/ueda/0↵
-rw------- 1 root ueda 40  4月  2 06:52 /var/lib/sudo/ueda/0
↓実行
# /etc/init.d/sudo start↵
# ls -l /var/lib/sudo/ueda/0↵
-rw------- 1 root ueda 40  1月  1  1970 /var/lib/sudo/ueda/0
```

▼図8 シェルスクリプトの出力をチャットで受ける

んが扱います。

ユーザ用のシェルスクリプト

次に、ユーザ用のシェルスクリプトの例を見てみましょう。個人で何かを自動化したい場合、シェルスクリプトを書けるとログイン時や決まった時刻にバックアップを取ったり、電子メールを飛ばしたり、Webサービスに投げたりということがてっとり早くできてしまいます(もちろん、それなりに習熟しないと時間はかかりますが)。この使い方は基本、自分用なので好きなコマンドをインストールして呼び出しても良いということになるので、制約はシステム用のものより緩くなります。コメントはないに越したことはありませんが、スクリプトの長さや個人の力量、他人に使わせるかどうかで分量や内容は変わるでしょう。

Slackに結果を投げる監視スクリプト

例として読むのは、いくつかのWebサイトがちゃんと稼動しているかを調べて結果をSlack(最近流行っているチャットサービス)に投げるシェルスクリプトです。Slackの設定方法などはWeb[注2]などに詳しいので割愛しますが、シェルスクリプト側は文字列を作ってインターネット上に放り投げるだけです。

コードより先に、このシェルスクリプトの出力をSlackで見たものを**図8**に示します。4つのサイトを監視し、正常な反応があったらOKのアイコンを出し、何か異常があったらHTTPステータスコード(404 Not Foundなどのあの番号のことですね)を出すようにしてあります。この例ではUSP友の会のサイト(https://www.usptomo.com)のURLを意図的に間違えて接続できなくして、(本来はステータスコードではないですが)未接続を表す「000」を表示しています。

▼リスト3 シェルスクリプトWEB_CHECK

```
01 #!/bin/bash -xv
02
03 tmp=/tmp/$$
04 slackurl=https://hooks.slack.com/services/xxxxx
05 curlstr='<%{url_effective}> %{http_code}@'
06
07 cat << FIN > $tmp-url-list
08 https://blog.ueda.asia/
09 http://at-home.cit-brains.net/
10 https://www.usptomo.co/
11 https://lab.ueda.asia/
12 FIN
13
14 cat $tmp-url-list                                  |
15 xargs -n 1 curl -Is -o /dev/null -w "$curlstr"     |
16 sed -e 's/200@/:ok:@/g' -e 's/@/\\r\\n/g'          |
17 (
18         echo 'payload={"text":"'
19         cat
20         echo '"}'
21 )                                                  |
22 curl -X POST -d @- $slackurl
23
24 rm -f $tmp-*
```

さてコードを見ていきましょう(**リスト3**)。このコードはRaspbianとOS Xのbashで動作確認をしています。Ubuntuでも動きます。まず、3～5行目で変数をいくつか定義しています。3行目の**$$**は、このシェルスクリプトが動作しているプロセス番号が入っている特殊変数です。7、14行目に**$tmp-url-list**という記述がありますが、これが、たとえばプロセス番号が100番なら**/tmp/100-url-list**という文字列に置き換わります。4行目のslackurlはデータを投げる窓口で、ここではSlackで指定されたURLを指定します。このURLはお見せできないのでダミーのxxxxxを入れています。5行目のcurlstrはcurlコマンドの出力フォーマットです。

curlstrを使っているのは15行目です。**curl -Is**……の部分を端末で試してみると**図9**のよ

注2) 「[10分で出来る]シェルスクリプトの結果をslackに投稿」 http://qiita.com/tt2004d/items/50d79d1569c0ace118d6

うになります。オプションについてはman curlで調べていただきたいのですが、出力を見ると<>で囲まれたURLとステータスコードが得られていることがわかります。<>はSlackでURLにリンクを張るための方言です(**図8**のURLはリンクになっています)。

7行目のcat << FIN > $tmp-url-listは、「次にFINという文字列が出てくるまで、書いた中身をファイル$tmp-url-listに書き出すぞ」という意味です。8〜11行目に書いたURLのリストがファイル$tmp-url-listに保存されます(第2章のヒアドキュメントの説明を参照)。

ところで、このコードを見て、実行するごとにファイルにURLを書き出しているのを無駄だと思う人がいるかもしれません。スクリプトの外にファイルを準備したり、変数にURLを入れたりするほうが普通ではないかと。動けばそれでも全然問題はないのですが、筆者がこうするのは次の理由からです。

・外にファイルを準備すると管理するファイルが増える
・データはなるべく標準入出力で扱いたい

ここではとくに後者について話をしておきます。コマンドの多くは標準入力からデータを受けて標準出力からデータを出すように作られており、しかも多くが行単位でデータを処理することを前提にしています。このことを考慮すると、コマンドに何かを入力する標準的なフォーマットは、コマンドの標準入力に接続しやすく行単位でデータが書かれたファイルということになります。筆者の場合、変数にはファイル名やコマンドのオプションなどの定数を記録しておくに止めるようにしています。bashは配列が使えますが、基本、筆者は使いません。

ただし、この議論は先ほどのシステム用シェルスリプトでは事情が変わります。システム用だと、スクリプトがエラーを起こして/tmpに中間ファイルが残るのも、パイプ中のエラーが捕捉できないのも、嫌なことなので、変数にデータを記録しながら1つずつコマンド(サブルーチン)を呼ぶような書き方をしたほうが良いという判断になることがあります。

ミスなくコマンドを呼び出すための手順書として見るか、データフロー言語として見るかで、シェルスリプトはまったく違った顔を見せます。

また、この例には当てはまりませんが、シェルスクリプトで扱うデータの量が大きい場合、いちいち変数にデータを格納したり、変数に格納したデータごとにコマンドを呼び出したりすると、極端に処理が遅くなります。逆にパイプと標準入出力を使うスタイルの場合、うまくやればコマンドが並行に動いて計算が早く終わります。

さて**リスト3**の続きを。14行目以降のパイプラインでは、14行目でURLリストをパイプに流し、15行目でxargsというコマンドを使ってURLのリストの1行1行をcurlコマンドの引数にしてcurlを実行しています。16行目ではsedを使い、ステータスが200番の場合に:ok:(Slackで絵文字になる)に置き換え、@を改行コードに変換しています。これはSlackのためにやっている処理です。

17〜21行目の()は、複数のコマンドをひとまとめにしてパイプにつなげるためのものです。ここではパイプから流れてきたデータをcatして、その前後にpayload={"text":"と"}をくっつけています。これでcurlの出力がJSON形式のデータに変換されます。できたJSON形式のデータは、22行目のcurlでSlackに向けて投げられ(POSTされ)ます。このJSONの作り方は無理やり感があるので、もう少し凝ったJSON

▼図9　Webサイトに接続してURLとステータスコードを出力する

```
$ curl -Is -o /dev/null -w '<%{url_effective}> %{http_code}@' https://blog.ueda.asia↵
<https://blog.ueda.asia/> 200@
```

▼リスト4　Glueスクリプトの例（手軽に使えて勝手に消える中間ファイル）

```
01 #!/usr/local/bin/glue
02 import PATH
03
04 # seq 1 100 | head -n 3 の結果を
05 # 中間ファイル「f」に入力
06 file f = seq 1 100 >>= head -n 3
07 # ファイルをcat
08 cat f
09 #fは勝手に消えます
```

のデータを作るときはjqコマンドを使うか、別の言語を使うという選択肢も考えないといけません。ただ、筆者はちまちまプログラミングするより、パイプラインで一気に加工したいので、これくらいならシェルスリプトで済ませてしまいます。

こうしてできたシェルスリプトは手で叩くと実行できますが、自動化して定期実行させるほうが良いでしょう。cronというしくみを使うとこれが可能となりますが、説明は拙著『シェルプログラミング実用テクニック』に譲ります。宣伝でした。

まとめ

シェルは、もともとコマンド（つまり別のプロセス）を呼び出すためのもので、言語として見たとき、プロセスの中の処理を記述するほかの言語とはまったく用途が異なるものです。

また、シェルスクリプトの用途を説明するときに、筆者はユーザが何かをする前の準備のためのシステム用と、ユーザが便利にコマンドを使うためのユーザ用とに分けて説明しました。偏った意見に毒されないために、「シェルスクリプト」にもいろいろあるんだと知っておくのも必要かと考えて書いたしだいです。

最後にちょっとだけ私的な紹介をします。結局、システム用のシェルスクリプトでもパイプの途中のエラーで止めたり、下手に中間ファイルを残さないようにしてやれば楽に書けるのではないかと考えた筆者は、昨年あたりにシェルを書いていました（シェルスクリプトではなく、GlueLangというシェルを）。コードは「https://github.com/ryuichiueda/GlueLang」にあり、個人的事情で現在、絶賛開発ストップ中です。

▼図10　リスト4のスクリプトのseqをsecにわざと間違えて実行

```
$ ./hoge.glue↵

Parse error at line 4, char 10
  line4: file f = sec 1 100 >>= head -n 3
                  ^

  Command sec not exist

  process_level 0
  exit_status 1
  pid 38049
```

リスト4に、現時点で動作するGlueLangのスクリプト（勝手にGlueスクリプトと呼んでいる）を示します。このように中間ファイルを気軽に作れて勝手に消してくれたり、**図10**のようにエラーへの対応を親切にしたり、がんばっていました。いや、がんばっています。ただ、シェル自体を書いてみてわかったことですが、

- 終了ステータスで条件分岐させるというシェルのしくみを踏襲しながら、if文やwhile文の文法をきれいに保つ
- シェル本体を小さく保つことと多機能のバランスを取る
- マルチプロセスで動かしつつ変数のスコープをわかりやすく設計する
- 普及しきったshのあとに普及させる

等々のことがたいへんです。シェルは古いですが、やはり今まで代替のものが出てこないのもわかります。シェルスクリプトは不格好かもしれませんが、少なくともプロセス内で処理を済ますことを最重視している一般的な言語とは用途も書き方も違うものです。シェルスクリプトが何なのか理解して、書けるようにはしておいていただければと。

あ、GlueLang、手伝ってくださる方募集中です。SD

シェル芸問題で腕を磨け！ テキスト処理・計算・調査の定石

Author 上田 隆一（うえだ りゅういち） 千葉工業大学／USP友の会

UNIX系OSで、CLI（Command Line Interface）を使っているなら、ちょっとしたテキストデータの加工、数値データの計算、調べものもCLIで済ませたくなります。「シェル芸」と呼ばれるコマンドのワンライナーのテクニックには、コマンドで自在にデータを操作するための定石がいくつか存在します。その定石を知るためにシェル芸の問題に挑戦してみましょう。

はじめに

第4章は、引き続きシェル芸の人がお送りします。というか、「第4章はシェル芸をお願いします」と正式に依頼されてしまったのですが、関係者各位、大丈夫でしょうか。

「シェル芸」というのは筆者が勝手に作ってしまった言葉です。㈱ハートビーツさんが主催するインフラエンジニア勉強会hbstudyで2012年10月に「GUIに慣れてしまった現代人に足りないのはシェル芸だ！」とスライドに書いたのが最初です。それ以来、2ヵ月に1度、「シェル芸勉強会」と銘打って定期的に勉強会を開いてきました。hbstudyから数えて2016年4月30日で22回になりました。

この章では、いつもの雰囲気そのままに、誌上でシェル芸勉強会をやってみます。

シェル芸って何?

まず、「シェル芸って何だ」ということですが、定義は「マウスも使わず、ソースコードも残さず、GUIツールを立ち上げる間もなく、あらゆる調査・計算・テキスト処理をCLI端末へのコマンド入力一撃で終わらすこと。あるいはそのときのコマンド入力のこと。」[注1]です。CLIというのはCommand Line Interfaceの略で、コマンドを打ち込む字だけの「黒い画面」と一部で恐れられているあれのことです。つまり、シェル芸というのは、黒い画面にコマンドを打ち込むだけで、手持ちのデータから何か自分のほしい情報やデータを作ることを指します[注2]。

たとえば、第3章の

```
$ ls -la | grep ^- | awk '{print $5,$NF}' | sort -n
```

もシェル芸です。「ファイル／ディレクトリの一覧からディレクトリを除いて、ファイルの容量だけを出力して、小さい順に並べたい（そして、それを見て大きなファイルがないか調べたい）」という「ニッチだけど日常では無数に存在する細かい需要」を、ls、grep、awk、sortという、たいていの環境に存在するコマンドを組み合わせて満たしています。この例は些細だと思われるかもしれませんが、これがパッとできないと調査に時間がかかったり、諦めてしまったりと、いろいろストレスのかかる状況に陥ります。こういうとき、最近はワンライナーを教えてくれる優しくこわい先輩が少なくなってしまったの

注1） https://blog.ueda.asia/?page_id=1434

注2） ベテランの人には、「それはコマンドのワンライナーだろ」という指摘をされますが、そのとおりです。ただ、「コマンドのワンライナー」だとあんまりキャッチーではないので、シェル芸と名付けたしだいです。また、「なんで、もっとかっこいい名前にしないんだ」という意見もありますが、IoTやFinTechなど、意識高い名前に釣られる面倒臭い集団が寄って来ないので結果オーライです。

で、「シェル芸」と騒ぐことでちょっとでも状況が好転しないかと活動しています。

今回の内容

ということで、本稿では普段のコマンドをより自在に使えるように、シェル芸の問題を解いてみます。シェル芸勉強会では実用的な問題を解く回とパズルを解く回をだいたい交互にやっていますが、今回はパズルをやってみます。ログ解析など、実用的な内容も検討しましたが、そのような特集は本誌ではたびたび取り扱われているので、ちょっと変わったことをということでパズルにしました。パズルを解くのは、ただ単に(一部の人に)楽しいからというだけではありません。メタな問題を解いておくと実用の際に機転が利くという効用もあります。

問題を解くにあたって約束事を。まず使うシェル、端末、コマンドなどですが、これは自分が普段使っているものをお勧めします。普段使っている端末を便利にという意図なので、とくに指定はしません。自身の環境で解答例が動かないことがありますが、それも自身の環境を知る良い機会ととらえて別解を考えてみましょう。本章では**図1**のようにUbuntu上のbashを使います。おっと思わずシェル芸が出てしまいました(早く本題にいきたいので、**図1**のワンライナーの解説はしません……)。もしUNIX系OSや端末は普段使わないという場合は、この環境を仮想マシンなどで作って試してみましょう。

また、シェル芸の場合は問題が解けてしまえば解が一般的でなくても良いという約束があります。あくまで「あるときに発生した仕事をいかに早く片付けるか」が目的です。シェル芸で出てきた答えは基本的に拙速なものなので間違えることがあります。ですので、重要な仕事の場合はしっかり検算を行う必要があります。

問題に使うデータは、「https://github.com/ryuichiueda/ShellGeiData/tree/master/sd201606」に置きました。

▼図1　本章で使用する環境

```
$ cat /etc/lsb-release | ➘
tr '=' ' ' | xargs -n 1 | tail -n 1↵
Ubuntu 14.04.1 LTS
$ env | grep SHELL↵
SHELL=/bin/bash
$ echo $LANG↵
ja_JP.UTF-8
```

第1問　「ファイル内の数字を合計する」

さて、第1問です。次のようなファイルがあります。まず、ファイルにカンマ区切りで書いてある数字を足してみましょう。先を読む前にちょっと考えてみてください。すぐわかる人はいくつも解法を考えてみましょう。

```
$ cat data↵
1,2,34,4
43,43,5,751,16,21,2,3,4
43,1453,9,117,6,2
```

筆者がまず思いついたのは**図2**です。trは基本的なコマンドで、標準入力からの文字列から特定の字を消したり、別の字に入れ替えたりするために使われます。numsumはインストールしないと使えないコマンドです。「numをsum」ということで、縦に1列に並んだ数字を足しあげます。numsumをUbuntuでは次のようにインストールします。

```
$ sudo apt-get install num-utils↵
```

Macでも、homebrewで`brew install numutils`と実行するとインストールできました。

`cat data | tr ',' '\n'`だけを実行してその出力を見てから、↑を押して[注3] | `numsum`と

▼図2　第1問の解答例

```
$ cat data | tr ',' '\n' | numsum↵
2559
```

注3) 一度打ち込んで実行したコマンドやワンライナーは、↑キーを押すことで再度表示させて実行することができます。また、コマンドやファイル名を打っている途中でTabキーを押すと、そのあとの文字列の補完ができます。かなりあとに知ったという人が多いので、念のため。

追記して実行してみてください。ワンライナーのそれぞれの記述がどんな処理を担っているのかがよくわかります。また、numsumはオプションを使うと横並びなどでも足せますので、man numsumで確認をお願いします。

別解もいくつか思いついたので、**図3**に挙げておきます。

図3の別解3で使っているxargsは第3章でも使いましたが、もともとは「標準入力からやってきたデータを別のコマンドの引数として渡して、その別のコマンドをデータが来るごとに実行する」というように使うものです。ただ、コマンドを指定しないと内部でechoを呼ぶ仕様で、結果として、**図4**のようにデータを横に並べたり縦に並べたりすることができます。ただし、いちいちechoすることになるので、AWKで折り返すよりは速くはありませんし、**図4**の最後の例のように-nや-eをechoへのオプションとして扱うなど、制限があります。

▼図3　第1問の別解

```
別解1
$ tr ',' '\n' < data | numsum↵
2559
別解2
$ cat data | tr ',' '\n' | ↵
awk '{a+=$1}END{print a}'↵
2559
別解3
$ xargs < data | tr ', ' '+' | bc↵
2559
別解4
$ tr ',' '+' < data | bc | numsum↵
2559
```

▼図4　xargsでデータを並べる(-n＜数字＞で一度にechoする数を調整)

```
$ echo 1 2 3 4 | xargs -n 1↵
1
2
3
4
$ echo 1 2 3 4 | xargs -n 2↵
1 2
3 4
$ echo 1 2 3 4 | xargs↵
1 2 3 4
↓ただしechoのオプションとデータが一致すると使えない
$ echo -n -e 3 4 | xargs↵
3 4
```

図3の別解4では、**図2**のように、まずtrで,を+に変換しています。<ですが、これはファイル(data)の中身をtrの標準入力に流すという意味です。この処理で次のように式が3つできます。

```
trで数式を作る
$ tr , + < data↵
1+2+34+4
43+43+5+751+16+21+2+3+4
43+1453+9+117+6+2
```

これをbcコマンドに突っ込むと各行の式を順に解いてくれて、次のような出力になります。

```
bcの出力
$ tr , + < data | bc↵
41
888
1630
```

そして、(最後の出力例は省略しますが)numsumに通すと各行の数字を足して出力してくれます。

ここでちょっと重要なことを言いますが、bcもnumsumも行単位でデータを考えていることに気づいたでしょうか。**図2**もいち早く数字を縦に並べていますが、解き方がよくわからないときは、**データを掃除しながら縦に並べてから考えると、その後のコマンドを思いつく場合があります。**

第2問　「英字と日本語文字を分離する」

第2問は、次のようなファイルを扱います。問題は、「日本語と英語の作文に分離してください」というものです。出力は多くの人が見て妥当ならOKとします。さて、どう解きましょう。

```
$ cat text↵
恥の多い生涯を送This
isってa来ました。pen.
自分には、There
is人間のnothing生活
というものが、more
見当toつかないのです。
say.
```

1つ、ワンライナーにこだわらないなら(念のため言っておくと、こだわる必要はありません)、まず英語の文字を消して、次に日本語の文字を消すという2回の操作でできてしまいます。やってみましょう。

まず、英語の文字を消してみます。「英語の文字」というのは口語的ですが、ここでは「ASCIIコードの文字」を意味します。これらの文字を除去すれば、日本語だけが残ります。**図5**のようにtrの-d(delete)オプションを使います。

\000-\177というのは、8進数で表したASCIIコードの範囲(10進数で0～127)です。10進数の127が8進数でどんな数になるかは、

```
$ printf '%o\n' 127
177
```

で確認できます。

次に英語だけを抽出する方法ですが、これはもう一捻り必要です。trに-dのほか、-c(complement、補集合)を付けると結果が反転します。が、これだけだと**図6**のように複数の単語がくっついてしまいます。

▼図5　英語の文字(ASCIIコードに含まれる文字)を消す処理

```
$ tr -d '\000-\177' < text
恥の多い生涯を送って来ました。自分には、
人間の生活というものが、見当つかないのです。
```

▼図6　単にASCII文字以外を削除すると単語がくっつく

```
$ tr -cd '\000-\177' < text
This
isapen.  ←is a pen.がくっつく
(..略..)
```

▼図7　ASCII文字でない文字を空白に置換

```
$ tr -c '\000-\177' ' ' < text
                    This
is      a           pen.
            There
is        nothing
                  more
      to
say.
$ tr -c '\000-\177' ' ' < text | xargs
This is a pen. There is nothing more to say.
```

どうするかというと、削除ではなくて第1問のように置換でtrを使えばうまくいきます。**図7**のようにASCII文字でない文字をすべて半角スペースにして、その後xargsで余計なスペースや改行を取り払います。

さて、普通はこれで十分ですが、パズルなのでワンライナーでこれを片付けるという問題も考えてみましょう。2回に分けていた操作を1回でしなければなりません。これには第1問の最後に言ったことを実践してみるとうまくいきます。このデータの英単語ひとつひとつとそのほかの部分を、1行1個にしてみます。

図8にsedを用いた例を示します。sed(stream editor)はストリーム(パイプ)に文字列を通してその文字列を編集するためのコマンドです。文字列の置換をするときは、引数に**'s/<置換対象の文字列・正規表現>/置換後の文字列/g'**を指定します。**図8**では置換対象を正規表現で指定しており、**[\x0-\x7f]**が16進数でASCII文字を表し、その後ろの**+**が1文字以上の繰り返しを意味します。**\+**と書いてあるように、**+**はエスケープしないと使えません。置換後の文字列の**\n&\n**は、**&**が置換対象の文字列を表し、**\n**が改行を表します[注4]。つまりこれで、ASCII

注4) Linuxに入っているsedは、たいていがGNU sedで、ほかの環境(MacやBSD系のOS)のsedとは異なるものです。GNU sedは高機能で、本章で使っている機能はほかのsedでは使えない場合があります。

▼図8　英単語や日本語要素を1行1個にする

```
$ cat text | sed 's/[\x0-\x7f]\+/\n&\n/g'
恥の多い生涯を送
This


is
って
a
来ました。
(..略..)
↓grepで「任意の1文字」を検索すると空行が消える
$ cat text | sed 's/[\x0-\x7f]\+/\n&\n/g' |
grep .
恥の多い生涯を送
This
is
って
(..略..)
```

文字のシーケンス(単語や単語にピリオドを付けた文字列)の前後に改行が入ります。

次に、日本語と英語を分けていきますが、**図9**に分け方の例を1つ示します。これもシェル芸ではよく出てくる定石なのですが、**ソートしたい順に各行に番号を付けて、その後、sortに入力する**という方法をとります。**図9**の最初のワンライナーは、①のsedでアルファベットで始まる行の頭に1と付けて、②のsedで頭が1でない行に2を付けています。②のsedの**^[^1]**は、「行の最初の1字が1で始まらない」という意味です。その後、**図9**の下のワンライナーのようにsortに入力しますが、引数の**-k1,1**は「1列目をキーにする」、-sは「安定ソート」という意味です。安定ソートというのは、キーが同じなら最初に並んでいた順番を変えないという意味で、ここでは出てきた語順を変えないために指定しています。

さて仕上げましょう。筆者は**図10**のように持ち込みました。説明は図のキャプションのとおりです。**say.恥**のところにスペースを入れたいところですが、これはお任せいたします。

第3問 「最長の重複文字列を調べる」

さて、最後の問題です。**図11**のようなデータがあるとします。問題は、「この中の重複する文字の並びで最長のものはどれでしょう」というものです。たとえばこの中でよく見ると「ctgg」という並びは3回登場しますが、5文字以上でこのような並びがないならば、「ctgg」が正解となります。改行は無視して考えましょう。

この問題も、攻略は**1行1データに加工することから始まります**。**図12**に、筆者がこの問題を解くためにまずやったことを示します。tr

▼図9 各行にインデックスを付けて安定ソートする

```
$ cat text | sed 's/[\x0-\x7f]\+/\n&\n/g' | ↵
grep . | sed 's/[A-Za-z]/1 &/' | sed 's/^[^1]/2 &/'↵
2 恥の多い生涯を送          ①                  ②
1 This
1 is
2 って
1 a
(..略..)
$ cat text | sed 's/[\x0-\x7f]\+/\n&\n/g' | ↵
grep . | sed 's/[A-Za-z]/1 &/' | sed 's/^[^1]/2 &/' | ↵
sort -s -k1,1↵
1 This
1 is
(..中略..)
2 恥の多い生涯を送
2 って
2 来ました。
(..略..)
```

▼図10 ソート後のデータを横に並べて数字と余計なスペースを消去

```
$ cat text | sed 's/[\x0-\x7f]\+/\n&\n/g' | ↵
grep . | sed 's/[A-Za-z]/1 &/' | sed 's/^[^1]/2 &/' | ↵
sort -s -k1,1 | xargs | sed 's/1 //g' | sed 's/ 2 //g'↵
This is a pen. There is nothing more to say.恥の多い生↵
涯を送って来ました。自分には、人間の生活というものが、↵
見当つかないのです。
```

▼図11 某ウイロイドのゲノム
(http://www.ncbi.nlm.nih.gov/nuccore/J02050.1)

```
$ cat cccvd↵
ctggggaaatctacagggcaccccaaaaaccactgcaggagaggccgcttgagggatccc
cggggaaacctcaagcgaatctgggaagggagcgtacctgggtcgatcgtgcgcgttgga
ggagactccttcgtagcttcgacgcccggccgcccctcctcgaccgcttgggagactacc
cggtggatacaactcacgcggctcttacctgttgttagtaaaaaaaggtgtccctttgta
gcccct
```

▼図12 AWKで1文字ずつシフトして5文字ごとに出力

```
$ cat cccvd | tr -d '\n' | ↵
awk '{for(i=1;i<=length($0);i++){print substr($0,i,5)}}'↵
ctggg
tgggg
gggga
(..中略..)
gcccc
cccct
ccct
cct
ct
t
```

は改行をとるためで、次のawkに1行で文字列を送り込むために使っています。AWK(コマンド名はawk)はC言語風の文法を持つ最古のスクリプト言語で、ここでは、読み込んだ1行のデータをfor文で1字ずつずらしながら5文字出力するという処理をしています。**$0**が読み込んだ行全体を表し、**substr($0,i,5)**がi番目から5文字の部分文字列を返す関数です。C言語と異なり、文字列を数えるときは0番でなく1番から始めます。

このあと、**図13**のように**sort | uniq -d**とやると、重複した文字の並びだけ出てきます。**uniq -d**の-dは「重複して存在する行のみを出力」という意味です。

そして、**図13**のsubstr関数の第3引数(文字列の長さ)を重複がなくなるまで増やしていくと、求めるべき答えが出てきます。**図14**にその様子を示します。答えは7で、長さ7の文字列が3種類、重複しています。

さて、これだと何回もコマンドを実行しなければならないので、ワンライナーでやってみましょう。いや、何回もコマンドを実行したほうが楽ですが……。**図15**に結果を示します。要は手で変えていた数字を変数にして、for文で変えているだけです。文字列には、文字列の長さを頭にくっつけて出力していますが、これがないとうまくいきません。**図12**の最後の数行を見るとわかりますが、各長さの文字列を出力した最後に、指定した長さより短い文字列が(しかも同じものが何度も)出てくるからです。

▼図13　重複した文字列を出力

```
$ cat cccvd | tr -d '\n' | ↵
awk '{for(i=1;i<=length($0);i++){print substr($0,i,5)}}' | ↵
sort | uniq -d↵
aaaaa
aaacc
aatct
(..略..)
```

▼図14　文字列の長さを増やしていく

```
$ cat 【中略】 substr($0,i,6)}}' | sort | uniq -d↵
aaaaaa
aggaga
(..中略..)
tacctg
$ cat 【中略】 substr($0,i,7)}}' | sort | uniq -d↵
ccgcttg
ggagact
ggggaaa
$ cat 【中略】 substr($0,i,8)}}' | sort | uniq -d↵
$
```

▼図15　AWKでfor文を1つ増やし、文字列の長さを可変にして実行

```
$ cat cccvd | tr -d '\n' | ↵
awk '{for(j=1;j<=length($0);j++)↵
for(i=1;i<=length($0);i++){print j,substr($0,i,j)}}' | ↵
sort -n | uniq -d↵
(..中略..)
6 ggggaa
6 tacctg
7 ccgcttg
7 ggagact
7 ggggaaa
```

まとめ

本章では3つ、シェル芸の問題を出して解答例を説明しました。出した問題はパズルでしたが、「データを行ごとにそろえる」など、実戦でも使える定石を紹介しました。

ワンライナーを見ただけだと、とても書こうという気にはならないかもしれませんが、データの加工の手順が頭の中で整理できるようになれば書くのはそんなにたいへんではなくなり、「ちょっとした調べ物」の効率が飛躍的に向上します。

シェル芸勉強会の過去問は「https://blog.ueda.asia/?page_id=684」に置いてありますので、ぜひ楽しんでワンライナーを書く力を付けていただければと。SD

仕事でシェルスクリプトを使うときに気をつけたいこと

Author 今泉 光之（いまいずみ みつゆき） USP友の会 幹事 Twitter @bsdhack

多くのユーザや、ビジネスにおける顧客に配布する「製品」にシェルスクリプトを組み込むとき、どんな点に気をつけて作成すれば良いのでしょうか？ 「可読性」「堅牢な設計」「汎用性」の3つの要件を満たすためにどんな心構えと具体的なコーディングが必要なのかを解説します。

本章で言う「仕事」とは製品の開発ではなく製品そのものを示します。製品として配布されるソフトウェアにはシェルスクリプトが付属する場合が多くありますが、ここではそのようなシェルスクリプトを作成する際の留意点などを取り上げたいと思います。

可読性について

シェルスクリプトをはじめとするスクリプト言語は、コンパイラ言語と異なりソースがそのまま読めてしまうので、できればきれいで読みやすいコーディングを心がけましょう。きれいなコーディングについては諸説あり、決定的な結論は出ていませんが、まずはインデントや改行をそろえること、変数名や関数名などの命名規則をそろえることを心がけると良いでしょう。

共通関数化

シェルスクリプト中での関数定義は賛否両論ですが、筆者は単機能の関数を定義して呼び出す方法をお勧めします。単機能の関数に処理単位を分けることで可読性が向上しますし、製品の仕様変更などによる修正も、関数化したほうが容易に対応できるからです。

また、ログの出力処理なども関数化しておくことにより、次のメリットがあります。

・出力メッセージのフォーマットが共通化可能
・ログの出力先をsyslogもしくは指定されたファイルに容易に変更可能

シェルスクリプトで関数を定義する方法はPOSIX（後述）に次の記述があります。

```
fname() compound-command[io-redirect ...]
```

ここでcompound-commandは、**(compound-list;)**か**{compound-list;}**と定義されているので、ある関数fooは、

```
foo() { コマンド; … }
```

もしくは、

```
foo() ( コマンド; … )
```

とすることで定義できます。

{compound-list;}形式の場合はカレントシェル（スクリプトを実行しているシェル）で実行されますが、**(compound-list;)**形式の場合はサブシェルで実行されるようになります。サブシェルとは、カレントシェルから起動された子プロセスのシェルで、シェル変数などはカレントシェルから引き継がれますが、サブシェルで変更したシェル変数はカレントシェルでは

参照できないので注意が必要です。

通常の場合、関数はカレントシェルで実行することになりますので、前者の形式で関数を定義することになります。

入出力はUNIX標準

関数を定義する場合、入出力は可能であれば標準入出力を利用すると良いでしょう。標準入出力を利用することで関数はフィルタ処理が可能となるので、入出力をパイプ|で連結できるようになります。またフィルタコマンドとして効率良く動作させるために、出力には余計な情報や装飾などを付加せずに、必要最低限な情報のみを出力するようにすると良いでしょう。エラーメッセージや診断メッセージなどを出力する必要がある場合は標準エラー出力に出力するようにすれば、パイプによる処理を妨げることはありません。

ヒアドキュメント

複数行の固定データを出力するときにechoを複数回実行すると、途中行を修正するときに、>>をついうっかり>にしてしまうといった間違いをする危険性があります。

```
echo "1行目" >> ${outputfile}
echo "2行目" >> ${outputfile}
echo "3行目" >  ${outputfile}
   ↑この行の修正をするときに>>を>にしてしまった
echo "4行目" >> ${outputfile}
echo "5行目" >> ${outputfile}
```

ヒアドキュメントを利用するとファイルのオープンクローズが一度で済むので効率的ですし、各行のクォートが不要になるので可読性も高まるでしょう(詳細は第2章P.27を参照)。

```
cat << EOF > ${outputfile}
1行目
2行目
3行目
4行目
5行目
EOF
```

スペース区切りのデータの構文解析

スペースで区切られた複数データの構文解析には、setコマンドの利用も検討する価値があります。シェル組み込みのsetコマンドに引数を指定すると、それぞれの引数を位置パラメータに代入できます。組み込みコマンドにより複数の値を一度に処理でき、効率が良いのでお勧めです。ただし、位置パラメータは上書きされてしまうので、あらかじめ保存しておくなど注意が必要となります。

```
set -- $(LANG=C date '+%Y %m %d')
year=$1
month=$2
day=$3
```

setコマンドの直後に指定している--は、setコマンドにオプションの処理を終了させ、以降の語を引数として解釈することを指示しています。--を指定することで、最初の引数が-で開始されていてもオプションとしては解釈されなくなります。

この例で利用している$(コマンド)はコマンド置換と呼ばれるシェルの機能で、全体がコマンドの実行結果(この場合ですとLANG=C date '+%Y %m %d'の実行結果)に置き換えられます。このコマンドは現在の日付を2016 4 1のようにスペース区切りで出力するので、結果としてset -- 2016 4 1が実行されることになり、それぞれの値が位置パラメータにセットされます。

変数スコープ

シェルスクリプトでは原則として変数のスコープはスクリプト全体となります。ループ変数などでよく利用される変数iなども、呼び出された関数内部で呼び出し元と同じ値を保持しますし、関数の内部で変更された場合は当然、呼び出し元の値も変更されてしまいます。そこで各関数の先頭でlocalコマンドを利用することで、変数のスコープを関数内に限定できます。

```
foo()
{
  # 関数に局所的な変数i, jを定義する
  local  i j

  # ローカル変数の初期値は空の値
  echo $i

  for j in *
  do
    :
  done

  # 呼び出し元の変数を直接変更する
  k="……"
}
```

また、前述のサブシェルを利用して関数を定義しても変数のスコープはすべて関数内に限定できますが、変数の値はカレントシェルから引き継がれ、関数内で呼び出し元の変数は変更できないので注意が必要です。

```
foo()
(
 # サブシェルなのですべての変数は局所的な変数となる

  # 変数の値はカレントシェルから引き継がれる
  echo $i

  for j in *
  do
    :
  done

  # 呼び出し元の変数は直接変更できない
  k="……"
)
```

堅牢な設計について

想定している環境とは異なった環境でも正しく動作し、万が一スクリプトの不具合などがあった場合も回復が困難／不可能な状態に陥らないようにすることが大切です。

シェルオプション

シェルには動作を変更するためのオプションがたくさん用意されていますが、製品としてシェルスクリプトを作成するにあたっては、次のオプションは設定しておくことをお勧めします。

-e

テスト状態にないコマンドの実行に失敗した場合、直ちにシェルを終了します。テスト状態とは**if**や**while**構文の制御、**&&**や**||**の左辺値としてのコマンドを言います(下のコラム参照)。それ以外でエラーが発生した場合はスクリプトを終了しますので、スクリプト自体の構文エラーや、存在しない外部コマンドを実行しようとした場合などに想定外の挙動を防止できます。

-u

値が設定されていない変数を参照しようとした場合、直ちにシェルを終了します。タイプミスなどで値が設定されていない変数を参照することで発生する想定外の挙動を防止きます。

◆ ◆ ◆

シェルオプションはsetコマンドで設定できます。

```
set -eu
```

外部コマンドの実行パス

シェルスクリプト中で実行される外部コマンドはコマンド検索パス(\$PATH 環境変数)に従って検索されますが、個人の環境の場合は独自のコマンド検索パスが定義されている場合が

column 短絡リスト演算子

&&や**||**は短絡リスト演算子(short-circuit list operators)と呼ばれる制御構造で、**if**などと同様、コマンドの終了ステータスを検査して制御する演算子です。それぞれ**command1 && command2**、**command1 || command2**の型式で使われます。**&&**は最初のコマンド(**command1**)を実行し、終了ステータスが0なら次のコマンド(**command2**)を実行します。**||**は最初のコマンド(**command1**)を実行し、終了ステータスが0以外なら次のコマンド(**command2**)を実行します。

多く、別ユーザにより実行される場合やcronなどから実行される場合では、コマンド検索パスの違いから実行される外部コマンドが異なるなどの影響が考えられます。

また、故意もしくは偶然によりカレントディレクトリがコマンド検索パスの先頭付近に含まれる場合は、重大なセキュリティホールにつながる恐れもあります。

対策として、スクリプト内で実行する外部コマンドをすべてフルパスで指定するという方法が考えられます。その際、外部コマンドを同名のシェル変数に格納することで、可読性を損なうことなく、より一層安全なスクリプトになります。

```
ls="/bin/ls"
wc="/usr/bin/wc"
```

さらに開発の初期段階では、rmやmvなどのコマンドをechoに置き換えておくことで、非可逆的な作用を伴うコマンドの実行を抑制できるので意外と便利です。

```
rm="echo /bin/rm"
mv="echo /bin/mv"
```

想定される環境によっては、すべての外部コマンドのフルパスを指定することが困難な場合もあります。そのような場合はスクリプトの先頭でコマンド検索パス(\$PATH)を明示的に初期化して指定するだけでも有効です。その場合は、コマンド検索パスに格納するディレクトリを厳選する必要があるでしょう。

コマンド検索パスにヌルパス(連続するコロン**::**)がある場合はカレントディレクトリとして解釈されてしまうので、設定するパスにシェル変数を利用する場合は注意が必要になります。

汎用性について

シェル(/bin/sh)は、すべてのUNIXやUNIX風のシステムに共通に存在する唯一のスクリプト言語といっても過言ではないでしょう。そのため、シェルスクリプトはすべてのUNIXやUNIX風のシステムで共通に動作すると言えます。

しかし、実際にはそのシェルにもいくつものバリエーションが存在していて、それぞれで機能や仕様が微妙に異なっています。今回の特集で取り上げられているbashもシェルのバリエーションの1つで、オリジナルのシェルと比較すると数多くの拡張が施されている高機能なシェルです。たとえば、シェル変数に配列が利用できる機能や、プロセス置換機能など、とても便利な機能があります。

自分だけ、もしくは自分のプロジェクトのメンーなど、実行する環境がある程度特定できる環境では、それらの機能を利用した高機能なシェルスクリプトを作成することは、悪いことではありません。しかし、製品として配布する場合は、どのようなシェルか特定できない場合もあります。

そこで、どのようなシェルでも正しく動作するように共通の機能のみを利用したスクリプトを作成すれば良いのですが、世の中に数多く存在するシステム上で動作するすべてのシェルの機能を包括的に調べることは現実的には不可能です。

そこで、共通の機能を定義したのがPOSIXです。

POSIXとは

POSIXとはさまざまなOSに共通のAPIを定義して、移植性の高いアプリケーションソフトウェアの開発を容易にすることを目的として、IEEEが策定した規格です。カーネルへのインターフェース(システムコール)、プロセス環境など多くの規定がありますが、ここではシェルとユーティリティに対する規定であるXSH項目について説明します。

POSIXはIEEEのサイト[注1]からPDF形式で無料でダウンロードできます。また、The Open Groupのサイト[注2]でいつでも自由に閲覧できます。とくにThe Open Groupのサイトには簡単な検索機能も備わっていますので、シェルの機能や外部コマンド、オプションがPOSIXで定義されているかどうかを簡単に検索できます。

POSIXの限界

共通のAPIを規定しているPOSIXに準拠すれば多くの(POSIX互換の)システムで動作することは規格が保証してくれます。それではPOSIXに完全に準拠したシェルスクリプトを作れば良いのでしょうか？

実は、POSIXには一般的に利用されるコマンドが定義されていないことが多々あります。たとえば、curlやwgetといったリモート環境へのアクセスコマンドをはじめとして、ifconfigやnetstatといった基本的なネットワークコマンドさえ定義されていません。

前述のlocalなどもPOSIXでは定義されていないので、関数内ローカルな変数の定義もできなくなってしまいます。ですので、POSIXの範囲内だけでスクリプトを作成することは、現実的には困難だったり不可能だったりする場合があります。もちろん、とくに製品としてのシェルスクリプトを開発する場合にPOSIXを意識するのは良いことではありますが、盲目的にPOSIXにこだわるのは現実的ではなく、良策とは言えないでしょう。

POSIXは今までにも何度も改訂されてきましたし、今後もニーズに合わせて改訂されることでしょう。そして改訂をうながすために、新しく便利な機能を積極的に利用して広めていくことも大切だと思います。

現実的な解決策

それでは、製品として配布するシェルスクリプトはどのように開発すれば良いでしょうか。

まず、シェルの機能はPOSIXで定義された範囲だけを利用するようにします。外部コマンドは、POSIXで定義されているコマンドの場合は、オプションを含めてPOSIXの範囲で利用するようにします。ifconfigなどPOSIXで定義されていないコマンドに関しては、製品がターゲットとしているシステムで利用可能かどうかをある程度情報収集したうえで、利用できそうであれば利用するようにします。どうしても標準以外のシェル機能や外部コマンドを必要とする場合は、シェルスクリプトによる実装ではなく、ほかの言語による実装を検討してみるのも良いと思います。

本来的にはシェルスクリプトは単純な処理を簡単に効率良く実施する用途に向いているので、あまりに複雑な処理はシェルスクリプトの範囲を超えている可能性もあります。

なぜ汎用性が必要なのか

たとえば、bash固有の機能を使ったシェルスクリプトは、bashでないと動作しません。共通のスクリプト言語ではなく、動作させるための特定のスクリプト言語(bash)が必要になります。「bashのインストールなんて簡単」という意見もありますが、金融機関など監査が非常に厳しい環境などでは、bashに限らずソフトウェアを導入すること自体が非常に困難な場合があります。それ以外にも、同種の競合ソフトウェアが複数存在する場合に、依存するソフトウェアの有無が決め手となる場合も実際にありました。

製品としてシェルスクリプトを作成する場合、特定のシェルや環境、システムに依存した機能を使用せず、汎用的なシェルスクリプトとしたほうが、より一層ビジネスチャンスにつながる機会が増えるでしょう。SD

注1) http://www.techstreet.com/ieee/products/vendor_id/5219

注2) http://pubs.opengroup.org/onlinepubs/9699919799/

[速報]Bash on Windowsのしくみ

Author 真壁 徹（まかべ とおる） 日本マイクロソフト㈱

Windows 10の上でUbuntu、bashが動くというニュース、気になっている人も多いのではないでしょうか？ 本章ではBash on Windowsが動くしくみについて整理し、WindowsでUbuntu、bashが動くことで開発者目線で何が変わるかについて考察します。

Windowsの上でbashが動く?

Microsoftの開発者向けカンファレンス「Build 2016」で、"Bash on Ubuntu on Windows"（以降Bash on Windows）が発表されました。その名のとおり、Windowsの上でUbuntu、bashを動かすしくみです。

はじめにお断りしておきます。この記事は発表直後、2016年4月時点の情報に基づいており、その段階ではまだプレビュー版で、insiderプログラム登録者に限って配布されています。ですので、今後変化していくであろう内部実装の細部や仕様、制約ではなく、提供にいたった背景や狙いを中心にまとめたいと思います。

誰のニーズに応えるため?

Bash on Windowsは、アプリ開発者のニーズに応えるべく開発されています。近年、OSSを活用したシステムやモバイルアプリ開発では、Windowsではなく Macを使うアプリ開発者が増えています。それはMac OS XがUNIXにルーツを持つOSで、Linux向けアプリが開発しやすいこと、また、iOSアプリが開発できること[注1]、この2点が大きな理由です。Bash on Windowsでは、前者のニーズに応えようというわけです。

裏を返せば、Bash on WindowsはLinuxアプリの本番実行環境を意図していません。また、現在PowerShellが担っているWindows管理者向けスクリプティングツールのポジションも狙っていません。Windows Serverではなく、あくまでデスクトップ／ラップトップ向けOSであるWindows 10の機能です。insiderプログラム登録者に配布されているWindows 10では、OS設定で開発者モードを有効にして、明示的に機能追加する必要があります。

Windows上でbashを動かすしかけ

では、どのようにWindows上でUbuntu、bashを動かすのでしょうか。そのしかけを見ていきましょう。

コンピュータリソースをコントロールするOSカーネルはあくまでWindowsです。ですが、その上でユーザモードのUbuntuが動きます。Windowsでbash.exeを起動するとUbuntuがロードされ、コンソールから利用できるようになります。

図1からわかるように、UbuntuとWindowsカーネルの間に中間層があります。この層で、Linuxのシステムコールを Windowsカーネルへと橋渡しします。これが新たに開発された、

注1）Build 2016では、iOSやAndroid向けアプリをWindows上のVisual Studioで開発できる「Xamarin」の無償提供も発表されました。

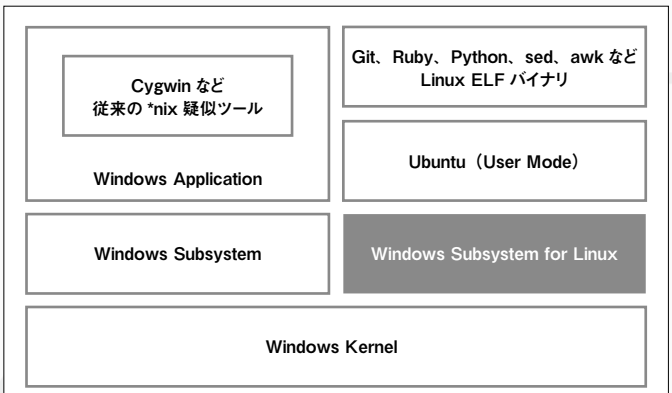
Windows Subsystem for Linux（WSL）です。

Hyper-VやVirtualBoxのように、仮想マシンの上でUbuntuを動かしているわけではありません。よって必要リソースが少なくて済みます。また、CygwinのようなWindowsアプリではないので、動かしたいアプリをBash on Windows向けにリコンパイルする必要もありません。Linux向けExecutable and Linkable Format（ELF）バイナリ形式で動きます。

WindowsはWindows NTの時代に、サブシステムの考え方を取り入れました。使用するサブシステムによって振る舞いを変えることができます。過去にはOS/2やUNIX向けサブシステムなどもありましたが、このたび新たなサブシステムとして、Linux向けサブシステム[注2]の提供を開始した、というわけです。

なおBash on Windowsは、Ubuntuの開発をリードしているCanonicalとのパートナーシップのもと、商用製品としてサポートできる体制を作っています。技術的にほかのディストリビューションが動く可能性は否定しませんが、製品として責任を持って提供するため、現在のサポート対象はUbuntuのみです。

▼図1　Windows Subsystem for Linux概念図[注3]



※シェルとしてのbashはubuntuに含まれています。

できることと、対応の「優先度」

ELFバイナリが動くということで、あらゆるLinux向けアプリが動くことを期待したくなりますが、そこはやはり新たな中間層をはさんだゆえの制約があります。実際、不具合も見つかっています。すべてを早期に解決することは難しいため、そもそもの目的である「アプリ開発者」のニーズを判断基準に、開発と対応の優先度を、たとえば次のように決めています。

優先度の高い機能

- 標準的な*nixコマンドラインツール（sed、awkなど）
- ニーズの多い言語向け開発、実行環境（Ruby、Pythonなど）
- 人気のある開発ツール（Gitなど）

優先度の低い機能

- GUI（GNOME、KDEなど）
- サーバ向けデーモン（MySQL、Redisなど）

もちろん、開発用途でMySQLなどのサーバ向けプロセスを動かしたいというニーズはあるので、そのようなケースでは長時間連続稼働への対応の優先度を下げて改善します。

ファイルシステム

bashからWindowsのファイルシステムへアクセスできます。たとえばCドライブは/mnt/c/と見えます（**図2**）。

ただし、現時点ではbashからWindows管理下のファイルシステムへアクセスすると、パー

注2） ちなみに、WSLにLinuxのコードは含まれていません。

注3） 簡略化した図ですので、詳細はhttps://blogs.msdn.microsoft.com/wsl/2016/04/22/windows-subsystem-for-linux-overview/をご覧ください。

ミッションが777である、シンボリックリンクが張れない、特殊ファイルのエラーが出る、などの課題があります。改善を期待しつつも、生まれが違うのでしかたない、という割り切りも必要でしょう。

期待される変化

では、開発者目線で、Bash on Windowsによってどのような変化が起こりそうなのかを考えてみましょう。

開発環境が、より軽量に、サクサクと

これまでWindowsでLinux向けアプリを開発するには、サーバにtelnet/sshするか、ローカルにLinux向け仮想マシンを作っていたと思います。それが不要になります。ネットワークに不自由な場所でも開発しやすく、また、仮想マシン用のメモリやディスク容量で悩まされることも少なくなります。起動も速いです。そして、Windowsと仮想マシンの間のファイル共有でSMBを駆使して、なんていう苦労もなくなります。Windows向けRubyやPythonでの“あるある”――パッケージ未対応やファイルパスの扱いでイラっとすることもありません。

OSSを企業が活用しやすくなる

Macを使うユーザが増えたとは言え、管理やガバナンスの観点から、Windowsに社内端末を統一している企業は多いでしょう。そのWindowsが製品機能としてbashを提供して環境が整うと、OSSを活用した開発がしやすくなるはずです。これは大きな変化だと思います。

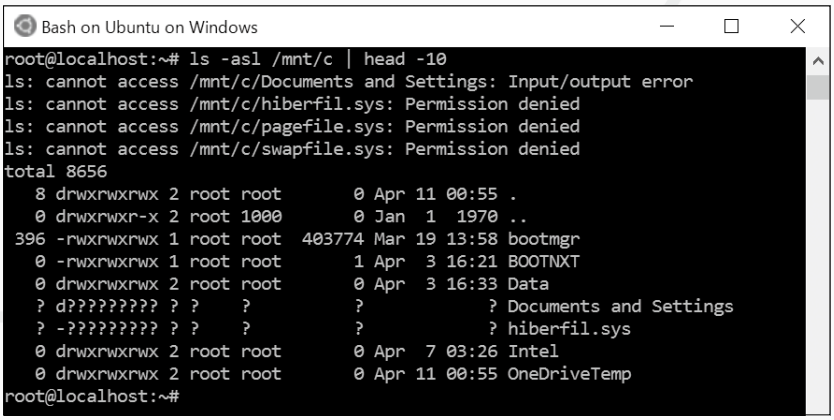
なお、「そんなことをしてMicrosoftのビジネスは大丈夫なの？」と言われることがありますが、心配はご無用です。なぜならMicrosoftはLinux、OSSを新しいビジネスの1つの柱と考えているからです。たとえばMicrosoftのクラウドサービスであるAzureでは、すでにLinux仮想マシンの比率が全体の25%を超えており、その勢いは日々増しています。

フィードバック歓迎

Bash on Windowsは始まったばかりのプロジェクトです。ということは、みなさんの要望が製品に反映される可能性、余地が大きいと言えます。要望や不具合を、フィードバックしてみませんか。要望はUser Vioce Portal[注4]、不具合はGithub Issue Tracker[注5]からお願いします。SD

注4）https://wpdev.uservoice.com/forums/266908-command-prompt-console-bash-on-ubuntu-on-windo/category/161892-bash
注5）https://github.com/Microsoft/BashOnWindows/issues

▼図2　bashコンソールからWindowsファイルシステムがみえる



bashならぬfishを知っていますか？

Author 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

fish（friendly interactive shell）は、高機能・設定いらず・シンプルが売りの最先端インタラクティブシェルです。本章はbash特集の番外編として、そのfishを紹介。大きな特徴である強力な補完・ハイライト機能から、bashとの違いまで解説します。

bashもいいけど「fish」もね

Linuxのシェルと言えばbashです。多くのLinuxディストリビューションが、bashをデフォルトのインタラクティブシェルに採用しています。一部のディストリビューションを除いて、シェルスクリプトにもbashが使われています。Mac OS Xもそのあたりは同じです。この夏のアップデートからWindows 10でもbashが利用できるようになる予定ですし、今後ますますbashに触れる機会は増えるでしょう。

シェルにさらなる機能を求める場合、またライセンス的にbashが利用できない環境ではzshが使われることもありますし、メンテナンス性や軽量であることを最優先してkshが使われるオペレーティングシステムもあります。歴史的にcshやtcshがデフォルトになったものもありますが、デスクトップやサーバで使われているインタラクティブシェルは、数の上でbashが圧倒的と言えると思います。

ここで1つ、日本ではまだあまり名前が知れていない「fish」[注1]というインタラクティブシェルを紹介したいと思います。主要なインタラクティブシェルの中ではもっとも後発のもので、bashの代わりを検討するのであれば有力な候補です。

注1） https://fishshell.com

便利で簡単! お手軽最強シェル

fishは最後発だけあってよくできています。bashもzshも似たような機能は実現できるのですが、fishではそれがデフォルトで提供されているという特徴があります。簡単に特徴をまとめるとすれば次のようになります。

- 設定せずとも効果的に利用できる状態になっている
- 補完機能がbashやzshを超える便利さ
- ハイライトが多用されていて便利
- 制御構文がシンプルで覚えやすい

bashやzshは開発された時代背景もあって、デフォルトの状態ではすべての機能が有効にはなっていません。fishが開発された時代はもっと後で、インタラクティブシェルがマシンパワーをすべて持っていってしまうといったことがない状況でしたので、デフォルトですべての機能を利用しようとします。しかも提供する機能は取捨選択されていて、とくに気にしなくてもいつの間にか使えている、といったところまで洗練されています。

Mac OS X、Linux、*BSDのどのプラットフォームでもfishのパッケージが提供されているようですので、それぞれの環境に合わせてインストールして一度使ってみてほしいと思いま

す。覚えるのは、困ったらTabを押すということと、うっすらと補完候補が出てきたらCtrlを押しながらFを押すということです。このCtrl＋Fがとくに強力で、一度慣れてしまうと、もうこの機能なしでは生きていけない体になってしまいます。

fishの便利な機能を見てみよう

fishをインストールしたらさっそく使ってみましょう。端末でfishとコマンドを実行すればfishに変わりますし、気に入ったらchsh(1)[注2]コマンドなどでデフォルトのシェルをfishに変更すれば良いと思います。

fishを起動すると次のようなプロンプトが表示されます。「ユーザ名@ホスト名 短縮パス >」がデフォルトのプロンプトです。コマンドを入力すると、次のようにコマンド部分に色がつきます(モノクロページですのでグレーですが)。

```
daichi@parancell ~> ls -al
```

間違ったコマンド名が入力されると、入力した文字の色が赤のままになり、入力ミスしていることに気づけます。

bashやzshのように、fishでもTabが補完処理のフックキーになっています。たとえば、次のようにコマンドを入力した状態でTabを押します。

```
1. fish  /Users/daichi/Documents/mynavi (ssh)
daichi@parancell ~/D/mynavi> cd
```

次のように補完候補が表示されます。

```
1. fish  /Users/daichi/Documents/mynavi (ssh)
daichi@parancell ~/D/mynavi> cd
creative/  lwt/        news-2008/  news-2012/  news/
java/      news-2005/  news-2009/  news-2013/  python/
javaapi/   news-2006/  news-2010/  news-2014/  tool.de/
kikaku/    news-2007/  news-2011/  news-2015/  zsh/
```

この場合コマンドとしてcdを入力してあるので、候補としてはディレクトリが表示されます。fishがおもしろいのはここからさらに先です。ここで再度Tabを押します。

```
1. fish  /Users/daichi/Documents/mynavi (ssh)
daichi@parancell ~/D/mynavi> cd news-2014/
creative/  lwt/        news-2008/  news-2012/  news/
java/      news-2005/  news-2009/  news-2013/  python/
javaapi/   news-2006/  news-2010/  news-2014/  tool.de/
kikaku/    news-2007/  news-2011/  news-2015/  zsh/
```

今度は補完候補の中から入力すべきものが選択できるようになります。候補の選択はカーソルキーで選択できますし、ここで文字列を入力するとその文字列で絞り込みを実施して表示させる候補を絞り込めます。とても便利です。

コマンドの出力結果に加工を行えます。次のようにgrep(1)コマンドで特定のキーワードを検索すると、その結果に対してキーワードに色が付くようになります。どの部分で一致したのかが一目瞭然です。

```
1. fish  /Users/daichi (ssh)
daichi@parancell ~> grep FreeBSD /COPYRIGHT
# $FreeBSD: releng/10.3/COPYRIGHT 292984 2015-12-31 11:23:39Z bz $
The compilation of software known as FreeBSD is distributed under the
Copyright (c) 1992-2016 The FreeBSD Project. All rights reserved.
daichi@parancell ~>
```

そしてここからが真骨頂です。fishではコマンド履歴や文脈を加味して入力の補完候補が表示されます。前述のgrepコマンドのあとで、もう一度grepと入力を始めると、その先に入力すると考えられる内容が、うっすらと先行表示されるようになります。

```
1. fish  /Users/daichi (ssh)
daichi@parancell ~> gre FreeBSD /COPYRIGHT
```

この状態でCtrl＋Fを押すと自動的に補完候補が入力されます。

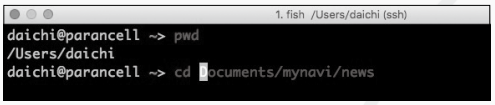
```
1. fish  /Users/daichi (ssh)
daichi@parancell ~> grep FreeBSD /COPYRIGHT
```

Ctrl＋Fの機能はとても便利で、一度このキーが手になじんでしまうと、もうこの機能のないインタラクティブシェルは使えなくなって

注2) 各コマンドの後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。

しまいます。

この機能だけだと、たとえばcdのように頻用するコマンドでは、直前に入力したディレクトリが候補として表示されるため役に立たないのではないかと思うところですが、fishは文脈を加味して補完候補を表示します。たとえば、次のようにcdコマンドでカレントディレクトリを移動したとします。



移動した先でさらにcdコマンドを入力すると、そのカレントディレクトリに適したディレクトリを候補として表示してくれます。

このように、fishではユーザが迷うことなく各種の補助機能を利用できるように工夫がされています。新しい設定をする必要があるとか、それらを特定のキーに割り当てる必要があるとか、そういった必要性がありません。Tabと Ctrl + F だけ覚えておけばユーザに便利で強力な環境を与えてくれます。

bashとの大きな違い

制御構文

fishとbashの大きな違いは制御構文にあります。bashからzshに移行した場合は、それほど違いに困りません。zshはBourne Shell系の制御構文を採用しているためbashと同じ書き方が利用できるからです。

fishではこの部分をばっさり切っています。

最終的には慣れの問題だと思いますが、fishの制御構文はbashのそれよりもかなりシンプルです。すでにbashの制御構文に慣れているなら、fishの制御構文を飲み込むのにほんと時間はかからないでしょう。bashがベースとしているBourne Shellは、制御構文がALGOLに類似しているため現在主流のプログラミング言語と比較して独特ですし、変数の展開と文字列のクォート規則はお世辞にも簡単とは言えません。そのあたり、fishはシンプルです。

次にfishの基本的な機能や制御構文の代表的な使い方をまとめておきます。

・変数代入

```
set a b
```

・コマンド置換からの変数代入

```
set a (コマンド)
```

・関数

```
function 名前
        コマンド
end
```

・or構文

```
or コマンド
```

・and構文

```
and コマンド
```

・not構文

```
not コマンド
```

・if構文

```
if コマンド
        コマンド
else if コマンド
        コマンド
else
        コマンド
end
```

・switch構文

```
switch $変数
case 項目 項目
        コマンド
case 項目
        コマンド
case '*'
        コマンド
end
```

・for構文

```
for i in リスト
        コマンド
end
```

・while構文

```
while コマンド
        コマンド
end
```

Bourne Shell系との大きな違いは、

- まず変数を設定するにあたって=を使わない
- コマンド置換が$()ではなく()である
- グルーピングの{}を使わない
- do〜done や if then〜fi、case〜esac のような表記ではなくendで閉じるだけである
- 1つ前のコマンドの結果が0かそれ以外かで

コマンドの実行可否を変えるor、andがある
・コマンドの成否結果を反転させるnotがある

と、いったところです。

fishの配布物にshareというディレクトリがあり、この下に補完設定やエイリアスなどに相当する機能が実装されています。とくにfunctionsディレクトリ以下のスクリプトが参考になると思います。

```
# tree -L 1 share↵
share
|-- completions
|-- config.fish
|-- functions
|-- man
\`-- tools

4 directories, 1 file
```

どのオペレーティングシステムでもこれらがインストールされるはずですので、ls.fishファイルをlocate(1)コマンドやfind(1)コマンドで探すなどして、その周辺のfishスクリプトを読んでみてください。いくつか読めば、fishスクリプトの書き方はほぼ把握できると思います。

エイリアスではなく関数

fishでは機能の種類もシンプル化されています。エイリアスは存在しておらず、関数を定義することがエイリアスに相当しています。bashやzshでは変数を定義したり組み込み関数を実行したりと、さまざまな方法で機能の変更・設定をすることになりますが、fishでは関数を定義するだけです。

問題といえば、bashやzshで使っているエイリアスがそのままではfishで使えないということです。これについては、たとえばbashやzshで使っているエイリアス設定を~/.aliasesといったファイルにまとめておいて、**リスト1**のようにfishで自動的に関数に変換して取り込むようにすれば共有できます。fishの設定ファイルは~/.config/fish/config.fishですので、このファイルに共有の設定を書き込んでおきます(下のコラム参照)。

まずは使ってみようfish

インタラクティブシェルはユーザの慣れが出る部分ですので使い始めには違和感を覚えると思いますが、いったん慣れてしまうとfishはもうほかのインタラクティブシェルに戻る気が起こらないような便利なシェルです。ぜひ一度使ってもらえればと思います。SD

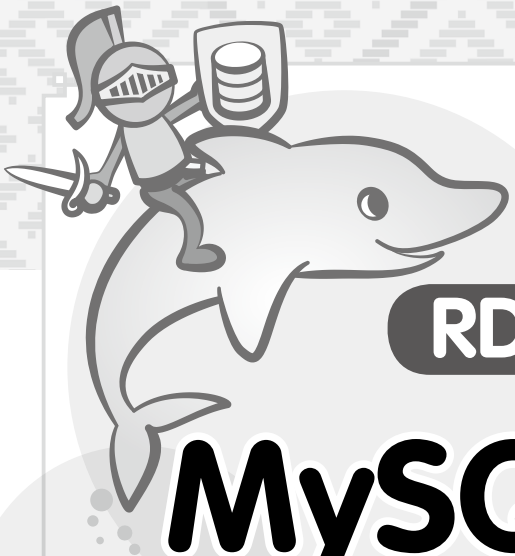
▼リスト1　bashやzshとエイリアスを共有する設定

