[特別付録]SORACOM Air SD Special Version

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

5
2016

2016年5月18日発行
毎月1回18日発行
通巻373号
（発刊307号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
本体 1,420円
+税

巻頭特集

特別SIMで始めよう
SORACOM でわかるIoT

特別付録
SIMカード
SORACOM Air
SD Special Version

Special Feature 1

Vim [実戦] 投入

コード編集の高速化から GitHub連携まで

Special Feature 2

2年ぶりのLTS
安定の Ubuntu 16.04 の新機能

一般記事

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

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。

巻頭特集

特別SIMで始めよう! SORACOM Airでわかる IoT

本誌特別付録 SORACOM Air SD Special Version利用上の注意

- 本SIMカードを利用する際は、本誌180ページからの「SORACOM Airサービス契約約款」を精読し了解いただき、各種通信環境を準備したうえで使用してください。
- 本SIMカードは、株式会社ソラコムのデータ通信サービス「SORACOM Air」専用です。それ以外は使用できません。
- 本SIMカードの利用には、株式会社ソラコムのWebサイトにて登録手続きが必要です。本SIMカードの**登録期限は2016年5月18日まで**です。登録手続きは、登録期限までに行ってください。登録期限が切れたSIMは、使用できません。
- 本SIMカードの登録には、クレジットカードが必要です。
- 本SIMカードおよびSORACOM Airについての詳細は、https://soracom.jp/services/air/を参照ください。
- 本SIMカードについての動作不良・不具合・技術的な質問、通信料金などは株式会社ソラコムのサポートサイトに問い合わせください。https://soracom.jp/contact/
- 本SIMカードを使用したことによるいかなる損害にも株式会社技術評論社および株式会社ソラコムは責任を負いません。
- 万一、本SIMカードが物理的に破損していた場合には良品と交換しますので、その特別付録SIMカードを下記までお送りください。その際、「交換希望」ということと、トラブルの概要をお書き添えくださいますよう、お願いいたします。トラブルを確認したうえで、良品と交換させていただきます。本誌購入時点での物理的な破損以外の質問は、下記ソラコムサポートサイトでお問い合わせください。

〒162-0846
東京都新宿区市谷左内町21-13
株式会社技術評論社　雑誌編集部ソフトウェアデザイン編集部
FAX 03-3513-6179　Mail sd@gihyo.co.jp

SIMカード取り扱い上の注意

- 本SIMカードに物理的に強い力を加えたり、水のような液体をかけたり浸したりしないでください。故障の原因になります。
- 高温多湿、直射日光のあたる場所で保管をしないでください。また車のダッシュボードやストーブの上など極端な高温の場所に放置しないでください。
- 本SIMカードの金属部分・金属接点部分には、なるべく触れないでください。また、金属部分を傷つけたり、ペンなどで書き込んだりしないでください。データの損失や故障の原因になります。
- 小さいお子様や乳幼児の手の届かないところに保管ください。

本SIMカードについての問い合わせ

株式会社ソラコム　SORACOMユーザーコンソール内サポートサイト(ユーザー登録が必要)
https://console.soracom.io/

第1章

さっそく使ってみよう SORACOM Air SD Special Versionの使い方

Author 平 愛美（たいら まなみ） Linux女子部 Twitter @mana_cat

SORACOM Airとは

SORACOM Airはソラコムが提供するIoT向けのSIMカードです。いわゆるMVNOです。NTTドコモの3G/LTE回線を利用しているため、日本全国だいたい人の住んでいるところであれば市街地でも山間部でもつながります。SORACOM Airはダウンロードよりもアップロード、日中よりも深夜時間帯の通信料金が安価に設定されています。ダウンロードをほとんど行わずにセンサーデバイスから生成される小さなデータを定期的にアップロードし続けたり、昼間に貯め込んだデータを夜間に一括でアップロードしたりするようなIoTデバイスに適した、1MBあたり0.2円～の従量制の通信料金設定になっています。回線を維持するための料金は1日あたり10円（SMS付きSIMだと1日あたり15円）、月額300円で1回線を維持できます。

SORACOM Airのアクティベーション方法

付録のSIMカード「SORACOM Air SD Special Version」は、利用する前にSORACOMアカウントの作成と、SIMカードのアクティベーション作業が必要です。アクティベーション期限が本誌発売日より1ヵ月以内となっており、アクティベーション作業は2016年5月18日までに行う必要があります。また、利用に際してクレジットカードの登録が必要となります。法人契約の場合、ソラコムに直接問い合わせると請求書払いも可能とのことです。

これからSORACOMアカウント作成とアクティベーション方法について説明していきます。

・まずは次のURLにアクセスし、SORACOMアカウントの作成を行います
https://console.soracom.io/#/signup
・メールアドレスとパスワードを入力する（図1）と、本人確認のメールが飛んできます
・本人確認のメール中に含まれるワンタイムURLに1時間以内にアクセスしてください
・SORACOMのユーザーコンソールのログイン画面（図2）に飛びますのでログインします
・ポップアップメッセージの「利用開始」をクリックします
・右上の通知メッセージの「今すぐ設定してください。」（図3）をクリックし、支払方法を登録します
・「新しいクレジットカードを登録」をクリック

▼図1 SORACOMアカウントの作成

▼図2 SORACOMアカウントにログイン

します
・クレジットカードの情報を登録します(図4)
・次にタブから「SIM管理」をクリックし、SIMカードを登録します
・「SIM登録」の青いボタンをクリックします
・登録にあたってはSORACOM Air SD Special VersionのSIMカードの裏面に記載されている15桁のIMSI番号と5桁のパスコードが必要です。名前は空白でも結構です。あとでわかりやすい名前を付けられます(図5)
・SIMカードの登録が成功したら、「終了して元の画面に戻る」をクリックします
・最後に使用開始処理を行います。対象となるSIMカードのチェックボックスをクリックし、「操作」→「使用開始」を選びます(図6)
・確認画面で「ステータスを変更する」をクリックします
・あとはSIMカードを好きなデバイスに差し込んでお使いください。次ページではRaspberry Piから使う方法を紹介します

SORACOM Air設定情報

NTTドコモもしくはSIMロックフリーのスマホなどで設定したい場合には、表1の項目を指定すると手動設定が行えます。

▼図3　右上のポップアップメッセージをクリック

▼図6　対象のSIMをチェックし「操作」→「使用開始」を選択

▼表1　手動設定のための設定情報

項目	設定
APN	soracom.io
ユーザ名	sora
パスワード	sora
認証方式	PAP
PDP Type	IP

※iPhoneやiPadの場合はAPNの設定ができないため、ソラコムが提供している構成プロファイルをインストールする必要があります。機器をWi-Fiなどでつなぎ、「https://soracom.jp/start/」よりダウンロードしてください。

▼図4　クレジットカードの情報を登録

▼図5　IMSI番号とパスコードを入力

接続方法

SORACOM AirはNTTドコモ系のMVNO回線ですので、NTTドコモが販売していたスマートフォン、Wi-Fiルータ、USBモデムなどで利用可能です。もちろんSIMロックフリーの端末でも利用できます。

NTTドコモで販売されていたLG L-02Cなどが好んで使われていますが、ヤフオク！や秋葉原の中古市場で入手する必要があります。今回は入手が容易で、かつ新品で購入できる富士ソフト㈱のFS01BUと、メカトラックス㈱の3GPIによる接続方法を紹介します。

FS01BUの接続方法

FS01BUはUSBドングルタイプの3Gモデム（**写真1**）で、PCやRaspberry PiのUSBポートに接続して使用できます[注1]。

FS01BUを使う場合、Raspberry PiのOSからダイヤルアップすることでインターネットに接続できます。OSがRaspbianなどの場合、wvdialというコマンドラインからダイヤルアップするツールがあります。今回はwvdialで使うまでの手順を紹介します。

wvdialのインストール

コマンドラインで次のとおり実行します。

```
Debian/Raspbianの場合
# apt-get install wvdial
Fedora/Pidoraの場合
# yum install wvdial
```

FS01BUの認識

FS01BUを差し込むと、次のようなデバイスがlsusbコマンドで確認できます。

▼写真1　FS01BU

▼リスト1　/etc/wvdial.conf（FS01BUの場合）

```
[Dialer Defaults]
Init1 = ATZ
Init2 = AT+CGDCONT=1,"IP","soracom.io"
Init3 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
Dial Attempts = 3
Stupid Mode = yes
Modem Type = Analog Modem
Phone = *99***1#
Dial Command = ATD
New PPPD = yes
Modem = /dev/ttyUSB1
Baud = 460800
ISDN = 0
Username = sora
Password = sora
Carrier Check = no
Auto DNS = 1
Check Def Route = 1
```

```
# lsusb
Bus 001 Device 005: ID 1c9e:6801 OMEGA TECHNOLOGY
```

このデバイスは、/dev/ttyUSB1に接続されたUSBシリアルポートとして認識します。

次に/etc/wvdial.confを編集します（**リスト1**）。wvdialはもともと電話回線でインターネット接続をしていたころのツールで、3Gモデムを利用する場合にはAPNを指定する必要があるため、ATコマンドと呼ばれる制御コマンドを指定する必要があります。ユーザ名とパスワードはsoraで共通となっているので、**リスト1**のまま入力してかまいません。

wvdialで接続確認

rootユーザでwvdialコマンドを実行します（**図**

注1） 現在、FS01BUはソラコムの直販サイト（SORACOMのユーザーコンソールにログインしたあとの「発注」をクリックすると出てきます）で、8,500円のところキャンペーン特価で6,900円にて購入可能です。

▼図7　wvdialコマンドを実行

```
# wvdial
--> WvDial: Internet dialer version 1.61
--> Initializing modem.
--> Sending: ATZ
OK
--> Sending: AT+CGDCONT=1,"IP","soracom.io"
AT+CGDCONT=1,"IP","soracom.io"
OK
--> Sending: ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
OK
--> Modem initialized.
--> Sending: ATD*99***1#
--> Waiting for carrier.
ATD*99***1#
CONNECT 14400000
--> Carrier detected.  Starting PPP immediately.
--> Starting pppd at Sat Mar 12 18:38:32 2016
--> Pid of pppd: 1315
--> Using interface ppp0
--> pppd: ???v
--> pppd: ???v
--> pppd: ???v
--> pppd: ???v
--> pppd: ???v
--> local  IP address 10.164.181.156
--> pppd: ???v
--> remote IP address 10.64.64.64
--> pppd: ???v
--> primary   DNS address 100.127.0.53
--> pppd: ???v
--> secondary DNS address 100.127.1.53
--> pppd: ???v
```

7)[注2]。

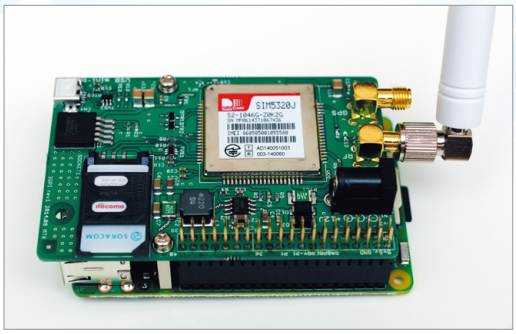
3GPIの接続方法

3GPIはRaspberry PiのGPIO 40pinに接続するタイプで本体上部にスタックできます(**写真2**)。GPIO 40pinは制御用に利用し、Raspberry PiとはUSBケーブルで接続します。3GPIの接続設定は、FS01BUよりも簡単でNetworkManagerでダイヤルアップ可能です。コマンドラインから設定や接続制御を行う場合、nmcliコマンドを利用します。**図8**を実行すると自動的に接続されるので、その後に接続状態を確認します。

▼図8　nmcliコマンドを実行

```
# nmcli connection add type gsm ifname "*" con-name soracom apn soracom.io user sora password sora
```

▼写真2　Raspberry Piと接続した3GPI



```
# nmcli connection show
```

切断方法

次のように実行すると切断できます。

```
# nmcli connection down soracom
```

SIMカードアダプターの使い方

本誌に付録するSORACOM Airは現在市販されているスマートフォンと同じNano SIMサイズのものです。先に紹介したFS01BUや3GPIのSIMスロットなどは標準SIMサイズですので、そのまま差し込むことができません。

そこでNanoサイズから標準SIMサイズに変換して大きくする必要があります。家電量販店などでもSIMカードアダプターが500円程度で販売されています。筆者の近所ではローソンストア100で売っていました。

SIMカードアダプターは、SIMスロットに差し込む際に外れてしまうとSIMカードスロットを破損させてしまうこともありますので、取り扱いには十分ご注意ください。心配な場合には標準SIMサイズのSORACOM Airも販売しているので検討すると良いでしょう。SD

注2）NetworkManagerサービスが稼働しているとwvdialと干渉するため、事前にNetworkManagerサービスを無効化しておきましょう。

第2章

おうちで楽しむ家庭内IoT

Raspberry Pi+クラウドでこんなことができる!

Author 平 愛美(たいら まなみ) Linux女子部 Twitter @mana_cat

家庭内IoTのススメ

第1章では、Raspberry PiにてSORACOM Airを使う方法を紹介しました。本章では、小規模で簡単に楽しめる我が家の家庭内IoTの活用例について紹介しましょう。Internet of Things、つまりモノがインターネットにつながっていることが重要です。プロトコルとしてはHTTPやMQTT(MQ Telemetry Transport)などがよく使われますが、何を使ってもかまいません。今回は、育児支援向け温湿度管理システム「IoTスーパーこまち」と「SORACOM IoTクリスマスツリー」を紹介していきます。

私が電子工作を始めたきっかけ

きっかけは些細な出来事でした。夫が海外出張中に、プラレールをリモコンで操作できる無線コントローラが壊れてしまって、4歳の長男が私に対してこのように言いました。

「ママは直せないよね。パパじゃないと直せないよね……パパが居なくて、寂しい……」

私は「そんなことないよ。大丈夫！ ママだって直せるよ」と息子に約束しました。そして、子どもたちを寝かしつけてから夜な夜な電子工作をするようになりました。はんだごてを手に取った母ちゃんは息子との約束を果たすために、プラレールのコントローラ自作を目指します。

IoTスーパーこまち

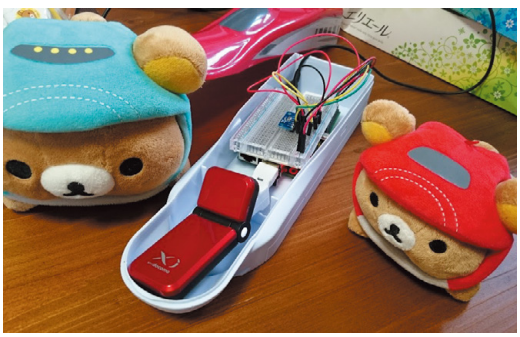
IoTスーパーこまち(**写真1**)は、いわゆるTwitterボットです。単なるボットではなく、温湿度センサーのBME280からI²C経由で温度と湿度を読み込んで、定期的にTwitter API経由でツイートします。よって、このTwitterボットをフォローしていれば、寝室の温度と湿度がタイムラインに流れてきます。家庭内IoTを長く続けるコツとしては、日ごろ定期的にチェックするシステムに結果を混ぜ込む、結果を確認するためだけにクライアントアプリを作らない、管理負担となるシステム／サーバを極力増やさないことがポイントです。

SORACOM IoT クリスマスツリー

SORACOM IoTクリスマスツリー(**写真2**)は、SORACOM Advent Calendar 2015の企画で期間限定で公開したシステムで、クリスマスイブに公開予定だったのでコンセプト先行で着手しました。

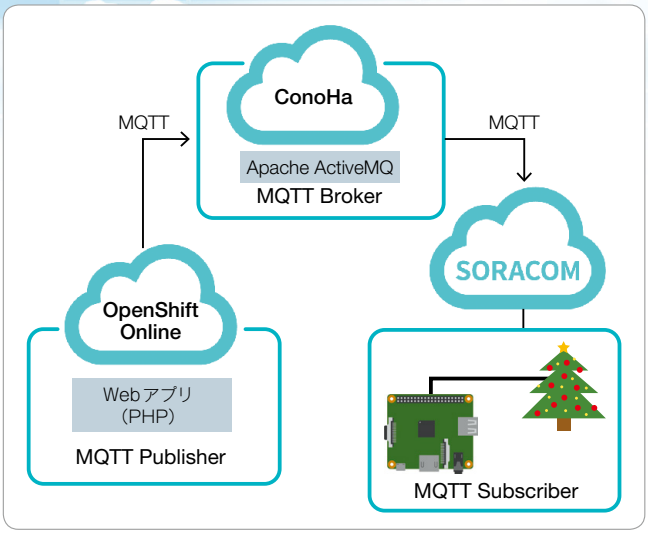
秋葉原でLEDイルミネーションキットを購入し、私が回路の配線を行い、息子が飾り付けを担当しました。そして、夫がMQTTプロトコルを使った本格的なアプリをプログラミングしてくれました。玄関のクリスマスツリーに設置したRaspberry PiをSubscriber、OpenShift Online上にデプロイしたアプリをPublisherとして、MQTT BrokerはApache ActiveMQを使っています。MQTT Brokerはグローバル IPアドレスが必要だったので、GMOインター

▼写真1 IoTスーパーこまち



▼写真2　SORACOM IoTクリスマスツリー

▼図1　システム構成



ネットのConoHa上で動かしました(**図1**)。

実際に使っていたソースコードとAnsible Playbookを GitHubで公開しています[注1]。MQTT Brokerのサーバ名をexample.comから任意のFQDNに書き換えて、デプロイすればお好きな環境で簡単に動かせます。

次回作 IoTプラレール

電子工作を初めて数ヵ月で、はてなブックマークのホットエントリーに取り上げられ、雑誌やムック本の執筆依頼が増えてしまい、そして次男の保育園活動と忙しさにかまけて、本来の目的が後回しになっていました。

最近、ついに「IoTプラレールはマダー？」と夫から突かれてしまい、Raspberry Pi Zeroを使って、インターネット経由で制御できるプラレールを鋭意製作中です。完成したら、どこかで公開したいと思います。お楽しみに！SD

column

IoTをささえるプロトコルMQTT

MQTTはPub-Sub(Publish-Subscribe)型モデルと呼ばれるシステム構成を組みます。メッセージの出版側をPublisherと呼び、購読側をSubscriberと呼びます。MQTT以外のPub-Sub型モデルのプロトコルには、AMQP、STOMPなどがあります。シンプルな実装となっているため、どのプロトコルを使っても同じようなことはできますが、MQTTのサポートを表明しているデバイスメーカーや、各OS、言語環境におけるライブラリの実装度合いが整っている点が魅力です。

互いにPublisherとSubscriberの両方を兼ねることも可能です。MQTTのPublisherとSubscriberの間は、直接通信するわけではなく、MQTT Brokerと呼ばれる仲介サーバを介して通信を行います。このようなしくみを取るため、Pub-Subとして見ると1:1、1:N、N:1、N:Mの双方向通信が可能となります。また、PublisherとSubscriberはグローバルIPアドレスを持つ必要がなく、NAT配下に存在していても、双方向通信が可能です。よって、SORACOM Airのようにデバイス側にプライベートIPアドレスしか付与されないタイプの回線であってもかまいません。Brokerさえグローバル IPアドレスが振られていれば、距離的に遠く離れている場所であっても通信できます。MQTT Brokerの実装としては、Apache ActiveMQ、RabbitMQ、Mosquitto Broker、Paho MQTT Brokerなどがあります。

注1）https://github.com/htaira/mqtt-xmas-tree-subscriber
https://github.com/htaira/mqtt-xmas-tree-webapp

第3章

APIを使ってSORACOMの便利さを体感!

“SORACOM Air”メタデータサービスで通信をコントロールしてみよう

Author 小熊 崇（おぐま たかし） ㈱ソラコム シニアソフトウェアエンジニア

IoTプラットフォーム“SORACOM”とは

本章では、SORACOM最大の特徴の1つである「APIによる通信のコントロール」を体感いただきます。クーポンも付いていますのでお試しください。

SORACOMの提供するサービス

SORACOMには、本誌付録のデータ通信サービス「SORACOM Air（エアー）」のほかにも、全部で6つのサービスがあります。データ転送支援サービスの「SORACOM Beam（ビーム）」、プライベート接続サービスの「SORACOM Canal（カナル）」、専用線接続サービスの「SORACOM Direct（ダイレクト）」、SIM認証サービスの「SORACOM Endorse（エンドース）」、クラウドリソースアダプタサービスの「SORACOM Funnel（ファネル）」。いずれもクラウド連携をスムーズにセキュアに利用する付加価値サービスです。

本章では、そのうちのSORACOM Airの API経由の操作を中心に解説します。そして、SORACOM Endorseを利用した本誌限定クーポンを読者全員にプレゼントします。SORACOM Beamほかのサービスにご興味をお持ちの方はソラコムWebサイト（https://campaign.soracom.jp/sd201605/）に試せる資料もご用意しました。では、さっそく使ってみましょう。

モバイルデータ通信サービス“SORACOM Air”

SORACOM AirはIoTデバイス向けデータ通信の基本となるサービスで、データ通信用のSIMカードを提供します。データ通信の開始（アクティベート）や休止、通信速度の変更、現在のデータ使用量の確認、データ使用量や状態の変化に応じたイベント実行など、さまざまな処理を、ユーザーコンソールもしくはAPIで行うことができます。

“SORACOM Air”で通信してみましょう

では、実際にデータ通信をしてみましょう[注1]。スマートフォンやタブレットの場合は、デバイスのWi-Fiを OFFにして、モバイルデータ通信に切り替わっていることを確認してからブラウザなどで好きなアドレスにアクセスしてみましょう。

無事インターネットに通信できていることが確認できたら、本章のこのあとのパートでは主にPCを使っていろいろ試していきますので、PCとテザリングしてPC側でもインターネット通信ができるかどうかを試してみてください。

Wi-Fiルータ やUSB 3G/LTEモデムをご利用の場合は、最初からPCと接続してお試しください。

column

SIM認証サービス“SORACOM Endorse”でクーポン取得

ここでは本誌限定で特別にご用意した、データ通信料金に使えるクーポンを取得してみましょう。クーポン取得には、2016年1月に開始した新サービスSORACOM Endorseを利用します。

クーポン取得は次のサイトから手順をご覧ください。

https://campaign.soracom.jp/sd201605/

取得したクーポンは、ユーザーコンソールのアカウント名のボタンを押して、「クーポン登録」から登録できます。

注1）SORCOM Airを利用するために必要な、SORACOMアカウントの作成と、SIMカードのアクティベーションの手順については、第1章を参照のこと。

“SORACOM Air” メタデータサービスの利用

SORACOM Airで通信ができるようになっただけでは、ほかのSIMとの違いはあまりありません。SORACOM最大の特徴の1つである「APIによる通信のコントロール」をさっそく試してみましょう。

ただ、SORACOM APIを利用するには、認証処理などの手順が少し必要です。初めて試していただく方向けに、ここではより簡単に利用できるSORACOM Airのメタデータサービスを使って、APIの便利さを体感していただきます。

メタデータサービスとは

メタデータサービスとは、SORACOM AirのSIMを使って通信している場合に限り、その通信しているSIM自身のAPIを認証不要で呼び出せるようになるサービスです(**図1**)。

SIMを用いてモバイルデータ通信を行っている場合、SIMを搭載したデバイスとドコモの設備との間で強力な相互認証が行われており、ほかのSIMのIDを詐称したりすることが事実上不可能となっています。そこで、そのような強力な認証を利用し、SIM自身に関する操作であれば簡単かつ安全にご利用いただくことができます。これがメタデータサービスです。

メタデータサービスを有効にする

メタデータサービスを使うには、明示的に本サービスを有効にする必要があります。有効／無効はグループ単位で設定します。つまり、有効に設定されているグループに属しているSIMはすべて本サービスを利用可能ですし、そうでないSIMは本サービスを利用できません。

また、メタデータサービスはデフォルトでは読み取り専用です。SIM自身の情報を読み取ることはできますが、変更はできません。たとえば、現在の速度クラスの設定を読み取ることはできますが、速度をより速いクラスやより遅いクラスへ変更することはできません。変更を可能にするには、これも明示的に読み書き可能に設定する必要があります。

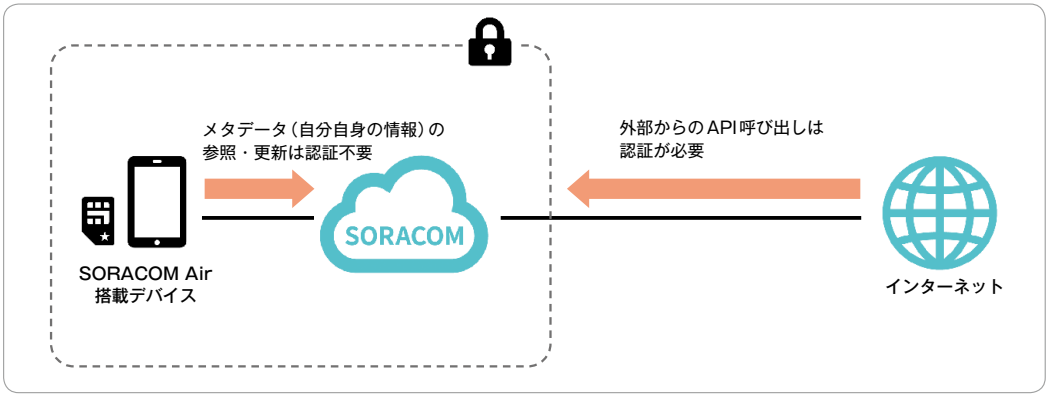
それでは、ユーザーコンソールでメタデータサービスを有効化し、読み書きも可能としてみましょう。おおまかな手順は次のとおりです。

①グループを作成し、そのグループの設定でメタデータサービスを有効化する
②SIMをそのグループに所属させる

それでは、まずグループを作成してメタデータサービスを有効化しましょう。

・ユーザーコンソールにログインし、画面上部のナビゲーションから[グループ]を選択します

▼図1　メタデータサービス



・[＋追加]ボタンを押し、グループを作成します。作成するグループ名は何でもかまいませんが、今回は「software-design-201605」としてみました(図2)
・グループを作成したら、グループの一覧の中にそのグループが現れますので、それをクリックして詳細画面に入ります
・[基本設定]タブの[SORACOM Air設定]を開き(図3)、[メタデータサービス設定]のスイッチをONにします。また今回は、メタデータを利用した設定変更も試すので[読み取り専用]のチェックボックスをOFFにします。[保存]ボタンを押して設定変更を反映します
・次に、今作成したグループにSIMを所属させます。画面上部のナビゲーションから[SIM管理]を選択します
・SIM一覧画面で、先ほど登録した本誌付属のSIMを選択します
・[操作]ボタンを押して[所属グループ変更]を選択します
・「新しい所属グループ」のドロップダウンメニューから、先ほど作成したグループを選択し(図4)、「グループ変更」ボタンを押します

SIM情報の確認

準備が整いましたので、メタデータサービスを使ってSIMの情報を取得してみましょう。

SORACOM Airを入れたデバイス自身、もしくはそのデバイスを使ってテザリングなどでインターネットに接続しているPCで、図5のコマンドを実行してみてください。JSON形式で図6のよ

▼図2 グループを作成する

▼図3 メタデータサービスをONにする

▼図4 SIMをグループに所属させる

▼図5 SIM情報を取得する

```
$ curl -s http://metadata.soracom.io/v1/subscriber
```

▼図6 取得したSIM情報の例

```
{"imsi":"44010xxxxxxxxx","msisdn":"8180xxxxxxxx","ipAddress":"10.xxx.xx.xx","apn":"soracom.io","type":"s1.fast", (略) }
```

※IMSIやMSISDNなどは、読者のSIMの情報が表示される

▼図7　jqコマンドを使って速度クラスの設定情報だけを表示させる

```
$ curl -s http://metadata.soracom.io/v1/subscriber | jq .speedClass
"s1.standard"
```

▼図8　速度クラスをs1.fastに変更する

```
$ curl -sX POST -d '{"speedClass":"s1.fast"}' -H 'Content-Type: application/json' \
 http://metadata.soracom.io/v1/subscriber/update_speed_class
```

▼図9　ファイルをダウンロードする

```
$ curl -O https://campaign.soracom.jp/sd201605/soracom.pdf
```

▼図10　速度クラスをs1.slowに変更する

```
$ curl -sX POST -d '{"speedClass":"s1.slow"}' -H 'Content-Type: application/json' \
http://metadata.soracom.io/v1/subscriber/update_speed_class
```

うな読者のSIM情報が表示されれば成功です。

jqコマンドが利用できる環境であれば、データを整形して表示したり、必要な部分だけ抜き出したりすることも可能です。たとえば、現在の速度クラスの設定は**図7**のようにして確認します。

通信速度を変更する

次に、メタデータサービスを使ってSIMの設定変更を行ってみましょう。

まず、s1.fastという速度クラスに変更します(**図8**)。この速度クラスではアップロード・ダウンロードともに2Mbpsの速度が出ます。

この状態で本当に速度変更が反映されたか、ダウンロード速度を判定してみましょう。**図9**のようにダウンロードを実行します。ダウンロードが完了したあとの“Average Speed Dload”の下に現れている数は、単位がバイトですので、これを8倍してみてください。電波状況の良い場所で、ブラウザを閉じるなどしてほかの通信が発生していないような状況では、およそ2Mbps弱と設定どおりになっているはずです。

次に速度をs1.slowに変更してみましょう(**図10**)。この速度クラスではアップロード・ダウンロードともに128kbpsに制限されます。

この状態で本当に速度変更が反映されたか、再度ダウンロード速度を判定してみましょう。**図9**と同じコマンドでダウンロードを行います。

先ほどよりダウンロードに時間がかかるようになったのではないでしょうか。今度はおよそ128kbpsとなったはずです。メタデータを利用したAPIでのコントロールは以上です。

終わりに

SORACOMは、SORACOM Airだけではありません。たとえば、SORACOM Beamは、IoTデバイスにかかる暗号化などの高負荷処理や接続先の設定を、クラウドにオフロードできるサービスです。Beamを利用することによって、クラウドを介していつでも、どこからでも、簡単にIoTデバイスを管理できます。大量のデバイスを直接設定する必要はありません。

SORACOMは、IoTの通信を提供するだけではなく、IoTシステムをスピーディにセキュアに構築するためのサービスも提供しています。SORACOM Beam以降のサービスを試せる資料を特設サイト(https://campaign.soracom.jp/sd201605/)にご用意しました。引き続きSORACOMでIoT通信をお試しください。SD

第4章

myThingsとSORACOM
——myThingsを使ってノンプログラミングでSORACOMをより便利に使おう

Author 山本 学(やまもと まなぶ)ヤフー㈱ スマートデバイス推進本部 大阪開発室

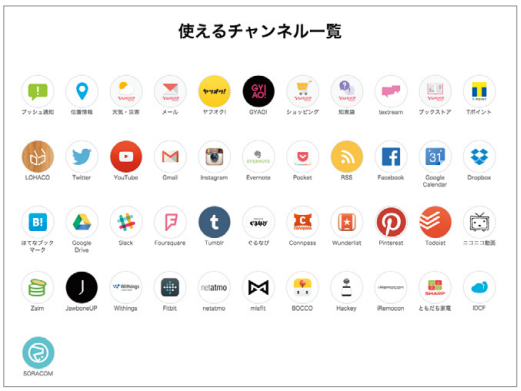
myThingsとは

myThingsはヤフー株式会社が構想しているWebサービスや市販のIoTデバイス、企業、人、街といった世の中のありとあらゆるものを繋げることを目指したプラットフォームです。そして、その「ありとあらゆるもの」のうち、WebサービスとIoTデバイスにフォーカスして、それらが持つ機能をユーザ自身で組み合わせて便利に使うことができるmyThingsアプリを2015年7月に発表しました。現在iOS、Android向けに提供されています(**図1**)。

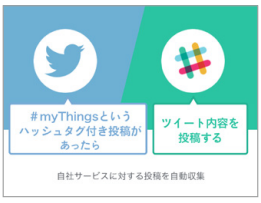
▼図1 myThingsアプリ

myThingsアプリで

▼図2 myThingsアプリが連携可能なサービス(2016年3月15日時点)



▼図4 TwitterとSlackの組み合わせ



▼図5 JawboneUPとiRemoconの組み合わせ

は「○○が××だったら、△△を□□する」という形式で、異なるサービス同士を連携させることができます。連携可能なサービスは45個あります(**図2**)。

また、myThingsでは、それら連携可能サービスのことを**チャンネル**、連携のきっかけを**トリガー**、連携時に実行する機能を**アクション**、トリガーとアクションのセットを**組み合わせ**と呼んでいます(**図3**)。たとえば**図4**や**図5**のような組み合わせを作ることができます。また、IDCFというチャンネルを活用することで(**図6**)、自分で工作したデバイスもmyThingsと連携させることができます(**図7**)。

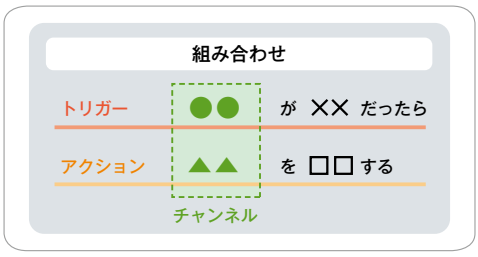
SORACOMチャンネル

2016年2月25日、myThingsの対応チャンネルにSORACOMが追加されました。SORACOMチャンネルを使えば、ノンプログラミングでSORACOM Airと他サービスを連携させることができます。ここではSORACOMチャンネルで利用可能なトリガー、アクションとその活用例を紹介します。

SORACOMチャンネルのトリガー

・特定のSIMの通信量が指定した値を上回ったらSORACOMユーザコンソールの監視設定でも通信量に閾値(いき)を設けてメール通知を行うことができますが、このトリガーを利用す