```
if [ -f ~/.aliases ]
        sed -e 's/$(/(/g' -e 's/&&/; and /g' ~/.aliases | source
end
```

column fishはパイプ中の組込みコマンドに対してサブシェルを生成しない

エイリアス共有のためのスクリプト(**リスト1**)を見て、シェルの構造に詳しい方なら1つ疑問を感じたはずです。あのコード、bashやzshでは思ったように機能しません。これはパイプに接続されたsource組み込みコマンドがサブシェルで実行されるため、本体のシェルには影響を及ぼせないからです。

fishはパイプに接続されていても、組み込みコマンドの実行時にはサブシェルを生成しません。そのままシェルで処理を行います。このため、source組み込みコマンドによる処理がシェル本体に反映されています。シェルのプロセス実行構造に詳しくないユーザにとっては、この挙動のほうが親しみやすいものだと思います。



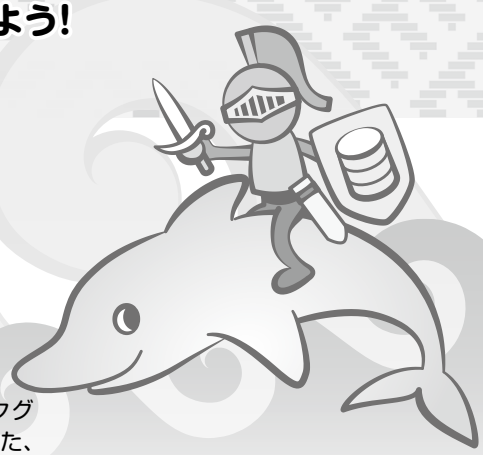
第2特集

RDBの学び方

MySQLを武器にSQLを始めよう!

ソフトウェア開発の基礎の基礎

ソフトウェア開発になくてはならないRDB。この操作にはSQLを学ぶことが必要です。もっとも身近なオープンソースRDBMSであるMySQLをベースに、SQLの基礎の基礎をしっかり学びましょう。最初はMySQLの歴史を振り返り、そのしくみと流れを理解します。プラガブルストレージエンジンという本質の1つを押さえてください。そのあとはMySQLのインストールです。Mac、Windows、Linuxの各環境ごとに解説をしました。最後の章は、実際にデータベースを操作してみます。電子掲示板システムを例に取り上げ、実際に手を動かして試してみてください。MySQLについて疑問が出てきたらユーザグループを頼りましょう。そのためのリンクもまとめました。SQLは一生使える技術です。これを機会にぜひマスターしておきましょう!



第1章 MySQLのしくみを探る

MySQL内部のアーキテクチャ

この章ではMySQLの内部で、フォアグラウンドスレッドとバックグラウンドスレッドがどのように動作しているかを解説します。また、MySQLのパラメータについても実例を挙げてグローバルスコープとセッションスコープの動作やその違いについて解説します。

Author yoku0825

MySQLのしくみ

MySQLはRDBMS(Relational DataBase Management System：リレーショナルデータベース管理システム)です。SQLと呼ばれる言語で問い合わせを行うことで、データの格納やデータの取り出しを行います。

とあるORM(Object-Relational Mapping：オブジェクト関係マッピング)の作者が「RDBはアプリケーションからすると一番アクセスしやすい永続化できるグローバル変数」と言っていたことがありますが、筆者にはこれは正鵠(せいこく)を射ていると思われ、「永続化を保証する(クラッシュしてもコミットに成功したデータは失われない)」「複数のサーバから同じ値に(場合によっては排他制御をしながら)アクセスできる」「複数の言語で対応するライブラリがある」など、アプリケーションで実装するには骨の折れる機能を提供してくれます。

RDBMSはこれらの(本来複雑な)実装を抽象化するためのレイヤとして機能します(もう1つ、述語論理に基づいた演算を提供することもRDBMSの重要な機能ですが、これについてはSQLの側面が強いため今回は置いておきましょう)。しかし、SQLの向こうの側、隠蔽されたMySQLの内部の世界に興味はないでしょうか。

本章では、MySQL内部のアーキテクチャをざっと説明していきます。

フォアグラウンドスレッドとバックグラウンドスレッド

MySQLはシングルプロセスマルチスレッドモデルを採用しています。1つのmysqldプロセスの内部に、いくつかのバックグラウンドスレッドと、1コネクションあたり1つのフォアグラウンドスレッドを起動します(Unix、Linux系のOSでMySQLを動かしたことがあればmysqld_safeというプロセスを見たことがあるかもしれませんが、これはmysqldをラップしているシェルスクリプトであり、RDBMSの機能を提供しているプロセスではありません)。

バックグラウンドスレッドの多くはInnoDBの非同期スレッドです。数で言えばMySQLのほとんどのスレッドはフォアグラウンドスレッドで占められます(繰り返しになりますが、1コネクションが1つのフォアグランドスレッドを占有しますので、MySQLに100のコネクションがあれば100本のフォアグラウンドスレッドが存在します)。レプリケーション関連のスレッド(マスタのBinlog Dumpスレッド、スレーブのI/Oスレッド、SQLスレッドとも)もフォアグラウンドスレッドと位置付けられています。

コネクションをハンドルするフォアグラウンドスレッドのライフサイクルは、**図1**のようになっています。SELECTステートメントをベースにしているため記述が偏っていますが、INSERTやUPDATEステートメントでもほぼ同様の動きに

なります。

コネクションをハンドルするスレッドの動作

まず、TCPの3306番ポート、UNIXソケットファイルなどMySQLへの通信を待ち受けるのはmysqldのメインスレッドです。メインスレッドが待ち受けているところにクライアントからの接続要求があると、メインスレッドは自身をcloneし、作成された子スレッドを接続要求に割り当てます(そしてメインスレッド自身は再び待ち受けループに戻ります)。

ここで作成された子スレッドがフォアグラウンドスレッドとなり、接続要求の続きの処理から結果セットを返送するところまで、その接続元からの要求に対するすべての処理を担当します。

少し詳しく見ていきましょう。

MySQLのユーザ認証

MySQLの認証はOSの認証機能とは独立しており、アカウントの情報はMySQLの内部にテーブルとして保管されています(厳密には認証に利用されるアカウント情報はmysqldのメモリ上に展開されており、mysql.userを始めとするテーブル群はアカウント情報をメモリ上にリロードするためのスナップショットなのですが、ここでは同じものとしておきましょう)。

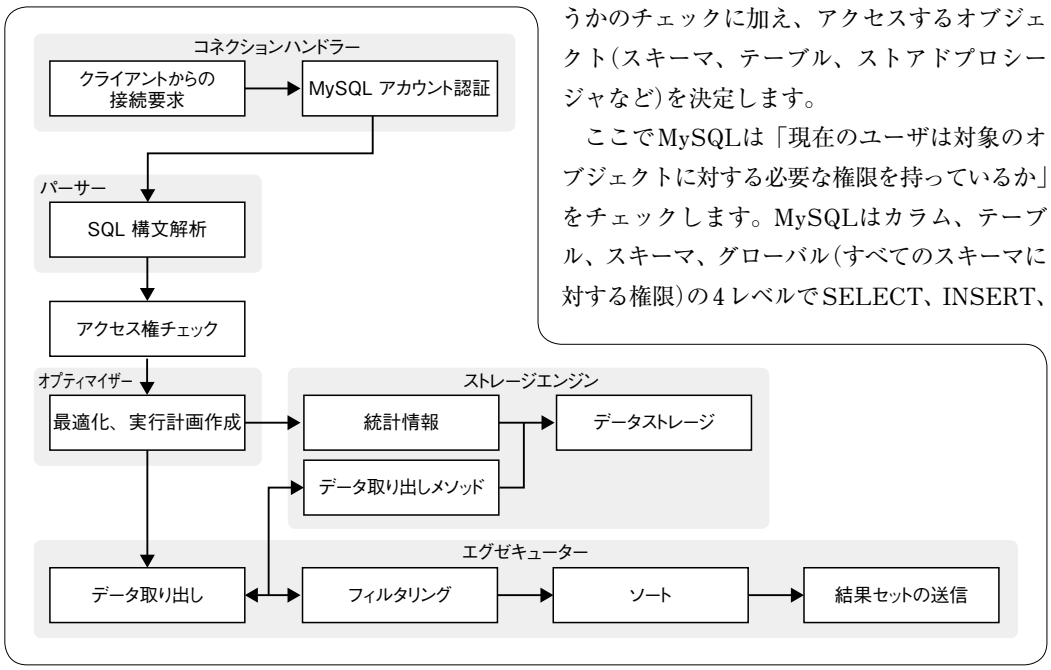
MySQLのアカウント情報は「ユーザ名」と「接続元ホスト」の組で一意に識別されます。“myuser@192.168.0.1”と“myuser@192.168.0.2”は別のアカウントです。この2つのアカウントしか存在しない場合に192.168.0.3のIPアドレスを持つサーバからmysql -umyuserでログインしようとした場合、「接続元ホスト」がマッチしないため接続できません。「ユーザ名」と「接続元ホスト」がアカウント情報にマッチした場合のみ、パスワードの検証に進みます(詳細は割愛しますが、デフォルトではチャレンジ・レスポンス認証を行います)。

SQL構文解析とアクセス権のチェック

アカウントの認証が済んだらSQLの構文解析を行います。ここではSQLの構文が正しいかどうかのチェックに加え、アクセスするオブジェクト(スキーマ、テーブル、ストアドプロシージャなど)を決定します。

ここでMySQLは「現在のユーザは対象のオブジェクトに対する必要な権限を持っているか」をチェックします。MySQLはカラム、テーブル、スキーマ、グローバル(すべてのスキーマに対する権限)の4レベルでSELECT、INSERT、

▼図1　フォアグラウンドスレッドのライフサイクル



UPDATE、DELETEなどの権限をそれぞれ設定できます。

クエリキャッシュ、ジェネラルログ(一般クエリログ)はこの段階で処理されます。

SQLの最適化と実行計画の作成

SQLは宣言型の言語です。データを「どのように」取り出すかを記述することはありません。

しかし実際問題データは「どこかに」格納されており、それを「どうにかして」「なるべく速く」取り出さなければなりません。

オプティマイザーと呼ばれるこのステージでは「要求された処理をなるべく速く実行するにはどのようにデータを取り出すのが良いか」を計算します。

インデックスを利用した方が良いのか、利用しない方が良いのか。このインデックスとそちらのインデックスではどちらがより速いのか。クエリを(数学的な等価性に基づいて)書き換えることで高速化はできないか。結合の順番はどちらを先にした方が速いのか。

これらを計算するために、オプティマイザーは「統計情報」を利用します。ここでいう統計情報とは、「テーブルに含まれるデータは全体でどの程度なのか」「インデックスのカーディナリティー(含まれる値のバリエーション)はどの程度なのか」などの情報です。

図1でも示したとおり、この統計情報は「ストレージエンジン」により提供されます。

データの取り出しから結果セットの返却まで

オプティマイザーで実行計画を決定したら、その実行計画に従ってデータを取り出します(あまり一般的な呼ばれ方ではなさそうですが、ソースコード上このステージはエグゼキューターと呼ばれていますので、本章でもそう呼ぶことにします)。

エグゼキューターはストレージエンジンに「どのインデックスを利用して(あるいは利用せずテーブルスキャンで)」「そのインデックス上からどの値を探しカーソルを合わせ」「最初の行を読み取り」「カーソルを次のリーフに移動し」「その行を読み取り」……という実行計画を指示し、指示を受けたストレージエンジンはエグゼキューターに対してデータを返却します。

ストレージエンジンからデータを集めながら、エグゼキューターは独自に結果をフィルタリングします。ストレージエンジンにできることはテーブルスキャンまたはインデックス単位での動作だけのため、インデックスだけで解決できないWHERE句やORDER BY句の処理はエグゼキューターが行います。

SQLで指示されたフィルタ処理とソート処理まですべてを適用し終えれば、最後はその結果セットをクライアントに返却します。

ここまでの複数のステージが、1つのフォアグラウンドスレッドによって行われています。バイナリログやスロークエリログの出力はこのステージで行われます。

レプリケーション関連スレッドの動作

MySQLは古く(バージョン3.23系列、2000年のリリースです)から長きに渡って非同期レプリケーションをサポートしてきました。レプリケーションとは「あるサーバで実行された更新ステートメントを」「別のサーバに転送する」ことで「データを同期する」しくみです。更新ステートメントを受け付けるサーバを「マスタ」、マスタへの更新ステートメントを転送されるサーバを「スレーブ」と呼びます(**図2**)。

マスタには「スレーブからのレプリケーション要求を受け付けるBinlog Dumpスレッド(またはsenderスレッド)」、スレーブには「マスタから更新情報を受け取るI/Oスレッド(またはreceiverスレッド)」と「更新情報を自身に適用するSQLスレッド(またはapplierスレッド)」が存在します。

また、コネクションスレッドが発行した更新ステートメントをシリアライズしてBinlog Dump

スレッドに渡すためのキューとして「バイナリログ」、I/Oスレッドが受信したバイナリログイベントをSQLスレッドに渡すためのキューとして「リレーログ」がそれぞれファイルとして存在します。

レプリケーションの構成方法はもう少し後に説明するとして、運用状態のレプリケーション構成は**図2**のようになっています(マスタ1台、スレーブ1台の構成の場合)。

運用中のレプリケーションの状態

図2中の"clientA"はマスタに接続し、更新ステートメントを実行するクライアントを表しています。先に説明したとおり"clientA"はマスタ上のフォアグラウンドスレッド(**図2**中では"コネクションスレッド")とやりとりし、マスタ上のデータを更新するSQLを実行します。

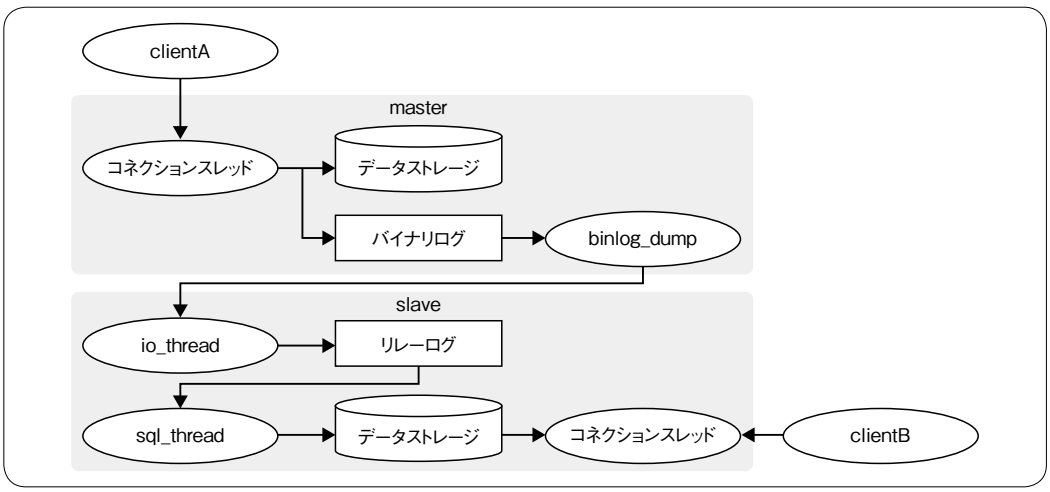
マスタ上のコネクションスレッドは先の説明のように認証を行い、SQLをパースし、実行計画に従ってマスタ上のデータストレージを更新するとともに、バイナリログへの書き込みを行います。コネクションスレッドが責任を持つのはここまでで、データストレージとバイナリログへの書き込みが成功した時点で"clientA"はステートメントのOK応答を受け取ります。ここまでがまず1つめのブロックです。

次に、マスタ上にはBinlog Dumpスレッドが常駐しており、これはバイナリログを監視しています。バイナリログへの書き込みを検知すると、書き込まれたイベントを読み込み、スレーブのI/Oスレッドに対してイベントを転送します。MySQLのレプリケーションはスレーブからのポーリングではなく、マスタからの自発的なプッシュです。Binlog Dumpスレッドはレプリケーション中のスレーブと1対1で存在し、スレーブの台数分Binlog Dumpスレッドが存在します。

スレーブに移ると、マスタからバイナリログのイベントを受け取る役割としてI/Oスレッドが存在します。I/OスレッドはBinlog Dumpスレッドからイベントを受け取り、リレーログにそれを書き込むところまでがその役割です。MySQL 5.7では複数のI/Oスレッドが稼働できますが、MySQL 5.6とそれ以前では1つのmysqldには1つのI/Oスレッドしか存在できませんでした(スレーブから見たマスタは常に1台しかありませんでした)。I/OスレッドとBinlog Dumpスレッドは常時接続し続け、マスタに更新ステートメントが実行されるたびにイベントがリレーログに記録されていきます。

最後の処理ブロックはスレーブのSQLスレッドを中心としたブロックです。SQLスレッドは

▼図2　マスタとスレーブの役割

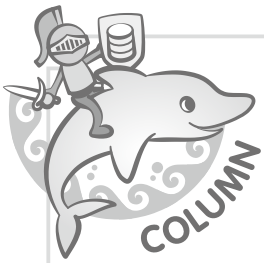


(マスタのBinlog Dumpスレッドがバイナリログに対してそうするように)リレーログを監視し、書き込みを検知するとそれをパースしてスレーブ自身のデータストレージに書き込みます。

図2中の"clientB"はスレーブに接続しデータを参照するクライアントです。このレプリケーションのしくみを通してデータの更新差分が伝搬され適用されるため、"clientA"が書き込みを行ったのはマスタのみであるにもかかわらず、スレーブに接続した"clientB"でも同じデータを読み込むことができます。

レプリケーションの構築

レプリケーションの構成にはいくつかのステッ



プラガブルストレージエンジン

ここまでの説明で何度か出てきましたが、「ストレージエンジン」はMySQLの中で「データの格納、保存、取り出しに責任を持つ」コンポーネントです。「統計情報の取得」や「1行を読み取る」「カーソルを次に進める」「1行を書き込む」などのAPIを持っています。

バージョン5.7.11現在のMySQLにはMyISAM、InnoDB、CSV、Merge、BlackHole、Memory、Archive、performance_schema、Federatedの9つのストレージエンジンがバンドルされており、それぞれ違ったフォーマットでデータを格納、保存、取り出します(データをいっさい格納しないストレージエンジンも含まれていますが)。

ストレージエンジンが分離されていることで得られるメリットは何でしょうか。それは、ストレージエンジンの差はエグゼキューターに隠蔽されている(さらには、エグゼキューターからもストレージエンジンを隠蔽するためのハンドラーというレイヤがあります)ため、SQLパーサーはストレージエンジン変更の影響を受けません。つまり、SQLを書き換えることなく複数のストレージエンジンを渡り歩くことができます。

たとえばMyISAMは旧来から長く使われているストレージエンジンで「少ないメモリ量でも軽快に動く」「排他制御がテーブル単位で並列性能が低い」「トランザクション非対応でクラッシュアンセーフ」「転置索引、空間索引に対応」などの特性を備え、InnoDBは「トランザクション対応」「排他制御はインデックス単位」「フォアグラウンドスレッドとバックグラウンドスレッドが協調して動作する」「転置索引、空間索引に対応」「クラッシュセーフを実現するための複雑な機構」などの特性を備えています。

Memoryストレージエンジンは「データ、インデックスをメモリ上にのみ格納する」ストレージエンジンです。そのため小さなテーブルをスキャンして少数の行を取り出す処理は非常に高速ですが、MySQLを停止させるとデータはすべて消えてしまいます。また、トランザクションにも非対応です。デメリットばかりのように思えますが、この特性はテンポラリーテーブルと非常に相性が良いです(集計関数の中間処理結果を格納するため小さなテーブルとなることが多く、ほかのスレッドから参照されないためクラッシュ後にデータが存在する必要はない)。

アプリケーションの特性に合わせて適切なストレージエンジンを選択することもDB設計上重要なこと……のように思えますが、2016年現在ではこれはさほど重要なことではありません。MyISAMはトランザクション非対応であり、本章の冒頭でRDBMSの利点として挙げた「永続化を保証する」ことができないからです。RDBMS側で永続化を保証できないとなると、それはアプリケーション側で保証しなければなりません。MyISAMへの書き込み命令はすべて行われたのか? 行われた更新はクラッシュ後もすべて適用されているか? 適用されていない更新がデータとして残っていないか? それらを逐一チェックするロジックを書くことは(少なくとも筆者には)とても面倒で困難です。せっかくのRDBMSを使うメリットを1つ手放すこともありませんので、ストレージエンジンはInnoDBを利用することをお勧めします。

プがあります。ここではCHANGE MASTER TOステートメントとSTART SLAVEステートメントを利用してレプリケーションを構築する時点で起こることについて説明しましょう。

まず大前提として、マスタとなるmysqldでバイナリログが出力される設定になっている(log_binオプション[注1]が有効にされている)必要があります(デフォルトでは無効になっています)。

また、バイナリログに「このクエリはどのサーバで最初に実行されたか」を識別するためのserver_id[注2]を、マスタ、スレーブで一意になるように指定します。マスタとスレーブで同じserver_idを指定した場合や、同じマスタに接続する複数のスレーブで同じserver_idを設定されたMySQLがある場合、レプリケーションが正しく動きません。MySQL 5.6とそれ以降ではserver_uuidもマスタとスレーブで異なる必要があります。server_uuidはデータディレクトリ(yumリポジトリを利用してインストールした場合のデフォルトは/var/lib/mysqlです)のauto.cnfファイルに記録されています。mysqldの起動時にauto.cnfが見つからなければ、server_uuidを自動生成してauto.cnfに書き込みますので、データディレクトリを丸ごとコピーした場合は複製先のデータディレクトリのauto.cnfを削除することを忘れないでください。

次に、マスタに対して「レプリケーション用のユーザ」を作成します。マスタから見るとスレーブは通常のクライアントと同じ扱いであり、レプリケーションの開始シーケンスの中でアカウント認証とコマンドパース、権限チェックが行われます(通常のSQLではないため、実行計画の計算などは行われません)。レプリケーションスレーブとなり、Binlog Dumpスレッドからバイナリログを受け取るためにはReplication_slave権限が必要です。**図3**のコマンドは、接続元ホスト"172.17.42.1"の"repli cator"ユーザを作成し、Replication_slave権限を割り当てる例です。Replication_slave権限はグローバル権限(ON *.*で表されます)のみが存在し、データベース単位、テーブル単位で指定することはできません(権限上データベース単位やテーブル単位での指定ができないだけで、別途レプリケーション設定のパラメータを利用してデータベース単位やテーブル単位でのレプリケーションを構築することはできます。この機能は「レプリケーションフィルタ」と呼ばれます)。

マスタ上でユーザを作成したあとは、スレーブから接続をテストしてみましょう(**図4**)。CHANGE MASTER TOステートメントを実行する前にmysqlコマンドラインクライアントで認証が行われることを確認しておくことをお勧めします(**図4**のコマンド例では、マスタのIPアドレスは"172.17.1.116"としています)。

MySQLのレプリケーションは「同じデータに対し」「同じ更新ステートメントを実行すれば」「再び同じデータに戻る」という考え方をベースにした結果整合性モデルで実装されています。

そのため、レプリケーション構成をスタートする前に、マスタとスレーブの両サーバのデータを同じものにしておく必要があります。データディレクトリのコピーやmysqldumpコマンドを利用したバックアップ／リストアの手順を用いてデータをそろえます。マスタ、スレーブとも新規構築の場合はアカウント情報を除いて同じデータ(＝お互い何もデータが入っていない)状態になっていますのでそのまま手順を進める

注1) http://dev.mysql.com/doc/refman/5.6/ja/replication-options-binary-log.html#sysvar_log_bin
注2) http://dev.mysql.com/doc/refman/5.6/ja/server-system-variables.html#sysvar_server_id

▼図3 "replicator"ユーザを作成し、Replication_slave権限を割り当てる例

```
mysql> CREATE USER replicator@172.17.42.1 IDENTIFIED BY 'replication_password';
mysql> GRANT REPLICATION SLAVE ON *.* TO replicator@172.17.42.1;
```

こともできますが、アカウント情報の差異がまれにレプリケーションの停止を引き起こす(マスタにしか存在しないアカウントをIF EXISTSキーワードなしでDROP USERした場合など)場合がありますので、本番環境でレプリケーションを構築する場合ではお互い初期状態でもダンプ、リストアの手順を踏むようにしましょう。

データがそろったところで、まずはマスタのMySQLでバイナリログの情報を確認します(**図5**)。

バイナリログが無効になっている場合は、SHOW MASTER STATUSには空の結果が返ってきますので設定を確認してください。

戻ってきた結果のうち、Fileは現在アクティブなバイナリログファイル名、Positionはアクティブなバイナリログの現在のポジション(アクティブなバイナリログの先頭からのオフセットバイト数)、Binlog_Do_DBとBinlog_Ignore_DBはレプリケーションフィルタの設定項目、Executed_Gtid_Setはマスタ上で実行済のGTIDの情報を表します(本章で説明するセットアップ手順ではGTIDを利用しないため、説明は割愛します)。このうち、FileとPositionの値がレプリケーションの構成に必要です。

バイナリログの情報が得られたら、今度はスレーブにログインしてCHANGE MASTER TOステートメントを実行します(**図6**)。

master_hostにはマスタのIPアドレス(またはホスト名)を、master_portにはマスタのポート番号(明示しない場合は3306と見なされますので、ポート番号を変更していない場合は指定しなくてもかまいません)を、master_userには作成したレプリケーション用ユーザ名を、master_passwordにはレプリケーション用ユーザのパスワードを、master_log_fileには先ほど確認したマスタ上のアクティブなバイナリログファイル名を、master_log_posにはアクティブなバイナリログの現在のポジションを、それぞれ指定します。

これは「master_hostのmaster_portにmaster_userとmaster_passwordを使用して接続し、master_log_fileのmaster_log_pos以降のイベントを受信するようにレプリケーションを構築する」という意味です。

バイナリログは、マスタ上で更新がコミットされるたびに後ろに書き込まれていき(**図7**)、遡(さかのぼ)ることはありません(バイナリログファイルも連番のサフィックスがついています)。バイナリログファイルとポジションは一意であり、最初にデータを同期した時点のポジション**以降の**イベ

▼図4　スレーブからの接続テスト

```
$ mysql -h 172.17.1.116 -u replicator -p
Enter password:
mysql> SHOW GRANTS;
+-----------------------------------------------------------------------------------------------+
| Grants for replicator@172.17.42.1                                                             |
+-----------------------------------------------------------------------------------------------+
| GRANT REPLICATION SLAVE ON *.* TO 'replicator'@'172.17.42.1' IDENTIFIED BY PASSWORD <secret> |
+-----------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

▼図5　バイナリログの情報を確認

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000002 |  2035312 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

ントを**すべて**適用していくことで、最終的に同じデータがスレーブでも再現できます(データを同期する以前のイベントを受信してしまうと、スレーブですでに同期されたデータを再度作成するSQLが実行されることになりますので、マスタとスレーブの間でデータの不整合が発生し

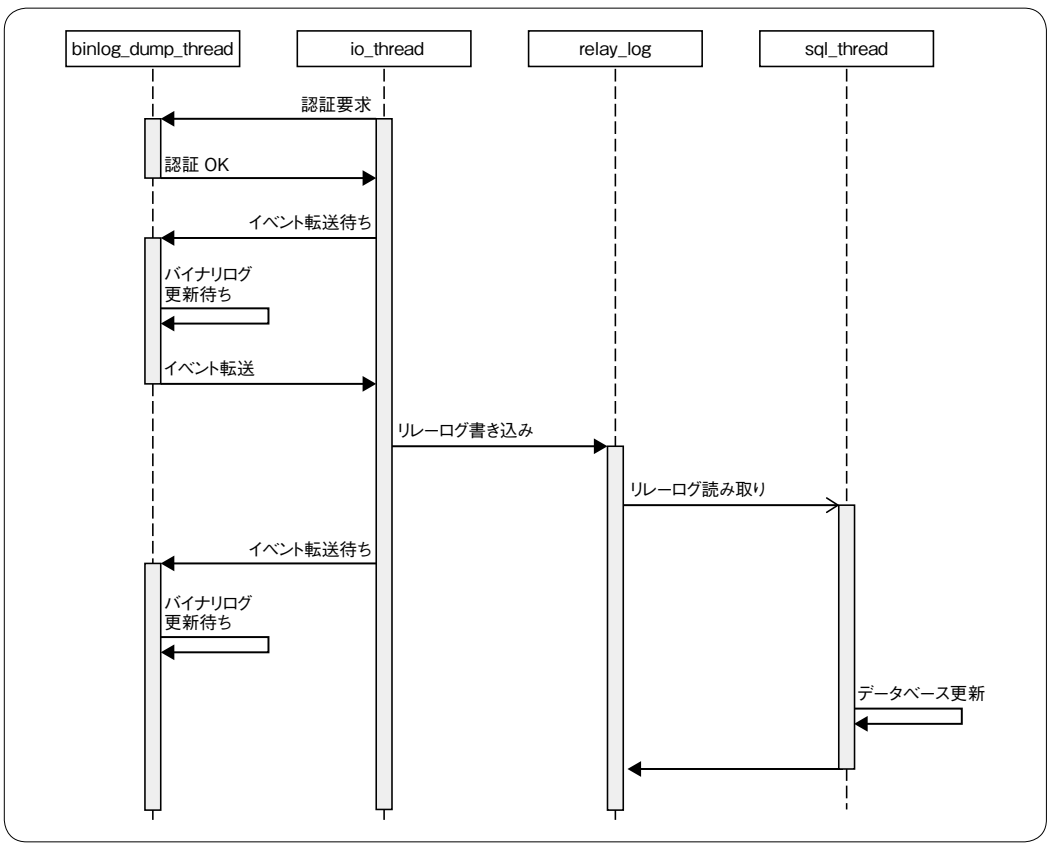
▼図6　スレーブでCHANGE MASTER TOステートメントを実行

```
mysql> CHANGE MASTER TO master_host = '172.17.1.116', master_port = 3306, master_user =
'replicator', master_password = 'replication_password', master_log_file= 'mysql-bin.000002',
master_log_pos= 2035312;
Query OK, 0 rows affected, 2 warnings (0.02 sec)

mysql> SHOW WARNINGS;
+-------+------+-----------------------------------------------------------------------------------------+
| Level | Code | Message                                                                                 |
+-------+------+-----------------------------------------------------------------------------------------+
| Note  | 1759 | Sending passwords in plain text without SSL/TLS is extremely insecure.                  |
| Note  | 1760 | Storing MySQL user name or password information in the master info repository is
not secure and is therefore not recommended. Please consider using the USER and PASSWORD
connection options for START SLAVE; see the 'START SLAVE Syntax' in the MySQL Manual for more
information. |
+-------+------+-----------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> START SLAVE;
Query OK, 0 rows affected (0.01 sec)
```

▼図7　ログの伝搬



ます。また、データを同期した以降のイベントを取り漏らしてしまうと**マスタとスレーブに同じ更新操作をすれば最終的にデータは同じになる**という前提から外れることになりますので、これもデータの不整合が発生します)。

マスタへのトラフィックをすべて停止した状態(または新規構築でありまだトラフィックが流れてこない場合)であれば間違えることはないかと思いますが、マスタが稼働中の状態でレプリケーションスレーブを構築する場合はこの点に注意する必要があります。

MySQL 5.6とそれ以降のバージョンでは、CHANGE MASTER TOステートメントにmaster_userとmaster_passwordを指定した場合、セキュリティに関連するNoteを出力するようになりました。レプリケーション構築そのものに影響することはありませんが、パスワードが平文でレプリケーション構成のファイルに記録されることが許容できない場合はNoteのメッセージに従ってSTART SLAVEステートメントでユーザ名とパスワードを指定するようにしてください。

パラメータのスコープ

代表的な2種類(コネクションをハンドルするスレッドとレプリケーション関連スレッド)の動作を紹介しましたので、次にそれらのスレッドが利用するパラメータについて紹介したいと思います。

一般的に「MySQLのパラメータ」「MySQLのオプション」と呼ばれるものはリファレンスマニュアル上ではサーバシステム変数[注3]と呼ばれています。これらのパラメータは「オプション形式」(mysqldの起動時にオプションの形で指定するもの)と「変数形式」(mysqldの起動後にSETステートメントで指定できるもの)があり、片方の形式でしか指定できないもの、両方の形式で指定できるものがあります(ただし、変数形式であっても読み取り専用のパラメータがあり、実質オプション形式しか受け付けないようなものもあります)。またパラメータごとに「グローバルのみ」(サーバ全体で1つの値を共有するもの)、「セッションのみ」(SET SESSIONステートメントでのみ設定可能)、「グローバルとセッションの両方」(単にSET GLOBALステートメントでもSET SESSIONステートメントでも指定ができるという意味では**ありません**。後ほど説明します)のスコープを持っています。

グローバルスコープとセッションスコープについて

MySQLのパラメータには3つのスコープがあり、それぞれ反映のタイミングと影響範囲が違います。

グローバルスコープしか持たないパラメータはシンプルです。SET GLOBALステートメントでパラメータを変更した時点から、MySQLに接続しているすべてのスレッドでその設定が有効になります(厳密には個々のスレッドに依存しない個所でそのパラメータが判定されている、というべきでしょうか)。たとえばSET GLOBAL innodb_flush_log_at_trx_commit= 0というステートメントを実行した場合、SETステートメントの完了以降すべての処理がinnodb_flush_log_at_trx_commit = 0として振る舞います。ほかのスレッドがトランザクションの途中であろうが、クエリ処理の真っ最中であろうが、ただ1つの値が使われます。

セッションスコープしか持たないパラメータも同様にシンプルです。SET SESSIONステートメントでパラメータを変更した時点から、そのセッションでのみ変更されたパラメータが有効になります。セッションスコープしか持たないパラメータは多くなく(timestamp、last_insert_idなどいくつかありますが)使う機会は少ないでしょう。

そしてMySQLのパラメータの多くを占めるグローバルスコープとセッションスコープの両

注3) http://dev.mysql.com/doc/refman/5.6/ja/server-system-variables.html

方を持つパラメータは、

・コネクションの確立時にグローバルスコープからスレッドごとのセッションスコープに値がコピーされる
・個々のスレッドの振る舞いはスレッドがすでに保持しているセッションスコープのパラメータに支配される

という動作をします。この動作により、SET SESSIONステートメントはグローバルスコープの値にかかわらずセッション単位でパラメータを上書きできますが、SET GLOBALステートメントは**すでに接続済みのスレッドには反映されない**という点に注意が必要です。アプリケーションの処理ごとにMySQLとのコネクションを張りなおすタイプのアプリケーションはSET GLOBALステートメントが(おそらく多くの人が)期待するとおりに動きますが、コネクションプーリングを利用している場合はそれらのセッションを一度解放して再接続しなければいけません。

グローバルスコープとセッションスコープの動作の違い

例を見てみましょう。あらかじめ接続されたconn1というコネクションとconn2というコネクションがあります。変数形式のパラメータは、SELECTステートメントで変数としてアクセスできます。これらにはglobalとsessionという名前空間が割り当てられており、グローバルスコープの値とセッションスコープの値を参照できます(**図8**)。

SET SESSIONステートメントでconn1のセッションスコープの変数を変更しました。これにより、conn1のソートバッファのサイズは1MBが実効値になりますが、conn2のソートバッファのサイズはデフォルトの256kBのままです(**図8**)。

次にconn2でSET GLOBALステートメントを利用して(グローバルスコープのパラメータのセットには"SUPER"権限が必要になります)グローバルスコープのソートバッファサイズを128kBに変更しました。conn2からもconn1からも@@global.sort_buffer_size変数は設定変更後の値が見えますが、実効値である@@session.sort_buffer_size変数はconn2が256kB、conn1が1MBと、SET GLOBALステートメント前と変更がありません(**図9**)。

SET GLOBALステートメントの実行後に接続してきた新しいconn3のコネクションは、@@session.sort_buffer_size変数が@@global.sort_buff

▼図8　グローバルスコープの値とセッションスコープの値

```
conn1> SELECT @@global.sort_buffer_size, @@session.sort_buffer_size;
+---------------------------+----------------------------+
| @@global.sort_buffer_size | @@session.sort_buffer_size |
+---------------------------+----------------------------+
|                    262144 |                     262144 |
+---------------------------+----------------------------+
1 row in set (0.00 sec)

conn1> SET SESSION sort_buffer_size= 1 * 1024 * 1024;
Query OK, 0 rows affected (0.00 sec)

conn1> SELECT @@global.sort_buffer_size, @@session.sort_buffer_size;
+---------------------------+----------------------------+
| @@global.sort_buffer_size | @@session.sort_buffer_size |
+---------------------------+----------------------------+
|                    262144 |                    1048576 |
+---------------------------+----------------------------+
1 row in set (0.00 sec)

conn2> SELECT @@global.sort_buffer_size, @@session.sort_buffer_size;
+---------------------------+----------------------------+
| @@global.sort_buffer_size | @@session.sort_buffer_size |
+---------------------------+----------------------------+
|                    262144 |                     262144 |
+---------------------------+----------------------------+
1 row in set (0.00 sec)
```

er_size変数と同じになります。また、接続済みのコネクションでも明示的にdefaultキーワードを使ってSET SESSIONステートメントを利用することでグローバルスコープのパラメータから再度値をコピーできますが、あまり一般的な方法ではなく、アプリケーションサーバを再起動する方法がよく利用されます(図10)。

まとめ

本章ではMySQLのベースアーキテクチャであるシングルプロセスマルチスレッドモデルの代表的なスレッドの動作と、パラメータのスコープについての説明をしました。

サーバサイドアプリケーションのエンジニアとなるとSQLの向こう側はブラックボックスとなりがちですが(そのための抽象化レイヤとしてSQLが存在しているので、このこと自体は悪いことではありません)、MySQLの基本的なしくみを知ることで、エンジニアリングの世界が広がることの助けになれば幸いです。SD

▼図9 グローバルスコープを変更しても、セッションスコープには影響はない

```
conn2> SET GLOBAL sort_buffer_size= 128 * 1024;
Query OK, 0 rows affected (0.00 sec)

conn2> SELECT @@global.sort_buffer_size, @@session.sort_buffer_size;
+---------------------------+----------------------------+
| @@global.sort_buffer_size | @@session.sort_buffer_size |
+---------------------------+----------------------------+
|                    131072 |                     262144 |
+---------------------------+----------------------------+
1 row in set (0.00 sec)

conn1> SELECT @@global.sort_buffer_size, @@session.sort_buffer_size;
+---------------------------+----------------------------+
| @@global.sort_buffer_size | @@session.sort_buffer_size |
+---------------------------+----------------------------+
|                    131072 |                    1048576 |
+---------------------------+----------------------------+
1 row in set (0.00 sec)
```

▼図10 SET SESSION sort_buffer_size = defaultでセッションパラメータの値にグローバルパラメータの値が反映される

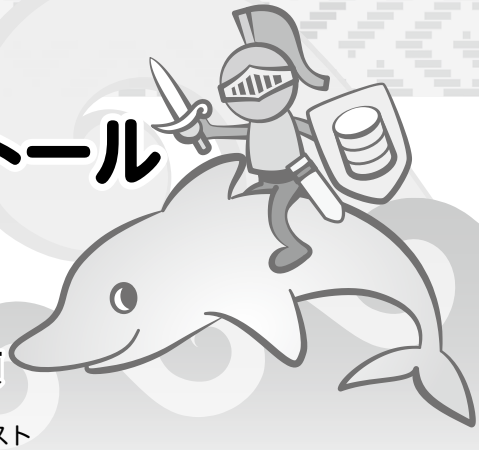
```
conn3> SELECT @@global.sort_buffer_size, @@session.sort_buffer_size;
+---------------------------+----------------------------+
| @@global.sort_buffer_size | @@session.sort_buffer_size |
+---------------------------+----------------------------+
|                    131072 |                     131072 |
+---------------------------+----------------------------+
1 row in set (0.00 sec)

conn1> SET SESSION sort_buffer_size = default;
Query OK, 0 rows affected (0.00 sec)

conn1> SELECT @@global.sort_buffer_size, @@session.sort_buffer_size;
+---------------------------+----------------------------+
| @@global.sort_buffer_size | @@session.sort_buffer_size |
+---------------------------+----------------------------+
|                    131072 |                     131072 |
+---------------------------+----------------------------+
1 row in set (0.00 sec)
```

MySQLをインストールしてみよう

RHEL、Ubuntu、Debian、Windows、Mac OSにおける手順



この章ではMySQLを各RHEL系、Ubuntu、Debianなどのディストリビューション、WindowsやMac OSにインストールする方法について解説します。また、MySQLのメジャーバージョン以外のインストール方法、バイナリパッケージの利用についても触れます。

Author yoku0825、kk2170、hito_asa

MySQLをインストールするにあたって

2016年3月現在、MySQLの最新リリースは2016年2月にリリースされたMySQL 5.7.11です。MySQLには最新版以外にもメンテナンスが継続されるリリースが存在します。まずはどのMySQLのリリースをインストールするかを決めるためにも、MySQLのバージョン体系について知っておきましょう。

MySQLのバージョンについて

MySQL x.y.zと書いた場合、xがメジャーバージョン番号、yがリリースレベル、x.yが1つのリリース系列となり、zがリリースシリーズ内でのバージョン番号になるとされています[注1]。たとえばMySQL 5.7.11はメジャーバージョン5、リリースレベル7、5.7系列の11番めのバージョンという意味です。

しかし実際のところ、互換性の単位やドキュメントの単位としてほぼx.yの「リリース系列」が単位となることから、開発元のOracleで開催されるセミナーなどで語られる際は、リリース系列とされているx.yが(メジャー)バージョン、zがマイナーバージョンとして呼ばれることが多くなっています。筆者も慣習的に「ドキュメント上のリリース系列」=「(メジャー)バージョン」、「リリース系列内でのバージョン番号」=「マイナーバージョン」と呼んでいますので、今回の説明の中でもその呼び方を使いたいと思います。

メジャーバージョンをまたぐバージョンアップをメジャーバージョンアップ、同一メジャーバージョン内でマイナーバージョンのみ変更されるバージョンアップをマイナーバージョンアップと呼びます[注2]。

互換性の単位について

先ほど「互換性の単位」という言い方を用いましたが、ここでいう互換性とはmysql_upgradeコマンドを用いたアップグレードの後方互換性のことを意味します(ただし、後ほど説明しますが、GA(General Availability)版以前のマイナーバージョンではこれらは保証されません)。

MySQLではメジャーバージョン(リリース系列)ごとにmysqlスキーマを始めとするシステム用のスキーマに含まれるテーブルの構造に変化があり、これを解消する(古い形式のシステム用スキーマから新しいシステム用のスキーマに変換する)ためにmysql_upgradeというコマンドが同梱されています。このコマンドを利用したアップグレードは原則「同じ、あるいは1つ前のバージョンのシステム用スキーマを、現在のバー

注1) ドキュメント上の記載は、https://dev.mysql.com/doc/refman/5.6/ja/which-version.htmlです。

注2) 余談ですが、たとえば「MySQL 5.6」は「マイエスキューエル ごーてんろく」と呼ばれることが多く、たまに「マイエスキューエル ごーろく」、英語圏に行くと「マイエスキューエル ファイブシックス」と呼ばれることが多いようです。

ジョンのシステム用スキーマに変更する」ことを意図して設計されていたため、2つ以上のバージョンをまたいだmysql_upgradeは失敗することがあります(成功することもあります。筆者が試した限りでは、MySQL 5.1.73からMySQL 5.6.29へのメジャーバージョンアップはmysql_upgradeコマンドで実施可能、MySQL 5.0.96からMySQL 5.6.29へのメジャーバージョンアップは失敗(MySQL 5.6がクラッシュする)でした。ドキュメント上の記載も「not recommended or supported(推奨されない、またはサポートされない)」となっています[注3])。

また、大きな機能追加や機能の廃止、SQL構文の変更(まれです)などは原則メジャーバージョン単位で行われます。

バージョンのステータスについて

MySQLのメジャーバージョンは、大きく分けて3つの状態に分けられます。

「サポートが終了したバージョン」「サポートされているバージョン」「開発が進められているバージョン」の3つです。2016年3月現在、MySQL 5.5とそれより以前のバージョンは「サポートが終了したバージョン」で、これ以上マイナーバージョンのリリースはありません。MySQL 5.6、5.7が「サポートされているバージョン」で、バグフィックスなどによる新しいマイナーバージョンが提供されることが期待できます。MySQL 5.7が製品リリースされたため現在は「開発が進められているバージョン」はありませんが、このままの予定でいけば、いずれMySQL 5.8が次の「開発が進められているバージョン」としてリリースされるのではないでしょうか(公式にはMySQL 5.8という名前はまだ出てきていませんが、バグレポートなどでMySQL 5.8の片鱗が見え始めています)。

また、「開発が進められているバージョン」には別の形態もあり、"lab"版、「実験室」版などと呼ばれています。これはMySQL Labs[注4]にて配布されている文字どおり「実験中」のリリースで、「本番環境での使用はやめてほしい」("Please, DO NOT USE THESE BINARIES IN PRODUCTION")とダウンロードページに記載があります(**図1**)。これらは今後の「開発が進められているバージョン」に取り入れられる可能性のある新機能を試すためのリリースであり、開発中のバージョンのツリーとは独立して開発が進められています。

また、MySQLはマイナーバージョンによっても大きく3つの状態に分けられます。

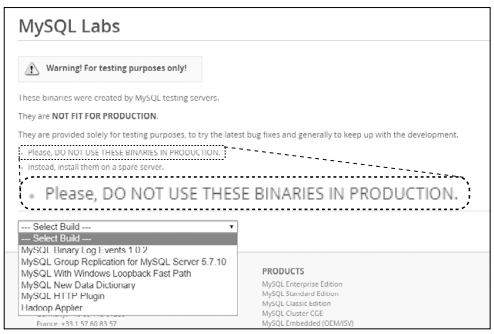
DMR("Development Milestone Release"、かつては「ベータ版」と名前が付けられていました)版、RC("Release Candidate"、「リリース候補」の名のとおり、DMRとGAの間に位置します。かつて「ガンマ版」と呼ばれていた時期もあったようですが、あまり流行らなかったようです)版、GA("General Availability"、かつては"Production"版とも名前が付けられていました。MySQLでは"stable"版という言い方は耳にしませんが、ほかのプロダクトで言う"stable"版に相当するのがこれです)版です。

各メジャーバージョンの最後のマイナーバージョンのステータスをもって、メジャーバージョン全体のステータスを呼ぶことも多々あります(たとえばMySQL 5.7のGA前の2015年9月時

注3) MySQLのメジャーバージョンアップに関するドキュメント上の記載は、https://dev.mysql.com/doc/refman/5.6/ja/upgrading.htmlです。

注4) http://labs.mysql.com/

▼図1 "Please, DO NOT USE THESE BINARIES IN PRODUCTION"の記載があるMySQL Labs



点で言えば、MySQL 5.5(5.5.45-GA)と5.6(5.6.26-GA)がGA、MySQL 5.7(5.7.8-rc)はRCでした。GAの中でも最新のものを"GA"、最新でないGAは"Previous GA"などと呼び分ける場合もあります)が、「サポートが終了したバージョン」に関してはGAとは呼ばれなくなります。

入門者が選ぶべきMySQLのバージョン

現在サポートされているバージョンのMySQLはMySQL 5.6とMySQL 5.7です。本来であれば、最新のMySQLであるMySQL 5.7の利用を勧めたいところなのですが、勉強用に初めて導入するMySQLということであれば、"Previous GA"であるMySQL 5.6をお勧めします。

これは、MySQLのドキュメントはリリース系列ごとに分冊されており(MySQL Documentation[注5]のページを参照してください)、2016年3月現在では日本語訳が存在するのはMySQL 5.6のみとなっていることが大きな理由です。ちなみに、2016年3月現在、アーカイブ以外で現在公開されているMySQLの公式ドキュメントとしてはMySQL 5.6の日本語版が**唯一の**英語以外のドキュメントです。入門者向けという意味では、最新に近い(日本語ドキュメントがリリースされた当時は最新でした)バージョンの母国語訳のドキュメントが存在するのはありがたいことです。

もう一つの消極的な理由は、MySQL 5.6からMySQL 5.7にかけては150を超える新機能が追加[注6]されており、ユーザによって公開されたWeb上の情報の多くはまだそれに対応していません。新機能のみならず変更された仕様も存在するため、入門用としては情報が多いバージョンの方がよいでしょう。

このため、本章で説明するインストールの手順はすべて2016/03現在のMySQL 5.6系列の最新版であるMySQL 5.6.29をベースに説明します。

注5) http://dev.mysql.com/doc/
注6) https://yakst.com/ja/posts/3037

インストール

MySQLのインストールの方法はいくつかありますが、ここでは次の、4つのOSとLinux汎用バイナリパッケージ、CentOS上でのソースコードからのコンパイルについて説明します。

①RHEL(Red Hat Enterprise Linux)およびその互換OS(CentOS、Oracle Linuxなど)
②Ubuntu、Debian
③Windows(Windows 10)
④Mac OS X
⑤Linux用の汎用バイナリパッケージ(.tar.gzパッケージ)を使用したインストール
⑥ソースコードからコンパイル(CentOS)

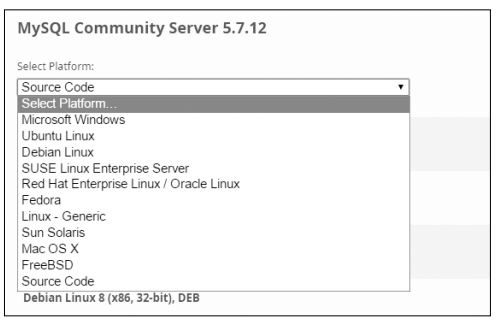
ここで紹介するプラットフォーム以外にも、MySQLがサポートしているプラットフォームはいくつかあります(**図2**)。

ダウンロードページ[注7]からダウンロードできるのはその時点での最新シリーズの最新ビルドのみです。今回のように前世代のGAをダウンロードする場合は、"Looking for previous GA versions?"のリンクから、ダウンロード対象のメジャーバージョンを選択します(**図3**)。

また、最新ビルド以外のマイナーバージョンをダウンロードする場合は、ページ上ほどにある"Archives"のリンクからアーカイブを探すことができます(**図4**)。

注7) http://dev.mysql.com/downloads/mysql/

▼図2 ダウンロードページからダウンロードできるプラットフォーム別のパッケージ



インストールの手順はMySQLのリファレンスマニュアル注8にも記載がありますので、そのほかのプラットフォームでのインストールや、詳細にインストールの内容を確認したい場合は合わせて参照してください。

RHELおよびその互換OS (CentOS、Oracle Linuxなど)

RHELおよびその互換OSに特有のインストール方法としては、次の2つがあります。

①yumリポジトリを使用したインストール
②rpmパッケージを使用したインストール

どちらもほぼ同じ構成でインストールされますが、MySQL 5.6まではyumコマンドでインストールする際とrpmファイルをダウンロードしてきてインストールする際でパッケージの名前やサービス名などが一部違いますので、別としてカウントしてあります(MySQL 5.7からはこの差異はなくなります)。

どちらの方法を用いてインストールしても、一般的に利用するぶんには提供されるMySQLの機能はほぼ変わりません。以前はMySQL公式のyumリポジトリが存在しなかったため、yumコマンドでMySQLをインストールしようとするとどうしても古いバージョンのものやサードパーティービルドのものになってしまう問題がありましたが、現在はその問題を気にする必要もありません(2016年3月現在、MySQL 5.5、MySQL 5.6、MySQL 5.7のバージョンをyumリポジトリからインストールできます)。

注8) http://dev.mysql.com/doc/refman/5.6/ja/installing.html

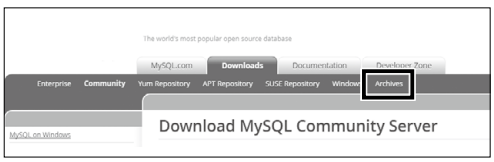
yumリポジトリを使用したインストール

yumリポジトリを利用する場合はインストール、アップグレードともにとても簡単ですが、インストールされるパスが固定されている(たとえば、mysqldは/usr/sbin/mysqldにインストールされます)ため、1台のサーバに複数のMySQLサーバを同居させることはできません。また、ほかのパッケージのアップグレードのために頻繁にyum upgradeを実行する機会がある場合、MySQLサーバも同時にマイナーバージョンアップされてしまうことがあるため(ただし、メジャーバージョンアップはされないように、メジャーバージョンごとに違うリポジトリ名が割り当てられています)、それが許されない場合はインストール後に自身でMySQLのリポジトリを無効化しておく必要があります。

図5はCentOS 6.x上でMySQLのyumリポジトリをインストールし、最新のMySQLサーバをインストールするコマンドの例です(URLなどは2016年3月現在のもので、今後変更になる可能性があります)。MySQLのyumリポジトリのダウンロード先はhttps://dev.mysql.com/downloads/repo/yum/から確認できます。

パッケージファイル名が示すとおり、このyumリポジトリはデフォルトでMySQL 5.7をインストールするように設定されています。今回は練習用に、あえて1世代前のMySQL 5.6をインストールしますので、/etc/yum.repos.d/mysql-community.

▼図4 マイナーバージョンをダウンロードする場合



▼図3 前世代のGAをダウンロードする場合

repoを編集して(**図6**)mysql57-communityリポジトリを無効化、mysql56-communityリポジトリを有効化します(MySQL 5.7でチャレンジしたい方はそのままでOKです)。

編集が終わったらyum installでインストールします(**図7**)。Repositoryがmysql56-community、Versionが5.6.xになっていることを確認してください。

インストールが終わったらMySQLを起動します。CentOS 6.xの場合はserviceコマンド、CentOS 7.xの場合はsystemctlコマンドで起動できます。MySQLをインストール後、初めて起動する際に、データベースの初期化(MySQL 5.6ではmysql_install_db、MySQL 5.7ではmysqld --initialize)が実行されます。

```
$ sudo service mysqld start
..
                                           [  OK  ]
Starting mysqld:                           [  OK  ]
```

rpmパッケージを使用したインストール

rpmパッケージを利用する場合もyumリポジトリを利用する場合とほぼ同等ですが、wgetでファイルをダウンロード後に(または、yumやrpmコマンドで直接ダウンロード先のURLを指定して)インストールすることになるため、設定ファイルを編集しなくとも自動でマイナーバージョンアップされてしまう問題はありません。また、本番環境が直接外部のyumリポジトリと通信できないように遮断されている場合でも利用できます。

▼図5　CentOS 6.x上で最新のMySQLのリポジトリをインストールする

```
$ sudo yum install http://dev.mysql.com/get/mysql57-community-release-el6-7.noarch.rpm
..
================================================================================
 Package                    Arch    Version  Repository                     Size
================================================================================
Installing:
 mysql57-community-release  noarch  el6-7    /mysql57-community-release-el6-7.noarch  7.8 k

Transaction Summary
================================================================================
Install       1 Package(s)

Total size: 7.8 k
Installed size: 7.8 k
..
```

▼図6　MySQL 5.6をインストールするように変更する

```
$ sudo vim /etc/yum.repos.d/mysql-community.repo
..
# Enable to use MySQL 5.6
[mysql56-community]
name=MySQL 5.6 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.6-community/el/6/$basearch/
enabled=1    ← 0から1に変更
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql

[mysql57-community]
name=MySQL 5.7 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/6/$basearch/
enabled=0    ← 1から0に変更
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

MySQL 5.6までは、rpmパッケージを利用した場合とyumリポジトリを利用した場合ではパッケージ名や構成に若干の違いがあります(**表1**)。注意するべきはyumリポジトリではmysql-community-libs(および-compat)だったものが、MySQL-shared(および-compat)になっていることくらいでしょうか。ほかはそれほど違いはありません。MySQL 5.7でこれらはyumリポジトリの命名規則に統一されました。

rpmパッケージの取得はhttp://dev.mysql.com/downloads/mysql/から行います。MySQL 5.6とそれ以前は「Red Hat Enterprise Linux / Oracle Linux」と「Linux - Generic」それぞれにrpmファイルがありCentOSではどちらも利用できましたが、MySQL 5.7ではrpmファイルは「Red Hat Enterprise Linux / Oracle Linux」からのみダウンロードできるようになっています(筆者の試した限りでは、問題なくインストールできました)。

ここでは「Linux - Generic」からMySQL 5.6用のrpmファイルをダウンロードしてインストールする手順を紹介します(**図8**)。

▼図7 MySQL 5.6のインストール

```
$ sudo yum install mysql-community-server
..
================================================================================
 Package                       Arch    Version         Repository          Size
================================================================================
Installing:
 mysql-community-server        x86_64  5.6.29-2.el6    mysql56-community   53 M
Installing for dependencies:
 libaio                        x86_64  0.3.107-10.el6  base                21 k
 mysql-community-client        x86_64  5.6.29-2.el6    mysql56-community   18 M
 mysql-community-common        x86_64  5.6.29-2.el6    mysql56-community  308 k
 mysql-community-libs          x86_64  5.6.29-2.el6    mysql56-community  1.9 M
 numactl                       x86_64  2.0.9-2.el6     base                74 k
 perl-DBI                      x86_64  1.609-4.el6     base               705 k

Transaction Summary
================================================================================
Install       7 Package(s)

Total size: 74 M
Total download size: 74 M
Installed size: 330 M
..
```

▼図8 「Linux - Generic」からMySQL 5.6用のrpmファイルによってインストールする

```
$ wget http://dev.mysql.com/get/Downloads/MySQL-5.6/MySQL-5.6.29-1.linux_glibc2.5.x86_64.rpm-bundle.tar

$ tar xvf MySQL-5.6.29-1.linux_glibc2.5.x86_64.rpm-bundle.tar
MySQL-embedded-5.6.29-1.linux_glibc2.5.x86_64.rpm
MySQL-test-5.6.29-1.linux_glibc2.5.x86_64.rpm
MySQL-server-5.6.29-1.linux_glibc2.5.x86_64.rpm
MySQL-devel-5.6.29-1.linux_glibc2.5.x86_64.rpm
MySQL-shared-5.6.29-1.linux_glibc2.5.x86_64.rpm
MySQL-shared-compat-5.6.29-1.linux_glibc2.5.x86_64.rpm
MySQL-client-5.6.29-1.linux_glibc2.5.x86_64.rpm

$ sudo rpm -i MySQL-*.rpm
$ sudo service mysql start
```

・「Linux - Generic」プラットフォームを選択(図9)
・「RPM Bundle」パッケージを選択(図10)
・「Download」をクリックするとOracle Web Accountでのログインまたは登録を求めるページに遷移するが(図11)、「No thanks, just start download」のリンクをクリックすることで、登録不要でダウンロードできる

OSインストール時などに"mysql-libs"に依存するパッケージをインストールしている場合、"MySQL-shared"と競合してしまいインストー

▼表1　rpmパッケージを利用した場合とyumリポジトリを利用したときの差異

	rpmパッケージ 5.6とそれ以前	yumリポジトリ 5.6とそれ以前	5.7 rpmパッケージ、yumリポジトリ共通	CentOS 6.6 Base リポジトリ (MySQL 5.1.73)
MySQLサーバ (mysqld)	MySQL-server	mysql-community-server	mysql-community-server	mysql-server
MySQLコマンドラインクライアント (mysqlなど)	MySQL-client	mysql-community-client	mysql-community-client	mysql
MySQLライブラリ (libmysqlclient.soなど)	MySQL-shared	mysql-community-libs	mysql-community-libs	mysql-libs
旧バージョンのMySQLライブラリ	MySQL-shared-compat	mysql-community-libs-compat	mysql-community-libs-compat	なし
MySQLヘッダファイル(mysql.hなど)	MySQL-devel	mysql-community-devel	mysql-community-devel	mysql-devel
MySQL単体テストスイート(mysql_test_run.plなど)	MySQL-test	mysql-community-test	mysql-community-test	mysql-test
サービス名 (serviceコマンドで参照する名前)	mysql	mysqld	mysqld	mysqld
データディレクトリの初期化	mysql_install_db インストール時	mysql_install_db service mysqld start時	mysqld --initialize service mysqld start時	mysql_install_db service mysqld start時
エラーログファイル名	/var/lib/mysql/ホスト名.err	/var/log/mysqld.log	/var/log/mysqld.log	/var/log/mysqld.log
初期ユーザ	root@localhostなどrootのみ	root@localhostなどと匿名ユーザ	root@localhostのみ	root@localhostなどと匿名ユーザ
rootアカウントの仮パスワード	/root/.mysql_secret	生成しない	/var/log/mysqld.log	実装なし
validate_passwordプラグイン	バンドルされているが有効化されていない	バンドルされているが有効化されていない	有効化されている 英大文字小文字数字記号の4種を含んだ8文字以上	実装なし
ユーザ作成	GRANTステートメント	GRANTステートメント	CREATE USERステートメント パスワード設定時はGRANTステートメントでワーニング(作成可) パスワード未設定時はGRANTステートメントがエラー(作成不可)	GRANTステートメント
LOAD DATA INFILEに対する制限 (secure_file_privオプション)	なし	なし	/var/lib/mysql-filesディレクトリの中のみ	なし

ルできません。この場合は、“MySQL-shared-compat”をインストールしてから“mysql-libs”を削除、あらためてそれ以外のパッケージをインストールしてください(依存関係は“MySQL-shared-compat”が受け継ぐため、エラーになることはないはずです)。

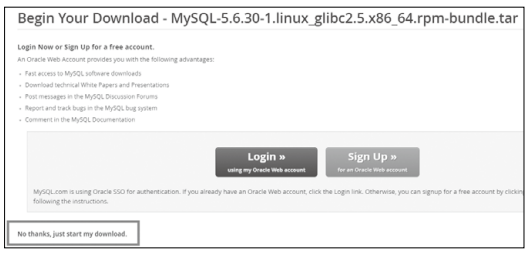
インストールが終わったらMySQLを起動します。CentOS 6.xの場合はserviceコマンド、CentOS 7.xの場合はsystemctlコマンドで起動できます。rpmパッケージ版のサービス名は“mysql”で、yumリポジトリ版と違いdがつきません。シェルの補完機能を使っている分にはそれほど違いを意識することはありませんが、ChefやVagrantなどでプロビジョニングスクリプトを書くときには気をつけましょう。

rpmパッケージでインストールした場合は、MySQL-serverのインストール時にデータベースの初期化処理が実行されます。

▼図9　プラットフォーム選択

▼図10　RPM Bundleパッケージの選択

▼図11　かの有名なNo Thanks



Ubuntu、Debian

Ubuntu、Debianおよびその派生ディストリビューションのOSに特有のインストール方法としては次の2つがあります。

①aptリポジトリを使用したインストール
②debパッケージを使用したインストール

これらはインストール方法や管理方法が異なるだけで、インストールされるパッケージは同じものになります。MySQL 5.6/5.7の最新のマイナーバージョンがインストールできればよい、という場合はaptリポジトリを使用したインストールをお勧めします。それ以外のバージョンを指定してインストールしたい場合や、ほかのパッケージのアップグレード時にMySQLのマイナーバージョンがアップグレードされてしまうのを避けたい場合はdebパッケージを使用したインストールをお勧めします。

aptリポジトリを使用したインストール

MySQL公式のaptリポジトリを使用することで、とても簡単にインストールできます。ただし、yumリポジトリを使用したインストールと同様に、ほかのパッケージをアップグレードするための作業によってMySQLのマイナーバージョンもアップデートされてしまう可能性があります。それが許されない場合はインストール後に`dpkg --set-selections`でバージョンを固定する必要があります。

図12はMySQL公式のaptリポジトリをインストールし、最新のMySQLサーバをインストールするコマンドの例です(URLなどは2016年3月現在のもので、今後変更になる可能性があります)。MySQLのaptリポジトリのダウンロード先は、https://dev.mysql.com/downloads/repo/apt/から確認できます。この手順はUbuntu 15.10で実行した際のものですが、Ubuntu 14.04 LTSおよびDebian 8でも同様です。

ここでインストールする対象を選択できます（**図13**）。1番のMySQL Serverを選択するため1↵と入力します。

続いてバージョンを確認されるので、1番のmysql-5.6を選択するため1↵と入力します（**図14**）。

▼図12　MySQL公式のaptリポジトリからMySQLサーバをインストールする

```
$ wget https://dev.mysql.com/get/mysql-apt-config_0.7.2-1_all.deb
... 'mysql-apt-config_0.7.2-1_all.deb' saved [19006/19006]
$ sudo dpkg -i mysql-apt-config_0.7.2-1_all.deb
Selecting previously unselected package mysql-apt-config.
(Reading database ... 13964 files and directories currently installed.)
Preparing to unpack mysql-apt-config_0.7.2-1_all.deb ...
Unpacking mysql-apt-config (0.7.2-1) ...
Setting up mysql-apt-config (0.7.2-1) ...
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot ➥
be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 76.)
debconf: falling back to frontend: Readline
Configuring mysql-apt-config
---------------------------

MySQL APT Repo features MySQL Server along with a variety of MySQL components. You may select
the appropriate product to choose the version that you wish to receive.