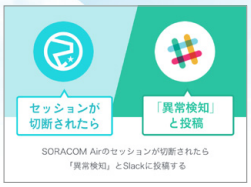
▼図3 myThingsアプリの用語解説



▼図6　位置情報とIDCFの組み合わせ

▼図8　SORACOMとプッシュ通知の組み合わせ

▼図9　SORACOMとSlackの組み合わせ



▼図10　SORACOMとYahoo!メールの組み合わせ

▼図7　家族の居場所を時計のようにマッピングできる自作デバイス

▼図11　JawboneUPとSORACOMの組み合わせ

▼図12　SORACOM同士の組み合わせ

るとSlackやプッシュ通知などメール以外の手段でも通知が可能になります(図8)。

・**特定のSIMが切断したとき／特定のSIMが接続したとき**

SORACOM Airのセッション状態をトリガーにすることができます。自作デバイスでこのトリガーを活用すれば死活監視に活用することができそうです(図9)。

・**指定した請求予定額を上回ったら**

通信量だけではなく、請求予定額に閾値を設けてトリガーにできます。組み合わせの設定次第では請求予定額を日々チェックできます(図10)。

SORACOMチャンネルのアクション

・**特定のSIMの速度設定を変更する**

何らかのトリガーが発火した際に、指定したSIMの速度設定を変更することができます。直接SORACOM APIを活用することなく、メールの件名に特定文字列があった場合に速度変更を行ったり、位置情報チャンネルと組み合わせて自宅や職場など自分のいる場所に応じて速度を変更する、Jawboneチャンネルと組み合わせて睡眠／起床に合わせて速度を変更するといったことが可能です(図11)。

・**特定のSIMを有効化する／特定のSIMを無効化する**

何らかのトリガーが発火した際に、指定したSIMの状態を変更することができます。決まった金額内で自作デバイスを運用した際など、前述の「特定のSIMの通信量が指定した値を上回ったら」トリガーや、「指定した請求予定額を上回ったら」と相性が良さそうです(図12)。

まとめ

以上、SORACOMチャンネルでできること、そしてその活用例を紹介しました。SORACOMには便利なAPIがたくさん用意されており、それらを使いこなせばさまざまなことが可能になります。ただ、直接APIを使うほどではない場面や、APIを実行する条件が複雑な場面(位置情報と連動したい、睡眠／起床といった人の状況と連動したいなど)では、myThingsを活用するとあっさりそれが実現できる場合があります。ぜひmyThingsを活用してノンプログラミングでSORACOM Airを便利に使いこなしてみましょう！SD

第5章

SORACOMとさくらのIoT Platform

モノのインターネットとモバイル通信の未来

Author 松本 直人（まつもと なおと） さくらインターネット㈱ さくらインターネット研究所 上級研究員

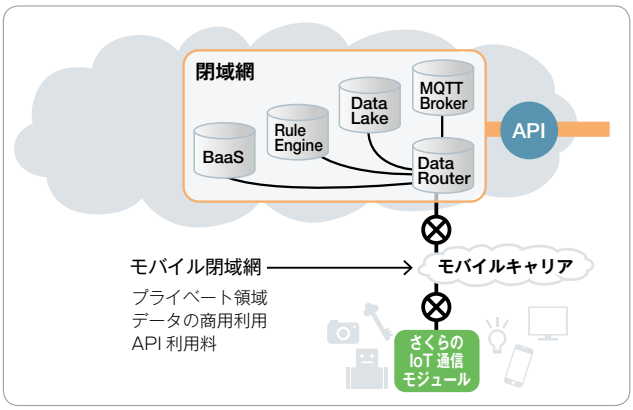
はじまりはモバイル閉域網

2016年2月8日に新サービスとなる「さくらのIoT Platform」の記者発表が行われました。この発表で述べられたことは、今回のシステムはデータセンターのクラウド環境とモバイル閉域網が直結した垂直統合型のしくみです（**図1**）。

古くはトヨタ自動車㈱のテレマティクスサービスなどでも使われているモバイル閉域網ですが、今回さくらインターネットでは、すでにIoT向けモバイル閉域網サービスを手掛けていた㈱ソラコムからの全面協力のもとαサービスの立ち上げることができました。

モノのインターネット（IoT: Internet of Things）で使われるセンサーやマイコンの多くは非力な性能と制限されたメモリ空間のため制約条件が多く、保護された安心安全なネットワーク環境で管理運用する必要があると考えられています。モバイル閉域網は広大なインターネットからの攻撃を受けることなく、またモバイル閉域網内部からの端末同士への攻撃も抑止できるしくみを持っています。今回さくらインターネットでは、これら優位点を加味したうえで、モバイル閉域網とデータセンターのクラウド環境を直結したシステム環境を構築するにいたっています。多くの企業が取り組みはじめたIoT向けサービスプラットフォームですが、世界でも大きく技術的な動きがみられています。ここからは先日バルセロナで開催されたMWC（Mobile World Congress）2016で発表された、モバイル通信の技術的な地殻変動についてみていきましょう。

▼図1 データセンターのクラウド環境に直結するモバイル閉域網



▼図2 eSIM対応製品を設定するまでの流れ

QCコード（モバイル・キャリア発行）
↓
スマートフォンアプリケーション
↓ Bluetooth
eSIM掲載スマートウォッチ
Samsung Gear S2 with 3G
↓
選択したモバイルキャリア3G/LTE

既存スマートフォンに依存するが柔軟にモバイルキャリア選択できるしくみ

MWC2016で起こったモバイル通信の地殻変動

筆者は、2年前のMWC2014にも研究調査活動で参加しました。この会場で描かれていた未来の1つに半導体チップを使ったeSIMと呼ばれる構想がありました。今回のMWC2016では、このeSIMの最初の商用製品がリリースされたのです。eSIMに対応する製品は、スマートフォンの助けを借りて通信キャリアのプロファイルをインターネット経由で入手し、eSIM内に書き込みます。その後eSIM対応製品は単体で3Gモバイル通信網へ接続することができるようになり、普段

使いのスマートフォンと同じように、その小さなデバイスからインターネットを経由してデータ通信ができるようになります(**図2**)。

これにより、通信キャリアはプロファイルが入ったSIMカードを事前に作成流通する手間がなくなり、またユーザにとっては発行されたeSIM向けのバウチャー(引換券)が入手できさえすれば、すぐにeSIM対応製品をその国や地域で利用できるようになるメリットがあります。eSIM対応はまだまだ始まったばかりのサービス領域ではありますが、モノのインターネットのように多数のデバイスが、多国間にまたがり流通することが想定されている状況を考えれば、とても面白い取り組みだと筆者は感じています。

低消費電力で低速データ通信に対応するLTE Cat.1の出現

モバイル環境の通信規格にも変化が生まれてきています。現在私たちが使っている3GやLTE回線には、それぞれ規格がありIoT対応のため、さまざまな規格が新たに生まれようとしています。その1つがMWC2016でも発表があったLTE Cat.1です。現在、LTE向けにCat.1、Cat.0、Cat.MそしてNB-IOT(Narrowband IoT)が検討段階にあり2017年導入を目指して作業が進んでいます。当然日本や各国において2017年以降に、チップを起こし、通信キャリア設備が対応し、サービス試験が行われ、通信モジュールの量産化と技術基準適合証明を受け、市販化が浸透して、やっと私たちの手元に届くことになります。筆者の感触としては、それはおそらく2018年か2019年になるのではと考えています(**図3**)。

現在私たちが使っているスマートフォンやモノのインターネット向けとして提供されている通信モジュールのほとんどがLTE Cat.4に属しているものらしく、非常に小さなデータを送受信するには、かなりオーバースペックであることは理解できます。それでも、モノのインターネットが普及拡大するためには、市場に種としてインターネットへ接続する、なにがしかの手段を広めていかなければいけません。「Connected」の状態になってはじめてモノのインターネットとしての最初の一歩を踏み出せるわけですから、これは当然といえば当然です。

日本国内のモバイル通信キャリアにおいては、これに加えて相互接続性試験(Inter-Operability Testing)と呼ばれる自社モバイル通信網とメーカー製品の相互接続性を確認する試験も行われており、これを通過した通信モジュールなどが推奨されてきた歴史があります。昨今SIMフリー端末や海外製品の増加などもありますが、電波を出す機器の取り扱いは、やはり各国の規制等もあり悩ましいことが多くあります(**図4**)。

免許不要の長寿命・長距離データ通信

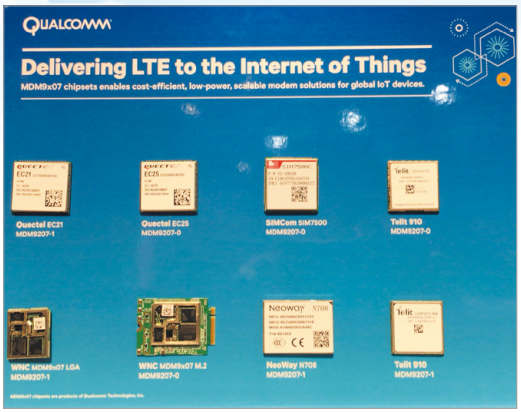
より低消費電力で長寿命を目指した規格としてLoRaWANがあります。LoRaはWide Area Networks for IoTを掲げるLow Power Wide Area

▼図3　新たに開発が進む低消費電力／低速データ通信向けモジュール

	LTE Cat.4	LTE Cat.1	LTE Cat.0	LTE Cat.M	NB IOT	LoRaWAN
Downlink	150Mbps	10Mbps	1Mbps	1Mbps	＜200Kbps	0.3-50Kbps
Uplink	50Mbps	5Mbps	1Mbps	1Mbps	＜200Kbps	0.3-50Kbps
導入予測	現在利用中	2016年	不明	2017年	2017年	2016年
利用用途	スマホ／自動車等		ウェアラブル等	ガス・電気等	ビル管理等	低消費／長寿命

出典：さくらインターネット株式会社／さくらインターネット研究所調べ(2016年03月)

▼図4 参考：Qualcommチップ搭載メーカー社製IoT向け通信モジュール一覧（技適マークは見られず）



Networkの規格であり“long range machine-to-machine connectivity”に名称の由来があります。前述のLTE Cat.1、Cat.0、Cat.M、NB-IOTはいずれもセルラー通信を前提としたものでしたが、LoRaWANは免許不要で長距離データ通信が行える通信規格として海外ではすでにArduino向けモジュールも売られています。データ通信速度は0.3～50kbpsと低速でも、そのエリアは数km単位に及び、ちょっとしたトランシーバー遊びをした世代にとっては、これほど面白いオモチャはないでしょう。

MWC2016でも、すでに製品化されたもしくは製品化目前のデバイスや基地局などが多数出展されており、バルセロナ市内に展開されたLoRaWAN対応デバイスからリアルタイムでセンサー情報が送信されマップに表示されているデモもありました（**図5**）。

こうした免許不要の長寿命・長距離データ通信には、まだまだ多くの異なる規格があり、自社独自規格も含めると、かなりの数があると筆者は感じています。日本国内においても920MHz帯がセンサーやスマートメーター、電力メーターであるHEMSなどに開放されており、空中線電力で1mW、20mWなど規制はあれど、その用途に期待は広がってきていたところです。大きな問題としては、こういった特定小電力向けに開放された電波は各国の周波数が異なっており、前述のLoRaWAN向け通信モジュールにも433MHz、868MHzが存在しています。今はまだ技適マークのついた日本向け製品はまだ存在していませんので、これからに期待したいところです。

まとめ

ここまでモバイル閉域網から始まり、免許不要の長寿命・長距離データ通信というIoTを取り囲む大きな潮流を見てきました。筆者はモノが「Connected」な状態でパソコンやスマホを使うように常に「I/O」を発生させる状態を作り出すためには、さまざまな企業の努力が必要だと考えています。

PHSの初期導入のころ、端末はゼロ円で配布していた過去を思い出すと、モノがしゃべりだす世界をしっかりと形作るには、それ相応のインフラ整備や考え方の転換が今後も重要になってくるだろうと感じています。2020年、500億のデバイスが全世界でつながる未来が近づいてくるといわれて2年。これからのIoTを取り巻く潮流に期待したいところです。SD

▼図5 免許不要の長寿命・長距離データ通信 LoRaWAN

5
2016
［目次］
Software Design

contents

Special Feature 1 第1特集

Vim [実戦]投入

コード編集の高速化からGitHub連携まで

Special Appendix 特別付録

SORACOM Air SD Special Version PRE-1

特別SIMで始めよう! SORACOMでわかるIoT

5
2016
［目次］
Software Design

Special Feature 2 第2特集

2年ぶりのLTS

安定のUbuntu 16.04の新機能

Extra Feature 一般記事

Special Report

àla carte アラカルト

contents

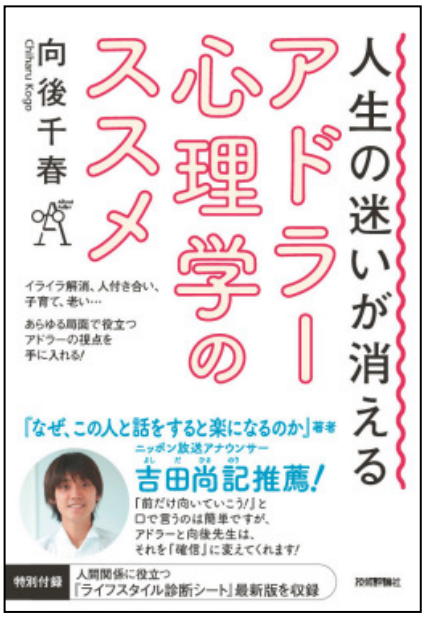
Column

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）
［表紙写真］
Akimasa Harada / 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（トップスタジオデザイン室）
＊伊勢 歩、横山 慎昌（BUCH+）
＊藤井 耕志（Re:D）
＊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。



ISBN978-4-7741-7980-3
四六判／160ページ
定価（本体1480円+税）

人生の迷いが消える
アドラー心理学のススメ

●向後千春 著

あなたの人生観を変えるアドラー心理学の視点を手に入れよう！著者の社会人向け人気講義をこの1冊に凝縮！
迷いをなくし、よりよい人生を送りたいすべての方へおすすめいたします。

●目次

ISBN978-4-7741-8074-8
A5判／240ページ
定価（本体1880円+税）

身につくベイズ統計学

●涌井良幸・涌井貞美 著

近年、爆発的に普及しているのがベイズ統計学です。数学、経済学、金融工学、情報科学、心理学など、幅広い分野で用いられています。迷惑メールのフィルタリングソフトやインターネットの検索エンジンの技術、機械学習などにも活かされており、日米ともに「知らないでは済まされない」強力な解析ツールです。本書では、こうしたベイズ統計の基本的な考え方や活用について、初学者向けテキストとして、ていねいに解説していきます。ベイズ統計学をこれから勉強したい大学生、ビジネスで実際に活用したい社会人に最適です。

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

Pocket Reference

小さくても、中身充実！

「あれ何だったっけ？」
「こんなことできないかな？」
というときに、すぐに調べられる
機能引きリファレンス。
軽くてハンディなボディに
密度の濃い内容がギュッと凝縮！
関連項目への参照ページもあって、
検索性もバツグン！

髙江賢 著／山田祥寛 監修
四六判／576ページ
定価（本体2680円+税）
ISBN978-4-7741-8030-4

山近慶一　著
四六判／688ページ
定価（本体2980円+税）
ISBN978-4-7741-8001-4

沓名亮典 著
四六判／608ページ
定価（本体2380円+税）
ISBN978-4-7741-7404-4

石田つばさ 著
四六判／448ページ
定価（本体2180円+税）
ISBN978-4-7741-4836-6

宮前竜也　著
四六判／224ページ
定価（本体2180円+税）
ISBN978-4-7741-7740-3

岡本隆史／武田健太郎／相良幸範 著
四六判／272ページ
定価（本体2480円+税）
ISBN978-4-7741-5184-7

朝井淳 著
四六判／640ページ
定価（本体1980円+税）
ISBN978-4-7741-3835-0

大垣靖男 著
四六判／648ページ
定価（本体2580円+税）
ISBN978-4-7741-7229-3

鶴長鎮一／馬場俊彰 著
四六判／400ページ
定価（本体2780円+税）
ISBN978-4-7741-7633-8

片渕彼富 著／山田祥寛 監修
四六判／512ページ
定価（本体2780円+税）
ISBN978-4-7741-7984-1

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

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

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）

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

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

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

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

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

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

第89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。

Eclipse Che

新しいクラウドIDE「Eclipse Che」

「Eclipse Che」は、統合開発環境「Eclipse」の提供元であるEclipse Foundationが開発しているクラウドベースの新しい開発環境です。Eclipse Cheでは、開発に必要となるさまざまなツールやプラグイン、ワークスペースがサーバ上に展開され、開発者はWebブラウザベースのIDE経由でそこに接続して開発を行います。

Eclipse Cheは“次世代のEclipse IDE”とうたわれてはいますが、Eclipseの次期バージョンがEclipse Cheになるというわけではなく、あくまでも独立した新しいIDEとして位置付けられています。従来のEclipseの次期バージョンは「Eclipse Neon」のコード名で開発が進められており、2016年6月にリリースされる予定です。

またEclipse Foundationが進めてきた別のプロジェクトに、Webブラウザで動作する開発環境「Orion」の開発がありますが、Eclipse CheはOrionとも異なります。Eclipse Cheはクラウドベースの IDEの開発を手がけるCodenvyの主導によって開発されたもので、Orionからはエディタ機能などの成果がEclipse Cheに取り込まれているとのことです。

Eclipse Cheのアーキテクチャ

Eclipse Cheでは、従来のEclipseとは異なり、ワークスペースにアプリケーションの実行環境（ランタイム）を内包するというアプローチをとっています。すなわち、ワークスペース自身がアプリケーションを実行したりデバッグしたりできる単独の“マシン”として動作するということです。これによってワークスペース単位でのライフサイクルの管理や、他の環境への移行などを容易に行えるというメリットが生まれます。このワークスペースの実現にはDockerが利用されており、通常のDockerコンテナと同様の手軽さで扱うことが可能です。

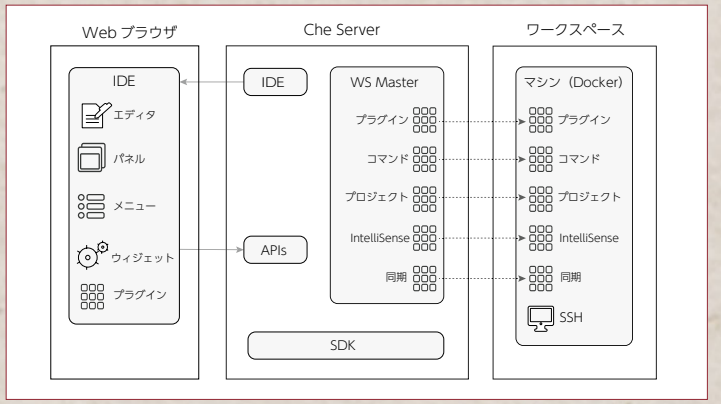
ワークスペースの管理機能やチームコラボレーション機能などはChe Serverによって提供されます。開発に使用するIDEやAPIもChe Serverにホストされます。Che Serverはローカルマシン上や自前のサーバにインストールする以外に、クラウド上にSaaS型のサービスとして展開することも可能です。開発チームのメンバーはChe Serverを介してワークスペースにアクセスしたり、他のメンバーとワークスペースを共有したりできます。

開発者が実際に使用する環境としては、Webブラウザベースの IDEが用意されています。このIDEはHTMLとJavaScriptおよびCSSによって作られているため、特定のブラウザに依存せずに利用できます。前述のようにエディタ部分にはOrionの成果が取り込まれており、コードのシンタックスチェックやオートコンプリートといった高度なサポート機能を備えています。IDE上で開発されたコードは、プロジェクトのワークスペースとして動作するDockerコンテナにデプロイされます。また、Eclipse Cheのワークスペースは、IDEを使う以外にもSSHを利用したアクセスにも対応しています。

上記に加えてEclipse Cheはプラグイン機能を備えており、各種プラグインによって拡張が可能なほか、独自のプラグインを開発するためのSDKも提供されています。

近年では、環境構築の手間を省く目的や、チーム開発との親和性の高さなどといった理由から、クラウドベースの統合開発環境に対する注目が高まっています。Eclipse Cheもそういった流れの中で生まれたものですが、ワークスペースをコンテナ化して可搬性を高めるというアプローチは極めて興味深いものと言えます。SD

▼図　Eclipse Cheのアーキテクチャ（公式サイトより）



Eclipse Che
https://eclipse.org/che/

DIGITAL GADGET vol.209

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

VR世界のユーザインターフェース

何度目かのVRブーム到来

最近注目のドローンレースではVRヘッドセットを装着し、ドローンから送られてくるカメラ映像で操縦しており、VR世界と現実の世界が融合したような環境になっています。これらVR（Virtual Reality）が盛り上がってきている1つの要因としては、旧来、数千万円規模の特殊な機材がなければ、体験、研究、制作ができなかった環境が、数十万円台、もしくは一般のスマートフォンを活用するなど、とても安価に実現できるようになったことが挙げられます。

期待のPlayStation VRは、映画などの映像を見るためのシネマティックモードを備え、2016年10月にリリース、399ドル（日本では税別4万4980円）と発表されました。性能に対しての期待値が高かった最新のOculus Riftがおおかたの予想以上に高額であったことからも、普及や浸透のためには魅力的なVRコンテンツが出そろうことと、VR機器の価格の両方が影響を与えてくるものと思われます。

一方、安価なダンボールでできたGoogle Cardboard、ハコスコなどのVR装置は、スマートフォンを活用し手軽に利用できつつも、継続して使ってもらうことが難しいと言われています。1回目の体験ではすごいと思っても、それを何度も体験したいと思うほどの要素や、魅力をもったVRコンテンツがまだまだ少ないのも1つの要因です。

VRコンテンツで多い系統は、スリラー、ホラー、ミュージックビデオ、ドキュメンタリー、キャラクタアニメーション、ボクシングやサーフィンなどスポーツの映像など、VRならではの、体験しなければわからない映像が注目されています。

毎年1月に米国で開催される若手作家達の作品を中心としたサンダンス映画祭に、2016年度は、新たにVR映画の部門が開設されました。そこで出展されたVR作品の一部は次のようなものです。

- **6×9:**囚人になって独房を体験するもの
- **Across the Line:**差別される人物を体験するもの
- **A History of Cuban Dance:**キューバのダンス映像
- **Cardboard Crash:**車の事故を体験するもの
- **Click Effect、theBlu:**海中ダイビングを体験するもの
- **Collisions、Condition One:**大自然のドキュメンタリー
- **Defrost:**未来を描いたSF映像
- **fabulous wonder.land:**ルイスキャロルの不思議な国の世界を描いたもの
- **Irrational Exuberance、Sequenced、The Martian VR Experience:**宇宙空間を描いたもの

CARDBOARD CRASH:
車の衝突事故や、衝突回避の様子をVRで再現したもの

The Marian VR Experience:
映画「オデッセイ」と同様、火星でのサバイバル

fabulous wonder land:
ルイスキャロルの不思議の国をVRで描いたもの

※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。

- **Sisters: A Mobile VR Ghost Story:**ホラー映画
- **Stonemilker、Surge:**ミュージックビデオ
- **Job Simulator:**職業体験シミュレーター
- **The Rose and I:**キャラクタアニメーション

作品を手がける監督らも、遊園地のアトラクションを手がける映像作家や過去にアカデミー賞にノミネートされたことのある人物など、単なるデモ映像の粋を超えた、本気の作品群ばかりです。映像演出の面で経験のある有能な人材がかかわり始めている一方で、単なる映像視聴だけではない、VRならではのインタラクティブな体験を得るために、VR空間内でのユーザインターフェースも重要な要素としてとらえられています。

現実世界を模倣するVR

VRの世界のユーザインターフェースの系統は、大きく4つに分かれると考えられます。

- 古くから研究されてきたバーチャルリアリティの文脈で操作するもの
- 3Dゲームで使われてきたヘッドアップディスプレイやファーストパーソンゲームなどの操作を引き継いだもの
- スマートフォン画面での操作の流れをくんだもの
- 現実世界における体の動きや、身振り手振りなどの振る舞いを模倣したもの

現在多くのVRコンテンツでは、両手で扱う使い慣れた家庭用ゲーム機のゲームコントローラが利用されています。一番慣れていると思われるゲームパッドでさえ、VR空間に没入しているときは手元を見ることはできず、触覚と記憶で操作することになります。一般的なマウスやキーボードはほとんど使い物になりません。現実世界では、手の操作を目で見て、それがフィードバックになってさまざまな操作を調整することができますが、VRの場合、目が見えているのに手元が見えない状態で、触覚のみを指針に操作しなければいけません。そのため握ったままで使えるVR専用の入力デバイスや、LeapMotionのようなジェスチャを認識するための機材、視線やまばたきといった、VRデバイスならではの入力方法が考えられています。

古いタイプのVRでは、ヘッドアップディスプレイと呼ばれる四角い枠が三次元空間内に唐突に表示され、それを操作することがよくありました。実際は視野や焦点が分断されるのであまりいい方法ではなく、理想は三次元空間内のすべてのものがシームレスに見られるよう、今見ている空間内に操作できるものを配置するのが適切です。

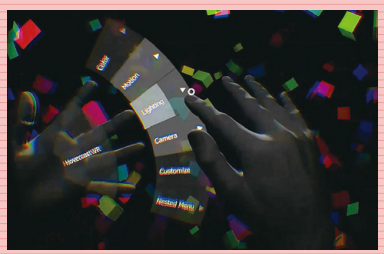
その場合、何を選択したのかがわかりにくい場合が多いので、色が変わったり、サイズが大きくなったり、音で補足したりと、二次元のユーザインターフェース以上に配慮が必要です。選択するオブジェクトやターゲットを指示するマークか何かを目立たせるのではなく、ターゲットそのものが光ったり、印がついたりしてわかるようにする工夫も有効です。

全体的に、あまり遠くにあるもの、あまり近くにあるものを見続けなくてもすむように演出を考えることです。これらの方策は、三次元空間内に自分の身体の一部が映り込んでいるのか、全体が映り込んでいて上から鳥瞰して見ているのか、自分の身体はまったく映り込んでおらず、無の状態なのかによっても変化します。ファーストパーソンシューティングと呼ばれる一人称のゲームタイプのもの、少し上から俯瞰して自分自身が見えるもの、建築物のウォークスルーのように視点や演出が決まっており、ただ傍観しているだけのVRコンテンツなどもあります。

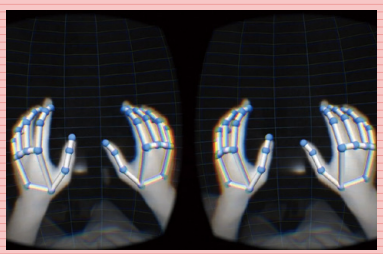
Oculus Touch、SteamVR、PlayStation MoveといったVR用として活用される入力デバイスで得られる入力系統は次のものが考えられています（すべてのデバイスが全部に対応しているわけではありません）。

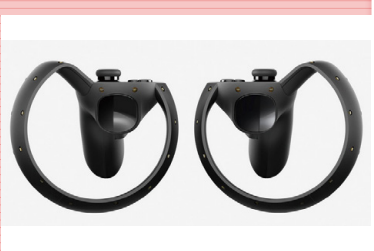
- 手の動きのトラッキング
- 人差し指トリガー（拳銃を打ったりする際に利用）
- 単純な押しボタン
- トラックパッド
- 接触センサー

実在感には、見られることと触れられることの2つの要素があり、手に持った入力デバイスでは、単に操作を入力するという意味だけではなく、つかむ、つまむ、動かす、つつく、といった操作で、現実感を増すための役割を果たします。



Hovercast VRメニュー。両手の指をうまく組み合わせて表示させる3Dメニュー


LeapMotionで両手のトラッキングをしている様子


6DOF（6軸）のセンサーを搭載したOculus Touch

一方、専用の入力デバイスではなく、VR空間内での振る舞いそのものを操作のきっかけとして利用もできます。あるアイコンを画面の中央に持ってきて、しばらく同じ場所を注視すると選択していることになるインターフェースや、普段あまり見ない、足下を見る感じで下を向くと、何かの操作の代用になるなどといった方法です。

統合すると、VR空間内では次のような要素がユーザインターフェースになり得ます。

- 頭の動き
- 手で持つ何らかのコントローラ
- 視線方向
- 三次元空間内での立ち位置
- 手の動きのトラッキング
- (コントローラで認識する)指の動き
- 音や光で、意識的に考えさせる要素

VR空間内での振る舞いを操作に対応づける場合、視覚や視野方向をリセットする手段を用意しておくと良いでしょう。また、現実世界をできるだけ模倣するのが得策で、現実世界にある見慣れた物体が、VRの世界でも同じような大きさで描かれていることが重要です。さらに、通常考える以上に、画面内のユーザインターフェースの密度を少なくする配慮も必要です。

一般的に、解像度や視野角の制限により通常のコンピュータ画面上の何分の一しか認知することができません。悩んだときは、実際の世界での立ち振る舞いを参考にし、判断できない部分やよくわからないときは、テストにテストを重ねるしか、いまのところ具体的な解決策はないと考えます。

スピルバーグが映画化を予定している人気のSF小説で、2044年の未来を描いた『Ready Player One』では、現実世界は荒廃してしまい、人類は皆「オアシス」という仮想空間に逃げこんでいる様子が描かれています。VR世界は決して逃げ込む場所ではなく、現実世界を拡張する存在であってほしいものです。SD

Gadget 1

VRGO

http://www.vrgochair.com/

VR用の椅子

VRGOは、VR映像の体験中に座る、バランスボールのようなVR専用の椅子です。起き上がり小法師のような椅子で、前後左右の動きをBluetooth経由でパソコンへ送信します。通常、キーボードやゲームコントローラで操作していたものを、椅子の操作と対応づけることができます。主な動作としては、前進、後進、左折、右折、回転などで、立って動くと不安定ですが、椅子に座った状態であれば転ぶこともありません。便利なのは、VR機材を収納するボックスにもなる点です。

Gadget 2

Gloveone

http://www.gloveonevr.com

VR専用手袋

GloveoneはVR専用グローブ(手袋)です。VR空間の物体を触った感触を手に伝えるためのグローブです。各指5ヵ所と手のひら5ヵ所の部分に振動素子が搭載されており、震える周波数と強さによって、擬似的な感触を伝えるしくみです。230ドルで販売の予定です。旧来のVR用グローブは仰々しい機械式であったり、振動センサーも大きく数が少ないものが多いなか、軽量で安価であることが特徴です。ただし、手の形状は動きなどをセンシングできないので、ほかのデバイスと組み合わせて活用する必要があります。

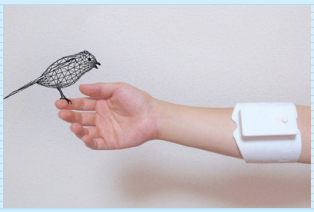
Gadget 3

UnlimitedHand

http://unlimitedhand.com/

腕用フォースフィードバック兼入力デバイス

UnlimitedHandは、日本の大学発ベンチャーが開発する、新しいタイプの入力デバイスです。腕に巻くだけで操作でき、ゲーム内の触感も得られるデバイスです。デバイスにはモーションセンサーと筋変位センサーが搭載されています。また、電気刺激によって装着者の筋肉を収縮させて、フィードバックを与えています。ダミーの振動ではなく、身体へのフィードバックであることが最大の特徴です。



Gadget 4

Razer Hydra

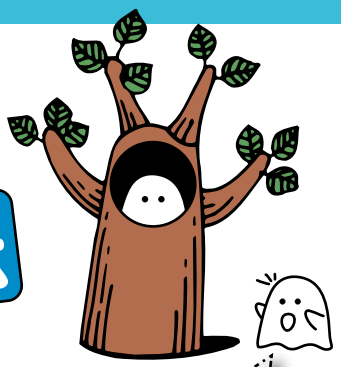
http://www.razerzone.com/gaming-controllers/razer-hydra-portal-2-bundle

両手コントローラ

Razer Hydraは左右の両手で持って使うゲームコントローラです。VR専用に作られたものではないですが、VR空間操作用に適しています。WiiやPlayStation Moveなどの家庭用ゲーム機におけるモーションコントローラと同等の機能を持ちます。球形のベースステーションからケーブルで接続された2本の棒状コントローラで操作します。親指で操作するアナログスティックと、5つのボタンが搭載されています。実際の操作は慣れるまでなかなか難しいそうです。599.99ドルで販売されています。

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

結城浩の再発見の発想法 36



DSL

DSL——ドメイン固有言語

DSLとは

DSLとは、Domain-Specific Languageの略で、**特定の分野や用途に限定した言語**のことです。日本語では「ドメイン固有言語」や「ドメイン特化言語」と呼びます。ここでいう「言語」とはプログラミング言語の場合が多いですが、必ずしもプログラミング言語とは限りません。

DSLはITの世界にたくさん存在します。たとえば、**正規表現**は「文字列のパターンマッチ」という用途に限定した言語と言えます。また、**CSS**（**Cascading Style Sheet**）は「Webページのスタイル記述」という用途に限定した言語です。**JSON**（**JavaScript Object Notation**）は「データ交換」という用途に限定した言語で、汎用プログラミング言語であるJavaScriptのサブセットとして作られています。

DSLの例：Makefile

開発の自動化ツールとして使われる**Make**では、Makefileと呼ばれるファイルを利用しますが、その書式はDSLの一種と言えます。Makefileの記述例を**リスト1**に示します。

ここには、

- fooというターゲットはfoo.cというソースに依存すること
- fooをfoo.cから作るためにはgcc -o foo foo.cというコマンドを実行すること

という情報が書かれています。Makeを実行すると、ファイルfooとファイルfoo.cのタイムスタンプを比較し、もしもfooのほうが古かったらコマンドを実行してfooを作ります。

「タイムスタンプを比較してソースよりもターゲットが古かったら、コマンドを実行してターゲットを作りなおす」というのは、プログラム開発でよく起きることです。Makefileには、その「よく起きること」のエッセンスとして「ファイルの依存関係と実行コマンド」を記述するのです。

先ほどのMakefileによる処理は、汎用のプログラミング言語でも書くことができます。たとえばRubyなら、**リスト2**のように記述できます。

しかし、ファイルの数が何十何百のように多い場合には、いちいちこのような記述をす

▼リスト1　Makefileの例

```
foo: foo.c
        gcc -o foo foo.c
```

▼リスト2　Rubyで記述した例

```
if not File.exists?('foo') or
  File.new('foo').mtime < File.new('foo.c').mtime
  system("gcc -o foo foo.c")
end
```

ることは困難ですし、間違う可能性も高いでしょう。

Makefileの書式を使えば、「ファイルの依存関係と実行されるコマンド」を非常に簡潔に記述できます。簡潔に記述できるのは、Makefileが「自動的な開発」という特定の用途に限定しているからです。Makefileは典型的なDSLと言えるでしょう。

DSLのメリット

DSLを使うと必要な情報を簡潔に記述できるので、生産性が高くなります。汎用プログラミング言語では、広い用途に対応するため冗長な記述が必要になることがありますが、DSLでは用途が限定されているため無駄のない記述ができるのです。DSLでは必要な情報を端的に記述できるので、手間を減らし、可読性を上げることができます。