Once you are satisfied with the configuration then select last option 'Apply' to save the
configuration. Advanced users can always change the configurations later, depending
on their own needs.

  1. MySQL Server (Currently selected: mysql-5.7)  2. MySQL Tools & Connectors (Currently
selected: Enabled)  3. MySQL Preview Packages (Currently selected: Disabled)  4. Ok
Which MySQL product do you wish to configure?
```

▼図13　インストール対象を選択

```
Which MySQL product do you wish to configure? 1↵

This configuration program has determined that no MySQL Server is installed on your ➥
system, and has highlighted the most appropriate repository package. If you are not sure
which version to install, do not change the auto-selected version. Advanced users can ➥
always change the version as needed later.

  1. mysql-5.6  2. mysql-5.7  3. None
Which server version do you wish to receive?
```

▼図14　バージョンの確認

```
Which server version do you wish to receive? 1↵

MySQL APT Repo features MySQL Server along with a variety of MySQL components. You may ➥
select the appropriate product to choose the version that you wish to receive.

Once you are satisfied with the configuration then select last option 'Apply' to save the ➥
configuration. Advanced users can always change the configurations later, depending
on their own needs.

  1. MySQL Server (Currently selected: mysql-5.6)  2. MySQL Tools & Connectors (Currently ➥
selected: Enabled)  3. MySQL Preview Packages (Currently selected: Disabled)  4. Ok
Which MySQL product do you wish to configure?
```

再度インストールする対象を選択するプロンプトに戻るので、MySQL Server (Currently selected: mysql-5.6)となっていることを確認し(**図14**)、4↵と入力します(**図15**)。

これでaptリポジトリのインストールが完了しました。続いてapt-getコマンドでMySQLをインストールします(**図16**)。

ここでMySQLのrootユーザに設定するパスワードを要求されるので、任意のパスワードと↵を確認も含めて2回入力します。これでインストールは完了です。

インストールが終わったらserviceコマンドでMySQLを起動します。名前はmysqldではなくmysqlですので注意してください。

▼図15　確認画面(図14からの続き)

```
Which MySQL product do you wish to ➡
configure? 4↵

OK
```

▼図16　apt-getでMySQLをインストール

```
$ sudo apt-get update
...
Reading package lists... Done
$ sudo apt-get install -y mysql-community-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  apparmor busybox-initramfs cpio init-system-helpers initramfs-tools initramfs-tools-bin ➡
klibc-utils kmod libaio1 libapparmor-perl libklibc libnuma1 mysql-client
  mysql-common mysql-community-client psmisc
Suggested packages:
  apparmor-profiles apparmor-profiles-extra apparmor-docs apparmor-utils libarchive1 ➡
bash-completion
The following NEW packages will be installed:
  apparmor busybox-initramfs cpio init-system-helpers initramfs-tools initramfs-tools-bin ➡
klibc-utils kmod libaio1 libapparmor-perl libklibc libnuma1 mysql-client
  mysql-common mysql-community-client mysql-community-server psmisc
0 upgraded, 17 newly installed, 0 to remove and 18 not upgraded.
Need to get 24.7 MB of archives.
After this operation, 169 MB of additional disk space will be used.
...
---------------------------------

Data directory found when no MySQL server package is installed

A data directory '/var/lib/mysql' is present on this system when no MySQL server package ➡
is currently installed on the system. The directory may be under control of server
package received from third-party vendors. It may also be an unclaimed data directory ➡
from previous removal of mysql packages.

It is highly recommended to take data backup. If you have not done so, now would be the ➡
time to take backup in another shell. Once completed, press 'Ok' to continue.

Please provide a strong password that will be set for the root account of your MySQL ➡
database. Leave it blank if you do not wish to set or change the root password at this
time.

Enter root password:

Now that you have selected a password for the root account, please confirm by typing it ➡
again. Do not share the password with anyone.

Re-enter root password:
```

```
$ sudo service mysql start
......
 * MySQL Community Server 5.6.29 is started
```

debパッケージを使用したインストール

debパッケージを使用してインストールする場合、事前に依存するパッケージがインストールされている必要があります。

MySQL 5.6のdebパッケージの取得はhttp://dev.mysql.com/downloads/mysql/5.6.htmlから行います。ここでは「Ubuntu Linux」からMySQL 5.6用のdebファイルをダウンロードしてインストールする手順を紹介します。

Select Platformから「Ubuntu Linux」を選択(**図17**)します。Debianの場合は「Debian Linux」を選択します。

「Download」をクリックすると(**図18**)Oracle Web Accountでのログインまたは登録を求めるページに遷移しますが、「No thanks, just start download」のリンクをクリックすることで、登録不要でダウンロードできます(rpmパッケージのダウンロードと同様です)。コマンドラインでダウンロードすることもできます(**図19**)。

ダウンロードが完了したらtarファイルを展開し、debパッケージをすべてインストールします(**図20**)。

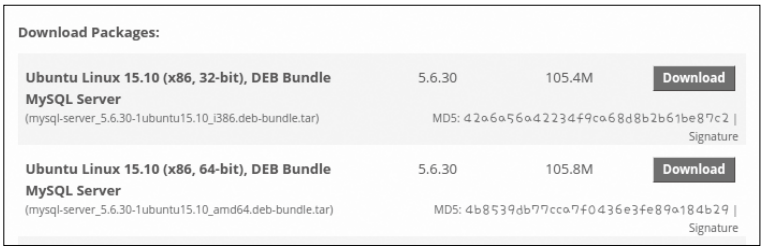
aptリポジトリからのインストール時と同様、インストール中にrootユーザのパスワードを確認されるので任意のパスワードを入力してください。またインストール中に**図21**のようなエラーが出た場合は、必要な依存パッケージがインストールされていないため、`apt-get -f install`を実行して不足しているパッケージをインストールしてください。

以上でインストールは完了です。インストールが終わったらserviceコマンドでMySQLを起動します(**図22**)。

▼図17 「Ubuntu Linux」を選択

※Download Packagesのリストから利用している環境に合わせた「DEB Bundle MySQL Server」パッケージを選択

▼図18 「DEB Bundle MySQL Server」パッケージを選択



※ここではUbuntu 15.10のx86 64 bit環境を利用している前提で「Ubuntu Linux 15.10 (x86, 64-bit), DEB Bundle MySQL Server」を選択

▼図19 コマンドラインでのダウンロード

```
$ wget http://dev.mysql.com/get/Downloads/MySQL-5.6/mysql-server_5.6.29-1ubuntu15.10_amd64.deb-bundle.tar
... 'mysql-server_5.6.29-1ubuntu15.10_amd64.deb-bundle.tar' saved [110888960/110888960]
```

Windows(Windows 10)

WindowsへのMySQLのインストール方法としては次の2つがあります。

①MySQL Installer
②zipパッケージ

MySQL Installerを使用したインストール

少しややこしいのですが、"MySQL Installer"とはMySQLのインストーラではなく、"MySQL Installer"という製品を指します(MySQL Installerの実態はMySQLと周辺製品のインストーラですので、特段気にすることはないかもしれませんが)。MySQL InstallerはWindows用の製品であり、ほかのプラットフォーム向けのサポートはありません。ちなみに、MySQL 5.5とそれ以前ではWindows向けMySQLサーバのパッケージとしてmsi形式のインストーラがダウンロードできましたが、MySQL 5.6とそれ以降ではMySQL Installerに変更されました(そのため、MySQLのダウンロードページの"Microsoft Windows"を選択してもzipパッケージのみ表示されます)。

Windows向けのMySQLとしてはどうもこのMySQL Installerが推奨されているようですが、筆者は(5.5とそれ以前のmsiパッケージ時代から)あまりよい思いをした経験がないため、zipパッケージの方が無難です。

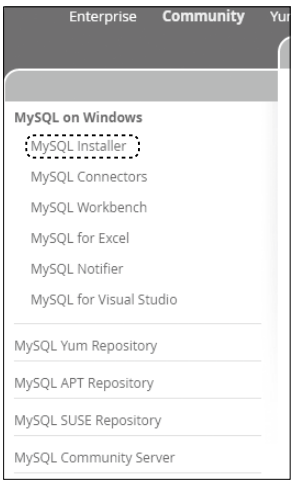
MySQL Installerのダウンロードページは今までのダウンロードページとは別に用意されており、dev.mysql.comの左のペインから"MySQL on Windows"→"MySQL Installer"を選択します。"Looking for previous GA versions?"を選択することでMySQL 5.6系列のMySQL Installerをダウンロードできます(**図23**)。

▼図20 ファイルを展開し、debパッケージをインストールする

```
$ tar xvf mysql-server_5.6.29-1ubuntu15.10_amd64.deb-bundle.tar
libmysqlclient-dev_5.6.29-1ubuntu15.10_amd64.deb
mysql-community-source_5.6.29-1ubuntu15.10_amd64.deb
libmysqld-dev_5.6.29-1ubuntu15.10_amd64.deb
mysql-server_5.6.29-1ubuntu15.10_amd64.deb
mysql-common_5.6.29-1ubuntu15.10_amd64.deb
mysql-testsuite_5.6.29-1ubuntu15.10_amd64.deb
mysql-client_5.6.29-1ubuntu15.10_amd64.deb
mysql-community_5.6.29-1ubuntu15.10_amd64.changes
mysql-community-test_5.6.29-1ubuntu15.10_amd64.deb
mysql-community-server_5.6.29-1ubuntu15.10_amd64.deb
mysql-community-bench_5.6.29-1ubuntu15.10_amd64.deb
libmysqlclient18_5.6.29-1ubuntu15.10_amd64.deb
mysql-community-client_5.6.29-1ubuntu15.10_amd64.deb

$ sudo dpkg -i *.deb
```

▼図23 MySQL Installerのダウンロード



▼図21 依存パッケージが足りないときのエラー

```
dpkg: error processing package mysql-community-server (--install):
 dependency problems - leaving unconfigured
```

▼図22 MySQLの起動

```
$ sudo service mysql start
......
 * MySQL Community Server 5.6.29 is started
```

MySQL Installerは“x86, 32-bit”のアーキテクチャしかないように見受けられますが、これは「32bit版のMySQL Installer」を表しており、「MySQL Installerによってインストールされる MySQL Serverは32bit, 64bitとも対応」しています(ややこしい……)。mysql-installer-web-communityはそれ自体にMySQLのバイナリを含まず、msiの実行時にMySQLをインターネットからダウンロードします(**図24**)。mysql-installer-communityはそれ自体にMySQLのバイナリを含んでおり、インターネットに接続されていない環境にもMySQLをインストールできます。

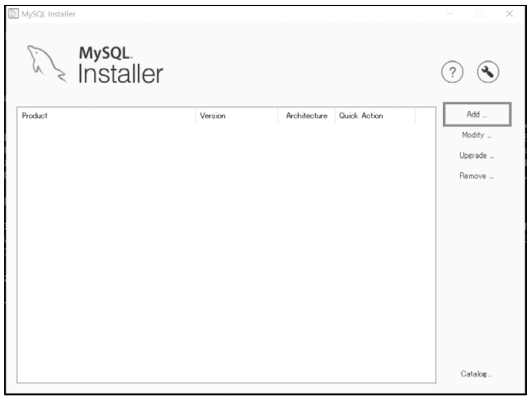
MySQL Installerのmsiファイルを起動すると、**図25**のような画面が表示されます。既にインストールされたMySQL関連の製品があった場合はここに表示されます。メニュー右側の“Add”を選択します。

使用許諾を確認したあと、インストールするコンポーネントを選択する画面になります(**図26**)。“Developer Default”に含まれているMySQL For ExcelやMySQL for Visual StudioはMicrosoft ExcelやMicrosoft Visual Studioがインストールされていないとインストールできないため、環境がそろっていない場合は“Server Only”を選択するか(Server Onlyと言いつつも、`mysql.exe`コマンドラインクライアントはインストールされます。ややこしい……)、後述するzipパッケージでインストールする必要があります。

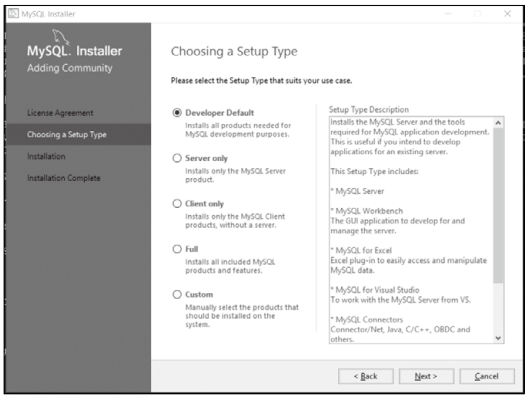
インストール自体はよくある形式で進みますので、とくに問題なく進むと思います。インストールの終了後に設定ウィザードが続けて開始され、利用するポート番号やrootのパスワード、Windowsサービスとして登録するかどうかなどが確認されます。ちょっと試す程度であれば、デフォルトのままとくに変更せず設定すれば“MySQL56”というWindowsサービスとして`mysqld.exe`が起動します。

▼図24　2つのインストーラ

▼図25　MySQL Installerの実行



▼図26　コンポーネント選択画面



zipパッケージを使用したインストール

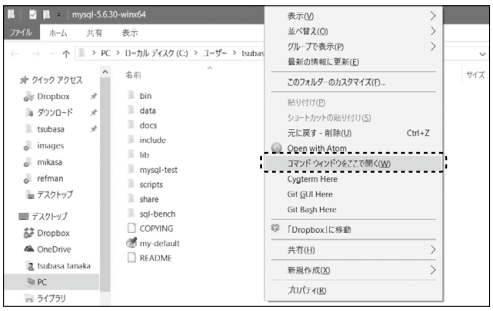
Windows用のzipパッケージは、MySQLのダウンロードページからダウンロードします(繰り返しになりますが、MySQLのダウンロードページとMySQL Installerのダウンロードページは別です)。

zipパッケージはダウンロードして解凍するだけで簡単に利用が始められます。Windowsサービスとしてインストールすることも可能ですが、そうでない場合はレジストリの更新も必要ありません。解凍先のフォルダでShiftキーを押しながら右クリックし、“コマンド ウィンドウをここで開く”を選択します(**図27**)。

コマンドプロンプトで`mysqld.exe`を実行することで、MySQLが起動します(**図28**)。`--console`オプションは、`mysqld.exe`の出力をコンソールに出力するためのオプションです。MySQL 5.6とそれ以前のバージョンでは、解凍したフォルダにはすでにデータベースの初期化が終わったdataフォルダが入っているため、解凍後すぐに`mysqld.exe`を起動できます(MySQL 5.7では変更され、`mysqld.exe --initialize`を実行する必要があります)。MySQL 5.6とそれ以前ではWindows上でデータベースの初期化を行うことは考慮されていないため、データを初期状態に戻したい場合はzipパッケージを解凍しなおしてdataフォルダを上書きします。

binフォルダには`mysql.exe`コマンドライン

▼図27 “コマンド ウィンドウをここで開く”を選択



▼図28 mysqld.exeの起動

```
C:¥Users¥yoku0825¥Desktop¥mysql-5.6.29-winx64>bin¥mysqld.exe --console
2016-04-14 06:23:59 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated.
Please use --explicit_defaults_for_timestamp server option (see documentation for more
details).
2016-04-14 06:23:59 0 [Note] bin¥mysqld.exe (mysqld 5.6.29) starting as process 11256 ...
2016-04-14 06:23:59 11256 [Note] Plugin 'FEDERATED' is disabled.
2016-04-14 06:23:59 11256 [Note] InnoDB: Using atomics to ref count buffer pool pages
2016-04-14 06:23:59 11256 [Note] InnoDB: The InnoDB memory heap is disabled
2016-04-14 06:23:59 11256 [Note] InnoDB: Mutexes and rw_locks use Windows interlocked
functions
2016-04-14 06:23:59 11256 [Note] InnoDB: Memory barrier is not used
2016-04-14 06:23:59 11256 [Note] InnoDB: Compressed tables use zlib 1.2.3
2016-04-14 06:23:59 11256 [Note] InnoDB: Not using CPU crc32 instructions
2016-04-14 06:23:59 11256 [Note] InnoDB: Initializing buffer pool, size = 128.0M
2016-04-14 06:23:59 11256 [Note] InnoDB: Completed initialization of buffer pool
2016-04-14 06:23:59 11256 [Note] InnoDB: Highest supported file format is Barracuda.
2016-04-14 06:23:59 11256 [Note] InnoDB: 128 rollback segment(s) are active.
2016-04-14 06:23:59 11256 [Note] InnoDB: Waiting for purge to start
2016-04-14 06:23:59 11256 [Note] InnoDB: 5.6.29 started; log sequence number 1625977
2016-04-14 06:23:59 11256 [Warning] No existing UUID has been found, so we assume that
this is the first time that this server has been started. Generating a new UUID: 0988f30b-
01be-11e6-9e38-00059a3c7a00.
2016-04-14 06:23:59 11256 [Note] Server hostname (bind-address): '*'; port: 3306
2016-04-14 06:23:59 11256 [Note] IPv6 is available.
2016-04-14 06:23:59 11256 [Note]   - '::' resolves to '::';
2016-04-14 06:23:59 11256 [Note] Server socket created on IP: '::'.
2016-04-14 06:23:59 11256 [Note] Event Scheduler: Loaded 0 events
2016-04-14 06:23:59 11256 [Note] bin¥mysqld.exe: ready for connections.
Version: '5.6.29'  socket: ''  port: 3306  MySQL Community Server (GPL)
```

クライアントやmysqldump.exeなども配置されており、ほかのOS版とほぼ同じ機能が利用できます。

Mac OS X

Mac OS XへのMySQLのインストール方法としては次の2つがあります。

①ネイティブパッケージ
②Homebrew[注9]を使用したインストール

②のHomebrewというパッケージマネージャーを使うことでLinuxで言うところのyumやaptのように簡単にインストール、アップグレードが行えますが、Homebrewのインストールが必要となるため今回は詳しく説明しません。CUIの操作に慣れている方やとくにこだわりがない場合は②のHomebrewを使用したインストールをお勧めします。

ネイティブパッケージを使用したインストール

MySQL公式のネイティブパッケージ(.dmgファイル)をダウンロードしてインストールを行う方法です。MySQLの公式ページにあるhttp://dev.mysql.com/downloads/mysql/5.6.html#downloadsからダウンロードを行います(**図29**)。この中からお使いのMac OS Xのバージョンに近いMySQLのDMG ARCHIVEを選択してダウンロードします。

注9) http://brew.sh/index_ja.html

ダウンロードしたDMGファイルをクリックすると**図30**のようなパッケージが見つかるので、クリックをしてインストールをしていきます。

インストーラが起動すると**図31**のような画面が開きます。ここでは、使用許諾契約を確認したりインストール先などを変更できます。とくに問題がなければ次へを押していきます。デフォルトでは/usr/local/mysql配下にインストールがされます。

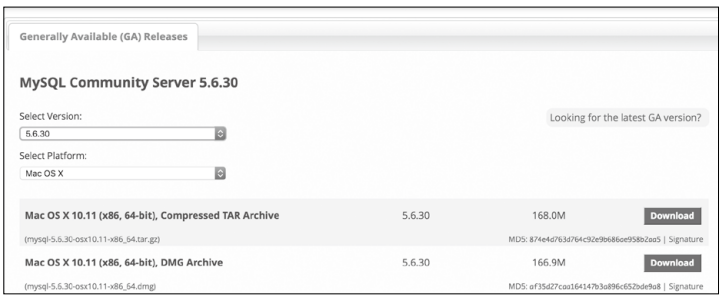
インストールが完了したところで、忘れずに/usr/local/mysql/binにパスを通しておきましょう。

.bashrcの例
```
export PATH=$PATH:/user/local/mysql/bin
```

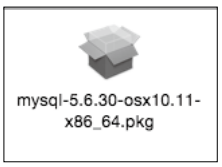
ネイティブパッケージを使用してインストールを行ったMySQLサーバの操作はシステム環境設定から行うことができます。**図32**のようにMySQLサーバの状態、MySQLサーバの起動・

▼図31　インストーラー起動画面

▼図29　DMGパッケージの選択



▼図30　DMGファイルを展開



停止・自動起動に関するオプションなどを操作・設定することができます。

またネイティブパッケージを使用した場合、データディレクトリへのパスはデフォルトでは/usr/local/mysql/data配下にあります。データのバックアップや初期化を行う際にはそちらのディレクトリを参照するようにしましょう。

Homebrewを使用したインストール

この手順でインストールする場合Homebrewが必要ですのであらかじめインストールしておいてください。Homebrewの公式ページのトップにあるインストール用のコマンドを実行すると、対話的に必要なソフトウェアの導入や権限を求められますのでコマンドラインに表示されたメッセージをよく読んで指示に従って進めてください。

Homebrewのインストールが終わった方は**図33**のコマンドを実行してMySQL 5.6をインストールしましょう(コマンドは2016年3月現在のもので、今後変更になる可能性があります)。ここで実行したMySQLのインストールコマンドの詳細はGitHub上のHomebrewのリポジトリ[注10]で確認できます(MySQL 5.7で挑戦したい人は`brew install mysql`でMySQL 5.7がインストールできます)。

インストールが完了するとMySQLの簡単な使い方が表示されます。ほかにもMySQLでインストールされるコマンドの大半がHomebrewによってpathが通っている場所にインストールされるため、mysqlクライアントなどもpathの設定をあらためて指定せずに、そのまま使用することができます。MySQLの起動コマンドは上記のインストール時の説明に書かれているコマンドを使用します。deamon化したい場合は`launchctl`コマンドでdeamon化し、今すぐ起動したい場合は`mysql.server`を使って起動しましょう。データディレクトリへのpathはデフォルトでは/usr/local/var/mysql配下にあります。また、yumやaptと同様にアップデートやアップグレードを行った際にアップグレードされてしまう問題がありますが、`brew pin homebrew/versions/mysql56`とすることでバージョンを固定化することができます。

▼図32 MySQLの操作パネル

▼図33 MySQL 5.6のインストール

```
$ brew install homebrew/versions/mysql56
...
server starting up correctly.

To connect:
    mysql -uroot

To have launchd start homebrew/versions/mysql56 at login:
  ln -sfv /usr/local/opt/mysql56/*.plist ~/Library/LaunchAgents
Then to load homebrew/versions/mysql56 now:
  launchctl load ~/Library/LaunchAgents/homebrew.mxcl.mysql56.plist
Or, if you don't want/need launchctl, you can just run:
  /usr/local/opt/mysql56/bin/mysql.server start
==> Summary
  /usr/local/Cellar/mysql56/5.6.29: 9,931 files, 313.9M
```

注10) https://github.com/Homebrew/homebrew-versions/blob/master/mysql56.rb

Linux用の汎用バイナリパッケージ(.tar.gzパッケージ)を使用したインストール(CentOS)

ここでバイナリパッケージと呼んでいるものは、コンパイル済みの実行ファイルを.tar.gz形式でアーカイブしたものを指しています。tarコマンドで任意のパスに展開できますので、1つのサーバに複数バージョンのMySQLを同居させることが容易ですが、コンパイル時オプションとして/usr/local/mysqlに展開されるものとして暗黙のデフォルトが設定されていますので、慣れない場合は暗黙のデフォルトに苦しめられることがあります。また、mysql_install_dbやuseraddのような付帯的にインストールに必要になる処理は自分で実行する必要があります。

バイナリ版はhttp://dev.mysql.com/downloads/mysql/ の「Linux - Generic」プラットフォームの「Compressed TAR Archive」を取得します。

図34の例は、MySQL 5.6.29のバイナリパッケージをダウンロードし、/usr/local/mysql5629に展開し、/data/mysql1ディレクトリをデータディレクトリとして初期化し、MySQLサーバを起動するまでの例です。

・「Compressed TAR Archive」パッケージを選択(**図35**)

これ以外に、MySQLのバイナリが内部的に依存しているパッケージがインストールされていない場合は別途自身でインストールする必要があります(libaioパッケージやperlコマンドが必要です)。この方法の利点としては、アップグレードが比較的準備しやすい(あらかじめ/usr/local/mysqlxxxxにアップグレード先のバイナリを用意しておき、古いバイナリのMySQLサーバを停止したあと、新しいバイナリで起動してmysql_upgradeをかければそれで済むケースが多いです)こと、1つのサーバに複数の異なるバージョンのMySQLサーバを起動させやすい(たとえばバックアップ専用のサーバを作る際に、1マスタに対して1プロセスを紐付けてスレーブにすることでサーバ数の削減が図れます)ことが挙げられます。ただし、バイナリのパッケージは「/usr/local/mysql」に展開されることを前提に暗黙のデフォルトが設定されているため、違うパスに展開する場合は--basedirなどのオプションで明示的に指定してやる必要があります。

筆者がプロダクション環境で利用するMySQLをインストールするときは、もっぱらこの方法を利用しています。展開先のパスの命名規則やデータディレクトリの命名規則を考慮しておけば、自動化もそれほど面倒なことではありません。

▼図35　TAR Archiveパッケージの選択

▼図34　MySQL 5.6.29バイナリパッケージのダウンロードからサーバの起動まで

```
$ wget http://dev.mysql.com/get/Downloads/MySQL-5.6/mysql-5.6.29-linux-glibc2.5-x86_64.tar.gz
$ tar xf mysql-5.6.29-linux-glibc2.5-x86_64.tar.gz
$ sudo mv mysql-5.6.29-linux-glibc2.5-x86_64 /usr/local/mysql5629
$ sudo useradd mysql
$ sudo su - mysql
$ cd /usr/local/mysql5629
$ ./scripts/mysql_install_db --datadir=/data/mysql1
$ ./bin/mysqld_safe --datadir=/data --port=3306 --socket=/tmp/mysql1.sock &
```

ソースコードからビルドしてインストール(CentOS)

MySQLはソースコードが公開されていますので、ソースコードをビルドしてMySQLのバイナリを得ることもできます。ですが、あまり他人にお勧めできるビルド方法ではないと思っています。ソースコードビルドが役に立つのは、「デバッグビルドのMySQLをビルドする」「ソースコードにパッチを当て、独自の機能を追加する」「標準ビルドには含まれない機能を有効にしてMySQLをビルドする」「デバッガーでmysqldにアタッチし、内部処理を追う」「サードパーティー製のストレージエンジンを組み込んでMySQLをビルドする」のようなケースです。あまり一般的なユースケースには当てはまらないと思います。

ソースコードはhttp://dev.mysql.com/downloads/mysql/の「Source Code」プラットフォームからダウンロードできます。パッケージ化されたソースコードのほか、「Generic Linux (Architecture Independent)」が純粋にソースコードのみを含んだ.tar.gzファイルです。また、最近ではGitHub(https://github.com/mysql/mysql-server)にソースコードが公開されており、マイナーバージョンごとにタグが打ってあるので、そちらからダウンロードしてくることもできます。

図36の例は、MySQL 5.6.29の「Generic Linux (Architecture Independent)」ソースコードをダウンロードし、/usr/local/mysql5629にインストールする手順です(make installのあとは.tar.gz版と同じため、起動までの部分は省略します)。cmakeオプションなど詳細リファレンスマニュアル[注11]を参照してください。なお、筆者のお気に入りのcmakeオプションは-iです。

MySQLの接続ライブラリの文字コードのデフォルトをlatin1以外の文字コードに固定したいので、ソースからビルドしたライブラリを作る、という使い方もあります。ソースコードからのビルドはそれほど難しいものではありませんが、MySQL 5.6以降ではサイズも大きくなり、ビルドに多少の時間やそれなりのメモリを必要とするようになってきているので、必要がなければあえてソースコードビルドを選ぶ理由はないと思います。

まとめ

本章では、複数のOS上でMySQLをインストールする方法を説明しました。

MySQL 5.1やそれ以前のバージョンでは、MySQLはもっぱらLinux上で稼働することを念頭に開発されていたため、それ以外のOSに対する最適化はあまり行われていませんでした(現在も、十分最適化されているとは言いがたい環境もありますが)。MySQL 5.5とそれ以降ではLinux以外のOS上でのパフォーマンスも改善されているため、選択肢としても十分にあり得るのではないでしょうか。また、開発マシンはMacでお手軽に、本番環境はLinuxというケースでも、複数OS上でMySQLを利用する機会は十分あると思います。

自身の使いやすいOSでお手軽にMySQLを利用してください。SD

注11) http://dev.mysql.com/doc/refman/5.6/ja/source-installation.html

▼図36　MySQL 5.6.29のGeneric Linuxをダウンロードしインストールする手順

```
$ wget http://dev.mysql.com/get/Downloads/MySQL-5.6/mysql-5.6.29.tar.gz
$ tar xf mysql-5.6.29.tar.gz
$ cd mysql-5.6.29
$ cmake -DCMAKE_INSTALL_PREFIX=/usr/local/mysql5629 -DBUILD_CONFIG=mysql_release .
$ make
$ sudo make install
```

第3章

MySQLでデータベースを作ってみよう!

自分で考える・学ぶ・やってみる

この章では実際にMySQLでデータベース「電子掲示板」を作ってデータを操作してみます。その前にまずはデータの管理について説明します。そしてユーザの作成、権限の付与のやりかた、データベースそのものの作成方法を解説します。レコードの登録／取り出し、更新／削除など基本的な操作においてSQL構文を実行し、その動作を学びます。

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

テーブル

MySQLに限りませんがRDBMS(Relational Data Base Management System)はデータをテーブルに格納して管理します。テーブルは行(ロー／レコード)と列(カラム／フィールド)で構成される二次元の構造を持っています。図で表現される場合は、通常は行が縦に列が横に描かれます(**図1**)。

ただしテーブルは単なる二次元の表ではありません。行と列は可換でなく明確に役割が異なります。行はデータを、列はデータの属性を表します。列(属性)には型や制約があります。同じテーブル内の各データは同じ属性であれば同じ型を持っています。

つまりテーブルは同じ構造を持ったデータの集まりなのです。「ほぼ同じ構造だけどあるデータだけ属性が1つ多い(または少ない)」という場合であっても、構造が異なるので同じテーブルにすべきではありません。

また、「今は問題ないが、もしかしたら将来何か属性が増えるかもしれないから、予備用にカラムを作っておく」というのも悪手です。

▼図1　RDBMSにおけるテーブル

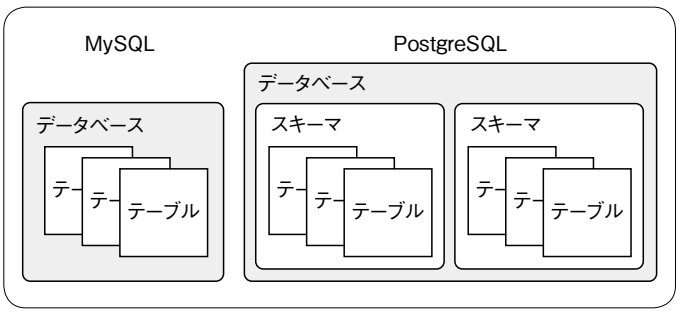
レコード1	カラム1	カラム2	カラム3	カラム4
レコード2				
レコード3				
レコード4				

データベース

MySQLでは複数のテーブルをまとめて管理するための構造をデータベースと呼んでいます(**図2**)。データベースをまたがって、異なるデータベース間のテーブルを同時に扱うこともできます。ほかのRDBMSとはデータベースの概念が異なる場合があります。

たとえばPostgreSQLではMySQLのデータベースに相当するものはスキーマと呼ばれます。PostgreSQLは複数のスキーマをまとめたものをデータベースと呼んでいます。MySQLにはPostgreSQLのデータベースに相当するものはありません。

▼図2　MySQLにおけるデータベース



電子掲示板データベース

電子掲示板アプリケーションを想定したデータベースを作ってみます。

ユーザ作成

MySQLのデータベースにアクセスするためにはユーザを作る必要があります。

ユーザごとにシステムやデータベース／テーブル／カラムに対する権限を設定できます。インストール直後はrootというユーザが作られていますが、このユーザはすべての権限を持つMySQLのスーパーユーザですので、MySQL自体の管理以外には使用すべきではありません。

電子掲示板アプリケーションのための専用ユーザを作りましょう。ユーザを作成するにはrootでMySQLに接続し、CREATE USER命令を使用します(**図3**)。

図3ではtmtms@localhostユーザをパスワードabcdefgで作成しています(実際に作成する場合はパスワードは強固なものにしてください)。

ユーザはユーザ名とクライアント名(ホスト名／IPアドレス)の組で指定します。MySQLのユーザ名はOSのユーザ名とは直接は関係ありません[注1]。クライアント名が異なればユーザ名が同じでも異なるユーザとして扱われます。どのクライアントからのアクセスでも許可したい場合は、クライアント名として%を指定します。

なお、この例で使用したlocalhostは、IPアドレス127.0.0.1のことではありません。TCP/IPではなくソケットファイル(/tmp/mysql.sockなど)を使用したローカルホストからのアクセスを意味します。

権限の付与

CREATE USERで作成したばかりのユーザには何の権限もありません。GRANT命令で権限を付与できます。

```
mysql> GRANT ALL ON bbs.* TO ➪
tmtms@localhost;
```

この例ではbbsデータベースに対する全権限をtmtms@localhostユーザに与えています。これはbbsデータベースの作成／削除の権限も含みます。権限には、テーブルの作成／破棄やレコードの参照／登録／更新／削除などの種類があり、データベース／テーブル／カラムごとに細かく設定することもできます。詳しくはMYSQLのマニュアル「MySQLで提供される権限[注2]」を参照してください。

データベース作成

データベースを作りましょう。MySQLにrootで接続中の場合はQUIT命令で切断して、先ほど作成したユーザで接続してください。

```
mysql> QUIT
Bye
% mysql -utmtms -p --default-character-➪
set=utf8mb4
Enter password: abcdefg
mysql>
```

「--default-character-set=utf8mb4」は接続の文字コードをUTF-8に設定するオプションです。

CREATE DATABASE命令でデータベースを作成します。

```
mysql> CREATE DATABASE bbs CHARSET ➪
utf8mb4;
mysql> USE bbs;
```

▼図3　CREATE USER命令

```
% mysql -uroot -p
Enter password: rootのパスワードを入力
mysql> CREATE USER tmtms@localhost IDENTIFIED BY 'abcdefg';
```

注1) Linuxではauth_socket認証プラグインを使用することでOSと同じユーザを使用できます。

注2) URL https://dev.mysql.com/doc/refman/5.6/ja/privileges-provided.html

ここではbbsという名前のデータベースを作っています。データベース名はOSのディレクトリとして用いられるため、大文字小文字が区別されるかどうかはシステムに依存します。Linuxの場合は大文字小文字は区別されます。

「CHARSET utf8mb4」はそのデータベース配下で使用する文字コードをUTF-8に指定しています。テーブルに日本語データを格納したい場合は、とくにこだわりがなければutf8mb4を指定しておくのがよいでしょう(コラム「文字コード」参照)。

2行目のUSE bbsはbbsデータベースの選択です。これ以降のテーブルに対する命令はbbsデータベース内のテーブルが対象となります。

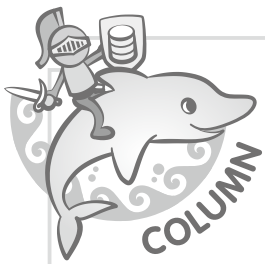
先に進む前にsql_modeを設定しておきます(**図4**)。詳しくはコラム「sql_mode」を参照してください。

テーブル作成

電子掲示板システムについて必要なデータを考えます。まずは投稿データしかない非常に単純なシステムにしてみます。

▼図4　sql_modeの設定

```
mysql> SET sql_mode='TRADITIONAL,NO_AUTO_VALUE_ON_ZERO,ONLY_FULL_GROUP_BY';
```



文字コード

MySQLは複数の文字コード(character set、charset)に対応しています。執筆時点(MySQL 5.7.11)では41個の文字コードがあります(SHOW CHARSET命令で見ることができます)。その中で日本語で使えるものは次の6個ですが、

ujis, sjis, cp932, eucjpms, utf8, utf8mb4

新しく作成するデータベースではutf8mb4を選んでおけば問題ないでしょう。

各charsetと、文字セットとエンコーディングの関係を**表A**に示します。

文字セットは文字の集合で、エンコーディングは文字に番号を割り当てる方法のことです。JIS X 0201、JIS X 0208文字セットには「①」「㈱」「髙」などの文字が含まれていません。Windows-31J文字セットにはそれらの文字は含まれていますが、「♡」「㊙」などの記号が含まれていません。

ユニコードは、日本語だけではなくすべての言語の文字が含まれています。ただし、utf8はU+10000以降の文字(4バイトUTF-8文字)を扱えません。そのため「🐬」「🍺」などの絵文字を扱うことができません。utf8mb4はすべてのユニコード文字を扱うことができます。

シフトJIS系エンコーディングはWindowsでよく使用されているため、以前はcp932が使用されることもあったのですが、現在はユニコードが普及したため、過去の経緯や1バイトでも節約したいなど[注A]の特別な事情がなければcp932を使用する理由はないでしょう。

▼表A　charsetと文字セットとエンコーディングの関係

文字セット＼エンコーディング	シフトJIS系	EUC系	UTF-8
JIS X 0201、JIS X 0208	sjis	ujis	-
Windows-31J	cp932	eucjpms	-
ユニコード(～U+FFFF)	-	-	utf8
ユニコード	-	-	utf8mb4

注A) UTF-8エンコーディングでは、ほとんどの日本語の文字は3バイトで表現されますが、シフトJIS系エンコーディングでは、2バイトで表現できます。

投稿データの属性は、通し番号、投稿者名(最大30文字)、投稿時刻、投稿内容(最大1000文字)の4つにします(**表1**)。

テーブルを作成するには、CREATE TABLE命令を使用します。CREATE TABLEの構文は`CREATE TABLE テーブル名 ( カラム名 型 , ...);`です。

今回のテーブルを作成するには次のようにします。

```
mysql> CREATE TABLE posts (
    -> id INT,
    -> name VARCHAR(30),
    -> timestamp DATETIME,
    -> message VARCHAR(1000)
    -> );
```

テーブル名は大文字小文字が区別されますが、カラム名は区別されません。なお、MySQLのテーブル名とカラム名には日本語も使用できます。日本語のテーブル名/カラム名はわかりやすくて良いのですが、扱いにくいので個人的には英数字だけで作成したほうが良いと思います。

あまりお勧めしませんが、「`」でくくると空白や記号を含むテーブル名/カラム名を使用できます。

MySQLにはさまざまな型がありますが、INT(整数)、VARCHAR(文字列)、DATETIME(日時)がよく使用されます。詳しくはマニュアル「データ型[注3]」をご覧ください。

レコード登録

作成したテーブルにレコードを登録してみます。レコードを登録するには、INSERT命令を使用します。

INSERTの構文は`INSERT INTO テーブル名 ( カラム名 ,... ) VALUES ( 値 ,... )`です(**図5**)。

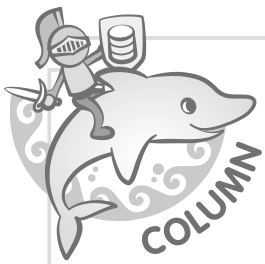
注3) URL https://dev.mysql.com/doc/refman/5.6/ja/data-types.html

▼表1 電子掲示板システムのテーブル例

属性	カラム名	型
通し番号	id	INT
投稿者名	name	VARCHAR(30)
投稿時刻	timestamp	DATETIME
投稿内容	message	VARCHAR(1000)

▼図5 INSERT命令でレコードを登録

```
mysql> INSERT INTO posts (id,name,timestamp,message) ↵
VALUES (1,'名無し','2016-04-11 12:34:56','はじめまして');
mysql> INSERT INTO posts (id,name,timestamp,message) ↵
VALUES (2,'名無し','2016-04-11 12:44:33','こんにちは!');
```



sql_mode

MySQLは標準では、指定された値のままレコードに格納できなくても、なるべくエラーにならないように処理を続けようとします(エラーではなく警告になります)。

たとえば、NULLを許さずデフォルト値も設定されていないカラムを指定せずにINSERTしてもエラーにならず、型に応じた暗黙のデフォルト値が設定されます。また、DATE型のカラムに`0000-00-00`や`2016-04-00`などの0を含む値が許されています。

ほかにも、文字列カラムで最大長を超えた長さの文字列を格納しようとしてもエラーにならずに切り捨てられたり、文字コード変換で正しく変換処理ができなかった場合などにもエラーになりません。

これらは一般に期待する振る舞いではないと思われます。

sql_modeを設定すれば、このようなMySQL特有の振る舞いを変更できます。

クライアント接続ごとに`SET sql_mode=...`を発行するのではなく、すべての接続でsql_modeの設定を有効にしたい場合は、mysqldの起動時オプションや設定ファイルで指定することができます。

sql_modeについて、詳しくはMySQLのマニュアル「サーバSQLモード[注B]」を見てください。

注B) URL https://dev.mysql.com/doc/refman/5.6/ja/sql-mode.html

レコード取り出し

テーブル内のデータを取り出すにはSELECT命令を使用します。

SELECTの基本的な構文は、SELECT カラム名,... FROM テーブル名 WHERE レコード抽出条件です。カラム名を並べる代わりに*を指定すると、テーブルの全カラムを取り出します。WHEREを指定しない場合は全レコードを取り出します(**図6**)。

mysqlコマンドはSELECTの結果を1行1レコードの表形式で出力します。末尾の;の代わりに\Gを使用すると1行1カラムの形式で出力できます(**図7**)。

カラムに長いデータが格納されている場合は標準の表形式では見にくくなるので、その場合は\Gを使うと見やすくなると思います。

更新

カラムの値を更新するにはUPDATE命令を使用します。UPDATEの構文は、UPDATE テーブル名 SET カラム名=値, ... WHERE レコード特定条件です。

WHEREを指定しない場合は全レコードが対象となります。id=2のレコードのmessageカラムの内容を変更してみます(**図8**)。

▼図6　SELECT命令でデータを取り出す

```
mysql> SELECT name,message FROM posts;
+----------+--------------------+
| name     | message            |
+----------+--------------------+
| 名無し   | はじめまして       |
| 名無し   | こんにちは！       |
+----------+--------------------+
mysql> SELECT * FROM posts WHERE id=1;
+----+----------+---------------------+--------------------+
| id | name     | timestamp           | message            |
+----+----------+---------------------+--------------------+
|  1 | 名無し   | 2016-04-11 12:34:56 | はじめまして       |
+----+----------+---------------------+--------------------+
```

▼図7　\Gで1行1カラム形式で出力した例

```
mysql> SELECT * FROM posts\G
*************************** 1. row ***************************
       id: 1
     name: 名無し
timestamp: 2016-04-11 12:34:56
  message: はじめまして
*************************** 2. row ***************************
       id: 2
     name: 名無し
timestamp: 2016-04-11 12:44:33
  message: こんにちは！
```

▼図8　カラムの値を更新する例

```
mysql> UPDATE posts SET message='こんばんは！' WHERE id=2;
mysql> SELECT * FROM posts;
+----+----------+---------------------+--------------------+
| id | name     | timestamp           | message            |
+----+----------+---------------------+--------------------+
|  1 | 名無し   | 2016-04-11 12:34:56 | はじめまして       |
|  2 | 名無し   | 2016-04-11 12:44:33 | こんばんは！       |
+----+----------+---------------------+--------------------+
```

削除

レコードを削除するにはDELETE命令を使用します。DELETEの構文は、DELETE FROM テーブル名 WHERE レコード特定条件です。WHEREを指定しない場合は全レコードが削除されます。id=2のレコードを削除してみます(**図9**)。

NOT NULL制約

RDBには値が未定の状態を表すNULLというものがあります。0や空文字列とも異なります。

使いようによっては便利なのですが、カラムがNULLになると困ることもあります。何も指定しないとカラムはNULLになり得るように作成されてしまいます。

たとえば、INSERTでカラムを指定しないとそのカラムはNULLになります(**図10**)。

NULLにしたくないカラムは、CREATE TABLE時にNOT NULLを指定します。今回の場合はNULLになると困るカラムばかりですので、すべてのカラムに指定します。

いったんテーブルを削除して作り直します。

```
mysql> DROP TABLE posts;
mysql> CREATE TABLE posts (
    -> id INT NOT NULL,
    -> name VARCHAR(30) NOT NULL,
    -> timestamp DATETIME NOT NULL,
    -> message VARCHAR(1000) NOT NULL
    -> );
```

これでカラムを指定しないでINSERTするとエラーになるようになります(**図11**)。

▼図9　データの削除方法の例

```
mysql> SELECT * FROM posts;
+----+----------+---------------------+--------------------+
| id | name     | timestamp           | message            |
+----+----------+---------------------+--------------------+
|  1 | 名無し   | 2016-04-11 12:34:56 | はじめまして       |
|  2 | 名無し   | 2016-04-11 12:44:33 | こんばんは！       |
+----+----------+---------------------+--------------------+
mysql> DELETE FROM posts WHERE id=2;
mysql> SELECT * FROM posts;
+----+----------+---------------------+--------------------+
| id | name     | timestamp           | message            |
+----+----------+---------------------+--------------------+
|  1 | 名無し   | 2016-04-11 12:34:56 | はじめまして       |
+----+----------+---------------------+--------------------+
```

▼図10　カラムをNULLにしてしまう例

```
mysql> INSERT INTO posts (id,name) VALUES (3,'名無し');
mysql> SELECT * FROM posts WHERE id=3;
+----+----------+-----------+---------+
| id | name     | timestamp | message |
+----+----------+-----------+---------+
|  3 | 名無し   | NULL      | NULL    |
+----+----------+-----------+---------+
```

▼図11　カラムが指定されていないときにエラーになる例

```
mysql> INSERT INTO posts (id,name) VALUES (3,'名無し');
ERROR 1364 (HY000): Field 'timestamp' doesn't have a default value
Error (Code 1364): Field 'timestamp' doesn't have a default value
Error (Code 1364): Field 'message' doesn't have a default value
```

テーブル構造を変更するのに毎回テーブルを破棄して作りなおすのはやってられません。

データを残したままテーブル構造を変更するのは ALTER TABLE命令を使用します。

・カラムの追加：ALTER TABLE テーブル名 ADD カラム名 型;
・カラムの削除：ALTER TABLE テーブル名 DROP カラム名;
・カラムの型変更：ALTER TABLE テーブル名 MODIFY カラム名 新しい型;
・カラム名と型変更：ALTER TABLE テーブル名 CHANGE 古いカラム名 新しいカラム名 新しい型;

idカラムの型をINT NOT NULLに変更するには次のようにします。

```
mysql> ALTER TABLE posts MODIFY id INT NOT NULL;
```

エラーにせず、指定されていないカラムはデフォルト値になるようにすることもできます。とくにDATETIME型の場合は現在時刻を設定できます(**図12**)。

一意性制約と自動採番

現在のpostsテーブルのidカラムには制約がないので、idカラムの値がレコード間で重複したとしてもエラーになりません(**図13**)。

テーブル内で一意であるべきカラムにはUNIQUEを指定することで一意性制約(UNIQUE制約)を設定できます(**図14**)。

また、idの値をいちいち指定しなくてもカラムにAUTO_INCREMENTを指定すると自動的に連番を割り当てることができます(**図15**)。

▼図12　指定されていないDATETIME型のカラムに現在時刻が設定される

```
mysql> ALTER TABLE posts
    -> MODIFY timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    -> MODIFY message VARCHAR(1000) NOT NULL DEFAULT '';
mysql> INSERT INTO posts (id,name) VALUES (3,'名無し');
mysql> SELECT * FROM posts WHERE id=3;
+----+-----------+---------------------+---------+
| id | name      | timestamp           | message |
+----+-----------+---------------------+---------+
|  3 | 名無し    | 2016-04-11 13:11:03 |         |
+----+-----------+---------------------+---------+
```

▼図13　idカラムの値が重複していてもエラーにならない例

```
mysql> INSERT INTO posts (id,name,message) VALUES (1,'太郎','1げっと');
mysql> INSERT INTO posts (id,name,message) VALUES (1,'花子','2げっと');
mysql> SELECT id,name,message FROM posts;
+----+--------+-----------+
| id | name   | message   |
+----+--------+-----------+
|  1 | 太郎   | 1げっと   |
|  1 | 花子   | 2げっと   |
+----+--------+-----------+
```

▼図14　カラムに一意性制約(UNIQUE制約)を設定する例

```
mysql> DELETE FROM posts;
mysql> ALTER TABLE posts MODIFY id INT NOT NULL UNIQUE;
mysql> INSERT INTO posts (id,name,message) VALUES (1,'太郎','1げっと');
mysql> INSERT INTO posts (id,name,message) VALUES (1,'花子','2げっと');
ERROR 1062 (23000): Duplicate entry '1' for key 'id'
```

テーブル分割とJOIN

現在のpostsでは毎回ユーザ名を入力しているため、同じ名前であっても同一人物が書き込んだメッセージかどうかわかりません。自分の書いたメッセージを削除するという機能を持たせることもできません。あらかじめ登録をしたユーザだけメッセージを投稿できるようにしてみましょう。まずはusersテーブルを作成し、ユーザを登録します(図16)。

usersテーブルと関連付くようにpostsテーブルを作り直します(図17)。ユーザに関する情報はusersテーブルで管理し、postsテーブルではユーザIDで関連付けるようにします(図18)。こうすることで、ユーザ名が途中で変更されたとしても、過去のメッセージが誰のものかがわか

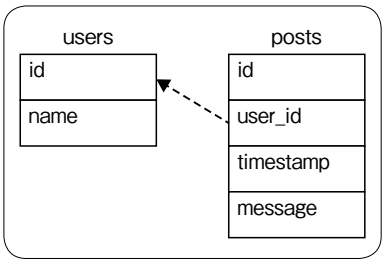
▼図15　自動的に連番を付ける例

```
mysql> DELETE FROM posts;
mysql> ALTER TABLE posts MODIFY id INT NOT NULL AUTO_INCREMENT UNIQUE;
mysql> INSERT INTO posts (name,message) VALUES ('太郎','1げっと');
mysql> INSERT INTO posts (name,message) VALUES ('花子','2げっと');
mysql> SELECT id,name,message FROM posts;
+----+--------+------------+
| id | name   | message    |
+----+--------+------------+
|  1 | 太郎   | 1げっと    |
|  2 | 花子   | 2げっと    |
+----+--------+------------+
```

▼図16　usersテーブルを作成し登録する

```
mysql> CREATE TABLE users (
    -> id INT NOT NULL AUTO_INCREMENT UNIQUE,
    -> name VARCHAR(30) NOT NULL
    -> );
mysql> INSERT INTO users (name) VALUES ('太郎'),('花子');
mysql> SELECT id,name FROM users;
+----+--------+
| id | name   |
+----+--------+
|  1 | 太郎   |
|  2 | 花子   |
+----+--------+
```

▼図17　テーブルのJOINの概念



▼図18　postsテーブルにはユーザ名を持たずユーザIDのみ

```
mysql> DROP TABLE posts;
mysql> CREATE TABLE posts (
    -> id INT NOT NULL AUTO_INCREMENT UNIQUE,
    -> user_id INT NOT NULL,
    -> timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    -> message VARCHAR(1000) NOT NULL DEFAULT ''
    -> );
mysql> INSERT INTO posts (user_id,message) VALUES (1,'はじめまして');          -- 太郎のメッセージ
mysql> INSERT INTO posts (user_id,message) VALUES (2,'こんにちわ');            -- 花子のメッセージ
mysql> INSERT INTO posts (user_id,message) VALUES (1,'よろしくお願いします'); -- 太郎のメッセージ
mysql> SELECT id,user_id,message FROM posts;
+----+---------+------------------------------+
| id | user_id | message                      |
+----+---------+------------------------------+
|  1 |       1 | はじめまして                 |
|  2 |       2 | こんにちわ                   |
|  3 |       1 | よろしくお願いします         |
+----+---------+------------------------------+
```

らなくなることはありません。

アプリケーションで投稿一覧を表示する際はユーザ名とメッセージが必要ですが、毎回usersテーブルとpostsテーブルそれぞれからデータを取得するのは面倒です。

1回のSELECTで複数のテーブルから値を取得することもできます。SELECTのFROMで`FROM テーブルA JOIN テーブルB ON 結合条件`という構文で記述します。

異なるテーブルに同じ名前のカラムがある場合(今回の場合はidカラム)は、**テーブル名.カラム名**と記述してどちらのテーブルのカラムかあいまいにならないようにする必要があります。

図19の例ではusersテーブルのidカラムとpostsテーブルのuser_idカラムを用いてusersテーブルとpostsテーブルを結合しています。

外部キー制約

usersとpostsはユーザIDで関連付けられていますが、これはあくまでもアプリケーションがそのように利用しているだけで、データベースとしてはそのような制約はありません。

usersに登録されていないIDを指定してpostsにレコードを作成することもできてしまいますし、usersからユーザを削除してもpostsのメッセージはそのままです。

外部キーで関連を明示する

外部キーを設定することで、テーブル間の関連を明示できます(**図20**)。

FOREIGN KEYは、そのカラムの値が外部のテーブルの指定したカラムに値が存在することという制約を設定します。この例ではpostsテーブルのuser_idカラムに入る値は、必ずusersテーブルのidカラムに存在している値でないといけないということを意味しています。

親テーブルと子テーブルの関連

FOREIGN KEYを設定したテーブル(この例ではposts)を子テーブル、参照先の外部のテーブル(この例ではusers)を親テーブルとも呼びます。

試してみると、usersテーブルに値が存在するuser_id=1のレコードは登録できますが、存在しないuser_id=3のレコードはエラーになることがわかります(**図21**)。

▼図19　テーブルを結合した例

```
mysql> SELECT posts.id,name,message FROM posts JOIN users ON posts.user_id=users.id;
+----+--------+------------------------------+
| id | name   | message                      |
+----+--------+------------------------------+
|  1 | 太郎   | はじめまして                 |
|  2 | 花子   | こんにちわ                   |
|  3 | 太郎   | よろしくお願いします         |
+----+--------+------------------------------+
```

▼図20　外部キーでテーブル間の関連を明示する例

```
mysql> ALTER TABLE posts ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users
(id);
```

▼図21　外部キーによりエラーになる

```
mysql> INSERT INTO posts (user_id,message) VALUES (1,'てすと');
mysql> INSERT INTO posts (user_id,message) VALUES (3,'てすと');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`bbs`.`posts`, CONSTRAINT `fk_user_id` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`))
```

また、子テーブルから参照されているレコードを親テーブルから削除しようとしてもエラーになります(**図22**)。

親テーブルから削除したときに子テーブルの関連するレコードを自動的に削除するようにすることもできます。FOREGIN KEYにON DELETE CASCADEを指定します。

usersから該当ユーザを削除すると、そのユーザが所有していたpostsレコードが削除されることが確認できます(**図23**)。

インデックス

データが少ないうちは良いのですが、データが大量に溜まってくると、特定のレコードを探し出す処理が遅くなります。分厚い本の中から特定のキーワードが書かれているページを先頭から順番に探すようなものです。末尾に索引がある本の場合は、キーワードが何ページにあるかを指し示してくれるので、すぐに見つけ出すことができます。

RDBで本の索引に該当するものはインデックスです。カラムにインデックスを設定しておけば、そのカラムを条件に指定した検索が高速になることが期待できます。

投稿した日時でレコードを絞り込みたい場合を考慮して、postsテーブルのtimestampにインデックスを設定してみます。

テーブル作成時にインデックスを指定する場合:

```
mysql> CREATE TABLE posts (
    -> id INT NOT NULL AUTO_INCREMENT ↵
UNIQUE,
    -> user_id INT NOT NULL,
    -> timestamp DATETIME NOT NULL ↵
DEFAULT CURRENT_TIMESTAMP,
    -> message VARCHAR(1000) NOT NULL ↵
DEFAULT '',
    -> INDEX (timestamp)
    -> );
```

既存のテーブルにインデックスを追加する場合:

```
mysql> ALTER TABLE posts ADD INDEX ↵
(timestamp);
```

まとめ

本章ではMySQLでのSQLの基本的な使い方を具体的な例とともに説明しました。

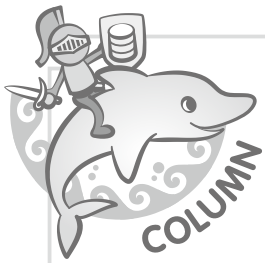
SQLは奥が深くここで説明したことはほんの入り口にしか過ぎませんが、この記事がRDBに初めて触れる方の一助になれば幸いです。SD

▼図22　子テーブルから参照されているレコードを親テーブルから削除するとエラー

```
mysql> DELETE FROM users WHERE id=1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
(`bbs`.`posts`, CONSTRAINT `fk_user_id` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`))
```

▼図23　子テーブルで参照しているデータを自動的に削除する例

```
mysql> ALTER TABLE posts DROP FOREIGN KEY fk_user_id;
mysql> ALTER TABLE posts ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users
(id) ON DELETE CASCADE;
mysql> SELECT id,message FROM posts;
+----+-----------+
| id | message   |
+----+-----------+
|  1 | てすと    |
+----+-----------+
mysql> DELETE FROM users WHERE id=1;
mysql> SELECT id,message FROM posts;
Empty set
```



MySQLのコミュニティと情報源 ——ひとりで悩むよりも仲間に相談

日本MySQLユーザ会

URL http://mysql.gr.jp/

日本MySQLユーザ会は、当時は日本でマイナーだったMySQLの普及と、MySQLで日本語を使用できるようにするのを目的として、2000年に発足しました。略称はMyNA(マイナ)です。メーリングリストをおもな活動の場としていましたが、現在はメーリングリストの流量はかなり減少しています。各地で開催されているオープンソースカンファレンスにも参加しています。また、不定期に「MyNA会」と称して、日本MySQLユーザ会独自のセミナーイベントも開催しています。

MySQL Casual

URL http://mysql-casual.org/

MySQL Casualは「もっと深く浅く、広く狭くMySQLを使っていこうと思っている趣旨の人とのつながりを作っていくための緩めのコミュニティ」です。MySQL Casual Talksというセミナーイベントを年に数回開催しています。カジュアルという名前に反して濃い話が多く、「カジュアルではなくガチュアル」という声もよく聞かれます。

メーリングリスト

URL http://www.mysql.gr.jp/ml.html

日本MySQLユーザ会のメーリングリスト。ユーザ会発足前から存在するメーリングリストです。過去のアーカイブも参照できます。普段はほとんど流量がありませんが、何か質問があった場合はメーリングリストに投稿すれば、1日2日くらいで有識者からの返事がつくと思います。

Slack

URL http://mysql-casual-slackin.herokuapp.com/　アーカイブページ　http://mysql-casual.slackarchive.io/

MySQL CasualのSlackチャネルがあります。チャットですので気軽にMySQLについて会話できます。

公式ドキュメント

URL http://dev.mysql.com/doc/refman/5.6/ja/

MySQLのリファレンスマニュアルはWebページを参照ください。バージョン5.0、5.1、5.5、5.6、5.7のマニュアルが用意されていて、5.6は日本語化もされています。

MySQLのマニュアルは非常に詳しいです。その分、量も多いため最初から最後まで通して読もうとすると厳しいものがありますが、目次だけでも眺めて、どこに何が書かれているかくらいは把握しておくと、少しわからないことが出てきたときに調べるのが楽になります。

本家メーリングリスト

URL https://lists.mysql.com

MySQLの公式メーリングリストは当然ながら英語です。ジャンルごとに複数のメーリングリストがあります。Announcementsメーリングリストに入っておくと、新しいバージョンがリリースされたときにメールが届くので最新リリース情報を知りたい人には便利です。

Android Wear アプリ開発入門
～より生活に密着するスマートデバイスの世界～

特別編（第7回） Android Wear最新動向

神原 健一（かんばら けんいち） Web http://blog.iplatform.org Twitter @korodroid
iplatform.orgにて情報発信するかたわら、「セカイフォン」などを開発。Droidconなどでのカンファレンス講演、MWC/CES/IFAでのプロダクト展示、執筆などの活動も実施。NTTソフトウェア株式会社テクニカルプロフェッショナル。現在はAndroid以外のモバイルOSにも取り組み、公私にわたってモバイルアプリの世界に没頭中。

はじめに

ご無沙汰しております。本誌2015年8月号で連載をいったんお休みさせていただいてから、Android Wear（スマートウォッチなどのウェアラブルデバイスを対象としたAndroidプラットフォーム）を取り巻く状況は大きく変化してきました。Android Wear（以降「Wear」と表記します）は、Google I/O 2014にて、対応デバイス、およびアプリを開発するためのSDKが発表されてからもうすぐ2年が経ちます。皆さんは、Wear実機をお持ちでしょうか？ Wearはユーザが常に腕につけているという、スマホやタブレット（以降「Handheld」と総称します）にはない特徴を持っています（**図1**）。Wearならではの良さを活用したアプリを開発すれば、これまでにない価値をユーザに提供できる可能性があります。Wearを搭載した新しい実機も「Casio Smart Outdoor Watch」「Tag Heuer Connected」「ASUS ZenWatch 2」など続々とリリースされています（**写真1**）。

▼写真1　Android Wear搭載の最新スマートウォッチ

▼表1　Android WearのOSの変遷

バージョン	主な特徴	リリース時期
Android 4.4W	Android Wearリリース時の最初のバージョン	2014年6月
Android 4.4W.2	Wear単体での音楽再生、時間表示の改善、バッテリー省電力化ほか	2014年10月
Android 5.0	Lollipopベース、映画館・太陽光モードのサポート、Watch Face APIの公式提供ほか	2014年12月
Android 5.1	Wear単体でのWi-Fi接続、手首ジェスチャー操作、画面常時表示、絵文字入力機能、Wearの画面ロックほか	2015年5月
Android 5.1	対話型Watch Faceのサポート	2015年8月
Android 6.0	新パーミッション対応、新ジェスチャー対応、円形・非円形別レイアウト、スピーカー対応、Intel x86サポートほか	2016年2月～順次

また、Wear搭載OSの最初のバージョンは“Android 4.4（KitKat）”ベースでした。その後バージョンアップにより進化を遂げ、現在は“Android 6.0（Marshmallow）”ベースになりました（**表1**）。これにより、Wearで実現できる世界はさらに広がりました。本稿では、Wearの最新機能について、特徴に加え、具体的な使い方やコード例をご紹介します。これらを活用することで、より面白いWearアプリを開発するヒントにしていただけると幸いです。

Android Wearの最新機能

Wearは Android 6.0ベースとなり、冒頭で紹介した新しい機能が追加されています。ユー

ザにとっては便利な機能が、開発者にとっても魅力的な機能が提供されました。同時にアプリ開発時に考慮すべきポイントが増えています。それでは、これらの特徴や活用方法など、具体的な内容を解説していきます。

新機能1：新パーミッションモデル

Androidアプリを6.0以上に対応させる場合、新パーミッションモデルの考慮が必須となりました。正しく実装されていないと、機能が正常に動作しない原因となります。

Wearの話に入る前に前提知識として、HandheldアプリをAndroid 6.0以上に対応させるときの注意点を簡単に紹介します。新パーミッションモデルとは、Android 6.0にて採用されたもので、アプリが利用するパーミッションの利用可否をユーザがコントロールできるしくみを指します。Android 5.xまでは、Playストアからアプリをダウンロードする契機で、利用するパーミッションをユーザが確認し、インストールするかどうかを判断していました。

一方Android 6.0以上では、危険度が高いパーミッションを必要とする機能(連絡帳の読み込みや、位置情報取得など)については、各アプリごとにその利用許可を原則、ユーザから明示的に得ることが必須となりました(**図2左**)。許可を得ない限り、アプリが同機能を利用することはできません。また、ユーザは一度許可した後でも、自らの意思でその可否をいつでも変更可能です。Androidの設定内の各アプリの詳細画面で変更できます(**図2右**)。パーミッションを剥奪すると、アプリはその機能を利用することができなくなります。ユーザにとっては便利な機能である一方、開発者はその影響を考慮した実装が必要であるというのが大事なポイントです。

次に、Wear搭載のOSがAndroid 6.0以上である場合、パーミッションに関する考慮をWearアプリでも行う必要があります。Wearならではの注意点を交えて解説します。

Handheldアプリの場合と同様に、該当する機能を利用するには、実行時にユーザから許可を得る必要があります(**図3**)。この振る舞いは、**リスト1**のようなコードにより実現できます。

まず、**リスト1の(1)**でパーミッションの取得状況を確認します。未取得の場合は、ユーザへの許可依頼が必要となります。そこで、**リスト1の(2)**を実行します。これにより**図3**の画面が表示され、ユーザに意思確認を行うことができます。許可もしくは拒否をユーザが選択すると、**リスト1の(3)**がコールバックされ、選択された結果を取得できます。許可された場合は、当然ながら、そのパーミッションを必要とする機能を利用できるようになります。

Wearの場合も、Handheldと同様、設定からパーミッションの利用許可状況を確認したり、変更することも可能です(**図4**)。

ここまでは、WearアプリがWearデバイス

▼図1　Wearの代表的な3つの特徴

①通知　②音声入力　③独自アプリ

▼図2　Handheldアプリにおける新パーミッションモデル

▶図3
Wearアプリにおける新パーミッションモデル①

▼リスト1　ユーザからの利用許可取得

```
private void reqPermission() {
    // (1)パーミッション取得状況の確認
    if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
        // (2)パーミッションの利用許可依頼
        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_
FINE_LOCATION}, MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
    } else {
        // 位置情報取得に関する処理
        // ...
    }
}

// (3)パーミッション利用許可結果のコールバック
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION: {
            // パーミッション取得：同意
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // 位置情報取得に関する処理
                // ...
            }
        }
    }
}
```

▼図4　Wearアプリにおける新パーミッションモデル②

上で機能を実行するときの話です。Wearアプリにおいて、ほかに注意すべきポイントは、Handheldアプリと連携動作するケースです。HandheldアプリとWearアプリが同一アプリだとしても、ユーザからのパーミッション利用許可はそれぞれ別に取得する必要があります。HandheldもしくはWearのいずれか一方でのみ許可を得ていたとしても、他方では許可を得たことになりません(許可を得ていない機能をアプリが利用することはできません)。

ここでは例として、健康管理アプリ(Wear上でボディセンサーにより心拍数を測定し、Handheld上の外部ストレージにデータを格納して表示するもの)を題材として、連携動作時の振る舞いを解説します。

▶▶(A)Wearアプリ→Handheldアプリ側の許可取得

Wearアプリ上で心拍数を取得後、そのデータをHandheldアプリ側に送信し、外部ストレージに格納したいとします。Handheldアプリが外部ストレージへの書き込みを行うには、その許可をあらかじめ得る必要があります。そのため**図5**のとおり、Wearアプリ→Handheldアプリを起動し、必要に応じてHandheldアプリ側でユーザにパーミッションを必要とする理由を提示し、許可取得ダイアログを通じて、許可を得るという流れです。このような機能を実現しておけば、Wearアプリ→Handheldアプリの連携動作時にも外部ストレージを利用できます。

▶▶(B)Handheldアプリ→Wearアプリ側の許可取得

続いて、HandheldアプリからWearアプリに対して、ボディセンサー情報を取得するための要求を送信したいとします。Wearアプリがボ

▼図5　Wearアプリ→Handheldアプリの許可取得例

▼図6　Handheldアプリ→Wearアプリの許可取得例

ディセンサー情報を取得するには、その許可をあらかじめ得る必要があります。そのため**図6**のとおり、Handheldアプリ→Wearアプリを起動し、Wearアプリ側で許可を得るという流れです。Wearは画面が小さいこともあり、今回のようにHandheldアプリ経由で許可を取得する場合、Handheldアプリ側で許可が必要な理由を細かく説明しておくのも良いでしょう。このような機能を実現しておけば、Handheldアプリ→Wearアプリへの連携動作時にもボディセンサーを利用できるようになります。

◆ ◆ ◆

新パーミッションについて、もう1つ注意しておくことがあります。ここまでは、HandheldとWearがいずれもAndroid 6.0(APIレベル23)以上である場合の挙動について解説しました。ただ市場には、現時点ではHandheldとWearのいずれについても、Android 6.0以上も6.0未満も両方とも存在します。それゆえ、これら4つの組み合わせ(**表2**)を考慮した開発・テストを実施しておくことが望ましいでしょう。

▼表2　考慮すべきバージョンの組み合わせ

		Handheld	
		6.0未満	6.0以上
Wear	6.0未満	○	○
	6.0以上	○	○

新機能2：サウンド出力

Wear実機の一部にはスピーカーを搭載しているものがあります(Zen Watch 2、Huawei Watchなど)。これらの機種では、Wear単体でサウンド再生が可能となりました。これによりWearアプリにおいて、情報を受信したときのサウンドや、ゲームの効果音の出力などに利用できます。開発者が本機能を活用すれば、よりリッチなWearアプリを開発できます。また、本機能に対応している実機では、設定内から音の有効・無効や音量をカスタマイズできます(**図7**)。

それでは、サウンドを再生するコードを実装しましょう。今回はサウンドファイルとして、アプリのリソース(res/raw)フォルダ内に、mp3ファイル(sample.mp3)を入れておき、これを再生する実装を紹介します。**リスト2**のコードにより実現できます。

Handheld向けのサウンド再生と同様です。詳細は割愛しますが、MediaPlayerを用いて実装しています。公式サイトの解説[注1]も参考にしてください。

最後にサウンドの停止処理です。先ほどと同様に、**リスト3**のコードにより実現できます。

新機能3：マルチスクリーン(円形・非円形)対応補助

ご存じのとおり、Wear実機のスクリーンには円形もあれば矩形もあります。アプリを開発するときは、いずれの形状でもレイアウトが崩れることなく画面が表示されるよう配慮する必要があります。しかし、Android 6.0以上では、“-round”と“-notround”という新しいリソース

注1) Media Playback
http://developer.android.com/guide/topics/media/mediaplayer.html

▼リスト2　サウンド出力（開始）の実装

```
private MediaPlayer mMediaPlayer;

private void playMusic() {
    if (mMediaPlayer == null) {
        mMediaPlayer = MediaPlayer.create(this, R.raw.sample);
        mMediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mp) {
                    Log.d(TAG, "再生が完了しました。");
            }
        });
    }
    mMediaPlayer.start();
}
```

▼リスト3　サウンド出力（停止）の実装

```
private void stopMusic() {
    if (mMediaPlayer != null) {
        mMediaPlayer.stop();
        mMediaPlayer.release();
        mMediaPlayer = null;
    }
}
```

▼図7　Wearのサウンド設定

修飾子をリソースフォルダに利用できるようになりました。

具体的には、円形と非円形で異なる画像やレイアウト、文字列を利用するといったことが容易に実現できます（**図8**）。開発者が本機能を活用することで、アプリのマルチスクリーン対応を実施しやすくなったと言えるでしょう。具体的な例として、**リスト4**のとおり、TextViewとImageViewを含む画面を用いて実装を紹介します。

円形と非円形で内容を切り替えるには、**図9**のとおりファイルを格納します。文字列については、strings.xmlを“values-round”と“values-notround”フォルダに格納します。画像については、image.pngを“mipmap-round-○○”と“mipmap-notround-○○”フォルダに格納します。○○にはldpi/mdpi/hdpi/xhdpiなどを設定します（round/notroundと併用できます）。これにより、**リスト4**内の"@string/textChangable"と"@mipmap/image"は、実行デバイスの形状により円形か非円形のうち該当する内容が設定されることになります。

円形と非円形の両サポートと言えば、WatchViewStubやBoxInsetLayout、WearableFrameLayoutなどのWear用UIライブラリを思い浮かべる方がいらっしゃるかもしれません。これらはAndroid 6.0においても以前のバージョンと同様に利用可能です。今回の“round/notround”修飾子は、容易にリソースを切り替え可能とするしくみです。そのためさまざまな画面形状をサポートする場合、従来のWear用UIライブラリと併用するのが良いでしょう。

注意すべきことが1つあります。本修飾子は、Android 6.0（APIレベル23）以上でしか利用できません。本稿執筆時点では、すべてのWear実機がAndroid 6.0にアップデートされている訳ではありません。APIレベル22以下の端末が存在する間は、本機能は補助的な利用にとどめ、下位バージョンの端末でもアプリが正常に動作するよう配慮しておきましょう。

そのほかの新機能

そのほかにも、Intel x86（TAG Heuer Connectedなどで採用されているCPUのアーキテク

▼リスト4　円形・非円形共通のレイアウトXML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.
android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/textChangable" />

    <ImageView
        android:id="@+id/image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitCenter"
        android:src="@mipmap/image" />
</LinearLayout>
```

チャ)がサポートされました。Javaコードだけ でWearアプリを実装する場合は考慮不要です が、NDKを用いた実装が含まれる場合は、各 CPUアーキテクチャに対応したライブラリを それぞれ準備するといった対応が必要なことに 注意しましょう。

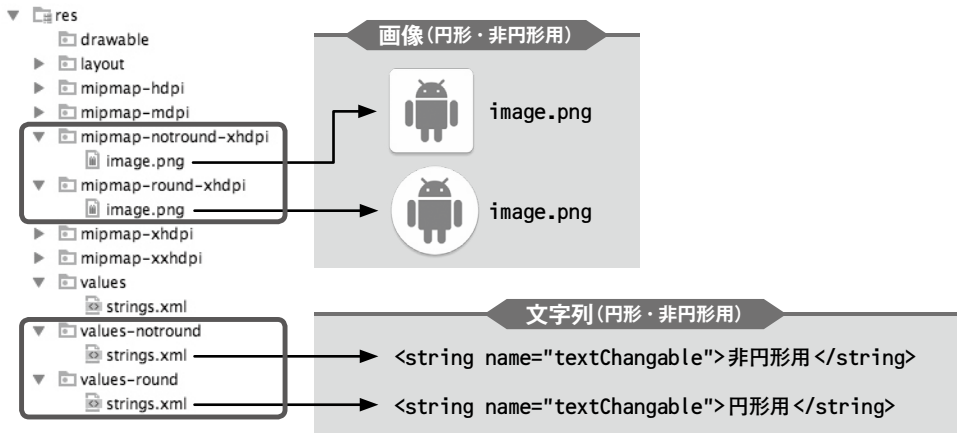
ほかにもWear実機で新しいジェスチャー操作に対応しました。具体的には、腕を振り下げることで通知の詳細情報を表示(タップ操作に相当)したり、腕を振り上げることで通知からウォッチフェイスに戻る(前の画面に戻る操作に相当)といった操作が可能となりました。ユーザがこれまで以上にWear端末を便利に使えるようになりましたね。

▼図8　円形・非円形別の画面

おわりに

Android Wearの概要に加え、WearのAndroid 6.0対応に伴うさまざまな新しい機能について紹介しました。Android Wearでアプリを作ってみようと感じていただけた方がいらっしゃるとうれしいかぎりです。またWearアプリ開発をする際は、拙著『Android Wearアプリ開発入門』(技術評論社刊)をあわせて参照いただけると幸いです。同書には、今回誌面の都合で紹介できなかったWearアプリ開発の具体的な作り方や、Wearならではの注意点、具体的なコード例を豊富に掲載しています。読者の皆さまもWearアプリを開発して、この世界を一緒に盛り上げていただけると大変うれしいです。では、またお目にかかる日まで！ SD

▼図9　リソースファイルの配置



セキュリティ対策はまずここから！

フリーで始めるサーバのセキュリティチェック

後編「OpenVASによる脆弱性スキャン」

ご注意　本稿に記載された内容を管理下または許可された環境以外に実施した場合に不正アクセス行為と判断され、法的措置をとられる可能性があります。ご自身の管理下または管理者より許可を取った環境に対してのみ実施してください。また、セキュリティチェックにより、対象となる環境でサービス停止などの影響が出てしまう可能性があります。事前に環境のバックアップを行うなど、何か問題が発生した場合に即時対応できるように注意を払ってください。

Author 小河 哲之（おがわ さとし） Twitter @number3to4 三井物産セキュアディレクション㈱

フリーでやろうぜ! セキュリティチェック

前編はNmapによるポートスキャンについて紹介しました。後編はサーバやネットワーク機器などに対するOpenVASを用いた脆弱性スキャンについて紹介します。脆弱性スキャンは、対象とする機器に存在するセキュリティ上の問題点を洗い出すために行います。脆弱性スキャンは一般的に2つの方法から存在する脆弱性を洗い出します。

①疑似攻撃などを行い、その挙動から脆弱性の有無を判定する方法
②レスポンス中に含まれるアプリケーションのバージョン情報をもとに脆弱性の有無を判定する方法

①は脆弱性ごとの検証用コードなどを用いて、脆弱性の有無を判定します。検証用コードがない脆弱性も多く存在し、その場合は①の方法での検出はできません。②は、①で検出できない脆弱性も検出することが可能です。製品にもよりますが、製品の提供元サイトのセキュリティ関連ページなどに脆弱性情報が記載されている場合があります。National Vulnerability Database[注1]やSecurity Focus[注2]、JVN iPedia[注3]などのサイトで製品に存在する脆弱性が公開されており、製品名およびバージョンなどの条件で検索できます。①、②のどちらの方法も既知の脆弱性を洗い出すためのもので、ゼロデイ（未知の脆弱性）は検出できません。

洗い出された脆弱性を優先度に応じて対応するために、どの機器でどのような脆弱性が検出されたのか適切に管理する必要があります。共通脆弱性識別子CVE（Common Vulnerabilities and Exposures）などの識別子を利用することで効率よく管理できます。多くの脆弱性はCVEが割り当てられており、一意に特定できます。CVE以外にもIPAとJPCERT/CCが共同で運営しているJVN iPedia（JVNDB）で利用されている識別子などもあります。JVNDBは、日本国内で利用されている製品の脆弱性情報を中心に蓄積しています。これらの識別子を用いて、脆弱性を管理していくことで、対応すべき脆弱性を可視化でき、対応漏れが起こる可能性を軽減できます。

後編では、脆弱性スキャンにOpenVASを用います。OpenVASは、前編で紹介したNmapと同様にオープンソース（GNU GPL）のソフトウェアです。ブラウザで設定や操作ができ、脆弱性スキャンだけではなく、スキャン対象機器で検出された脆弱性を管理する機能も有しています。「http://www.openvas.org/」からソース、パッケージやOVAイメージがダウンロードできます。新バージョンはOpenVAS-8です。

どんな環境が必要?

前編同様、次の2つの仮想環境を使用します。

注1）URL https://web.nvd.nist.gov/view/vuln/search
注2）URL http://www.securityfocus.com/bid
注3）URL http://jvndb.jvn.jp/

- **Kali Linux**[注4]
  セキュリティチェックを実施する環境。IPアドレスを192.168.1.1とする
- **Metasploitable2**[注5]
  セキュリティチェックの対象となる環境。IPアドレスを192.168.1.100とする

OpenVASは**表1**のとおり、大きく4つの機能から構成されています。

Kali Linux 2016.01でのインストールは次の手順で行います。

```
# apt-get update
# apt-get upgrade
# apt-get install openvas
```

インストール完了後、OpenVASのセットアップを行います(**図1**)。セットアップが完了すると管理者ユーザであるadminのパスワードが表示されるので、一時的にひかえておいてログイン後変更してください。もし、パスワードをひかえるのを忘れた場合は、openvasmdコマンドでパスワードを再設定できます(**図2**)。

また、Webインターフェースはデフォルトではループバックアドレス(127.0.0.1)の9392ポートで稼働しています。外部からアクセスする場合、「/lib/systemd/system/greenbone-security-assistant.service」ファイルの9行目を**リスト1**のように変更する必要があります。リッスンするIPアドレスは状況に合わせて変更してください。

インストールされたOpenVASは**図3**のとおり起動します。

セットアップが完了したら、openvas-check-setupコマンドで正常に終了したかを確認してください(**図4**)。"It seems like your OpenVAS-8 installation is OK."が表示されれば正常にセットアップが完了しています。

▼図1　OpenVASのセットアップ

```
# openvas-setup
[i] This script synchronizes an NVT collection with the ➥
'OpenVAS NVT Feed'.
[i] The 'OpenVAS NVT Feed' is provided by 'The OpenVAS ➥
Project'.
(..中略..)
Rebuilding NVT cache... done.
User created with password 'e562cdd0-afbc-4579-a7af-➥
794ab6faf85d'.
```

adminのパスワード

▼図2　パスワードの再設定

```
# openvasmd --user=admin --new-password=パスワード
```

▼リスト1　外部からWebインターフェースにアクセスするための設定(greenbone-security-assistant.service)

```
変更前
ExecStart=/usr/sbin/gsad --foreground --listen=127.0.0.1 ➥
--port=9392 --mlisten=127.0.0.1 --mport=9390
変更後
ExecStart=/usr/sbin/gsad --foreground --listen=リッスンする➥
IPアドレス --port=9392 --mlisten=127.0.0.1 --mport=9390
```

▼図3　OpenVASの起動

```
# /etc/init.d/redis-server start
# openvas-start
Starting OpenVas Services
```

▼表1　OpenVASのおもな機能

機能名	概要
OpenVAS Scanner	脆弱性スキャンを行うための機能
OpenVAS Manager	Scannerの制御や脆弱性情報の管理、OpenVASのユーザ管理などを行う機能
Greebone Security Assistant	Webインターフェースを提供する機能
OpenVAS CLI	Managerを操作するバッチプロセスを生成するためのコマンドラインを提供する機能

注4) Debianベースのディストリビューションで最新バージョンは2016.01。「URL https://www.kali.org/」からダウンロード可能。Kali Linux 64 bitを使用する。

注5) 複数の脆弱性が包含された環境で、脆弱性診断などのテスト環境として活用される。VMwareのイメージが公開されており、「URL https://sourceforge.net/projects/metasploitable/」からダウンロードできる。

CentOS 7でのインストールは**図5**の手順になります。CentOS 7では、デフォルトでSELinuxが有効になっていますが、無効にする必要があります。有効になっている場合、openvas-check-setupコマンドでSELinuxを無効にするようエラーが出力されます。

Let's 脆弱性スキャン

OpenVASを用いた脆弱性スキャンを行うために、Webインターフェースにログインします。「https://設定したIPアドレス:9392」にアクセスします。自己署名証明書を使用しているため、ブラウザでアクセスした際に警告が表示されます。自己署名証明書を使用することのリスクについては本稿では割愛しますが、今回は警告を無視して進みます。ログインすると**図6**の画面が表示されます。

OpenVASでは画面上のアイコンを使って各種の操作を行います。使用するおもなアイコンを**表2**に挙げます。

OpenVASでの脆弱性スキャンはImmediately scan(方法①)とカスタマイズされたスキャン(方法②)の2つがあります(**図7**)。Immediately scanは、IPアドレスまたはホスト名を指定するだけで脆弱性スキャンを実施できます。とてもシンプルに実施できますが、UDPに対する脆弱性スキャンを行わず、あらかじめ登録されたTCPの4481ポート分のみのスキャンになります。カス

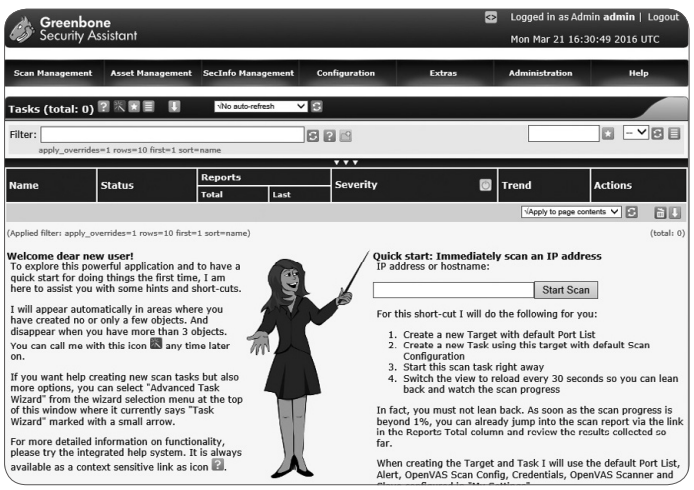
▼図4　セットアップが正常終了したかの確認

```
# openvas-check-setup
openvas-check-setup 2.3.3
  Test completeness and readiness of OpenVAS-8
（..中略..）
It seems like your OpenVAS-8 installation is ➡
OK.
（..略..）
```

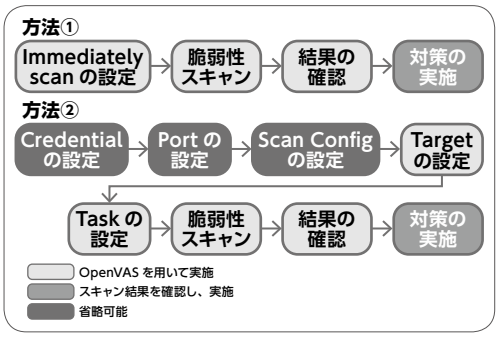
▼図5　CentOS 7でのインストール手順

```
# yum install wget bzip2
# wget -q -O - http://www.atomicorp.com/➡
installers/atomic | sh
# yum upgrade
# yum install openvas
# openvas-setup
# openvasmd --rebuild
# openvas-check-setup
```

▼図6　OpenVASのログイン後



▼図7　脆弱性スキャンの方法



▼表2　OpenVASで使用するおもなアイコン

アイコン	アクション
	新規登録
	変更
	コピー
	一覧表示
	削除
	ダウンロード
	更新

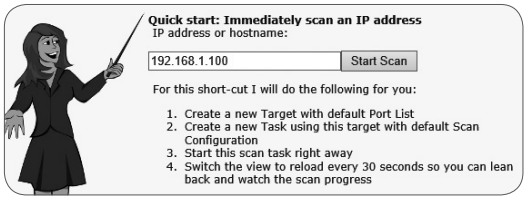
タマイズされたスキャンは、Credential(認証情報)やスキャン対象のPortの設定などを個別に定義し、脆弱性スキャンを行う方法です。Credentialの設定は省略することが可能です。また、Portの設定やScan Configの設定も、テンプレートを用いることで個別に設定しなくても脆弱性スキャンを行えます。

Immediately scanによるスキャン

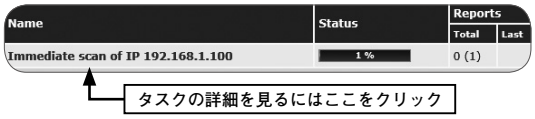
Immediately scanはIPアドレスまたはホスト名を入力し、"Start Scan"をクリックすると開始されます(**図8**)。"Status"カラムに脆弱性スキャンの進捗状況が表示されます(**図9**)。完了すると"Done"と表示されます。

図9の"Name"カラムのリンクをクリックすると実施した日時や実施した脆弱性スキャンの設定を確認できます(**図10**)。スキャン結果を見るには、"Results"項目の数字をクリックしてください。検出された脆弱性が一覧表示されます(**図11**)。各カラム名をクリックすることで項目ごとにソートできます。検出された脆弱

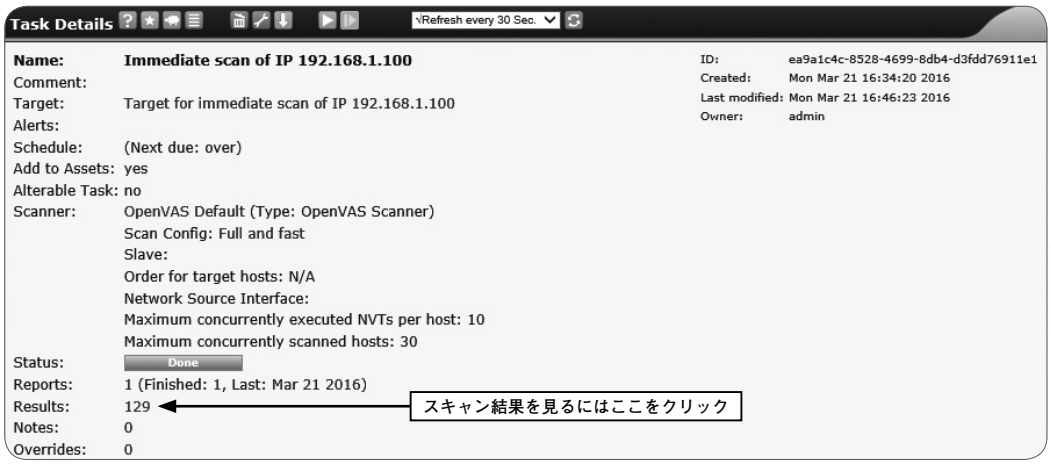
▼図8　Immediately scan



▼図9　Immediately scanの進捗状況



▼図10　Taskの詳細



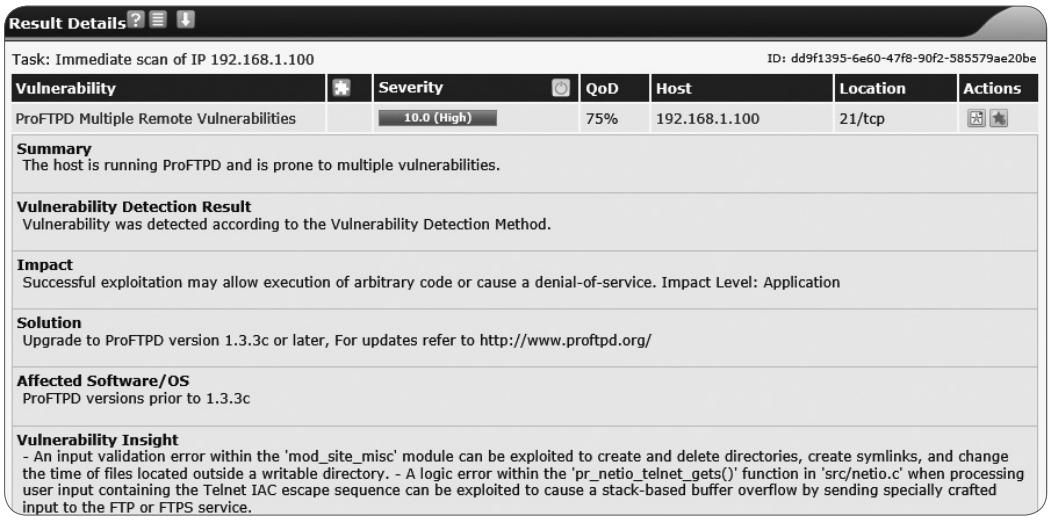
▼図11　検出結果

性の内容を参照するには、“Vulnerability”カラムの脆弱性名をクリックします。脆弱性の概要や対策、参考URLなどを確認できます(**図12**)。

脆弱性の結果を俯瞰して確認することに適した機能としてReportsがあります。“Scan Management”－“Reports”をクリックすると、脆弱性スキャンの結果サマリを一覧で確認できます(**図13**)。

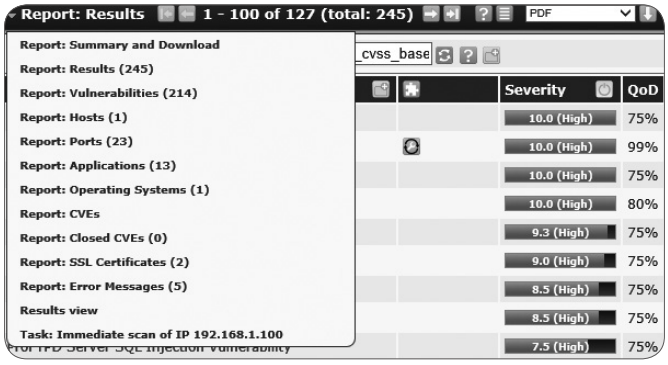
結果の分析を補助する機能も用意されています。“Date”カラムの日付をクリックし、左上にある“Report Results”をフォーカスするとサブメニューが表示されます(**図14**)。ここから検出された脆弱性や稼働しているポート、アプリケーションなど一覧で確認できます。 をクリックすれば、横のメニューで指定した出力形式でレポートを出力できます(**図15**)。

▼図12　脆弱性の詳細

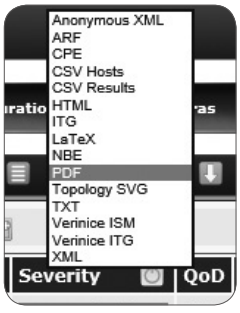


▼図13　脆弱性のサマリ

▼図14　結果の分析



▼図15　レポートの出力



カスタマイズされたスキャン

Immediately Scanは簡単に脆弱性スキャンを実施できますが、スキャン範囲やシグネチャ[注6]は一部のみとなります。スキャン範囲やシグネチャをすべて実施するためには、設定をカスタマイズする必要があります。設定をカスタマイズするには、Taskを定義する必要があります(**図16**)。Taskは、Target(スキャンを行う対象ホスト)とScan Config(シグネチャ)を設定することで定義します。Targetはさらに Credential(認証情報)とPort List(対象ポート)を設定することで定義します。

Credentialの設定

Credentialを設定すると、OpenVASは設定されたCredentialを使用して認証を行い、システム情報を確認します。設定したCredentialはSSH、SMB、ESXiで利用できます。Credentialの設定は省略することが可能ですが、Credentialを設定して脆弱性スキャンを実施した場合、外部からでは検出が困難な脆弱性も洗い出すことができるため、脆弱性の検出数は増える可能性が高まります。

Credentialの設定を行うには"Configuration"－"Credentials"をクリックし、**図17**の★をクリックします。**図18**の画面が開いたら、"Name"には適当な名前を入力し、"Login"にユーザ名を"Password"にパスワードを入力します[注7]。

Port Listの設定

Portの設定では、スキャン対象とするポート範囲を指定します。OpenVASではいくつかテンプレートが用意されています。テンプレートを直接変更することはできませんが、コピーして変更することは可能です。また、テンプレートを使用せずに新しく作成することも可能です。"Configuration"－"Port Lists"をクリックすると**図19**の画面になります。テンプレートは9つあり、それぞれのスキャンを行うポート数が一覧で表示されます。

新しく登録する場合は★をクリックしてください。テンプレートをコピーする場合、該当のテンプレートの📄をクリックしてください。**図20**の画面になったら、"Name"には適当な名前を入力し、"Port Ranges"にはスキャンするポートを入力します。TCPは`T:`以降に、**表3**のようにカンマ区切りかハイフンで指定します。UDPは`U:`以降にTCP同様に指定します。

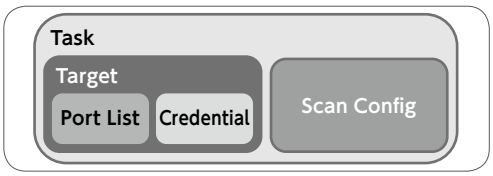
Scan Configの設定

Scan Configは実施する脆弱性スキャンのシグネチャセットを管理するための機能で、

注6) 脆弱性を検知するためのルールや定義。

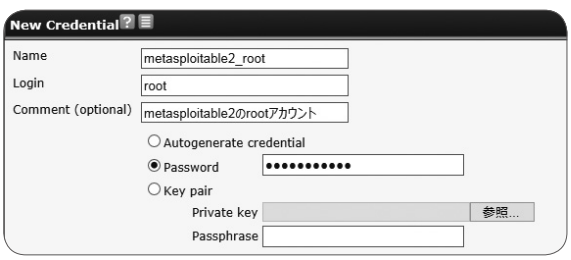
注7) `mkdir /var/lib/openvas/gnupg`を事前に実行する必要があります。

▼図16 Task定義に必要な設定項目の関係性(イメージ図)



▼図17 Credentialの一覧

▼図18 Credentialの設定



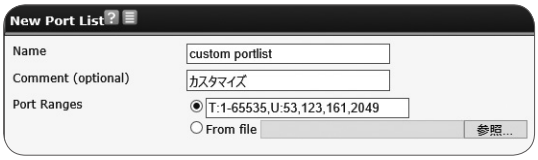
“Configuration”－“Scan Configs”をクリックすると内容が確認できます(**図21**)。デフォルトで8つテンプレートが用意されていますが、Port List同様、テンプレートを直接変更することはできません。そのため、Scan Configをカスタマイズする場合は、テンプレートの  をクリックし、コピーしたものを変更してください。通常、“Full and very deep”を使用することを推奨します。“Full and very deep ultimate”は、“Full and very deep”に危険なシグネチャを追加したテンプレートであるため、“Full and very deep ultimate”を実施する場合は細心の注意を払ってください。

テンプレートに登録されているポートスキャンはConnect Scanで実施されます。これをSYN Scanに変更する場合、Scan Configをカスタマイズする必要があります。ベースとするテンプレート“Full and very deep”をコピーし、コピーされたScan Configの  をクリックします。Scan Configは複数の設定項目から構成されており、その中のLaunch Nmap for Network ScanningおよびNmap(NASL wrapper)のTCP

▼図19　Port Listの一覧

Name	Port Counts			Actions
	Total	TCP	UDP	
All IANA assigned TCP 2012-02-10	5625	5625	0	
All IANA assigned TCP and UDP 2012-02-10	10988	5625	5363	
All privileged TCP	1023	1023	0	
All privileged TCP and UDP	2046	1023	1023	
All TCP	65535	65535	0	
All TCP and Nmap 5.51 top 100 UDP	65634	65535	99	
All TCP and Nmap 5.51 top 1000 UDP	66534	65535	999	
Nmap 5.51 top 2000 TCP and top 100 UDP	2098	1999	99	
OpenVAS Default	4481	4481	0	

▼図20　Portのカスタマイズ



▼表3　ポートの指定方法

複数対象の指定方法	指定例
カンマ区切りによる複数指定	T:21,22,25,80,443,U:53,123,161
ハイフンによる範囲指定	T:1-65535,U:1-1023

▼図21　Scan Configの一覧

Name	Families		NVTs		Actions
	Total	Trend	Total	Trend	
Discovery (Network Discovery scan configuration.)	20		1721		
empty (Empty and static configuration template.)	0		0		
Full and fast (Most NVT's; optimized by using previously collected information.)	59		46186		
Full and fast ultimate (Most NVT's including those that can stop services/hosts; optimized by using previously collected information.)	59		46186		
Full and very deep (Most NVT's; don't trust previously collected information; slow.)	59		46186		
Full and very deep ultimate (Most NVT's including those that can stop services/hosts; don't trust previously collected information; slow.)	59		46186		
Host Discovery (Network Host Discovery scan configuration.)	2		2		
System Discovery (Network System Discovery scan configuration.)	6		28		

scanning techniqueのオプションを変更します。それぞれの  をクリックし、“TCP scanning technique”を“connect()”から“SYN”に変更します(**図22**)。

Targetの設定

Targetの設定は、脆弱性スキャンを行う対象に関する設定になります。“Configuration”－“Targets”をクリックし、Targetの一覧画面の  をクリックすると、**図23**の画面になります。“Name”は適当な名前を入力してください。“Hosts”にはスキャン対象のIPアドレスまたはホスト名を入力します。複数の対象を指定する場合、CIDRやカンマ区切り、ハイフンなどで指定もできます。“Port List”は、一覧の中から選択します。事前にCredentialを設定している場合は、“SSH”、“SMB”、“ESXi”のそれぞれで選択します。

Taskの設定

Taskの設定は、脆弱性スキャンを行うタスクを作成します。“Scan Management”－“Tasks”をクリックし、Taskの一覧画面の  をクリックすると、**図24**の画面になります。“Name”は適当な名前を入力します。“Scan Targets”はTargetの設定で作成した対象を選択します。“Scan Config”は、スキャンのテンプレートまたはカスタマイズしたものの中から選択します。

Taskを登録したらTask一覧画面に追加されます。一覧の“Name”カラムのリンクをクリックするとTaskの詳細を確認できます(**図25**)。**図25**の  をクリックすると脆弱性スキャンが開始されます。スキャン結果やレポートの参照方法はImmediately scanと同じため割愛します。

脆弱性の管理

OpenVASには検出された脆弱性を管理するための機能が用意されています。“Asset Management”－“Hosts”で脆弱性スキャンを実施したホストごとに検出された脆弱性をサマリで確認できるため、対象の優先度、脆弱性の重大度

▼図22　Scan Configの変更

TCP scanning technique	○ connect() ◉ SYN ○ ACK ○ FIN ○ Window ○ Maimon ○ Xmas tree ○ Null ○ SCTP Init ○ SCTP COOKIE_ECHO	connect()

▼図23　Targetの登録

New Target

Name	metasploitable2
Comment (optional)	
Hosts	◉ Manual 192.168.1.100 ○ From file 参照...
Exclude Hosts	
Reverse Lookup Only	○ Yes ◉ No
Reverse Lookup Unify	○ Yes ◉ No
Port List	All TCP and Nmap 5.51 top 1000 UDP
Alive Test	Scan Config Default
Credentials for authenticated checks (optional):	
SSH	-- on port 22
SMB	--
ESXi	--

▼図24　Taskの登録

New Task

Name	metaploitable2
Comment (optional)	
Scan Targets	metasploitable2
Alerts (optional)	-- +
Schedule (optional)	-- ☐ Once
Add results to Asset Management	◉ yes ○ no
Alterable Task	○ yes ◉ no
Scanner	
◉ OpenVAS Scanner	OpenVAS Default
Scan Config	Full and very deep
Slave (optional)	--
Network Source Interface	
Order for target hosts	Sequential
Maximum concurrently executed NVTs per host	4
Maximum concurrently scanned hosts	20

に応じて対応する際にとても便利な機能です(**図26**)。"Last Report"カラムの日付部分をクリックすると該当機器で実施された一番新しいレポートを参照できます。

検出された脆弱性に対して、メモを残したり脆弱性の重大度を再評価したりすることができます。脆弱性一覧画面(**図11**)の"Actions"カラムにあるをクリックすると、該当の脆弱性に対してメモを残せます(**図27**)。たとえば「サービスを利用していないのでアンインストール予定」などの対応方針を記載し、複数名での共有や対応したログとして保存できます。メモがある脆弱性は脆弱性一覧にが表示されます。該当の脆弱性をクリックすると記載したメモを確認できます(**図28**)。

OpenVASが誤って、稼働しているアプリケーションとは異なるアプリケーションの脆弱性を検出する場合があります。その場合、検出された脆弱性を誤検知(False Positive)として再設定することができます。"Actions"カラムにあるをクリックし、**図29**の"New Severity"を"False Positive"に変更します。また、あとで変更した理由がすぐわかるように"Text"に変更した理由などを記載しておくことを推奨します。

▼図25　Taskの詳細

Task Details　Refresh every 30 Sec.

Name:	**metasploitable2**	ID:	7f1882be-34c6-4016-a9bc-ccd3adbc007e
Comment:		Created:	Sun Apr 3 09:21:46 2016
Target:	metasploitable2	Last modified:	Sun Apr 3 09:21:46 2016
Alerts:		Owner:	admin
Schedule:	(Next due: over)		
Add to Assets:	yes		
Alterable Task:	no		
Scanner:	OpenVAS Default (Type: OpenVAS Scanner)		
	Scan Config: Full and very deep		
	Slave:		
	Order for target hosts: Sequential		
	Network Source Interface:		
	Maximum concurrently executed NVTs per host: 4		
	Maximum concurrently scanned hosts: 20		
Status:	New		
Reports:	0 (Finished: 0)		
Results:	0		
Notes:	0		
Overrides:	0		

▼図26　ホストごとのサマリ

Filtered Hosts　1 - 4 of 4

IP	High	Medium	Low	Log	Last Report	OS	Ports	Apps	Distance	Prognosi
192.168.1.100	23	38	5	97	Apr 3 2016		31	25	1	Low
192.168.1.180	0	8	1	65	Apr 3 2016		11	4	1	Low
192.168.1.201	2	8	2	47	Apr 3 2016		9	4	1	Log
192.168.1.254	2	4	1	17	Apr 3 2016		4	1	1	

Total: 4

▼図27　メモの登録

New Note

NVT Name	ProFTPD Multiple Remote Vulnerabilities
Active	◉ yes, always ○ yes, for the next 30 days ○ no
Hosts	○ Any ◉ 192.168.1.100
Port	○ Any ◉ 21/tcp
Severity	○ Any ◉ > 0.0
Task	○ Any ◉ metasploitable2_ssh
Result	◉ Any ○ Only the selected one (71ffa2e6-9e5c-4137-a6f3-986956886aa0)
Text	サービスを利用していないのでアンインストール予定

▼図29　脆弱性の再評価

New Override

NVT Name	TightVNC ClientConnection Multiple Integer Overflow Vulnerabilities (...
Active	◉ yes, always ○ yes, for the next 30 days ○ no
Hosts	○ Any ◉ 192.168.1.100
Port	○ Any ◉ 5900/tcp
Severity	○ Any ◉ > 0.0
New Severity	◉ False Positive ○ Other:
Task	○ Any ◉ metasploitable2_ssh
Result	◉ Any ○ Only the selected one (a68f6a2f-2f86-4546-a71f-e9d81e186893)
Text	使用していないアプリケーションで誤検知であるため、変更。

▼図28　メモの確認

References

CVE: CVE-2010-3867, CVE-2010-4221
BID: 44562
CERT: DFN-CERT-2011-0368 , DFN-CERT-2010-1551 , DFN-CERT-2010-1550 , DFN-CERT-2010-
Other: http://secunia.com/advisories/42052
http://bugs.proftpd.org/show_bug.cgi?id=3519
http://bugs.proftpd.org/show_bug.cgi?id=3521
http://www.zerodayinitiative.com/advisories/ZDI-10-229/

Note

サービスを利用していないのでアンインストール予定
Last modified: Sun Apr 3 12:25:32 2016.

コマンドライン(CLI)によるスキャン

OpenVASには一部の機能でCLIが用意されています。Credentialの設定やPortの設定などはブラウザで実施する必要がありますが、Taskの登録や脆弱性スキャンはCLIで実施できます(**図30**)。

まずはOpenVAS内でのTargetの内部キーを確認します(**図31**)。次にScan Configの内部キーを確認します(**図32**)。

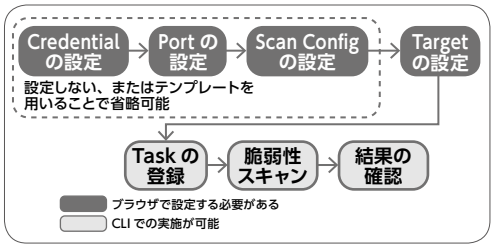
TargetとScan Configの内部キーの確認を終えたら、Taskを登録します(**図33**)。この場合、Targetの内部キーは"071c1b78-b67a-4141-a6bd-a0e86c1f1860"を、Scan Configは Full and very deep Customを使用するため、内部キーは"c1101209-bb41-422e-8dae-1a5b1ef37275"を指定します。タスク名は任意の名前を指定します。内部キーの値を間違えているなどして登録が失敗した場合は、"Failed to create task."が出力されます。登録が成功するとTaskの内部キーが出力されます。

登録されたTaskを実行します(**図34**)。この場合、Taskの内部キーは"bb40a233-8d8e-4e8e-93b3-c9cf8026d438"を指定します。正常に実行された場合、Reportの内部キーが出力されます。

-GオプションでTaskの状況を確認できます(**図35**)。また、-Gオプションの引数にTaskの内部キーを指定するとTaskの詳細情報を参照でき、Reportの内部キーも確認できます。

-Rオプションで脆弱性スキャンの結果を確認できます(**図36**)。デフォルトではXML形式で出力されます。ReportのFormat(出力形式)を変更することもできます(**図37**)。まず、-Fオプ

▼図30　CLIでの対応範囲



▼図31　Targetの内部キーの確認

```
root@kali:~# omp -u admin -T
Enter password:
b493b7a8-7489-11df-a3ec-002264764cea  Localhost
071c1b78-b67a-4141-a6bd-a0e86c1f1860  metasploitable2
(..略..)
```

Target「metasploitable2」の内部キー

▼図32　Scan Configの内部キーの確認

```
[root@localhost ~]# omp -u admin -g
Enter password:
(..中略..)
c1101209-bb41-422e-8dae-1a5b1ef37275 Full and very deep Custom
74db13d6-7489-11df-91b9-002264764cea Full and very deep ultimate
(..略..)
```

Scan Config「Full and very deep Custom」の内部キー

▼図33　Taskの登録

```
[root@localhost ~]# omp -u admin -C -n タスク名 --target=071c1b78-b67a-4141-a6bd-a0e86c1f1860 ↵
-c c1101209-bb41-422e-8dae-1a5b1ef37275
Enter password:
bb40a233-8d8e-4e8e-93b3-c9cf8026d438
```

Scan Configの内部キー / Targetの内部キー / Taskの内部キー

▼図34　Taskの実行

```
[root@localhost ~]# omp -u admin -S bb40a233-8d8e-4e8e-93b3-c9cf8026d438
Enter password:
1202b02f-71fa-419b-b30d-0a8c2b519bb0
```

Taskの内部キー / Reportの内部キー

ションで変更したいFormatの内部キーを確認します。-fオプションで指定されたFormatの内部キーの形式で結果が出力されます。

運用について

ここまでOpenVASを用いた脆弱性スキャンを紹介してきましたが、うまく活用するための3つのポイントを見ていきましょう。

・スキャン対象を決める
・スキャンの実施タイミングを決める
・スキャン結果の対応方針を決める

まずは、スキャン対象を決めます。すべてのサーバおよびネットワーク機器を対象とすることを推奨しますが、スキャン対象が多い場合、DMZ(DeMilitarized Zone)にあるサーバなどを対象とし、それ以外は重要度に応じてスキャン対象を決定することが良いでしょう。

次に、スキャンの実施タイミングですが、公開前の機器はカットオーバー前に実施し、検出された脆弱性に対して適切に対応する必要があります。すでに公開済みの機器は定期的なスキャンを推奨します。DMZにある機器は4半期に1回、それ以外の機器は年1回など重要度に応じて実施するなど検討してください。また、Open

▼図35　Taskの状況確認

```
[root@localhost openvas]# omp -u admin -G
Enter password:
bb40a233-8d8e-4e8e-93b3-c9cf8026d438  Running 42%  task1
[root@localhost openvas]# omp -G bb40a233-8d8e-4e8e-93b3-c9cf8026d438 -u admin
Enter password:                                    ← Taskの内部キー
bb40a233-8d8e-4e8e-93b3-c9cf8026d438  Done         task1
  1202b02f-71fa-419b-b30d-0a8c2b519bb0  Done     10  19   3  51  2016-04-05T14:13:11Z
  ↑ Reportの内部キー
```

▼図36　脆弱性スキャン結果の確認

```
[root@localhost openvas]# omp -u admin -R 1202b02f-71fa-419b-b30d-0a8c2b519bb0
Enter password:                                    ← Reportの内部キー
<report id="1202b02f-71fa-419b-b30d-0a8c2b519bb0" format_id="a994b278-1f62-11e1-96ac-
406186ea4fc5" extension="xml" type="scan" content_type="text/xml">
(..略..)
```

▼図37　ReportのFormatの変更(CSV形式で出力する例)

```
↓Formatの内部キーの確認
[root@localhost openvas]# omp -u admin -F
Enter password:                                    ← Format「CSV Results」の内部キー
  (..略..)
c1645568-627a-11e3-a660-406186ea4fc5  CSV Results
6c248850-1f62-11e1-b082-406186ea4fc5  HTML
  (..略..)
↓Formatの内部キーを指定してスキャン結果を出力
[root@localhost ~]# omp -R 1202b02f-71fa-419b-b30d-0a8c2b519bb0
-f c1645568-627a-11e3-a660-406186ea4fc5 -u admin   ← Reportの内部キー
Enter password:                                    ← Format「CSV Results」の内部キー
IP,Hostname,Port,Port Protocol,CVSS,Severity
  (..略..)
```

VASのシグネチャの更新は不定期であるため、脆弱性スキャンを実施する前に“Administration”の“NVT Feed”、“SCAP Feed”、“CERT Feed”でシグネチャを更新してください。

3番目に、スキャン結果の対応方針について述べます。検出された脆弱性すべてに対して適切に対応することが好ましいのですが、現実的に困難なことが多いため、事前に指標を決めておくことが良いでしょう。たとえば、OpenVASで検出されたHigh、Mediumの脆弱性については対応するというように一定の水準を決めておくことを推奨します。

最後に

多くの脆弱性は、バージョンのアップデート、セキュリティパッチの適用または設定変更を実施することで対策できます。具体的な対策方法は脆弱性ごとに異なるため、対応するべき脆弱性の対策を確認のうえ、適切に実施してください。根本的な対策ではありませんが、攻撃を受けるリスクを低減できるため、可能な範囲でアクセス制限することを推奨します。

新しく発見された脆弱性はOpenVASでシグネチャが作成されるまで検出できないため、IPAやJPCERT/CCのサイトで注意喚起されているものを個別に対応してください。

使用しているソフトウェアのサポート期限についても適切に管理する必要があります。一般的なソフトウェアにはサポート期限が設定されており、期限の切れたソフトウェアは重大な脆弱性が発見されたとしても、開発元からセキュリティパッチなどの対策が提供されません。そのため、サポート期限の切れたソフトウェアを使用することがリスクになります。期限が切れる前に、別なソフトウェアに切り替える必要があります。設計、構築時にソフトウェアのライフサイクルについても考慮し、スケジュールなどを計画することが重要です。すでにサポート期限が切れたソフトウェアを使用している場合、ソフトウェアの切り替えを計画し早急に対応してください。また、切り替えが完了するまでの間、アクセス制限を厳しくするなどの軽減措置も併せて実施することを推奨します。

診断ツールも完璧ではないため、誤検知(False Positive)や検知漏れ(False Negative)が発生します。誤検知は脆弱性がないにもかかわらず脆弱性を報告することです。使用していないソフトウェアの脆弱性やすでに対策されている脆弱性を検出する場合があります。検知漏れは、脆弱性があるのに報告されていないことです。診断ツールの診断内容や判定方法が不十分であるために起こる場合があります。

誤検知を軽減するためには、検出された脆弱性の内容や、開発元のサイト、IPAなどから公開されている脆弱性情報を確認し、個別に該当するのか判断する必要があります。検知漏れを軽減するためには、複数の診断ツールを組み合わせるなどして総合的に判断する必要があります。これは非常にハードルが高いため、厳密に実施する必要がある場合などはセキュリティベンダに任せるなどの方法を検討したほうが良いと言えます。

◆ ◆ ◆

前編および後編でサーバのフリーで始めるセキュリティを紹介しましたが、セキュリティで重要なことの1つとして、適切な運用を継続することが挙げられます。これは非常に難しいことですが、紹介したツールなどを活用いただき、少しでもお役に立てれば幸いです。SD

参考サイト

- OpenVAS　http://www.openvas.org/
- Kali Linux　https://www.kali.org/
- IPA　https://www.ipa.go.jp/
- JPCERT/CC　https://www.jpcert.or.jp/
- National Vulnerability Database
  https://web.nvd.nist.gov/
- Security Focus　http://www.securityfocus.com/
- JVN iPedia　http://jvndb.jvn.jp/
- フリーでやろうぜ！セキュリティチェック！
  http://www.slideshare.net/zaki4649/free-securitycheck

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4回 実行計画の確認はSQLチューニングの基本中の基本

アプリケーション（AP）サーバでのループ処理の問題は今回で一段落。しかし、SQL一発でほしいデータを取得するようにしても、SQLのしくみを理解した書き方ができていないと、ほかで思わぬ失敗をします。そんなときの原因分析に役立つのが実行計画です。

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。

ぐるぐる系SQL、使ってませんか?

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。RDBへの問い合わせ言語のSQLは一般的な手続き型言語とは設計思想が違うため、間違った使い方をしてしまっているケースが多いことに長年問題意識を感じており、SQLへの理解を広めるためにこの連載を書いています。

前回まで書いてきたように大量データの抽出、集計、結合のような集合操作は本来SQLで処理するべきなのですが、このことが今でもあまり理解されていません。そして、その理解度を測るのに役立つのが「ぐるぐる系SQL」があるかどうかをチェックすることです。

「ぐるぐる系SQL」という言い得て妙な表現は『SQL実践入門』[注1]を出されているミックさんがブログで書かれていたもので、当連載でも前回までに書いてきたとおり「単純なSQL文をぐるぐる回すように呼び出してAP（アプリケーション）サーバ側のループ処理で集合操作をする用法」のことです。SQLを理解していない人はよくこれをやってしまうので、自社のシステムでも使っていないかどうかチェックしてみると良いでしょう。ぐるぐる系SQLに対して、集合操作をRDB側でやらせる方式を一発系SQLと呼ぶと、両者には**図1**のような違いがあります。

注1）ミック 著『SQL実践入門――高速でわかりやすいクエリの書き方』技術評論社、2015年

SQLから「逃げる」ほど問題は悪化する

一発系はSQLが複雑化するため、SQLをよく理解していないとメンテナンスができなくなります。そこで「わからないから使いたくない」と思ったときの逃げ道がぐるぐる系SQLで、これをすると仕様書とコードが複雑化し、当然その結果バグが増え工数もかさみ性能も出ない、という三重苦を引き起こすわけです。

要するに「逃げる」からかえって問題が起きます。本気で勉強すればSQLの集合指向の概念もそれほど難しいものではないし、そのほうが簡単になるので本来はきちんと学んで一発系SQLを使うべきです。会社としてもそんな技術者を育てなければいけないのですが、それをさぼって手続き型言語の延長で考えようとするから理解できないのが現実です。純粋に技術的に言え

ばぐるぐる系SQLを使ったほうがいい理由など1つもないのです。しかし、その主張がそのまま組織内で通用するとは限りません。

「わかっていない」ベテランの理解を得る方法とは?

「あんたSQLわかってませんやろ、なんて言ったらケンカになりますわ……」
と五代さんが悩んでいるのはそこです。浪速システムズの社内で別チームのリーダーが「SQLはよくわからんから使いたくない」という、一発系SQLから逃げるタイプでした。大道君のような若者は技術を素直に吸収してくれることが多いのですが、プライドだけが高いベテランは聞く耳を持たないことがあります。それをどう納得させるかは頭の痛い問題でした。

「ですんで……こんな方針でどうですか？」
と、五代さんがある提案を切り出しました。

- 【状況提示】……一発系SQLを使えば「こんなお困りを解決できますよ」というありがちなケースを示す
- 【根拠説明】……「手続き型言語とSQLを適材適所で使い分けるべきである」という技術的な理由は淡々と説明する
- 【友好姿勢】……困ったときはご相談ください、とにこやかに言う

五代さんが言うには、人は敵対的な相手の言うことは聞かないので、とにかく味方だと思わせること。そのためには「困ったときに役に立ってあげる」ことが大事で、それにはタイミングよく相談してもらう必要があるので、「こんなときは」という、いかにもありがちなケースをにこやかに伝えておく。今の段階では、技術的な理由については説明はするがゴリ押しはしない、と。

「困って泣きついてくるのを待つということですか。時間がかかりますよね……」

「しゃあないです。人って本気で困らんうちは考えを変えませんから」

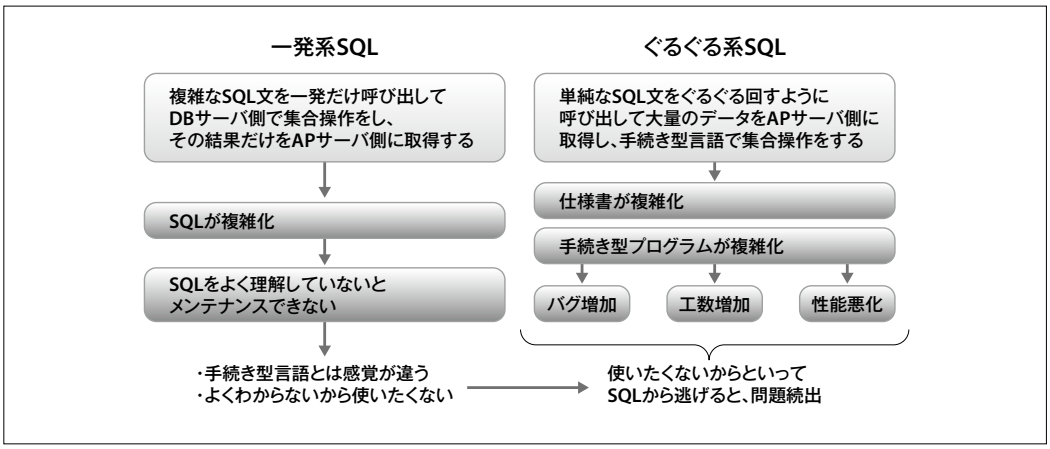
「泣きついてくるときって、デスマーチになってもう相当なダメージ食らっちゃってるときじゃないですか？」

「そうならないように根拠説明はしますよ。でもきっと聞く耳もたんでしょう。それで痛い目見るのは彼の自己責任ってもんです。幸いと言っちゃ何ですが、向こうは別チームです。うちらは困ったときにすがりつける手を用意しとけばいいんです」

冷たいようですが、それが一番現実的な考えに思えました。我々は彼が間違った判断をしても、直接すぐに被害を被るわけではありません。気長に考えてその方針でいくことにしました。

そして、「困って泣きついてくる」機会は実際、すぐにやってきたのです。

▼図1　一発系SQLとぐるぐる系SQLの比較



しくみを理解せずに使えば一発系も遅くなる

五代さんの方針に沿ってSQL嫌いのチームリーダーに「状況提示」「根拠説明」をしたものの、予想どおり一発系SQLの採用には難色を示したため、「友好姿勢」でいったん話を収めたその数日後のこと。再び大道君を通じてヘルプコールがかかってきました。

「今度はなんやねん？」

「月次請求処理だそうです」

バッチ処理の一部が遅いので調べてほしいということでした。いくつものSQLを発行する複雑なプログラムでしたが、少し調べてみると原因は1ヵ所に絞り込めました。問題が起きていたSQLの細部を削って単純化したエッセンスだけを載せたものが**リスト1**です。

「なるほどこれか。さて、こいつが遅いのはなぜだと思う？」

「これは、ぐるぐる系じゃないですよね？」

そのとおりで、2つのテーブルにJOINをかけて一発で結果セットを引いてくるものですから、ぐるぐる系ではありません。

「サブクエリが問題なんでしょうか」

「まあ、これなら本来はサブクエリ使わんでもいいSQLだからそれで解決するやろな」

サブクエリでパフォーマンス劣化が起きやすいことはよく知られていますし、ここまで単純化していれば誰でも見つけられることでしょう。

しかし実際に使われていたSQL文はもっと複雑で、それがわかりにくくなっていました。だからこそ「複雑なSQLを嫌う」考え方が出てきやすいのでしょうが、それこそが「SQLを理解していないから逃げようとする」本末転倒な発想です。

「で、どうしてサブクエリ使うと遅くなるん？　理由は？」

「えっと……」

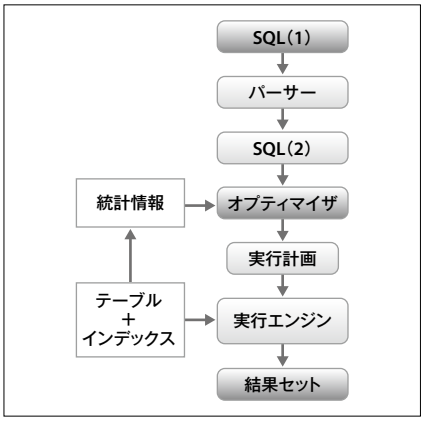
この理由がわからないと、本来サブクエリが役に立つときでもかたくなに使おうとしないケースもあります。やはりしくみをきちんと知ることが大事です。

「これは今までとは違う問題やけど、RDBとSQLを理解するには重要なポイントやから知っておくとええよ。ほな、実行計画見てみよか」

実行計画の確認はSQLチューニングの基本!