「よく使う処理をまとめて効率化を図る」というのはプログラミングでよく行われることです。たとえば、関数やライブラリも、よく使う処理をまとめていることになります。DSLの場合には「言語」という形でまとめている点がおもしろいと言えるでしょう。

当然のことですがDSLにも良し悪しがあります。「よく使う処理」の切り出し方が不適切だったり、DSLそのものの設計が悪かったりすると、思ったほど生産性は高くなりません。

また、DSLは1つの言語ですから、習得に掛かる時間も無視できません。記述が簡潔になるといっても、自分がやりたいことの記述方法を調べる時間が掛かり過ぎては困りますね。

1つの用途に限定しても、DSLが1つに定まるとは限りません。たとえば、Makeのような自動化ツールはAnt、Maven、Rakeなどたくさん存在し、ファイルの依存関係を記述するそれぞれ別の書式を持っています。

日常生活とDSL

DSLはあくまでITの世界の概念ですが、日常生活にもDSLの発想を応用できるものがあります。

たとえば、病院で書く**問診票**があります。患者は、問診票に住所、氏名、年齢、そして現在の症状や体温などを書きます。医者が患者と対話を行っても同じ情報は得られますが、問診票に患者が記入すれば、短時間で必要な情報をもれなく得ることができます。これは、用途を限定することで、必要な情報を簡潔に表現できるDSLと同じ発想にあります。

また、作業員が毎日書く**日報**はどうでしょうか。定型化された作業に限定して考えるなら、日報の記述方法もある程度定型化できそうです。つまりそれは、日報のための「言語」を考えることにつながります。

あるいは、ファミリーレストランやファーストフードの**接客マニュアル**もDSLに似ています。接客マニュアルでは、繰り返し発生する状況を適切に処理する定型化がなされていますから、接客マニュアルを使って、教育の効率を上げることができるでしょう。また、サービスの品質を均一に保つ効果もあります。

もちろん、問診票であれ、日報であれ、接客マニュアルであれ、それが効果を発揮するのは適切に設計されている場合だけです。問診票の設計がまずければ情報がうまく整理されず、医者と患者の対話の効率は上がりません。日報の記述が形骸化してしまっては、報告の役目を果たさなくなります。接客マニュアルの設計がまずければ、教育効果も上がらず、個人的に良い接客ができる従業員の対応まで均一に悪くしてしまう場合もあるでしょう。

◆ ◆ ◆

あなたの周りを見回して、記述に毎回手間取っているものはありませんか。用途を限定し、簡潔に記述できる「言語」を設計することで、その手間を軽減することはできないでしょうか。

ぜひ、考えてみてください。SD

増井ラボノート

コロンブス日和

第7回　Gyaim

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

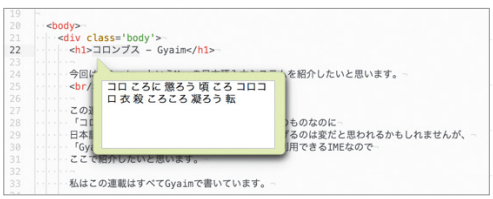
かな漢字変換システム

今回は「Gyaim[注2]」というMacの日本語入力システムを紹介します。

この連載は、シンプルなのに便利な「コロンブスの卵」的なシステムを紹介する趣旨のものなのに、日本語入力システムのような複雑なものを取り上げるのは変だと思われるかもしれませんが、「Gyaim（ギャイム）」は単純な原理にもかかわらず実用的に利用できるIMEですので、ここで紹介したいと思います（**図1**）。私はこの連載をすべてGyaimで書いています。

現在のパソコンでは、「かな漢字変換システム」で日本語入力を行うのが普通になっています。かな漢字変換システムは、1978年に東芝から販売された日本語ワープロで初めて導入されたものですが、パソコンの黎明期から標準的に利用されてきています。かな漢字変換システム以外にもさまざまな日本語入力システムが提案されてきましたが、それほど努力しなくても普通のユーザがとりあえず使えるうえに、熟達すれば高速に入力が可能だという特長があるため、かな漢字変換方式の日本語入力が廃れることはなさそうです。

▼**図1**　AtomでGyaimを使っているところ



注1）http://thinkit.co.jp/free/article/0709/19/
注2）http://masui.github.io/GyaimMotion/

連文節変換

現在のパソコンのかな漢字変換システムでは、いわゆる**連文節変換**が主流になっています。連文節変換とは、文章の読みをすべて入力してから、漢字混じりの日本語文字列に一気に変換する手法のことで、たとえば「わたしのなまえはなかのです」という入力を「私の名前は中野です」に変換するというものです。「私の」「名前は」のように、息継ぎできる場所で文を区切った単位を「文節」と呼びますが、連文節変換では複数の文節を含む文を一気に変換できるのが特長です。

連文節変換は一見便利そうですが、実は次のようなさまざまな問題があります。

- 正確な入力が必要
  watashinonamaeha...のような文字列を1文字も間違えずに入力する必要がある
- 完全な読みの入力が必要
- 変換誤りの訂正が必須
- 非力なマシンで利用しにくい

これからのユビキタス社会においては、パソコンの熟練者だけが使える入力手法ではなく、どこでも誰でも簡単に使えるシンプルで柔軟な入力方式が必要で、次の要件が満たされる必要があるでしょう。

- 必要なキーやボタンの数が最小限
- 操作の種類や量が少ない

・ユーザがカスタマイズ可能
・さまざまな環境で同じ方式が利用可能

▼図2 SKK利用例

▼図3 日本語入力としてGyaimを選択

このような要件を満たすためには、単純なアルゴリズムに基づいて単純な操作で入力を行えるようなコロンブスの卵的な入力システムが必要だと思われます。

SKK

連文節変換を利用しない場合、かな漢字変換は辞書を参照して読みを漢字に置き換える単純な検索作業に近く、動詞の活用への対応などを除けばそれほど難しいものではありません。

単純なしくみで日本語入力を行うシステムの例として、Emacs上のSKKというシステムがあります。SKKは1987年ごろ、京都大学名誉教授の佐藤雅彦氏が東北大学にいたとき開発したEmacs用の日本語入力システムです(**図2**)。

SKKには英数字モードと日本語モードがあります。日本語モードでローマ字を入力するとひらがなが入力されますが、「書く」のような漢字を含む動詞を入力したい場合は「KaKu」のように大文字と小文字を混ぜたローマ字を入力します。最初の大文字「K」は変換の開始を意味しており、2つめの大文字「K」は送り仮名の開始を意味しています。SKKの辞書には次のようなエントリが含まれており、

```
かく /核/格/各/角/画/確/:
かk /書/掛/欠/
```

「Kaku」と入力された場合は「核」「格」「各」のような漢字への変換が行われ、「KaKu」と入力された場合は「書く」や「欠く」のような文字列に変換されるようになっています。つまり変換開始場所や送りがなの場所をユーザが明示することによって、単純なアルゴリズムでの変換を可能にしているわけです。

Gyaim

SKKは単純なしくみで効率的な日本語入力ができる、コロンブスの卵的な優れた日本語入力システムなのですが、一般的なかな漢字変換システムと同じように、すべての読みを正確に入力する必要があるうえに、漢字や送り仮名についてユーザが明示的に指定しなければならないためユーザの負担が大きく、万人向けとは言い難いものでした。私は1990年代ごろはSKKを愛用していたのですが、携帯電話の予測型テキスト入力システム「POBox[注3]」の開発後はあらゆる場所でPOBoxに準じた入力手法を使っており、MacではGyaimを使っています。

GyaimはMacで動くシンプルな日本語入力システムです。Gyaimは約1,000行のRuby(Ruby Motion[注4]を使用)で記述されており、スマホの予測入力システムと同じように利用できます。

Gyaimは次の手順でインストールします。

・http://masui.github.io/GyaimMotion/ からGyaim.dmgをダウンロードして展開
・Gyaim.app を ~/Library/Input Methods に置く

このように設定を行ったあと、環境設定画面の[キーボード]→[入力ソース]でGyaimを選択し、追加します(**図3**)。

Gyaimの実装

IMEの作成には、ユーザインターフェースとかな漢字変換アルゴリズムが必要です。

注3) https://ja.wikipedia.org/wiki/POBox
注4) http://www.rubymotion.com/jp/

IMEのユーザインターフェース

Macには、IMKit[注5]というIME作成用ライブラリが用意されており、それを利用してIMEを作成できます。IMKitにはさまざまな機能が用意されていますが、GyaimではIMKInputControllerクラスのhandleEventというAPIのみを利用しています。候補を表示したり選択したりするには、任意のCocoaライブラリを利用できます。

かな漢字変換アルゴリズム

ユーザインターフェースはシステムごとに用意する必要がありますが、かな漢字変換アルゴリズムはAndroidでもMacでも同じものを利用できます。GyaimではRubyでかな漢字変換アルゴリズムを実装していますが、同じアルゴリズムをJavaやJavaScriptで実装すれば、Androidやブラウザなどで利用できます。

GyaimではSKKと同様に読みと漢字の対応辞書を使ってかな漢字変換を行います。たとえば日本語入力モードで「とうきょう」と入力されたとき、辞書から「とうきょう」という読みを持つ単語を検索して、リストして候補として表示し、ユーザの操作で候補を選択してテキストに貼り付けます。

「東京」のあとには「駅」や「大学」のような単語が続くことがあります。「東京駅」「東京大学」のような単語をすべて辞書に登録しておくのはたいへんですので、「東京は地名である」「地名のあとには駅や大学が続くことがある」という情報を辞書に登録しておくことにより、「東京駅」が辞書に登録されてなくても「とうきょうえき」を「東京駅」に変換できるようにしています。

まったく同じ手法で動詞の変化形も扱うことができます。たとえば辞書に、

・「書」の読みは「か」である
・「書」のあとにはカ行五段活用語尾がつながる
・「か」はカ行五段活用語尾である

といった情報を定義しておけば「かかない」を「書かない」に変換できます。このように、単語の読みと属性、接続情報を定義しておくだけで、それなりに自然言語を入力できますし、日本語以外の文字入力でも利用できます。

Gyaimの特殊機能

Gyaimでは前述のような非常に単純な変換手法を使っていますが、特殊な機能も用意しています。自前のIMEだと好きな機能を自由に実装できるのが楽しいところです。

単語登録機能

一般的なIMEは単語登録が面倒なことが多いので、頻繁に単語登録を行っている人は多くないでしょう。たいていのIMEにおいて、単語登録は入力とはまったく異なるインターフェースになっていますが、Gyaimでは入力操作と登録操作をほとんど同じにすることにより簡単に単語登録が行えるようになっています。Gyaimでは「選択中の文字列があったりコピーバッファに文字列があった場合は候補として表示する」という単純な方法で単語登録を可能にしています。

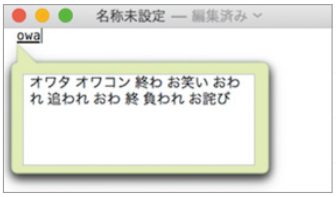
たとえば「＼(^o^)／」という文字列を「owata」という読みで登録したい場合(**図4**)、「＼(^o^)／」という文字列を選択またはコピーしておいてからGyaimで「owata」と入力すると「＼(^o^)／」が候補の先頭に表示され(**図5**)、これを選択して確定することにより単語登録が終了します(**図6**)。

画像変換

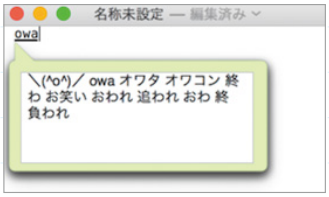
Gyaimでは、辞書の中で文字列の代わりにGyazoの画像ファイル名を登録しておくと漢字の代わりに画像を入力できます。たとえば私の顔画像を「masui」という読みで登録しておくと、

注5) https://developer.apple.com/library/mac/documentation/Cocoa/Reference/InputMethodKitFrameworkRef/

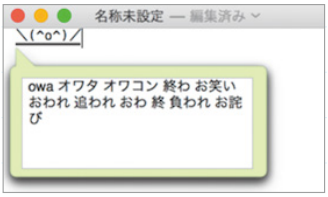
▼図4 「＼(^o^)／」が登録されていない状態



▼図5 「＼(^o^)／」がコピーバッファにある状態で「owa」と入力



▼図6 「＼(^o^)／」を選択。確定すると単語登録される



Gyaimで「masui」と入力することにより顔画像を入力できます(**図7**)。

時刻の入力

Gyaimでは「ds」と入力すると現在時刻が候補に出るようになっており、文章やプログラムの中に手軽にタイムスタンプを記録しておくことができます。数式演算機能のような特殊な機能を追加することも簡単です。こういったちょっとした機能が用意されているIMEはありますが、GyaimはRubyを少し書くだけで特殊な機能でも簡単に追加できるのがうれしいところです。

▼図7 画像変換

秘密文字列の入力

ブラウザなどでパスワードやクレジットカード番号を入力するのは面倒なものですが、IMEでこれらを覚えておけば簡単に入力を行うことができます。もちろんこれらを生テキストで単語登録するのは危険ですが、秘密の文字列を変換することによりクレジットカード番号が候補に表示されて入力可能にしています。

Google変換の利用

Gyaimの辞書は貧弱ですのでうまく変換できないこともよくありますが、そういうときはGoogle入力ツールのようなWeb上の変換APIを利用して変換を行えるようにしています。

◆ ◆ ◆

IMEは複雑な要素を含んでいるのでGyaimの詳細について書くことはできませんでしたが、とりあえず簡単なIMEをMacで作ることができることが理解いただけたでしょうか。

私のように単純な変換方式を好む人はGyaimのような方法を工夫すれば良いでしょうし、連文節変換が好きな人はGoogleなどの変換APIを利用することもできます。IMEの作成がたいへんだった昔と異なり、現在はIME用のAPI／使いやすい強力な言語／Webの変換API／高度な入出力機能などを利用できますからIME作成のハードルは極めて低くなってると言えるでしょう[注6]。IMEを自分で作る試みが、今後もっと増えると良いと思っています。SD

注6) Gyaimのソースコードは https://github.com/masui/GyaimMotion で公開しています。

宮原徹の

オープンソース放浪記

第3回 OSC東京春と金沢での勉強会

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

最大規模のOSC東京春開催

2016年2月26日(金)・27日(土)の2日間、明星大学で開催されたOSC東京春は、2日間で1,550名(650名+900名)もの参加者が集まりました(**写真1**)。

開催期間中は人の出入りが見渡せる受付にいますが、久しぶりに会う人、わざわざ遠方から出張と合わせて足を運んでくれる人(OSC東京に合わせて東京出張の人が多い!)と会話するのも楽しみの1つです。さらに、いろいろなお土産やお酒の差し入れも多く、どんどん酒瓶が増えていきます。

懇親会も大規模です

OSC東京の半分は懇親会でできていると思えるぐらい、200名近くが参加する一大イベントです。今回は事務局を受け持っているびぎねっと(筆者の会社)15周年記念の感謝パーティーを兼ねるということもあり、飲み物は発泡酒からビールに格上げして思う存分飲んでもらいました。さらに有志からお祝いにベルギービールを6リットル瓶(!)で寄贈してもらいました(**写真2**)。中身だけでも6キロ以上、瓶の重さも半端なく、片手では持ち上げられないほど。両手で抱えて会場内を歩き回り、みなさんに振る舞わせていただきました。

「Bar root」を出張開店

懇親会でのもう一つの楽しみは、臨時のバーを開店することです。「Bar root」という店名は、私の名前である「とーる(徹)」をひっくり返したのと、システム管理者rootを引っかけてのネーミング。普段はオフィスなどで親しい人を招いて開店していますが、OSCでは大々的にお酒を振る舞います。持ち込んだお酒の由来や味などを説明しながら飲んでもらうと、本当にバーでお客さんを相手にしているような気分になれますね(**写真3**)。

OSCはみんなでやるもの

これだけ規模の大きいイベントとなると、事務局スタッフだけでは人手が足りないので、前日準備や当日運営はたくさんのボランティアスタッフに支えられています(**写真4**)。ただし、OSCではボランティアスタッフも自由にセミナーを聴講したり、展示を見に行けるようにしています。OSCをシステムと考えると、疎結合な自律分散型のアーキテクチャで設計しています。たとえば、展示スペースの配置決定や、セミナーのスケジューリングなどは事務局が行いますが、そのあとは割り当てを受けた各出展者がそれぞれ自律的にセミナーや展示を行います。例外的な処理が発生したときのみ事務局が対応する、というしかけです。そのため、例外が発生していないか常時ウォッチしておくことが重要ですので、通常の参加者と同様にセミナーや展示を見て回ることがスタッフの重要な仕事になります。

▼写真1 閉会式後、有志に残ってもらい記念撮影。たくさん参加してくれました

▼写真2 6リットル瓶。共同設立者の濱野氏と

▼写真3 Bar rootに並べられた酒瓶。今回は焼酎、泡盛多めです

▼写真4　片付け終了後、スタッフ皆で記念撮影。お疲れ様でした!

大規模なイベントでも、システム設計しだいで人手をかけなくても廻るというよい例になっていると思います。興味がある方は、ぜひOSCにスタッフとして参加してみてください。とくに学生のみなさんには、交通費の支給やスタッフTシャツのプレゼントなど特典がたくさんありますので、たいへんお得ですよ。

北陸新幹線で金沢へ

OSC未開催の地域の1つが北陸です。北陸新幹線が開通して1周年の3月14日に、石川県金沢市で地元のクラウド関係のユーザグループが合同の勉強会を開催すると知り、勉強会に参加してきました(**写真5**)。当日は50名近くの参加者が集まり、かなりの熱気です。AWSやAzure、SoftLayer、Bluemix、Dockerなど、それぞれのサービスのエヴァンジェリストが最新情報を提供する、なかなか充実した勉強会でした。ライトニングトークの時間もあったので、私もOSCの紹介をさせていただき、OSC金沢を開催したいと思う人は声をかけて、とお話したあとに、懇親会へ。まずは会場で1時間ほどお茶とお菓子で懇談したあとで、市内の居酒屋へ移動して懇親会へ(**写真6**)。

北陸は海産物が美味しいのと、日本酒も美味しいですね。すでに土日でいくつかの日本酒銘柄は飲み比べていましたが、まだ飲んでいない銘柄を一通り味わって、そのお店にある銘柄はすべて制覇したのでした。北陸の日本酒はすっきりしていて飲みやすく、翌日にも残らない、なかなか素敵な味わいでした。OSC金沢を開催したいというご意見もたくさんいただけたので、開催の際にはまた美味しい日本酒が味わえそうです。

懇親会のあとは、さらに東京から来ていた講師陣と居酒屋で2次会、さらに謎のレトロゲームバー「Hello,World!」でファミコンカセットのコレクションを眺めながら、金沢の夜は更けていくのでした。SD

▼写真5　セミナーの様子

▼写真6　懇親会で記念撮影

Report

Bar root出張開店
生ハム原木で大盛り上がり

OSC東京でのBar root出張開店の目玉が生ハム原木。昨年秋の開催時に初めて出してみて、1回の懇親会で全部食べきれたので、今回も用意してみました。これが大好評で、生ハムを食べたいと大行列ができました。生ハムを1切れ切るのにもけっこう時間がかかるのでしかたがないんですよね。最後尾の人が生ハムにありつけるまで5分待ち。スライサーを使って切ればあっという間なのですが、手で切るからこその「味」もある気がしています。参加人数が少ないと食べきれず、そのあとの保管がたいへんですが、50人以上集まるようなら食べきれると思いますので、みなさんもお試しあれ。生ハム原木1本(5キロ前後)に固定する台、ナイフのセットで1万5千円ぐらい。ナイフは別途切れ味の良いものを用意しておくとよいでしょう。

▲生ハムを求めて列を作る人々。待つから並ばないで~、と言っても並びます

▲生ハムを必死になって切る筆者。段々と生ハムカットスキルが上がっていきます

ツボイのなんでもネットにつなげちまえ道場

第11回 クラウドサービスにつないでみる

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

IFTTT

前回はmbed LPC1768をEthernetに接続して、HTTPで通信を行うところまでやってみました。マイコンから手元のスマートフォンにちょっとした通知を送りたいとき、通知を中継するサーバの開発や、スマートフォンのアプリケーションを開発するのは面倒です。そこで、今回は、お手軽なサービスを使って、マイコンから送った通知を受信できるしくみを作ってみましょう。

ところで、みなさんはIFTTT（イフト）[注1]というサービスをご存じでしょうか（**図1**）。IFTTTは、If This Then Thatの頭文字から名付けられたサービスで、その名のとおり「もしコレなら、アレをする」といった単純なしかけを、レシピと呼ばれる形式で保存して走らせることのできるものです。コレとかアレの部分には、EvernoteやFacebook、Twitterといったほかのサービス、あるいはスマートフォンの機能、BMWの車（日本では未対応）やPhilipsのHueという多機能電球といった製品など幅広い選択肢が提供されています。こういったコレやアレに当てはめることができるものは、チャンネルという名前で登録されていて、記事執筆時点では286個のチャンネルが登録されています。

IFTTTを使うと、翌日の天気予報が雨であれば、iOSやOS Xのリマインダに、傘を忘れないように自動的に記入をさせるといったことができます（**図2**）。

同様のサービスに、Zapier（ザピエル）[注2]というものがあります。IFTTTはすべて無償で提供されていますが、Zapierには有償プランがあります。筆者は、無償だけど機能がシンプルなIFTTT、あるていど使うならば有償プランが欲しくなるけれどもより多機能なZapier、だと思っています。

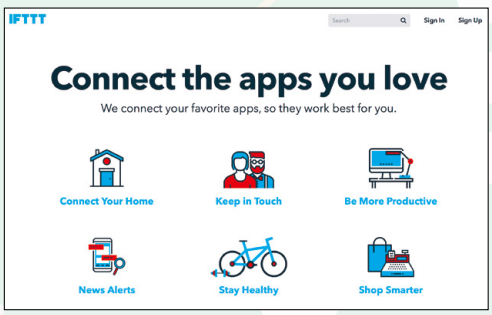
Maker Channel

そんなIFTTTに、自分が作った機器などをコレとかアレの部分に当てはめることが手軽に

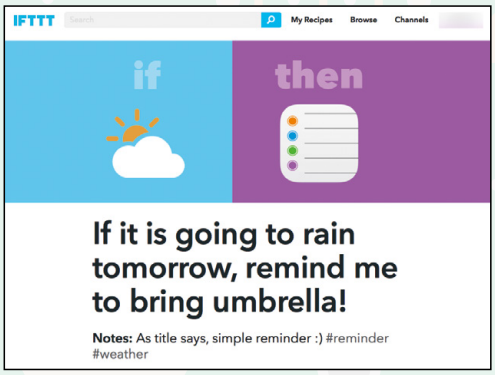
注1） https://ifttt.com

注2） https://zapier.com

▼図1 IFTTT



▼図2 リマインダ



できるMaker Channelというチャンネルが追加されました。何のことはなく、HTTPやHTTPSでIFTTTのコレ(This)にトリガを送ることができ、また、アレ(That)としてHTTPやHTTPSリクエストを発行させることができます。

たとえば、IFTTTに何かを処理するためのきっかけ(トリガ)を作るには、マイコンから、

```
https://maker.ifttt.com/trigger/
{event_name}/with/key/{secret_key}
```

にアクセスをするだけです。**{event_name}**は、自分が設定する名前で、**{secret_key}**は、IFTTTのWebサイトで自動的に発行される値です。マイコンから情報を渡したいときには、3つまでJSONで渡すことができます。

Maker Channelを使えるようにする

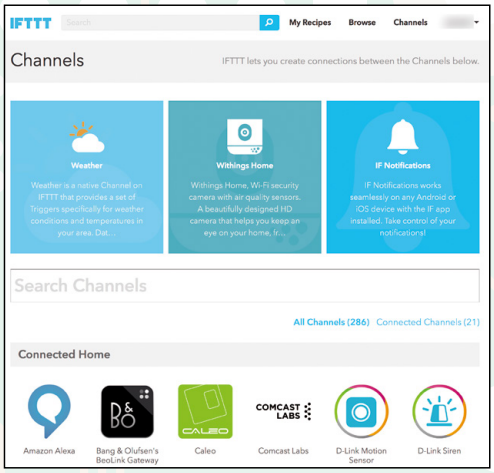
Maker Channelを使えるようにするには、まずIFTTTのアカウントを作ってください。IFTTTのWebサイトにアクセスすると、右上に「Sign Up」というボタンがあります(**図1**)。ここから、簡単にアカウントを登録できます。

ログインしたところで、画面右上の「Channels」をクリックすると、チャンネルの一覧が表示されます(**図3**)。ここで「Search Channels」の欄に「Maker」と入力してMaker Channelを見つけて、選択します(DIY Electronicsというカテゴリにあります)。Maker Channelに入ると大きなMのロゴの右側にConnectというボタンがあるので、クリックします(**図4**)。

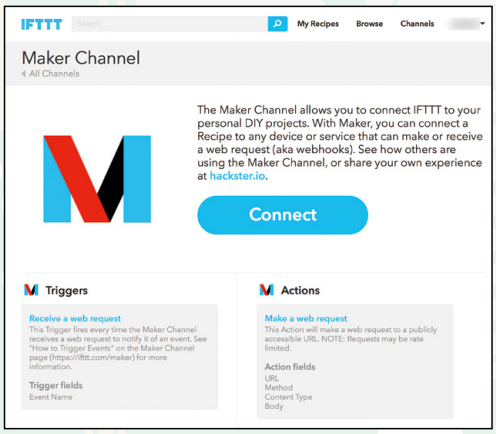
Maker ChannelをConnectすると、**図5**のような画面になります。「Your key is」の下に、このチャンネルを使うときに**{secret_key}**として用いるキーが記されています。このキーをクリックすると、このキーを使ってどのようにIFTTTのMaker Channelを使うか説明が表示されます。

次に、レシピを作りましょう。今回は、マイコンからトリガされたら、プッシュ通知を行うレシピにします。先ほどのMaker Channelの

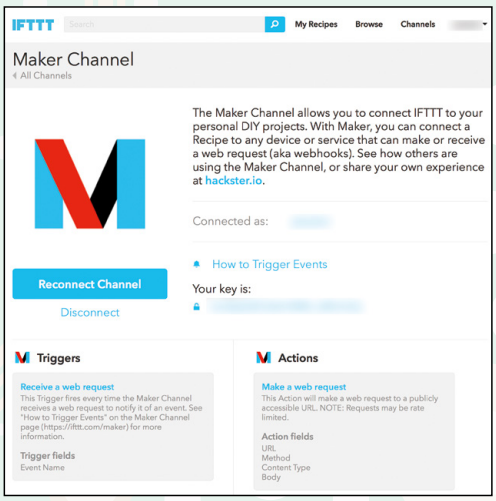
▼図3　チャンネルの一覧



▼図4　Connectをクリックする



▼図5　Maker ChannelをConnectしたところ



ページなどの上部に、「My Recipes」というボタンがありますので、これをクリックしてください。すると、自分のレシピの一覧が表示されます。ここで、「Create a Recipe」ボタンをクリックして、レシピ作成画面に移ります(図6)。ここで「this」をクリックすると、トリガに使うチャンネルの一覧が表示されますので、Maker Channelをクリックします。

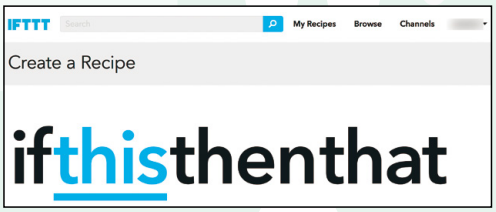
ここで、チャンネルのどの機能をトリガに用いるかを確認されるので、「Receive a web request」(Webリクエストを受信)を選択します(図7)。といっても、今のところ、これしか選択肢がありません。次にイベント名の入力です(図8)。ここに入力した値は、先ほどの{event_name}の部分に使います。今回は「test」にしておくことにします。

トリガを作ることができました。ここからは、アレ(That)、アクションの部分の設定です。「that」をクリックしてください(図9)。するとアクションに用いるチャンネルを選択する画面(図10)になるので、「IF Notifications」を選択します。これは、IFTTTのアプリ、IFを使って通知を送るというものです。チャンネルのどの機能をアクションに用いるかを確認されるので、ここで唯一の選択肢である「Send a notification」をクリックします。

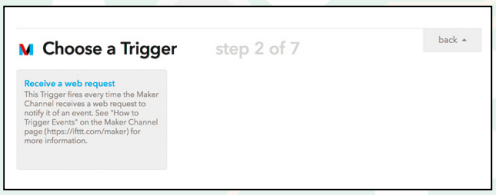
だいたい設定ができました。今度は通知文の内容を編集する画面です(図11)。とりあえずテストですので、編集せずに進みます。最後にレシピのタイトルを確認されますが、これも編集せずに進めます(図12)。

これでレシピが完成しました(図13)。このレシピはIFTTTのアプリであるIFに通知を送るものですので、IFアプリをお手元のスマートフォンにインストールする必要があります。アプリへのリンクは、このページにありますので、お手持ちのスマートフォンのOS用のアプリをインストールしてください。

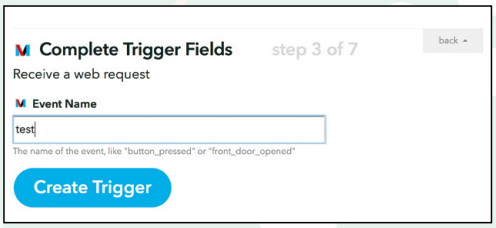
▼図6　レシピ作成画面



▼図7　トリガに使う機能の選択



▼図8　イベント名の入力



▼図9　アクションの設定

▼図10　アクションに使うチャンネルの選択

▼図11　通知文の編集

mbedのプログラム

IFTTTのMaker Channelを利用するライブラリは、mbed.orgで公開されています[注3]。これを使えば、簡単にmbedからIFTTTをトリガできます。サンプルコードは、**https://developer.mbed.org/users/ytsuboi/code/IFTTT-Example/**に置いておきました。いつものように、ご自分のオンラインコンパイラにImportしてコンパイルすれば、手軽に試せます。

main.cppにある、

```
ifttt.addIngredients("hogehoge");
```

のように記述することで、トリガと同時にIFTTTにデータを送ることができます。送られたデータを通知に使うには、

```
The event named "{{EventName}}" occurred on the Maker Channel. Value 1 is "{{Value1}}".
```

といった具合に、レシピのアクションの項を書き換えるだけです(**図14**)。

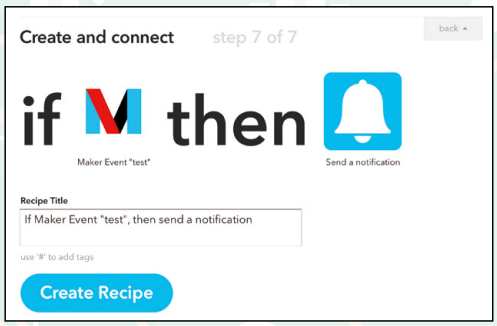
mbedをアプリケーションボードに取り付け、オンラインコンパイラからダウンロードしたバイナリファイルをコピーして、リセットボタンを押すと数秒でお手元のスマートフォンに、**図15**のように通知が届きます。

最後に

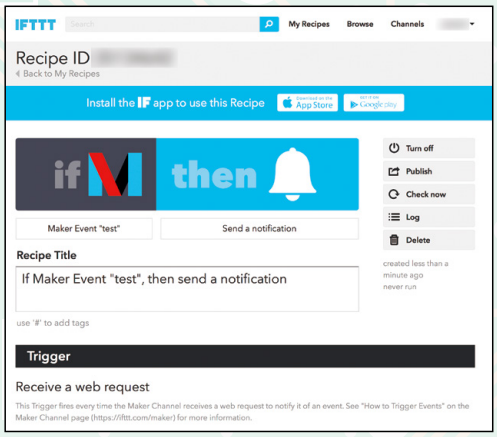
IFTTTのMaker Channelを使えば、このように手軽にスマートフォンに通知を送ることができます。ちょっと工夫すれば、ドアが開いたときに通知を送るなどのしくみに簡単に応用できるでしょう。通知を送るにも、IFアプリを使うほかにPushBulletというサービスを使うという手もあります。また、IFTTTの豊富なチャンネルとの組み合わせで、SNSへのポストなど、さまざまなWebサービスへの接続ができるようになります。SD

注3) https://developer.mbed.org/components/If-This-Then-That-IFTTT/

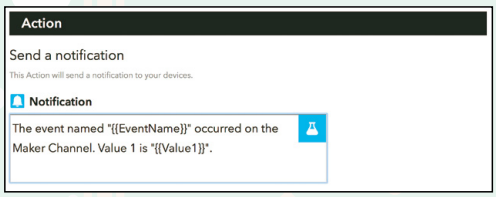
▼図12　レシピのタイトルを編集



▼図13　レシピの完成



▼図14　アクションの設定



▼図15　通知の様子

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5月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

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

01 Raspberry Pi 3 Model B　1名

名刺サイズのコンピュータボードです。Ethernet／4基のUSB2.0／HDMI／micro SDのソケットを持ち、さまざまなデバイスと接続して自分だけのガジェットを作れます。前モデルからCPUが強化され（1.2GHz・64bit）、無線LAN機能（2.4GHz帯・11b/g/n）とBluetooth（4.1）が新たに搭載されました。

提供元　アールエスコンポーネンツ　http://jp.rs-online.com

02 Power Plus 3 13400mAh DANBOARD version　2名

漫画作品「よつばと!」に出てくる『ダンボー』のデザインのモバイルバッテリー。付属のmicroUSBケーブルで本体を充電し、AUTO-IC機能搭載のUSBにて給電します（2A出力、2ポート）。ゴールド／レッドのどちらかをプレゼントします。

提供元　ティ・アール・エイ　http://www.cheero.net

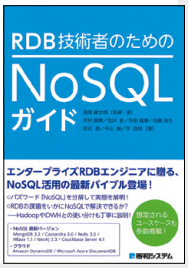
03 microSDカード「TS16GUSDHC10V」　2名

防水性、温度耐性、静電耐性、X線耐性、衝撃耐性に優れた高耐久microSDHCカード（32GB）。ドライブレコーダー、セキュリティカメラ、監視システムといった書き込み頻度の高いアプリに最適です。最大12,000時間のフルHD録画を実現します。

提供元　トランセンドジャパン　http://jp.transcend-info.com

04 RDB技術者のためのNoSQLガイド

河村 康爾 ほか 著



一般のRDBエンジニアを対象に、NoSQLの基礎知識とエンタープライズ視点での活用方法を解説したテキストです。CassandraやMongoDBといったプロダクトの最新情報やユースケースを多数掲載。

提供元　秀和システム　http://www.shuwasystem.co.jp　2名

05 アルゴリズムの基本

トーマス・H・コルメン 著、長尾 高弘 訳

アルゴリズムの入門書です。擬似コードを使わず、少ない数式と日本語でアルゴリズムを解説しています。名著『Introduction to Algorithms』の著者が、同書のエッセンスをまとめたものです。

提供元　日経BP　http://www.nikkeibp.co.jp　2名

06 Swiftポケットリファレンス

WINGSプロジェクト　片渕 彼富 著

iPhone/iPadアプリ開発にも、Swift学習にも便利な1冊です。iOSのフレームワークから、3D Touchまで幅広く解説しています。逆引き形式で、目的からすぐ機能を探せます。

提供元　技術評論社　http://gihyo.jp　2名

07 はじめてのLisp関数型プログラミング

五味 弘 著

Lispで関数型プログラミングを学ぶ1冊。さまざまなLispプログラム（ハノイの塔、エイトクイーン、オンライン書店など）を実装しながら、高階関数や再帰といった関数型プログラムの肝を習得します。

提供元　技術評論社　http://gihyo.jp　2名

コード編集の高速化からGitHub連携まで

Vim[実戦]投入

フリーのテキストエディタの中では圧倒的な人気を誇る「Vim」。システム／インフラエンジニアの第一歩として、また自分のコーディングのさらなる高速化／効率化のために、新たにVimを選択する人は多いのではないでしょうか。本特集では、Vim業界で著名な5人が、初心者に向けた「Vimのはじめかた」、プログラマに役立つ「高速編集術」「正規表現」「GitHub連携」、さらには「Vim開発の最新事情」まで解説します。そして章末には、Vim初学者向けにコマンドを厳選した「Vimチートシート」も付属！　明日からの「実戦」に耐え得る大特集です。

Contents

表記注釈　本特集では、一部キーの入力について次のように表記します。

・Ctrlキー……Ctrl または <Ctrl>　　・Escキー……Esc または <Esc>

・Ctrlキーを押しながらaキーを押す……<Ctrl-a> または <C-a>

Part 1

Vimとの長い付き合いのはじめかた

Author 氏久 達博(うじひさ たつひろ)
URL https://github.com/ujihisa Twitter @ujm

千里の道も一歩から。まずはVimのインストールから始めましょう。vimtutorで基本操作を押さえたあとは、Vimの必須知識、.vimrc・Vim plugin・Vim scriptについて学びます。最後はVimのカスタマイズについて。あなたはデフォルト教？　それともカスタマイズ派？

はじめに

Vim特集にようこそ。はじめはまず、読者のみなさんの前提知識をとくに何も仮定せず、テキストエディタ「Vim」そのものの導入について説明していくことにしましょう。

Vimの導入は非常に簡単です。と言うのも、世のソフトウェア開発用に使われるOSのほとんどにはすでにVimがインストールされており、ターミナルからvimと入力するだけでいきなりVimを使うことができるからです。しかしながら、最初からインストールされているVimはかなり古いバージョンで、かつGUI版がなくCUIでのみ動作というものがほとんどです。

そこで、まずはどのようなVimの配布が利用できるかを整理し、筆者お勧め構成でのインストール方法について解説していくことにしましょう。

どのVimを導入するか

インストール対象となり得るVimは、次のような観点で大雑把に分類できます(2016年3月現在の情報)。

- 古いバージョンか新しいバージョンか(例：Vim 7.3系、Vim 7.4系)
- ターミナル上のみで動作するCUIか、専用のウィンドウを持つGUIか
- VimかNeovimか
- 特定OS固有のVimか(例：Mac OS X用のMacVim/MacVim-KaoriYa、Windows用の香り屋Vim)
- ビルド済みのバイナリ配布か、自分でビルドしたものか

古いVimを積極的に選択する理由はありません。自分でインストールするならば、少なくともVim 7.4以上のものを使いましょう。

CUI版とGUI版のどちらを使うかについては、GUI版をお勧めします。CUI版はスクロールが遅く、また Alt などをVim内のカスタマイズに利用できないなど、ターミナル固有の制限を受けます。CUI版を使う方がモテると考えている方もいるかもしれませんが、GUI版の使用も同程度にモテますので、これを理由にCUI版を選択する必然性はありません。リモートサーバにてファイルを編集するとき、sshのX11 ForwardingやVNCなどを用いるのではなくターミナルからsshを使用するならば、CUI版のVimをリモートで起動する程度にし、あとはなるべく、ローカルではGUI版のVimを積極的に活用していきましょう。

どの形態で配布されているVimを用いるか。Vimの開発最新版は新しい機能に関する実装・仕様が日々変更されており、それを追っていくのは確かに知的好奇心をそそられるもので

はあります。しかし、まさにこれからVimを初めて学習していこうという方には不向きでしょう。そういう意味で、自分で最新版のVimのソースコード持ってきて手動でビルドするよりは、導入のラクなバイナリ配布や、あるいはすでに手元にあるPortage/Homebrew/Portsなどの既存のパッケージマネージャにお任せすることから始めることをお勧めします。

特定OS固有のVimについて。「香り屋Vim」はKoRoNさんが作成・メンテナンスをしている、“進化したイケてるヤバイ”便利版です。任意の環境で利用できるためのパッチと、Windows用にパッチが適用され、ビルドされた結果のバイナリが公式に配布されています。また、splhackさんによるMac OS X用の香り屋Vimも、MacVim-KaoriYaとして配布されています。

Neovimについて。これ、かなり特殊です。詳しくはPart5を参照ください。

◆ ◆ ◆

ざっとそれぞれについて説明しましたが、ここで著者のお勧めをまとめると、

・Macなら最新安定版MacVim-KaoriYa
・Windowsなら最新安定版香り屋Vim
・それ以外なら少なくとも7.4系であるgvim

となります。

OS別のインストール

前述のお勧めに沿って、環境ごとのインストール方法について簡単に解説します。

・Mac

①MacVim-KaoriYa GitHubのページ[注1]の説明を読む
②説明に沿って、リリースページから最新版をダウンロード
③ほかのMac OS Xのアプリケーションと同様、ダウンロードした.dmgファイルの中のMacVimアイコンをApplicationsディレクトリにドラッグ

・Windows

①香り屋Vimの配布ページ[注2]の説明を読む
②使用しているWindowsのバージョンに対応する最新版のバイナリをダウンロードする

・Linuxなど

①ディストリビューションごとに配布されている最新版gvimをインストールする[注3]
②そのバージョンが、少なくともVim 7.4以上であることを確認する

実際にGUIのVimを起動してみましょう。

MacではVimをDockに入れておくか、MacVim-KaoriYaの/Applications/MacVim.app/Contents/MacOS/にPATHを登録したうえでmvimコマンドで起動します。

Windowsの場合は、ダウンロードしたVimプログラムフォルダ内のgvimをダブルクリックして起動します。

Linuxなどの場合は、gvimコマンドを用います。

余談ながら、筆者はGentoo Linuxを使用しているので、gvimをPortage経由でインストールし、OS起動後ログインされた状態になるとgvimが自動で立ち上がるようにしています。このgvimをOS終了の最後の瞬間まで使い続けるというスタイルです。

注1） https://github.com/splhack/macvim-kaoriya

注2） http://www.kaoriya.net/software/vim

注3） ここでは、Linux版は香り屋版ではないことに注意してください。香り屋版を容易にインストールできるLinuxディストリビューションは2016年3月現在整備されておりません。ここでは簡単のため、非香り屋版の導入の解説にとどめました。

Vimの基本の基本：vimtutor

本章でもっとも力強く主張したいことは、vimtutorの重要性です。vimtutorはVimが公式に配布するチュートリアルで、Vimをインストールすると必ず付いてきます。つい先ほどインストールしたGUIのVimだけでなく、100万人のユーザに毎秒大量のリクエストをさばいているような運用のサーバにすら、Vimが入っているならばvimtutorも入っています。vimtutorには必要な内容が簡素にまとまっており、実際にこれを起動して自分の手を動かしつつ読み進めるだけで、Vimの基本の基本をスムーズに習得できます。

香り屋系Vimでは、まずGUIのVimを起動し、

```
:Tutorial
```

とすると起動できます。最後にEnterが必要です。

香り屋以外のVimの場合、さまざまな起動方法がありますが、一番簡易なのはコマンドラインのvimtutorコマンドを使うことでしょう。この場合CUI版が起動します。

```
$ vimtutor
```

vimtutorの本文はログインユーザの言語設定が使われます。もし特定の言語を指定したい場合は、引数で言語を指定します。

```
# 日本語
$ vimtutor ja

# ポーランド語
$ vimtutor pl
```

余談ながら、筆者はvimtutor esをスペイン語の学習のために活用しました。

vimtutorの起動は確認できましたか？　もし今手元にパソコンがあるならば、次の節を読み進める前に少なくとも起動だけはしておきましょう。本文中にあるように、25〜30分程度で完了できますし、いつでも中断／再開できます。今すぐ試してみましょう。今手元にパソコンがないならば、パソコンのあるところに物理的に移動しましょう。

vimtutorはVimのあるとき、いつでもどこでもできます。一度ですべてを理解して覚えられなければ、不定期に“vimtutorって”みましょう。Vimの基本操作、モード遷移については、章末のチートシート(P.65)もご利用ください。

初期設定：.vimrc

Vimそのもののインストールが終わったところで、次にVimの初期設定に入ります。Vimの個人用の設定は.vimrcと呼ばれるファイルに記述していきます。.vimrcファイルの位置としては、すべての環境で$HOME/.vimrc、ただしWindowsのみ_vimrcが使われます[注4]。たとえば、ユーザ名がujihisaの場合、

- Linux：/home/ujihisa/.vimrc
- Mac OS X：/Users/ujihisa/.vimrc
- Windows：C:\Users\ujihisa_vimrc

となります。

.vimrcの中身はVim scriptで記述します。Vim scriptはVimの中で使われるプログラミング言語で、現在普及しているモダンなプログラミング言語と比べるとちょっとクセがあるものの、よくできたかわいい言語です。

.vimrcには先人が書いた人気のVim scriptの文を記述できますが、多くの場合はロジックではなく、Vimの挙動を上書きする設定ファイルのような見ための文を書くことになるで

注4）厳密には、$HOME/.vim/vimrcなども許可されます。詳しくはお使いのVim内で**:h vimrc**して参照ください。

しょう。実際、複雑なロジックを記述するならば、後述のVim plugin側で行ったほうが良いです。

余談ながら、Vimの中で動作するVimのために作られたプログラミング言語の正式な名前は、「Vim script」です。JavaScriptのように「VimScript」と呼んだり、あるいはスペースを除去して「Vimscript」と呼んだり、あるいはGitHubのようにそれ以外の独自の呼び方「VimL」などと誤って呼んでしまわないよう注意してください[注5]。

さて、実際に.vimrcを編集するにあたって、まずは手元にある.vimrcファイルを覗(のぞ)いてみましょう。そう、多くのVim配布パッケージには最初からそれぞれの.vimrcが付いてきているのです。しかし、これらは前述した規定の位置ではなく、別の位置にあることが多いです。これがどこにあるのかを誌面で説明してしまっても良いのですが、せっかくですので、とっておきの技をお伝えしておきましょう。

Vimに関するほとんどすべての情報はVim内で得られます。先ほど手元で実際に確認したvimtutorで説明されているとおり、`:help`のあとに適当なキーワードを入れることで、それに関する公式のドキュメントをいつでもどこでもVimで見ることができます。今回は.vimrcについて調べたいわけですので、さっそく`:help vimrc`と入力しましょう。なお、毎回`:help<Space>`と入力するのはたいへんですから、短縮名の`:h<Space>`を用いることにして、キーボード打鍵数を50%削減しましょう。

というわけで、さっそく`:help`を活用し、最初から入っている.vimrcの位置を`:h vimrc`で確認し、そのファイルを実際に読んでみましょう。初めて読む場合、わからない部分がほとんどだと思います。そこにでてくる単語をその場で`:help`で引くことで、実用的な学びを得ることができます。

標準で入っていて、すでに動作しているVimで有効になっている設定を読むだけでなく、インターネット上に存在する多くのVim使いの方々の.vimrcファイルを見てみるのも楽しいです。GitHub上で公開されている.vimrcを輪読するオンラインの定期的イベントなどもありますので、詳しくはPart5を参照ください。このときも`:help`が大活躍します。余談ながら、著者の.vimrcも公開しています[注6]。

しかしながら、他人の.vimrcをそのまま使うのは簡単ではありません。実際、著者の.vimrcは他人が使うことを想定した作りにはなっておらず、部分的にコードの再利用をする程度をお勧めします。世には、一般に配布するための汎用の.vimrcファイルや、簡単な.vimrcファイルジェネレータなどもあります。しかし筆者としては、それらをそのまま使うのではなく、初期提供された最低限の.vimrcを基に、自分が理解できる範囲内で少しずつ機能の追加、既存機能の置き換えなどを行っていくことをお勧めします。

“初期設定”といいましたが、.vimrcファイルなどは今後ずっといじり続けることになります。自分の.vimrcを書き換えることは、俗に“Vimを育てる”と表現されます。

Vim plugin

.vimrcにkey mappingを追加するよりも、もっと大掛かりな機能追加を行うには、.vimrcに大量のVim scriptを書くのではなく、複数のVim scriptを標準のディレクトリ構成に配置したパッケージを利用します。これをVim pluginといいます。

実は、Vimは最初から、たくさんのVim pluginを含んでいます。.vimrcファイルを開い

注5) GitHub側もこの問題を認識しており、GitHub内でのVimLという誤表記を修正しようとする動きはあるものの、2016年3月現在は技術的な問題で修正作業が頓挫している模様です(http://ow.ly/ZhAqN)。

注6) http://github.com/ujihisa/config

たときに、Vim scriptがいい感じにシンタックスハイライトされていたと思いますが、それはsyntax/vim.vimで定義されています。Vim内で`:new $VIMRUNTIME/syntax/vim.vim`とすると、そのファイルを実際に見ることができます。こちらも中身はVim scriptで記述されています。

Vim pluginの役割はさまざまです。役割ごとに分類され、それぞれ特別な名前を付けたディレクトリに保存されています。前述のsyntax/vim.vimはシンタックスハイライトを提供するという役割でしたが、中身を見るまでもなくそのファイルが保存されているディレクトリ名から役割がわかります。ほかに、**表1**のようなディレクトリと役割があります。

$VIMRUNTIMEディレクトリには、これらが標準ですべて入っていますが、自分でインターネット上にあるVim pluginを導入するとき、あるいは自分で新しくVim pluginを作りたいといったときは、ほかのファイルと一緒に$VIMRUNTIMEディレクトリにばらして配置するのではなく、まとめて別個のディレクトリに保存すると整理がラクです。

Vim pluginの整理や、あるいは具体的なインストール作業を行ってくれるツールがいくつか存在します。ここでは「neobundle」[注7]というツールの使い方を紹介しましょう。GitHub

注7) https://github.com/Shougo/neobundle.vim

▼表1 Vim pluginのおもなディレクトリと役割

ディレクトリ	役割
autoload/	関数を定義するのみ。ほかのファイルからライブラリとして動的に読み込まれる
colors/	シンタックスハイライトなどに用いる色の一覧を定義した「カラースキーム」を設定する
compiler/	C言語などを書くときのコンパイラのエラーメッセージを解析してVim内で便利に使うためのもの
dict/	入力補完に用いるためのキーワードの辞書。これだけVim scriptではなく.datという形式で記述される
ftplugin/	特定ファイルタイプ(例：vim、haskell、clojure)専用の機能を提供するためのもの
plugin/	ファイルタイプに依存しない、Vim全体の挙動を変更するためのもの。:で始まるコマンドの提供もここで行う
doc/	もっとも重要なディレクトリ。Vim独自の検索機能が利用できるドキュメントを提供する

※このほか、indent/・lang/・syntax/・macros/・spell/などがある

Column

Vim plugin manager

neobundleのようなVim用のツールのことを、一般にVim plugin managerなどと言います。歴史的には「vimball」という長らく唯一の存在として君臨していたものや、RubyGemsのような気分で使えるperlによる実装の「Vimana」、インストールは行わないがディレクトリ構成のサポートだけを行う「pathogen」、そしてneobundleの前身となった「vundle」などがありました。

あまり普及はしなかったものの、独自の思想に基づいた力強いものもあり、「vim-addon-manager」「vim-plug」「VimJolts」なども根強い人気があります。neobundleは、実は2016年3月時点で開発は完了しており、作者は今「dein」というツールの開発に取り組んでいます。本誌発売時点ではまだneobundleを利用するほうが安定安心に使えますが、数ヵ月後には少し事情が異なってくるかもしれません。

のページの指示に従い、git cloneして.vimrcにneobundle用の最低限の設定を記述しましょう。vimを再起動して`:NeoBundleInstall`すれば、neobundleがVim pluginを勝手にインストールしてくれます。

はじまるVimの道

これから先のVim力の高め方の方針について。すでにそこにあるVimに合わせて自分側を改変していくのを重視するか(デフォルト教)、あるいはVimをどんどんカスタマイズして自分の手により馴染むようVim環境側を育てていくのを重視するか(カスタマイズ派)。Vimはどちらのタイプの人にも優しくできており、インストール後のデフォルトの状態でも、いきなり主戦力として使えるような設計になっていると同時に、強力かつ簡単に行えるカスタマイズのサポートがあります。今後のVim学習の大まかな方針を考えるにあたって、自分のタイプを認識すると良いかもしれません(**図1**)。

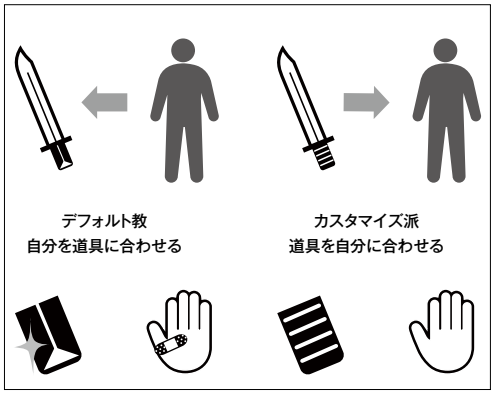
デフォルト教

なるべくカスタマイズせず、デフォルトの挙動のまま各種ツールを使うことを重視する人たちはデフォルト教[注8]などと呼ばれます。この方針には、

- カスタマイズする行為そのもの
- ツールのアップデート時の自分のカスタマイズの互換性の保持
- ツールのバグレポート時、実は自分のカスタマイズが原因であるかどうかの判別

などといった、本来の作業目的以外に煩わされないというメリットがあります。Vimの場合は(後述の理由によって)、互換性についてはそれほど考慮しなくても良いという一方、デフォルトの挙動がなかなかに悪くないという特徴があります。香り屋Vimを配布しているKoRoNさんはスパルタンVim[注9]という概念を提唱しており、人間がツール側に適応していくことの重要性を主張しています。

▼図1　デフォルト教とカスタマイズ派の違い



カスタマイズ派

著者がこのタイプです。Vimのおもしろいところは、自分で行ったカスタマイズの互換性が保たれることが重要視されるという文化にあります。カスタマイズを行うには.vimrcにVim scriptを記述することになりますが、これはほかのIDEたち——XMLやバイナリフォーマットでGUIのインターフェースから自動生成されたデータを保持することでカスタマイズを行うもの——と大きく異なります。

この性格は、Vim本体が提供する機能のキーマッピングの上書きなどに限らず、第三者が作成したVim pluginにも当てはまります。こういった背景があることから、安心してカスタマイズを行い、それに依存した"いきいきとした"生活をすることができます。

注8) 高林哲さんの「年を取ると環境設定がどうでもよくなる現象」(http://0xcc.net/blog/archives/000170.html)や、同氏による「bkノート」(https://plus.google.com/101463981287086074128)より

注9) 「スパルタンVim」(http://www.kaoriya.net/blog/2012/01/19/)。ほかにもバージョンがあり、2016年3月現在スパルタンVim 4.0まで公開済み。

余談ながら、Vim本体の進化における互換性の重視の姿勢に関しては、良くもあり悪くもありです。互換性の名の下に、一貫性のないバグにしか見えないような挙動が仕様として受け継がれてしまっている例がいくつかあります。

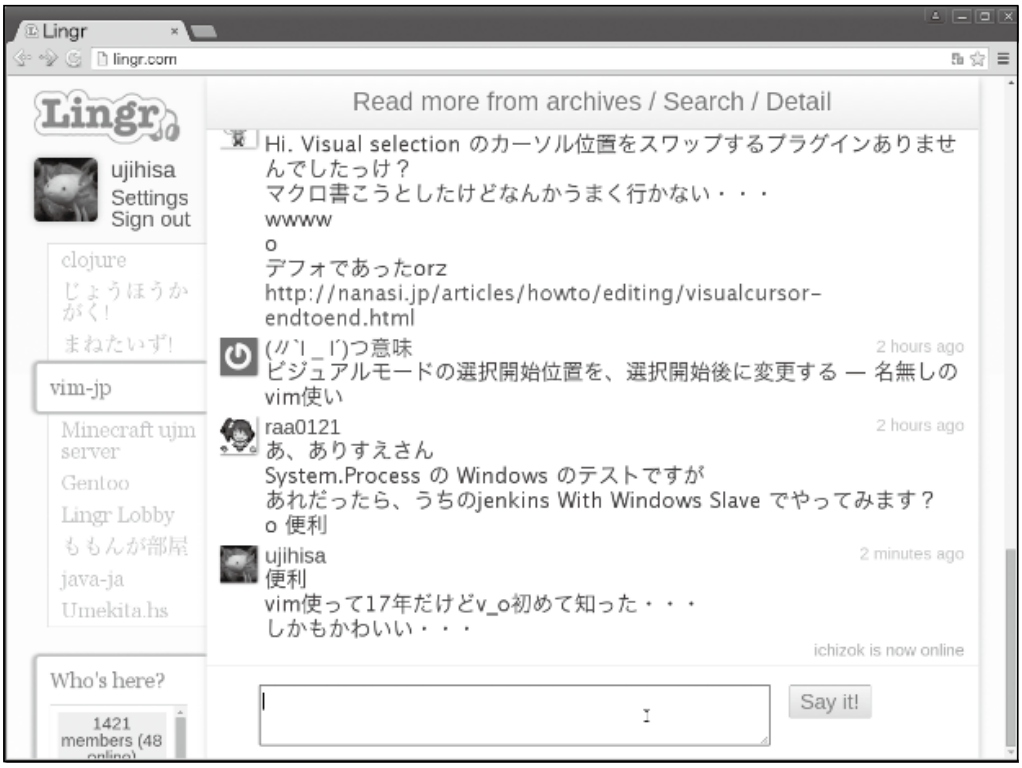
確かに、Vimのカスタマイズは本来の作業の目的そのものと合致しないかもしれませんが、そもそも本来の作業の目的がVimのカスタマイズならば問題ありません。うれしい副作用として、カスタマイズの過程でVimそのものへの理解が自然と深まっていきます。また自分のカスタマイズを一般化できそうならば、それをVim pluginとして公開することもできます。このようにしてモテるVim使いになることは、人生に大いなる彩(いろどり)を与えることになるでしょう。

◆ ◆ ◆

いずれの方針にせよ、Vimが標準で提供する機能についての理解が必要です。次章ではその具体例として、基本的な編集についてと、それらを調べるための`:help`などについて解説していきましょう。

なお、個別のVim pluginやそのエコシステムについては、時とともにゆっくりと陳腐化していくものがあります。自動補完を行うVim pluginであるneocomplcacheなどはその大きな例でしょう。neocomplcacheに置き換わるものとして開発・メンテされた自動補完プラグインであるneocompleteも、今後Neovimがメインストリームになるならば、Neovim用に開発されたdeopleteによって陳腐化されることになります。より低レイヤなライブラリに関しても、現在ほぼ必須であるvimprocが、Neovimまたはchannelⓘ(詳しくはPart5)によって陳腐化される可能性が示唆されています。

▼図2 LingrのVim部屋



これら日々変化していく情報はどのようにして把握すればいいのでしょうか。また、自らがこれらの動向に関与することはできるのでしょうか。実は簡単にできてしまいます。これらの技術の進化に関する議論や実際の開発はすべてオープンに行われており、誰でも見たり参加したりできます。日本国内のVimコミュニティで今もっとも活発に情報交換が行われているのがオンラインチャットサービスLingrにあるVim部屋[注10]でしょう(**図2**)。LingrのVim部屋への新規登録は次のURLから行えます。

・Vim部屋への登録ページ
http://goo.gl/LdPDBx(短縮URL)

注10) http://lingr.com/room/vim

おわりに

Vimとは過程です。一般にチームでのソフトウェア開発では、ほとんどの場合書いたコードを必ず相互にレビューすることでしょう。どのような過程でコードを書くかは人それぞれですが、その結果得たソースコードの品質を保つための努力は、自然と継続的に行われます。一方、その過程として用いたVim捌(さば)きそのもののレビューを受けることは、意識しない限り起こり得ません。同僚とペアプログラミングをしたり、帰宅後のプライベートのソフトウェア開発時に作業の様子をインターネットで生放送したり……、どのようにVimを使っているかを見せることで情報交換することは非常に有意義です。SD

Column

Vimのためのキーマップを考える

Vimの標準のKey mappingには、Ctrlを用いたものも多くあります。Insert modeでかなり多用されるほか、Normal modeでもVim内Windowの移動に**<C-w>**がプリフィックスとして使われるなど、Vimの幅広い操作に慣れれば慣れるほどCtrlの利用率が高まることでしょう。一般的なUS配列のキーボードでは、Ctrlはキーボードの一番左下、左SHIFTの下というとんでもない僻地に配置されています。これをそのまま使うよりは、素直にAの左に配置されているCaps Lockを潰してCtrlに割り当てましょう。これはVim内ではなく、その外側のレイヤで行いましょう。LinuxではX.orgの設定やautokeyで、Macではは Karabiner(旧 KeyRemap4MacBook)で、Windowsではautohotkeyなどのツールを用いて再割り当てしましょう。なお、JIS配列のキーボードの一部は最初からCtrlがAの左に配置されているため、安心してデフォルトのまま快適なVimライフを送ることができます。

Profile

氏久 達博(ujihisa)

・カナダのバンクーバーでHootsuite Media Incに勤務し、Vimで Scalaなどを書くソフトウェア開発者
・世界最大規模のVimのカンファレンスであるVimConfの発起人、運営者の1人
・Vim pluginのための汎用Vim scriptライブラリ「vital.vim」などのメンテナ

Part 2

Vimだからできる、一歩先行く編集術

Author thinca URL https://github.com/thinca Twitter @thinca

本章では、ほかのテキストエディタにはあまりない、矩形選択・モーションとオペレータ・テキストオブジェクト・ドットコマンド・gnコマンドといった、Vimならではの編集機能について見ていきます。どの機能も強力ですので、使いこなせれば編集効率がアップするのは間違いないでしょう。

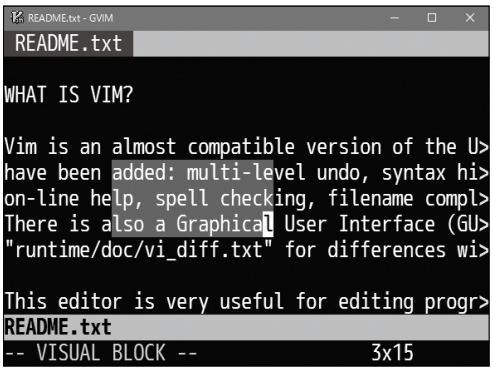
矩形選択

Vimのモードの1つにビジュアルモードというものがあり、Vimの持つ多彩なカーソル移動コマンドを使用してテキストを選択できます。ビジュアルモードには、文字単位選択、行単位選択、そして矩形選択の3つの選択方式があります。この節では矩形選択の便利な使い方について紹介していこうと思います。

矩形選択とは何か

矩形選択は、テキストを連続した文字の単位ではなく、行を跨いだ四角形の範囲でテキストを選択する機能です。テキストエディタとしては定番の機能と言えるでしょう。Vimの場合は、ノーマルモードから<C-v>を入力することで矩形選択モードに入れます(図1)。

▼図1 矩形選択



Vimの矩形選択は、ほかの選択方式と同様にdによる削除やyによるコピーはもちろん、一般的なテキストエディタではあまり見ない機能も備えています。

列にまとめて文字列を追加

矩形選択中にIやAを押すと、それぞれ範囲内の一番上の行の先頭や末尾にカーソルが移動して挿入モードになります。この状態で文字列を入力して挿入モードを抜けることで、選択範囲の同じ列にまとめて同じ文字列を入力できます(図2〜4)。またこのとき、$で行の末尾までの範囲を矩形選択で指定してからAを使うことで、すべての行の末尾に文字列を挿入することもできます。

Iでの挿入とAでの挿入には、選択範囲内に範囲に届いていない短い行がある場合に、微妙な違いがあります。Iの場合は短い行に対して何も行われないのに対し、Aの場合は同じ位置に文字が挿入されるように空白が挿入されます(図5〜7)。

文字をまとめて変更

cを押すことで、選択範囲の削除とAによる挿入が一度に行えます(図8〜10)。

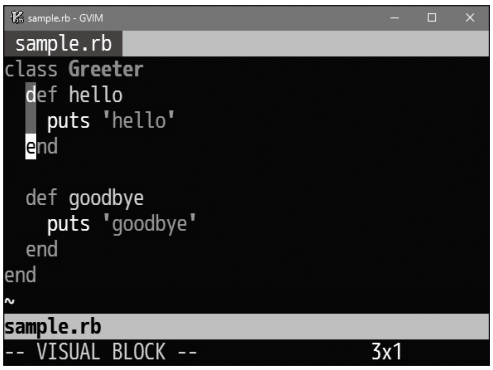
選択範囲内だけを検索／置換

選択範囲内だけを置換したい場合、行単位

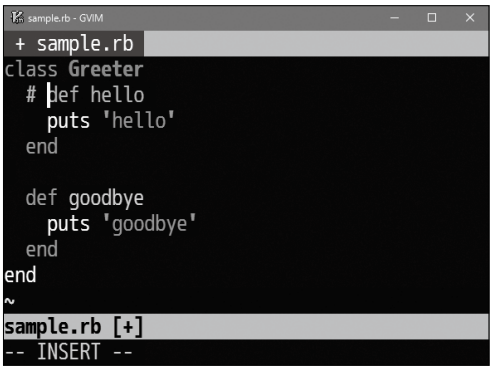
であれば、ビジュアルモードからそのまま**:**を押すことで、**:'<,'>**という入力となり、ビジュアルモードで選択した行のみの範囲を指定したことになります。しかしこれは行単位であり、矩形選択の範囲の場合はこの方法は使えません。

そこで、**\%V**という正規表現を使用します。これは選択範囲内にのみマッチする特別な正規表現で、検索パターンを**\%Vabc\%V**のように**\%V**で囲うことで、検索範囲内のパターンにのみマッチさせることができます。

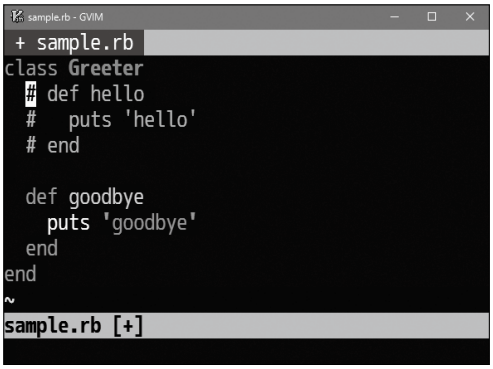
▼図2　矩形選択



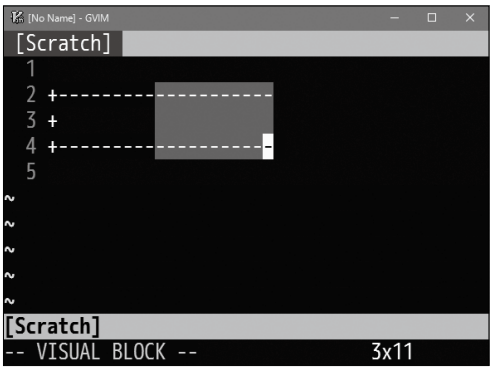
▼図3　Iで挿入モードに入り文字を入力



▼図4　<Esc>で抜けると文字列が挿入される



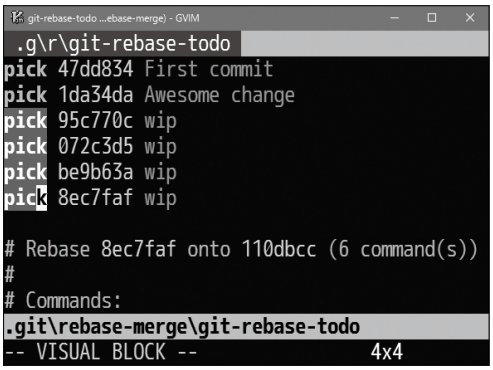
▼図5　短い行を含む範囲を矩形選択



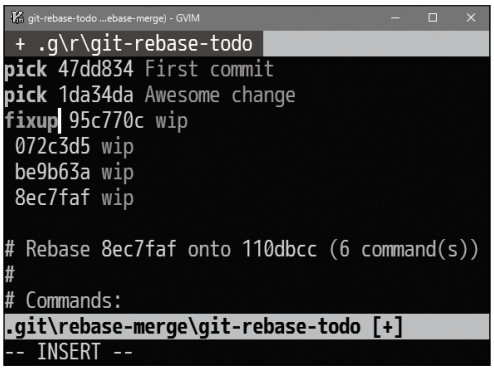
▼図6　Iで挿入した場合

▼図7　Aで挿入した場合

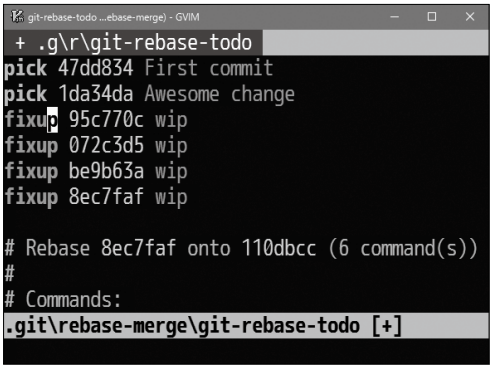
▼図8　pickをまとめて矩形選択



▼図9　cを押してからfixupを入力



▼図10　<Esc>で抜けると、すべてfixupに変換される



モーションとオペレータ

モーションとオペレータはVimを使ううえで重要な概念です。正しく理解して使いこなしましょう。

モーションとは何か

モーションとは、移動コマンドのことです。お馴染みの`h j k l`や、`w $ %`、果ては検索コマンド`/`も、カーソルが移動するものは基本的にモーションと考えてかまいません。モーションコマンドにはものすごく多くの種類があるため、ここですべては解説しません。