SQL文がDBサーバで処理される基本の流れは**図2**のようになります。発行されたSQL文(図中のSQL(1))はパーサーによる冗長部分のカットなどの加工を経て単純化され(SQL(2))、そ

▼図2　SQL文の処理の流れ



▼リスト1　月次請求処理のSQL(単純化したもの)

```
SELECT *
FROM 売上データ U
    LEFT OUTER JOIN
        (SELECT * FROM 顧客マスタ WHERE 削除FLG <> 1) C
    ON U.顧客ID = C.ID
WHERE
    -- 売上データに対する絞り込み条件;
```

れをもとにオプティマイザが実際のデータ処理アルゴリズムを組んだものが実行計画です。「実行計画」は手続き型言語で言うところのソースコードに該当するため、遅いSQLがあったら実行計画を確認するのは基本中の基本です。

「実行計画はあんまり見たことなくて……」

「これ読むとクエリの実行ロジックが推測できるようになるから、いろいろ見てみるとええよ。まずは**リスト2**と**リスト3**を比べてみると、何が違う？」

リスト2、3はそれぞれあるSQL文と実行計画（MySQL 5.6.1で生成）の組を掲載しました。SELECT文の下の表形式の部分が実行計画です。詳しい読み方はMySQLのリファレンスに任せて、本稿ではポイントのみ触れます。

「違いは顧客マスタテーブルの絞り込みでのサブクエリの有無ですよね。えーと……サブクエリを使うとインデックスが使われない？」

「そのとおり！　サブクエリの結果テーブルへのアクセスにはインデックスが使われない。だから注意が必要なんよ。まあ、今回の**リスト1**についてはそもそもサブクエリを使わないようにすれば、それで解決すると思うよ」

リスト3の実行計画最下行、DERIVED 顧客マスタ～の部分がSQLの削除FLG<>1で顧客マスタを抽出するサブクエリで、その上のPRIMARY <derived2>～の部分がその結果の読み込みです。注目すべきはどちらもtypeがALLになっていることで、これはインデックスを使わずにテーブルを全件読み込んでいることを意味します。

「サブクエリを使うと遅くなる、というのはそれが理由だったんですか」

「実行計画を読むと、実際どんなアルゴリズムでテーブルにアクセスするかがわかるから、性能トラブルシューティングするときは必ず確認するとええよ。複雑なSQLでも実行計画を見れ

▼リスト2　MySQL実行計画　サブクエリなし

```
SELECT *
FROM 売上データ U
    LEFT OUTER JOIN 顧客マスタ C
      ON U.顧客ID = C.ID
      AND 1 <> C.削除FLG
WHERE U.売上日 = '2016/04/01'
```

売上データが絞り込まれている

売上データテーブルの絞り込みにインデックスを使用

```
+----+-------------+-------+--------+---------------+-----------------+---------+-------------+------+-------------+
| id | select_type | table | type   | possible_keys | key             | key_len | ref         | rows | Extra       |
+----+-------------+-------+--------+---------------+-----------------+---------+-------------+------+-------------+
|  1 | SIMPLE      | U     | ref    | 売上データ_IDX1 | 売上データ_IDX1 | 3       | const       |  829 | NULL        |
|  1 | SIMPLE      | C     | eq_ref | PRIMARY       | PRIMARY         | 4       | demo.U.顧客ID |    1 | Using where |
+----+-------------+-------+--------+---------------+-----------------+---------+-------------+------+-------------+
```

顧客テーブルの検索にインデックスを使用

▼リスト3　MySQL実行計画　サブクエリあり

```
SELECT *
FROM 売上データ U
    LEFT OUTER JOIN
    (SELECT * FROM 顧客マスタ WHERE 削除FLG <> 1) C
      ON U.顧客ID = C.ID
WHERE  U.売上日 = '2016/04/01'
```

サブクエリで顧客マスタテーブルを全件読み込み、その結果のテーブルにインデックスを使わず全件アクセス

```
+----+-------------+-------------+--------+---------------+-----------------+---------+-------------+-------+-------------+
| id | select_type | table       | type   | possible_keys | key             | key_len | ref         | rows  | Extra       |
+----+-------------+-------------+--------+---------------+-----------------+---------+-------------+-------+-------------+
|  1 | PRIMARY     | U           | ref    | 売上データ_IDX1 | 売上データ_IDX1 | 3       | const       |   829 | NULL        |
|  1 | PRIMARY     | <derived2>  | ALL    | NULL          | NULL            | NULL    | demo.U.顧客ID |    10 | NULL        |
|  2 | DERIVED     | 顧客マスタ  | ALL    | NULL          | NULL            | NULL    | NULL        | 20083 | Using where |
+----+-------------+-------------+--------+---------------+-----------------+---------+-------------+-------+-------------+
```

ば遅くなるところは見つけやすいんよ。とくにtype=ALLには注意する」

「はい！」

「ただし、実行計画はDBの製品、バージョン、データの状態で変わるから、実環境と同じ環境でやらんと意味ないんでそこは注意してな」

実は近年MySQLも賢くなっていて、バージョン5.6.3からはサブクエリの結果に対して自動的にインデックスを生成して使用しています。もちろん、インデックスを作りなおすため、もともとあるインデックスを使うよりは遅いですが。

また、たとえばテーブルのデータ件数が少ない場合はインデックスがあっても使わずに全件読み込みをしたほうが速い場合もあるため、オプティマイザが実行計画を生成する際は、読み込むテーブルのデータ量やインデックスの有無などについての統計情報をヒントにしています(**図2**)。そのため、SQL文が同じでも、データが違えば異なる実行計画になることがあります。

「そうなんですか」

「DB製品によっての違いも大きくて、Oracleはこのへんの処理が賢いんよ」

Oracleでは**リスト3**のようなSQL文を、パース処理の段階でサブクエリを使わない形に書き換えてしまいます。また、MySQLにはないアルゴリズムを利用して、インデックスなしでも高速JOINが可能なのもOracleの特徴です。

「インデックスがなくても高速にJOINできるというのは、どんなしくみなんですか？」

「じゃあ、この機会に3種類のJOINアルゴリズムについてやっとこうか」

3種類のJOINアルゴリズム

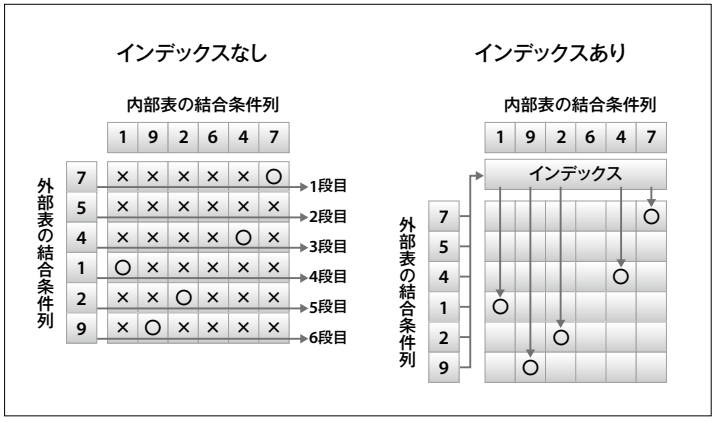
大きなテーブルを結合するとDBに負荷を与えがちなため、何かと敬遠されることも多いJOIN機能の実装アルゴリズムは大まかにネステッドループ、ソート／マージ、ハッシュの3種類があります。Oracleはこの3種類すべてを実装しているのに対して、MySQLではネステッドループ方式だけが実装されています。

本稿ではそれぞれの動作ロジックが性能面でどのような影響を与えるか、という点に絞ってチャート(**図3〜5**)を用意しました。

ネステッドループ結合

ネステッドループ結合の処理イメージは**図3**です。「外部表(売上データ)」の全件に対して、「内部表(顧客マスタ)」の中からマッチするものを探して結合する処理であると考えてください。インデックスがない場合は外部表と内部表の全件について結合条件値を2次元に展開した表を作り、そのすべてについて結合判定を行うと考えるとわかりやすいです。1段目、2段目……とループが重なるのでネステッドループ結合と言います。○と×の両方をチェックしていくため、件数が増えると爆発的に負荷が増えるのが直観的にわかることでしょう。基本的には外部表の一部をインデックスのある内部表に結合する場合に向いています。インデックスがあるとマッチするデータを直接探すことができるため、「×」のチェックが不要になり、高速に処理することができます。

▼図3　ネステッドループ結合



ソート／マージ結合

ソート／マージ結合の処理イメージは**図4**です。外部表と内部表の結合条件列の双方をソートしたうえで、双方の値を少しずつ増やしながらマッチングをかけます。**図4**のように対角線上の部分だけをチェックするイメージになるため、インデックスなしのネステッドループに比べてチェック件数が減り、表の大部分のデータ同士でも高速に結合することができます。非等価結合でも使用可能です。ただし、いったんソートする負荷がかかるため、基本的には結合するカラムの双方にインデックスがあって改めてのソートが不要なときに使われます。

ハッシュ結合

ハッシュ結合の処理イメージは**図5**です。内部表の結合条件列の値をハッシュ関数にかけてハッシュ表を作ったうえで、外部表の値を同じハッシュ関数にかけてマッチングの候補となる内部表の値を探します。ハッシュ表はすべての結合条件値をできるだけまんべんなく散らばるように分類したもの、と考えることができます。分類ですので1つの分類には複数の値が属しますが、「全件」に比べればはるかに少ないため、結合一件あたりに必要な等価条件チェック数が激減します。しかも重いソート処理が不要のため、インデックスのないテーブルを効率よく結合することができます。ただし、等価結合の場合のみ使用可能です。

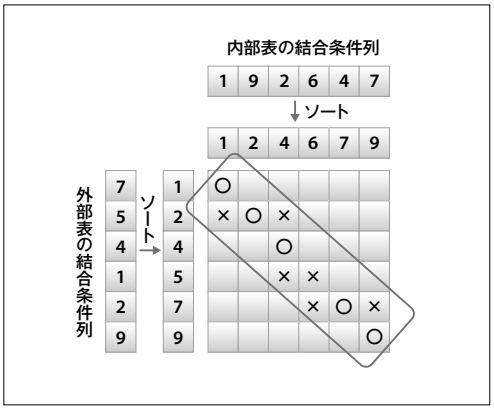
SQLはしくみを理解して使うことが重要

一発系SQLを嫌ってぐるぐる系SQLを使いたがる理由の1つとしてよくあがるのが、「複雑なSQLはDBに負荷がかかるから」というものです。しかし、実際に複雑な一発系SQLで高負荷なものを調べてみると、本稿で触れたようなインデックス、サブクエリ、結合操作といったしくみを理解せずに下手なSQLを書いていることが原因の場合がほとんどです。

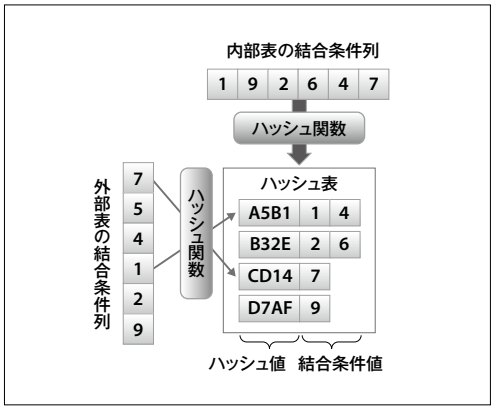
SQLはゴールを示す仕様書のようなもので、実際のデータ処理ロジックを作るオプティマイザがプログラマ(PG)にあたります。データの件数や分散具合をヒントにして、SQLが示すゴールを得るための実行計画を作るのがPG(オプティマイザ)の役割です。このため、同じSQLでもデータの状況が違えば違う実行計画を生成しますし、DBMSのバージョンが上がれば賢くなり、ベンダによっても性格が違います。ギリギリのチューニングをするなら、その性格を見越してSQLを書く必要があるわけです。

これらのしくみは一見取っつきにくそうに見えますが、落ち着いて考えればけっして難しいものではありません。しくみを理解して、「逃げずに」SQLを使ってみませんか？　実行計画を見ることはそのための第一歩なのです。SD

▼図4　ソート／マージ結合



▼図5　ハッシュ結合



コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

presented by
Japan Android Group
http://www.android-group.jp/

第6回 ルンバにAndroidスマホで命令だ!

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

金 祐煥(きん ゆうかん)
日本Androidの会
神戸支部・京都電創庵

takagig
日本Androidの会
神戸支部・GDG神戸所属

Androidでロボットを操作してみよう

最近の技術動向では、人工知能(以下、AI)やそれに関する技術が注目を集めています。ロボットと人工知能の組み合わせは、SF映画の世界が再現されるようでワクワクする気持ちになります。今回は、身近な掃除ロボットの「ルンバ」とAndroidをつないで操作してみたいと思います。

ルンバには、プログラミングをして自由に操作するための「Create」と呼ばれるシリーズが販売されています[注1]。ですが今回はCreateではなく、市販のルンバを使ったデモを動かしてみたいと思います(**図1**)。

ルンバとAndroidのつなぎ方

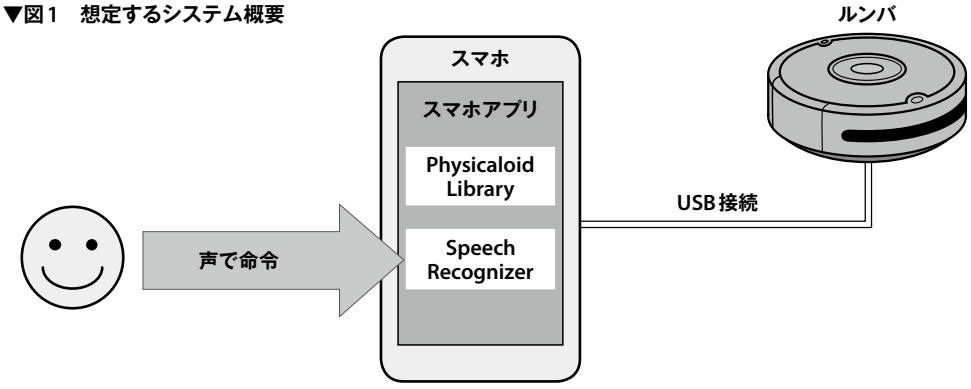
iRobot社のロボット掃除機ルンバにはミニDINコネクタの外部シリアルポートがついており、500以降のシリーズではRoomba Open Interfaceとして仕様が公開されています(**図2**)。このポートにシリアル通信機能のあるマイコンなどをつないで、外部から簡単にルンバを制御することが可能です[注2]。

・Roomba Open Interface仕様書
http://irobot.lv/uploaded_files/File/iRobot_Roomba_500_Open_Interface_Spec.pdf

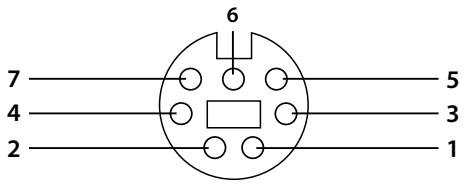
注1) iRobot Create
http://www.irobot.com/About-iRobot/STEM/Create-2.aspx

注2) ルンバ980からはRoomba Open Interfaceはなくなっており、アイロボットサービスセンターに問い合わせても状況を確認することはできませんでした(2016年3月末時点)。

▼図1 想定するシステム概要



▼図2 外部シリアルポートミニDINコネクタのピン配列（Roomba Open Interface仕様書より作図）



Pin	Name	Description
1	Vpwr	Roomba battery + (unregulated)
2	Vpwr	Roomba battery + (unregulated)
3	RXD	0 - 5V Serial input to Roomba
4	TXD	0 - 5V Serial output from Roomba
5	BRC	Baud Rate Change
6	GND	Roomba battery ground
7	GND	Roomba battery ground

ルンバにUSBシリアル変換アダプタを介してAndroid端末を接続し、外部からコントロールしてみます。今回は次の機材を利用します。

使用した機材
・ルンバ871
・Android端末（Nexus 6）
・USBシリアル変換アダプタ
・USBホストケーブル
・USBケーブル
・ジャンパワイヤ（オスーオス）

ルンバのシリアルポートは5V動作のため、USBシリアル変換アダプタは5V動作可能なものを使います。今回はスイッチサイエンスの「FTDI USBシリアル変換アダプタ（5V/3.3V切り替え機能付き）[注3]」を使用しました。

USBシリアル変換アダプタには、DTR、RX、TX、VCC、CTS、GNDの6つの端子がありますが、ルンバとシリアル通信するためには、RX、TX、GNDの3つを使用します。USBシリアル変換アダプタの電圧切り替えジャンパは**写真1**のように5V側にしておきます。

USBシリアル変換アダプタとルンバをオスーオスのジャンパワイヤで接続しますが、TXとRXD、RXとTXD、GNDとGNDを接続します（**表1**）。

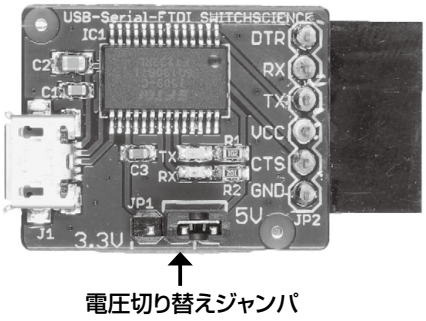
配線は絶対に間違えたりショートしないように気をつけてください、とくにルンバのPin1とPin2はバッテリーに直結しているため危険です。

TX、RXはそれぞれTXD、RXDと同じ意味ですので、**図3**のように互い違いにつなぐことになります。

配線はこれだけ、ハンダ付けも不要です。全体としては**図4**のように接続されているはずで

注3）https://www.switch-science.com/catalog/1032/

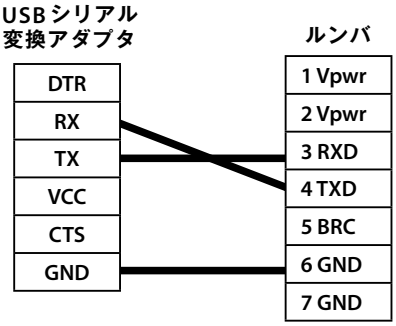
▼写真1 FTDI USBシリアル変換アダプタ（写真提供：スイッチサイエンス）



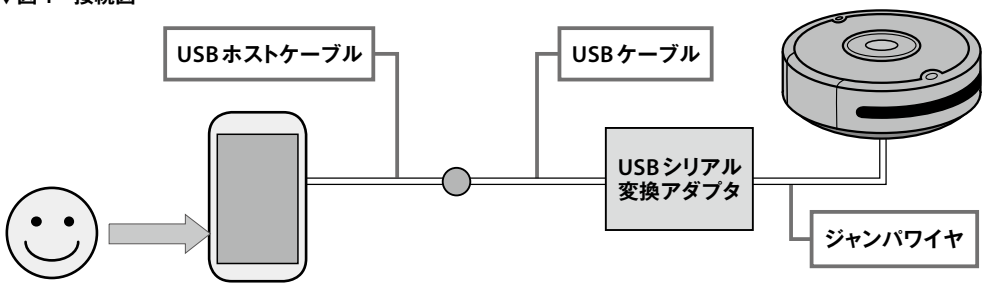
▼表1 USBシリアル変換アダプタとルンバのピン対応

USBシリアル変換アダプタのピン	ルンバのピン
TX	RXD（Pin3）
RX	TXD（Pin4）
GND	GND（Pin6かPin7）

▼図3 シリアル配線図



▼図4　接続図



▼表2　Physicaloidクラスのコンストラクタと主なメソッド

コンストラクタ	説明
public Physicaloid(Context context)	インスタンス生成
メソッド	**説明**
public boolean open()	シリアルポートをオープン
public boolean close()	シリアルポートをクローズ
public int read(byte[] buf, int size)	データ読み込む
public boolean addReadListener(ReadLisener listener)	読み込みリスナを登録
public void clearReadListener()	読み込みリスナをクリア
public int write(byte[] buf, int size)	データを書き込む

す。ルンバのシリアルポートへの接続はよく見て絶対に間違えないように、くれぐれもショートさせないように気をつけてください。ハンダ付けのできる方は7ピンもしくは8ピンのミニDINプラグを使用すると良いでしょう。

制御アプリ説明

Android端末とルンバを接続できたら、制御アプリを作ります。今回は音声でルンバに命令をして走らせてみます。

プロジェクトの準備

Android端末からUSBシリアル変換アダプタを利用できるようにするため、ksksue氏が作成したPhysicaloid Library[注4]を使用します。Physicaloid Libraryは次のようにしてAndroid Studioのプロジェクトへ組み込んでください。

注4）https://github.com/ksksue/PhysicaloidLibrary

①libsにライブラリファイルを追加

Android Studioでプロジェクトを作成したら、libsフォルダにPhysicaloid Libraryからphysicaloidlibrary.jarとd2xx.jarの2つのファイルをコピーします。

②build.gradleを修正

build.gradleのdependenciesに「compile files('libs/physicaloidlibrary.jar')」を追加します（**リスト1**）。

③AndroidManifest.xmlにパーミッションを追加

USBホスト機能を有効にするために、AndroidManifest.xmlにパーミッションを追加します（**リスト2**）。

Physicaloid Libraryの利用方法

次にPhysicaloid Libraryの使い方を見ていきましょう。**表2**に記載されているPhysicaloid

▼表3 Roomba Open Interfaceのモード

モード	説明
オフ	電源投入後はオフモードとなる
パッシブ	スタートコマンドやクリーニングコマンドを送るとパッシブモードとなる。センサーコマンドを使用して、センサーデータを受け取ることができる。ルンバを制御するには、フルかセーフモードに切り替える必要がある
セーフ	セーフコマンドを送るとセーフモードとなる。セーフモードでは、いくつかの安全条件の範囲内でルンバを制御できる
フル	フルコマンドを送るとフルモードとなる。ルンバを完全に制御できる

▼表4 Roomba Open Interfaceのコマンド

名称	コード(バイト)	説明
スタート	128	パッシブモードに変更する。ほかのコマンドを開始する前に、常にこのコマンドを送信する必要がある
フル	132	フルモードに変更する
直接ドライブ	145, 右速度上位バイト, 右速度下位バイト, 左速度上位バイト, 左速度下位バイト	左右の車輪に独立して速度を指定する。符号付き16bit数値をバイトごとに与える。数値範囲は-500～500mm/s

クラスのメソッドを利用して開発していきます。

Physicaloid Libraryでデータをシリアルポートから送信する例を**リスト3**に示します。この例では「hello」という文字列をバイトデータ列に変換し、シリアルポートに送っています。

このようにUSBシリアル変換アダプタとPhysicaloid Libraryを使うと簡単にAndroidでシリアル通信ができるようになります。

Roomba Open Interfaceの確認

次に、ルンバの仕様を見てみます。Roomba Open Interfaceには**表3**に示す4つのモードがあります。

Roomba Open Interfaceにはルンバを制御するコマンドが多数用意されていますが、今回作成するサンプルで使用するコマンドを**表4**に示します。このほかにも、掃除を開始するコマンドや音楽を鳴らすコマンドなどがあります。詳しくはRoomba Open Interface仕様書を参照してください。

Roomba Open Interfaceのコマンドは1バイトのオペコードで始まり、その後にパラメータとして必要な数のバイトデータが続きます。

たとえば直接ドライブコマンドは、先頭にオペコードとして145を送った後、2つの符号付き16bit数値を2バイトのデータに分解し、上位、下位の順番に送ります。少しわかりにくいかもしれませんので、**リスト4**にあるサンプルコードもあわせてご覧ください。

ルンバにコマンドを送るときは、まずスタートコマンドを送ります。スタートコマンドを受信するとルンバはパッシブモードとなりますが、ルンバを制御するためにはセーフモードかフルモードにする必要があるので、フルコマンドを送ってフルモードにしてから、直接ドライブコ

▼リスト1 build.gradleの修正個所

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.3.0'
    compile files('libs/physicaloidlibrary.jar')
}
```

▼リスト2 パーミッションの追加

```
<uses-feature android:name="android.hardware.usb.host" />
```

▼リスト3 Physicaloid Libraryでのデータ送信実装例

```
// Physicaloidのインスタンスを生成
Physicaloid mPhysicaloid = new Physicaloid(this);
// オープン
if(mPhysicaloid.open()) {
    // 通信速度設定
    mPhysicaloid.setBaudrate(115200);
    // データを書き込み
    byte[] buf = "hello".getBytes("UTF-8");
    mPhysicaloid.write(buf, buf.length);
    // クローズ
    mPhysicaloid.close()
}
```

▼リスト4　ルンバの車輪を制御するメソッドの例

```
/**
 * 車輪制御コマンドを送信する
 *
 * @param l 左車輪パラメータ、-500～500 (mm/s)
 * @param r 右車輪パラメータ、-500～500 (mm/s)
 */
private void sendDriveDirect(int l, int r) {
    byte[] commands = new byte[7];
    commands[0] = (byte) 128;        // Start

    commands[1] = (byte) 132;        // Full

    commands[2] = (byte) 145;        // Drive Direct
    commands[3] = (byte) (r >> 8);   // Right velocity high byte
    commands[4] = (byte) r;          // Right velocity low byte
    commands[5] = (byte) (l >> 8);   // Left velocity high byte
    commands[6] = (byte) l;          // Left velocity low byte

    // バイト列をシリアルに送信
    sendCommand(commands);
}
```

マンドを送ります。

①スタートコマンド(128)を送る
②フルコマンド(132)を送る
③直接ドライブコマンド(145)とパラメータを送る

リスト4にルンバの車輪を制御するメソッドの例を示します。

Speech Recognizerの利用方法

音声認識の処理についても簡単に解説します。onCreateで音声認識処理用のIntentを準備します。ボタンを押下されるとstartListen処理が呼び出されます。その結果、Speech Recognizerの機能が呼び出され、音声認識処理が実行されます。音声認識の結果は、文字列の配列として取得されます。今回のサンプルでは、ArrayList<String> candidates変数に処理結果を格納して、配列のいずれかにコマンドに相当する認識結果が含まれているかどうかを判定しています(**リスト5**)。

サンプルアプリの使い方

サンプルアプリのプロジェクトを用意しています。次のサイトからダウンロードしてください。

https://github.com/titoi2/RoombaVoiceControlSample

ダウンロードしたら、Android StudioでビルドしてAndroid端末にインストールしてください。音声入力を使っているため、Android 6では許可設定が必要です。Android 6端末の場合、[設定]→[アプリ]→本アプリを選択→[許可]→[マイク]をオンにしてください。

アプリを起動したら、**図4の接続図**のとおりにAndroid端末とルンバを接続しますが、USBホストケーブルとUSBケーブルを最後につなぎます。すると「USBデバイスへのアクセスを許可しますか？」というダイアログが出ますので、OKを押してください。これで準備が整いました。

入力開始ボタンを押すと、音声入力のダイアログが表示されるので話しかけてください。「前」「後」「右」「左」を含む音声を認識するとその方向にルンバが動きます。入力開始ボタンを押すとルンバが停止しますので、また話しかけてください。ルンバが止まらなくなった場合は、ルンバをリセットしてください。800/700シリーズの場合はCLEANボタンを10秒以上、600/500シリーズの場合はSPOTボタンを押したままDOCKボタンを10秒以上押します。

まとめ

今回は音声操作でルンバを動かしてみました。思ったよりも簡単にスマホからロボットが操作できたのではないでしょうか。音声認識の技術は成熟してきており、Google Cloud Speech

API[注5]のリリースにもあるように、精度が高い認識処理をアプリにどんどん組み込んでいける時代になっています。声を使ったロボットの操作は、仕事やゲームなどの幅広い領域で使える技術になるはずです。

また、Androidを使っているので、音声だけではなく、画像認識による操作やメール受信による操作などさまざまな機能と組み合わせることができます。今話題のVRメガネと組み合わせて、仮想現実の世界で動かせば、現実のロボットも動くようなMixed Reality(MR:複合現実)を実現できるかもしれません。ぜひ今回のサンプルを試してみて、未来の活用方法に思いを馳(は)せてはいかがでしょうか。SD

注5) https://cloud.google.com/speech/

▼リスト5 Speech Recognizerを利用するメソッドの例

```
(..略..)
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
(..略..)
        mButtonStart = (Button) findViewById(R.id.buttonStart);
        mButtonStart.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // 開始ボタン押下処理
                roombaStop();
                startListen();
            }
        });
(..略..)
        // 音声認識のIntentインスタンスを生成
        mSpeechIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        mSpeechIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.➥
LANGUAGE_MODEL_FREE_FORM);
        mSpeechIntent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 10);
        mSpeechIntent.putExtra(RecognizerIntent.EXTRA_PROMPT, "音声を入力");
    }
(..略..)
     private void startListen() {
        // インテント発行
        startActivityForResult(mSpeechIntent, REQUEST_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        Log.v(TAG, "onActivityResult");
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == REQUEST_CODE) {
            if (resultCode == RESULT_OK) {
                // 認識結果を取得
                ArrayList<String> candidates = data.getStringArrayListExtra(Recognizer➥
Intent.EXTRA_RESULTS);
                // コマンド解析
                String command = commandAnalyze(candidates);
                mTvLog.setText(command);
            } else {
                roombaStop();
            }
        }
    }
(..略..)
```

一歩進んだ使い方のためのイロハ

Vimの細道

第8回

ファイル操作を柔軟にするCtrlP

mattn
twitter:@mattn_jp

今回紹介するのは、VimでVisual StudioやEclipseといったIDEの持つファイル検索機能を実現するプラグイン「CtrlP」。そのCtrlPについて、基本的な使い方とより便利になるカスタマイズ方法、CtrlPをさらにプラグインで拡張する方法について解説します。

Vimでファイルを開くとき

みなさんはVimでファイルを開く際に、コマンドラインからファイル名を指定して起動するほうが多いでしょうか？　それとも、とりあえずVimを起動したあと、:eコマンドやほかの方法でファイルを開くほうが多いでしょうか。もちろん、編集したいファイルの数が多いかどうかで使い分けるという方もいるでしょう。人によっては毎回シェルに戻って、別のファイルを開き直すのを好む人もいます。

とくに昔のviに慣れていた人達には、viでの編集は1ファイルに限るといった習慣があり、筆者もあまりVimを常駐したままにはしないほうです。これは、プロジェクト内でファイルを探すための手段が、昔はfindやgrepしかなかったことが原因で、頭の中でファイルを探したいならシェルに戻るという意識が染みついてしまっているからです。しかし現代ではVimも、そしてそれを取り巻くツール群も発達してきており、Vimを起動したままIDEのように扱ったほうが、何かと便利な場面が増えてきました。

昔はソースコードが1枚岩で書かれていることも多かったのですが、近代では1クラス1ファイルに分割する管理方法も当たり前になってきています。そして1つのプロジェクトに10個や20個、多くなれば数百個のファイルがあるなんてことも珍しくはなくなってきました。

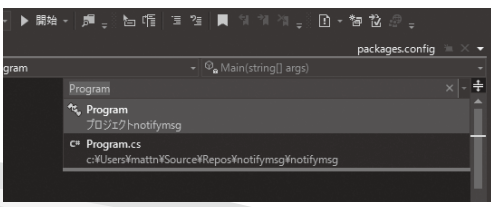
大規模な開発では当然、EclipseやVisual StudioといったIDEが使われます。もちろん、これらのIDEにはプロジェクト内のファイルを簡単に開く方法が提供されています。Visual StudioであればCTRL-,とタイプすると、[移動コマンド]のウィンドウが表示されます(**図1**)。

ここにファイル名の一部分だけ入力して Enter をタイプすると、マッチしたファイルが開かれます。EclipseであればCTRL-SHIFT-rで同様のウィンドウが開きます(**図2**)。

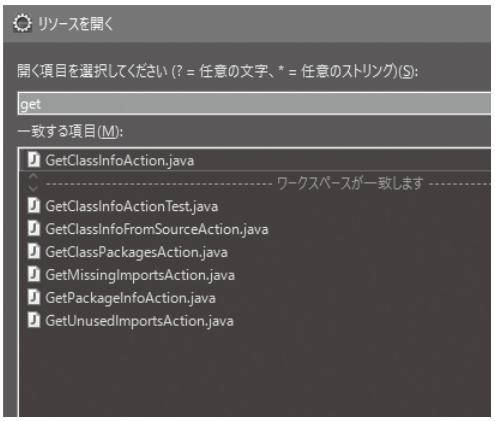
とても便利な機能なのですが、Vimには標準では備わっていません。しかし、**CtrlP**[注1]というプラグインを利用することで、同様の機能が実現できます。最近のVimユーザにはけっこう認知されてきたとは思いますが、あらためてCtrlPの使い方やカスタマイズ方法を紹介したいと思います。

注1）URL https://github.com/ctrlpvim/ctrlp.vim

▼図1　Visual Studioの[移動コマンド]



▼図2　Eclipseの［リソースを開く］



CtrlPとは

本連載でも幾度かその名前を紹介してきました。CtrlPはkien氏が開発を始め、現在はctrlpvimというグループで開発しており、筆者を含む数名がメンテナを務めています。CtrlPはその名のとおり、CTRL-pをタイプして起動します。CtrlPの特徴は次のとおりです。

・Vim scriptだけで書かれている
・Vimの正規表現を使った検索
・MRU(Most Recently Used)ファイルモニタリングおよび検索
・ルートディレクトリ検知
・複数ファイルの同時オープン
・ファイルおよびディレクトリの作成
・開いているファイル上でのExコマンド実行(行や文字列へジャンプ、もしくはほかの動作)
・キャッシュと履歴を使ったクロスセッションと高速な初期化
・マッピングとVimとの親和性

CtrlPの基本的な使い方

起動すると**図3**の画面が表示されます。CtrlPはデフォルトでファイル検索、バッファ検索、MRU(Most Recent Used)検索が有効となっています。基本的なキー操作は**表1**のとおり。

▼図3　CtrlP起動

画面下部のコマンドラインに文字を入力して一覧を絞り込みます。絞り込んだ内容の中から選択してファイルを開きます。コマンドラインのキー操作は**表2**のとおり。

ファイル操作はUNIXのシェルと同じ感覚で操作します(**表3**)。これに合わせてCTRL-jとCTRL-kで一覧のカーソルを上下できます。CTRL-zで一覧の中から現在の行を選択できます。複数選択もできます。

なお、端末版(CUI)のVimからCtrlPを使う場合はCTRL-sが効きません。これはCTRL-sが

▼表1　CtrlPの基本操作

キー	動作
CTRL-p	CtrlPを起動
CTRL-cまたはESC	CtrlPを終了
CTRL-d	パスモードを切り替え(絶対パス／ファイル名のみ)
CTRL-r	検索モード切り替え(正規表現／あいまい検索)
CTRL-fまたはCTRL-b	機能切り替え(ファイル／バッファ／MRU)

▼表2　CtrlPのコマンドラインのキー操作

キー	動作
CTRL-a	カーソルを行頭に移動
CTRL-e	カーソルを行末に移動
CTRL-u	コマンドラインをクリア
CTRL-nまたはCTRL-p	コマンド入力履歴をたどる
CTRL-\	コマンドラインに挿入※
Tab	ディレクトリ名を補完

※カーソル上の単語、カーソル上のファイル名、検索パターン、ビジュアル選択内容、クリップボード、レジスタから選択

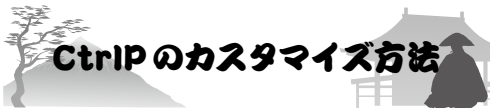
▼表3　CtrlPのファイル操作

キー	動作
Enter	カレントウィンドウで開く
CTRL-t	タブで開く
CTRL-v	垂直分割で開く
CTRL-sまたはCTRL-ENTER	水平分割で開く
CTRL-y	コマンドラインに入力中のファイル名を開く
CTRL-o	現在の行もしくはマークされている行のファイルを開く※

※タブ、垂直分割、水平分割、入れ替え、非表示、ユーザ指定(設定がある場合のみ)から選択

端末のサスペンド動作になっているからです。サスペンドを使わないのであれば.bashrcなどに次の行を追加しておくと、CtrlPでCTRL-sが使えるようになります。

```
stty stop undef
```



まずは起動方法

CtrlPは標準で、ファイル一覧機能にのみマッピングが行われます。ほかの機能にはマッピングが行われません。しかし、ファイル一覧と同じくらいよく使う機能があります。MRU(Most Recent Used)です。CtrlPは表示する情報をファイルにキャッシュすることで高速に一覧を表示し、ユーザが開きたいファイルへ簡単にアクセスできるようになっています。筆者もとくにこの機能はよく使っており、,,というマッピングを割り当てています。

```
nnoremap ,, :<c-u>CtrlPMRUFiles<cr>
```

CtrlPでは、このようにコマンドに対してマッピングを行います。

除外ファイル

CtrlPでファイル一覧から特定のファイルを除外したい場合は、g:ctrlp_custom_ignoreを使用します。**リスト1**のようにファイル、ディレクトリを個別に設定できます。linkには実際に除外したいシンボリックリンクのパスを記述します。

ファイル検索ユーザコマンド

ファイルの一覧を表示する際、CtrlPはデフォルトでVim scriptのglob関数を使っています。しかし、巨大なディレクトリツリーを扱うにはVim scriptの機能では若干不利になります。これをVimの機能で実現するためには、あるディレクトリ配下のファイルをすべて検索し、ドットファイルや拡張子が.exeのものを除外しなければいけません。しかしながらこれらは本来、検索する過程で対象から除外できるはずです。そこでCtrlPではこのファイル検索機能を、外部コマンドに任せられるようになっています。

筆者の場合はファイルを一覧するためだけに作った自作コマンドfiles[注2]を使っています。次のように設定します。

```
let g:ctrlp_user_command = 'files %s'
```

ただし、このg:ctrlp_user_commandを使ってファイル一覧を得る場合には、前述のg:ctrlp_custom_ignoreは適用されません。外部コマンド側で除外ファイルを指定しなければ

注2) URL https://github.com/mattn/files

▼リスト1　CtrlPで、ファイル一覧から特定のファイルを除外

```
let g:ctrlp_custom_ignore = {
  \ 'dir':  '\v(^|[\/])(\.git|\.hg|\.svn|\.settings|target|bin|node_modules)$',
  \ 'file': '\v\.(exe|so|dll|png)$',
  \ 'link': '/path/to/some_bad_symbolic_links',
  \ }
```

▼リスト2　fileでの除外ファイル指定

```
let g:ctrlp_user_command = 'files -i "(^|[\/])(\.git|\.hg|\.svn)$|\.(exe|so|dll)$" %s'
```

▼リスト3　fileでの除外ファイル指定（ctrlp_custom_ignoreの設定がすでにされている場合）

```
let g:ctrlp_custom_ignore = {
  \ 'dir':  '\v(^|[\/])(\.git|\.hg|\.svn)$',
  \ 'file': '\v\.(exe|so|dll)$',
  \ 'link': '/path/to/some_bad_symbolic_links',
  \ }

let g:ctrlp_user_command = printf('files -i %s %%s',
  \ shellescape(join(map(['dir', 'file'],
  \ 'substitute(g:ctrlp_custom_ignore[v:val],''\\v'',"","")'), '|')))
```

▼リスト4　agでの除外ファイル指定

```
let g:ctrlp_use_caching = 0
let g:ctrlp_user_command = 'ag %s -i --nocolor --nogroup -g ""'
```

ならず、その指定方法はコマンドにより異なります。たとえばfilesコマンドであれば**リスト2**のように設定します。また、すでに**ctrlp_custom_ignore**の設定をたくさんしているのであれば、**リスト3**のように設定することもできます。

またag（the silver searcher）を使う場合は、**リスト4**のように設定します。ユーザの中には、agを使うときは**リスト4**のようにキャッシュを無効にして使う人もいるようです。

スクロール

通常、CtrlPで表示されるファイル一覧では、CTRL-jとCTRL-kで移動する際に表示範囲外へ移動できません。次の設定を追加することで、「パス」を提供するモードのみでスクロールが有効になります。

```
let g:ctrlp_path_nolim = 1
```

日本語入力

CtrlPのコマンドラインは、実はキーマッピングにより実装されており、可視文字に対しその文字を挿入する関数呼び出しがマッピングされます。つまりこのマッピングがされていない、たとえばマルチバイト文字などを入力しても何も挿入されません。英語圏の人々にはそれでも良いのですが、ファイル一覧にマルチバイト文字が含まれている場合には絞り込みができなくなります。次の設定を行うことで、キー入力待ちによる実装に切り替わり、きちんとマルチバイト文字が扱えるようになります。

```
let g:ctrlp_key_loop = 1
```

マッチャー

ファイル一覧で何か文字を入力した場合、通常はあいまい検索で一覧が絞り込まれます。しかしこの機能はVim scriptの正規表現で実装されているため、大量のファイル名を扱う場合にはパフォーマンスが悪くなり、また速度を優先しているためマッチング精度も褒められるほどではありません。CtrlPではこの絞り込み機能（マッチャー）を拡張できるようになっています。サードパーティからいろいろ提供されています（**表4**）。筆者はこの中でもCPSMを愛用しています。ビルドにboostやcmakeが必要で導入の敷居が若干高いですが、入れるだけの価値はあると思っています。インストール後、次の設定で有効となります。

▼表4　マッチャーの拡張プラグイン

名前	特徴	URL
ctrlp-py-matcher	Pythonのみで実装。Pythonの正規表現を使ってマッチ	https://github.com/FelikZ/ctrlp-py-matcher
ctrlp-cmatcher	C言語で書かれたPython拡張。いくぶん高速	https://github.com/JazzCore/ctrlp-cmatcher
fzf	Go言語で書かれた実行モジュール。いくぶん高速	https://github.com/junegunn/fzf
CPSM	C言語で書かれたPython拡張。スレッドを使って多重検索。高速	https://github.com/nixprime/cpsm

```
let g:ctrlp_match_func = ↵
{'match': 'cpsm#CtrlPMatch'}
```

省略入力

CtrlPのコマンドラインはVimのコマンドライン(:)と同様に、Vimのコマンドが入力できます。このコマンドラインではVimのabbr(省略入力)と同様の機能が用意されています。誌面の都合で内容の説明は省略しますが、**リスト5**を行ったあと、コマンドラインにcd pを入力するとcd ~/.vim/pluggedに展開されます。そのままEnterをタイプすると、実際にカレントディレクトリが~/.vim/pluggedへ移り、ファイル一覧が更新されます。

▼リスト5　CtrlPでの省略入力

```
let g:ctrlp_abbrev = {
  \ 'gmode': 't',
  \ 'abbrevs': [
    \ {
      \ 'pattern': '^cd p',
      \ 'expanded': '@cd ~/.vim/plugged',
      \ 'mode': 'pfrz',
    \ },
    \ {
      \ 'pattern': '\(^@.\+\|\\\@<!:.\+\)\@<! ',
      \ 'expanded': '.\{-}',
      \ 'mode': 'pfr',
    \ },
    \ {
      \ 'pattern': '\\\@<!:.\+\zs\\\@<! ',
      \ 'expanded': '\ ',
      \ 'mode': 'pfz',
    \ },
    \ ]
  \ }
```

▼リスト6　\mfですぐにメモを取れる(mapleaderがデフォルトの場合)

```
nnoremap <leader>mf :<c-u>CtrlP ~/memo<cr>
nnoremap <leader>mc :<c-u>MemoNew<cr>
```

MemoListとの連携

筆者はメモ取りにmarkdown記法を使うようにしています。何か記録する必要があると思ったときには、VimからMemoList[注3]というプラグインを使ってメモを記述しています。どんな状態でも、どんなディレクトリにいてもメモが書き始められるように、**リスト6**の設定をしています。メモを取りたくなったら、\mfをタイプします。

CtrlPはファイルの一覧を選択するためだけのプラグインではありません。拡張を作るためのAPIを公開しており、そのルールに従った処理を実装すると、CtrlPに表示する一覧をプラグイン側から提供でき、かつ選択した際のアクションにも独自の処理を実装できます。

ctrlp-launcher

ctrlp-launcher[注4]はVimからいろいろなものを起動するためのランチャーです。インストール後、

注3) URL https://github.com/glidenote/memolist.vim
注4) URL https://github.com/mattn/ctrlp-launcher

▼リスト7　ctrlp-launcherの設定例

```
cmd     :!start cmd
gimp    :!start C:\Program Files\GIMP 2\bin\gimp-2.8.exe
synid   :echo synIDattr(synID(line("."), col("."), 1), "name")
```

```
nmap <c-e> <plug>(ctrlp-launcher)
```

のように設定してからCTRL-eをタイプすると「--edit-menu--」という行が表示されるので、Enterで開いたあと、次の書式で設定します。

```
項目名［タブ文字］コマンド
```

筆者はリスト7のように設定しています。GIMPやコマンドプロンプト、そのほかVimで使うユーティリティなどを登録しています。

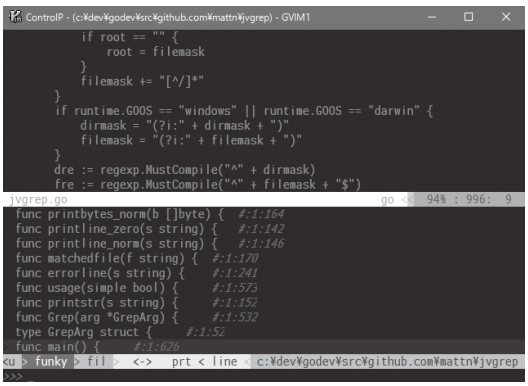
ctrlp-funky

ctrlp-funky[注5]はあらゆるプログラミング言語の関数をナビゲートするためのCtrlP拡張です。編集中ソースファイルの関数を一覧表示し、簡単にジャンプできるようになります。ctagsなどを使わず独自の関数名抽出処理を行っており、38種類ものファイル種別をサポートしています。

可能性はたくさんある

そのほかおもしろいところでは、Music Player Daemonで管理されている音楽ファイルを簡単に再生できるctrlp-mpc[注6]や、GitHubリポジトリのローカルクローンへ簡単にアクセスできるctrlp-ghq[注7]などがあります。前述のMemoListの例のように、ファイルブラウザとして連携するだけでも普段使いのVimが便利になります。アイデアが浮かんだらぜひGitHubで公開してみてください。SD

▼図4　ctrlp-funky



注5）URL https://github.com/tacahiroy/ctrlp-funky

注6）URL https://github.com/lucidstack/ctrlp-mpc.vim
注7）URL https://github.com/mattn/ctrlp-ghq

Vim自習

有名Vimmerに学ぶ

今年の頭、「How I Vim」というサイトに注目が集まりました。

・How I Vim
http://howivim.com

インターネット上で有名なVimmerへのインタビュー記事が集まっており、各VimmerのVimの使い方やお気に入りの設定、プラグインが紹介されています。筆者もインタビューの依頼を受け、つたない英語ですが記事が掲載されています。日本人Vimmerだとほかにも Shougo さん、tyru さんへのインタビュー記事が掲載されています。すべて英語ですが、各VimmerのVimに対する考え方の違いがわかって、とても参考になります。

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26回 シェルコマンドを活用しよう（中編）

前回に続き、Emacs上で使える基本的なシェルコマンドについてみていきます。ネットワーク系コマンドからPythonやRubyの対話環境を起動するコマンド、さらにはコンパイルのためのコマンドからgrepまで、外部プログラムと連携する幅広いコマンドがEmacsには取り揃えられているのです。

Emacsにおけるシェルコマンド活用

ども、るびきちです。前回はEmacsでシェルコマンドを呼び出す基本的なコマンドを紹介しました。

M-x shellやM-x eshellはEmacs内でシェルを動かします。画面指向のプログラムを直接実行できなかったり、補完や履歴が弱いといった欠点はあるものの、バッファ内でシェルが動くことにより次の恩恵を受けられます。

①出力を遡（さかのぼ）れる
②消さない限り実行結果が残る
③同時に複数のシェルを実行できる
④実行結果をバッファに貼り付けられる
⑤端末に切り替えなくても済む

シェルコマンドの実行結果を貼り付けたい場合は、手始めにバッファ内シェルを使えば良いでしょう（**図1**）。M-!（shell-command）はその場でシェルコマンドを実行し、実行結果を表示します。シェルバッファとは違い、バッファ切り替えを伴いません（**図2**）。C-u M-!とすると、実行結果をバッファに貼り付けます。M-&（async-shell-command）は、シェルコマンドを実行してもEmacsの動作が停止しません。実行に時間がかかるプログラムやユーザ入力を伴うプログラ

▼図1 M-x shell

```
rubikitch@/tmp$ pwd
/tmp
rubikitch@/tmp$ date
2016年  3月 14日 月曜日 05:05:02 JST
rubikitch@/tmp$
U:**-  *shell*        All L5     (Shell:run)
```

▼図2 M-! ifconfig lo

```
emacs-test
;; This buffer is for notes you don't want to save, and for Lisp evaluation.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.

U:---  *scratch*      All L5     (Lisp Interaction)
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (ローカルループバック)
        RX packets 796488  bytes 521057843 (496.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 796488  bytes 521057843 (496.9 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

U:**-  *Shell Command Output*   All L1     (Fundamental)
```

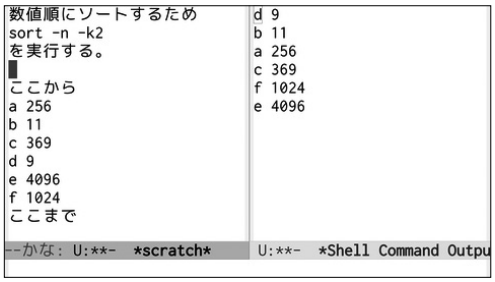
▼図3 M-& vmstat 1で1秒ごとにリソースの状況を表示

```
emacs-test
;; This buffer is for notes you don't want to save, and for Lisp evaluation.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.

U:---  *scratch*      All L4     (Lisp Interaction)
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 441248 4653592 26228140    0    0   145    17    1    1  2  0 97  0  0
 0  0      0 441232 4653592 26228140    0    0     0     0 1821 1183  2  0 98  0  0
 1  0      0 441108 4653592 26228140    0    0     0     0 1826 1172  2  0 98  0  0
 1  0      0 440984 4653592 26228140    0    0     0     0 1834 1176  3  0 97  0  0
 0  0      0 440984 4653592 26228140    0    0     0     0 1827 1237  2  0 98  0  0
 0  0      0 440984 4653592 26228140    0    0     0     0 1808 1247  4  0 96  0  0

U:**-  *Async Shell Command*   All L9     (Shell:run)
```

▼図4　regionを指定し、M-| sort -n -k2



ムを実行するときなどに使います（**図3**）。

M-|（shell-command-on-region）はregionを標準入力としてシェルコマンドを実行します（**図4**）。C-u M-|とすると、regionを実行結果に置き換えます。このC-u M-|は、Emacsのコマンドの能力を超えた処理をシェルコマンドに任せるときに便利です。

前回紹介したこれらのコマンドは、どれも「シェルコマンドを実行して終わり」です。今回は、用途に特化した標準コマンドを紹介していきましょう。

専用Emacsコマンド

ネットワークから対話環境まで！

シェルコマンドを扱う基本的なEmacsコマンドは以上ですが、特定のプログラムに特化したEmacsコマンドは本当にたくさん存在します。

たとえば、ネットワーク関係のプログラム（ifconfig、iwconfig、netstat、arp、route、ping、nslookupなど）は同名のEmacsコマンドがあります。これらはM-&同様に非同期に実行されます。

M-x nslookupはシェルのように対話モードで実行されますが、M-x shell同様すぐに入力できるようにカレントバッファが選択されます。

スクリプト言語の対話モードを実行するEmacsコマンドがあります。M-x run-pythonは「python -i」を実行し、M-x run-rubyやM-x inf-ruby（要inf-rubyパッケージ）はirbやpryを実行します。これらもM-x shell同様の操作感覚です。

M-x manは内部でmanプログラムを実行し、manpageを開きます。強調文字などがハイライトされ、ファイルやほかのmanエントリへのハイパーリンクも作成されます（**図5**）。

これらは基本的にはプログラムを実行し、場合によっては後処理をするものの、実行して終わりというものです。

vc

vcの各コマンドは、カレントディレクトリで使われているバージョン管理システムを自動判別して、適切なプログラムを実行します。

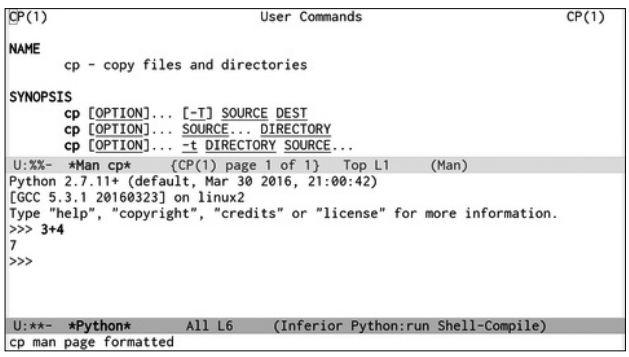
リポジトリへのコミット、前バージョンとの差分の閲覧、履歴の閲覧、古いバージョンのファイルを開くなどのEmacsコマンドが用意され、リポジトリに応じたプログラムを内部で実行します。

vcは、実行するプログラムを適切なインターフェースによって隠蔽するタイプです。vcの詳細については、本連載2016年3月号をご覧ください。

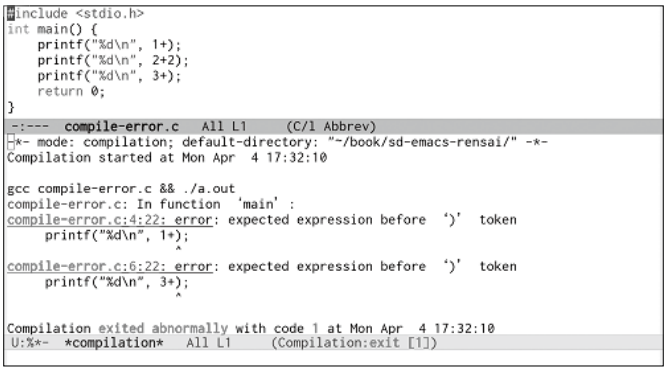
dired

実はdiredでは、内部でls -lを実行しています（Windowsなどでlsプログラムがインストールされていない場合は、elispによるlsエミュ

▼図5　上：M-x man cp／**下：**M-x run-python



▼図6　M-x compileでコンパイルエラー



レーションが使われます)。これは、diredの画面がls -lそっくりなことからも予想がつくことでしょう。

カレントディレクトリのファイルに何があるのかを知るために、シェルで頻繁にlsとタイプする癖がある人もいるとは思いますが、diredを使えばそれ以上のことができます。シェルバッファにおいては、lsする代わりにC-x d RETあるいはC-x C-f RETでdiredを開けばディレクトリの内容がわかります。あとはご存じのとおり、現在行が指し示すファイルを開いたり、コピー・移動・削除などの各種ファイル操作やシェルコマンドの実行ができます。また、マークを使えば複数のファイルに対してコマンドが作用します。

diredはシェルコマンド実行結果を基に、次に続く処理を実現するタイプです。

コンパイルコマンドとその亜種

コンパイル

開発中のプログラムをコンパイルするときはEmacsを使うと便利です。シェルでコンパイルしてコンパイルエラーが起きた場合は、わざわざ手動でそのファイルを開き、該当行に移動する必要があります。

M-x compileを使うと、Emacsの中でコンパイル処理が走り、コンパイルエラー行に簡単にジャンプできます(図6)。エラーメッセージの出力形式はソフトウェアによってまちまちですが、多くの形式を自動判別してくれます。

M-x compileを実行すると、コンパイルのためのシェルコマンドの入力が求められます。デフォルトでは「make -k 」となっていますが、変数compile-commandを設定すれば変更できます。

コンパイルエラーの行にジャンプするには、*compilation*バッファのエラー行でEnterを押すか、エラー行にジャンプするコマンドを使います。C-x `あるいはM-g M-n(next-error)で次のエラーにジャンプします。M-g M-p(previous-error)は、前のエラーにジャンプします。

これらのコマンドを使えば、わざわざ*compilation*バッファに移動することなくエラー行に移動できて便利です。ちなみに、next-error/previous-errorコマンドは、後述するM-x grep、M-x executable-interpretでも使えます。

grep

多数のファイルの中から特定の文字列や正規表現を検索するには、昔からgrepが使われています。

grepを実行したあと、マッチしたファイルを開き、その行に移動したいこともしばしばあります。そんなときシェルでgrepを実行しても、わざわざ手動でファイル名と行番号を入力する必要があります。ましてや、マッチした行を順次たどるのはかなり骨が折れます。

M-x grepはM-x compileと同系列のコマンドですので、コンパイルエラーにジャンプするコマンド(next-error/previous-error)を使ってマッチした行にジャンプできます(図7)。

grepプログラムにはさまざまな実装が存在するので細かいオプションは異なりますが、それ

はEmacs側がうまく吸収してくれます。実際M-x grepを実行したときに出てくるデフォルトのコマンドラインは、どのバージョンのgrepが使われているのかを自動判別し、適切なオプションを用意してくれます。筆者の環境（Debian GNU/Linuxでの GNU grep）では「grep -nH -e」と出ます。ユーザ側はとくに設定をしないでも、すぐにM-x grepを使えます。

▼図7 M-x grep（GREP_OPTIONSの警告はEmacs 25.1で修正されます）

```
;; Rscript and littler interpreters recognized.  XEmacs entries can
;; be regexps, which complicates matters as "r" on its own matches
;; other interpeters like "perl".
(add-to-list 'interpreter-mode-alist '("Rscript" . r-mode))
(add-to-list 'interpreter-mode-alist
             (cons (if (featurep 'xemacs) "r$" "r")    'r-mode))

-:---  ess-site.el    36% L294   (Emacs-Lisp Outl)
-*- mode: grep; default-directory: "~/.emacs.d/elpa/" -*-
Grep started at Wed Apr  6 09:34:12

grep -nH -r -e interpreter-mode-alist
grep: 警告: GREP_OPTIONS は廃止されました。alias またはスクリプトを使用してくださ
い
ess-20141019.806/lisp/ess-site.el:294:(add-to-list 'interpreter-mode-alist '("
Rscript" . r-mode))
ess-20141019.806/lisp/ess-site.el:295:(add-to-list 'interpreter-mode-alist
U:%*-  *grep*         Top L6     (Grep:exit [matched])
```

「-n」オプションはM-x grepを実行するうえで必須条件となるもので、出力行に行番号を含めます。このおかげで、正確に該当行にジャンプできます。

「-H」オプションは出力行に必ずファイル名を含めます。grepは指定されたファイルが1つの場合はファイル名を出力しないようになっているので、このオプションも必要となります。このオプションが存在しないgrep実装では、Emacs側が/dev/null（ヌルデバイス）を付加して強制的にファイル名も出力させます。

「-e」オプションのあとでパターンを指定するのですが、多くの場合このオプションはなくてもかまいません。ハイフンから始まるパターンを指定する場合などには必要となります。

M-x grepで実行できるプログラムはgrepに限らず、出力形式が「ファイル名:行番号:〜」であれば何でも良いです。たとえば、ソースコードに特化した超高速grepであるag（the silver searcher）を「ag --nogroup PATTERN」で実行すれば、実行結果にジャンプできます。grepやagをパイプで数珠つなぎした絞り込み検索もできます。

スクリプト実行

M-x executable-interpretはPerlやRubyなどのスクリプトの実行に特化したコンパイル系列コマンドです。M-x compileでもスクリプトを実行できますが、M-x executable-interpretはユーザ入力を伴うスクリプトも実行できます。

デフォルトのコマンドはカレントバッファのファイル名になっていますので、実行属性が付いていればコマンドライン変更なしに実行できます。M-x executable-set-magicで「#!行」と実行属性を付けられます。

M-x executable-interpretでは、コンパイルエラーにジャンプするコマンドも使えます。

◆ ◆ ◆

前回と2回に渡ってシェルコマンド実行関係の標準コマンドを紹介しました。次回は、さらに便利にする外部パッケージを紹介します。

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

書いて覚える Swift 入門

第15回　文字列の扱い

Writer 小飼 弾（こがい だん）　twitter @dankogai

文字列

今回はいよいよ文字列を扱います。文字列。最も多用されるデータ型でもあり、Swiftを含め、およそありとあらゆるプログラムも文字列で表記されています[注1]。にもかかわらず、筆者は今までSwiftにおける文字列を扱うのにモジモジしてきました。それには深いわけがあります。

> *> Swift's String and Character types provide a fast, Unicode-compliant way to work with text in your code. -- The Swift Programming Language*

「SwiftのStringおよびCharacter型は、高速でUnicode準拠したテキスト処理を提供します」。「高速」はとにかく、「Unicode準拠」というのがなかなかの難物なのです。なぜ難物なのか。それを知るためには、コンピュータにおける文字列処理の歴史を紐とかねばなりません。

A Brief History of Characters

文字列==文字の列。前世紀までは、文字列というものの理解はその程度で間に合っていました（**表1**）。

0と1という2種類の数値しか根源的に扱えない電子計算機（あえてこう書く）において文字を扱うにあたり、先人たちがやったのは基本的に次のものでした。

・各文字に番号を振り
・その番号を並べる

往時のプログラマたちにとって幸いなことに、当初扱わなければならない文字列は英語のみ。そして英語というのは必要な文字種がとても少ない言語でした。アルファベット26文字、大文字と小文字を区別しても52文字。アラビア

▼表1　文字列の扱いの歴史

Year	Event	Comment
1963	EBCDIC	初の文字コード規格
1963	ASCII	最も普及した文字コード規格
1969	JIS X 0201	初の日本語文字コード（カタカナのみ）
1978	JIS C 6226 (JIS X 0208)	かな漢字を含む文字コード
1982	Shift_JIS	
1985	EUC-JP	
1991	Unicode 1.0	現在のデファクトスタンダード
1992	UTF-8	拡張ASCIIとしてのUnicode
1993	ISO-2022-JP	電子メールにおける標準日本語文字コード
1995	Java 1.0	16bit Character
1995	JavaScript 1.0	
1996	Unicode 2.0	サロゲートペア標準化
2000	Python 2.0	バイト列 != 文字列
2002	Perl 5.8	Unicodeを言語としてフルサポート
2007	Ruby 1.9	
2008	Python 3.0	UCS2事実上の廃止
2010	Unicode 6.0	絵文字追加
2014	Swift 1.4	Version 1.0からUnicodeをフルサポート

注1） ScratchやPietのように画像として表記するプログラミング言語もないわけではありませんが。

数字を加えても62文字。これにスペースやタブや改行などを加えても、7bits = 128種類にらくらく収まったのです。これが、現在でも使われているASCII。量産されているCPUで扱える最も小さなデータ型である1byte = 8bits[注2]にそのまま入れても1bitあまります。

2byte文字の誕生と混乱

文字とは1byteに収まるものであり、それを並べたものが文字列である。

そのような時代が長いこと続きました。

その原則を破ったのが、日本です。カナだけでも48文字、しかもカタカナとひらがなと2種類。これだけでも8bitに収まらないのに、さらに数千種類の漢字も扱いたいとなると1つの文字を表現するのに12〜13bitsは必要。しかも漠然と並べるのではなく、ASCIIとの共存も考えると1つの文字を表記するのに2bytesは必要……こうして登場したのがShift-JISでありEUC-JPでありISO-2022-JPです。ここで注目すべきは、各文字に振られた番号はJIS X 0208という単一の規格なのに、それをどう並べるかで複数の規格が乱立したことです。それぞれ一長一短あるのですが、そうなってしまった理由は、「最後に正しいものではなく、今すぐ使えるものを」という「電子立国日本の現場圧力」ではなかったかと思われます。インターネットどころかパソコン通信すらまだ一般的ではなく、データ互換性はせいぜいメーカーがそれぞれ自社製品のみ担保されていた時代、まず大事だったのはパソコン、いやワープロ(もはや死語？)で入力できて出力できることだったのです。

Unicodeの誕生

その状況は、ネットの登場で一変します。コンピュータは単独で使用するものから、他のコンピュータ、強いてはそのコンピュータのユーザ同士をつなげるものとなったのです。そんな時代、各国ごとにバラバラの規格を使っていたのではメーカーはたまったものではありません。各国ごとに乱立していた「ASCII + 自国語文字コード」から、世界共通の文字コードへの移行の機運は高まっていたのです。「世界共通の文字コード」、それがUnicodeです。

最初に登場した段階におけるUnicodeは、過去との互換性はほとんど気にかけていませんでした。Shift-JISやEUC-JPといったASCII互換の日本語文字コードが可変長だったのに対し、1.0段階のUnicodeは16-bit固定長。ここにカナもハングルも日中韓の漢字も全部収める予定だったのです。そのためには、各国で用いられている文字コードをまとめて割り振るのではなく、並べ替えが必要となりました。それが(悪名高き)Han Unificationです。それの何が問題だったかといえば、文字コード変換。Unicode以前の日本語文字コードの相互変換は、元になる「背番号」が共通だったこともあり単純計算でOKだったのが、変換表が必要になったのです。

Unicodeの不幸

その一方で、Unicode陣営も1.0の「ゼロベースで文字列を再定義する」やり方がそのままではうまく行かないことに気づいてきました。まず、「同じ文字に同じ番号」という原則が破られます。Unicodeコンソーシアムのベンダ自身が、それまでの文字コードからUnicodeに変換したあとの逆変換がきちんと成立することを求めたからです。かくして半角カナも生き残りました。次に、16bitではとても足りないことに気がつきました。それを解決すべく、Surrogate Pairというものが登場しました。16bitで1文字から、「文字によっては16bit、2つで1文字」というわ

注2) 厳密にはbyteというのは「あるCPUで扱える最小のデータ型」のことでCPUごとに異なるものですが、現在量産されているCPUはほぼすべて1byte = 8bitsになっています。ちなみに峻別したい場合、8bitsは1 octetと呼びます。

けで、固定長という原則がここに崩れたのです。その代償として、Unicodeでは最大(16 + 1) * 2**16 == 1,114,112 文字まで扱えるようになりました。そしてASCII互換性も、UTF-8で解決されました。1文字の長さは1〜4bytesの可変長になる代わりに、ASCIIはこれまでどおり1byte。ASCIIを前提としていた数多のソフトウェア、とくにCコンパイラでもそのまま扱えます。

我々にとって不幸だったのは、「使えるUnicode」である2.0が登場する直前に、「インターネット爆発」が起こってしまったこと。とくにJavaとJavaScriptが1文字16bitとしてしまったのは今もなお尾を引いているのは本誌の読者であればご存じでしょう。おかげで絵文字の長さはそのままでは2文字になっちゃうとか...

文字コード統一の奇跡

Han Unificationのような「理念の押し付け」のあとに、Surrogate PairsやUTF-8のような「日和見(ひよりみ)」。それだけの犠牲を払って、世界をUnicode化するだけの価値はあったのでしょうか?

私自身、Perl 5.8のUnicode化という形でその片棒を担いだ以上、中立の立場とはとても言えないということをあらかじめお断りしたうえで言えば、その価値は確かにあったと断言します。Unicode以前の世界、小はワンライナーから大はOSに至るまで、ソフトウェアには各国語版がつきものでした。EmacsにはNemacs、PerlにはJPerl、Mac OSには漢字Talk……。しかし今や、世界中で使用されるソフトウェアは真の意味で世界的です。EmacsもPerlもOS Xも日本語版はもはや不要。iOSやAndroidにいたっては、その誕生時点からUnicodeを前提にできています。インターネットの普及によりTCP/IP以外の通信プロトコルは事実上滅亡しましたが、それによって世界が画一化したようにはとても思えません。我々の遺伝子だって、RNA-DNA-タンパク質というシングルアーキテクチャであることを考えれば、文字コードのように基礎的なデータ構造が(およそHan Unificationの混乱を除けば)統一できたのは奇跡なのではないでしょうか。

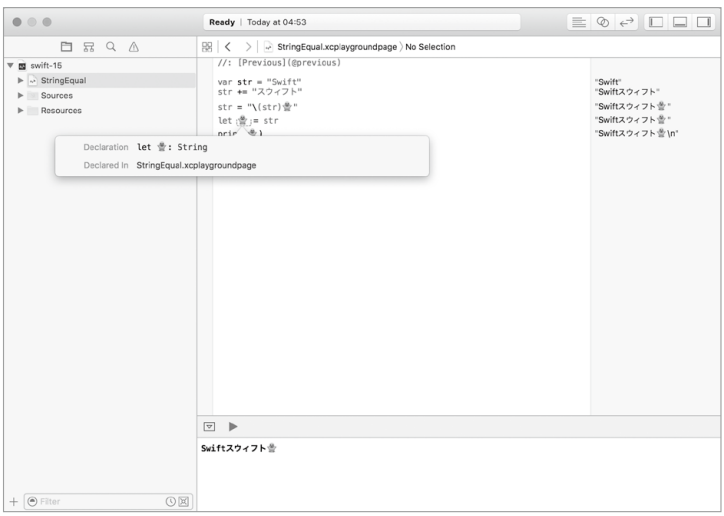
Swiftにおける文字[列]?

ずいぶんと前置きが長くなりました。それではSwiftの文字を実際に見てみましょう。次のコードをplaygroundかREPLで実行してみてください(**図1**)。