オペレータとは何か

オペレータは、後にモーションを続けて入力することで、モーションによってカーソルが移動する範囲に対して操作を行うことができます。言葉だけで説明してもわかりにくいので、例を使って説明します。次のようなテキストがあるとします(反転部分がカーソル位置)。

```
Vim is a difficult text editor.
```

`d`は削除を行うオペレータです。`w`は次の単語の先頭まで移動するモーションです。ここで`d`を押し、続けて`w`を押すと、次のようになります。

```
Vim is a text editor.
```

これで、「単語を削除する」という操作を行うことができました。

組み合わせによるメリット

なぜモーションとオペレータに分かれているのでしょうか。続けて例を見てみましょう。先ほどの編集前のテキストに、同じオペレータ`d`と、今度は別のモーション、行末まで移動する`$`を適用してみましょう。

```
Vim is a
```

「行末まで削除する」操作が行われました。

また最初のテキストに戻り、今度はモーションwはそのままにして、dの代わりに別のオペレータであるgUを使った場合を見てみましょう。gUは、アルファベットを大文字にするオペレータです。

```
Vim is a DIFFICULT text editor.
```

削除されずに、単語が大文字になりました。

さて、それでは「行末まで」の「アルファベットを大文字に」したいとすれば、どうすればいいでしょうか。あなたはもうその方法を知っているはずです。新しいコマンドを覚える必要はいっさいありません。gU$と入力します。

```
Vim is a DIFFICULT TEXT EDITOR.
```

「行末まで」の「アルファベットを大文字にする」ことができました。

このようにモーションとオペレータが分かれていることで、その組み合わせによってできることが増えていきます。あなたが新しいモーションやオペレータを覚えれば、あなたの使える操作は乗算式に増えていくのです。

ビジュアルモードでオペレータを使う

ビジュアルモードでオペレータを実行すると、選択範囲に対してオペレータの操作を実行できます。vでビジュアルモードに入り、モーションを使って選択範囲を指定、最後にオペレータのキーを押すだけです。自由に対象を指定できるので、単一のモーションでは対応できないような場合に使うと良いでしょう。

これだけ聞くと、常にビジュアルモードを使用すれば複雑なオペレータ＋モーションというしくみは必要ないのではないか、と思う方もいるでしょう。確かにそれで済む場合もありますが、オペレータとモーションの組み合わせにはもう1つ大きなメリットがあります。それは後述する「ドットコマンド」の節で解説します。

テキストオブジェクト

オペレータは、続けてモーションを入力することでオペレータの対象を指定できました。このとき、モーションの代わりにテキストオブジェクトというものを使うこともできます。

テキストオブジェクトとは何か

テキストオブジェクトを使うと、操作対象を論理的に意味のある単位で指定できます。

たとえば、前述の単語を表すモーションwは次の単語へ移動するものでした。ただし、カーソルが単語の先頭にない場合でも、カーソルは次の単語の先頭へ移動します。モーションで操作対象を指定した場合、それは現在のカーソル位置から移動先のカーソル位置の範囲になるので、単語の途中からの操作になってしまいます。

```
Vim is a difficult text editor.
↓ dw
Vim is a difftext editor.
```

しかしそうではなく、単語内のどこにカーソルがあっても「現在の単語」を対象として指定できると便利でしょう。それを可能にするのがテキストオブジェクトです。

テキストオブジェクトの使い方

テキストオブジェクトは基本的にaもしくはiから始まります。先ほど例に挙げた単語を表すテキストオブジェクトは、awとiwになります。それぞれ、境界を含むか含まないかの違いがあり、境界が何になるかはテキストオブジェクトによって異なります。両者の違いがない

場合もあるなど厳密なルールがあるわけではなく、各テキストオブジェクトは「aは境界を含む／iは境界を含まない」という方針で定義されている、と考えておけば良いでしょう。

あらためてawとiwについて見てみましょう。これら2つのテキストオブジェクトの境界は、周囲の空白文字になります。具体的には、それぞれ次のような範囲になります。

```
          iw
       ┌─────┐
Vim is a difficult text editor.
       └─────┘
          aw
```

たとえば、単語を削除したい場合は、dawと入力することで実現できます。

```
Vim is a text editor.
```

別の例を挙げましょう。cを使って単語を別のものに書き換えたい場合は、ciwが使えます。たとえば、ciweasy<Esc>のようにします。

```
Vim is a easy text editor.
```

また、テキストオブジェクトはノーマルモードでのモーションとしてカーソルを移動することには使えませんが、ビジュアルモードで使用することで対象を選択できます。

どんなテキストオブジェクトがあるのか

表1によく使うテキストオブジェクトを掲載します。これがすべてではありません。詳しく知りたいときは、:help text-objectsを見てみてください。

ドットコマンド

ドットコマンドは、ドットリピートと呼ばれることもあります。一度使い方を覚えれば、手放せなくなる便利な機能です。

ドットコマンドとは

ドットコマンドは、ドットキー「.」を押すことで、最後に行った変更を繰り返す機能です。具体的にどのような動作をするのか、例を見てみましょう。次のようなテキストがあります。

```
Vim is a very simple text editor.
```

ここで、dawを実行してみましょう。

```
Vim is a simple text editor.
```

dは削除、awは単語を表すテキストオブジェクトですので、「単語を削除する」コマンドが実行されました。

ここで次に、textを消したいとしましょう。

▼表1　おもなテキストオブジェクト

キー入力	対象	境界	例
aw / iw	単語	周囲の空白	word
aW / iW	WORD（非空白文字の連続）	周囲の空白	foo,bar
a(,a),ab / i(,i),ib	丸括弧で囲まれた範囲	()	(foo)
a{,a},aB / i{,i},iB	波括弧で囲まれた範囲	{}	{foo}
a[,a] / i[,i]	角括弧で囲まれた範囲	[]	[foo]
a<,a> / i<,i>	山括弧で囲まれた範囲	<>	＜foo＞
a" / i"	ダブルクォートで囲まれた範囲	""	"foo"
a' / i'	シングルクォートで囲まれた範囲	''	'foo'
at / it	htmlタグで囲まれた範囲	<tag></tag>	＜tag＞foo＜/tag＞

そのためには、まずwでカーソルをtextの上に持っていきます。次に、同じようにdawを実行しても良いですが、これは先ほどと同じコマンドです。ここでdawの代わりに.を押します。

```
Vim is a simple editor.
```

無事、現在のカーソル位置に対して「単語を削除する」コマンドが再実行されました。これがドットコマンドです。

操作の単位について

ドットコマンドで繰り返されるのは、最後に行ったテキストを変更する操作になります。前述のオペレータの実行もそうですし、挿入モードでのテキストの挿入やpでのテキストの貼り付け、ほかにも<C-a>／<C-x>による数値の変更や~による大文字小文字の変換なども含まれます。

ここで、Vimのコマンドの豊富さに関する話をしましょう。Vimには一見、既存のコマンドを組み合わせれば不要となるように思え

Column オペレータやテキストオブジェクトはプラグインで増やせる

オペレータやテキストオブジェクトは、プラグインや自作で増やすこともできます。次にプラグインの例を挙げます。気になるものがあったらインストールしてみてください。

表Aはオペレータプラグイン、**表B**はテキストオブジェクトプラグインです。中でも「operator-user」「textobj-user」はフレームワークとなっており、これに依存しているプラグインもあるので、インストールしておくと良いでしょう。

▼表A　オペレータプラグイン

プラグイン名	操作	GitHubのページ	作者（敬称略）
operator-user	オペレータを簡単に定義するためのフレームワーク	kana/vim-operator-user	kana
operator-replace	対象をレジスタの中身で置き換える	kana/vim-operator-replace	kana
operator-surround	テキストオブジェクトの境界部分自体を追加／置き換え／削除する	rhysd/vim-operator-surround	rhysd
operator-camelize	snake_case と CamelCase を相互に変換する	tyru/operator-camelize.vim	tyru

▼表B　テキストオブジェクトプラグイン

プラグイン名	操作	GitHubのページ	作者（敬称略）
textobj-user	テキストオブジェクトを簡単に定義するためのフレームワーク	kana/vim-textobj-user	kana
textobj-indent	同じインデントレベルの範囲を扱う	kana/vim-textobj-indent	kana
textobj-entire	バッファ全体を扱う	kana/vim-textobj-entire	kana
textobj-line	改行を含まない行全体を扱う	kana/vim-textobj-line	kana
textobj-parameter	関数の引数部分を扱う	sgur/vim-textobj-parameter	sgur
textobj-between	任意の指定した文字で囲まれた範囲を扱う	thinca/vim-textobj-between	thinca

るコマンドがあります。これらが提供されているのは、単純によく使われるからという理由だけではなく、別のメリットがあるのです。

たとえば、テキストを変更するcオペレータは、実行すると対象範囲を削除したあと挿入モードに移行し、代わりのテキストを挿入できます。これは一見するとdによる削除を行ったあとにiでテキストを挿入すれば済みそうに思えます。しかし、実際にはそうではありません。cは「対象を削除し、代わりのテキストを挿入する」オペレータであり、.で繰り返す場合もそのように動作します。

実際に例を見てみましょう。次のようなテキストがあります。

```
var one = 1;
var two = 2;
var three = 3;
```

テキスト中のvarをletに置き換えたいとします。矩形選択、一括置換コマンド……、Vimで行うならば、いくつか方法が考えられます。今回はドットコマンドを使ってみましょう。

最初のvarにカーソルを持っていき、ciwlet<Esc>と入力します。

```
let one = 1;
var two = 2;
var three = 3;
```

「単語をletに変更する」コマンドが実行され、最初のvarが置き換わりました。残りのvarにも適用するために、j.j.と入力します。

```
let one = 1;
let two = 2;
let three = 3;
```

無事、目的を達成できました。削除と挿入を別々に実行していた場合、これは.で繰り返すことはできませんでした。

ほかにも、挿入モードに入るコマンドとしてIやAがあり、これらを使うと行頭／行末から挿入モードに入ることができますが、これらを.で繰り返すことで、複数の行の行頭／行末に同じテキストを挿入していくことが簡単にできます。

ビジュアルモードとドットコマンド

オペレータがビジュアルモードでも利用できることはすでに説明しました。これは便利ではあるのですが、通常のオペレータとモーションを使って実行した場合とは、実行されるコマンドが異なります。

例を見てみましょう。次のようなテキストがあります。

```
let nums = ["zero", "one", "two"]
```

ここで、各文字列を空文字にしたいとします。ビジュアルモードで削除してみます。vi"dですね。

```
let nums = [", "one", "two"]
```

zeroが選択され、削除されました。ここで、oneの先頭にカーソルを持っていき、.で繰り返してみましょう。

```
let nums = ["", "", "two"]
```

"が1つ余計に消えてしまいました。

実は、ビジュアルモードでは文字数で操作対象が記録されています。つまり、最初の操作は「4文字消す」というように記録されていました。よって、.での繰り返しも「4文字消す」操作が行われ、"が1つ余計に消えてしまったというわけです。

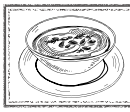
この例では、ビジュアルモードを使わずにdi"と操作すれば、「"の内側を消す」操作となり、繰り返しがうまくいきます。繰り返しを利用したい場合は、ビジュアルモードでの挙動に注意する必要があります。

gnコマンド

ちょっと変わった便利機能であるgnコマンドを紹介したいと思います。この機能は、Vim 7.3.610で追加されました。2012年7月にリリースされているので、最近のVimであれば利用可能だと思います。

gnは、検索の次のマッチへ移動するnの亜種です。ノーマルモードで使うと、次のマッチへ移動し、対象をビジュアルモードで選択します。これだけだと大したことはないのですが、このコマンドはオペレータに続けて使うと、テキストオブジェクトのように動作します。つまり、次の検索のマッチに対してオペレータを実行できるのです。

たとえば、「操作の単位について」で挙げたコード例では、まず/varで検索を行い、cgnvar<ESC>で「次のマッチをletに変換する」コマンドを実行すれば、あとは..と繰り返すだけで終わらせられます。カーソルを動かす必要はありません。ちょっとしたことですが、とても便利ですのでぜひ使ってみてください。



まだまだある便利な機能

誌面の都合上詳しく紹介ができませんが、Vimにはほかにも Vimならではの機能があります。いくつか触りだけ紹介します。

マクロ

.では直前の操作を繰り返せましたが、もっと長い操作を繰り返したい場合には、一般的にマクロと呼ばれる機能を利用できます。

qに続けて任意のアルファベットを入力すると、記録モードに入ります。その状態で操作を行い、最後に再びqを押すことで、アルファベットに対して操作を記録できます。

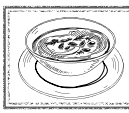
記録した操作を実行したい場合は、@に続けて記録したアルファベットを入力するだけです。記録している最中に行ったキーボード操作がそのまま実行されます。あらゆる操作をキーボードのみで行える、Vimらしい機能だと言えます。

この機能は非常に強力ですが、慣れるまでは使い道に戸惑うかと思います。ここでは紹介しきれませんので、使用例や応用方法などを調べてみると良いでしょう。

数値の増減

<C-a>と<C-x>を使うと、カーソル下もしくはカーソル以降にある数値の値を増減できます。事前に数値を入力してから使うことで、たとえば5<C-a>のように入力すれば、数値を「5」増やすといったこともできます。

この機能はドットコマンドやマクロと非常に相性が良いです。意外に使えるシーンはあるので、覚えておくと便利です。



少しずつ身に付けよう

いかがでしたでしょうか。Vimには本当に多くの機能があり、ここで紹介したのはほんの一部にすぎません。いきなりすべてを使いこなすのは難しいですが、少しずつ物にしていけば大丈夫です。自分のペースで身に付けていくと良いでしょう。SD

Profile

thinca

・多数の Vim プラグインを開発、公開している。代表作に quickrun.vim など
・vimrc 読書会主催　http://vim-jp.org/reading-vimrc/
・C# 開発環境の OmniSharp の Vim 向けプラグイン「omnisharp.vim」のメンテナ

Part 3

Vimの強力な正規表現を使いこなそう

Author tyru URL https://github.com/tyru Twitter @_tyru_

Vimでの編集作業を効率良くするには「検索」と「置換」がカギを握っています。そしてこの検索に大きくかかわっているのが本章で取り上げる「正規表現」です。正規表現の強力なパターンマッチングの書き方を覚えれば、いろいろなところで応用も利きます。

Vimと正規表現

Vimで正規表現を扱う一番多い操作は検索です。Vim以外のエディタでは検索といえば入力した文字列をそのまま検索しますが、Vimの検索コマンドである**/**や**?**は正規表現のパターンを入力しなければなりません。よって正規表現を知らないと「なぜ入力した文字列が検索にヒットしないのかわからない」となってしまいます(ちなみに正規表現ではなく文字列で検索するには、後述する**\V**を使います)。

では正規表現を習得しようとなったとき、先に気をつけておかなければならないことがあります。正規表現にはさまざまなツールやプログラミング言語などによって、微妙な差異(方言)があるという点です。**図1**のものはその一例です。

現在多くのプログラミング言語やツールでは後者の拡張正規表現がデフォルトの正規表現になっています。そのためVimデフォルトの基本正規表現は慣れない方もいるでしょう(ちなみに完全ではありませんが、**\v**をパターンの冒頭に付けることでVimでも拡張正規表現が使えます)。

Vimでの拡張正規表現の表記例

```
\v(V(im|isual Studio Code)|E(macs|clipse))
```

ここまで聞くと正規表現はなんて面倒なんだろうと思うでしょう。確かに正規表現には初心者がつまずきやすい個所がたくさんあります。しかし、正規表現によるファイルをまたぐ検索と一括置換を実際に経験してしまうと、逆になぜ使わずに作業できていたのか不思議に思うほどの恩恵にあずかれます。例としてVimの引数で指定されたファイルにある、すべての**pattern**を**string**に置き換える(一括置換する)には、コマンドラインで**図2**のように実行します。

筆者は実は仕事ではVimを使っていません。しかし正規表現はVimに限らずさまざまなエディタ、ツール、プログラム、日常的なシェ

▼図1　正規表現の方言の例

```
●検索対象：「Vim」「Visual Studio Code」「Emacs」「Eclipse」のいずれか
基本正規表現(Vim, grep)      \(V\(im\|isual Studio Code\)\|E\(macs\|clipse\)\)
拡張正規表現(Perl, grep -E)  (V(im|isual Studio Code)|E(macs|clipse))
```

▼図2　複数ファイルの一括置換の書き方例

```
$ vim -c 'argdo %s/pattern/string/gce | update' {file1} {file2} ...
```

ルスクリプトやPowerShellスクリプト、OSを問わずどこでも使えます。そして正規表現を身に付ける一番の近道は、**日常的に使うエディタで実際にトライ＆エラーを繰り返すことです**。正規表現を身に付ければ新人の方に限らず確実に1つの武器になるでしょう。

本記事はリファレンス的に使えるよう、わかりやすい見出しでまとめてありますので、皆様の正規表現への理解に少しでも力になれば幸いです。またできればSoftware Design 2016年4月号の連載「Vimの細道」の「第7回 Vimの正規表現をマスターする」も併せてご覧ください。

Vim正規表現のハイライト

まず正規表現による検索を行う前に、次の設定を.vimrcに書いておくと便利です(.vimrcの場所については氏久氏によるPart1を参照してください)。

```
set hlsearch
set incsearch
```

それぞれhlsearchオプションはハイライトを有効、incsearchオプションはパターンを入力中に次にマッチするテキストをハイライトします[注1]。

Vim正規表現の方言

冒頭でVimの正規表現には方言があると述べました。この方言を緩和するため、**初心者の方はまずパターンの最初に\vを付けて検索することをお勧めします**。\vを含めないほうが場合によっては見やすくなりますが、そのほうがほかの正規表現エンジンで正規表現を書く際に混乱することが少なくなるからです(「正規表現エンジン」とは正規表現の各実装のことです)。ちなみに\vを付けない場合は、次のように+や()といったメタ文字(正規表現での特殊な文字)の前にバックスラッシュを付ける必要があります。

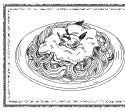
```
\vを付ける場合   \vVim?
\vを付けない場合  Vim\?

\vを付ける場合   \v(GitHub|BitBucket)
\vを付けない場合  \(GitHub\|BitBucket\)
```

詳しい違いについては :help /\v を参照してください。本記事では**\vを付けると意味が違う場合のみ**比較のため両方記載することにします。また上記を読んで\vを付けようと決めた方は、次のようなマッピングを.vimrcに書いておくと便利かもしれません。

```
nnoremap / /\v
nnoremap ? ?\v
```

これで常に\vを含めた状態で検索パターンを入力することができます。

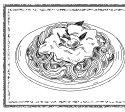


初めての正規表現

さて、前置きが長かったですが、実際に正規表現のパターンを書いてみましょう。「abc」という文字列にマッチさせたい場合はそのまま「abc」と書きます。

```
●検索対象：abc
パターン  abc
```

こういった正規表現の特殊ではない文字を「リテラル」と呼びます。



行頭・行末^、$

先頭にある文字をマッチさせたい場合は^を使います。

注1) incsearch.vim(https://github.com/haya14busa/incsearch.vim)や prompter.vim(https://github.com/mattn/vim-prompter)というプラグインを使うことにより、リアルタイムで画面上のすべての文字列がハイライトされるようにもできます。

●検索対象：先頭に「#」が付いた行
パターン　^#

逆に行末にマッチさせたい場合は$を使います。

●検索対象：行末に「;」が付いた行
パターン　;$

量指定子：繰り返し *、 +

パターンの繰り返しには*または+を使います。*は0個以上の繰り返し、+は1個以上の繰り返しを表します。\vを付けない場合は*または\+と書きます。

●検索対象：String str
パターン（\vあり）　\vString\s+str
パターン（\vなし）　String\s\+str

こういった\sや+のような特殊な文字を「メタ文字」と呼びます。また*や+のようなマッチの個数を指定するメタ文字を「量指定子」と呼びます。まず冒頭で述べた\vを先頭に付けています。次に\sは空白文字を表します。\sはVimでは半角スペースとタブ文字にマッチします。\s+と書くことで「半角スペースかタブ文字の1文字以上の繰り返し」という意味になります。

量指定子：0個または1個 ?、 =

?または=は0個または1個の量指定子です。

●検索対象：「function foo(」
　　または「function(」
パターン（\vあり）
```
\vfunction(\s+\w+)?\(
```
パターン（\vなし）
```
function\(\s\+\w\+\)\?\(
```

Vim以外の正規表現エンジンでは?と表されることが多いですが、Vimでは=も?と同じ意味のメタ文字です。

量指定子：回数指定 {n}、{n,m}

回数指定の繰り返しには{n}や{n,m}を使います。次のものは少し複雑な例ですが、日付にマッチする正規表現です。

●検索対象：2016/01/31
パターン（\vあり）
```
\v\d{4}\/\d{2}\/\d{2}
```
パターン（\vなし）
```
\d\{4}\/\d\{2}\/\d\{2}
```
↑「{」の前にバックスラッシュを付ける

ちなみに\dは正規表現エンジンによっては使えない場合もあるので、そういう場合は文字クラスを使い[0-9]と書きます（文字クラスについては「文字クラス」の節で解説）。しかし検索するためにわざわざ長いパターンを入力するのも面倒です。Vimでは\dが使えるのですからどんどん使っていきましょう。

また次節の「/のエスケープ」で述べますが、/を検索するためにバックスラッシュでエスケープしています。

上記の正規表現では「2016/1/31」のような日付にマッチしません。なぜなら月の部分が2文字の数値ではないからです。月や日が0埋めされない場合も考慮するなら次のように書きます。

●検索対象：2016/1/31
パターン（\vあり）
```
\v\d{4}\/\d{1,2}\/\d{1,2}
```
パターン（\vなし）
```
\d\{4}\/\d\{1,2}\/\d\{1,2}
```

\d{1,2}とすることで、1文字から2文字までの数字（\d）にマッチします（1が最小の繰り返し回数、2が最大の繰り返し回数です）。ちなみに先ほど述べた*と+はそれぞれ{0,}、{1,}のように書くこともできます。このように最小の繰り返し回数や最大の繰り返し回数を省略することもできます。

/のエスケープ\/

文字の/を検索したい場合は\/のようにエスケープする必要があります。これは正規表現というよりVimの/コマンドと?コマンドの仕様によるものです。本記事では解説しませんので、詳しくは:help search-offsetを参照してください。

ちなみにSoftware Design 2016年4月号の記事「Vimの正規表現をマスターする」にもありましたが、次のようなマッピングを.vimrcに書いておくと自動でエスケープされて便利かもしれません。

```
cnoremap <expr> / getcmdtype() == '/' ? ↵
'\/' : '/'
```

しかし自動エスケープを好まない人もいるため、お好みで設定してみてください。エスケープの具体例については「量指定子：回数指定」の解説を参照してください。

固定文字列を検索 \V

正規表現のメタ文字を無視して固定文字列で検索したいとしましょう。その場合は\Vをパターンの先頭に付けることで固定文字列検索が可能です(\vと逆の意味を持つため\vは指定しません)。

●検索対象：1*2+3
パターン \V1*2+3

しかし\Vを含めた場合でも/と\だけはエスケープする必要があるので注意してください。

大文字と小文字のケースを無視・区別する\c、\C

Vimで大文字と小文字のケースを無視するには\cをパターンに含めます。逆に無視しないようにするには\Cをパターンに含めます。どちらも含めない場合はignorecaseオプション、smartcaseオプションの値によって決められます。ignorecaseオプションがオンだとケースを無視しますが、smartcaseオプションがオンだと大文字が含まれる場合はignorecaseオプションがオフのように振る舞います。**表1**にそれぞれのオプション値がオン／オフだった場合の挙動を記載します(Vimのhelpからの引用です)。

表1のとおり、\cと\Cは\vと違い、パターンのどこに含めても全体で有効になります。\vは含めた個所以降にのみ効果が現れます。ちなみにignorecase、smartcase、\c、\Cは、いずれも文字クラス内部には影響しません。文字クラスについては次節で解説します。

文字クラス

a、b、cのいずれかの文字にマッチさせたい場合は[abc]のように書きます。この記法は「文字クラス」と呼びます。そのほかにも文字クラス内ではさまざまな指定ができます。いくつか分けて紹介しましょう。

▼表1　ignorecase、smartcaseの有効／無効による挙動

パターン	ignorecase	smartcase	マッチするもの
foo	オフ	-	foo
foo	オン	-	foo, Foo, FOO
Foo	オン	オフ	foo, Foo, FOO
Foo	オン	オン	Foo
\cfoo	-	-	foo, Foo, FOO
foo\C	-	-	foo

文字クラス：範囲[x-y]

文字コードのある特定の範囲にマッチさせたい場合があります。たとえば数字を表す場合は[0-9]と書きますが、これは文字コード上で0～9の文字が連続しているためこのように書くことができます。ほかにも16進数の文字(0～9、A～F、a～f)にマッチする正規表現は[0-9A-Fa-f]のように書くことができます。このように複数の範囲を含めることもできます(ちなみにVimでは16進数は文字クラスを使う代わりに\xと書いても構いません)。

文字クラスに-(ハイフン)を含めたい場合はどうすればいいでしょうか？　いくつか方法があります。1つは\でエスケープする方法です。

●検索対象：数字またはハイフン
　　または「A」から「F」までの文字
パターン　[0-9\-A-F]

もう1つは-を文字クラスの先頭か最後に持ってくる方法です。

●検索対象：数字またはハイフン
パターン　[-0-9] あるいは [0-9-]

文字クラス：否定[^...]

たとえば、バッファにマルチバイト文字が含まれているかどうかを検索したい場合はどうすればいいでしょうか？　ASCIIコードに含まれない範囲(1～255)の文字と考えると次のように書けます。

●検索対象：マルチバイト文字
パターン　[^\x01-\x7f]

文字クラス：POSIXブラケット表現[:...:]

本記事でいくつか出てきた\sですが、これは[[:space:]]というふうにも書くことができます。ややこしい書き方ですが、\sは文字クラス内では\とsを表しますが、[[:space:]]の場合はちゃんと半角スペースとタブ文字として扱われます。

```
[\s]　これは\とsにマッチしてしまう!
[[:space:]]
 ↑これは半角スペースとタブ文字にマッチする
```

上記の例だと単純に\sと書けばいいのですが、POSIXブラケット表現を使うことでほかの文字と合わせて使えるという利点があります(また文字クラスの否定などとも組み合わせられます)。

●検索対象：半角スペース、タブ文字、
　　数字のいずれか1文字
パターン　[[:space:]0-9]

このように角カッコで囲んだ文字クラス表現を「POSIXブラケット表現」と呼びます。

文字クラス：メタ文字や]のエスケープ

*や+や.などのメタ文字は文字クラス内だと特殊な文字とはみなされません。

●検索対象：「*」「+」「.」のいずれか1文字
パターン　[*+.]

また文字クラス内に]を含めたい場合は、終了の]とみなされないようにエスケープする必要があります。

●検索対象：「(」「)」「[」「]」の
　　いずれか1文字
パターン　[()[\]]

選択子 foo|bar|...

文字クラスを使うといずれか1文字にマッチさせることができますが、1文字ではなく2文字以上のいずれかの文字列にマッチさせたい場合には|を使います。

●検索対象：fooまたはbar
パターン（\vあり）
```
\vfoo|bar
```
パターン（\vなし）
```
foo\|bar
```

グループ（...）

(...)で囲むことでひとまとまりにできます。これは実際に例を見たほうが早いでしょう。次のように選択子(|)と組み合わされる場合が多いです。

●検索対象：「foo =」または「bar =」
パターン（\vあり）
パターン（\vあり）：\v(foo|bar)\s*\=
パターン（\vなし）
パターン（\vなし）：\(foo\|bar\)\s*=
「(」「)」「|」の前にバックスラッシュを付ける、「=」はメタ文字と解釈されないようそのまま入力

=および\=はVimではメタ文字のためエスケープしていることに注意してください。また|は^より優先順位が低いため、^foo|barとすると「先頭にあるfoo、または（先頭にあるとは限らない）bar」という意味になってしまいます。barも先頭にあるものとしてマッチさせたければ^(foo|bar)と書きます。このような優先順位の落とし穴もあるため、選択子を使う場合はグループと併せて使うことが多いでしょう。

メタ文字のエスケープ\

ここまでいくつかメタ文字を紹介しましたが、文章中にあるメタ文字を検索したい場合はバックスラッシュを付けてエスケープする必要があります。またバックスラッシュ自身を検索したい場合もエスケープする必要があります。

●検索対象：\s+
パターン（\vあり）
```
\v\\s\+
```
パターン（\vなし）
```
\\s+
```

●検索対象：/etc/hosts
パターン
```
\/etc\/hosts
```

●検索対象：C:\Windows\system32
パターン
```
C:\\Windows\\system32
```

しかし\vの有無でバックスラッシュがエスケープになったり別の意味に変わってしまうので注意してください（詳しくは:help /\v）。

単語の境界<...>

たとえばhostsやhostnameではなくhostのみを検索したい場合はどうすればいいでしょうか？　そういう場合には単語検索を使います。

●検索対象：host
パターン（\vあり）
```
\v<host>
```
パターン（\vなし）
```
\<host\>
```
「<」と「>」の前にバックスラッシュを付ける

<と>は単語の境界にマッチします。単語とはVimでは「iskeywordオプションに含まれる文字で構成される1文字以上の文字列」という定義になっていますが、だいたいの正規表現エンジンでは次のような定義になっています。

・a〜z（小文字）
・A〜Z（大文字）
・0〜9（数字）
・_（アンダーバー）

これは\wまたは文字クラスを使って、[0-9A-Za-z_]のように表すことができます。よってhttp://localhost/というテキストがあるとき、単語の境界とは次の|で表した部分を意味します（単語の境界は\v<.|.>というパターンで検索すれば確認できます）。

```
|http|://|localhost|/
```

また、hostsやhostnameにはマッチさせたくないけど、localhostやroyalhostなどhostの前に単語が付けられている文字列にマッチさせたい場合は次のように書きます。

●検索対象：localhostやroyalhost
パターン(\vあり) \v\w+host>
パターン(\vなし) \w\+host\>

\wは先ほど紹介した単語([0-9A-Za-z_])を表すメタ文字です。このように単語境界は片方だけにも付けることができます。実際にVim以外の正規表現エンジンでは括弧で表されず、\bを使って\b...\bのように表されることが多いです。

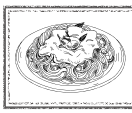
短縮されたすべての候補にマッチする%[...]

このメタ文字はVimの独自機能です。これに関しては例を見てもらったほうがわかりやすいでしょう。

●検索対象：testまたはtesまたはteまたはt
パターン(\vあり) \vt%[est]
パターン(\vなし) t\%[est]

このように括弧の中の文字列を省略していったすべての候補にマッチします。

ちなみにVimのExコマンド[注2]はhelpに:gr[ep]のように表記されていますが、これはgr、gre、grepのどのコマンドでも同じコマンドを表すという意味です。上記3つのコマンドを%[...]を使ってマッチさせようとするなら、ほぼそのまま\vgr%[ep]またはgr\%[ep]でマッチできます。



肯定先読み、肯定後読み\@=, \@<=

肯定先読み、肯定後読み(戻り読み)とは簡単に言うと「○○の前後に現れる△△」を表現できる記法です。ちなみに本記事では解説しませんが否定先読み、否定後読み(戻り読み)というものもあります。

注2) Vimのコマンドラインで実行するコマンド(例：":quit", ":write")

●検索対象：「Visual」と「<半角スペース>Code」に挟まれた「Studio」
パターン(\vあり)
\v(Visual)@<=Studio(Code)@=
パターン(\vなし)
\(Visual\)\@<=Studio\(Code\)\@=

何だかとても複雑で面倒そうですね。先読み、後読み(戻り読み)という名前もどっちが先か後か混乱しやすいところです。しかしVimにはもっと読みやすく素晴らしいメタ文字があります。それが次節で解説する\zsと\zeです。

マッチの開始地点・終了地点\zs、\ze

このメタ文字はVimの独自機能です。前節では肯定先読みと肯定後読みについて解説しましたが、Vimでは\zsと\zeというメタ文字を使うと「○○の前後に現れる△△」にマッチするパターンをもっと気軽に書けます。

●検索対象：「Visual」と「<半角スペース>Code」に挟まれた「Studio」
パターン Visual\zsStudio\ze Code

「\zsのsはstartのs」「\zeのeはendのe」と覚えましょう。置換する場合は(...)によるグループを使うのもいいですが、グループの対象が1つだけなら\zsや\zeが便利です。また置換する場合はグループとキャプチャを使って次のようにも書けます。

Visual Studio Community 2015をVisual Studio Codeに置換
●検索対象：「Visual」と「<半角スペース>Code」に挟まれた「<半角スペース>Studio」
パターン(\vあり)
\v(Visual Studio) Community 2015
パターン(\vなし)
\(Visual Studio\) Community 2015
置換後文字列 \1 Code

しかし置換には使えても、やはり「Visual Studio」だけ検索・ハイライトさせたい場合は肯定先読みや肯定後読み、\zsや\zeを使う必

要があります。

さて、ここで置換の話題に触れたので、そろそろ検索だけではなく置換も試してみたくなったのではないでしょうか？　次節ではいよいよ置換の解説をします。

置換:s[ubstitute]/{パターン}/{置換後文字列}/[フラグ]

{パターン}に関してはこれまで紹介してきたパターンを指定するだけなのでとくに解説はしません。しかし「Vim正規表現の方言」や「/のエスケープ \/」の節で挙げたマッピングを行っている場合は**:s[ubstitute]**コマンドには適用されないので、手動でエスケープなどを入力する必要があることに注意してください。**{置換後文字列}**は置換した後の文字列ですが、いくつか特殊な文字が使えます。詳しくは**:help s/\&**とその周辺を見てもらうとして、本節でもいくつか解説します。

また、**:s[ubstitute]**コマンドの実行方法はコマンドラインモードで**図3**のように実行します。ちなみに**//**のようにパターンが指定されなかった場合は、直前に**/**コマンドや**:s[ubstitute]**コマンドで検索した際の検索パターンが使われます。この挙動は本記事でも解説する**:vim[grep]**コマンドや**:gr[ep]**コマンドと同じです。

{置換後文字列} マッチしたパターン全体&、\0

現在行をダブルクォートで囲みたい場合は次のように書けます。

```
:s/.*/"&"/　または「:s/.*/"\0"/」
```

▼図3　:s[ubstitute]コマンドの実行方法

```
現在行を置換
:s/{パターン}/{置換後文字列}/[フラグ]
現在のファイルのすべての行を置換
:%s/{パターン}/{置換後文字列}/[フラグ]
（vかVか<C-v>で選択したあと、:を押して）選択された行を置換
:'<,'>s/{パターン}/{置換後文字列}/[フラグ]
```

&に置換したい場合は**\&**のようにエスケープします。

{置換後文字列} n番目にマッチした()内の文字列\1, \2, ..., \9

\1と書くことで1番目の括弧の中身、**\2**と書くことで2番目の括弧の中身、というふうにマッチした文字列を参照できます。たとえばCSVファイルで**foo,bar,baz**というレコード(行)を**foo,hoge,bar,baz**というレコードに変更したいとします。その場合は次のようなコマンドを実行します(CSVファイルに**,**を含んだレコードはないものとします)。

```
:s/\v^([^,]*)(,.*)/\1,hoge\2/
```

{置換後文字列} 前回の置換後文字列~

次のようにすると直前の置換を繰り返すことができます(直前の検索パターンと直前の置換後文字列で置換)。

```
:s//~/
```

上記は例として挙げましたが、同様のことをする**&**コマンドというものがあります(ノーマルモードで**&**を入力)。また**:s**でも同様のことができます。

ですので中級者以上の方も普段はほぼ存在を忘れていて、ふと置換後文字列に**~**を指定したら期待どおりにいかなかったという場合が多いでしょう。そういう場合は**\~**のようにエスケープすると**~**に置換できます。

{置換後文字列} 改行\r

改行に置換したい場合は**\r**を使います。次のコマンドはCSVファイルの1レコード(行)のコンマを改行に変換する例です。

```
:s/,/\r/g
```

{置換後文字列} Vim scriptの式で置換 \=

\=を先頭につけるとVim scriptの式として評価し、その結果の文字列に置換します。具体例として、4タブのインデントを2タブに修正するには図4のように実行します[注3]。

まず、置換後文字列のVim script式に/を含むため、区切り文字として:を使っています。:sコマンドの区切り文字は/以外にも英数字、\、"、|以外の文字であればどの文字でも使えます(:help E146)。次にパターンは\v^ +です。これによって先頭の連続している半角スペースにマッチします。

置換後文字列はVim scriptの式です。\=によって後続する文字列がVim scriptの式であることを示しています。submatch(0)で(\=を使わない場合の)\0と同じ結果が取得でき、その文字列の長さを4で割って2で掛けています。これにより修正後のインデント数を取得できるので、その回数をrepeat(" ", {回数})で半角スペースを{回数}回繰り返して連結した文字列を生成しています。このようにVim scriptが書けるようになると、より高度な置換を行うことができます。

[フラグ] 行中のすべてのマッチを置換/g

行がfoo,bar,fooのとき、:s/foo/hoge/だとhoge,bar,fooになりますが、:s/foo/hoge/gだとhoge,bar,hogeになります。

[フラグ] 置換前に確認する/c

すべてのマッチ対象を置換する前に「...に置換しますか？(y/n/a/q/l/\^E/\^Y)」のように確認します。それぞれの文字の意味については:help :s_cを参照してください。

[フラグ] マッチの個数を表示する/n

バッファ中でパターンにマッチした個数を表示して実際には置換を行いません。:help count-itemsからいくつか紹介します(表2)。

%s/\v\w+/&/gnと実行すると、

```
905 箇所該当しました（計 368 行内）
```

のように表示されます。

ファイルをまたぐ検索 :vim[grep]、:gr[ep]

複数のファイルや特定ディレクトリ以下から正規表現にマッチするファイルを検索したい場合、:vim[grep]コマンドと:gr[ep]コマンドを使います。前者はVimが、後者は実際にgrepprgオプションに指定されたプログラムを起動して検索します。後者のほうが高速ですが、前者はVim組み込みのため、どのOSでも使え、grepprgオプションに指定されたプログラムの引数解釈などの違いを意識せずに済みます。

:vim[grep]コマンドは図5のように使用します。:vimgrep /{パターン}/ {パス}のようにパターンを/でくくることに注意してください。ちなみに//のようにパターンが指定さ

▼表2　マッチする個数を表示する書き方の例

実行するExコマンド	意味
:%s/./&/gn	文字
:%s/\v\w+/&/gn	英単語
:%s/^//n	行
:%s/\v<the>/&/gn	"the"(単語一致)

注3) ちなみにこれと同じようなことをhttps://github.com/tyru/codingstyle.vimの:CSChangeSpaceIndentコマンドで実行しています。

▼図4　4タブのインデントを2タブに修正する例

```
%s:\v^ +:\=repeat(" ", strlen(submatch(0)) / 4 * 2):
```

れなかった場合は:s[ubstitute]コマンドと同じく直前に検索した際の検索パターンが使われます。また、##を使うことで引数リストのファイルに置き換えることができます。この機能は図6の:gr[ep]コマンドでも使うことができます(引数リストについて詳しくは:help arglist)。

次に、:gr[ep]コマンドは次のような順序で実行されます。

❶ %や#などの文字をカレントファイル名や代替ファイル名に置き換える(詳しくは:help cmdline-special)
❷ そのままgrepprgで指定したプログラムに渡す
- その際Linuxではシェルを介するため*は**展開されます**[注4]
- しかし**/*はLinuxでデフォルトでインストールされていることの多いshやbashなどのシェルでは**展開されない**ため使えません
- Windowsではシェルにあたるものはないので*も**/*も**展開されません**

何だか面倒そうですが、❶の**%や#などの特殊文字をエスケープする必要があることだけ気を付ければ**コマンドラインでgrepprgオプションに指定されたプログラムを起動するのと同じ感覚で実行するだけです。

コマンドラインに指定するのとほぼ同じなので、文字によってはエスケープしたりダブルクォートあるいはシングルクォートで囲む必要があります(図7)。

:vim[grep]コマンドや:gr[ep]コマンドで検索を実行したあとに検索結果を見るには:cope[n]コマンドか:cw[indow]コマンドを使います。:cope[n]コマンドが常にウィンドウを開くのに対し、:cw[indow]コマンドは検索結果が1件以上存在する場合のみウィンドウが開きます。

注4) 展開を抑制するためには第二引数をシングルクォートで囲みます。

▼図5 :vim[grep]コマンドの使い方

```
 /path/to/dir 直下のみを検索
vimgrep /^[^#]/ /path/to/dir/*
 /path/to/dir 配下を再帰的に検索(「:vim」は「:vimgrep」の省略形)
vim /^[^#]/ /path/to/dir/**/*
 /etc/cron.*/ 配下からコメントアウト以外の行を再帰的に検索
vim /^[^#]/ /etc/cron.*/**/*
```

▼図6 ##による引数リストの参照

```
args /etc/cron.*/**/*
vim /^[^#]/ ##
 パターンが空の場合は直前の検索パターンが使われる
vim // ##
```

▼図7 :gr[ep]コマンドの使い方
(WindowsではデフォルトでfindstrコマンドがあるためこのコマンドはWindowsでは動きません。詳しくはfindstrコマンドのヘルプを参照してもらうか、コラムで紹介するjvgrepコマンドを使用してください)

```
 /path/to/dir 直下のみを検索(# をエスケープしていることに注意!)
grep '^[^\#]' /path/to/dir/*
 /path/to/dir 配下を再帰的に検索
gr -r '^[^\#]' /path/to/dir/
 /etc/cron.*/ 配下からコメントアウト以外の行を再帰的に検索
gr -r '^[^\#]' /etc/cron.*/
```

Column

ag.vim, jvgrep

ディレクトリをまたぐ横断検索ツールとして、grepコマンドより高速に動作するagコマンドなどがありますが、Vimから使えるようにするにはag.vimというプラグインをインストールする必要があります。

ちなみに筆者のお勧めはVim、日本語、マルチプラットフォームとの親和性が高いjvgrepです。

日本語grepが出来るjvgrepというのを作った。
http://mattn.kaoriya.net/software/lang/go/20110819203649.htm

jvgrepコマンドをインストールしたら**set grepprg=jvgrep**のように指定するだけで動作します。jvgrepコマンドがインストールされているかのチェックと併せて、.vimrcに次のように記載するといいでしょう。

```
if executable('jvgrep')
    set grepprg=jvgrep
endif
```

jvgrepの素晴らしい点は次のとおりです。詳しくは上記URL先を参照してください。

❶ *と**/*を展開してくれる(WindowsとLinuxでの差異を意識しなくていい)
❷ 日本語を含む正規表現も正しく解釈してくれる
❸ Vimでプラグインを入れなくても、**set grepprg=jvgrep**だけで認識する

Column

プログラムで正規表現を使うときはメンテナンス性を重視しよう

最近GitHub製のエディタAtomにこんなバグがありました。

私がどのようにしてAtomの奇妙なバグを修正したか：正規表現が暴走を起こすとき
http://postd.cc/how-i-fixed-atom/

ネストした*が開いているファイルの文字列にマッチしようとして失敗するのを何度も繰り返し、特定の状況で改行しようとすると30分もかかってしまったそうです。この問題はCatastrophic Backtracking(壊滅的なバックトラック)と呼ばれており、ReDoS(正規表現によるDoS攻撃)と呼ばれるDoS脆弱性の原因にもなり得ます。

正規表現をユーザからそのまま受け付けたり、動的に生成するようなプログラムを書くのは止めましょう。また上記記事の正規表現のように量指定子を安易にネストさせてはいけません。だいたいの場合ネストさせることなく書けるはずです。

また最適化された正規表現なら素早くマッチさせることができますが、十分な最適化を行うには正規表現エンジンがどのように正規表現を解釈・実行しているかの理解が必要となります。

速度ももちろん重要ですが、何より正規表現はわからない人にはまったく読めないうえ、少し複雑なパターンを書いただけで簡単に見づらくなってしまいます。検索や置換で使う分には便利ですし、正規表現なしでプログラムの文字列加工処理を行うのも面倒ですが、プログラミング言語から正規表現を使う際は見やすさを意識して、なるべく乱用は避けましょう。

バッファ内を検索してExコマンドを実行:g[lobal]、:v[global]

:g[lobal]または**:v[global]**を使うことで、特定のパターンにマッチする行にのみ指定されたExコマンドを実行させることができます。

```
g/^#/d
```

この例では**^#**というパターンにマッチする行を探し、見つかった行に対して**:d[elete]**コマンドを実行しています。**:d[elete]**コマンドは現在行を削除するコマンドなので、**:g[lobal]**コマンドと組み合わせることで「特定のパターンを含む行を削除」することができます。

逆に「特定のパターンを含まない行を削除(=特定のパターンを含む行を残す)」をしたければ、**:g[lobal]**コマンドの代わりに**:v[global]**コマンドを使ってください。

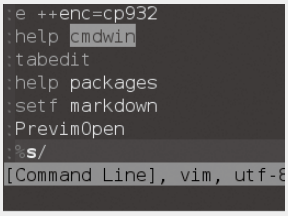
パターンが指定されなかった場合は直前に検索した際の検索パターンが使われます。SD

Column コマンドラインウィンドウ

/コマンドや:s[ubstitute]コマンドで複雑な正規表現を書くとき「hjklで移動できないので挿入モードで編集したい！」と思ったことはないでしょうか？コマンドラインウィンドウという機能を使うと普通のウィンドウでテキストを編集するのと同じようにコマンドラインを編集できます。

```
:%s/(ここで<C-f>を押す)
```

すると図のようにバッファが開きます。



コマンドラインウィンドウを開くキーはデフォルトで**<C-f>**になっていますが、**cedit**オプションで変えられます。

```
:set cedit=<C-l>
```

最初からコマンドラインウィンドウが開いてほしい場合は以下のようにマッピングするという手もあります。

```
nnoremap : q:
```

しかし通常の操作体系と大きく変わってしまうのでお好みで設定してみてください。

Profile

tyru

・都内のシステム会社に勤務しさまざまな現場に赴き働く（おもに）Java と JavaScript なエンジニア
・仕事では Vim を使っていないが「Vim 以外の開発環境について勉強する良い機会」と割り切って（開き直って）仕事している
・休日は vital.vim や自作プラグインのメンテナンス、最近は Vim 本体の開発にも口を出している

Part 4

VimでGitHubをもっと使いやすくする

Author 林田 龍一(はやしだ りゅういち) URL https://rhysd.github.io/

本章ではVim初級者～中級者を対象にGitHubでの開発をより効率的にするためのVimの使い方やプラグインについてご紹介します。プラグインについては簡単に使えて初心者の方でもすぐに効果を実感していただけるようなものをチョイスしました。

GitHubとGit入門

ここではGitやGitHubに触れたことがない方に向けて、それらの概要を紹介し、より深く知るためのリンクをご紹介します。本稿ではこの2つとコマンドの実行方法や簡単な設定の書き方などのVimの基礎は習得済みとして話を進めますので、まだ知らないという方は下記で紹介しているサイトをのぞいてみたり、検索してみてください。

Gitとは

GitはVCS(= Version Control System;バージョン管理システム)のうちの1つです。ソフトウェアをリポジトリという単位で管理し、いつ誰がどんな変更をコードに加えたかを記録し、変更をやり直すなどの操作が行えます。VCSは昨今のソフトウェア開発(とくに複数人による開発)においてはほぼ必須となっています。

Gitはとても複雑なソフトウェアでたくさんのサブコマンドがありますが、「とりあえず使ってみる」ためにはそのうちいくつかを知っているだけで大丈夫です。Gitの基礎を学ぶにはGitHubが公開しているGitチュートリアルサービスTry Git[注1]をやってみると少し感覚がつかめるでしょう。また、Pro Git[注2]という無料の書籍も公開されており、Gitについて深く学ぶことができます。

GitHubとは

GitHubとはソフトウェアプロジェクトをホストすることができるWebサービスです。Gitのリポジトリを GitHub上に置くことができ、ソフトウェア開発の拠点として利用できます。好きなときにGitHub上に置かれたリポジトリを手元に複製(clone)することができます。公開リポジトリは無料で置くことができるため、趣味のソフトウェア開発で利用している方が大勢いる一方、有料で非公開リポジトリをつくりビジネスのためのソフトウェア開発も多く行われています。GitHubではGitHub Training[注3]というGitHubの基本的な使い方がわかる学習サイトを公開しています。

VimとGitHubとGit

VimもGitHubもソフトウェア開発にとても便利ですが、Vimでコードを書きながらGitHubやGitを使っているとVimとブラウザやコマンドラインの往復になってしまい、作業効率が下がってしまうこともあります。本稿ではそういった問題を解決していきます。

注1) https://try.github.io/levels/1/challenges/1

注2) https://progit-ja.github.io/

注3) https://training.github.com/classes/

基礎編：VimからGitを使う

GitHubを使うにはGitを使う必要があります。Gitを多用する場合、ターミナルやGUIなGitアプリとの行き来が発生します。そういった無駄な作業をなるべく減らし、コードを書きながらVimからすぐにGitを使えるようにするためのVimのコマンドや設定をいくつか紹介します。

:! コマンド

Vimにはターミナルのコマンドを Vimの中から直接実行する**:!**というコマンドがあります。**:!**の後にシェルコマンドを指定すると、そのコマンドをVim内から直接実行できます。よって**表1**のようにすると、直接Gitのコマンドを実行することができます。Vimの中で**git**コマンドを使うことで、**git**コマンドにも徐々に慣れていくことができます。もちろんGUIのVimからも使えます。

:!git add %で編集中のファイルをaddできるのは、Vimの**:**で始まるコマンドライン内では、**%**は現在編集中のファイルのパスになるためです。

なお、GUIのVimを使っている場合はコミットで使うエディタをGUIのVimにするために、あらかじめ次のようにGitの設定を済ませておきましょう。

```
$ git config --global core.editor 'vim -g'
```

これでgit commitした際にGUIのVimでコミットメッセージを書くことができます。

▼表1　:!コマンドを使ったGitコマンド実行例

コマンド	説明
:!git status	現在のリポジトリの状態を確認
:!git log	コミット履歴を確認
:!git add %	現在開いているファイルをadd
:!git commit	コミットする

お勧めのVim設定

VimからGitを使う方にお勧めの設定を2つ紹介します。

```
runtime ftplugin/man.vim
nnoremap gc :<C-u>!git<Space>
```

1行目はVimからmanを使うための設定です。たとえば**git add**のマニュアルが読みたくなった場合は**:Man git-add**とすると、色のついた**man**コマンドの出力を見ることができます。

2行目はお勧めのマッピングです。次のようにマッピングを割り当てると、**gc**と入力すると**:!git** まで自動で入力されるため、すぐにGitのサブコマンドを入力できます。

最後にコミット時に便利な設定について紹介します。Gitのコミットメッセージを書くとき、英語で書かれる方も多いのではないでしょうか？　Vimにはスペルチェック機能があり、設定を有効にするだけで利用できます。また、**git commit**時はすぐにコミットメッセージを書き始めたいと思いますので、**:startinsert**でデフォルトで挿入モードにしておくのも便利です(**図1**)。

▼図1　コミットメッセージを書くのに便利な設定

```
英語のスペルチェックを有効にする
set spelllang=en,cjk
Gitのコミット時に自動でスペルチェックを有効にする
autocmd FileType gitcommit setlocal spell
autocmd FileType gitcommit startinsert
```

応用編：プラグインでVim＋Git＋GitHubをもっと便利にする

ここからはGitHubやGitで開発を行ううえで便利なVimプラグインを紹介していきます。プラグインのインストール方法については Part1や**:help add-plugin**を参考にしてください。

VimからGitのコミットを閲覧する

Gitを使って開発を行っていると、どこでどんな変更を入れたのかコミットのログを見たいことが多々あります。**git log --oneline**を使って一覧表示をしても良いですが、コミットメッセージだけではどんな変更が実際に加わったのかを見ることができません。また、かといって**git log -p**のように出力に各コミットでのコードのdiff情報を追加してしまうと、それだけで場所を取ってしまいログを見渡すことができません。

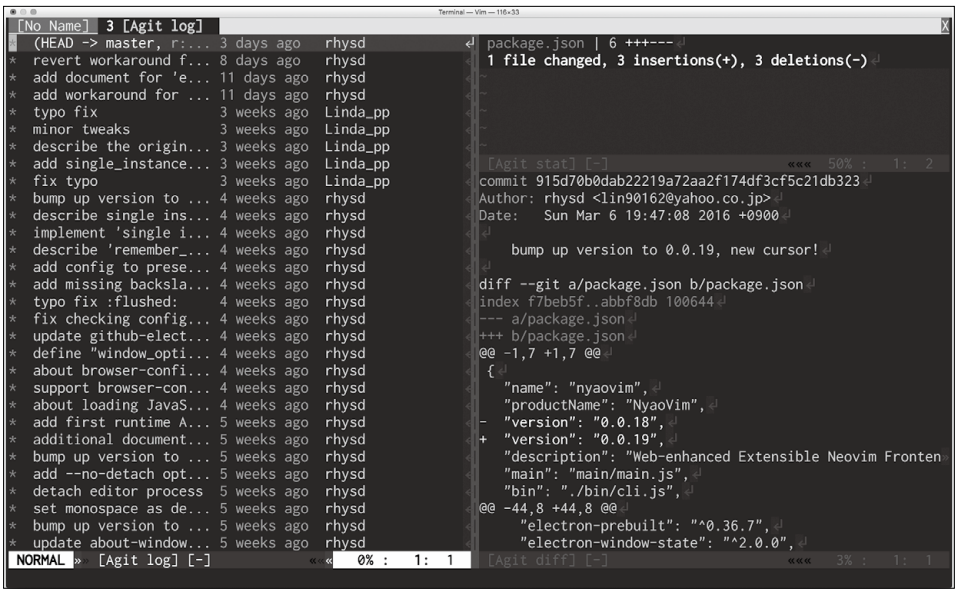
そこで、Gitのコミットを簡単に見ることができるagit.vim[注4]をご紹介します。agit.vimは“コミットブラウザ”に分類されるツールです。コマンドラインで使えるコミットブラウザとしてはtig[注5]が人気ですが、agit.vimはVim内で直接コミット履歴を確認できます。Gitによる変更履歴を追いながら各変更点における変更内容(diff)を見ることができます。

インストールしたらGitリポジトリ内でVimを開いて**:Agit**としてみましょう。**図2**のような画面が表示されるはずです。

画面左半分のウィンドウがGitの変更履歴、画面右上の小さいウィンドウが現在カーソルがある行のコミットのファイル変更情報、画面右下のウィンドウがその変更diffです。コミットは普通のVimの操作のように、**j**と**k**で自由にたどることができ、カーソルが左側のウィンドウ内で移動すると右側のウィンドウは自動で更新されます。**yh**でカーソルがある行のコミットのハッシュ値をコピー(yank)することができます。コピーしたハッシュ値はその

注4) https://github.com/cohama/agit.vim
注5) https://github.com/jonas/tig

▼図2　agit.vimメイン画面



▼表2　agit.vimの主なコマンド

コマンド	説明
:Agit	現在Vimが実行されているリポジトリでコミットブラウザを開く
:Agit --dir=/path/to/repo	リポジトリのパスを指定したいとき
:AgitFile	編集中のファイルの変更履歴を見る
:AgitFile --file=file-path	特定のファイルの変更履歴を見る

コミットを指す一意な値ですので、:!gitなどによってさまざまなコマンドでそのコミットを指す値として使えます。最後にqを押すことでバッファを閉じ、元いた場所にカーソルを戻すことができます。

これでVimからサッとコミット履歴を確認できるようになりました。

また、編集中のファイルや特定のファイルだけに関連する履歴を追いたいときは:AgitFileコマンドを利用できます(表2)。

すでにtigなど別のコミットブラウザを使っている場合は、前章の:!コマンドと組み合わせて:!tigとして使うのもお勧めです。

Vimで編集中のファイルをブラウザで開く

Vimでコードを書いていると、Vimから現在編集しているリポジトリのGitHubページを開きたいことがよくあります。たとえばVimで開いているファイルのカーソルがいる行あたりのコードをほかの人と共有するために、編集中のファイルに対応するGitHubのページのURLがほしいときなどです。

そのような用途で使えるopen-browser-github.vim[注6]というプラグインを紹介します。

まず、このプラグインはopen-browser.vim[注7]の拡張ですので、こちらも一緒にインストールする必要があります。open-browser.vimはカーソル下のURLやWeb検索結果をブラウザで開くといった機能を提供してくれるとても便利なプラグインです。open-browser-github.vimはopen-browser.vimの機能を利用してGitHubのページをブラウザで開きます。

さっそくセットアップしてみましょう。:OpenGithubFileコマンドで現在のファイルのページを開くことができますが、次のようにマッピングしておくと開きたいと思ったときにすぐに開けます。

```
nnoremap go :<C-u>OpenGithubFile<CR>
xnoremap go :OpenGithubFile<CR>
```

この設定によりノーマルモードでgoと入力すると対応するファイルのGitHubページがブラウザで開かれます。これだけでも十分に便利ですが、ビジュアルモードで使うとさらに便利です。

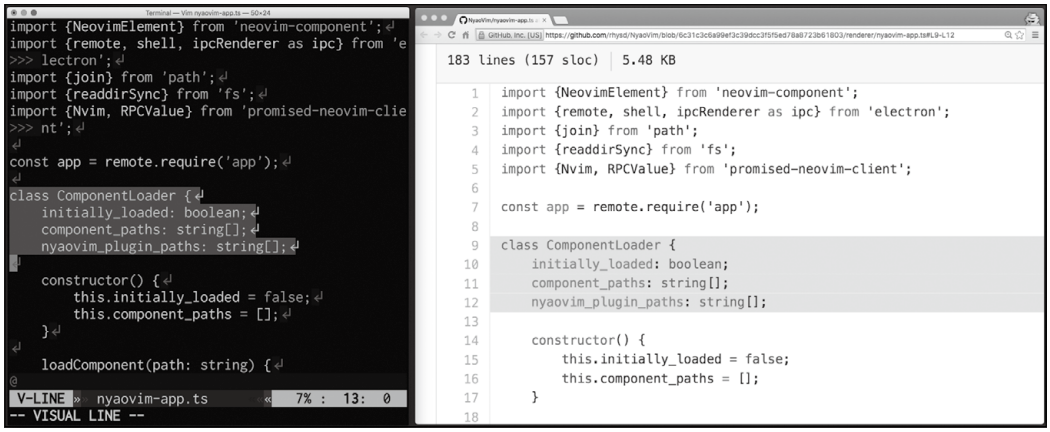
図3のスクリーンショットを見てください。左のVim内でビジュアルモードで範囲選択しgoと入力すると、右のブラウザ内で対応する範囲がハイライトされた状態のページが開かれます。

ビジュアルモードのVimで選択している範囲がブラウザ上でもそのままハイライトされているのがわかります。範囲や行を指定することで自動でその位置までスクロールされるだけでなく、「(特定行を指して)この行でおかしくなってます」や「(関数全体を指して)この関数長過ぎませんか？」など、ファイルのうち特定の場所を示すことができます。

また、現在のGitのコミットに対応したページを開いてくれるのも地味ながら重要な点です。通常GitHubの特定ファイルを表示するページは、https://github.com/user/repo/blob/

注6) https://github.com/tyru/open-browser-github.vim
注7) https://github.com/tyru/open-browser.vim

▼図3　Vimでの範囲選択（左）がGitHubページを開いた際に反映される（右）



`master/...`のようなURLになるのですが、open-browser-github.vimで開くURLは`https://github.com/user/repo/blob/{コミットハッシュ}/...`となります。前者のURLは常にリポジトリの最新を指すため、URLを開いた後に対象のファイルが更新されてしまうと行数などがずれてしまうことがありますが、後者のURLはコミットが固定されているためその心配がありません。

GitHubのissueをVimから確認する

GitHubを使って仕事をしたりオープンソース開発をしている方はGitHubのissue機能を活用していると思います。ワークフローによっては、不具合報告だけでなくタスク管理などをissueで行っている場合もあるようです。issueの本文やコメントにはURLリンクや画像などが含まれるため、基本的にはそれらはブラウザで見るのが一番適当です。ですが、「あの不具合どんな内容だったっけ」や「今実装中の機能の要件は……」といった、ちょっとissueを確認したいときに毎回GitHubの該当リポジトリのissueページを開くのは面倒です。

そこで、そんなときに使えるgithub-issues.vim[注8]を紹介します。

github-issues.vimは名前のとおりGitHubのissuesにVim内でアクセスできるプラグインです。リポジトリ内のissueを一覧表示し、それぞれのissueの詳細（本文・コメント）をVim内で直接確認できます。画像を見ることはできませんが、少し文面を確認する程度であればこれで十分です。

インストールしたら早速使ってみましょう。使い方は至ってシンプルで、`:Gissues`コマンドを入力します。すると**図4**のようにissueのリストが表示されます。続いて、見たいissueの上にカーソルを移動して Enter を押すと、issueの中身が**図5**のように表示されます。なお**図4、5**は筆者の開発したプラグインclever-f.vim[注9]を対象にした例です。

VimでGitHub Flavored Markdownを書く

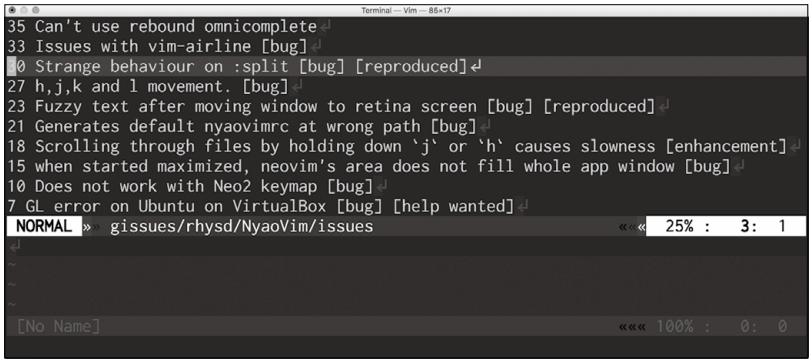
GitHubではドキュメントをいくつかのマークアップ形式（もしくはプレーンテキスト）で記述できます。中でもMarkdown形式は人気があり、GitHub Flavored Markdownという拡張された記法で、多くのREADMEやAPIドキュメントが書かれています。ファイルの拡張子は`.md`や`.markdown`が多いようです。

GitHubユーザはREADMEを始め多くのド

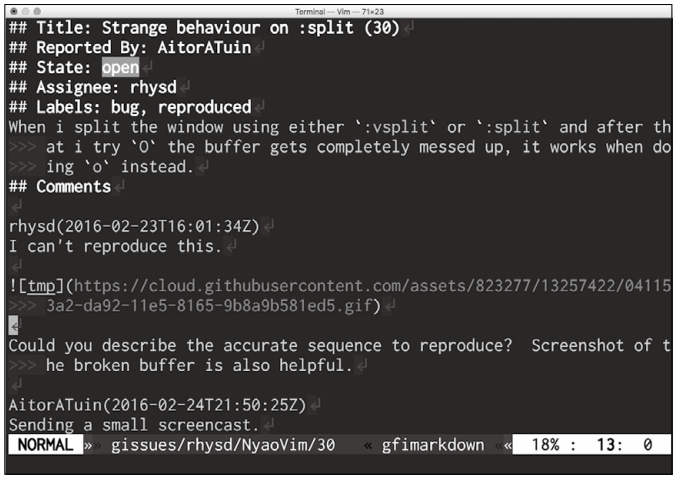
注8）https://github.com/jaxbot/github-issues.vim

注9）https://github.com/rhysd/clever-f.vim

▼図4　リポジトリのissueをリスト表示



▼図5　issueの中身を表示



キュメントをMarkdownで書いており、これらをVimで効率よく編集するのは作業の効率アップにつながるでしょう。

ここではMarkdownテキストを効率よく編集するための2つのプラグインをご紹介します。

Markdownドキュメントのプレビュー

Markdownで書かれたテキストはテキストのままでも読みやすいですが、画像やリンクなども含めて書いたテキストがGitHub上でどのように表示されるかもプレビューしたいことがあります。

そこでご紹介するのがprevim[注10]です。これもopen-browser.vimに依存しています。使い方は非常にシンプルで、

❶ Vimで Markdownファイルを開く

❷ Vimのウィンドウの隣にブラウザのウィンドウを開く

❸ `:PrevimOpen`コマンドを実行

の3ステップだけです。❸の実行後にブラウザの新しいタブが自動で開かれ、その中で現在編集中のファイルのプレビューが表示されます。**図6**がその実行例です。Vim側でテキストを変更すると自動でプレビューも更新されるため、ブラウザを手動で更新する必要はありません。また、ブラウザを開く処理はopen-browser.vimを使って行われるため、余分な設定は不要です。

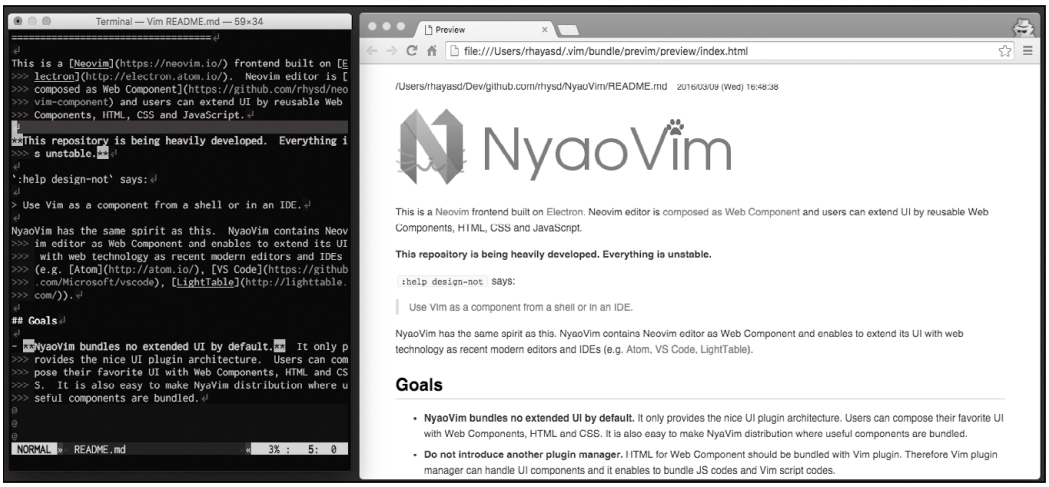
GitHub Flavored Markdownの表記法

本来Markdownには表を書く記法はありませんが、それでは不便なのでGitHub Flavored Markdown拡張として表の記法が定義されています(**図7**)。

HTMLの`<td>`や`<tr>`タグを使って書くよりははるかに良いのですが、新しい行を追加し

注10) https://github.com/kannokanno/previm

▼図6　左にVimでMarkdownテキストが、右にブラウザでそのプレビューが開かれている



▼図7　作表例

```
| Variable  | Type   | Description      |
|-----------|--------|------------------|
| v:version | number | Version number.  |
| v:errmsg  | string | Error message.   |
```

たときに表のレイアウトを修正したりするのが手間です。

そこでご紹介するのがvim-table-mode[注11]です。vim-table-modeを使うと簡単に表を記述・整形できます。さっそく使ってみましょう。vim-table-modeは汎用的な表記法のためのプラグインですので、GitHub Flavored Markdownの表記法に合わせるために次の設定を**.vimrc**に書いておきます。

```
let g:table_mode_corner = "|"
```

Markdownテキストを開いたら、まずは**:TableModeToggle**というコマンドを実行します。デフォルトで**<Leader>tm**というマッピングも定義されているので、そちらを使っても良いでしょう。これだけでは何も変化はありませんが、試しに**|foo|bar|**と入力してみましょう(空白は一切不要です)。入力してみるとこのプラグインの便利さに気付くはずです。

```
| foo | bar |
```

上記のようにマージンとなるスペースが自動的に適切に挿入されました。次に改行して**||**と入力してみましょう。

```
| foo | bar |
|-----|-----|
```

ヘッダ行が自動で生成されたと思います。次に、表の本体部分を書いてみましょう。また改行し、**|this is foo|this is not foo|**と入力してみましょう。

```
| foo         | bar             |
|-------------|-----------------|
| this is foo | this is not foo |
```