```
var str = "Swift"
str += "スウィフト"
str = "\(str)"
let  = str
print()
```

Swift スウィフト とprintされたはずです。たった5行のコードでも、これだけのことがわ

▼図1　Swiftにおける文字コードの出力結果



かります。

・型宣言は不要。リテラルから適切に推論される
・ちなみに型の名前はStringo。Xcodeなら識別子をoption+clickすれば確認できる
・文字列リテラルは""で囲まれた内部
・\()で変数展開(interpolation)
・識別子(identifier)にASCII以外のUnicodeを用いることもできる

もっともこのレベルのサポートは、Perl 5.8がもう12年以上前に実現していました。Swiftがすごい——熟練プログラマの多くがやりすぎると感じるかも——のはここからです。次のコードをご覧ください。

```
let me  = "だん"
let me2 = "\u{3060}\u{3093}"
let nfd = "\u{305f}\u{3099}\u{3093}"
me == me2
me == nfd
```

me == me2がtrueなのは当然としても、me == nfdまでもtrueになるのです。なぜ等しいか？

ひらがなの「だ」一文字と「た」+濁点が等しいとSwift(正確にはfunc ==(_:String,_:String)->Bool)がみなしているからです。

文字とは何か？　等しい文字とは何か？　文字列を文字にバラして見てみましょう。

```
for c in "\u{305f}\u{3099}\u{3093}".characters {
    print(c, String(c).unicodeScalars.count, ➥
String(c).utf8.count)
}
```

結果は次のとおりになるはずです。

```
だ 2 6
ん 1 3
```

Swiftにおける「文字」=Characterは、1byteではもちろんなく、1 code pointですらなく、1 graphemeなのです。graphemeというのは難しい言葉ですが、OS Xの辞書によると「書記素(書き言葉の最小単位)」だそうです。「文字列の比較はgraphemeをもってせよ」というのは確かにUnicode Consortiumの文書[注3][tr15]にあるのですが、これをきちんと言語レベルで実装しているのは筆者の知る限りSwiftだけです。

これはある意味、1.0時点でのUnicodeの理念を実現した格好にもなっているのですが、その後Unicodeが現実に対してずいぶん妥協したのは前述のとおりで、実際"ﾀﾞﾝ" == "ダン"、半角カナの"ﾀﾞﾝ"と全角カナの"ダン"を==で比較してもfalseとなります。

「Unicode潔癖症」？

2.2におけるSwiftの「Unicode潔癖症」は、添字(Subscript)にも見られます。たとえばJavaScriptでは(ES5以降は正式に)、

```
"JavaScript"[4] // "S"
```

という具合に文字列を文字の配列とみなして文字を取り出すことができますが、

```
"Swift"[4]
```

は"t"ではなく(そのままでは)エラーです。だからと言って添字を使えないわけではなく、次のようにすれば似たようなことはできます。

```
let str = "Swift"
let idx = str.startIndex
str[idx.advancedBy(4)]
```

これを利用すれば、extensionを使ってIntによる添字を後付けすることは一応できます。

```
extension String {
    subscript(idx:Int)->Character {
        return self[self.startIndex.➥
advancedBy(idx)]
    }
}
"Swift"[4]// "t"
```

注3) http://unicode.org/reports/tr15/

しかし、デフォルトでそうしようとすればできるのに、今のところはそうなっていません。文字列を文字に分解する際も、わざわざ.charactersというメソッドを経由しています。Swiftに限らずありとあらゆるソフトウェアは理念と現実の狭間にありますが、Swiftの組込み型の中で、文字と文字列の扱いは突出して理念が先行しているように筆者は感じています。

String... the final frontier?

前回「Swiftの'ミステリー'」として、一重引用符''がSwiftではまだ使われていないことを指摘しました。このことはSwiftの現在の文字列の振る舞いに対し、中の人がまだ試行錯誤している証なのかもしれません。実際Swiftの文字列処理は、C/C++/Objective-CよりずっとマシとはいえPerlやRubyはおろか、JavaScriptにも劣るというのが実感です。Swiftの当初の「檜舞台」がモバイルアプリケーション開発であることを考えれば、それでもObjective-Cしかなかったころに比べればずっとマシではあるでしょうし、最悪ややこしい処理はサーバに丸投げしちゃっても良いとはいえ。

しかしオープンソース化され、Linuxにも移植され、サーバサイドでも使われるようになるごく近い将来、現状のSwiftの文字列処理の貧弱さが気がかりです。

裏を返せば、そこが宝の山という見方もできなくもありません。Swiftライブラリで一旗上げるなら、文字列処理を狙うのもよさそうです。SD

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）

2015年12月号

第1特集
［決定版］Docker自由自在
実用期に入ったLinuxコンテナ技術

第2特集
ネットワーク・システム管理の定石
SNMPの教科書

短期連載
・クラウド時代のWebサービス負荷試験再入門

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。

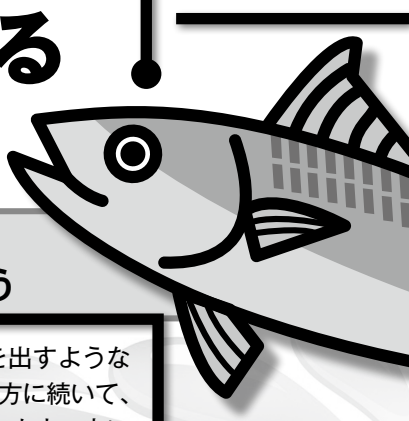
デジタル版のお知らせ　DIGITAL

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。

Mackerelではじめるサーバ管理

Writer 松木 雅幸(まつき まさゆき) ㈱はてな
Twitter @songmu



第16回 mackerel-agentのチェックプラグインを書いてみよう

Mackerelではメトリックの監視以外にも、単にOKかNGかでアラートを出すような「チェック監視」も行えます。前回のカスタムメトリックプラグインの書き方に続いて、今回はそのチェック監視を行うためのプラグインを自作する方法を解説します。良いプラグインが完成したら、ほかのMackerelユーザと共有しましょう！

前回は、Mackerelのプラグインの仕様と、カスタムメトリックプラグインの書き方について説明しました。今回はチェックプラグインの仕様と作り方について説明していきます。

チェックプラグインとは

カスタムメトリックプラグインとの使い分け

前回紹介したカスタムメトリックプラグインとチェックプラグインはどのように使い分ければ良いのでしょうか。

継続的にメトリックを取得し、閾値(しきいち)設定を行って監視したい場合には「メトリック監視」が適しており、任意の値を投稿したい、グラフ定義を指定したいといった場合、カスタムメトリックプラグインを使います。

逆に、定期的に単なるOK/NGのチェックだけをしたいものに関しては「チェック監視」が適しており、これにはチェックプラグインを使います。具体的にはログのキーワード監視や、プロセスの死活監視、ポート監視、接続監視、URLのステータス監視などがチェック監視に適しています。また、数値を取るにしても継続的に可視化する必要がないものに関しても、チェック監視で十分な場合があるでしょう。

▼表1　チェックプラグインの仕様

終了ステータス	状態
0	OK
1	WARNING
2	CRITICAL
0,1,2以外	UNKNOWN

チェックプラグインの仕様

少しおさらいになりますが、チェックプラグインの仕様についてあらためて説明します。

多くの監視システムで使われている、チェック監視プラグインのための共通仕様があります。それは、プログラムの終了コードで、監視対象の状態を表現するものです。これはPOSIXで、終了コードが0だと正常終了、それ以外だと異常終了であるという仕様を拡張したものです。終了コードと状態の対応は**表1**のとおりです。

プラグイン実行時の標準出力は、補助的なメッセージとして利用されます。このようにシンプルな仕様であり、bashやPerl、Python、Rubyなどあらゆる言語で記述できます。このチェック監視プラグインの共通仕様はMackerelのチェック監視だけではなく、次のソフトウェアでも採用されています。

- Nagios NRPE(Nagios Remote Plugin Executor)
- Sensu check plugin
- Consul script check

つまり、NagiosやSensuのチェックプラグ

インを、そのままMackerelのチェックプラグインとして利用できるということです。これは既存の監視システムからMackerelに監視設定を移行したいといったときに、非常に便利です。

シェルスクリプトによる例

たとえば、サイコロのように1〜6のランダムな数値を返し、その数値が1〜3であればOK、4か5であればWARNING、6の場合はCRITICAL、というチェックプラグインは次のように書けます。

```
#!/bin/sh
dice=$(($RANDOM%6) + 1))
echo "Roll a dice and get a $dice!"
case "$dice" in
  [1-3]) exit 0;;
  [45])  exit 1;;
  "6")   exit 2;;
esac
```

実際にこれをdice.shという名前で保存して、実行権限を付けて実行してみると次のような出力が表示されます。

```
% ./dice.sh
Roll a dice and get a 3!
```

この場合、3の目が出ています。終了コードがどうなっているかecho $?として確かめてみましょう。

```
% echo $?
0
```

1〜3は正常終了ですので、終了コードは0となっています。ちゃんと動いてそうです。では、dice.shを6が出るまで何度か実行し、その後終了コードを確かめてみましょう。

```
% ./dice.sh
Roll a dice and get a 6!
% echo $?
2
```

終了コードはちゃんと2(CRITICAL状態)になっていました。

公式のヘルパーライブラリを用いたカスタムメトリックプラグインの作成

さて、いよいよ実際のプラグインの作成方法を解説していきます。チェックプラグインの作成には、github.com/mackerelio/checkers[注1]というユーティリティライブラリを利用すると便利です。公式プラグインはすべてこのライブラリを利用して作成されています。また、公式のチェックプラグインでは、コマンドライン引数のパースにgithub.com/jessevdk/go-flags[注2]を統一的に利用しています。

チェックプラグインのソースコード構成

❶package宣言とimport文
❷コマンドライン引数用のstructの定義
❸main()関数の定義
❹run()関数の実装

ここでは、check-uptime[注3]を例にとってそれぞれを見ていくことにしましょう。

❶package宣言とimport文(リスト1)

そのまま実行コマンドとなるのでpackageはmainで宣言します。そのほか必要なライブラリのimportを行います。

▼リスト1　package宣言とimport文

```
package main

import (
    "fmt"
    "os"
    "time"

    "github.com/jessevdk/go-flags"
    "github.com/mackerelio/checkers"
    "github.com/mackerelio/golib/uptime"
)
```

注1）URL https://github.com/mackerelio/checkers
注2）URL https://github.com/jessevdk/go-flags
注3）URL https://github.com/mackerelio/go-check-plugins/tree/master/check-uptime

▼リスト2　コマンドライン引数用のstructの定義

```
var opts struct {
    WarnUnder *float64 `short:"w" long:"warn-under" value-name:"N" description:"Trigger a warning
 if under the seconds"`
    CritUnder *float64 `short:"c" long:"critical-under" value-name:"N" description:"Trigger a critial
 if under the seconds"`
    WarnOver  *float64 `short:"W" long:"warn-over" value-name:"N" description:"Trigger a warning
 if over the seconds"`
    CritOver  *float64 `short:"C" long:"critical-over" value-name:"N" description:"Trigger a critical
 if over the seconds"`
}
```

▼リスト3　main()関数の定義

```
func main() {
    ckr := run(os.Args[1:])
    ckr.Name = "Uptime"
    ckr.Exit()
}
```

❷コマンドライン引数用のstructの定義（リスト2）

プラグインが受け取るコマンドライン引数の設定を行います。この設定方法は、github.com/jessevdk/go-flagsの仕様に準じますので、詳しくは、go-flagsのドキュメント[注4]をご覧ください。

コマンドライン引数では、閾値の設定やミドルウェアのポートなどを指定します。check-uptimeの場合、uptimeが一定の時間を下回っていた場合、上回っていた場合それぞれのために、warn-under、critical-under、warn-over、critical-overという4つのオプションの設定を可能にしています。

❸main()関数の定義（リスト3）

main()関数はコマンドのエントリポイントですが、3行だけと非常にシンプルです。run()関数により、*checkers.Checkerオブジェクトを受け取り、それにNameの設定を行って終了しています。Nameは、プラグイン実行時に先頭に表示される単語を統一的に指定するためのものです。たとえば、check-uptimeを実行すると**図1**のような出力が表示されますが、この出力の先頭のUptimeがそれにあたります。

❹run()関数の実装（リスト4）

check-uptimeの処理は、ほとんどこのrun()関数の中で実装されています。run()関数のシグネチャはfunc run(args []string) *checkers.Checkerとなっています。これは、コマンドライン引数を受け取り、監視結果オブジェクトである*checkers.Checkerを返すという処理になります。

コメントで補足していますが、このrun()関数も、おおまかに次の4つの部分に分けられます。

(1)コマンドライン引数の処理
(2)uptimeの取得
(3)閾値比較と終了ステータスの決定
(4)出力メッセージの組み立て

●コマンドライン引数の処理

コマンドライン引数の処理では、flags.ParseArgs()によりコマンドライン引数の内容がoptにマッピングされます。

●uptimeの取得

github.com/mackerelio/golib/uptimeパッケージのuptime.Get()を利用してuptimeを取得し

▼図1　check-uptimeを実行

```
% check-uptime
Uptime OK: 0 day(s) 3 hour(s) 17 minute(s) (11877 second(s))
```

注4）URL https://godoc.org/github.com/jessevdk/go-flags

▼リスト4　run()関数の実装

```
func run(args []string) *checkers.Checker {
    _, err := flags.ParseArgs(&opts, args) //(1)コマンドライン引数の処理
    if err != nil {
        os.Exit(1)
    }
    ut, err := uptime.Get() //(2)uptimeの取得
    if err != nil {
        return checkers.Unknown(fmt.Sprintf("Faild to fetch uptime metrics: %s", err))
    }

    //(3)閾値の比較と終了ステータスの決定
    checkSt := checkers.OK
    if opts.WarnUnder != nil && *opts.WarnUnder > ut {
        checkSt = checkers.WARNING
    }
    if opts.WarnOver != nil && *opts.WarnOver < ut {
        checkSt = checkers.WARNING
    }
    if opts.CritUnder != nil && *opts.CritUnder > ut {
        checkSt = checkers.CRITICAL
    }
    if opts.CritOver != nil && *opts.CritOver < ut {
        checkSt = checkers.CRITICAL
}

    //(4)出力メッセージの組み立て
    dur := time.Duration(ut * float64(time.Second))
    hours := int64(dur.Hours())
    days := hours / 24
    hours = hours % 24
    mins := int64(dur.Minutes()) % 60
    msg := fmt.Sprintf("%d day(s) %d hour(s) %d minute(s) (%d second(s))¥n", days, hours, mins, int64(
(dur.Seconds()))
    return checkers.NewChecker(checkSt, msg)
}
```

ています。

●閾値比較と終了ステータスの決定

opt内の閾値と比較しながら、終了ステータスの決定を行います。終了ステータスには、checkers.OK/WARNING/CRITICAL/UNKNOWN 定数を用います。

●出力メッセージの組み立て

出力メッセージを組み立て、最後に*checkers.Checkerオブジェクトを返しています。ここではreturn checkers.NewChecker(checkSt, msg)のようにして、returnしていますが、終了ステータスが決まっている場合、return checkers.Critical("message")と、簡潔に書ける機能もcheckersには備わっているので、状況に応じて使い分けてください。

これで一通り、チェックプラグインの作成方法を説明しました。みなさんもぜひチャレンジしてみてください。

プラグインをより洗練させる

これまでで、Go言語でのカスタムメトリックプラグインとチェックプラグインそれぞれの書き方についてざっと説明しましたが、よりしっかりしたプラグインを書くための方法を最後に説明します。具体的には、テストの書き方と、公式のプラグイン集へのcontribute方法についてです。

テストを書く

Go言語でのテストの作法に従って、plug

inname_test.goなどのようなファイル内にテストを書くと良いでしょう。標準のtestingパッケージだけでかまいませんが、github.com/stretchr/testify[注5]などを補助的に使っても良いと思います。

Mackerelのプラグインは、コマンドを実行するなどして外部から値を取得することが多いため、テストが書きづらい場合があります。

そういった場合は、コマンド実行処理とパース処理を分けて、パース処理のテストを書くのが定石です。つまり、コマンド実行処理が文字列を返し、その文字列をパース処理に受け渡すようにすれば良いのです。具体的には**リスト5**のような形です(エラー処理は簡単のために省略してあります)。

このようにして、parseOut()のテストをしっかり書くと良いでしょう。環境やミドルウェアのバージョンによって、出力が異なる場合もあるので、それを留意してテストを充実させられるとなお良いです。テストの書き方は、ほかの公式プラグインのテストも参考にしてみてください。

公式プラグイン集にcontributeするために

便利なプラグインができたら、ぜひ公式プラグイン集にpull requestを送ってください。そのためのガイドラインを説明します。独自にプラグインを作成する場合でも、このガイドラインに沿っておくと良いでしょう。

● forkして開発する

カスタムメトリックプラグインであれば、https://github.com/mackerelio/mackerel-agent-plugins、チェックプラグインであれば、https://github.com/mackerelio/go-check-pluginsをそれぞれforkしてください。

作りたいプラグインに応じて、カスタムメトリックプラグインの場合はmackerel-plugin-*、チェックプラグインの場合はcheck-*というディレクトリを作り、その中でプラグインの開発を行ってください。

▼リスト5　コマンド実行処理とパース処理を分ける

```
out, _ := exec.Command("/path/to/cmd", args...).Output()
result := parseOut(string(out))
```

● テストと構文チェック

クオリティ担保のために最低限のテストを書いてください。また、gofmtによるコードフォーマットと、go vet、golintによる構文チェックを行うようにしてください。構文チェックは、それぞれのリポジトリのCIでテストされるようになっており、エラーがある場合はマージができないようになっています。

● READMEを書く

README.mdをプラグインディレクトリに配置してください。README.mdのフォーマットは自由ですが、DescriptionとUsageは含むようにしてください。ほかのプラグインのREADME.mdを参考にするのもお勧めです。

これでプラグインの体裁は整ったので、あとは、pull requestを送ってみてください。お待ちしています。

おわりに

前回と今回でプラグインの書き方を説明しました。最初は複雑に思えるかもしれませんが、慣れてくると非常に簡単です。ぜひ、自分でプラグインを書いてみて、便利な使い方をどんどん共有してください。SD

注5）URL https://github.com/stretchr/testify

ブラウザハック

Wade Alcorn、Christian Frichot、Michele Orru 著／園田 道夫、はせがわようすけ、西村 宗晃 監修／株式会社プロシステムエルオーシー 訳
B5変型判／496ページ
4,600円＋税
翔泳社
ISBN＝978-4-7981-4343-9

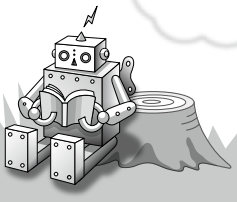
本書は、Webブラウザをハッキングし攻撃に利用する方法を解説する。攻撃とはつまり、他者のブラウザに任意のコードを送りこみ、実行させる手段のこと。標的となる人物に、コードを埋め込んだWebサイトを開かせる手法が一般的なようだ。そのためには、標的者が閲覧しそうなサイトにコードを埋め込む、偽サイトを作りフィッシングで誘導するなどが挙げられる。偽サイトがばれないように、URLを難読化しよう、QRコードで誘導するのも効果的……。そんな調子で、本格的な攻撃手法（セッションを奪い他者になりすます、証明書を改ざんするなど）を次々に扱う。もちろん本書の目的は、ブラウザのリスクを理解し、セキュリティに活かすこと。理論でしか学んだことのない攻撃手法を、実際のコードレベルで理解するのに役立つ。

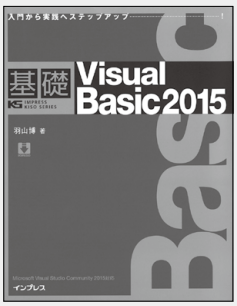
Python チュートリアル 第3版

Guido van Rossum 著／鴨澤 眞夫 訳
A5判／256ページ
1,800 円＋税
オライリー・ジャパン
ISBN＝978-4-87311-753-9

Pythonは汎用のプログラミング言語ながら、データサイエンスに適した言語ということで近年注目されている。その理由としては、データ分析ツール「pandas」、ディープラーニングのためのライブラリ「Chainer」といった外部機能が充実している点が挙げられる。またスクリプト言語なので、「アイデアをすぐに実現できる」「データ分析のタスクに集中できる」という利点も影響しているという。本書はチュートリアルと銘打っているだけあって、言語仕様と基本的な文法といった初歩の初歩を解説している。最初から最後まで、書かれているコードを実行しながら読み通すことで、基礎を十分にカバーできるだろう。データ分析にPythonを使いたいと思っている人も、まずは基礎を固めるために、この1冊を読破することをお勧めする。

SD BOOK REVIEW





基礎 Visual Basic 2015

羽山 博 著
B5変型判／416ページ
3,200円＋税
インプレス
ISBN＝978-4-8443-8020-7

Visual Studio Community 2015上で、Visual Basicを使い、GUIの「Windowsデスクトップアプリ」を作るための方法を解説した入門書である。Visual Studioのインストールを行い、コントロールの配置、プロパティの設定、イベントハンドラーの作成という開発の大きな流れを説明したあとは、Visual Basicの言語仕様・文法についての解説が続く。

Windows 10が登場し、Windows Phoneがメーカーから続々と発表されたことで、その上で動く「ユニバーサルWindowsアプリ」は今後大いに注目されるだろう。本書にはユニバーサルアプリ開発についての記載はないが、それに応用できるVisual Basicの基礎部分の習得には役立つはずだ。

ドキュメント作成システム構築ガイド

伊藤 敬彦、吉村 孝広 著
A5判／208ページ
2,480円＋税
技術評論社
ISBN＝978-4-7741-8036-6

作成した技術文書が読みにくいと、エンジニアがせっかく制作した成果物の使用方法がわからなかったり、サポート対応が増えたりしてしまう。わかりやすい技術文書を作成するスキルは今やエンジニアに必須と言える。そこで本書では、ソフトウェア開発で取り入れられている技法を用いてドキュメント作成システムを構築する。Gitを用いたバージョン管理、GitHubによる共同編集、RedPenによる品質チェック、CIツールによる継続的改善など、エンジニアが普段使用しているツールを本書にならって利用すれば、技術文書の品質は上がるだろう。応用としてAsciidoctorによるドキュメントのスタイル調整についても解説している。技術文書を作成する際にはぜひこの書籍を参考にしてみてほしい。

Sphinxで始めるドキュメント作成術

清水川 貴之 SHIMIZUKAWA Takayuki Twitter @shimizukawa

第15回 ドキュメント翻訳フローの自動化

「みんなで翻訳」に必要なこと

前回の本連載では、Sphinxの国際化機能を使ってドキュメント翻訳を行うメリットと手順を紹介しました。今回は、翻訳を複数人で行うための環境の作り方と、その手順を自動化する方法を紹介します。

Sphinxの国際化機能を使えば、gettextのメッセージカタログ形式(.po)で翻訳できます。このしくみを使うことで、複数の言語に翻訳しやすく、原文の更新への追従が簡単になります。そのため、翻訳しているドキュメントの翻訳済み文章の維持や、新たに追加された部分の翻訳がしやすくなります。また、支援ツールsphinx-intlによってgettextの煩雑な作業からも解放され、ドキュメントの翻訳に集中できます。

前回は、次の手順を紹介しました(**図1**)。

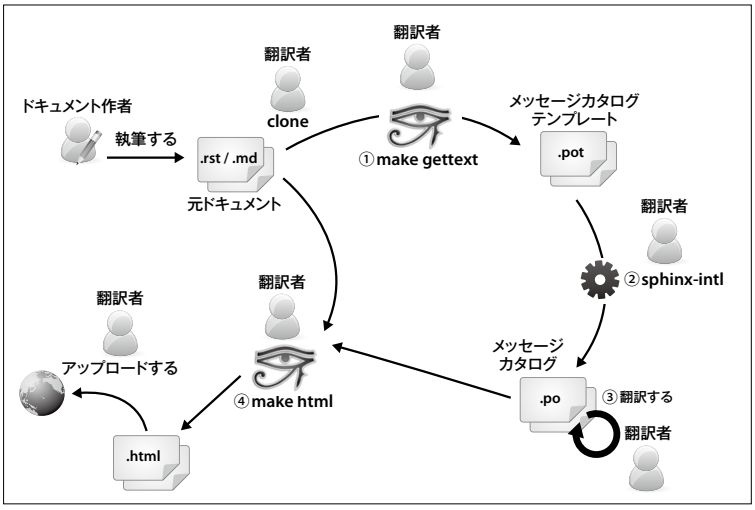
①make gettextコマンドで元ドキュメントから.potファイルを生成
②sphinx-intl updateコマンドで.potファイルから日本語用に.poファイルを生成
③.poファイルを編集して翻訳
④make html SPHINXOPTS=-D language=jaコマンドで言語を指定してHTMLをビルド

上記の手順で煩雑な作業はなくなります。しかし、翻訳そのものにかかる時間は短くなりません。また、一度翻訳が終わっても原文の更新は続いていきます。このようなドキュメントの翻訳を長期間維持し、最新の翻訳を提供し続けるには、ほかの人の協力が必要になることも多いでしょう。

しかし、いざ「みんなで翻訳しよう！」と思って複数人で翻訳する方法を考え始めると、ファイルの受け渡しや分担をスムーズに行うのが意外と難しいことに気がつきます。

ファイルの受け渡し方法としてまず思いつくのは、.poファイルをメールなどで渡して翻訳してもらう方法です。一見わかりやすい方法に思えます

▼図1 Sphinxでの翻訳に必要な手順



が、大きな単位での依頼となるため、お願いしづらかったり、協力しづらいという問題があります。また、進捗の共有をリアルタイムに行えないため、作業範囲の行き違いなどの問題が起きやすい方法です。

では、バージョン管理ツールのGitを使う方法はどうでしょうか。Gitはソースコードの管理ツールとしてとても優秀ですが、誰もがすぐに使える簡単なツールではありません。みんなにGitの操作を覚えてもらったとしても、編集個所がコンフリクト(競合)したらマージが必要ですし、間違えてほかの人の翻訳を上書きしてしまうこともあります。Dropboxなどのファイル共有サービスは比較的わかりやすいツールでしょう。競合や上書きの問題はまだ残りますが、始めやすいのが良いところです。

ファイルの受け渡し以外にも課題があります。ドキュメントに翻訳を反映して確認するには、Sphinxのビルド環境を用意する必要があります。しかし、`make html`を実行してHTML出力をビルドするには、PythonやSphinxをインストールしなければなりません。翻訳を分担しているみんなが環境を整えるのは「簡単」とは言えないでしょう。

環境を整えるためにツールのインストール方法や使い方、手順などをどこかにまとめておくと負担が少なくなります。しかし、翻訳を始めるまでの準備作業が多いほど、協力しようという気持ちはしぼんでしまいます。多くの人に気軽に協力してもらうためには、必要な事前準備は最小限にして、翻訳だけに集中できる環境があると良いでしょう。

そこで、多人数で翻訳を進めるためのサービスとして、Transifex[注1]を紹介します。

翻訳支援サービス Transifex

TransifexはSphinxの公式ドキュメントの翻訳[注2]でも使用されている翻訳支援サービスです。Sphinx以外にも、Python公式ドキュメント[注3]や、Django公式ドキュメント[注4]の翻訳に使用されています。

Sphinx公式ドキュメントの翻訳でTransifexを利用したのは次のメリットがあったからです。

- サービス提供されているので翻訳者がすぐに参加できる
- 複数の翻訳者が同時に翻訳する前提で作られているので、トラブルが起こりづらい
- API／クライアントツールが提供されていて、メッセージカタログのアップロードとダウンロードができる
- オープンソースプロジェクトの翻訳には無料で利用できる
- 翻訳支援機能が充実している

一般的な翻訳支援機能について少し詳しく説明します。翻訳支援は大きく2つの機能に分けられます。1つはGoogle Translateなどの機械翻訳です。もう1つはTransifexなどのコンピュータ翻訳支援(CAT：Computer Assisted Translation)です。CATツールの代表的な機能に翻訳メモリ(Translation Memory)と用語集があります。

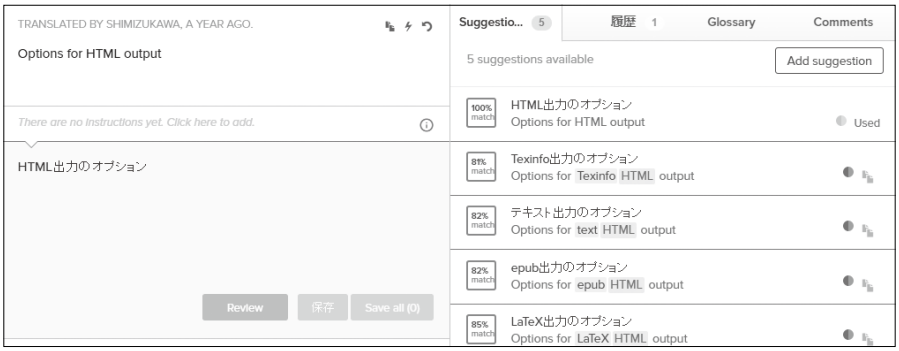
翻訳メモリは、すでに翻訳された文章を記憶しておいて、似たような原文があれば提示してくれます(**図2**)。この機能によって、似たような文章を何度も翻訳する労力をかけなくてよくなります。さらに、複数人で作業しても翻訳のゆれが少なくなり、文章全体の一貫性や品質が向上します。

また、gettextでは1文字変更しただけでも既存の翻訳文にはfuzzyフラグが立ちSphinxにおいて無効扱いとなりますが、Transifexでは過去の翻訳文を提示し、どこに差分があるのかハイライト表示してくれます。そのため、翻訳しな

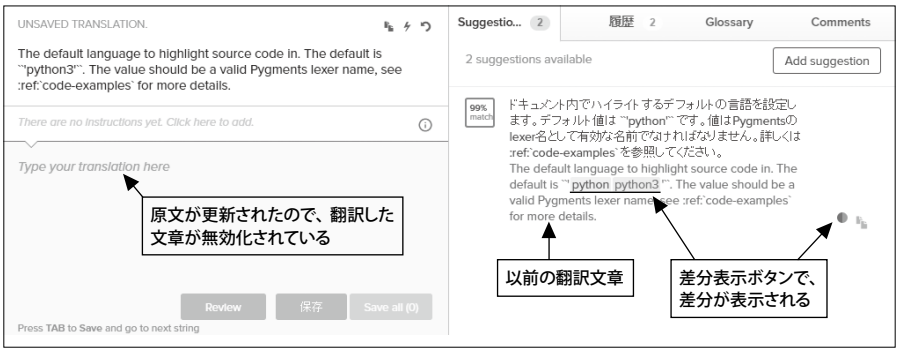
注1) https://www.transifex.com/

注2) https://www.transifex.com/sphinx-doc/sphinx-doc-1_4/
注3) https://www.transifex.com/python-doc-ja/public/
注4) https://www.transifex.com/django/django/

▼図2　翻訳メモリからの提案（似ているけど少し異なる文章）



▼図3　翻訳メモリからの提案。単語を1つ変更した文章



COLUMN

CATツール

世の中にはさまざまなCATツールがあります。ローカル環境にインストールするもの、OSSで公開されておりサーバを自分で立てて利用するもの、サービス提供されていてすぐに使えるもの、扱えるファイル形式が豊富なもの、有料なもの、無料なもの、など特徴もさまざまです。

Transifex以外にもよく使われるCATツールを**表A**に挙げます。なお、Sphinxと連携するにはgettextのメッセージカタログ形式を扱える必要があります。

また、ツールや翻訳の一般的な話題について紹介しているサイト「http://localization-guide.readthedocs.org/」（英語）があります。こちらも参考にしてみてください。

▼表A　Transifex以外のCATツール

名称	ツールの形態・ライセンス・サポートする形式など
OmegaT	Javaで実装されたデスクトップアプリケーション。GPLライセンスで提供されている。ファイルの入出力はgettextやHTMLなど非常に多くの形式をサポートしている。 http://www.omegat.org/
Pootle	Pythonで実装されたサーバアプリケーション。GPLライセンスで提供されている。ファイルの入出力はgettextやXLIFFなど多くの形式をサポートしている。 http://pootle.translatehouse.org/
Wordfast	デスクトップアプリケーション。Word、Excel、PowerPoint、などのMicrosoft製品のファイルを扱える。gettextフォーマットも変換ツールを使って扱える。 http://www.wordfast.com/
Trados	プロ向けの有償ツール。高い品質で効率的に翻訳を進めることができる。 http://www.sdl.com/jp/

おす個所を見つけるのに時間をかけずに済みます(**図3**)。

それでは、Transifexを使ってどのように翻訳を進めていくのか見ていきましょう。

SphinxとTransifexを連携する

Transifexはメッセージカタログ形式のファイルを、TransifexのWebページやAPIを通じてアップロード、ダウンロードできます。また、公式のコマンドラインツールtransifex-client[注5]を使えば複数ファイルのアップロード、ダウンロードを手軽に行えます。

transifex-clientのインストールは次のように行います。

```
$ pip install transifex-client
```

これでtxコマンドが使用できるようになります[注6]。txコマンドでは、.potファイルをTransifexにアップロードするtx push -sコマンドと、指定した言語の.poファイルをTransifexからダウンロードするtx pull -l jaコマンドの2つをおもに利用します。

Sphinxの国際化機能とtxコマンドを組み合わせると、手順は**図4**、**5**のようになります。

この一連のコマンドを自動的に実行することで、翻訳作業以外の手順を自動化できます。自動化を実現するにはCI(継続的インテグレーション)環境の利用が有効です。自前でサーバを用意しJenkins[注7]などで定期的に実行する方法や、Drone.io[注8]やTravis CI[注9]などのサービスを利用する方法などがあります。

Sphinx-users.jpでは、Drone.ioを使用してSphinx公式ドキュメントの日本語訳を自動ビルドしています[注10]。また、筆者がPyCon JP 2015で発表した資料[注11]でも自動化の例を紹介してい

注5) http://docs.transifex.com/client/

注6) Windowsでは、Pythonの Scriptsディレクトリにあるtxを「python Path¥To¥Scripts¥tx」のように実行する必要があります。

注7) https://jenkins.io/

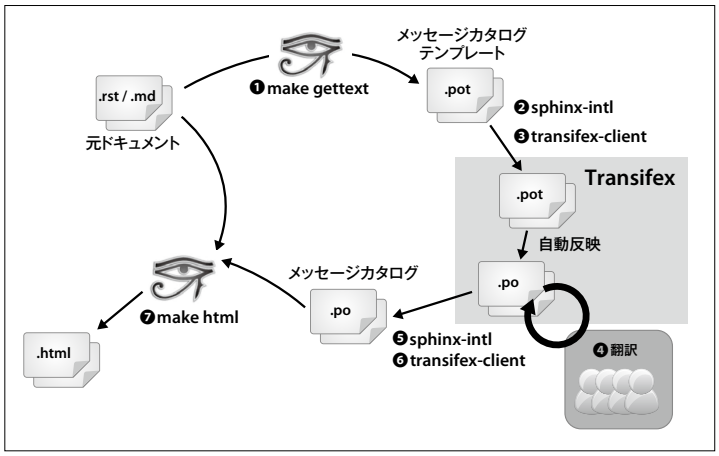
注8) https://drone.io/

注9) https://travis-ci.org/

注10) https://drone.io/bitbucket.org/shimizukawa/sphinx-doc14/admin

注11) http://www.slideshare.net/shimizukawa/sphinx-53764167

▼図4 SphinxとTransifexを連携して翻訳する手順(図解)



▼図5 SphinxとTransifexを連携して翻訳する手順

▼前準備
・sphinx-intl create-transifexrcでtransifexの接続設定を行う
・sphinx-intl create-txconfigでtransifexのプロジェクト設定を行う

▼手順(番号は図4中の番号を指す)
❶make gettextで元ドキュメントから.potファイルを生成
❷sphinx-intl update-txconfig-resourcesでTransifexと連携するファイル一覧を設定
❸tx push -sで.potファイル群をTransifexにアップロード
❹TransifexのWebコンソールでメッセージを翻訳
❺❷と同じコマンドで連携ファイル一覧を更新
❻tx pull -l jaで日本語に翻訳した.poファイル群をダウンロード
❼make html SPHINXOPTS=-D language=jaで翻訳したドキュメントのHTMLをビルド

ますので、参考にしてみてください。

翻訳に参加する

翻訳者は、Transifexにホスティングされているメッセージカタログを Web 画面上で翻訳します。このとき必要になるのは、インターネット接続とブラウザ、Transifexのアカウントです。手元のコンピュータにソフトウェアをインストールする必要はありませんし、メッセージカタログのダウンロードやアップロード、ドキュメント原文の更新や、翻訳の競合についても考える必要はありません。

Transifexでの翻訳は**図6**のようにWebコンソールで行います。翻訳者は、**図6**の(1)～(3)の手順で翻訳を進めていきます。

(1)メッセージ一覧から翻訳対象のメッセージを選択し、
(2)原文フォームに表示された内容を読み、
(3)翻訳文フォームに入力し、保存ボタンで保存する

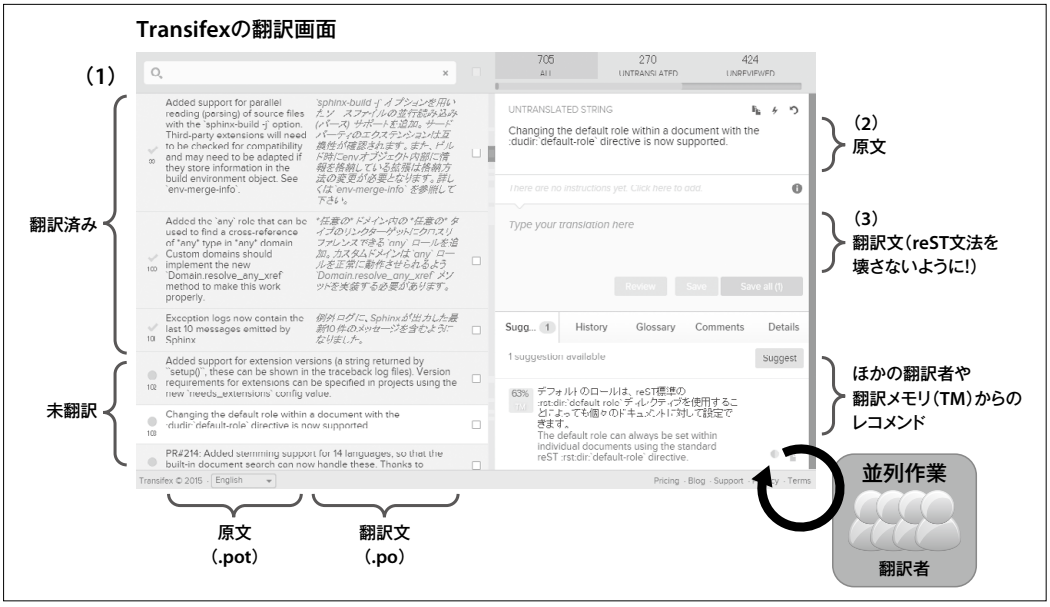
これを繰り返します。Transifexの場合、外部サービスと連携することでこのフォーム上で機械翻訳もできます。機械翻訳した結果をそのまま使うと、おかしな日本語になってしまうことがほとんどですが、翻訳作業の助けになることもあります。メッセージによってはほかの翻訳者や翻訳メモリからの提案が表示されるので、内容を確認して使用するのも良いでしょう。

Transifexでは、翻訳がどのくらい進んでいるかが一覧で表示されます(**図7**)。最終目標はこの達成率を100%にすることですが、100%になれば翻訳に誤りがあっても良いというわけでもありません。オープンソースのドキュメントを翻訳していておかしな翻訳文を見つけたなら、遠慮することなく修正していきましょう。

もし間違った翻訳文を採用しないようにチェックしたいのであれば、Transifexのレビュー機能が使えます。Transifexは.poファイルのダウンロード時に、未レビューのメッセージを含めるかどうかを選択できます。レビュー担当者を決めて、すべてのメッセージをレビューすることで間違った翻訳文が採用されないようにできます。翻訳の進捗率(**図7**)には、レビュー済み率も濃い色で表示されています。

ちなみに、Sphinx公式ドキュメントの翻訳で

▼図6　TransifexのWebコンソールでの翻訳作業



は、未レビューを含むすべての翻訳文をそのまま採用しています。ときどき間違っている翻訳文がそのまま公開されていますが、レビューのためのコストをかけるよりも、気づいた人が直せば良い、という運用にしています。

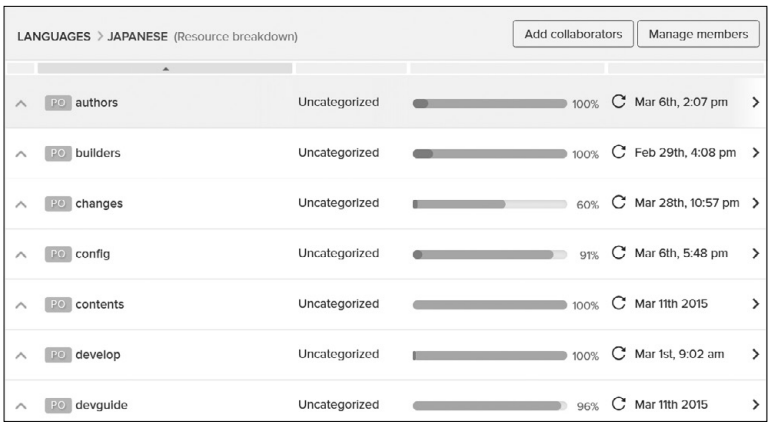
なお、Sphinx-users.jpでは、公式ドキュメントの翻訳を手伝ってくれる方を募集しています。本記事の練習がてらやってみたいという方は、Transifexのページ[注12]にある「sphinx-1.4の翻訳にご協力ください」[注13]から参加登録してみてください[注14]。

次回予告

次回は、Sphinxで運用ドキュメント、作業手順書を書いていきます。Sphinxのincludeディレクティブやreplaceの記法を使ってドキュメントを変数化する方法について紹介します。SD

注12) https://www.transifex.com/sphinx-doc/sphinx-doc-1_4/
注13) 英語ページの場合は、「HELP TRANSLATE "sphinx-doc-1.4"」。
注14) Sphinx-users.jpのSlackでチャットによるサポートも行っています。チャット参加はこちらからどうぞ。 http://sphinxjp.herokuapp.com/

▼図7 Transifexで翻訳率の一覧を表示（Sphinx公式ドキュメントの日本語翻訳率）



COLUMN

Sphinx-1.4.1リリース

2016年4月12日に、Sphinx-1.4.1をリリースしました[注A]。1.4系は1年ぶりのメジャーバージョンアップで、1.3.6から49の新機能、17の非互換性、40のバグ修正があります[注B]。

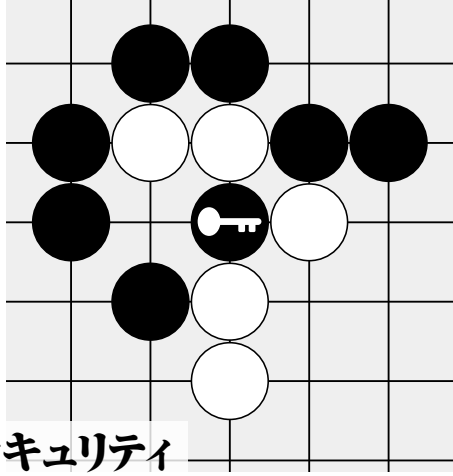
新機能のいくつかを紹介します。

- ドキュメントの国際化で、言語ごとに別の画像ファイルを使用可能。画像に文字が書かれている場合などに便利。conf.pyのlanguageとfigure_language_filenameを設定することで利用できる
- 2つのビルダー「dummy」と「epub3」を追加。dummyは何も出力せず、文法のチェックだけを行う。epub3はまだ実験的な実装
- 日本語検索に分かち書きアルゴリズムとしてJanomeを選択可能
- Sphinx拡張「sphinx.ext.githubpages」を追加。GitHub Pages用にドキュメントを生成する
- Sphinx拡張「sphinx.ext.autosectionlabel」を追加。`:ref:`のターゲットにセクションタイトルを指定できる

注A) https://pypi.python.org/pypi/Sphinx/1.4.1
注B) 不具合報告はメーリングリスト（http://sphinx-users.jp/howtojoin.html）、またはSphinxのGitHub（https://github.com/sphinx-doc/sphinx）へお願いします。

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第三三回】ソフトウェアのライフサイクルとセキュリティ

2016年3月号の本連載「セキュリティを意識したソフトウェア開発」では、ソフトウェアのライフサイクルと脆弱性について言及しましたが、今回は、QuickTime for Windowsの脆弱性の事例をもとに、ソフトウェアのライフサイクルとセキュリティについてもう少し深く考えてみます。

QuickTime Windows版にゼロデイ攻撃

2016年4月15日、JVN(Japan Vulnerability Notes)のサイトに「JVNTA#92371676　QuickTime for Windowsに複数のヒープバッファオーバフローの脆弱性」という情報が載りました[注1]。US-CERTでは「Alert(TA16-105A)　Apple Ends Support for QuickTime for Windows; New Vulnerabilities Announced」という形で報告が出ています[注2]。

このヒープバッファオーバフローの詳細に関しては、ゼロデイ・イニシアティブのサイトで公開されています[注3]。

今回のケースは、QuickTime for Windows(以下、QuickTime Windows版)の製品開発元であるApple社が、同製品のサポートを終了させたために、対策としてはソフトウェアのアップデートをするという選択肢はなく、QuickTime Windows版を使わない(アンインストールする)という選択肢のみが示されています。

あまり普及していないソフトウェアの場合や、代替品がすでにあってそれなりに旧式で時間が経っているようなソフトウェアの場合、サポートを終了するということはこれまでにもありました。しかし、QuickTime Windows版のように普及していて、かつまだ一線で使われているようなソフトウェアでは珍しいと言えるでしょう。

一般的にゼロデイ攻撃とは、ベンダがその攻撃に対応するアップデートを提供する前に攻撃が発生するという、ユーザ側には守ることができない攻撃を意味しており、ある種の奇襲攻撃とも言えます。

今回は、ベンダがアップデートを提供しないため、ユーザ側は守ることができない攻撃になっています。もちろんアンインストールすることはできますが、形式的にはこれはゼロデイ攻撃と同様と言えるでしょう。筆者はそう考えますが、ゼロデイ・イニシアティブの見解でも同様です。

ヒープバッファオーバフローの影響

ゼロデイ・イニシアティブは、QuickTime Windows版で任意の悪意のコードが実行可能である、複数のヒープバッファオーバフローの存在を指摘しています。

ヒープバッファオーバフローを使って任意のコードを実行する細工をしたQuickTimeファイルを開くと発現するわけですが、ファイルを開くという動

注1) JVNTA#92371676 QuickTime for Windowsに複数のヒープバッファオーバフローの脆弱性　https://jvn.jp/ta/JVNTA92371676/
注2) US-CERT Alert TA16-105A　https://www.us-cert.gov/ncas/alerts/TA16-105A
注3) (0Day) Apple QuickTime moov Atom Heap Corruption Remote Code Execution Vulnerability
http://zerodayinitiative.com/advisories/ZDI-16-241/
(0Day) Apple QuickTime Atom Processing Heap Corruption Remote Code Execution Vulnerability
http://www.zerodayinitiative.com/advisories/ZDI-16-242/

作は、Webサイトに掲載されているQuickTimeの動画を観賞するということでもあります。

たとえば、メールに動画が添付されているならば、怪しいので捨てることは可能です。しかし、Webサイトをブラウジングしていて不意にどこかのサイトに飛んでしまい、そこにしかけられたQuickTimeを不意に見てしまった（自動的に動画がスタートするなど）だけでも同じことが起こるでしょう。これまでもAdobe Flash Playerの脆弱性などで見られた攻撃方法なので、目新しい攻撃方法ではないのですが、攻撃側にはさらに選択肢が増えたということは理解しておかなければなりません。

重要度はCVSSv3では6.3、CVSSv2では6.8となっています。アップデートがされない以上、QuickTime Windows版がインストールされているならばアンインストールすることが妥当です。

アンインストールにも問題が

QuickTime Windows版をそのままアンインストールして問題がなければ良いのですが、QuickTimeを組み入れているソフトウェアを使っている場合があります。ソフトウェアがWindows環境でのQuickTimeを使っているが、代替品がなかったり、あるいはデータフォーマットの関係で利用しなければいけないケースもあるでしょう[注4]。

QuickTimeは、Apple製品上では動画／画像に関

注4）一例として、Adobe社の製品の例が挙げられます。
Adobe社QuickTime Windows版のサポート終了について
https://blogs.adobe.com/creativestation/video-apple-ends-support-for-quicktime-windows

◉ QuickTime 7 for Windowsをアンインストールしてみた

QuickTime Windows版（QuickTime 7 for Windows）の製品開発元のApple社が、同製品のセキュリティアップデートをしないという情報は、Trend Micro社のセキュリティブログ“Urgent Call to Action: Uninstall QuickTime for Windows Today”[注A]と、それを参照しているUS-CERT Alert TA16-105Aの告知だけです。

QuickTime 7 for Windowsを明示的に「サポートしない」とか、「セキュリティアップデートをしない」といったことを示しているドキュメントやアナウンスは、少なくとも筆者は見つけることができませんでした（2016年4月18日現在）。

Apple社のサイトのドキュメント“Uninstall QuickTime 7 for Windows”[注B]では、“Most recent media-related programs for Windows—including iTunes 10.5 or later—no longer use QuickTime to play modern media formats.”と書かれてはいます。しかし、これが今後は「サポートしない」「セキュリティアップデートをしない」と同等な意味に解釈すべきかどうかは微妙に判断が分かれるところです。

ただ、現実にはアップデートできないわけですから、US-CERTやJVNが示すように当面はアンインストールして安全性を高めておくべきでしょう。このドキュメントには、タイトルどおり、アンインストール手順が書かれているので、アンインストールの際は参考にすると良いでしょう。

さて、筆者の手元にあるWindows 10の上のQuickTime Playerをチェック（ただし、最後に立ち上げたのは1ヵ月以上前）すると、QuickTime Playerのバージョンは7.7.7、QuickTime自体のバージョンも7.7.7となっていました。確認している最中にiTunes 12.3.3とApple Software Update 2.2のインストールを指示するアップデータが自動的に立ち上がったので、その指示に従いソフトウェアをアップデートしましたが、その後もQuickTimeのバージョンはそのままでした。

Windowsの検索機能でキーワード“QuickTime”を入力し、検索した際にメニューに出てきた「QuickTimeアンインストーラ」を選択しました。そうやってインストーラを立ち上げ、削除を選び、「QuickTime for Windowsを完全に削除する」を選択したところ、QuickTime 7およびQuickTime Playerが削除されました。筆者の環境では、その後の問題はありませんでした。

注A）Urgent Call to Action: Uninstall QuickTime for Windows Today
http://blog.trendmicro.com/urgent-call-action-uninstall-quicktime-windows-today/
注B）Uninstall QuickTime 7 for Windows https://support.apple.com/en-us/HT205771

する標準的なソフトウェアです。そのため、別のプラットフォームでも、Apple社がそのプラットフォーム向けに出しているQuickTimeを使うというのも、理解できます。サードパーティーが提供するソフトウェアを使うよりも、あるいは自社で開発するよりも、優先されるのも当然かと思います。

というのも、このような画像処理や動画処理を扱う商用ソフトウェアには数々の特許がかけられており、それらを使うためには（たとえ、コードをゼロから自社で作ろうとも）特許料を払うなどの手続きが必要になるケースがあるからです。なるべくベンダが直接提供しているものを使うというのも、無理からぬ話かと思います。

今回の問題で露呈したように、一方的にベンダ側がソフトウェアを打ち切った場合の影響は大きいのです。単純にソフトウェアを作成する工数が必要というだけではありません。特許などを保持して、その技術を囲い込んでいるような場合は、互換のソフトウェアを作るにしても手続きや契約、あるいはライセンス料の支払いなど、いろいろな制約が出てきます。「なければ作る」とは簡単にはいきません。

ソフトウェアのライフサイクルとしては特殊

今回の例は、準備期間がない、あるいは極めて短い準備期間しかないベンダの一方的な打ち切りですので、雑誌などを細かくチェックしている人ならわかるでしょうが、一般のPCユーザがこのような情報をチェックしているかどうかは、疑問です。

AppleのQuickTimeというメジャーな製品ならまだ良いですが、これがWindows系PCを購入した際にプレインストールされているような存在さえよくわかっていないソフトウェアだったらどうでしょうか？　気がつかないままサポート終了になっているソフトウェアもあるはずです。

ソフトウェアのライフサイクルを意識する

仕事で使うPCであっても、つい最近までは「使えるまで使う」というのが一般的な考え方だったと思います。「ベンダがサポートを終了するのと同時に利用を控える」という受動的なケースはあっても、「ベンダのサポート期間の残存を考えて利用する」という能動的なケースはまれだったように思います。

ところが今は、ライフサイクルを前提としないと、今度はサポートが切れてしまった状態で、脆弱性が現れた場合に対応できない（利用を放棄するしか方法がない）という時代になってしまいました。

ソフトウェアも時間とともに「劣化」する時代だと言えます。もちろんソフトウェアはデジタルなので、物理的に何かが劣化するわけではありません。しかし、今回のQuickTimeのように、ベンダが脆弱性に対応せずサポートを中止してしまったソフトウェアを使い続けるのは、金属疲労を起こしていつ壊れるかわからない機械と同様な扱いになるはずです。

Microsoftのライフサイクル

Microsoft社では、2002年から同社のプラットフォームのライフサイクルを明確化し、ユーザに告知しています。2016年4月現在の主力商品のライフサイクルは**表1**のとおりです。

2014年にWindows XPの延長サポートが終了となる前後は話題になりました。しかし、セキュリティ的にはWindows XPのライフサイクルだけを議論しても意味がなく、その上で動作しているアプリケーションのライフサイクルも一緒に議論しなければ意味はありません。そして、それがプラットフォー

◆**表1　Windowsのライフサイクル**[注5]

オペレーティングシステム	メインストリームサポート終了	延長サポート終了
Windows XP	2009年4月14日	2014年4月8日
Windows Vista	2012年4月10日	2017年4月11日
Windows 7	2015年1月13日	2020年1月14日
Windows 8	2018年1月9日	2023年1月10日
Windows 10	2020年10月13日	2025年10月14日

注5）Microsoft社　Windowsライフサイクルのファクトシート　http://windows.microsoft.com/ja-jp/windows/lifecycle

ム（ここではWindows XPのこと）ときちんと連動していなければ意味がありません。この連載で繰り返して説明しているバレルセオリーと同じで、一部でも弱い部分があれば、セキュリティはその弱い部分からほころんでしまうのです。

また、Windows 7や8からWindows 10への自動アップグレードは、いつの間にかアップグレードのためのパッケージがダウンロードされていて、アップグレードを勧められることで有名です。しかし、脆弱性をそのままにされてしまう可能性などを考えると、ここまで強制的とも言えるユーザへの働きかけも必要なのではないでしょうか。

Windows 8のメインストリームのサポート終了は実質的に2017年いっぱいなので、だいたい残り19ヵ月です。筆者としてはかなり短いと思っているのですが、現在でもWindows 8へのダウングレード可能なプレインストールPCが売られていることを考えると、ライフサイクルを考えつつアップグレードするというのは難しいものだと思います。

GNU/Linuxのライフサイクル

GNU/Linuxの中でも、このライフサイクルを非常にわかりやすく提供しているのがUbuntuです。Ubuntuには、バージョン番号の後ろに“LTS”と付く5年を目処とした長期サポートバージョン（Long Term Support）と、1年程度の短いサイクルでサポートを終了させるものを組み合わせてリリースしています（**表2**）。次期バージョンのLTSの間隔は2年ですので、あまりソフトウェアのバージョンが古くなることもなく、かつ、サポートも5年間保証しているので、非常にライフサイクルを考えやすくなっています。

ライフサイクルについては、Ubuntu以外のメジャーなGNU/Linuxディストリビューションでも強く意識しています。近年ではlinuxlifecycle.com[注6]というサイトがあり、各ディストリビューションの一覧を確認することができます。

ちなみに、linuxlifecycle.comで確認すると、Red Hat Enterprise Linux、CentOS、SUSE Linux Enterprise Serverといったサーバ系のディストリビューションは10年で、さらにオプションで2年あるいは3年延長することが可能のようです。

減価償却資産の耐用年数

個人でPCを購入している方には関係ないのですが、職場でPCを導入する場合、減価償却資産の耐用年数が関係してきます。PCとして使うものに関しては4年です。PCソフトウェアのライフサイクルが3年程度ですから、微妙にかみ合いません。減価償却資産の耐用年数がまだ来ていないので、ライフサイクルが過ぎた古い機材を使わざる得ないという状況もあるのではないかと筆者は心配しています。今日的な減価償却資産の耐用年数は、ソフトウェアのライフサイクルを勘案して最長でも3年程度、あるいはそれ以下が妥当なのではないかと思います。

まとめ

脆弱性の対応は、問題があって、その都度対応するという事後対応的な性質になりがちです。経営戦略というと大げさですが、ソフトウェアのライフサイクルを勘案しながら、新しいPC環境にアップグレードしていくことで積極的な脆弱性対応も可能かと思います。

みなさんも、これからはソフトウェアのライフサイクルを意識するようにしてはどうでしょうか。SD

◆**表2　Ubuntuのライフサイクル**

コードネーム	バージョン	リリース日	サポート期限
Xenial Xerus	16.04 LTS	2016年4月21日	2021年4月
Wily Werewolf	15.10	2015年10月22日	2016年7月
Vivid Vervet	15.04	2015年4月23日	2016年1月
Trusty Tahr	14.04 LTS	2014年4月17日	2019年4月
Precise Pangolin	12.04 LTS	2012年4月26日	2017年4月

注6）linuxlifecycle.com　Support Life Cycles for Enterprise Linux Distributions　http://linuxlifecycle.com/

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のファイルシステムが階層構造であること、CLIでのファイル操作やプログラミング／スクリプティング時にも基本となるパスの指定方法、基本的なコマンドを使ってファイルやディレクトリの確認方法を紹介します。操作は簡単ですが、大切な情報をさまざまな視点から確認できる重要なコマンドを解説します。

第2回 ディレクトリとファイルの構造・属性

コンピュータ技術者といっても多種多様です。クラウド化が進む今日、マネージドサービスを担えるエンジニアが不足しています。マネージドサービスは、さまざまなシステムやソフトウェアを組み合わせてアーキテクチャ設計をおこない、性能、可用性などを維持・運用するサービスです。エンジニアにはソフトウェアからネットワーク、ハードウェアまでの総合的な知識や、不明の問題にあたったときの問題解決能力など、高い技術やコミュニケーション能力が要求されるやりがいのある仕事です。

みなさんも本連載でコマンドライン、シェルを通じてOSの深淵を理解し、一歩進んだエンジニアを目指してみてはどうでしょうか。

ファイルとファイルシステム

前回は、CLIと小さなツールを組み合わせて使うUnix的な考え方と魅力、manコマンドとその周辺ついて紹介しました[注1]。調べたいことがあったら「Googleの前にman」でしたね。

それでは、ファイルとディレクトリのお話をしましょう。

使っているOSによって、複数のファイルを入れておくしくみを**フォルダ**＝OS Xとか、**ディレクトリ**＝Unixなど別の呼び方をすることがあります。

PCを使っていると「Excelのファイルを編集する」、「このファイルをフォルダに入れて」、「ファイルをバックアップする」など当たり前のように「ファイル」という単語を使います。でも**ファイル**ってなんでしょうか。筆者の社内でも、新人のエンジニアさんに同じ質問をしてみました。

結果は、「a.ファイル＝データ？」、「b.フォルダに入れるもの？」、「c.情報が書かれたもの？」、「d.編集するもの？」など曖昧(あいまい)な返答が出ました。

少し議論をしてから考えると、

e. ただのデータではなくて、アプリケーション・プログラムが使う「データの集まり」(Dataという単語自体複数形ですが)とか「意味のある形式のデータ」などではないか

という意見が出ました。👍

コンピュータを離れて考えると、ファイルは文書や写真などを集めて管理する対象です。ファイルを複数入れておくのがフォルダだったり、棚だったりします。紙の上に書かれたインクのシミに、意味を見出しているのは人間様ですし、それらを分類整理しているのも人です。

コンピュータの内部(CPUやメモリ、HDD)ハードウェアレベルでは「これはExcelのファ

注1) 紙幅のせいで泣く泣く掲載を見送った内容を、補足情報としてWebに公開しました。ぜひあわせて読んでください。http://gihyo.jp/magazine/SD/archive/2016/201605/support

イル」、「鈴木さんのファイル」などといった区別があるわけではありません。あちこちに散らばった0と1を一塊の情報として名前を付け、意味を与え、さまざまな機能を提供しているのはOSです。この整理や管理のしくみをファイルシステムと呼び、Unixにかぎらず一般的なOS(もしくは周辺のサブシステム)が提供する重要な役割の1つです。

e.の意見のように、私達が操作の対象とするファイルとは、OSによって意味付けされ管理されているデータの集まりです。続いてUnixはどのようにファイルを管理・整理しているのかを見ていきましょう。

ファイルシステムの基本 ～階層構造のファイルシステム

図1のように、Unixのファイルシステムは階層構造になっています。ファイルシステムは、一般的なファイルに加えて、ディレクトリと呼ぶファイルやディレクトリを入れられる器、スペシャルファイルなどを階層的な木構造で管理します。**図1**の一番上の/をルートディレクトリと呼び、以下にファイルやディレクトリを持つことができます。各ファイルやディレクトリの位置は、パスネーム(各ディレクトリを/で区切って並べたもの)で特定できます。

絶対パスと相対パス

パスの指定方法は、ルートディレクトリから指定する/bin/ls、/usr/bin/rubyなどの絶対パス方式と、自分のワーキングディレクトリ(.で表します)を起点に/で始まらない記法で指定する相対パス方式があります。

ディレクトリ名には、**表1**のような特別な意味の名前がありますので覚えておきましょう。

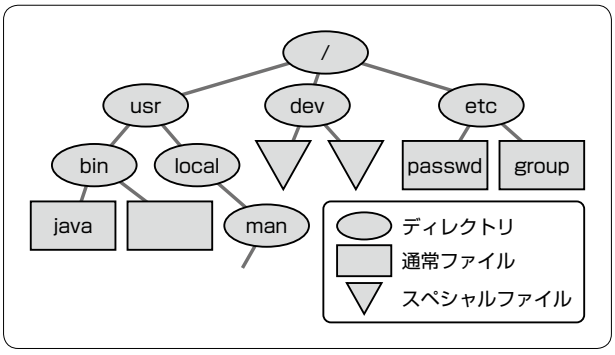
図2中のディレクトリ/usr/localをワーキングディレクトリとすると、絶対パス/usr/bin/javaへの相対パスは、../bin/javaとなります。

相対パス指定で自分のワーキングディレクトリを明示的に示す場合は、./をパス名の最初に付けますが、**図2**中の/usr/local/etcディレクトリを、/usr/localから参照する場合は、単にetcを記述しても、./etcと記述してもかまいません。同じしくみで、../bin/javaは、./../bin/javaと同じことです。

STEP UP! 大文字と小文字の区別など、ファイルシステムによって異なること

MS-DOSをルーツとするWindows系のOSでは、ファイル名の大文字と小文字の区別はありません。MS-DOSの時代にはすべてのファイル名が、小文字で命名したものも大文字に変換されてしまいました。Windows 10などでは命名したとおりのファイル名が保存されますが、内部的には大文字と小文字を同一視しています。

▼図1　ディレクトリ階層構造



▼表1　特別な意味を持つディレクトリ

表記	名前	意味
/※	ルートディレクトリ	ファイルシステム(STEP UP! 複数のファイルシステムがマウントしてあっても、現在利用しているOSにおいて)の最上層のディレクトリ
.	カレントワーキングディレクトリ	現在のディレクトリ
..	ペアレントディレクトリ	1つ上のディレクトリ。親ディレクトリ
~	ホームディレクトリ	環境変数$HOMEで設定されているディレクトリ(ユーザによってそれぞれ異なる)

※/の前に何もつかないことに注意

たとえば、“Work”という名前のディレクトリがすでに存在している場合、“work”という名前のディレクトリを作ろうとしても、すでに同一の項目が存在しているのでディレクトリの作成は失敗します。

こうした、文字の扱いは、ファイルシステムによってそれぞれ異なります。OS X においても、OSがインストールされるファイルシステムは、大文字小文字を区別しないフォーマットになっています。その一方で外付けのドライブなどは、大文字小文字を区別するように指定してフォーマットすることもできます。

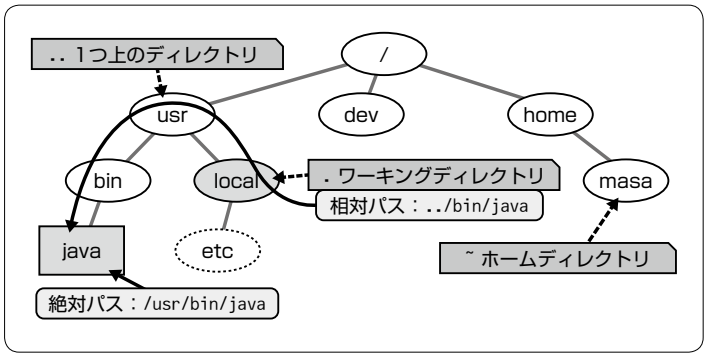
英語では、大文字小文字を区別する場合を“case-sensitive”と言い、区別しない場合を“case-insensitive”と言います。エラーメッセージや、新たにファイルシステムをフォーマットする際の選択肢に現れることがあるので覚えておいたほうが良い単語です。

Linuxのext4ファイルシステムや、Unixの伝統的なファイルシステムであるUFSなどでは大文字小文字が区別されます。同じOS上で使うファイルシステムでも、Unix系のLinux、OS XはもちろんWindowsなどであっても複数の種類のファイルシステムを扱えます。ファイルシステムの違いは、コマンドの動作などにも違いを生じさせることがありますので、操作の対象となるファイルシステムがどういった形式なのか知っておくことは重要です。

ファイルの種類

私達がUnix上で扱う標準的なファイルには、ファイル、ディレクトリ、シンボリックリンク[注2]などがあります。通常のファイルであっても、実行可能なバイナリファイルやシェルスクリプトなど、ファイルの用途や目的はさまざまです。

▼図2　相対パスと絶対パス



file— FILE —ファイルの種類を知る

そのファイルが何なのか確認したくて、ファイルをエディタで開いたり実行してしまうと、思わぬ結果になってしまうことがあります。こんなときには、便利なコマンドfileが用意されています。たとえば、/bin/lsと/etc/passwdがどんなファイルなのか確認してみましょう。