入力したセルの幅に応じて、表全体のレイアウトが自動で整形されています。続けて必要な行を記述していけば表の完成というわけ

注11) https://github.com/dhruvasagar/vim-table-mode

です。これで、表のレイアウトを調整しながら表を書いていく必要がなくなりました。

さらに、<Leader>tddマッピングで行を削除したり、<Leader>tdcで列を削除したりすることができ、表の修正も簡単です。もし何かの拍子にレイアウトが壊れてしまったら、:TableModeRealignコマンドを使ってカーソルがある場所の表のレイアウトを整えることができます。まさに至れり尽くせりですね。

さらにさらに、上級者向けに表のセルを選択できるテキストオブジェクトや、セルの中身を式で指定できる:TableAddFormulaといった機能もあります。デフォルトでマッピングを定義されてしまうのが気に入らない場合は:help table-modeでドキュメントを見ながら設定しましょう。また、vim-table-modeのREADMEに載っている紹介用のYouTube動画[注12]はとても参考になるので、目を通しておくと良いでしょう。

GitHubの絵文字、issue番号、リポジトリ名などを補完する

READMEやコミットメッセージをVimで書いていると、GitHub上のリポジトリへのリンクやユーザ名、絵文字記法、issueやPull Request(PR)の番号を記述することが多くなります。これらは忘れやすく、毎回ブラウザで確認しているとせっかくVimで効率よく編集していたはずが、いつの間にかブラウザとVimの往復で非効率になってしまいます。

そこで、github-complete.vim[注13]という補完プラグインをご紹介します。Vimにはオムニ補完という、コンテキストを考慮した賢い補完を提供するための機能があります。github-complete.vimはこの機能を利用し、GitHubのさまざまな要素を補完する機能を提供します。

それではインストールしたら試しに使ってみましょう。Gitリポジトリの中のMarkdownファイルを新たに生成するか開いてください。

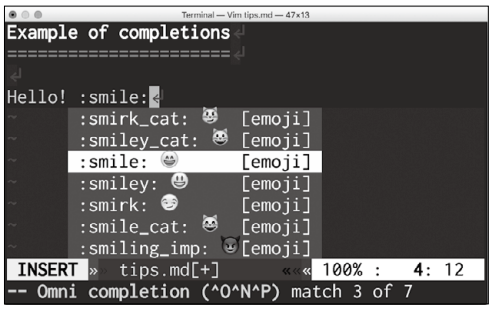
GitHubでは:dog:のように特定のワードをコロンで挟むと絵文字になる機能があります。絵文字は種類が多く、なかなか覚えることができません。そこで:と入力した後、挿入モードのまま<C-x><C-o>(Ctrl + x Ctrl + o)と入力してみてください。<C-x><C-o>はオムニ補完を発動する一般的なキーマッピングです。すると図8のように、絵文字が補完候補に表示されます。

候補として絵文字が表示できるかどうかは環境によって決まります。絵文字が表示できない環境の場合は、絵文字の説明がテキストで表示されます。そのような環境では次のように設定すると、絵文字の説明を日本語に変更できます。

```
let g:github_complete_emoji_japanese_workaround = 1
```

次にissue/PR番号を補完してみましょう。試しにissueやPRが存在するリポジトリのMarkdownファイルを開いてください。開いたら、#と入力し、その直後にまた<C-x><C-o>としてみてください。図9のように、issue/PR番号とそのタイトルが補完候補に表示されました。候補を選択すると該当のissue/PR番号が入力されます。GitHubでは#の後に数字を入力すると、その番号のissue/PRへ自動的にリンクされます。

▼図8　GitHubの絵文字を補完



注12) https://www.youtube.com/watch?v=9lVQ0VJY3ps
注13) https://github.com/rhysd/github-complete.vim

また、これとほぼ同様にユーザ名も補完できるようになっています。ユーザ名は**@名前**という形式で、たとえば**@rh**と入力した後に**<C-x><C-o>**を入力すると**rh**で始まるユーザ名が補完候補として列挙されます。

最後にリポジトリへのリンクの補完機能について紹介します。GitHub上のドキュメントでは、依存関係にあるパッケージへのリンクなど、**[名前](https://github.com/user/repo)**といったGitHub上のリポジトリへのリンクを記述したいことがよくあります。ですが、毎回GitHub上のページをブラウザで探して入力していたのでは面倒です。そこでgithub-complete.vimではそれを補完できるようになっています。

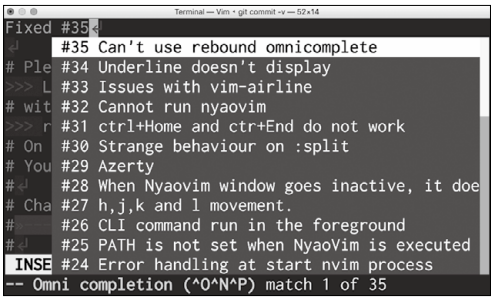
たとえば、**[clever-f](https://github.com/clever-f.vim)**というプラグインへのリンクを書きたいとします。その場合、**[clever-f](**まで入力して**<C-x><C-o>**を入力すると、github-complete.vimはラベル部分の**clever-f**でGitHubを検索し、その検索結果のURLを補完候補にして出してくれます。**図10**がその様子です。後は候補からリンクを選ぶだけです。

これらの補完機能はデフォルトでGitのコミット時のバッファとMarkdownドキュメント編集時に有効になっています。また、上記のgithub-issues.vimと一緒に使う場合は機能が一部コンフリクトしてしまっていますので、次のように設定してください。

```
let g:github_issues_no_omni = 1
```

github-complete.vimはAPI tokenを使ったプライベートリポジトリへの対応やneocompleteサポート、オムニ補完ではなく独自の補完マッピングを提供するといった機能も持っています。カスタマイズしたいときはドキュメントやREADMEを読んでみてください。

▼図9　issue/PR番号を補完



▼図10　リンク記法中のリポジトリURLを補完

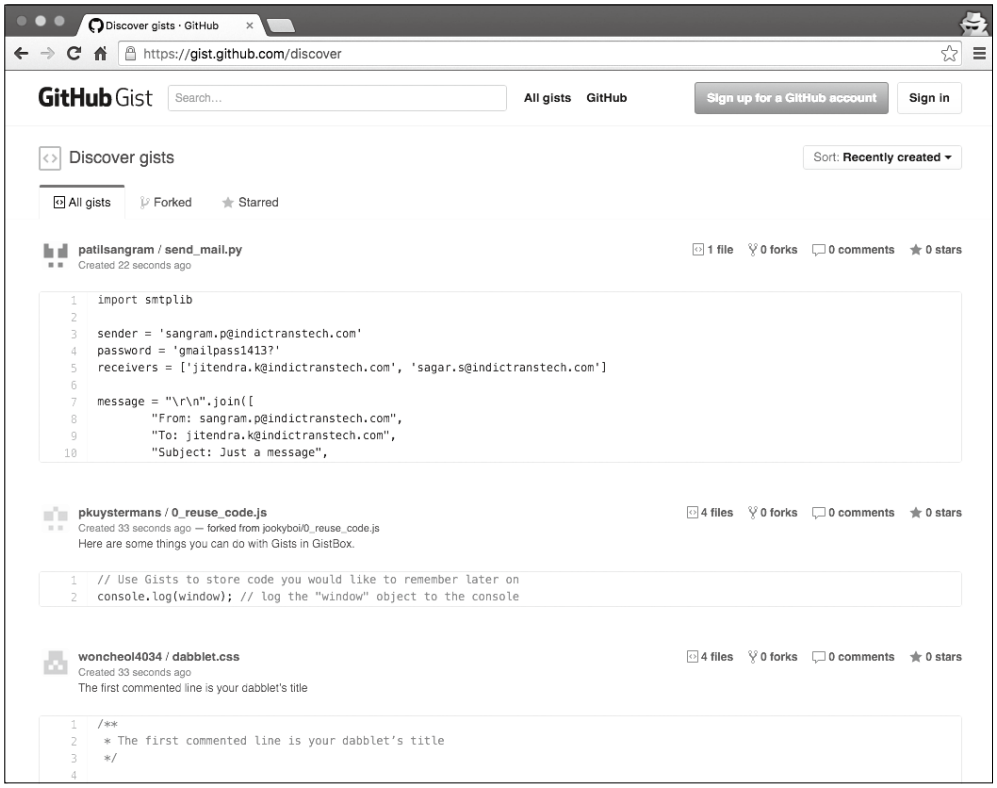
Gistを Vimから使う

Gist[注14]はGitHubの姉妹サービスで、GitHubのリポジトリ管理よりも細かいgistという単位でコードを管理できます(**図11**)。たとえば、

・ちょっとスクリプト書いてみた
・こんなエラーログが出たんだけど……
・メモ置き場

注14) https://gist.github.com/

▼図11　Gistメインページ



といった用途で、ちょっとしたコードやテキストを保存しておけます。URLを教えることでほかの人と共有したり、コードスニペットをメモしたりできます。もちろんコードはすべてGitでバージョン管理されており、ほかの人からも検索できるpublic gistと、検索できないprivate gistがあります。

ちょっとしたコードやメモを共有したいときに、Vimで編集したテキストをわざわざブラウザを開くことなく直接Gistにアップロードしてurlをクリップボードにコピーしたり、すでにGist上にあるファイルをダウンロードしてくることなく手元のVimで編集したりしたいですよね？

それを可能にするVimプラグインgist-vim[注15]をご紹介します。依存しているwebapi-vim[注16]と一緒にインストールしたら早速遊んでみましょう。

Gistにアップロードしたとき、URLがクリップボードにコピーされているとすぐにURLが共有できて便利なので、**図12**のように**.vimrc**に設定しておきます。

また、GitHubのユーザ名をGitの設定として登録しておきましょう。

```
$ git config --global github.user <username>
```

準備が済んだら何かファイルを用意して、それをVimで開いたら、**:Gist**コマンドでアッ

注15) https://github.com/mattn/gist-vim
注16) https://github.com/mattn/webapi-vim

▼図12　URLをクリップボードに自動コピーする設定

```
Linuxの場合
let g:gist_clip_command = 'xclip -selection clipboard'
OS Xの場合
let g:gist_clip_command = 'pbcopy'
アップロード時にブラウザで開きたい場合
let g:gist_open_browser_after_post = 1
:w! でgistを自動更新
let g:gist_update_on_write = 2
```

プロードしてみましょう(**表3**)。

編集中のファイルがGistへアップロードされます。gistを誰かと共有したい場合はクリップボードに入っているURLをどこかに貼り付けるだけです。また、この後も引き続き編集し、**:Gist -e**でgistを更新することもできます。

アップロードの仕方はわかったので、次はVimからGistを閲覧する方法について見ていきましょう。

表4のいずれかのコマンドを実行すると、対応するgistがリストで一覧表示されます。さらに各gistの閲覧・編集を行いたければ、カーソルを開きたいgistの上に移動して Enter キーを押すと、自動でそのファイルを手元のVimで開いてくれます。これで「Gistにメモ上げといたから見といて」などと言われてもわざわざブラウザを開く必要すらありません。

▼表3　gist-vimのアップロードコマンド

コマンド	説明
:Gist	publicなgistとしてアップロードする
:Gist -p	privateなgistとしてアップロードする
:Gist -a	匿名ユーザとしてアップロードする
:Gist -m	Vimで開いているすべてのバッファのファイルを一度にアップロードする

▼表4　gist-vimの閲覧コマンド

コマンド	説明
:Gist -l	自分のアップロードしたgist一覧
:Gist -l rhysd	@rhysdさんのアップロードしたgist一覧
:Gist -ls	starを付けたgist一覧

そのほかにもいっぱい

今回は誌面の都合上紹介できませんでしたが、まだまだGitやGitHubを便利にするプラグインはたくさんあります。

- **git**コマンドのwapperである「vim-fugitive[注17]」
- **git commit**のときに開くウィンドウをリッチにする「committia.vim[注18]」
- Gitの操作を独自のUIで行う「vimagit[注19]」
- 高速なgistのリスト表示や編集ができる「vim-gista[注20]」
- GitHubのアクティビティが見られる「vim-github-dashboard[注21]」
- リッチなUIでGit操作が行える「vim-gita[注22]」

ぜひ、自分なりのGitの使いこなしやお気に入りのプラグインなどを見つけて、GitHubでの開発を楽しんでください！SD

注17) https://github.com/tpope/vim-fugitive
注18) https://github.com/rhysd/committia.vim
注19) https://github.com/jreybert/vimagit
注20) https://github.com/lambdalisue/vim-gista
注21) https://github.com/junegunn/vim-github-dashboard
注22) https://github.com/lambdalisue/vim-gita

Profile

林田 龍一（はやしだ りゅういち）

・都内でテスト自動化のQAエンジニアとして働いています。犬とプログラミングツールが好きです
・趣味でLLVMを使って自作言語のコンパイラをつくったり、Vimプラグインを書いたり
・最近はElectron + TypeScriptでデスクトップアプリもつくってます。もちろんVimで

最後の章は、キーボードから少し手を離して、Vimの歴史と今後についてみていきましょう。『Vimの細道』の筆者が、Vimの誕生から流行、ユーザコミュニティの役割や性格、そしてVim開発のウラガワについて語ります。章の最後では急成長するNeovimと、変わりはじめたVimについても紹介していきます。

みなさんはもちろんご存じだと思いますが、Vimはとても古い歴史のあるテキストエディタです。Bram Moolenaar氏がAmigaにviを移植し始めたのが1988年ですから、もう25年も昔に誕生した骨董品です。ソースツリーには、いまだにAmiga向けのコードが含まれています。「UNIXエンジニアが使うテキストエディタの鉄板として、よく今まで残ってきたな」そう思うことすらあります。

今回はそんなVimの歴史を、筆者の昔話も交えながら話していきたいと思います。

Vimの黎明期

Vimとの出会い

筆者が初めてVimを知ったのはバージョン2.0のとき、viと比べてまだそれほど機能が足されていないころでした。UNIXで開発をしていた筆者はHP-UX上でVimをビルドし、その目新しさにワクワクしました。しかし、viと大して変わらなかったことに加え、マルチバイト文字列を正しく扱えなかったので、常用は諦めざるを得ませんでした。

そのころから筆者はvi[注1]を使っており、viのキビキビとした動作の虜になっていました。

そんなとき、Vimのバージョン2.0を日本語化した有志の方が現れ、JVimが誕生しました。日本語入力機能としては、Cannaを扱えるように作られたonewというライブラリを取り込み、JVim+onewと呼ばれていました。もちろん筆者もJVimのビルドにチャレンジし、なかなかうまくいかなくて苦労した記憶があります。しかし、それでも筆者はviとの機能差に魅力を感じなかったため、JVimに移行することはありませんでした。

それから1年か2年が過ぎたころ、VimがWindows対応してとても驚いたのを覚えています。Windowsでそれほどプログラミングをしたことがなかった筆者にとってVimのソースコードはとても新鮮で、その目まぐるしい開発スピードに毎日ワクワクしながら、時間が空いたらVimのソースコードをビルドする、といったことをやっていました。

JVimに抱いていた疑問

そのころのテキストエディタ界はとても賑わっており、Emacsやその派生版のXEmacs、MeadowやngといったEmacsクローンの誕生を始め、viにおいてもElvisのWindows対応など、とにかく日々新しい何かが起こっていた時代でもありました。その後、JVimが2.0から3.0に移行し、筆者もある時期はJVimを使っていました。しかし2000年ごろから筆者の中に疑問が生まれ始めます。

注1）実際には、elvisのマルチバイト対応版jelvisだったと思います。

「どうしてJVimはVimの後追いなのだろう。なぜ、JVimの良い機能はVimにフィードバックされないんだろう」

そんなフラストレーションを抱きながら、あるときVimの開発者メーリングリストvim-devで1人の日本人らしき名前を見つけます。「香り屋」というWebサイトでVimのWindowsバイナリを配布しているKoRoNさんです。そのころKoRoNさんは、Vimの正規表現のマルチバイト文字対応を行っており、とても活発にパッチを送っておられました。筆者がvim-devにパッチを送り出したのもそのころだったと思います。

「そうだ、JVimが変わらないのなら、自分で変えていけばいい」

そう気づいたのもこのころです。

その後、KoRoNさんが提供していた日本人向けVimのメーリングリストvim-jpに参加し、中平さんはじめいろいろな方を知りました。そのメーリングリストはその後閉じられてしまいましたが、vim-users.jpというサイトでkanaさん、thincaさん、Shougoさん、ujihisaさんほか、多くの日本人によって情報共有が行われていました。

そのころの筆者とKoRoNさんはというと、Vimのマルチバイトに関するバグを見つけてはパッチを送り、「どっちが多くパッチを送るか」といった背比べをやっていました(意識していたのは筆者だけだと思いますが)。

vim-jp誕生

そして2011年、筆者とKoRoNさんの思いつきで始まったvim-jp.orgという日本人向け情報共有サイトがオープンし、VimのTips共有やパッチの作成、カンファレンスの開催など、コミュニティとしての活動が始まりました。今もなお、vim-jpからは多くのパッチがvim-devに提供され、現在ではvim-devに流れるパッチのおよそ半分が、日本人が作ったパッチとなっています。k-takataさんやh-eastさんほか、いろいろな方がとても活発的に活動しています。

vim-jpは開発者集団

vim-jpにはいろいろな人が集まってきます。Vimのプラグイン作者やVimにコードで貢献したい人、やたらと端末に詳しい人、MacVimに詳しい人、Windowsに詳しい人、翻訳が上手な人……。皆がそれぞれの得意分野から知恵を持ち寄って、足りない機能に対する議論やバグの修正方法、vim-jpの方向性について検討しています。いろいろな人の力でいろいろなことができています。そのいくつかを紹介します。

翻訳マニュアル

`:help`で表示されるVimのマニュアルも、vim-jpの有志の力で翻訳されています[注2]。Vimプラグインを管理するプラグインマネージャをお使いの方であれば、これをそのままチェックアウトしていただければ、`:help`で日本語訳を表示できます。次はvim-plugでの設定方法です。

```
Plug 'vim-jp/vimdoc-ja'
```

また、この翻訳リポジトリで変更された内容は自動でHTMLに変換され、vim-jpのサイト[注3]に反映されます。最近では表記の揺れ対策にも力を入れており、品質がとても良くなりました。実はVimに付属しているチュートリアルプログラムvimtutorの日本語訳を行ったのは筆者なのですが、今読み返すとけっこ

注2) https://github.com/vim-jp/vimdoc-ja
注3) http://vim-jp.org/vimdoc-ja

う恥ずかしくなってしまうような訳が多く、よくこれを皆の目にとどまるところに公開したなと、昔の自分のアクティブさに嫉妬してしまいます。

vim-cpp

Vimに同梱されているsyntaxファイルには、新しいものもあれば古いものも存在します。中にはオリジナルのメンテナに連絡が取れなくなってしまったものもあります。その中の1つがC++用のsyntaxファイルでした。C++の予約語が追加された場合や、修正が必要となった場合にオリジナルのメンテナにメールを送っていましたが、いっこうに返事がいただけなかったので、vim-jpがメンテナの代役を名乗り出ました。現在C++用のsyntaxファイルは、vim-jp管理のリポジトリ[注4]でメンテナンスされています。C++用syntaxに問題があった場合は、GitHubからissueを登録してください。

reading-vimrc

Part1でも触れられていますが、Lingrというオンラインチャットサービスに「vim」という部屋があります。こちらにはVimに精通した人たちが頻繁に集まります。有名なVimプラグイン開発者もたくさんやってきます。このチャットルームでは毎週土曜日の夜23時から、「vimr読書会」というオンラインの勉強会イベントを行っています[注5]。

GitHub上にあるいろいろな人たちのvimrcをお題に、設定内容の間違いを見つけたり、良い設定を参考にしたり、プラグインについて情報交換したり、わからない設定内容について質問し合ったりします。このイベント、実は開始されたのが2012年の7月ですが、今まで一度も欠かすことなく続けられています。

vimconf

Vimに関する国際カンファレンスで、2013年から開催されています。昨年は15名のVim有識者による発表が行われ、約60名の一般参加がありました。サイト[注6]もプロのフロントエンジニアの方が有志で作成し、とてもカッコいいデザインのサイトができあがりました。もちろん、その方もVim使いです。

vital.vim

vital.vim[注7]はVim plugin開発者向けに提供されるライブラリ群です。とても巨大なライブラリ群なのですが、モジュール形式になっており、必要なモジュールのみをインポートして使用します。また、vital.vimのバージョン差異によって問題が発生しないように、特定のVimプラグインに必要なモジュールのみ同梱させられるようになっています。

issues

そしてvim-jpと言えば、issues[注8]を欠かすことはできません。2011年から2016年現在まで、数えきれないほどのパッチがvim-jpからvim-devに送り込まれました。

issuesは単なるBTS(バグトラッキングシステム)ではありません。Vimでおかしな挙動に遭遇したとき、バグかどうか判別できない人へのアドバイスも行います。そして質疑の結果、それがVimのバグであることがわかると有識者がパッチを作成し、vim-devへ送付する、という流れを作っています。このパッチを書く人たちはよく「パッチ職人」と呼ばれています。2011年当初から比べると、ずいぶんパッチを書いていただける方が増えたように思います。とてもうれしいことです。

注4) https://github.com/vim-jp/vim-cpp
注5) http://vim-jp.org/reading-vimrc
注6) http://vimconf.vim-jp.org
注7) https://github.com/vim-jp/vital.vim
注8) https://github.com/vim-jp/issues

このように、日本のVimコミュニティは時代とともに小さくなっていくのではなく、実は2000年あたりからどんどんと大きくなってきているのです。骨董品のようなテキストエディタで、なぜそのようなコミュニティが継続しているのかと思うかもしれませんが、その原動力はVimの豊富な機能と抜群の編集能力にあると筆者は思っています。

Vimの開発事情

2015年、Googleはコードリポジトリのホスティングサービス Google Codeをシャットダウンすることを発表しました。Vimのソースコードは Google Codeでホスティングされていたため、シャットダウンまでの数ヵ月間、リポジトリの移転先について検討しました。有名なリポジトリホスティングサービスの社員から「ウチのサービスを使うといいよ」という持ち掛けがきたこともありました。議論の結果、Bram Moolenaar氏はGitHubへの移行を決めました。それまでGoogle Code上では、Mercurialを使ってソースコードが管理されていましたが、GitHubに移ったことでGitによる管理となりました。Google Codeだったころと比べると、開発者からのコントリビュートが増え、vim-devにとっての良い潤滑油になったようです。

Vimの開発スタイルは一風変わっています。GitHubに移行する前はメーリングリストでパッチがやりとりされ、コミットされていました。Vimのリポジトリへのコミットは Bram Moolenaar氏だけが行います。その際、変更内容を1ファイルに書いたパッチファイルが、新しいバージョン番号のファイル名でリリースされます。ftp.vim.orgには、メジャーリリースされた際のtarボールと、それ以後にリリースされたパッチファイルがホスティングされており、Vimをビルドしたい人はその両方を持ってきては、自分でパッチを適用するという運用が行われていました。ずいぶんと昔の開発手法のように見えますが、リポジトリがGitHubに移行した今でも、このスタイルは変わりません。ftpにはパッチファイルが置かれ、メーリングリストに新しいバージョンのアナウンスが行われています。

このパッチファイルには冒頭に**リスト1**のような文章が書かれています。それぞれ、このパッチをリリースするにあたっての問題点と解決策、そして修正ファイル一覧となります。このSolution部にはパッチを提供した人の名前が記載されます。GitHubなどでプルリクエストがマージされ、コミット一覧に名前が載ることも開発者にとってはとてもうれしいことなのですが、この名前が記載されたパッチファイルがメーリングリストに流れる瞬間が、我々パッチ職人にとって誇らしい瞬間であったりもします。

▼リスト1　パッチファイルの冒頭部分

```
Patch 7.4.1468
Problem:    Sort test doesn't test with ↵
"1" argument.
Solution:   Also test ignore-case ↵
sorting. (Yasuhiro Matsumoto)
Files:      src/testdir/test_sort.vim
```

Neovimの登場

2003年、筆者は「Vimからソケット通信をするためのパッチ」を書いてvim-devに送りました。残念ながらそのパッチが取り込まれることはなかったのですが、そのときにBram Moolenaar氏にこう言われました。

I think this is not something that is directly related to text editing.（テキストを編集することには直接関係するものではない）

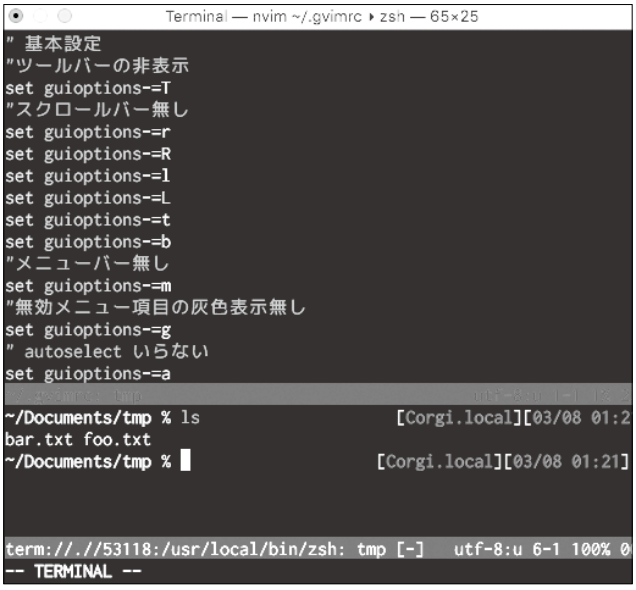
この時点で筆者は、Vimには今後もこういったパッチが取り込まれることはない、そう確

信しました。しかし、ユーザがVimに期待している内容も時代の流れによって変わってきました。そんな中2013年から2014年に掛けて、vim-devにタイマーを実行するためのパッチと、非同期でタスクを実行するためのパッチが流れました。Bram Moolenaar氏はこのパッチの議論に参加することはありましたが、結局このパッチも取り込まれることはありませんでした。それから数ヵ月後、Vimをforkした新しいVim「Neovim」が現れます。

これまで多くのパッチでツギハギを当て続けてきた結果、Vimのソースコードはお世辞にも良いとは言えない状態になっていました。そのソースコードをリファクタリングし、Vimに取り込まれることがなかった次の機能をNeovimが開発し始めました。

- Vim scriptではなく、Luaによるエディタの拡張
- 分離されたGUI
- プロセス間通信を使用したプラグイン拡張
- msgpack-rpcを使ったプラグイン間連携
- viminfoの再実装(shada)

▼図1　Neovimのterminal機能



Neovimは、Thiago Arruda氏がユーザに資金援助を募ってフルタイムで開発を行っています。その援助だけでなく、GitHub上であらゆる開発者からのコントリビュートを受けることで俊敏な開発スピードを獲得し、その目新しさもあってか、一部のユーザはVimからNeovimへ移行していきました。しかし、ボランティアウェアであるVimに対して、資金を受けて開発を行うNeovimのスタイルが気に入らないとするユーザも一部にはいます。

Neovimが流行った要因の1つとして、terminal機能があります。terminal機能は、Neovimの分割ウィンドウに仮想端末を表示するというものです(**図1**)。たとえば、nginxの設定ファイルを修正しながら端末ではnginxを再起動するといったことがシームレスに行えるようになります。なお、Vimにおいてもこの端末機能はTODOリストに入っているため、いずれ実装されるかもしれません。

Neovimが流行っているもう1つの要因は、GUIの分離にあります。GUIを分離してAPIを提供することで、サードパーティによるユーザインターフェースの実装が行えるようになります。

たとえば、Neovim自身はPythonで実装されたGTKのユーザインターフェースしか提供していませんが、コミュニティにより**表1**のユーザインターフェースが扱えるようになっています。とくに、本特集Part4を執筆しているrhysdさんが開発しているNyaoVimについては、Electron UI上でNeovimが動作しており、HTML/CSSによる拡張性に富んだテキストエディタになっています。レンダラがcanvasであり、HTML/CSSの上で動作しているので、ポップアップメニューやMarkdownプレビューといったダイナミックなユーザイ

▼表1　Neovimで使えるUI（コミュニティ提供）

Platform	Project（GitHubのページ）
Atom Integration	carlosdcastillo/vim-mode
Electron UI	coolwanglu/neovim-e
Electron UI	rhysd/NyaoVim
GTK/Python UI	neovim/python-gui
Mac OS X	qvacua/nvox
Mac OS X	rogual/neovim-dot-app
Mac OS X	stefan991/NeoVimX
Qt 5	equalsraf/neovim-qt
Rust IDE	oakes/SolidOak
gnome-terminal	fmoralesc/neovim-gnome-terminal-wrapper
Konsole-wrapper	harish2704/neovim-konsole

ンターフェースを簡単に実装できています（**図2**）。

Vimが変わりはじめる

Neovimは確かに先進的で、ワクワクするプロダクトであるのは間違いない事実です。「このままVimは、Neovimとユーザを二分していくのかな」と筆者も思っていました。しかしある日突然、Bram Moolenaar氏がVimにソケット通信を実装するためのパッチをリリースしました。2003年に筆者がソケット通信パッチを送ってから、実に14年の歳月を経てVimが変わり始めたのです。

もちろん、Vimはテキストエディタです。テキストを編集している最中にソケット通信でブロッキングするわけには行きません。そこでBram Moolenaar氏は、channelという通信路のしくみを取り入れ、またjob-controlというプロセス間通信のしくみを実装しました。

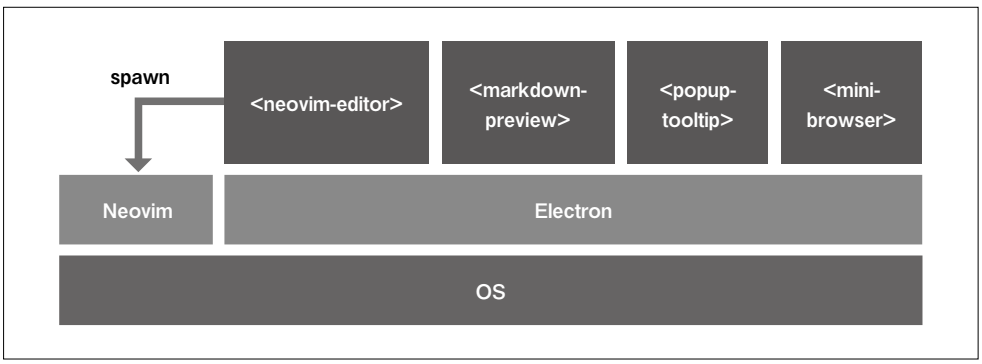
channel

リスト2のVim scriptを実行すると、Vimはlocalhostの5000番ポートで稼働するサーバに接続してソケット通信を行います。その後サーバに対して「hello world」という文字列を送信します。サーバから非同期にデータを受信したときは、Callbackという関数が呼ばれます。modeを指定することで、JSONの送受信もできます。

job-control

リスト3のVim scriptを実行すると、Vimは「python app.py」を実行し、コマンドの標準入力に「hello world」を書き込みます。起動したプロセスの標準出力からデータが得られると、そのメッセージをVimで表示したあとにプロセスを停止します。これらのスクリプトはUNIXでもWindowsでもほぼ同じように動作します。また、標準入力をバッファやファイルから入力させたり、標準出力を特定のファイルやバッファに指定したりすることもできます。

▼図2　NyaoVimのアーキテクチャ



▼リスト2　channelを使う

```
function! Callack(ch, msg)
  echo a:msg
endfunction

let ch = ch_open('localhost:5000', {'callback', 'Callback'})
call sendraw(ch, 'hello world')
```

リスト4を実行すると「pipe-output」というバッファに非同期でテキストが追加されます。ちょうどtailコマンドのような動作になります。たとえば、筆者が作ったコンソール向けTwitterクライアントtwty[注9]のストリーミング機能を使うと、リスト5を実行するだけでTwitterのタイムラインが非同期にVimのバッファに流れ続けます。その間も、Vimでは異なるファイルを編集できます。

timer

リスト6のVim scriptを実行すると、Vimはテキスト編集が可能な状態に戻りますがバックグラウンドでタイマーが実行され、1秒置きに現在時刻が表示されます。オプションで**repeat**を「-1」に指定すると繰り返しのタイマーが作成できます。Vimプラグイン開発者にとっては、これが一番欲しかった機能とも言えます。

json

プロセス間通信のフォーマットとして、Neovimではmsgpackを使用していますが、Vimでは可読性のあるJSONが採用されています。これまでVim scriptで扱える型は、

- 数値
- 文字列
- 関数リファレンス
- 配列
- 辞書
- 浮動小数点

のみでしたが、新たに、

注9）https://github.com/mattn/twty

▼リスト3　job-controlを使う

```
function! Callack(ch, msg)
  echo a:msg
  call job_stop(job, "hup")
endfunction

let job = job_start('python app.py')
let ch = job_getchannel(job)
call sendraw(ch, 'hello world')
```

▼リスト4　「pipe-output」というバッファに非同期でテキストを追加する

```
let job = job_start('python app.py', {'out_io': 'buffer', 'out_name': 'pipe-output'})
```

▼リスト5　Twitterのタイムラインを非同期にVimのバッファに流す

```
let job = job_start('twty -S', {'out_io': 'buffer', 'out_name': 'Twitter'})
```

▼リスト6　timerを使う

```
function! Callback(timer)
  redraw | echo strftime('%c')
endfunction

let t = timer_start(1000, function('Callback'), {'repeat': -1})
```

▼リスト7　設定ファイルからJSONオブジェクトを生成

```
let obj = json_decode(join(readfile("config.json"), "¥n"))
```

・真偽値(v:true/v:false)
・無効値(v:null, v:none)
・Job オブジェクト
・Channel オブジェクト

が追加され、データ型についてはJSONとVim scriptを相互に変換できるようになりました。これまでVim scriptでJSONを扱う場合は、筆者が開発していたwebapi-vim[注10]や、そのvital.vim版であるWeb.JSONが使われてきましたが、Vimがネイティブにサポートすることになり、たとえば設定ファイルからJSONオブジェクトを生成するのであれば、**リスト7**のような短いコードでできるようになりました。

gtk3対応

Linux版gvimのユーザインターフェースは、これまでgtk2という若干古いユーザインターフェースライブラリが使われてきましたが、gtk3を使ってビルドできるようになりました。

CIの導入

リポジトリがGitHubに移ってからvim-jpのメンバが積極的に活動し、継続的インテグレーションCI(Continuous Integration)のしくみとして、Travis CIとAppVeyorを使った自動テストが実行されるようになりました。これにより、既存の機能がコードの変更に伴って壊れていないか調べるチェックが、自動的かつ継続的に行われるようになり、Bram Moolenaar氏も積極的にソースコードを修正できるようになりました。