```
OS X 10.11.4の例
$ file /bin/ls /etc/passwd↵
/bin/ls: Mach-O 64-bit executable x86_64
/etc/passwd: ASCII English text
```

/bin/lsは、64bitのバイナリ実行可能なファイルであることと、/etc/passwdがアスキー文字セットを使った英語のテキスト形式であることがわかりました。

このようにfileコマンドは、ファイルを実行したり開いたりしなくても、ある程度のあたりをつけてくれます。ただ、万全にすべての形式を区別しているわけではないので、まずfileコマンドを使って識別してから次の段階で、たとえばテキスト形式と判定されたらエディタで開いてみるなど、きちんと確認をするようにしましょう。

lsコマンドを使いこなす

lsはさまざまな視点からファイルの情報を

注2）別の機会に説明します。

表示してくれます。そのため、とても多くのオプションのあるコマンドでもあります。ここでは、これだけは知っておきたいという基本的な操作を紹介します。

ls—LiSt—ファイルやディレクトリをリストアップする

/etcディレクトリにあるファイルやディレクトリをlsで確認してみましょう。**図3**はOS Xでの例です。アルファベット順に段組みされて表示されます[注3]。

次は、-Fオプションを付けてみます。このオプションは、ディレクトリには「/」、通常ファイルは無印、シンボリックリンクには「@」などのタイプ別に「印」をファイル名の後ろに付けてくれるオプションです。

```
OS X 10.11.4の例
$ ls -F /etc↵
/etc@
```

「あれ、/etcの中にはたくさんファイルがあったはずなのに……/etc@だけ？」

-lのロング表示オプションを付けて、もう少し詳しく見てみましょう。

すると**図4**のように、ls出力の最後のフィールドが**/etc@ -> private/etc**となっていますね。このことから/etcは確かに、/private/etcへのシンボリックリンクであることが確認できました[注4]。

さらにシンボリックリンク先を**図5**のように確認してみましょう。/private/etcに**図3**で表示されていたものがきちんと入っていることが確認できました。aliasesがシンボリックリンク(@)、apache2がディレクトリ(/)、他のファイルはおそらく通常ファイルであることなどもさらにわかりました。fileコマンドを使わなくても、これらの区別はできます。

lsの引数で指定した対象がディレクトリのときは、その中身が表示されます(**図6**)。ディレクトリを中身ではなく、そのまま表示させたいときには-dオプションを指定します。

```
$ ls -d /bin↵
/bin
```

注3) ここでは、lsにターゲットを/etcとして指定しています。lsは、ターゲットを指定しなければカレントワーキングディレクトリの内容を表示します。

注4) 新しいOS Xでは、セキュリティ上の理由から/etcは実体ではなくシンボリックリンクです。Linuxなど他の環境では、/etcはシンボリックリンクではありません。

▼図3　ls(OS X 10.11.4の例)

```
$ ls /etc↵
afpovertcp.cfg          hosts~orig              pf.conf
aliases                 irbrc                   pf.os
aliases.db              kern_loader.conf        php-fpm.conf.default
apache2                 krb5.keytab             php.ini.default
...後略...
```

▼図4　ls -lF(OS X 10.11.4の例)

```
$ ls -lF /etc↵
lrwxr-xr-x@ 1 root  wheel  11 10 11 07:41 /etc@ -> private/etc
```

▼図5　/private/etcの中身を確認(OS X 10.11.4の例)。図3の結果と同じであることが確認できます

```
$ ls -F /private/etc↵
afpovertcp.cfg          hosts~orig              pf.conf
aliases@                irbrc                   pf.os
aliases.db              kern_loader.conf        php-fpm.conf.default
apache2/                krb5.keytab             php.ini.default
...後略...
```

▼図6　lsでディレクトリを指定した場合（OS X 10.11.4の例）

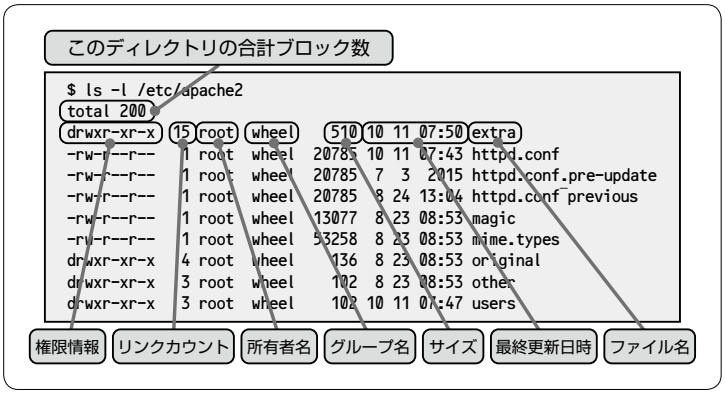
```
$ ls /bin↵
[           cp          df          expr        launchctl   mkdir       pwd         sh          tcsh        zsh
bash        csh         domainname  hostname    link        mv          rcp         sleep       test
...後略...
```

-lについてもう少し詳しく

-lは最も使うオプションですので、表示の内容を**図7**で解説しておきます。

ファイルやディレクトリには「所有者グループ、その他のユーザ」という所有権限があります（詳細は別の回で）。それぞれに“r：読み取り”、“w：書き込み”、“x：実行／ディレクトリの場合はディレクトリ内に入る”権限があります。これらを「rwx bits」と呼びます。-lオプションを付けたlsの出力で、r、w、xが表示されているところはその権限があることを示し、-と表示されていれば、その権限がないことを示しています。ファイルやディレクトリにはrwx bitsのほかにも属性が付いていますが、それらについても別の回で説明します。

▼図7　ls -lの実行例と意味（OS X 10.11.4の例）



lsとカラー

UbuntuやCentOSでは、lsを実行するとタイプ別に色が付いています。もちろん、端末が色を表示できる設定・状態でなければ色は付きません。もし色が付く環境を使っていれば、いちいち-Fオプションを付けなくても、一目で区別がついて便利ですね。これは、lsのオプションで色が付くように指定してあるんです。Linux系のOSを使っているなら、確認してみましょう。

```
$ alias ls↵
ls --color=auto
```

aliasコマンドはシェルの機能[注5]で、別名を定義できる便利なしくみです。この例では、lsというalias（別名）が定義されているかを表示させました。lsと入力されたら、ls --color=autoと置き換えるという指示が定義されています。つまり、仮にこのaliasが定義されていないとしても、Linux系のOSでlsに色を付けるには、--color=autoというオプションを指定してあげればいいのです。

OS Xは、BSD系Unixなのでlsのオプションが Linux系とは一部異なります。OS Xでlsに色を付けるには、-Gオプションを指定します。OS Xの方は試してみましょう。

```
$ ls -G
```

OS Xでもaliasを指定して、標準で-Gオプションが付いている状態にすることはできます。

```
$ alias ls='ls -G'
```

「ターミナル」アプリケーションの環境設定で、色を含む挙動をいろいろと選べますので、そち

注5）組み込み機能＝builtin：OS X、CentOSでman aliasするとbuiltin(1)が表示されます。Ubuntuなどではmanページはありません。シェルのプロンプトでhelp aliasもしくは、man bashで確認しましょう。

らも確認してみてください。

隠されたファイル!?—デフォルトでは表示されないファイルとディレクトリ

じつは、lsはデフォルトでは.で始まるファイル名を表示しないのです。Unixでは.で始まるファイルやディレクトリは、慣用的に管理目的など一般には使わないものの名前として使っています[注6]。これらを表示させるには、-aオプションを使いましょう。

図8の上下で比較してみるとわかるとおり、たくさんの.で始まる名前のファイルが表示されました。カレントディレクトリである.や、親ディレクトリ..も表示されていますね。

結果をファイルに

次は、ファイルに保存してみましょう[注7]。

```
$ ls > ls_log.txt
```

次に、catコマンドで内容を確認します。

```
$ cat ls_log.txt↵
Applications
Applications (Parallels)
Creative Cloud Files
Creative Cloud Files (unknown)
Desktop
...後略...
```

注6）.で始まるファイル名は、あくまで名前であり、隠しファイル属性ではありません。

注7）>は出力をリダイレクトするための表記です。これによって、これまで画面に出力されていた内容がファイルに出力されるようになります。

このようにターミナル(tty)以外に出力させると、マルチカラムだったものが1行に1ファイルになります。

ファイルにリダイレクトしたり別のプログラムにパイプで接続してもマルチカラムで出力させたいときは-Cオプションを付けます。

```
$ ls -C > ls_log.txt
```

逆にターミナル(tty)に対してでも、マルチカラムではなくて1行に1ファイルを表示したいときは、-1オプションを用います。

これらレイアウトを指定するオプションは、画面での見やすさや、出力結果を別ツールで処理するなどで、適宜使い分けるとよいでしょう。

lsにはほかにもたくさんの機能があります。そのほかのさまざまなオプションについては、man lsで調べてみてください。

今回のまとめと次回について

今月の確認リスト

【manで調べるもの(括弧内はセクション番号)】
ls(1)、file(1)、cat(1)

今回はファイルシステムが木構造になっていること、ファイルやディレクトリの確認方法を紹介しました。次回は、プロセスの操作について解説しようと思います。SD

▼図8　OS X 10.11.4でホームディレクトリを表示した例(上が-aオプションなし。下が-aオプション付き)

```
-aオプションなし
$ ls↵
Applications                Downloads       Movies          Work
Applications (Parallels)    Dropbox         Music           backup_log
...後略...

-aオプション付き
$ ls -a↵
.                           .config         .ssh            Games
..                          .cups           .subversion     Git
.CFUserTextEncoding         .dropbox        .viminfo        Googleドライブ
.DS_Store                   .dvdcss         .vimrc          Library
.Rapp.history               .fontconfig     .wine           Movies
.Trash                      .games.txt.un~  .xbmc           Music
...後略...
```

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第31回 ❖使ってみようmtree(8)コマンド

◉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。

ディレクトリやファイルの配置を表すmtree

UNIXでは、ファイルやディレクトリの配置、それぞれのパーミッション情報などを表す方法としてmtreeという表記方法を使っています。FreeBSDでは/etc/mtree/にmtree(5)形式のファイルがあります。mtree(5)はかなり古くから使われているフォーマットなのですが、この形式をよく知らないユーザが少なくないように思います。mtree(5)は、ファイルやディレクトリの構造とそのパーミッションなどをまとめて保持しておけますので、配布物を適切なパーミッションでインストールしたり、構築したシステムを適正な状態に保ったり、または変更などをチェックしたりといった目的で利用します。今回はこのmtreeについて説明しようと思います。

リスト1の/etc/mtree/BSD.sendmail.distファイルはsendmail関係のディレクトリ構造とパーミッション、持ち主などの情報を表したファイルです。実際のディレクトリは**図1**のようなパーミッションになっています。mtreeの記載どおりになっていることを確認できます。

故意にパーミッションを変更して、それをmtreeファイルを使って元に戻すということをやっているのが**図2**の操作です。mtree(5)フォーマットの総合的な操作はmtree(8)コマンドで実施します。4行目に実行したコマンドで、/etc/mtree/BSD.sendmail.distファイルに記載されているものと違う持ち主になっていることを発見し、これを修正していることを確認できます。

mtree(5)で適切な情報を表現し、mtree(8)コマンドでその状態をチェックしたりキープしたりできます。またmtree(8)コマンドは、既存のディレクトリやファイル構造から自動的にmtree(5)フォーマットのデータを生成する機能も提供しています。このコマンドを活用することで、構築するシステムを想定した、適切な状態を保てるようになります。

▼**リスト1** /etc/mtree/BSD.sendmail.dist（FreeBSD 10.2-RELEASE）

```
...（略）...
/set type=dir uname=root gname=wheel mode=0755
.                nochange
    var              nochange
        spool            nochange
            clientmqueue     uname=smmsp ↵
gname=smmsp mode=0770
            ..
        ..
    ..
..
```

mtree(5)フォーマット

mtree(5)フォーマットのルールを簡単に説明しておきます。mtree(5)はディレクトリやファイルの階層構造、それらの属性を表現することを目的とした、テキストベースのフォーマットです。もともとは4.3BSD-Renoのタイミングで導入されたフォーマットで、それ以降、随時機能拡張が取り込まれてきました。FreeBSD

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。

2.1でMD5ダイジェスト機能、FreeBSD 4.0でファイルフラグとSHA-1/RIPEMD160ダイジェスト機能、FreeBSD 6.0でSHA-256ダイジェスト機能が取り込まれています。

mtree(5)のフォーマットのおもな規則は次のとおりです。

- 行頭のホワイトスペース(空白)は無視される
- 空行は無視される
- #から始まる行は無視される
- 1行ごとにファイルやディレクトリの情報を示している
- /setから始まる行は特殊コマンド。キーと値の指定が空白区切りで指定される。以降の行にこの行の設定がデフォルト値として適用される
- /unsetから始まる行は特殊コマンド。1つ前の/setで設定されたデフォルト値を削除する。削除するキーが空白区切りで指定される
- ..から始まる行はカレントディレクトリを親ディレクトリへ変更。この行に指定されたオプションは無視される
- /または..以外から始まる行はカレントディレクトリにあるファイルまたはディレクトを表す。対象がディレクトリだった場合にはカレントディレクトリをそのディレクトリへ変更する
- 印刷可能なASCII文字95個分以外の文字列はバックスラッシュから始まる3桁の8進数表記で表現される

言葉でまとめるとちょっとわかりにくいですが、基本的に「**空行**」「**コメント行**」「**デフォルト設定の行**」「**ディレクトリの行**」「**ファイルの行**」「**親ディレクトリに移動する行**」の6種類の行で構成されたフォーマットだと考えるとわかりやすいかもしれません。

たとえば、次のような構成のディレクトリとファイルを考えます。

```
.
├── Makefile
├── commands
│   ├── 001
│   ├── 002
│   └── 003
├── sources
│   ├── 001
│   ├── BSD.root.dist
│   └── BSD.sendmail.dist
└── typescript.xml

2ディレクトリ、8ファイル
```

これをmtree(5)フォーマットで表現すると、たとえば**リスト2**のようになります。ファイル名やディレクトリ名の書いてある行に、キーと値が記載

▼図1　/etc/mtree/BSD.sendmail.distで指定されているディレクトリのパーミッション

```
% ls -dl / ↵
drwxr-xr-x  24 root  wheel  1024 Mar  9 08:40 /
% ls -dl /var ↵
drwxr-xr-x  25 root  wheel  512 Mar 11 17:01 /var
% ls -dl /var/spool ↵
drwxr-xr-x  8 root  wheel  512 Nov 12  2014 /var/spool
% ls -dl /var/spool/clientmqueue ↵
drwxrwx---  2 smmsp  smmsp  512 Mar 12 05:20 /var/spool/clientmqueue
```

▼図2　/etc/mtree/BSD.sendmail.distの内容にしたがいパーミッションなどを元に戻す

```
% chown daichi:daichi /var/spool/clientmqueue ↵
% ls -ld /var/spool/clientmqueue ↵
drwxrwx---  2 daichi  daichi  512 Mar 11 03:13 /var/spool/clientmqueue
% mtree -deU -f /etc/mtree/BSD.sendmail.dist -p / ↵
var/spool/clientmqueue:
        user (25, 501, modified)
        gid (25, 501, modified)
% ls -ld /var/spool/clientmqueue ↵
drwxrwx---  2 smmsp  smmsp  512 Mar 11 03:13 /var/spool/clientmqueue
```

されている行があるかと思いますが、ここに指定できるのが**表1**のようなキーワードです。どのキーワードが使用できるかは実装しているコマンドによって異なりますが、代表的なキーワードはだいたい実装されています。

mtree(8)コマンドの使い方

次の構造のファイルとディレクトリがあるとします。

```
% tree freebsd ↵
freebsd
├── Makefile
├── commands
│   ├── 001
│   ├── 002
│   └── 003
├── sources
│   ├── 001
│   ├── BSD.root.dist
│   └── BSD.sendmail.dist
└── typescript.xml

2 directories, 8 files
```

図3のようにmtree(8)コマンドを実行すると、キーとしてファイルの種類、ユーザID、グループID、パーミッションの情報が記載されたmtree(5)データを作成できます。仮に、この出力をmanifest.mtreeというファイルに保存したとしましょう。保存したmanifest.mtreeというファイルを使うことで、対象となっているファイルとディレクトリがmtree(5)データに指定されたものと同じであるかをチェックできます。次のようにmtree(8)コマンドを実行すると、mtree(5)データと異なる設定になっているファイルやディレクトリが報告されます。

```
% mtree -f manifest.mtree -p freebsd ↵
commands/001:
        permissions (0644, 0666)
commands/002:
        permissions (0644, 0666)
commands/003:
        permissions (0644, 0666)
```

この出力では、パーミッションが0644となっているべきファイルが0666になっている、と報告があります。次のようにmtree(8)に-Uオプションを指定すると、検出した内容に従ってファイルやディレクトリの設定を更新してくれます。

```
% mtree -U -f manifest.mtree -p freebsd ↵
commands/001:
        permissions (0644, 0666, modified)
commands/002:
        permissions (0644, 0666, modified)
commands/003:
        permissions (0644, 0666, modified)
% mtree -U -f manifest.mtree -p freebsd ↵
%          ←すでに修正されたので何も出力されない
```

▼**リスト2　mtree(5)によるディレクトリとファイルの構造**

```
# .                                    ←コメント
/set type=file uid=501 gid=20 mode=0644 ←ファイルのデフォルト設定を指定
.               type=dir mode=0755     ←カレントディレクトリとそのパーミッションの指定
    Makefile                           ←ファイル Makefile がある
    typescript.xml                     ←ファイル typescript.xml がある
                                       ←空行
# ./commands                           ←コメント
commands        type=dir mode=0755     ←ディレクトリ commands がある。その設定と commands へ移動
    001                                ←ファイル 001 がある
    002                                ←ファイル 002 がある
    003                                ←ファイル 003 がある
# ./commands                           ←コメント
..                                     ←親ディレクトリに移動
                                       ←空行
# ./sources                            ←コメント
sources         type=dir mode=0755     ←ディレクトリ sources がある。その設定と sources へ移動
    001                                ←ファイル 001 がある
    BSD.root.dist                      ←ファイル BSD.root.dist がある
    BSD.sendmail.dist mode=0444        ←ファイル BSD.sendmail.dist がある。そのパーミッションの設定
# ./sources                            ←コメント
..                                     ←親ディレクトリに移動
```

mtree(5)データに記載されていないファイルを検出して削除する、といったこともできます。次のように記載されていないファイルが存在していると、mtree(8)はそれを検出して報告してくれます。

```
% mtree -f manifest.mtree -p freebsd
extra: commands/004
```

mtree(8)に-Uではなく-rを指定すると、この存在していないファイルを削除してくれます。

```
% mtree -r -f manifest.mtree -p freebsd ↵
extra: commands/004, removed
% mtree -r -f manifest.mtree -p freebsd ↵
%          ←すでに削除されたので何も出力されない
```

このようにmtree(5)データとmtree(8)コマンドを使うことで、ファイルやディレクトリを本来あるべき状態に保持する、といった操作を実施できます。パーミッションや持ち主設定は、管理者が操作している間に変わったまま放置されていることがあります。ですので、パーミッションが変更されると問題がある場合などは、cron(8)に登録して定期的にチェックして不整合が見つかった場合に管理者にメールで報告する、といったことを実施できます。

◆ ◆ ◆

FreeBSDではmtree(8)コマンド以外に、bsdtar(1)、install(1)、makefs(8)などのコマンドがmtree(5)フォーマットのデータに対応しています。しかし、実装はそれぞれのコマンドやライブラリが持っていて、サポートしている機能やキーワードにばらつきがあるというのが実態です。

FreeBSDプロジェクトでは現在、mtree(5)データを操作する統一されたライブラリの開発が進められています。プロジェクトの詳細は「mtree parsing and manipulation library」[注1]で確認できます。実装はGitHub[注2]にまとまっています。

これら実装がFreeBSD 11.0-RELEASEまでにマージされるかどうかは執筆段階では未定ですが、順調に作業が進めば11.0Rに、遅くとも11.1Rにはマージされるのではないかとみられます。SD

注1 https://wiki.freebsd.org/SummerOfCode2015/mtreeParsingLibrary
注2 https://github.com/mratajsky/libmtree

▼表1 mtree(5)で使用するキーワード

キーワード	内容
uid	ユーザID
uname	ユーザ名
gid	グループID
gname	グループ名
mode	ファイルパーミッション(数値または文字列)
size	ファイルのサイズ(バイト)
time	最終更新時刻。「秒.ナノ秒」で表現。ナノ秒は9桁の数字
type	ファイルの型を指定。block、char、dir、fifo、file、link、socketを指定可能
link	シンボリックリンクの数
nlink	ハードリンクの数
nochange	ファイルかディレクトリが存在するかどうかのみをチェックしてほかの属性は無視する
cksum	chsum(1)コマンドが出力するチェックサム値
md5	MD5メッセージダイジェスト
md5digest	md5の別名
sha1	SHA-1メッセージダイジェスト
sha1digest	sha1の別名
sha256	SHA-256メッセージダイジェスト
sha256digest	sha256の別名
rmd160	RIPEMD160メッセージダイジェスト
rmd160digest	rmd160の別名
ripemd160digest	rmd160の別名
flags	chflags(1)で指定できるフラグ名
ignore	このファイル以降の階層構造は無視する指定

▼図3 mtree(8)でmtreeフォーマットテキストを生成する例

```
% mtree -c -p freebsd -k type,uid,gid,mode ↵
... (略) ...
# .
/set type=file uid=501 gid=20 mode=0644
.                type=dir mode=0755
    Makefile
    typescript.xml

# ./commands
commands         type=dir mode=0755
    001
    002
    003
# ./commands
..

# ./sources
sources          type=dir mode=0755
    001
    BSD.root.dist
    BSD.sendmail.dist \
                 mode=0444
# ./sources
..
```

36

Debian Developer　やまねひでき　henrich@debian.org

ライセンス問題はディストリビューションの悩みどころ!?

Debian Hot Topics

MicrosoftのSDNはDebian上で動作!

Microsoft社のSDN(Software Defined Network)である「SONiC(Software for Open Networking in the Cloud)」がGitHub上でオープンソースとして公開されました……とだけ書くと、「なんでDebianの記事に？」と疑問を持たれる方も多いかと思います。

ですが、なんとこのSDNは、Debian上で動かしている("Today SONiC runs on Debian"[注1])ということなのです。ユーザが目に触れることのない、クラウド内部のごく一部とはいえ、Microsoftが自社のビジネスの核となるMicrosoft Azure内でDebianを利用しているという事実は衝撃的だったようで、一部のメディアは「Microsoft has crafted a switch OS on Debian Linux. Repeat, a switch OS on Debian Linux(MicrosoftはDebian上で動作するネットワークスイッチOSを作成した！　繰り返す、Debianで動作するネットワークスイッチOSだ！)」という驚きを表す見出しを付けていました[注2]。

nvidia「バイナリ」モジュールの削除

これまでDebianのnon-freeリポジトリに入っていたnvidia-graphics-modulesパッケージ(NVIDIAのグラフィックチップ用のドライバ)は、Linuxカーネルのライセンスとの非互換から、Bug#815060にて削除が提案されていました。そして議論の結果、リポジトリから削除されました[注3]。

では、NVIDIAのグラフィックチップのハードウェアを持っていて、オープンソースのNouveauドライバ[注4]ではなく、NVIDIA純正のプロプライエタリドライバが使いたいユーザはどうするの？というと、同じくnon-freeリポジトリに含まれているnvidia-kernel-dkms、あるいはnvidia-kernel-sourceパッケージを利用することで、今までと同様に利用可能となっています(今回の削除は、あくまでコンパイル済みのバイナリパッケージの削除、ということです)。

dkms(Dynamic Kernel Module Support)とは、ユーザがパッケージ導入時にソースを手元でコンパイルするという形で利用ができるようにしているものです。NVIDIAのドライバのように、ライセンスがプロプライエタリなカーネルモジュールについては、「バイナリ形式での配布」はGPL v2(GNU General Public License 2.0)との非互換性により問題があるものの、このdkmsのようにユーザ自身がソースからバイナリをビルドして使う範囲においては問題とな

注1) SONiCのGitHubにて公開されているFAQより。
URL https://github.com/Azure/SONiC/blob/gh-pages/FAQ.md

注2) URL http://www.theregister.co.uk/2016/03/09/microsoft_sonic_debian/

注3) URL https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=815060

注4) 「ぬーぼー」と読む。X.Org Foundationとfreedesktop.orgによって開発されているフリーなNVIDIAのグラフィックチップ用ドライバ。

りません。

当該バグの議論の中で、ZFS[注5]も引き合いに出されていましたが、Debianプロジェクトリーダーの Neil McGovern 氏のブログ[注6]によれば、こちらも dkms を使ったパッケージを予定しているようです。

GPLとCDDLを巡るお話

Debianはディストリビューションなので、さまざまなソフトウェアライセンスの話を避けて通ることはできません。さて、今回取り上げたNVIDIAドライバやZFSのライセンスは、Linuxカーネルに使われているフリーソフトウェアの代表的ライセンスであるGPLとの組み合わせでどのような点で問題になるのでしょうか？

NVIDIAドライバのほうは、ガチガチのプロプライエタリライセンスです。これは、Debianでは「non-free」に分類されるもので、改変再配布ができないなど、GPLと複数の点で明らかな矛盾があります。

一方、ZFSは、旧Sun Microsystems社（以下、Sun）・現Oracle社が所有するSolaris用のファイルシステムとして開発されました。オープンソース寄りな姿勢になっていたSunがSolarisを始めとする各種ソフトウェアを独自のオープンソースライセンスで公開した時期があり、この際に作成／適用されたライセンスがCDDL[注7]で、ZFSもCDDLの下で提供されています。

GPLとCDDLの違い

CDDLはオープンソースライセンスであることは間違いないのですが、残念ながらCDDLとGPLの組み合わせには問題があります。

どのような部分かというと、CDDLの3.1項では、「Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License」とあります。この文章のポイントは「only」です。「CDDLライセンスで配布するバイナリは、ソースコードでも入手できなければいけないし、ソースコードはCDDLライセンスの条件のもと『のみ』で配布する必要がある」という内容になります。

このCDDLの文言と、GPL v2の2.b項「You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License」（プログラムまたはその一部を含む著作物、あるいはプログラムかその一部から派生した著作物を頒布あるいは発表する場合には、その全体をこのライセンス（= GPL v2）の条件に従って第三者へ無償で利用許諾しなければならない）が矛盾することになります。

ほかにも、サブライセンス[注8]はGPL v2では明確に禁止されているのにもかかわらず、CDDLでは可能となっている点などもあります。ZFSのもともとのライセンスであるCDDLに従いつつ、GPL v2にも従うというのは結構な無理筋です。

この矛盾によって、GPLとCDDLの2つのライセンスからなるソースコードを組み合わせたバイナリは再配布ができません[注9]。

この見解は一般的なもので、2010年にOpen Solarisの公開が停止して以来、ZFS開発をフォーク（分岐）して進めているOpenZFSプロ

注5） Solarisで使われているファイルシステム。一時期ソースコードがオープンソースライセンスの下で公開されていた際に、LinuxやFreeBSDなどほかのOSに移植された。現在は最新版のソースコードはライセンスが再びプロプライエタリなものに変わり、非公開となっている。安定して使いたい場合はSolarisを使うのが無難で、Linux版で使うとどの程度うれしいのかは未知数……。

注6） URL http://blog.halon.org.uk/2016/01/on-zfs-in-debian/

注7） Common Development and Distribution License の 略。Mozilla Public License（MPL）1.1をベースに策定された。

注8） ライセンスを受けた者が、ライセンスされた特許や商標をさらに第三者にライセンスすること。

注9） 決定的なところは法廷に出て判例が確定しないと言えませんが、少なくともそう解釈するのが妥当だと筆者も考えます。

ジェクトのFAQには、このライセンス非互換性の問題が明記されています[注10]。また、GPLを策定しているGNUプロジェクトを支援するフリーソフトウェア財団のFAQでは、CDDLをGPLと互換性がないライセンスであると紹介しています[注11]。

なお、CDDLでライセンスされたものであっても、Java EEアプリケーション・サーバのGlassFishのようにGPLの「Classpath例外条項」[注12]を明示的に適用して、GPLで配布されているJavaライブラリにリンクされるコードにGPLが適用されないような形にして、CDDLと矛盾が起きないように回避していたものもありました。

UbuntuのZFS問題

ところが先日、Ubuntu開発元のCanonical社の開発者Dustin Kirkland氏が突如、「Ubuntu 16.04のコンテナではZFSを使う」旨をブログでぶちあげて話題となりました[注13]。「Ubuntuシステムに、ZFSのカーネルモジュールが自動的にビルド／インストールされる。DKMSでビルドされるモジュールは不要だよ！(You'll find zfs.ko automatically built and installed on your Ubuntu systems. No more DKMS-built modules!)」と、dkmsを使わないことを示唆しており、確かにUbuntuのLinuxカーネルのGitリポジトリにはZFSが含まれています[注14]。

これに対して、SFLC[注15]やSoftware Freedom Conservancy[注16]など、GPLに関するアドバイス／是正／訴訟を専門に扱う複数の団体が、ZFSモジュールをバイナリ配布するのは不適切である旨を表明しています。

彼いわく、「Canonicalでのリーガルレビューの結果、問題ないという結論になった。不適切であるという意見はあくまでも意見だ」[注17]とのことなのですが、詳細なロジックが出されていないので、「Ubuntuユーザを無用のリスクに晒すのではないか」などと、疑念は晴れていないようです。

それにもかかわらず、その後4月22日のUbuntu 16.04のリリース時点ではカーネルモジュールにZFSが含まれているようです。手元で確認したところ、zfs.koがしっかりとLinuxカーネルパッケージ内に存在しています。

今後、Linuxカーネルの著作権者やZFSの権利者であるOracleがどのようなアクションを取るのか(あるいは取らないのか)、争いになったとしてどのようなロジックでこれを退けようとするのかに注目が集まります(カーネルモジュールは独立した著作物である、というような論理あたりになるのでしょうか？)。

◆ ◆ ◆

Debianでは、先に紹介したようにzfs-linuxパッケージについては、「ソースコードをユーザの手元でビルドする」形として配布を予定しており、矛盾したライセンスを持つバイナリの配布を明示的に避けています。早く利用できるようになると良いですね。

なお、DebianでZFSを今すぐ使ってみたいという場合は、GPLのLinuxカーネルではなく、BSDライセンスのFreeBSDカーネルを使った「Debian GNU/kFreeBSD」があり、そちらを使うという変化球もありますので、ご検討ください。SD

注10) URL http://open-zfs.org/wiki/Talk:FAQ
注11) URL http://www.gnu.org/licenses/license-list.html#GPLIncompatibleLicenses
注12) URL http://www.gnu.org/software/classpath/license.html
注13) URL http://blog.dustinkirkland.com/2016/02/zfs-is-fs-for-containers-in-ubuntu-1604.html
注14) URL http://kernel.ubuntu.com/git/ubuntu/ubuntu-xenial.git/tree/zfs
注15) The Software Freedom Law Center。弁護士などの法律の専門家らがボランティアでFLOSS活動に対する支援を行う組織。GPL v3の策定に携わったEben Moglen氏らが参加している。今回の件は、URL https://www.softwarefreedom.org/resources/2016/linux-kernel-cddl.htmlを参照。
注16) FLOSS向けの法的援助組織。URL https://sfconservancy.org/blog/2016/feb/25/zfs-and-linux/
注17) URL http://blog.dustinkirkland.com/2016/02/zfs-licensing-and-linux.html

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第28回 shはやっぱり

シェルといえば、「武田久美子の貝殻ビキニ」を思い出す傍若無人な編集長に脅されて、シェルにまつわるマンガを書きました(そんな僕に愛のメッセージを)。

to be Continued

普段使っているシェルはbashです。たまにzshを試しますが、やっぱりbashに戻って来てしまいます。他のシェルも魅力的ですが、変えてみようと思うたびに設定方法調べたり、設定ファイルを作る・試すのが激しく面倒になったり、ある機能が使いたくて移行してみたら数ヵ月後にbashでも同じ機能が使えるようになったとか……。こういうことが続くとLinuxを触ってる間はbashにココロを捧げたほうがむしろ楽ではないかと思うわけです。UNIX互換OSを使ううえで一番触れるのがコマンドラインです。zshやtcshをカジュアルに選ぶのもイイですし、「一番肌に触れるもの」と肌着を選ぶように慎重にシェルを探すのもLinux環境を使う楽しみの1つですね。

Ubuntu Touchの日本語入力

Ubuntu Japanese Team／(株)創夢
柴田 充也(しばた みつや)　mail：mty.shibata@gmail.com

Ubuntu Touchの最新リリースであるOTA-10から、待望の日本語入力をサポートするようになりました。今回はこの日本語入力機能が取り込まれるまでの流れと、その簡単なしくみについて紹介します。

Ubuntu Touchの入力メソッド

「Ubuntu Touch」はスマートフォン／タブレット向けに開発しているUbuntuです。本連載ではこれまでに何度か紹介していますが、簡単にまとめておくと次のような機能を持っています。

- PC向けUbuntuと同等の機能を持つように設定したLinuxカーネル
- コアシステム部分のイメージベースのアップデート
- 次世代ディスプレイサーバであるMirとQt5/QMLベースのUnity8を使ったインターフェース
- アプリごとに隔離環境を構築するClickパッケージングシステム
- Click用のアプリストア
- デスクトップモードとのシームレスな切り替え

基本的には用語が異なるだけで、AndroidやiOSのような今風のスマートフォンと同じしくみを作ろうとしています。ただしコアシステムは従来のUbuntuと同じで、bashもvimもpython3もインストールされています。もちろん光学ドライブをつなげればejectだって実行できます。管理者権限が必要ならsudoを使いますし、SSHサーバも動かせます。APTによるパッケージのインストールは原則として無効化されていますが、フル機能のglibcが存在するので必要なバイナリをコピーして動かすことぐらいならできます。現在はまだ実験的ではあるものの、LibreOfficeやMikutterといったGUIアプリもそのまま動かせます。

すでにヨーロッパや中国ではUbuntu Touchインストール済みのスマートフォンが販売されていますし、本誌が店頭に並ぶころにはタブレットも販売されていることでしょう。日本でもベンダのオンラインショップから購入可能ではあるのですが、電波法という制約のためにこれらのデバイスを使うことはできません。現時点で日本でTouchを使いたいなら、Nexus 4やNexus 7(2013)にインストールするしかありません。

スマートフォン／タブレット向けのOSとして開発を始めたので、普段の入力手段はソフトウェアキーボードです。正確には「Maliit[注1]」と呼ばれる入力メソッドフレームワークをUbuntu用にカスタマイズして使用しています。MaliitはNokia N9をはじめとしたMeeGo端末で採用されていたフレームワークで、もともとプラグインという形で日本語キーボードも開発されていました。しかしながらUbuntu用にカスタマイズする過程でそれらのプラグインは取り込まれず、またUbuntu側ではより新しいQt5に移行したこともあって、プラグインをそのまま使うことはできません。

日本語キーボードがほしいという要望はおもに海外ユーザから上がっていたものの、誰も手をつけて

注1) http://maliit.github.io/

いない状態でした。しかも2013年当時、ほかの「第3のOS」と比較すると日本でUbuntu Touchを使用しているユーザはかなり限られていたので、日本人による自発的な実装はあまり期待できません[注2]。そこで一念発起して、筆者が実装することにしました（**図1**）。イベントで「Ubuntu Touchでは日本語入力できない」ことを伝えるたびに、慈愛と哀れみが相半ばした微笑みを返されるのはもう嫌だったのです。

日本語入力部分の実装

日本語入力の機能はおもに次の3つのパーツに大別できます。

① ユーザがかなやローマ字を入力するUI
② 入力した、かなを変換するかな漢字変換機能
③ ①と②の橋渡しをする入力メソッドエンジン（IME）

たとえばデスクトップ版Ubuntuの場合、①がFcitxとツールキットの組み合わせで、②がMozc、③はfcitx-mozcが担当します。このうちIMEは①と②を自由に組み合わせられるように作るレイヤです。よって入力メソッドフレームワークによっては不要な場合もあります。

Ubuntu Touchでは①をMaliitが担当し、②と③は各言語ごとに実装する必要がありました。ただし②については、既存のかな漢字変換プログラムを使うため、最低限実装しなくてはならないのは①と③になります。

かな漢字変換ライブラリの選定

入力メソッドエンジンの実装内容は、使用するかな漢字変換ライブラリに依存します。そこでまずはかな漢字変換ライブラリの選定から始めました。

当時のUbuntuが標準で使用していたかな漢字変換ライブラリはAnthyです。また日本語Remixは13.10ぐらい、本家も15.10からより新しく人気のあるMozcを使っています。機能面だけ見るとTouchでもMozcを使えるとベストなのですが、おもに実装までの作業量が多くなりそうだったので、早い段階から採用を断念していました。

さらに当時はAnthyの代替として、新しくlibkkc[注3]というプロジェクトが現れた時期でもありました。libkkcは変換精度もよく、Fedoraの標準かな漢字変換ソフトウェアとして採用されるなど、将来性も十分でした。実際、Maliitの開発者が来日されたときに日本語キーボードの実装について相談したのですが、そのときも「Anthyよりはlibkkcの方がいいのではないか」とアドバイスされました。

libkkcについてはそもそもDebianパッケージ化されていなかったため、Touchで使うとなるとまずはDebianパッケージにして公式リポジトリに取り込んでもらう必要があります。

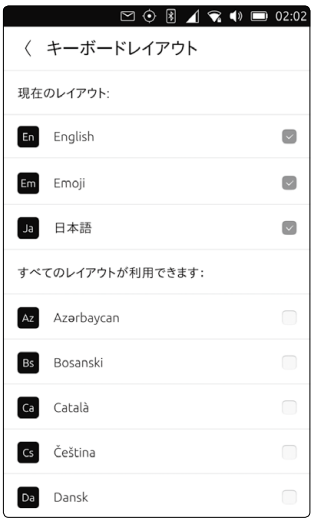
MARISA/libkkcのパッケージング

libkkcは辞書データの保存形式としてMARISA[注4]を使用しています。これもDebianパッケージ化されていなかったので、まずはこのパッケージングから始める必要がありました。これ自体は2013年に行われた大統一Debian勉強会で作業して、本誌Debian

注2） 国内でも比較的人気のあったTizenやFirefox OSにSailfishは、いずれもUbuntuよりだいぶ前に日本語入力機能が実装されていたようです。

注3） https://github.com/ueno/libkkc
注4） https://github.com/s-yata/marisa-trie

図1　設定画面から日本語を選択できる



Hot Topicsでもお馴染みのやまねさんにsponsorになってもらって、2014年の1月ぐらいにはリポジトリに取り込まれたのです。MARISAはswigを用いてPython、Ruby、Perlのバインディングも生成しています。それぞれの言語のパッケージングルールや、MultiArch対応のためのビルドスクリプトのカスタマイズに苦労しました。

MARISAの次はlibkkcのパッケージングです。しかしながら、そこからリポジトリに取り込まれるまでは時間がかかりました。ちょうどDebian側の新規パッケージの審査キューがたまっていたこと、リジェクトされてもすぐに対応できる時間がとれなかったことなどから、最終的にリポジトリに取り込まれるまで半年近く経過しています。これが2014年の初夏ぐらいの話です。

結果としてlibkkcが取り込まれるのを待たずに、Anthyを使ってUIとIMEの実装を先に進めることにしました。あくまで暫定的な先行実装です。本番ではあらためてlibkkcで実装しなおします。そう心に誓ったのです。

ビルド環境構築

まずはMaliitのビルド環境を構築します[注5]。Ubuntu Touch用の構築方法はほぼドキュメントがない状態だったので、トライアンドエラーで対応することになりました。Maliitはコアコンポーネント部分をQt5/C++で、UI部分をQMLで実装しています。C++の部分はクロスコンパイルなりネイティブビルドが必要ですが、QMLの部分はアーキテクチャ非依存ですのでただのコピーでも問題ありません。当初は実機上でビルドしていましたが、途中からx86エミュレーターが動くようになったので、そちらで試すようになりました。

Maliitのフレームワーク部分はmaliit-framework、UIとIME部分はubuntu-keyboardという名前のパッケージです。今回はUIとIMEの修正だけですので、ubuntu-keyboardパッケージのみを変更しています。

注5）本記事での「Maliit」は「Ubuntu Touchのソフトウェアキーボード」も意味することとします。

フリック入力の実装

スマートフォン向けの日本語入力は50音をテンキーに配置したフリック入力が一般的です。ただし、とくにタブレットの場合、英字キーボードを用いたローマ字入力もそれなりに利用されています。可能であれば両方実装したいと考えているのですが、今のところはフリック入力のみ実装しています（**図2**）。

もともとUbuntu Touchに実装されていたソフトウェアキーボードは英字配列で、記号類はレイアウトを切り替えるというタイプでした。また、フリック入力のような十字のポップアップ機能はありません。ただし、キーを長押しすると選択肢が出てダイアクリティカルマークありの文字を選べるしくみは存在しました。そこで日本語キーボードではこのしくみを参考にフリック入力部分を実装しています。

フリック入力やレイアウト切り替えについてはQMLのみで実装しています。入力したキーに対する濁点・半濁点の追加や小書き文字への変更は、JavaScriptを使いました。レイアウトのカスタマイズだけであれば、テキストエディタでQMLファイルを編集し、システム上のQMLファイルを置き換えるだけですのでとても簡単です。ただ、Ubuntuデザインを継承しつつ日本語入力に適したデザインにする部分は苦労しました（**図3**）。

とくに未確定文字列（プリエディット）領域の扱いは、日本語とそれ以外の言語で異なります。たとえば英語の場合、プリエディットは自動補完・自動校正でのみ使っています。英数字を入力するとプリエディット領域を校正しつつ、補完候補が表示され、候補を選ぶか英数字以外を入力すると確定というしくみです。

つまり日本語のようにプリエディット状態で文字列の途中を編集できません。編集しようとカーソルを移動した段階で、確定します。また英語では、一度確定した単語を編集した場合は未確定状態になります。この未確定にする単語の区切りは半角空白ですので、日本語の場合入力した一連の文字を未確定状態にしようとして、「長過ぎる」とエラーになります。つまりそのままだと未確定の文字列も確定した文字列も編

集できないのです。タイピングは常に一発勝負という、なかなかスリルのある入力方式です。

実際のところこのままでは使い物にならないので、日本語キーボードの場合は自動補完設定の有無にかかわらずプリエディットを有効にし、プリエディット状態でのカーソル移動や文字の追加と削除ができるように変更しました。また確定文字列を未確定に変更する機能は無効化しています。

マージリクエストの送付

2日で1時間ぐらいの作業を1ヵ月ほど続けて、フリック入力とAnthyを用いた最低限の日本語入力はできるようになりました。しかしながら、まだ足りない機能があったのと将来的にlibkkcに移行するつもりだったのもあって、先にフリックによるかな入力だけ取り込んでもらうことにしました。Ubuntuキーボードのソースコードは Launchpad 上の Bazaar で管理されています。このためGitHubでいうところのフォーク／プルリクエストのように、Launchpad上ではブランチマージリクエストを送ります。このマージリクエストを送ったのが2014年の3月ぐらいです。

リクエストから数日で開発者から「レビューするので待ってて」という返事をもらいました。しかしそこから12月ぐらいまで、とくに反応がない状態が続きます。さらに間の悪いことに、ツールキットの大きな更新があって、実装を見直す必要が出てきました。

そのあと、なかなか時間がとれなかったのですが2015年の夏ごろにようやく再実装が完了し、8月15日に再度リクエストを出すことができたのです。今度は前回の状況をふまえて、一度にマージしてもらえるようAnthy対応部分もセットで提出しました。前に「本番ではlibkkc」と書いてあるかもしれませんが、見なかったことにしてください。

そこからは問題点の指摘とその修正の繰り返しです。だいたい1ヵ月に一度ぐらいの頻度でレビューがあり、指摘事項の修正を行いました。なんか文通をしている気分です。通算9度の再提出を経て、ようやく2月22日に本家に取り込まれました。最初にリクエストを送付してから1年が経とうとしていました。

日本語入力のこれから

無事にUbuntu Touchでも日本語入力をサポートするようになったものの、機能としては全然足りていません。

文節の長さの変更やアンドゥといった当たり前の機能も未実装ですし、記号や絵文字の入力も一手間必要です。タブレット向けにフルキーボードによるローマ字入力も対応しなくてはなりません。Anthyではどうしても変換精度が劣るため、libkkcに移行したいとも考えています。またインターネットを使った変換機能もほしいところです。

Ubuntu TouchにBluetoothキーボードを接続すること、デスクトップのようなマルチウィンドウのインターフェースに切り替わります。このときの入力はハードウェアキーボードを使うことになるのですが、このときの日本語入力はまだ未対応です。

このようにやらなくてはならないことはまだまだ山積みです。もし日本語入力やソフトウェアキーボードの実装に興味があるなら、ぜひ開発に参加していただければうれしいです。SD

図2　フリック入力

図3　QMLによるポップアップ

Linux カーネル観光ガイド

第51回 Linux 4.5で新たに導入されたcgroup v2への変化

Text：青田 直大　AOTA Naohiro

3月26日にLinux 4.6-rc1がリリースされ4.6シリーズのデバッグが続き、4月10日に4.6-rc3がリリースされています。順調にいけば、この号が出ているころにはLinux 4.6がリリースされているでしょう。

今回は、Linux 4.5におけるcgroupのv2への変化を中心にcgroupについて紹介します。

cgroupとは何か

cgroupとはControl Groupsの略で、プロセスをグループ分けし、各グループにリソース割り当ての制限や、デバイスへのアクセス制限といったグループごとのさまざまな制御を可能とするシステムです。

グループは階層的構造をとり、グループ階層はファイルシステムとしてユーザ空間から見え、mkdirやechoを使ってグループの作成、プロセスのグループへの割り当てを行うことができます。

分類された各グループに、リソース制限など何らかの操作を行うモジュールをsubsystemと呼びます。subsystemには、グループの使用メモリを制限するMemory Controller、ブロックI/Oを制限するIO Controllerといった各種リソースを制限するものもあれば、あるいはグループ単位でperfのイベントの記録を行うperf_events Controllerや、グループ内のプロセスをまとめてfreezeするfreezer Controllerのようにグループ内のプロセスに対して一定の操作を行うためのものもあります。

2008年にLinux 2.6.24でcgroupが初めてマージされてから、さまざまなsubsystemが導入されてきましたが、プロセスのグループ分けを行うコア部分は長らく大きな変化はありませんでした。しかし、Linux 4.5においてcgroup v2という新しい“バージョン”のcgroupが導入されました。それまでのcgroup（cgroup v1）とcgroup v2とではmountの方法も違えば、可能なツリーの構成も違いますし、同じリソースを制限するためにアクセスするファイルの名前も変わっています。今回はcgroup v1について見たあと、cgroup v1にあった問題点とv2での解決策、そしてcgroup v2の使い方を紹介します。

cgroup v1

では、cgroup v1について見ていきましょう。前述したようにcgroupではプロセスをツリー状にグループ分けでき、そのグループはファイルシステムツリーでアクセスできるようになっ

ています。まずは、そのファイルシステムツリーがどこにmountされるのかを見てみましょう。本来はcgroupのツリーは自分でmountするものですが、現在多くのディストリビューションで採用されているsystemdは自動的にcgroupのツリーをmountしています。mountコマンドの出力からcgroupのものを抽出すると**図1**のようになっています。

/sys/fs/cgroupはデフォルトで用意されているcgroupのmountポイントです。1つのツリーだけをmountするときはここに直接mountすればよいです。systemdのように複数のツリーをmountする場合は、ここにtmpfsをmountし、その下にツリーごとのディレクトリを作成してそれぞれmountするのが一般的です。

/sys/fs/cgroupの下にはsystemdによって“blkio”から“systemd”まで複数のディレクトリが作られています(**図1**)。“systemd”ディレクトリ以外はcgroupの各subsystemに対応していて、mount optionにもそれぞれのディレクトリで有効なsubsystemが表示されています。たとえば、“/sys/fs/cgroup/devices”の行を見るとoptionが“(rw,nosuid,nodev,noexec,relatime,devices)”となっていてdevices subsystemが有効なツリーであることがわかります。cgroup v1では、このように各mount pointで有効なsubsystemを選択できます。

systemdによるmountではほとんどのsubsystemに個別のツリーを割り当てていますが、中には“/sys/fs/cgroup/cpu,cpuacct”、“net_cls,net_prio”のように複数のsubsystemが同時に有効になっているものもあります。これらは役割が似通っているsubsystemに行われているもので、たとえば“cpu”は各グループにどれだけのCPU時間を割り当てるのかを設定できるsubsystemで、“cpuacct”は各グループが実際にどれだけのCPU時間を使っているのかを計測するsubsystemとなっています。

mount pointの下はどうなっているのか見てみましょう。/sys/fs/cgroup/devicesの中は**図2**のようになっています。“cgroup.*”および“tasks”、“notify_on_release”、“release_agent”はどのsubsystemにも共通のファイルで、たとえば“cgroup.procs”は各グループに分類されているプロセスのIDを保持／設定するファイルで、“tasks”は各グループに分類されているスレッドのIDを保持/設定するファイルです。

“devices.allow”、“devices.deny”、“devices.list”がdevices subsystemに固有のファイルです。“devices.allow”と“devices.deny”がそれぞれ、そのグループのスレッドからアクセス可能なデバイスのホワイトリストを設定するためのファイルで“devices.list”がそのホワイトリストを読み出すためのファイルになっています。

そのほかに“init.scope/”、“system.slice/”、“user.slice/”の3つのディレクトリがあり、それぞれsystemdによってグループが作成されています。“init.scope/”と“user.slice/”の下にはsystemd

▼図1　mountされているcgroupツリー

```
# mount | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release_agent=/usr/lib/
systemd/systemd-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
# ls /sys/fs/cgroup
blkio  cpuacct      cpuset   freezer  memory   net_cls,net_prio  perf_event
cpu    cpu,cpuacct  devices  hugetlb  net_cls  net_prio          systemd
```

のserviceごとにグループが作られています。そのほとんどは“devices.list”が“a *:* rwm”となっていて、すべてのデバイスについて読み／書き／device fileの作成ができるという状態になっています。制限がついている例を見てみると、たとえば“systemd-timesyncd”は、/dev/nullなどのcharacter device(1:3など)と5:0、5:2、136:*といった端末のデバイスにだけアクセスできるように設定されています。

では、ほかのsubsystemのツリーはどのようになっているのでしょうか。cpu subsystemのツリーを見てみると、ディレクトリが何も作られていないことから、こちらでは何のグループも作られていないことがわかります(**図3**)。

このようにcgroup v1では各mount pointでまったく別のグループ分けを行えるようになっています。各subsystemごとにグループを作れることから、柔軟なポリシー作りが可能となります。たとえば大学のマシンで、学生にはCPU時間の30%、教員にはCPU時間の70%を与えるポリシーがある状態を想定してみましょう。ここに大規模なデータ処理を行うプロジェクトが発足し、そのプログラムにはメモリの80%が使えることを保障したくなった場合、1つのツリーではうまくグループを作るのが困難です。しかし、それぞれのsubsystemで別のツリーがあれば簡単にこのポリシーを実現できます(**図4**)。

さて、このようにプロセスはそれぞれのsubsystemごとに別のグループに分類され得ることがわかりました。この環境では、これらのツリーの中で一番細かく分類されているのは“name=systemd”のツリーになっています。

“name=systemd”のツリーは“release_agent=/usr/lib/systemd/systemd-cgroups-agent,name=systemd”のoptionでmountされていて、どのsubsystemも有効になっていません。name=systemdはこのツリーを識別するためのoptionで、release_agentはあるグループのス

▼図2　devices subsystemのcgroups

```
$ ls -F /sys/fs/cgroup/devices
cgroup.clone_children  cgroup.sane_behavior  devices.deny  init.scope/       release_agent  tasks
cgroup.procs           devices.allow         devices.list  notify_on_release  system.slice/  user.slice/
$ cd /sys/fs/cgroup/devices
$ ls -F init.scope user.slice
init.scope:
cgroup.clone_children  cgroup.procs  devices.allow  devices.deny  devices.list  notify_on_release  tasks

user.slice:
cgroup.clone_children  cgroup.procs  devices.allow  devices.deny  devices.list  notify_on_release  tasks
$ ls -F system.slice
boot-efi.mount/        devices.allow      -.mount/            systemd-fsck-root.service/      systemd-udevd.service/
cgroup.clone_children  devices.deny       nfs-idmapd.service/  systemd-journald.service/       systemd-udev-trigger.service/
cgroup.procs           devices.list       nfs-mountd.service/  systemd-journal-flush.service/  systemd-update-utmp.service/
dbus.service/          dev-mqueue.mount/  nfs-server.service/  systemd-logind.service/         systemd-user-sessions.service/
…
$ grep . system.slice/libvirtd.service/devices.list
a *:* rwm
$ grep . system.slice/systemd-timesyncd.service/{cgroup.procs,devices.list}
system.slice/systemd-timesyncd.service/cgroup.procs:707
system.slice/systemd-timesyncd.service/devices.list:c 1:3 rwm
system.slice/systemd-timesyncd.service/devices.list:c 1:5 rwm
system.slice/systemd-timesyncd.service/devices.list:c 1:7 rwm
system.slice/systemd-timesyncd.service/devices.list:c 1:8 rwm
system.slice/systemd-timesyncd.service/devices.list:c 1:9 rwm
system.slice/systemd-timesyncd.service/devices.list:c 5:0 rwm
system.slice/systemd-timesyncd.service/devices.list:c 5:2 rw
system.slice/systemd-timesyncd.service/devices.list:c 136:* rw
```

▼図3　cpu subsystemのcgroups

```
$ ls -F /sys/fs/cgroup/cpu,cpuacct
cgroup.clone_children  cpuacct.stat          cpu.cfs_period_us  cpu.rt_runtime_us  notify_on_release
cgroup.procs           cpuacct.usage         cpu.cfs_quota_us   cpu.shares         release_agent
cgroup.sane_behavior   cpuacct.usage_percpu  cpu.rt_period_us   cpu.stat           tasks
```

レッドが終了し、結果なんのスレッドもそのグループに所属しなくなったときに呼び出されるプログラムです。すなわち、このプログラムを使って空になったグループを検出し、グループの削除を行うことになります。“devices”ではuser.sliceの下が分かれていなかったのが、“name=systemd”ではさらに細かく“user-1000.slice/session-1.scope”と分かれているように、このツリーはシステムの全体的なグループ分けを担当しているようです。

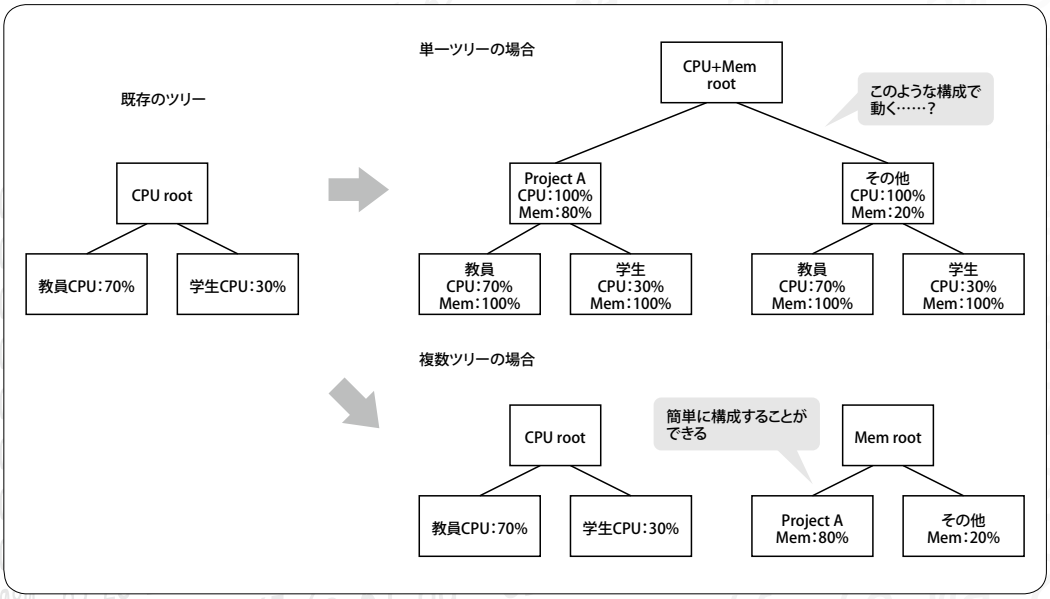
このようにsystemdではデフォルトではsubsystemによっては細かいグループ分けをしていませんが、リソースの使用状況を見る設定を行うとグループ分けが行われます。たとえば“systemctl edit”で適当なサービスに設定を行い、該当サービスをrestartするとdevices以外のツリーにも 各serviceに対応するグループが作られます。リソースの使用状況は“systemd-cgtop”コマンドで見ることができます。

cgroup v2

さて、このようにさまざまなsubsystemが実装され発展してきたcgroupですが、その進展のうちにさまざまな問題を抱え、cgroup v2として大きな変更が入ることとなりました。まずはcgroup v1にどのような問題点があったのかを見ていきましょう。

最も重要な問題点は、cgroup v1において柔軟な運用が可能になるといわれていた、システム全体で複数のツリーを持てることにありました。複数のツリーを持てるといっても、あるツリーで有効なsubsystemはほかのツリーでは有効にできないという制限があることから、freezerやperf_eventのようなどのようなツリーでも本来使えていいはずの機能もどれか1つのツリーでしか有効にできません。もちろん、一度有効になったsubsystemはツリーが有効な間は、ほかのツリーでは有効にできないので、結局systemdがやっているように(いくつかの似たようなものだけをくっつけて)subsystemごとに1つのツリーを作るようになってしまいます。さらにsubsystemごとに別々のツリーを持ち得るために、ほかのsubsystemのツリー情報を使うsubsystemを実装することは難しく、たとえばメモリとI/Oなどのように関連するsub

▼図4　単一ツリーと複数ツリーの比較



systemをきれいに実現するのが困難となってしまいます。

2つめの問題はツリーの途中のノードにもプロセスを分類できてしまうことです。途中のノードのプロセスをどのように扱うのかは難しい問題でそれぞれのsubsystemが別の対応をしています。たとえば、**図5**のようなツリーを考えてみましょう。cgroup v1ではAにもA-1にもプロセスが所属できます。では、AとA-1のプロセスにはそれぞれどのようにリソースを割り当てるべきでしょうか？ io subsystem 1つとってもその挙動は変化し、使うスケジューラや設定によって、すべてのノードがフラットであるかのように扱うこともあれば、見えないleaf nodeを作ってちゃんと階層構造が反映されるように動くこともあります。このように途中のノードにプロセスがあると実装に混乱を招くことになります。

また、そのほかにもスレッド単位でグループ分けができるがsubsystemによってはそれに意味がなく無視されていたり、subsystemごとにリソース統計用のファイルの名前が変わったり、その表示方法が変わるなどのインターフェースの混乱があるという問題もあります。

この問題点を解決するため、cgroup v2では、

①システム全体でただ1つのcgroupツリーを持てるようにする
②中間ノードにはプロセスを割り当てられないようにする
③スレッド単位ではなく、プロセス単位のグループ分けにする
④インターフェースの統合と整理

といった変更が行われています。一番大きな違いが複数のツリーを持てなくなったことであることからか、cgroup v2のことを“unified cgroup hierarchy”と呼ぶこともあります。

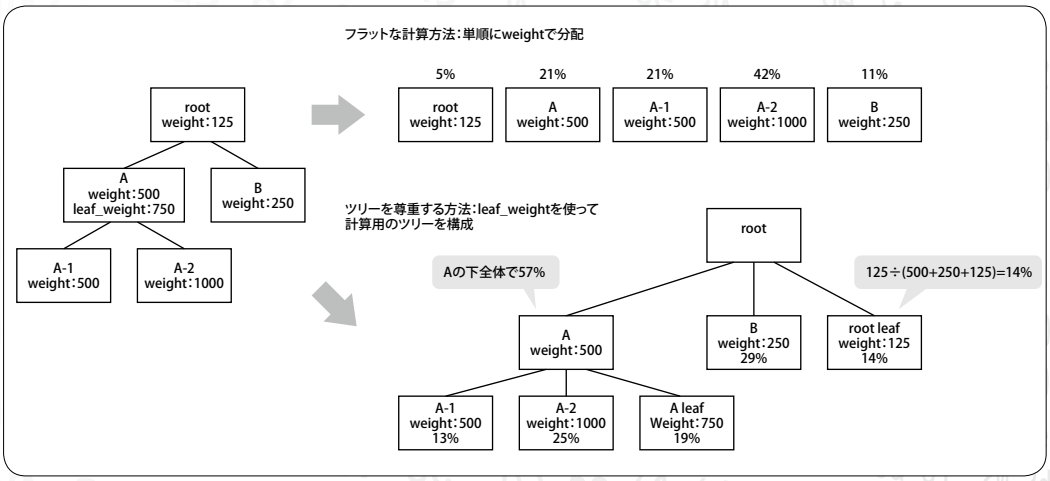
cgroup v2の使い方

cgroup v2にはまだcgroup v1にあったすべての機能が移植されているわけではありません。Linux 4.5の時点ではI/O、メモリ、プロセス数制限のみが実装されています(CPUがまだなんですね……)。そのため、cgroup v1とv2とは共存できるようになっていて、v1で使われていないものだけがv2で使えるようになっています。

カーネルのコマンドラインに“systemd.unified_cgroup_hierarchy”をつけると、systemd-230以降(執筆時点ではまだ開発中で未リリース)であれば、cgroup v1の代わりにv2でツリーがmountされます。

cgroup v2はファイルシステムタイプcgroup2で

▼図5　2つのリソース割り当ての計算方法



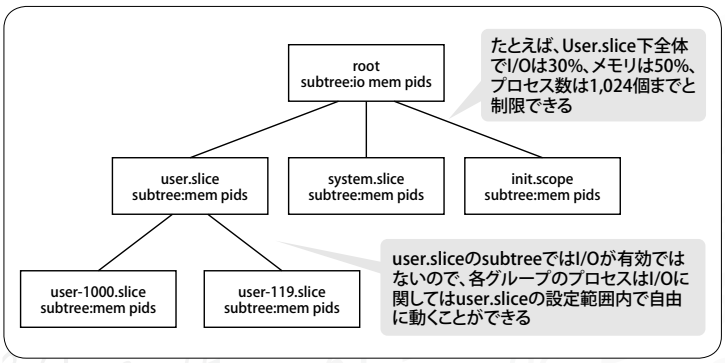
mountされます。すべてのsubsystemが1つのツリーで動作するようになるのでそのほかのmount optionはありません。ツリーの中を見てみると、systemdによってグループが作られています。

“cgroup.controllers”には、そのディレクトリまでで有効なsubsystemが書かれており、“cgroup.subtree_control”にはそのディレクトリより下(subtree)で有効なsubsystemが書かれています。ここでは“io”、“momory”、“pids”の3つのsubsystemが利用可能で、subtreeでは“memory”と“pids”だけが有効になっています。“user.slice/”の中には、たしかに“memory.*”や“pids.*”といったファイルがあるのが見えます。ここで“+<name>”や“-<name>”を“cgroup.subtree_control”に書くことで、subtreeで有効なsubsystemを追加／削除できます。たとえば“+io”を書いてみると、“user.slice/”の下に“io.max”などのio subsystemのファイルができることがわかります(**図6**)。

上のディレクトリの“cgroup.subtree_control”を変更しても、そこから下のディレクトリの“cgroup.subtree_control”には影響はありません。すなわち、現在ツリーのリソース制限は**図7**のように働くことになります。また、前述したようにcgroup v2では中間のノードにはプロセスを所属させることはできませんから、“user.slice/cgroup.procs”は空ですし、プロセスIDを書こうとしてもエラーになっています。

現状ではI/Oとメモリ、プロセス数しかcgroup v2では動かないため、実用には厳しい面もありますが、v2での大胆な変更によってインターフェースが統一され、cgroup v1にあったさまざまな問題が解決しています。これからどんどんsubsystemのv2への移植が進んでほしいところです(とくにCPUはドキュメントには載っているのに、実物はない状態ですので早くほしいですね:-))。SD

▼図7　現在ツリーのリソース制限



▼図6　cgroup v2のツリー

```
# mount|grep cgroup
cgroup on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime)
# ls -F /sys/fs/cgroup
cgroup.controllers  cgroup.procs  cgroup.subtree_control  init.scope/  system.slice/  user.slice/
# cd /sys/fs/cgroup
# grep . cgroup.{controllers,subtree_control}
cgroup.controllers:io memory pids
cgroup.subtree_control:memory pids
# ls user.slice/
cgroup.controllers  cgroup.procs           memory.current  memory.high  memory.max   memory.swap.current
pids.current  user-1000.slice
cgroup.events       cgroup.subtree_control  memory.events   memory.low   memory.stat  memory.swap.max
pids.max      user-119.slice
# echo +io > cgroup.subtree_control
# cat cgroup.subtree_control          # IO subsystemが有効になった
io memory pids
# ls -F user.slice/                   # io.max などが増えている
cgroup.controllers  cgroup.procs           io.max   memory.current  memory.high  memory.max   memory.swap.
current  pids.current  user-1000.slice/
cgroup.events       cgroup.subtree_control  io.stat  memory.events   memory.low   memory.stat  memory.swap.
max      pids.max      user-119.slice/
# cat user.slice/cgroup.procs         # 中間ノードにはプロセスが入らない
# echho $$ > user.slice/cgroup.procs
-bash: echo: write error: Device or resource busy
```

June 2016

NO.56

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
松山 直道　MATSUYAMA Tadamichi　ko@soum.co.jp

多分野の技術者が集結、これからのIoTを議論する

今回は、3月に一般財団法人日本インターネット協会（IAjapan）と一般社団法人日本ネットワークインフォメーションセンター（JPNIC）の主催で行った「IoTネットワークプログラミングワークショップ」（**写真1**）について報告します。

jusは、本イベントの協力団体であるIPv4アドレス枯渇対応タスクフォースに参加している立場から、イベントの告知や講師の推薦という形で貢献しました。

IoTネットワークプログラミング ワークショップ

■IoTネットワークプログラミングワークショップ
【日時】2016年3月16日（水）13:30～17:00
【会場】東京大学 電気系会議室5

昨今注目を集めるInternet of Things（IoT：モノのインターネット）の世界においては、クラウドからモバイル、インフラからサービス、およびビッグデータ処理と幅広い要素技術が必要となり、さまざまな分野やレイヤのプレイヤー同士が協力しあうことが求められます。そこで、プレイヤー間で状況や課題の共有を図ることを目的としてワークショップが開催されました。

今回のワークショップは、IoTに関連する各分野の方におもにそれぞれの分野における要素技術について講演いただき、最後に全員でディスカッションを行うことで分野を越えた共通の課題を共有するという形で進められました。会場の定員60名はほぼ埋まり、参加者は技術者が中心でした。しかし、その所属は通信事業者、SI事業者、ハードウェア／ソフトウェア製造業、教育機関と多彩で、各方面からの関心の高さがうかがえました。

■IoTシステム開発とビッグデータ処理

ワークショップはアラクサラネットワークス㈱の新善文さんの司会で始まり、はじめに東京大学の江

写真1　会場の様子

写真2　開会の挨拶　新氏（左）と江崎氏（右）

崎浩さんから、開会の挨拶としてワークショップ開催の背景と目的が説明されました（**写真2**）。

続いてアマゾンウェブサービスジャパン㈱の福井厚さんから、AWS（Amazon Web Services）が提供するサービスを活用したIoTシステム開発について、最初の講演をいただきました。IoTでは各ノードとの通信を担う中心的な役割としてクラウド上にサービスを構築することが多く、IoTが求める要件を満たすサービスであるAWS IoTの概要を中心に、AWSの取り組みが紹介されました。

引き続きビッグデータ処理システムであるHadoop/Sparkシステムの構築と運用について、Cloudera㈱のテクニカル・エバンジェリストの嶋内翔さんにお話いただきました。講演の冒頭で嶋内さんが会場に問いかけたところ、すでに実際にIoTに取り組んでいるという人は会場内にはまだほとんどおらず、ビッグデータを扱っている人もまだ少数ということにやや驚いておられました。多くの参加者は、これから取り組むための情報収集を目的として参加したものと思われます。

■IoTにおける物作りとセルラー通信

次の講演は少し雰囲気を変えて、Raspberry Piを使ったIoTの物作りと題して、Japanese Raspberry Pi Users Groupの太田昌文さんからRaspberry Piの紹介と、Raspberry PiでIoTをやってみる際のTipsやユースケースなどのお話をいただきました。おもに学生向けの教育用PCであるRaspberry Piは、高性能とは言えませんが低価格かつ頑丈であり、IoT用ノードを作る際の入門用としては最適な素材と言えそうです。

休憩を挟み、㈱ソラコムの安川健太さんからIoTにおけるセルラー通信活用ということで、ソラコムが提供する安全なモバイル通信網についてのお話をいただきました。ソラコムはMVNO（Mobile Virtual Network Operator：仮想移動体通信事業者）として、1日あたり10円の基本料金に従量料金を加えた安価なSIMを販売しており、個々の通信トラフィックは多くないが多数となるノードを扱うためのIoTプラットフォームを構築するソリューションを提供しています。また、モバイルノードの接続先であるソラコムから専用線でAWSほかのクラウドサービスに直結しており、インターネットを経由せず比較的セキュアな閉域IoTシステムを作れるところが魅力と言えそうです。

■IoTとセキュリティ

最後の講演は、「未来のIoTの姿とあるべきセキュリティを考える」と題し、国立研究開発法人産業技術総合研究所の大岩寛さんから、15年後を見据えたIoTのセキュリティについてのお話をいただきました。今のIoTの形はノードとクラウド上のサービスによるクライアント／サービス型のシステムが多数並立するものであり、それら相互の横方向の通信は考えられていないため、IoTというよりは「Internet of Intranet」となってしまっているのではないか、またプライベートネットワークならセキュアだという想定は正しいのか、という問題提起が行われました。IoTシステム同士が相互に通信しあうことで新たな可能性が埋まれることがIoTの本質であり、そういった環境をスケーラブルかつセキュアに実現する方法を考えることが今後15年間の課題となるだろうという主旨のお話となりました。

また、この講演のあと、すぐにディスカッションに入り、IoT機器に設定されたデフォルトパスワードの問題や、長期に渡って設置されるIoT機器に対するソフトウェア自動更新の問題、IPv6の必要性やプライベートネットワークの安全性神話などがおもな課題として共有されました。

■終わりに

今回は多岐に渡る講演テーマが集まり、いろいろな視点の人が意識を共有できた有意義な3時間半だったと感じました。第一歩として、当初の目的は達成できたように思います。

引き続き活動を継続していきたいということで、最後にIAjapan/ISOC-JPの藤崎智宏さんから挨拶をいただいて閉会となりました。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
及川 卓也　OIKAWA Takuya
Twitter @takoratta
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka

第54回 南相馬小高ハッカソン

2016年3月5日と6日に福島県南相馬市の小高区にて開催された帰宅支援ハッカソンについてレポートします。Hack For Japanからは及川がメンターおよび審査員として、高橋がプレイヤーとして参加しました。

開催までの経緯

小高フリーペーパー制作委員会[注1]と会津大学OpenAppLab[注2]の主催、南相馬市の後援で、会場は常磐線の小高駅（この付近の常磐線はまだ運行を再開していません）のすぐ近くにある双葉屋旅館にて開催されました。

南相馬市の小高区は福島第一原発から10kmから20kmの場所にあり、これまで避難指示の対象となっている地域です。近いうちに避難指示が解除される予定になっており、住民の皆さんの帰宅が始まろうとしています。今回のハッカソンはそれにあたっての課題を解決するものを作ろうという意図で企画されました。地元の皆さんに加えて、同じ福島県の会津若松、宮城県の石巻、そして東京などから30名ほどの参加者が集まりました。

「小高区での帰宅後の生活を支援するITサービス」というテーマで、一度住民がすべて避難してしまった街に再び戻ってくるという特殊な状況の中で、現地の生活や住民間のコミュニケーションを便利にするためのWebサービスやアプリを開発しようと参加者が奮闘した2日間でした。

注1　http://odaka.shosuriki.jp/
注2　http://www.u-aizu.ac.jp/research/uarc/oal/

ちなみに、初日のお昼ごはんには地元の小高商業高校の皆さんが考えたお弁当「復興味わい弁当」が供されました。豚の生姜焼き、小松菜の煮びたしなど福島県産の食材が使われたものでした。2日目のお昼にはおにぎりとカニ汁が供され、カニ汁には相馬の試験操業で取れたカニと、相馬松川浦の海苔が使われていました。おいしいごはんは、ご当地ハッカソンの楽しみの1つです。

チームビルディング、そして開発スタート！

事前のアイデアソンにて課題抽出が行われており、会場には交通・インフラ、コミュニティ、地域の情報や観光などの課題について19ものアイデアの種が貼りだされていました。その中から興味のあるものの場所に集まることでチームビルディングが行われました。こういった課題解決のためのハッカソンでは実際の開発ができるエンジニアの比率が低いということもよくあるのですが、今回はエンジニアの参加も多く、バランスの良いチームビルディングになったと思います。

初日午後の時点では4つのチームができ、各チームごとに部屋に別れて開発が始まりました。まずはコンセプトを詰めて、場合によっては地元の方にヒアリングを行い、開発するものの内容を決めていき

Column ハッカソン参加のいきさつ

2016年の3月でHack For Japanの活動を始めてから5年が経ち、これまでの活動でさまざまなところでつながりができています。今回のハッカソンも2011年4月に会津大学でミーティングを開催したときのご縁、そして2014年2月に行われたRace for Resilienceハッカソンの石巻会場（2014年9月号の本連載にてレポートを掲載）で知り合った森山貴士さんがその後南相馬に移住して活動されており、その森山さんが企画したイベントということもあって参加してきました。このようなつながりを広げ、継続していくこともHack For Japanの活動の1つだと考えております。

ます。

夕食後は引き続き開発を続けるチームや、お酒を飲みながら歓談する人たちなどさまざまな様子が見られました。会場が旅館だったため眠くなったら寝る場所があり、お風呂もあるというのは移動の手間がなく、また、運営にかかわった皆さんのきめ細やかな気配りのおかげもあって参加者は安心して開発に集中することができました。

成果発表

2日目の午後2時から行われた発表会は数多くの地元の皆さんも見守る中で行われました。

実質開発にかけることができた時間は約24時間で、途中で合体したり、1日目の夜になってから発生したチームもあり、最終的に発表までこぎつけたチームは次の5つです。

●e-またないタクシー

すでにある「e-まちタクシー[注3]」をさらに良くして「待たない」タクシーにしようと考えたもので、高齢の方の出かけたい希望をかなえます。電話をかけると自動音声の応答にしたがって「行き先が原町なら1」というように電話のボタンを押すことで呼び出すことができます（**写真1**）。

ご近所の信頼基盤を元に隣近所で登録制にしてあり、登録した人のところに通知が行き、呼び出した人と通話をして確認してから行くことになります。有料だと白タクになってしまうのですが、あくまで知り合いの互助がテーマなので法律には抵触しません。本当の目的は車で地域をつなぐことで、世代間、ニーズ、地域の話題をつなぎ、コミュニケーションの交差点となることです。

●プロジェクトおだかびと

空き家問題の解決を目指したもので、貸す側には地域とのつながりがない、すなわち信用のない人に貸すことへの不安、借りる側には不動産屋がない、市のサイトの空き家バンクでは情報が少ないという問題についての取り組みです。

おだかびとカードと名付けた住民カードで借り手の信用を担保し、貸し手の不安を解消します。カードにはおだかびとポイントが記録されており、ポイントはたとえば消防団に入ったり、地元のお店で買物をしたりすることで増えていくようにすることで、ポイントが高いと信頼できる人ということになります。この住民カードを使ってログインするとより多くの物件情報が得られるしくみを提供することができます（**写真2**）。

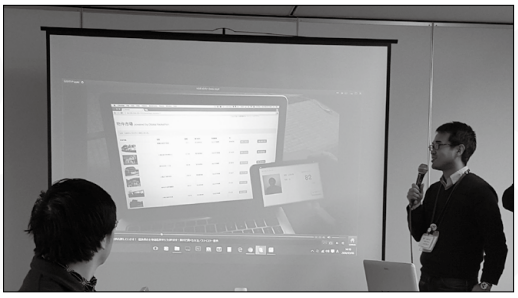
●シェア馬

南相馬には、相馬野馬追（のまおい）のために馬がたくさん飼われています。その中には気軽に乗馬させてくれるような飼い主さんもいたり、G1レースに出場していたような名馬もいるのですが、知られていないためほとんどの人が関係ない出来事になっているという現状です。こうした馬の情報を公開したり、乗馬を予約したりすることで、町の活性化を狙うサービスとして考えられました（**写真3**）。馬情報シェアリングサービスで性別、血統、経歴、競争成績などを表

注3　http://www.shokokai.or.jp/07/0756310003/index.htm

◆写真1　電話の自動応答による呼び出しのデモ

◆写真2　住民カードでのログインのデモ



示して乗馬予約のしくみを提供して、機会が少ない割に維持費が大きく、大きな費用がかかっているという問題を解決するものです。

●防犯ジョギングツール

南相馬市では「防犯ジョギング」という活動が始まっています。見回りを兼ねてジョギングするのですが、みんなが見回りしたところ、あまり行けていない場所などを可視化して、きめ細かいパトロールに役立てるしくみです。今回開発した範囲はジョギング時の位置情報を計測、表示するAndroidアプリとデータを保存するサーバサイドのAPIで、より多くの人が通ったメッシュは色を濃く表示することで可視化しています（**図1**）。

●チーム五十嵐 - Hack For Low

そのままではわかりづらい法律、条例の例文を身近に感じることができるしくみです（**写真4**）。

何かをやろうとしたときに、やって良いことなのか、許可がいることなのかがわからない……そんなときに判例をコンピュータに調べてもらったり、意見を提案できるものです。処理系にSWI-Prolog[注4]を用いて実現しています。

各賞と受賞チーム

今回のハッカソンでは、審査員を務めた小高ワーカーズベースの和田智行さんと加賀谷友典さん、そしてHack For Japanスタッフの及川の3名の名前を冠した賞と最優秀賞が用意されました。

注4　http://www.swi-prolog.org/

◆写真3　シェア馬の予約画面のデモ

各賞の受賞チームは次のようになりました。

◆図1　メッシュ情報の色の違いによる可視化

●小高ワーカーズベース賞

この賞は、より生活に密着した近々に困っている課題に取り組んだチームに、ということで、「e-待たないタクシー」に送られました。賞品は小高ワーカーズスペースで作っているガラスアクセサリーでした。魅力的な仕事と楽しい職場を作ることで若者に戻ってきてほしいという、このガラスアクセサリー作りに賭ける思いとともに賞品が贈られました。

●加賀谷賞

この賞は、タイトルの面白さもさることながら、実際のサービス内容がユニークで面白いということで、「チーム五十嵐」に贈られました。審査員の加賀谷さんは、難しい文章を読むのが苦手なところがあるというお母さんの話を例に出し、条文や規約などとっつきにくい文章をわかりやすく、簡単な問いで判定してくれるところを高く評価しました。

賞品は加賀谷さんがプロデュースしたnecomimiです。本誌の読者ならご存じの方も多いと思いますが、これは脳波で動作する猫耳の形状をした頭に装着するデバイスです。

●及川賞

この賞は、小高で実際に起きている課題を解決できることと他地域などへの発展性もあるということで、「プロジェクトおだかぴと」に贈られました。

日本のどこの地方都市も人口減少が課題であり、他地域からへの移住者を誘致しているのですが、地元に住んでいる人とどのように融和していくかが課題となっています。まだよく知りもしない人に、たとえば自分の家を貸すことに対して抱く不安と、地域にどのように馴染んでいくかその方法がわからな

いという移住者の双方の課題を、日々の活動の中から信用を蓄積していくというユニークなアプローチで解決しようという点を及川は評価しました。

賞品は及川が過去に執筆や監訳などでかかわった書籍の中からメンバーそれぞれに選んでもらい、それを贈ることになりました。

●最優秀賞

最優秀賞は「シェア馬」に決定しました。

審査は審査員3名で行われましたが、参加者と会場にいらした地元の方からも最も良かったチームを選んでいただきました。このシェア馬は会場の方々からも最も支持されたチームでした。

「南相馬と言えば野馬追」と言われるくらいに全国的にも知られている、野馬追のために飼われている馬の有効活用としてのアイデアが秀逸でした。馬好きにとっては野馬追に出ている馬に乗れるだけでも魅力的ですが、その中に過去にJRA所属の馬がおり、重賞レースでの入賞馬もいるということはかなりアピールできる点です。さらには、とくに昨今増えているインバウンドの外国人ツアー客はこのような日本の歴史を感じさせるものを好みます。このようにトータルで考えた場合の可能性は大きく、また実現可能性も高いということで最優秀賞に選ばれました。Webのシステムもすでに実績のある類似サービスを開発したメンバーがいたこともあり、完成度が高かったことも評価されました。

審査員の及川からは商品価値の高い野馬追の馬を活用し、南相馬の魅力を訴え、人を呼び寄せてほしいとコメントしました。同じく審査員の加賀谷さんからも、未来に希望を抱かせると今後へのさらなる期待が述べられました。

受賞チームメンバーからは、地元の方々と実際の事業化に向けての話し合いも持ちたいと、今後に向けての意気込みが語られました。

なお、最優秀賞の賞品としては、菜種油の石鹸、ホッキ飯のもと、地元の銘菓、福島県産の米「天のつぶ」5kgが送られました（**写真5**）。

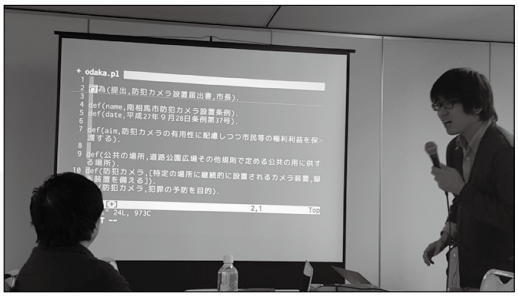
総評

最後に審査員を務めた及川と加賀谷さんから総評が寄せられました。

及川は、過去のHack For Japanなどが行ったハッカソンなどで、実際のニーズを把握できないことがあったことを振り返り、今回住民の皆さんからの声を反映させて開発を進められたことの意義を改めて強調しました。また、震災前と同じ街を作るのではなく、新しい魅力のある街にするための、このような地域外の人と地域の人が継続してかかわっていく取り組みを高く評価しました。

加賀谷さんは南相馬出身で今でも南相馬にご実家があります。小高駅前に来るのは1年半振りだったそうなのですが、当時と比べると格段に綺麗に明るくなったと復旧の進展を振り返りました。また、開催中に安倍首相が訪問したり、NHKでも特集を組まれているなど、小高に流れが来ているように感じると感想をお話されました。加賀谷さんは、ハッカソンと並行して地元高校生などとともに10年後を見据えた企画会議を開催していたのですが、この会議をさらに多くの地域の方や地域外の方と継続していきたいと抱負を語られました。SD

◆写真４　南相馬市の条例を例にしたしくみの解説



◆写真５　審査員の２人と優勝景品を手にするシェア馬チームの皆さん

温故知新
IT むかしばなし

第55回

DEBUGとSYMDEB～x86アセンブラの強力な支援ツール～

速水 祐（HAYAMI You）　http://zob.club/　Twitter：@yyhayami

はじめに

デバッガは、ソフトウェア開発になくてはならない支援ツールです。1980年代前半のマシン語開発ではデバッグの開発環境が十分には整わず、作成→実行→暴走を繰り返して、苦労してマシン語プログラミングを行っていました。80年代中盤になってMS-DOSが一般に普及すると、アセンブラはマクロアセンブラMASM（エムアセム）で、デバッグはDEBUG.COMとそれに続くSYMDEB.EXE（シンボリック・デバッガ。以下、SYMDEB（シムデブ））を使うことで、なんとかx86アセンブラの壁に立ち向かうことができました。今回は、このSYMDEBのお話をしましょう。

マシン語モニタからSYMDEBへ

1980年代の8bitマイコンはBASIC-ROMの中にモニタ機能[注1]を有しており、MONコマンドでモニタモードにして利用できました。16進データのマシン語をキーボードから入力し、ダンプコマンドでデータを確認して、実行コマンドで実行すると入力ミスがどこかにあり暴走。まず間違いなくデータは消滅してしまい、セーブを忘れていると悲惨で、毎度、そんなことを繰り返していました。

当時のマイコン雑誌では、貧弱なMONコマンドの機能を拡張したユーザプログラムが数多く公開されていました。その中には、簡易なアセンブラや逆アセンブラの機能を持つものもありましたが、デバッグ機能を持つものは少なかったようです。

1983年にMD-DOS 2.0が日本の16bitパソコンにも搭載され、MASMを中心としたLINKやLIBなどの開発ツールが普通に使えるようになり、デバッグ機能を持つDEBUG.COMが標準でMS-DOSツールに入りました。ただしDEBUG.COMは、前述した高機能モニタ程度の機能しかなく、ステップ実行機能やレジスタ表示機能はあるものの、実行途中で止めて状態を表示するブレークポイントがなく、デバッガとしては非力でした。

1985年にMS-DOS 3.1が登場すると、そこにはDEBUG.COMを機能アップしたSYMDEBがあり、コマンド名のとおりラベルやグローバル変数などのシンボルを扱う機能とブレークポインタ（一時停止位置）機能も追加され、コマンドラインながら使えるデバッグツールになっていました[注2]。

SYMDEBの機能

EXE形式の実行ファイルは、SYMDEBでロードするとアドレスが再配置され、またプログラム修正を行うとアドレスが変化するので、常にアセンブルリストと首っきりでブレークポイントの設定やメモリ内容の確認・設定をしなければなりません。

SYMDEBは、16進数のアドレスの代わりにラベルや変数名などのシンボルをそのまま使用できたので、デバッグ作業の効率が大きくアップしました。

ただしシンボルを扱うためには、プログラム作成時に若干の操作が必要になります。SYMDEBでうまくデバッグができる

注1）マシン語などを直接入力／セーブ／ロード／実行するモード。

注2）PC-9801用のMS-DOS 3.1ではMASMが拡張ツールとして切り離され、SYMDEBも標準MS-DOSディスクには含まれなくなりました。IBM-PC用のMS-DOS 3.1でも、MASMツールとしてMASMとともにSYMDEBは別売になっています。

ような、シフトJIS漢字文字列を表示するプログラムを作成してみました。

MASMでアセンブルしたあと、ファイル.OBJからファイル.EXEを作成するためにLINKコマンドを使います。その際、/MAPオプションをつけてシンボリックファイルの元となるファイル.MAPを生成します。

```
>MASM HELLO.ASM
>LINK HELLO /MAP
```

このあと、MAPSYMコマンドで、ファイル.SYMを作成し、このシンボリックファイルをEXEファイルと同時にSYMDEBに読み込むことで、シンボリックデバッグが可能になります。

```
>MAPSYM HELLO
>SYMDEB HELLO.SYM HELLO.EXE
```

PC-9821Np[注3]でSYMDEBを実行しました。逆アセンブルをしたあと、プログラムを実行しNEXTラベルの位置でプログラムを停止して、レジスタの内容を表示しています(**図1**)。

ここまでできると、かなり楽にデバッグ機能を使いこなせることになります。

Windowsの中にあるDEBUG.EXE

Windows 7(32bit版)まで、DEBUG.EXEが入っています。コマンドプロンプトから、

```
>debug
```

で起動します。簡単なx86マシン語プログラミングを試してみることができます。?コマンドで使用できるコマンド一覧が表示されます。また、aコマンドで直接、アセンブラプログラムを入力できるようになります。

```
-a
0B14:0100 mov ah,9
0B14:0102 mov dx,10c
0B14:0105 int 21
0B14:0107 mov ax,4c00
0B14:010A int 21
0B14:010C db 41,42,43,24
0B14:0110 ↵
-g
```

数値はすべて16進数[注4]です。「int 21」は、MS-DOSのファンクションコール(機能呼び出し)でWindows上でも動作します。ahレジスタにファンクションコールのコードを入れてからint 21を実行することで、さまざまなMS-DOSの機能を呼び出せます。たとえば「ah=9」は文字列表示のファンクションコールでdxレジスタに文字列の先頭アドレスを入れてからコールします。文字列の最後は「$」(コード:0x24)になります。プログラムを終了するには、「ah=0x4C」を入れ、alレジスタに終了コード(正常終了は0)を入れて実行します。

プログラムの入力が完了したら、改行キーだけを入力するとプロンプトに戻ります。gで実行できます。プログラムが暴走してリセットしなければならないという悲惨な目に遭わないように(短いプログラムなので入れ直せばよいが)、uコマンドで逆アセンブラ表示をすることで正しく入力できたかどうかを確認してから実行した方がよいでしょう。

文字列「ABC」が表示されれば正常に動作しています。30年前のアセンブラプログラムを経験することに、ぜひチャレンジしてみてください。

SYMDEBのその後

x86系のデバッガは、Windowsの前に一時期流行ったテキストベースの統合環境で動作する「CodeView」へ発展して、Windowsの時代になってからは「Visual Studio」へとつながり、デバッガを意識しないデバッグ環境が当たり前になったのです。

終わりに

現在は便利に使えるデバッグ環境がありますが、それに大きく頼るようなコーディングは改める必要があるかもしれません。仕様・設計をしっかり固めて、資料を深く理解して慎重に見通しの良いプログラムを記述すれば、デバッガの必要度は低くなり、信頼性の高いプログラム作成につながるはずです。SD

▼図1　SYMDEB画面

```
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [80286]
-u
_TEXT:START:
1684:0000 B88516        MOV     AX,DGROUP
1684:0003 8ED8          MOV     DS,AX
1684:0005 BA0200        MOV     DX,0002
_TEXT:NEXT:
1684:0008 B409          MOV     AH,09
1684:000A CD21          INT     21
1684:000C B8004C        MOV     AX,4C00
-g next
AX=1685  BX=0000  CX=001F  DX=0002  SP=0100  BP=0000  SI=0000  DI=0000
DS=1685  ES=1674  SS=1686  CS=1684  IP=0008   NV UP EI PL NZ NA PO NC
_TEXT:NEXT:
1684:0008 B409          MOV     AH,09
-
   C1    CU    CA    S1    SU      UOID  NWL   INS   REP
```

注3) PC-9821Np。筆者が最近復活させPC-9821の調査のために活用しているノートパソコン。Intel DX4 75 MHz 640×480 PEGC:プレーンアクセスモードとパックドピクセルモードという2種類のモードを持つ数少ないPC-98です。

注4) 各行の先頭の「0B14」はコードセグメントでマシン環境によって異なります。

うまくいく チーム開発のツール戦略

第2回 Bitbucket Server+SourceTreeで快適Git環境!

Author リックソフト㈱ 阿部 賢一(あべ けんいち)、大塚 和彦(おおつか かずひこ)

前回はメールとJIRAとの連携によるプロジェクト管理を説明しました。今回は、ソースコード管理・バージョン管理について説明します。

最近、「DevOps(デブオプス)」という言葉を耳にします。これは「開発(Development)」と「運用(Operations)」を組み合わせた言葉であり、開発と運用が連携して協力する開発手法の1つです。DevOpsを実現する要素として「ソースコード管理」「バージョン管理」「構成管理」などが必要といわれており、これらの要素をカバーするのが「バージョン管理システム」です。近年では集中型リポジトリのSubversionから、分散型のGit(ギット)が主流になりつつあるので、まずはGitのメリットや有効に活用する方法の紹介をします。

Gitのメリット

Subversionに対するGitの最も大きな利点は、ローカルリポジトリで作業できることと、マージが簡単なことです。

●ローカルリポジトリ上で作業できる

各個人は、リモートリポジトリをクローン(リポジトリをまるごと複製すること)したローカルなリポジトリで作業します。これにより、ほかのメンバーを気にせず作業でき、外部にアクセスしなくてよいので軽快に作業ができます。

コミットツリー編集機能を使えば、最終的に共有する前に作業内容を整理できるため、コミットやブランチの作成などを気軽に行えます。またGitは分散型なので、リモートリポジトリの負荷集中を避けられるというメリットもあります。

●マージが簡単

Gitではブランチ作成とマージがとても簡単です。ブランチを作る目的の1つは、メジャーバージョンアップやカスタマイズ案件などに向けた開発ツリーの分岐です。このケースはあまりマージが発生しないため、Subversion時代も普通に行われていました。

もう1つはチーム開発で重要な目的で、複数の開発者が安全に効率よく並行開発できるようにすることです。機能追加、修正のときに必ずブランチを切るようにすることで、ほかのメンバーによる変更との衝突チェックの煩わしさなしに、頻繁にコミットできます。積極的なコミットは、コードレビュー時や不具合個所の追跡時にとても役立ちます。

SubversionからGitへのスムーズな移行

GitはもともとSubversionの使いにくいところを改良し、良いところを伸ばす方針で開発されたものなので、Gitを利用するとコードを書くことにより集中できるようになります。

とはいえ従来のバージョン管理システムに慣れた開発者が乗り越えなければならない壁(考え方の違い)が確かに存在します。チームや社内全体にひろめていくためには、Gitの使い勝手を良くするツールは必須と考えてよいでしょう。次に紹介するBitbucket Server(Gitリポジ

トリ管理)、SourceTree(Gitクライアント)のようなサポートツールを使うことで、移行のハードルはずいぶんと低くなります。

Bitbucket Server

Bitbucket Serverはマスタ Gitリポジトリの管理などの機能を備えた便利なツールです。Gitリポジトリにプロジェクト階層を導入し、開発プロジェクト単位で管理できるようにします。プロジェクトにはアクセス権限を設定でき、さまざまな関係者が混在する大規模プロジェクトでもセキュリティを確保した開発が可能です。

開発者目線で注目したいのは、プルリクエスト(通称プルリク)やコードレビューの支援機能です。プルリクとは、プライベート環境で行った変更のマージをマスタに要求することです。これはコードレビューを行うのにちょうどよいタイミングであり、レビュー対象としてもほどよい粒度です。また、コードレビューの支援機能としては、レビュアの承認機能やインラインディスカッション機能があります。ソースコードにインラインでコメントを追加し議論することで、ソースコードと紐づいた形で議論が残るため、将来の修正で役に立つでしょう。

Bitbucket Serverはプロジェクト管理システム(JIRA)と統合されており、よく使われる機能をJIRAの課題上から操作できます。課題画面の開発セクションには、ブランチやコミット、プルリクなどの情報、操作リンクがあり、課題ごとの実装状況(実装中、レビュー中など)が一目でわかります。問題発生時には、課題→プルリク→頻繁なコミット→ソースコードの該当個所と、段階的な追跡が可能になります。

ほかにもセキュリティの考慮などさまざまなサポートがありますが、詳しくは過去記事(http://gihyo.jp/dev/serial/01/project_manager_tool/0003)をお読みください。

SourceTree

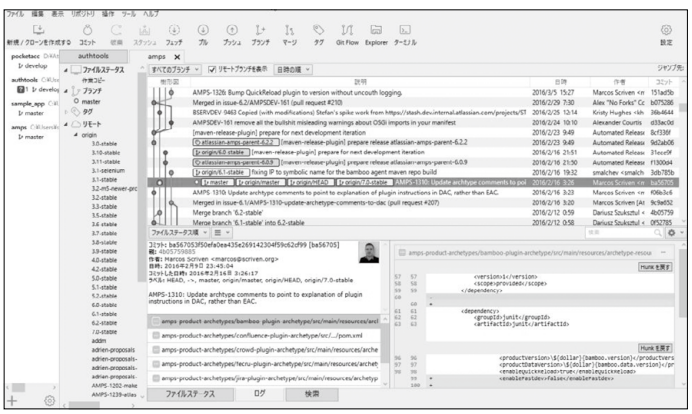
ローカル環境でGitを用いてソースコードを管理する方法の1つはコマンドライン操作ですが、それなりに覚えることが多く、Gitの概念をよく理解していないと扱いづらいです。ここでは、アトラシアン社が無料で提供しているGitクライアントアプリケーションであるSourceTreeを紹介します。

SourceTreeはBitbucket Serverと親和性が高く、グラフィカルなUIを持っています。単体としてもよくできたGitクライアントで、コードを書くことに集中できます。

たとえば実装を始める際は、まずサーバにあるリポジトリをローカル環境にクローンしますが、SourceTreeではクローン作成操作時にリポジトリ一覧が表示されるので、それを選ぶだけで準備が完了します。よく使う機能はツールバーとして前面に出ているので、基本的な操作に迷うことはないでしょう。使用頻度の低い機能ももれなくサポートしているので、入門者だけでなく熟練者にもお勧めです。

SourceTreeの画面には、ブランチ構造やブランチ名、コミット履歴などが見やすい形で表示されます(**図1**)。チーム開発ではコミットツリーの出入りが激しくなるので、状況を逐一理

▼図1 SourceTree



解するための視覚化は重要です。

また、SourceTreeは変更内容を一時的に保存するスタッシュコマンドもサポートしています。Git操作が正しいか不安になりがちな入門者にとっては、とりあえず変更個所を保存できるスタッシュコマンドはたいへん心強い存在です。スタッシュ一覧や、保存内容の差分表示などは便利で手放せません。

使ってみる

バージョン管理システムとプロジェクト管理システムの連携によって実装作業がどのように効率化されるかを、実際に少人数のアジャイル開発を行っている現場の開発作業の実例を交えて説明します。まずは、実装準備フェーズです。

0. 今やるべき課題を見つける

始めに、課題の一覧やカンバン、スクラムボードから次に手を付ける課題を選び、課題を表示します(**図2**)。

1. JIRAチケットからトピックブランチを切る

トピックブランチを切り、選択したJIRA課題を解決する実装を行います。JIRA課題の開発セクション(課題画面右下)にある「開発」→「ブランチを作成」をクリックすると、Bitbucket Serverのブランチ作成ページが表示されます。Bitbucket Serverから課題をたどれるように、課題キー(ここではGY-1)をブランチ名に残しておきましょう。ブランチの作成が成功すると、新規に作成されたブランチのファイル一覧画面が表示されます。JIRA課題からBitbucket Serverへの移動を意識せずにブランチを切ることができます。

▼図2　JIRA課題画面と開発セクション

2. SourceTreeでブランチをチェックアウト

次に、サーバ側に作成したブランチをローカル環境にチェックアウトします。SourceTreeのツリー画面で、リモート階層以下にあるブランチGY-1を右クリックし、チェックアウトを選びます(ブランチが表示されない場合は、メニューから「リモートのステータスを更新」を実行します)。チェックアウトしたファイルの場所は、サイドバーでリポジトリを選択し、ツールバーのExplorer(Windows環境の場合)をクリックすると取得できます。

これで実装の準備が整ったので、ローカル環境で実装を行っていきます。

3. ソースコードを編集＆コミット

ソースコードをIDEやエディタで編集していきます。意味ある変更ができたら積極的にコミットします。コミット一覧画面上の「コミットされていない変更があります」を選択すると、未コミットの変更内容が表示されます。コミット対象に含めたいファイルにチェックを入れると「Indexにステージしたファイル」に項目が移動します。差分表示画面の「Hunkをステージへ移動」ボタンで、ファイル内の変更の一部のみをコミット対象とすることもできます。

コミットは、「ファイルステータス」タブでコミットメッセージを入力し、コミットボタンを押すと実行されます。サーバにプッシュされたコミットは、JIRA課題画面にすぐに反映されます。

課題の要件を満たす実装を終えたら、次に、レビューの準備のためにプルリ

クを作成します。

4. Bitbucket Server: プルリク発行

SourceTreeのブランチを右クリックして表示されるメニューから、プルリクエスト作成画面へ移動します(JIRA課題からもプルリクを発行できます)。プルリクエスト作成画面では、このブランチで行った作業内容を記述し、レビューを依頼するメンバーを指定して、「作成する」ボタンをクリックします。これでオンラインレビューの準備が完了しました。

もしマスタ側と衝突が発生していれば、プルリク画面に目立つ形で警告表示されます。その場合は、SourceTreeでマスタを選択して作業中のブランチにいったんマージし、衝突個所を修正します。プルリク画面をリロードすると、衝突が解消されたかどうかがわかります。

5. Bitbucket Server: レビュー実施

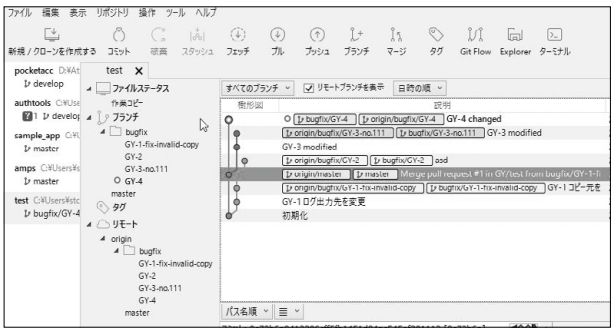
プルリクエスト全体に対するコメントのやりとりや、ソースコードが差分表示されたビューへのインラインコメント機能を通してレビューを行います。私たちはBitbucket Serverでオンラインレビューした後に、必要があればFace-to-Faceで追加レビューします。オンラインレビューで問題点、問題個所が明確になっているので、Face-to-Faceレビューに必要な時間はわずかです。

6. Bitbucket Server: プルリクの承認

レビューメンバーは、プルリクエストに対して最終的に承認または却下の意思表示を行います。最後に承認したメンバーが「マージ」ボタンを押して、マージを行います。これで実装作業の1サイクルが終了します。

チーム開発では、このサイクルを複数個同時に走らせることになりますが、このプロセスは破綻することなく機能します(**図3**)。

▼図3 複数開発者による並列開発



最後に特殊なプルリクについて紹介します。私たちは実験や調査のために、最終的にマージを目的としない、一般にWIPプルリクと呼ばれるプルリクを作成することがあります。このプルリクを使って、実装前に調査結果について議論したり、実装方法について助けを求めたりといった使い方をします。軌道修正を早めに行える、衝突の発生を早めに知ることができるなどのメリットがあります。

まとめ

Gitによって手軽になったブランチやマージを活用することで、複数のチーム開発者が迅速かつ安全に並行開発できます。さらに、Bitbucket Serverでプルリク時にオンラインコードレビューを行うことによって、大幅なレビューコストの削減、品質向上が得られます。実装工程で蓄えたコミットコメント、レビューコメントなどの資産は、問題発生時の原因追跡にもたいへん効果があります。ソフトウェア開発のみならず、コードレビューを含めた効率の良いチーム開発が必須となる、たとえばハードウェアのフロントエンド設計などの分野でも有効でしょう。

次回は、開発中に蓄積したチケットとブランチ、コミットとの紐づけ、細かなコミットを活かして、現行サービスに問題が発生した場合に、いかに解決するかについて説明していきます。SD

Event

「U-22プログラミング・コンテスト2016」開催決定

「U-22プログラミング・コンテスト2016」の開催が決定、Webサイトがオープンした。

本コンテストは、優れた才能を持ったイノベイティブなIT人材の発掘と育成、単にプログラムのできる人材ではなく、アイデアに富んだソフトウェア開発に取り組む人材の発掘を目的として開催されてきた。今年のキャッチフレーズは「未来を拓く創造力！　プロをうならせるアイデアと技術」。昨年に引き続き、「プロダクト」「テクノロジ」「アイデア」の3つの評価ポイントにより審査される。

参加資格は日本国内に居住する1994年4月2日以降に生まれた個人やチーム（チーム参加の場合、同じ学校に所属する学生であれば23歳以上のメンバも参加可）で、募集する作品は、未発表または2015年9月1日以降に発表したオリジナル作品。

●開催スケジュール（予定）

4月1日	応募要領発表／2016年版公式Webサイト開設
7月1日～8月25日	応募受付期間
8月～9月	事前審査、一次審査
10月2日	最終審査会・特別講演・各賞発表
10月3日	経済産業大臣賞・商務情報政策局長表彰式

CONTACT
U-22プログラミング・コンテスト2016
URL http://www.u22procon.com

Service

グレープシティ、クラウドサービス「Masume」を提供開始

グレープシティは複数のExcelファイルを1つにまとめる集約作業を自動化するクラウドサービス「Masume（ますめ）」を4月20日に提供開始した。

Masumeでは、集約したいExcelファイルをブラウザからアップロードするだけでExcelシート内の集計対象となるセルを認識し、1つのテーブル（表）に集約、データ集計まで行う。そして集計結果は、再利用可能なExcelファイルとしてMasumeからダウンロードできる。従来の運用フローやフォーマットとして使っているExcelを変更する必要はなく、Webブラウザだけですぐに利用できる。さらに、操作体系や画面はExcelに合わせられているため、学習負担も少ない。

本サービスはInternet Explorer、Microsoft Edge、Chrome、Safariなどの最新ブラウザに対応している。料金体系は月額制で、3,980円／月のライトプラン（5ユーザでストレージを500MBまで利用可能）と、7,980円／月のスタンダードプラン（10ユーザでストレージを10GBまで利用可能）の2つがある。

同社が提供している、Excel感覚で手軽にWebアプリを開発・運用できるツール「Forguncy（フォーガンシー）」とともに、社内のExcel業務をカイゼンできるだろう。

CONTACT
グレープシティ㈱　URL http://www.grapecity.com

Report

アイレット、「システムエンジニアが、クラウド業界で活躍するには？」座談会開催

4月15日、dots.イベントスペース（東京都渋谷区）にて、オンプレミスのシステムエンジニア向けの座談会形式セミナー「システムエンジニアの『キャリアを考える』座談会」がアイレット㈱cloudpack事業部主催で開催された。

週刊BCN編集長の畔上文昭氏による基調講演から始まり、アイレットディビジョンリーダの石田知也氏がこれまでの実体験に基づいたクラウド導入の奮闘の歴史を講演した。その後、週刊BCN編集委員の谷畑良胤氏をファシリテータに「クラウド環境で働くエンジニアに必要なスキルは何か」「オンプレミスとクラウドでのスピード感の違い」など現場ならではの話題をテーマに座談会が行われた。

▲左からcloudpack事業部 菊池 康之氏、岸上 健太郎氏

▲左からcloudpack事業部　比嘉 東一郎氏、武川 努氏、㈱BCN　谷畑 良胤氏

CONTACT
アイレット㈱　URL http://www.iret.co.jp

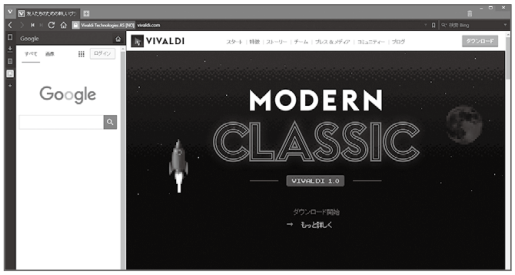
Software

Vivaldi Technologies、Webブラウザ「Vivaldi1.0」をリリース

Vivaldi Technologiesは4月6日、同社が開発するWebブラウザの正式版「Vivaldi1.0」をリリースした。

Vivaldiは、Opera Softwareの創設者兼CEOであったヨン・スティーブンソン・フォン・テッツナー氏によって立ち上げられたVivaldi Technologiesが開発を行っている、ChromiumベースのWebブラウザ。おもな特徴は次のとおり。

・タブの位置や移動の挙動などをカスタマイズできる
・タブをグループ化できる「タブスタック機能」
・ショートカットが豊富で、キーボードでの操作も快適
・「Webパネル」で複数のWebページを同時に閲覧可能



▲公式ページを閲覧。サイドバーにあるWebパネルではGoogleを閲覧

CONTACT
Vivaldi Technologies URL https://vivaldi.com

Service

日本電信電話、サーバアーキテクチャ「MAGONIA」を発表

日本電信電話㈱（NTT）のネットワークサービスシステム研究所は4月12日、新サーバアーキテクチャ「MAGONIA」を発表した。

MAGONIAはNTTの通信系システムのノウハウを活かした基盤技術（ミドルウェア）で、分散処理に強みを持つ。分散サーバ（物理サーバ、VM、コンテナ含む）上に本ミドルウェアを導入することで、その上に載せるアプリケーションを大きく変更することなく、スケーラビリティや信頼性、リアルタイム性に優れたシステムを開発できる。MAGONIAは現状、Linuxのうえで稼働することを前提としており、公開されているAPIをアプリから呼び出す形での利用となる。

MAGONIAの分散処理基盤では、現用系（処理中）のサーバが同時にほかのサーバのデータを冗長化して持つ予備系も兼ねる「N-ACT型クラスタ」を採用しており、あるサーバが故障してもすぐにほかのサーバが処理を引き継ぐことができる。㈱NTTデータが開発を進める「渋滞予測・信号制御システム」の処理基盤に採用され、交通シミュレーション中にサーバが故障しても、複数のサーバに迅速にフェイルオーバーして分析を継続できることが確認できているという。

CONTACT
日本電信電話㈱ URL http://www.ntt.co.jp

Topic

ウェブルート、「ウェブルート脅威レポート2016」を発表

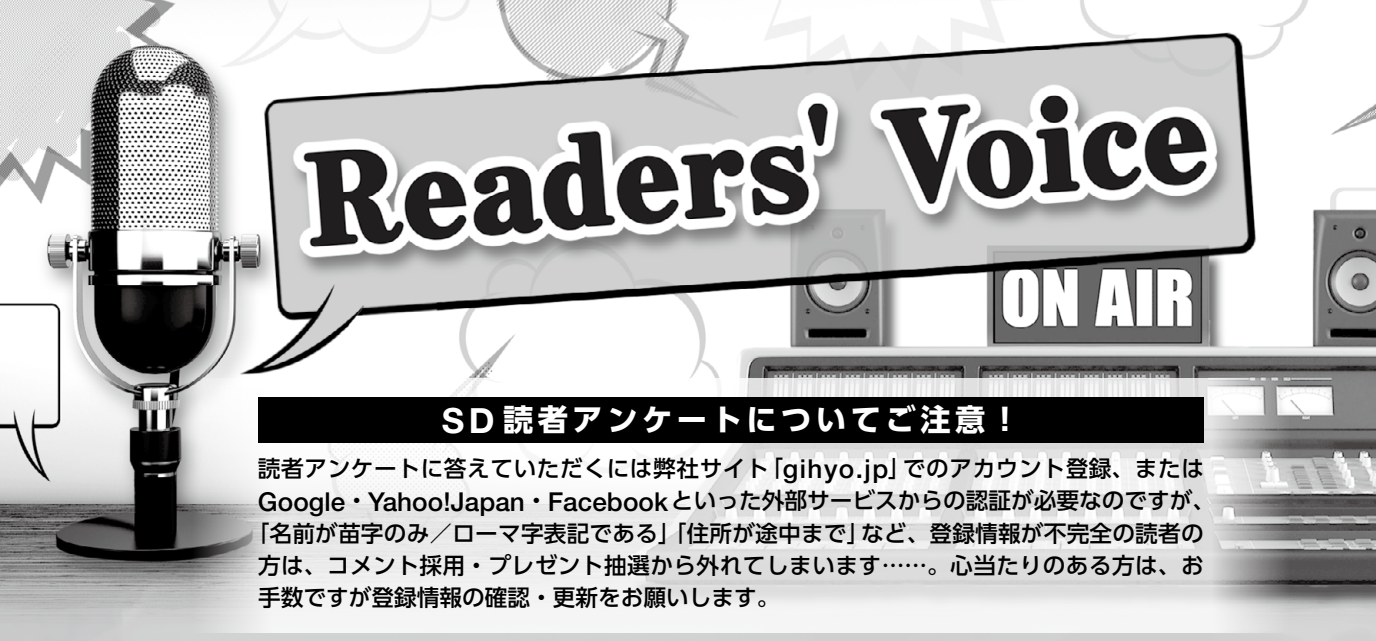
ウェブルートは4月19日、「ウェブルート脅威レポート2016」を発表した。

「ウェブルート脅威レポート」は同社が毎年発行しているセキュリティレポート。「Webroot Threat Intelligence Platform」が検知する270億以上のURL、6億以上のドメイン、40億以上のIPアドレスの分析結果を基にしている。おもな調査結果は次のとおり。

・マルウェアやPUA[注1]の圧倒的多数がポリモーフィック型で占められるようになり、マルウェアの97%がエンドポイントごとにユニークな形に姿を変える
・ウェブルートのユーザが1年間にゼロデイフィッシングサイトの攻撃を受ける確率は約50%で、2014年の約30%から上昇
・2015年後半の新規または更新アプリのうち52%がPUAもしくは不正なアプリであり、その割合は21%に過ぎなかった2014年から大幅に上昇
・不正なIPアドレスが最も生成されている国のトップ10に日本がランクイン（3位）、全体に占める割合は6%に大きく上昇

注1) Potentially Unwanted Applications：ユーザの個人情報やセキュリティを侵害する可能性があるプログラム。

CONTACT
ウェブルート㈱ URL http://www.webroot.com/jp/ja



SD読者アンケートについてご注意！

読者アンケートに答えていただくには弊社サイト「gihyo.jp」でのアカウント登録、またはGoogle・Yahoo!Japan・Facebookといった外部サービスからの認証が必要なのですが、「名前が苗字のみ／ローマ字表記である」「住所が途中まで」など、登録情報が不完全の読者の方は、コメント採用・プレゼント抽選から外れてしまいます……。心当たりのある方は、お手数ですが登録情報の確認・更新をお願いします。

2016年4月号について、たくさんの声が届きました。

第1特集 良いプログラムの書き方

C言語、Java、C#、Ruby、フロントエンド言語（JavaScript+HTML+CSS）について、それぞれの言語のベテランエンジニアが、“今すぐ実践できる”プログラミングのテクニックを紹介しました。脱初級者、目指せ中・上級者！

普段はJavaでしか開発をしていませんが、ほかの言語でも参考になることが多いので勉強になりました。
binaさん／東京都

最近Rubyを触っているので「他言語からRubyに来たユーザのハマりどころ」というテーマはとてもタイミング良く、勉強になりました。あとはJavaのラムダ式。そして最近の流行らしい、nullで死なない言語仕様。C99での「//」のコメントやgetsの廃止などがおもしろかった。いろいろ進化するものですね。
atachibanaさん／東京都

良い書き方って人によって違うから、教育する立場だと書き易さよりも可読性を優先してるけど、それで良いのかなって思ってる。
ももんがさん／静岡県

Javaの章を読みました。基本的な事項ですが、ためになりました。
山崎さん／神奈川県

Javaの仕事は1.4時代のメンテばかりなので仕様の変化を追いかけていなかったが、いろいろ仕様が変わっていて勉強になった。この特集は、言語仕様に左右される部分はあるものの、言語を問わず本質的に伝えたいことは変わらないと思うので、今年の新人に読ませたい。
隼さん／岩手県

メインで使っているPHPがないのが残念でしたが、C#もJavaScriptも書くので参考にさせていただきました。
hiroさん／神奈川県

ただの書き方ではなく、“良い”書き方ということで、ためになったという声が多く寄せられました。ただ、PHPの章も欲しかった……という声が（本当に）多かったです。

第2特集　オブジェクトストレージの教科書

オブジェクトストレージは、APIでファイルをシンプルに出し入れできる新しいストレージの形。本特集では、オブジェクトストレージのしくみ、OpenStack Swift・Cephのアーキテクチャを解説し、そして3つのオブジェクトストレージサービスを紹介しました。

基本から書かれていて、わかりやすくて良い。
田伏さん／愛知県

大容量データの保存に適しており、いずれ活用したいと思っていたところだったので、グッドタイミングでした。
オミオさん／宮城県

Amazon S3などのオブジェクトストレージについて、そろそろ使いたいなと思っていたところでしたので、ちょうどぴったりでした。
菊地さん／愛知県

オブジェクトストレージという単語を聞いたことはあったのですが、よく知らなかったため、概要を知る良い機会になりました。
橿山さん／埼玉県

名前を聞いたことがある、まさに今使いたいと思っていたなど、エンジニアの中でも認知度が高まっているようでした。システムを開発するとき、0から何かを作り上げるのはできるだけ避け、積極的に外部サービスを利用しようとするのは最近のトレンドですね。

4月号のプレゼント当選者は、次の皆さまです

①Wi-Fiホームルータ「Aterm WF1200HP2」
人儚 由夢様（福岡県）

②ヘッドマウントディスプレイ「DN-13539」
大澤和宏様（香川県）、佐藤法子様（千葉県）、Game-Maker様（東京都）

③Paragon Camptune X
ＹＹ様（神奈川県）、Tayu様（千葉県）

④『エンジニアのためのGitの教科書』
藏谷滋様（大阪府）、山口禎二様（青森県）

⑤『Effective Python』
加納一輝様（千葉県）、原田誠史様（神奈川県）

⑥『ネットワーク運用管理の教科書』
とーふや様（神奈川県）、石内博子様（福岡県）

⑦『【改訂新版】サーバ構築の実例がわかるSamba［実践］入門』
中山周様（宮城県）、Age Project.様（高知県）

※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。

特別企画　LANケーブリングの教科書［番外編］

2016年2月号第2特集「適切なLANケーブリングの教科書」で紹介しきれなかった「光ファイバ」「ラック・電源」について、あらためて特集しました。

◆その1　光ファイバの知識
光ファイバの規格、光モジュールと光コネクタの関係、配線の注意事項について解説しました。

自分では触らないので、むしろ興味がありました。 タブレットほしいさん／奈良県

ちょうど仕事で関係する分野なので、たいへん助かりました。 中山さん／東京都

Software寄りの雑誌で物理エンジニアの知識が学べるのは良い。
raihennさん／東京都

◆その2　ラック選定・電源の知識
ラックについては、その各部の名称、購入方法、機器搭載の注意事項を、電源についてはPDU、給電方式、UPSなどについて解説しました。

使い勝手が改善される点、興味深い。
鳥居さん／愛知県

データセンターのラック内に機器が配置されるまでには、いろいろな考察を経ているのだと実感しました。
永作さん／東京都

ラックの設置にまで議論がおよぶのであれば、「接地」についても触れてほしいと感じました。 鈴木さん／熊本県

仕事に関係がある／まったく関係がないと読者の声が割れました。IT系企業でも、社内ラックを持たない、データセンターに立ち入らないといったところが増えているのでしょうか。ただ、仕事に関係がないという人からも、知識としてためになったという声が多く寄せられました。

特別企画　春の嵐呼ぶ！DevOps座談会

DevOpsとは、開発と運用が連携して高頻度のデプロイを実現するアプローチ。今回はTKC、U-NEXT、Pivotalジャパンの各システム部門に在籍する方々に、「DevOpsは日本に定着するのか」をテーマに議論していただきました。

現場で働いている生の人の意見を読むと勉強になります。今使われているのはどういったものなのか、なぜそれが使われているのか、今後は何が必要になってくるのかというのが見えてくるからおもしろいです。 りょうじさん／東京都

企業におけるDevOpsに対する考え方が垣間見えて興味深かったです。
村橋さん／北海道

DevOpsに限らず、ツールよりも「考え方」というところが響きました。実践は難しいですが。
NGC2068さん／愛知県

DevOpsは情報が少ないので、もっと取り上げてほしい。 かずさん／千葉県

名前が独り歩きしている感もあるDevOps。企業で実際にどう取り組まれているのか、隣の様子が気になるという読者の方が多いようです。明確なルールがない以上、企業によって色が出やすいということがわかりましたね。

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
July 2016

2016年7月号
定価（本体1,220円+税）
192ページ

6月18日発売

［第1特集］ネットワークは怖くない!
「プログラマが知っておくべきTCP/IP」
プロトコルとコードで本質的理解

［第2特集］
手を動かし結果を見て学ぼう 正規表現入門
～プログラミング／エディタ作業の効率大幅アップ～

※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

「使って考える仮想化技術」（第2回）は都合によりお休みさせていただきます。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2016年5月号
●P.36　第1特集「Vim［実戦］投入 Part3」（量指定子：0個または1個?、=）
［正］¥vfunction(¥s+¥w+)?¥(　［誤］¥vfunction(¥s+¥w+)¥(
［正］function¥(¥s¥+¥w¥+¥)¥?(　［誤］function¥(¥s¥+¥w¥+¥)(
●P.58　第1特集「Vim［実戦］投入 Part5」右段最終行、注3）
［正］http://vim-jp.org/vimdoc-ja　［誤］https://vim-jp.org/vimdoc-ja

SD Staff Room

●今年の表紙写真は猫。なぜ猫かと問われれば、犬もやったしそろそろ猫かなと深い意味はありません。昨年末には決めていたのですが猫ブームが来ているので、それに乗ったと思われると辛い。本誌を持っていて、なごめるのが目論見なのです。冬の表紙はミカンとコタツ猫とかそのあたりまで決めています。（本）

●今はタケノコが旬。週末に朝掘りの大きめのものを調理する。若竹煮や筍御飯、煮付けなど旨い。自宅庭には何年か前に植えたアスパラが生えてきていて、採れたてはすごく甘い。山菜のコシアブラも取り寄せておひたしや天ぷらで初夏の香りを楽しむ。あっ、GW前に庭と水耕の栽培計画を立てなくては……。（くいしんぼ幕）

●今年度の健康診断は追加検査が2つも。1つは過去経験済みの胃カメラ。もう1つは胸部レントゲン撮影。胃カメラは「喉元過ぎれば熱さを忘れる」で割と気楽に受けたんですが、喉元を過ぎるとき……涙出ました。つらかった。レントゲン撮影では人生初のCTスキャンを体験。身体は大事にしないと。（キ）

●前々号の「良いプログラム」特集の企画の際、言語の選択に悩みました。読者アンケートの使用言語に関する回答では、いつもC#よりPHPのほうが多いのですが、SD読者の中では利用者が少なめのC#を載せるほうが新たな発見もあるのでは?と思い、あえてC#にしてみました。どうだったでしょう?（よし）

●休日に創作料理を作ることにハマっています。暇も潰せて腹もふくれて一石二鳥です。この間作ったのは『塩鮭サラダ』！　焼いた塩鮭をほぐし、生のベイビーリーフ・きゅうり、茹でたジャガイモ・アスパラと絡め、オリーブオイルと塩コショウで味付けしたものです。ギリギリ食べられる生臭さでした。（な）

●1年間PTA役員を務めた広報では、4月の年度始まりについての広報誌をGW明けに発行し終えたら、新役員さんに引き継ぎして交代！　年間2回発行とわりとゆるやかだったけれど、次の締め切りに追われることもなくなるし、ちょっと一息つけそう♪　、、、と思ったら、年賀状素材集のお仕事依頼がそろそろ来る時期でわ…!?　（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2016年6月号

発行日
2016年6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。