MS-DOS、Win16のコードを削除

これまでVimのリポジトリにはMS-DOSや16bit Windowsに対応するための古いソースコードが含まれていましたが、もはや使用している人はいないだろうという判断がなされ、コードベースから削除されました。これらは、筆者がVimを初めて見たときには現役のソースコードだったこともあり、感慨深く感じました。

Vimはこれからも変わり続ける

リポジトリがGitHubに移行して以来、Vimはかなりの速度で成長を続けています。2016年3月時点ではパッチ数が1,500を超えてしまいました。取り込まれるとは思っていなかった機能がどんどん追加されていっています。それに合わせて、Vimプラグイン開発者もjob-controlやchannelを使ったダイナミックなVimプラグインの開発が行えるようになっていくでしょう。筆者も、今からワクワクしています。SD

注10) https://github.com/mattn/webapi-vim

Profile

mattn

・Slerという仕事の側ら、Vimやgolangなど、あらゆるOSSにパッチを投げ続けるエンジニア
・本誌では毎月「Vimの細道」を連載中
・vim-jpの発起人。emmet-vim、gist-vimなどの作者でもあり、vital.vimのメンテナも務める

ノーマルモード

以下のコマンドはノーマルモードから入力もしくはタイプします。**<ESC>** は Esc 、**CTRL-** で始まるものは Ctrl を押しながら次のキーをタイプします。
: / ? で始まるコマンドは最後にリターンキーをタイプします(**:e!** や **:q!** など末尾に **!** が付くコマンドは強制実行)。

ファイル操作(コマンド)	
ファイルを開き直す	**:e, :e!**
ファイルを開く	**:e** ファイル名
ウィンドウを消す	**:q, :q!**
上書き保存	**:w, :w!**
上書き保存してウィンドウを消す	**:wq**(もしくは **:x, ZZ**), **:wq!**
すべて上書き保存して終了	**:wqall, :wqall!**
保存せずにウィンドウを消す	**ZQ**
モード切り替え	
インサートモード	**i**
行頭でインサートモード	**I**
カーソル直後でインサートモード	**a**
行末でインサートモード	**A**
行を追加してインサートモード	**o**
上に行を追加してインサートモード	**O**
ノーマルモード	**<ESC>**
ウィンドウ操作	
新しいウィンドウを開く	**:new, :new** ファイル名
新しいタブを開く	**:tabnew, :tabnew** ファイル名
ウィンドウを分割してファイルを開く	**:split** ファイル名
ウィンドウを横分割してファイルを開く	**:vsplit** ファイル名
ウィンドウを隠す	**:hide**
ウィンドウを閉じる	**:close**
現在のウィンドウのみを表示	**:only**
1つ前のウィンドウに移動	**CTRL-w w**
ウィンドウを閉じる	**CTRL-w c**
ウィンドウを消す	**CTRL-w q**

移動	
上/下/左/右	**k/j/h/l**
ページアップ/ダウン	**CTRL-u/CTRL-d**
行頭/行末	**0/$**(**^**は最初の文字へ)
次の単語/前の単語	**w/b**(**W/B**で空白区切りでジャンプ)
文字xまでジャンプ	**fx**
逆方向へ文字xまでジャンプ	**Fx**
[や {、[の対へ移動	**%**
先頭行へ移動	**gg**
最終行へ移動	**G**
検索	
textを検索	**/text**
textを逆方向に検索	**?text**
次を検索	**n**
逆方向に次を検索	**N**
カーソル下の単語を検索	*****
置換	
行内で置換(fooをbarに置換)	**:s/foo/bar/**
全置換(fooをbarに置換)	**:s/foo/bar/g**
編集	
カーソル下の文字をxで置換	**rx**
行を連結	**J**
カーソル下の文字を削除	**x**
大文字/小文字に変更	**~, gU/gu**のあとにモーション(★)を指定
行をヤンク(コピー)	**yy, Y**
ヤンク(コピー)	**y**のあとにモーション(★)を指定
貼り付け	**p**
前方向に貼り付け	**P**
削除	**d**のあとにモーション(★)を指定
行を削除	**dd**
カーソル位置から行末までを削除	**D**
指定の部分を変更	**c**のあとにモーション(★)を指定
インデント	**==**, 選択して**=**
やりなおし	**u**
繰り返し	**.**

★モーション	
上/下/左/右	**k/j/h/l**
次の単語/前の単語	**w/b**(**W/B**で空白区切りでジャンプ)
文字xが見つかるまで	**tx**
行頭/行末	**0/$**(**^**は最初の文字へ)
文字xまで	**fx**
逆方向で文字xまで	**Fx**
ヴィジュアル選択	
領域選択	**v**
矩形選択	**CTRL-v**
行選択	**V**
すべての行を選択	**ggVG**

<ESC> :

コマンドモード

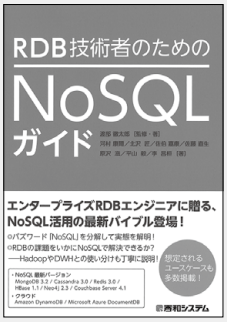
挿入操作	
レジスタaの中身を貼り付け	**CTRL-r a**(**CTRL-r +**でクリップボードからペースト)
編集中カーソル下の単語	**CTRL-r CTRL-w**(**CTRL-r CTRL-a** で空白区切りの単語)
編集中カーソル下のファイル名	**CTRL-r CTRL-f**
特別文字	
現在のファイル名	**%**(例:**:w % .bak**で.bakを付けてファイルをバックアップ)
編集中の別のファイル	**#**
N番目の別のファイル	**#N**
すべての引数(オプション以外、空白区切り)	**##**

<ESC> i

インサートモード

補完	
前の候補(もしくはバッファ内補完)	**CTRL-p**
次の候補(もしくはバッファ内補完)	**CTRL-n**
ファイル内の行	**CTRL-x CTRL-l**
ファイル内のキーワード	**CTRL-x CTRL-n**
ファイル内のincludeファイル	**CTRL-x CTRL-i**
タグ	**CTRL-x CTRL-]**
ファイル名	**CTRL-x CTRL-f**
ユーザ定義補完	**CTRL-x CTRL-u**
omni補完	**CTRL-x CTRL-o**
スペル修正	**CTRL-x CTRL-s**

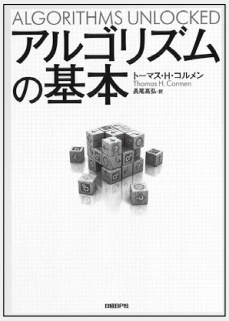
Vim CHEAT SHEET
By mattn



RDB技術者のための NoSQLガイド

河村 康爾、北沢 匠、佐伯 嘉康、佐藤 直生、原沢 滋、平山 毅、李 昌桓 著／渡部 徹太郎 監修
A5判／568ページ
3,400円＋税
秀和システム
ISBN＝978-4-7980-4573-3

NoSQLと聞いても、ピンとこない人は多いのではないだろうか。本書は、そもそも「NoSQLとは何なのか」から、現在どのようなプロダクトがあるのかまで解説しており、NoSQL分野の全体像を概観できる。本書によれば、NoSQLという言葉はRDB以外のすべてのデータベースを指すものではなく、「KVS」「ドキュメントDB」「グラフDB」へと細分される、ターンアラウンドタイムを重視したDBの総体を指すものである、とのこと。この分類に基づいて、RDBとの比較、NoSQL同士の比較が細かく行われている。プロダクトとしては、Redis、Cassandra、HBase、Amazon DynamoDB、MongoDB、Couchbase、Microsoft Azure DocumentDB、Neo4jの概要と用例が書かれており、製品選定の際に役立つだろう。

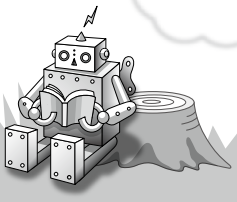


アルゴリズムの基本

トーマス・H・コルメン 著／長尾 高弘 訳
A5判／288ページ
2,400円＋税
日経BP
ISBN＝978-4-8222-8543-2

アルゴリズムの本というと、具体的なプログラミング言語や擬似コードを使った実装例を見ながら処理の流れを追う、“カロリー”の高いものが多いが、本書はその点、少ない数式と日本語でアルゴリズムの手続きの流れを表しており、プログラミングに疎い人でも親しみやすい。それぞれの章では、中華料理の作り方やホッケーの防具を着る順番といったエピソードを使ってアルゴリズムを解き明かしていく。扱う領域としては、ソートと探索、最短経路、文字列操作、暗号の基礎、そしてデータの圧縮である。最後の章では「ハードな問題」として、巡回セールスマン問題なども扱っている。解説するアルゴリズムがどんな分野でどのような役割を担っているのかという導入と、参考文献が示されており、学習の次の一歩につながる1冊と言える。

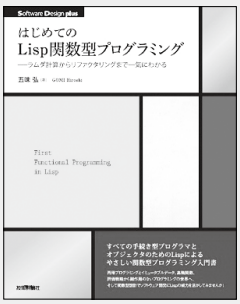
SD BOOK REVIEW



Swiftポケットリファレンス

WINGSプロジェクト　片渕 彼富 著／山田 祥寛 監修
四六判／512ページ
2,780円＋税
技術評論社
ISBN＝978-4-7741-7984-1

オープンソース化が発表されてから人気を伸ばし続けるSwift。興味を持っている読者も多いだろう。Swiftはnull参照によるトラブルを防ぐOptional型や、実行結果を即時確認できるplaygroundなど、生産性向上に寄与する多くの特徴を備えている。文法もモダンで、経験が浅くても理解しやすいのも利点だ。Linuxでも活用できる可能性が生まれた今、これからプログラミングを始める人も、すでに習得している言語がある人も、一度は触っておくべき将来有望な言語と言える。本書は逆引き形式で、実装したい機能を短いコードで紹介し、気軽に試せるようになっている。解説はコンパクトだが躓きやすいポイントは押さえてあるので、iOS開発の経験がなくても困ることはないはずだ。ぜひ本を片手に洗練されたSwiftを体感してほしい。



はじめてのLisp関数型プログラミング

五味 弘 著
B5変形判／272ページ
2,580円＋税
技術評論社
ISBN＝978-4-7741-8035-9

関数型プログラミングのメリットとして、コードの再利用性・並列処理・バグの少なさが挙げられる。しかし、手続き型やオブジェクト指向に慣れてきた人にとっては、これらメリットを担保する「副作用のない」という特性が大きな障壁となり、書きながらその感覚を身に付けていくほかない。本書は、Lispで関数型プログラミングを学ぶというコンセプトの1冊。なぜLispかというと、「言語仕様が小さい」「インタプリタ型」「おまじない不要」「動的型付け」というのが著者の挙げるメリット。言語自体の障壁が低いので、関数型の学習に専念できるのだ。本書の中核の演習の章では、ハノイの塔やエイトクイーンなどをLispで書きながら、再帰プログラム、高階関数、イミュータブルデータ、遅延評価といった関数型のエッセンスを学べる。

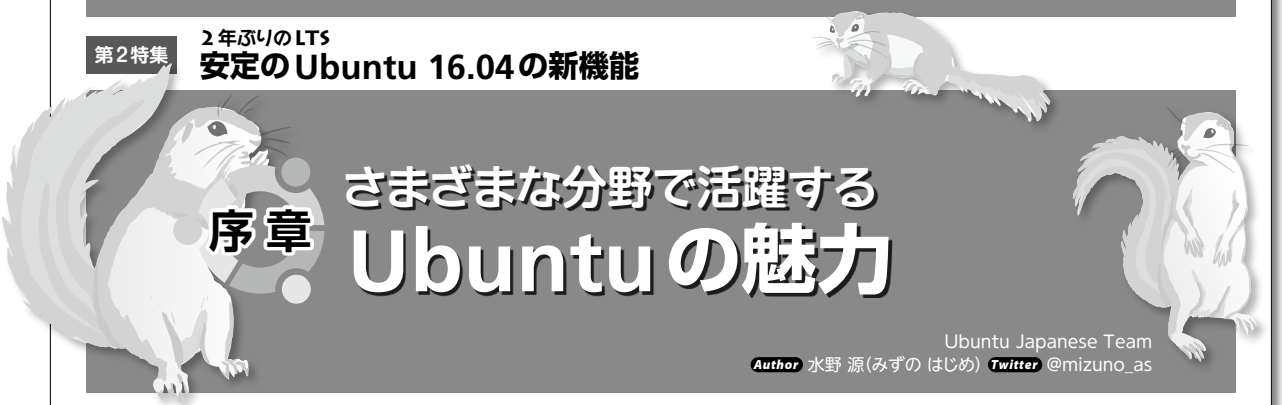
第2特集

2年ぶりのLTS

安定のUbuntu 16.04の新機能

本特集では、Linuxディストリビューションの中でもデスクトップOSとして一番利用されているUbuntuの最新バージョン16.04の新機能、派生する各フレーバー、最新情報を紹介します。今回はLTS(Long Term Support：長期サポート版)で5年間サポートされます。前のLTSである14.04から2年が経過し、半年に1度のバージョンアップを追いかけていない方も、そろそろ本バージョンに乗り換えることを考えてみてはいかがでしょうか。

序章では、Ubuntuの概要や現在の立ち位置について解説します。そして第1章では、16.04での新機能や変更点について、第2章では、Ubuntuから派生する各フレーバー(Kubuntu、Xubuntu、Lubuntu、Ubuntu GNOME、Ubuntu MATE、Ubuntu Studio)と日本語入力メソッドについての解説、第3章では、サーバ用途として使われるUbuntu Serverについて独自機能(Juju、MAAS、LXD)を交えて紹介します。



序章 さまざまな分野で活躍する Ubuntuの魅力

Ubuntu Japanese Team
Author 水野 源(みずの はじめ) Twitter @mizuno_as

2016年4月21日、Ubuntuプロジェクトは最新の長期サポート版となるUbuntu 16.04 LTSをリリースします。本特集ではUbuntu 16.04 LTSの新機能や特徴を解説していきますが、その前にそもそもUbuntuとはどんなOSで、どんな魅力があるのかを、あらためておさらいしておきましょう。

使いやすいデスクトップを目指したLinux

UbuntuはDebian GNU/Linuxをベースに開発されているLinuxディストリビューションです。Ubuntuは「世界中のあらゆる人が利用できる、高品質なデスクトップ環境を提供すること」を目標に掲げ、2004年10月に最初のバージョンである4.10がリリースされました[注1]。

現在、UbuntuはWindows、OS Xに続く第3のデスクトップ用OSとして名前が挙げられるほどに成長しました。これはUbuntuがデスクトップOSとして機能的に優れているだけでなく、見た目の華やかさや、初心者でも簡単に設定が行えるインターフェースといった部分を重視している点が、コンピュータの専門家ではない人々にも受け入れられたためです。

現に海外では、Ubuntuプリインストールの PCがさまざまなメーカーから販売されています。また公的機関に採用されるケースも増えています。Intel Compute Stickのような小型PCにもLinux搭載モデルがラインナップされていますが、そのような用途でも、多くのケースでUbuntuが利用されています。

Ubuntu 16.04 LTSのデスクトップと各種フレーバーの特徴、そしてUbuntu Japanese Teamがリリースしている「日本語Remix」については、第2章であわしろいくや氏が解説します。

サーバ、クラウド分野での活用

使いやすいデスクトップLinuxとして確固たる地位を築いたUbuntuですが、そのイメージが先行するためか、サーバには不向きであると誤解されることがあります[注2]。しかしUbuntuはサーバ用のOSとしても広く使われています。たとえば、Dockerの Ubuntuイメージは3,500万回以上起動されており、2015年にはAWSをはじめとする各種クラウドサービス上で、少なくとも2,000万以上のUbuntuインスタンスが起動されたとするレポートもあります[注3]。日本国内でも、多くのクラウドサービスやVPSで、Ubuntuサーバのマシンイメージが標準で用意されています。

Ubuntuがサーバ分野でも人気なのは、「Canonical社による強力な開発支援」「有償サポート」「定期的なリリース」「明確なサポート期間[注4]」といった、サーバ運用に適した環境が整っていることが大きな理由です。またMaaS[注5]やJuJu[注6]といった、サービスの構築運用をサポートする各種ソフトウェアの開発整

注1) Ubuntuのバージョンはリリースされた西暦の下2桁と月2桁をドットで区切って表します。今回のリリースは2016年4月にリリースされたから16.04というわけです。

注2) ごく初期のリリース(4.10～5.10)においてはサーバ版が存在しなかったことも、この誤解のもとになっているかもしれません。

注3) http://blog.dustinkirkland.com/2015/12/more-people-use-ubuntu-than-anyone.html Ubuntuは Windows や RHELと異なり、正確なユーザ数を把握することは困難なため、数字の信憑性については微妙な部分もあります。しかしそれでもサーバ、クラウド分野でUbuntuが広範囲に渡り利用されていることは間違いないでしょう。

注4) Ubuntuのサポート期間は通常9ヵ月ですが、2年に一度リリースされる「LTS」は5年のサポートが約束されています。タイムベースリリースと明確なサポート期間によって、導入やリプレースの計画が立てやすいのがメリットです。

注5) http://maas.io/

注6) http://www.ubuntu.com/cloud/juju

備を進めているところも、Ubuntuならではの魅力と言えるでしょう。

サーバOSとしてのUbuntu 16.04 LTSの特徴については第3章で、Ubuntuの国内ミラーの管理者でもある吉田史氏が解説します。

IoT、スマートデバイス分野への進出

パーソナルコンピューティングのメインストリームは、もはやPCからスマートフォンやタブレットへ移ったと言ってよいでしょう。Ubuntuもこの流れに乗り、デスクトップPCやサーバだけでなく、スマートデバイスや流行りのIoTの分野にも進出しています。Ubuntuでは「Ubuntu Touch」と呼ばれるモバイルデバイス向けのOSを開発しており、すでに海外ではこのOSを搭載したスマートフォンやタブレットが登場しています。

もちろん、単にモバイル向けのOSをリリースすることがUbuntuのゴールではありません。Ubuntuは、モバイルデバイスとデスクトップにおいて同じアプリケーションが、統一されたインターフェースと操作性の下で、それぞれの環境に合わせた形で利用できる環境を実現することを目指しています。この概念は「Convergence」と呼ばれています。たとえばUbuntu PhoneにBluetoothマウスを接続すると、それまで全画面で動作していたアプリケーションがマルチウィンドウ表示に変化します。このような「デスクトップとモバイルの融合」がすでに実現されています。

IoT分野では「Snappy Ubuntu Core」の利用も進んでいます。また定番のロボット開発プラットフォームである「ROS」はUbuntuをベースにしていることもあり、ロボット開発の分野ではUbuntuがデファクトスタンダードとなっています[注7]。ホワイトボックススイッチといったネットワーク機器上でもUbuntuは動作していますし、さらにジョークのようですが、Ubuntuを搭載した冷蔵庫[注8]といった事例も発表されています。個人レベルでは、教育用の低価格ARMボードの代名詞であるRaspberry Pi上でも、Ubuntuを動かすことができます[注9]。

このように、Ubuntuは単なるPC、サーバ向けOSの枠にとどまらず、より多くの分野で利用されはじめています。Ubuntu 16.04 LTSとスマートデバイスについては、本誌連載中のUbuntu Monthly Reportにおいて、柴田充也氏が解説しています。こちらも本特集と併せてご覧ください。

オープンなコミュニティと企業サポートの両立

Ubuntuの開発はコミュニティによって行われています。Ubuntuコミュニティは開発プロセスや情報をオープンにしており、誰もがUbuntuの開発、アプリケーションの翻訳、新機能の提案、不具合の報告、不具合の修正、ほかのユーザのサポート、世界各地でのプロモーションといった活動に参加できます。

そしてそのUbuntuコミュニティをサポートしているのが、英国に本社を置くCanonical社です。Ubuntuプロジェクトの創始者であるマーク・シャトルワース氏によって設立されたCanonical社は、Ubuntuを資金、技術の両面で強力に支援し、Ubuntuの品質向上に大きな役割を果たしています。

オープンなコミュニティによる開発と、企業によるサポートが非常に高いレベルで両立しているのも、Ubuntuの魅力の1つです。

Ubuntuを使ってみよう

このようにUbuntuは、初心者にもやさしいデスクトップを提供し、エンタープライズ向けのサーバとしての使用にも耐え、教育用や組み込み用途でも広く使われはじめている、そんなOSです。

すでにUbuntuを利用している人はもちろん、これからUbuntuを始める人も、本特集を読んでUbuntu 16.04 LTSの世界を体験してみてください！ SD

注7） Ubuntuが搭載されたドローンも登場しています。
注8） 実用性はとりあえず置いておくとして、Ubuntuのこれからの可能性を示す発表の1つではあるでしょう。
注9） ただし搭載しているSoCの都合上、Raspberry Pi 2以降のモデルに限られます。

第1章 GNOMEソフトウェア採用、Python 3への移行など多岐にわたる Ubuntu 16.04 LTSの新機能の概要

Ubuntu Japanese Team／株式会社 創夢
Author 柴田 充也(しばた みつや)

2年ぶりの長期サポート版であるUbuntu 16.04 LTSは、前回の14.04と比べると下まわりが大きく変わっています。そこでこの章では総称的な「Ubuntu」においての概要を紹介しつつ、14.04からの変更点も併せて解説します。

Ubuntu 16.04 LTSの新機能

ここ数年、Ubuntuの開発リソースはスマートフォン／タブレット向けのTouchや、IoT向けのSnappy、クラウドまわりのツールに集中していました。そのためここ数回のリリースでは、とくにデスクトップユーザがはっきりと実感できるような大きな変更点はそこまでなかったのが実状です。

Ubuntu 16.04 LTSの開発期間も、TouchやSnappyに比べると変更点は少なめです。しかしながら2年ぶりのLTSであること、そして何より今後5年間サポートしなくてはいけないことを見据えて、いろいろな機能の整理が行われています。

そこで、まずはUbuntu 16.04 LTSで登場する予定の新機能や変更点について紹介します。

GNOMEソフトウェア

デスクトップユーザにとって最も大きな変更点は「Ubuntuソフトウェアセンター」から「GNOMEソフトウェア」への移行でしょう。「Ubuntuソフトウェアセンター」は9.10から導入されたアプリケーションストアです。単純なアプリケーションの検索とインストールだけでなく、アプリケーションの評価やコメント、スクリーンショットの表示、商用アプリや書籍の購入といった、スマートフォンであれば当たり前の機能を、デスクトップにも提供するために開発されたのです。これに対してGNOMEはUbuntu以外のディストリビューションでも使われます。そこでより汎用的なストアアプリとして、2013年ごろからGNOMEに導入されたのが「GNOMEソフトウェア」です(**図1**)。

GNOMEソフトウェアは、パッケージ管理機能としてAPTだけでなくDNF/yumやZYppなどにも対応しています。またアプリケーションのメタ情報は、AppStreamに従って作られたデータを参照しています。このAppStreamはLinuxディストリビューションやデスクトップ環境に依存しない形で、アプリケーションのメタ情報を作成できるようにFreedesktop.orgが策定した規約です。これによりGNOMEソフトウェアは、ディストリビューションをまたいで同じUIや同じ機能を提供できるようになりました。

今のところUbuntuで採用するGNOMEソフトウェアはまだカテゴリ機能がなかったり、デスクトップアプリケーション以外の情報が用意されなかったりと、ソフトウェアセンターに比べると機能的に足

▼図1 新しいストアアプリであるGNOMEソフトウェア

りない部分がまだまだたくさん存在します。しかしながらLTSが5年間サポートされることを考慮して、あえてソフトウェアセンターではなくGNOMEソフトウェアを採用することになりました。

なおUbuntuではPackageKitではなく直接APTを使用し、ソフトウェアセンター時代の評価・コメントを引き継いで表示するようにGNOMEソフトウェアを改変しています。UnityのDash画面から直接インストールする方法も引き続き提供されています。

ちなみにUbuntuはデスクトップ向けとは別に、スマートデバイス向けのアプリケーションストアとして「Ubuntu Store」を、またIoT向けとして「Web DM」を開発しています。将来的にはデスクトップのストアもUbuntu Storeに統合される見込みです。

Python 3への移行

Ubuntu 16.04 LTSの大きな目標の1つは「デスクトップにおけるPython 3への移行」でした。16.04は2021年の4月までサポート予定ですが、Python 2系のサポート期間は2020年までの予定です。このため、すでにサーバやTouchは初期状態だとPython 3のみがインストールされるようになっていました。しかしながらデスクトップイメージのみ、まだPython 2にも依存していたのです。そこで標準アプリケーションの構成を見直したうえで不要なパッケージは削除し、必要なパッケージのうちまだPython 2に依存しているものは修正する作業を、ここ数回のリリースで少しずつ進めていました。

3月上旬の時点で、残る依存関係はNautilus関連のみとなっています。正確にはNautilusで各種デバイスのマウントを担当しているgvfs-backendがSambaに依存しています。このSambaパッケージにPython 2への依存が残っていることが原因です。プリンタ設定のように、必要に応じてユーザがSambaをインストールするためのUIを追加できればよいのですが、まだ結論が出ていない状態です。

標準でPython 3のみインストールされるだけで、Python 2パッケージが削除されるわけではありません。16.04でもパッケージをインストールさえすれば、引き続きPython 2を利用できます。ちなみに/usr/bin/pythonは常にPython 2を向いています。Python 3のみの環境になったからといって、/usr/bin/pythonがPython 3を示すわけではありません。

標準アプリケーションの選別や更新

Python以外では、CD/DVD書き込みツール「Brasero」、メッセージングツール「Empathy」が、どちらもあまりメンテナンスされていないことを理由に、標準ではインストールされなくなりました。またスケジューラーとして「GNOMEカレンダー」が追加されています。

AMDGPUドライバに対応したX.orgモジュールであるxf86-video-amdgpuも最初からインストールされます。このモジュールとMesa 11.1の組み合わせにより現行世代のミドルレンジ以上のAMD向けグラフィックボードやノートPC向けAPUであれば、インストール後に何もしなくてもUnityを快適に利用できます。AMDGPUドライバについては、より新しいカーネルからもいくつかのコードを取り込んでいます。

AMDGPUの導入とAMD側の都合により、AMD向けのプロプライエタリなビデオドライバを提供するfglrxパッケージは削除されました。AMDGPUで動かないAMD製のGPUを使っている場合は、当分は14.04を使い続けた方がよいでしょう。来年頭ぐらいにリリース予定の16.04.2まで待てば、解消される可能性があります。

ちなみにCirrusやTrident、Savageといった古いグラフィックチップ向けのX.orgモジュールは、標準ではインストールされなくなりました。必要であればDashから「追加のドライバ」を検索・起動し、リストアップされるドライバをインストールしてください。

Unityの変更点

Ubuntu標準のデスクトップUIであるUnityは、引

き続き7.x系を採用しています。このためUbuntu 15.10はもとより、14.04と比べてもそこまで大きな変化はありません。ただし細かい機能追加や修正は行われています。

・Dash画面からの電源操作

Dash画面で「shutdown」や「reboot」「logout」と入力することで、シャットダウン／再起動／ログアウトなどのダイアログを表示できるようになりました(**図2**)。これによりマウスやファンクションキーに手を伸ばすことなく、電源管理系の操作を行えます。

・Launcherアイコンの中クリック機能

Launcherに表示されているデバイスファイルやゴミ箱アイコンを、マウスの中ボタンでクリックすると、新規ウィンドウでファイルブラウザが起動するようになりました。

・ウィンドウ切替中にウィンドウを閉じる

Alt + Tabキーでウィンドウを切り替えている間に、特定のウィンドウに対してAlt + Qを入力するとそのウィンドウが閉じるようになりました。切り替え中に不要なウィンドウがあることに気がついたときに便利です。

・スクロールバーの見た目の変更

Dashの右端に表示されるスクロールバーが、Unity独自の実装からGtk+ベースの実装に変更となりました。これによりほかのウィンドウと操作性が同じになります。

▼図2 Dashからシャットダウン可能に

・プライバシー機能の有効化

システム管理の「セキュリティとプライバシー」で設定できる「オンラインの検索結果を含める」がインストール直後の状態ではオフ(含めない)になりました。これはDashで検索したときに、Canonicalのサーバに検索文字列を送りAmazonなどのオンライン検索結果をDash上に表示するかどうかの設定です。実装当時から、オンライン検索を標準で有効化することはプライバシー的に問題があるのではないかという指摘があり、数年越しにようやく安全側に寄せるようにしたということになります。この変更に合わせて、メンテナンスがされていないオンライン検索系Scopeも削除されています。

Ubuntu 14.04 LTSからの変更点

今はUbuntu 14.04 LTSを使い続けており、16.04のリリースを機にLTS間のアップグレードを考えているユーザも多いことでしょう。そんなユーザのための注意点や、14.04からの変更点もまとめておきます。

まず初めに16.04へのアップグレードのタイミングですが、16.04の評価をするなど特別な場合を除いて16.04.1がリリースされるまで待つようにしてください。これまでの慣例から考えると、16.04のリリース直後には利用者が一気に増えることでさらに多くの不具合が洗い出されるため、1ヵ月程度は頻繁な修正アップデートが続くからです。実際、新規リリースを通知するソフトウェアもデスクトップやサーバの両方で、16.04.1がリリースされて初めてLTS間のアップグレードが可能になった旨を通知する予定です。よって通知が行われるまでは引き続き14.04を使い続けた方が無難でしょう。

もちろん不具合を洗い出すために16.04を使う場合はこの限りではありませんし、利用者が増えないと十分に不具合を修正できません。安定性を求める場合以外は、積極的に新しいリリースを利用してください。

14.04からの主だったパッケージのバージョンの違いを**表1**にまとめました。この表は2016年3月現

在のバージョンを元にしています。リリース時はさらに更新されている可能性があります。ちなみにFirefoxやThunderbird、OpenJDKなどは、アップストリームのリリースに合わせて最新のバージョンに更新されるため、表には掲載していません。

なおAPTの更新により古いアルゴリズムで署名されたリポジトリは警告が出ます。3月時点でVirtualBoxなどのリポジトリに警告が出ていますので注意してください。

Upstartからsystemdへ

16.04にアップグレードする際に最も注意すべきは、initデーモンがUpstartからsystemdに変わったことです。これはDebianに追随した変更で、すでに15.04では標準のinitデーモンがsystemdに置き換わっていました。14.04から16.04にアップグレードした場合、独自に作成したUpstart用のサービスは起動しません。必要に応じてsystemdのサービスファイルを作成してください[注1]。

ちなみにデスクトップの場合は、ユーザセッションで使われるアプリケーションの起動にUpstartを使っています。このためUpstartそのものがなくなったわけではありません。

注1）http://gihyo.jp/admin/serial/01/ubuntu-recipe/0358

▼表1　14.04 LTSと16.04 LTSの主だったパッケージのバージョンの違い

ソフトウェア	14.04 LTS	16.04 LTS
基本パッケージ		
Kernel	3.13	4.4
Kernel（HWE）	4.2	4.4
Upstart	1.12	1.13
systemd	204	229
Coreutils	8.21	8.25
vim	7.4.052	7.4.963
GNU Emacs	24.3	24.5
Git	1.9.1	2.7.4
APT	1.0.1	1.2.9
byobu	5.77	5.104
デスクトップアプリケーション		
Unity	7.2.6	7.4.0
X.org Server	1.15.1	1.18.1
Mesa	10.1.3	11.1.2
LibreOffice	4.2.8	5.1.1
LightDM	1.10.6	1.18.0
Network Manager	0.9.8.8	1.0.4
IBus	1.5.5	1.5.11
Fcitx	4.2.8.3	4.2.9.1
Mozc	1.13.1651.102	2.17.2116.102
Nautilus	3.10.1	3.14.3
Gedit	3.10.4	3.18.3
GNOME Terminal	3.6.2	3.18.3
PulseAudio	4.0	8.0
BlueZ	4.101	5.37
サーバアプリケーション		
Apache	2.4.7	2.4.18
Nginx	1.4.6	1.9.12
Node.js	0.10.25	4.2.6
MySQL	5.5.47	5.7.11
SQLite	3.8.2	3.11.0
PostgreSQL	9.3+154	9.5+172
Samba	4.1.6	4.3.6
livbirt	1.2.2	1.3.1
QEMU/KVM	2.0.0	2.5
Docker	1.5	1.5
LXC	1.0.8	2.0.0
rsyslog	7.4.4	8.16
OpenStack	2014.1.5	13.0.0
MAAS	1.7.6	2.0.0
開発言語やドキュメンテーション		
GCC	4.8.2	5.3.1
Python 2	2.7.3	2.7.11
Python 3	3.4.0	3.5.1
Ruby	1.9.3	2.3.0
Perl	5.18.2	5.22.1
Rakudo	2013.12	2015.11
PHP 5	5.5.9	5.6.17
PHP 7	-	7.0.4
Go	1.2.1	1.6
Qt	5.2.1	5.5.1
TeX Live	2013.20140215	2015.20160320
Sphinx	1.2.2	1.3.6
Pandoc	1.12.2.1	1.16.0.2

IBus/AnthyからFcitx/Mozcへ

Ubuntu標準の入力メソッドフレームワークがIBusからFcitxに移行しました。日本語環境における、かな漢字変換システムもAnthyからMozcに変わっています。これにより日本語Remixを使わなくても、インストール直後からより快適な日本語入力環境を構築できます。詳しくは第2章を参照してください。

余談ではありますが、絵文字グリフを提供するfonts-symbolaが最初からインストールされるようになりましたので、インストール直後から絵文字の表示や入力ができるようになりました(**図3**)。また、GoogleとAdobeが共同開発した高品質なフォントであるNoto Sans CJKもfonts-noto-cjkパッケージとして提供されています。

デスクトップ関連

前述のとおりGNOME関連のパッケージが大幅にアップデートされています。とくにGNOME端末は長年の懸案だった、「あいまいな文字幅(East Asian Width)」を調整できるUIが追加されています(**図4**)。

比較的多くのユーザに影響する部分としては、Network Manager、PulseAudio、BlueZのメジャーバージョンが上がっています。それぞれ無線・有線ネットワーク、サウンド、Bluetoothに影響しますので、アップグレード前にライブ環境でこれらのデバイスが動作することを確認しておきましょう。

X.orgやMesaの更新により、Intel Skylakeマシンでもとくに不自由なく3Dアクセラレーション機能を利用できるはずです。

デスクトップの新機能、とくにほかのフレーバーの状況については第2章を参照してください。

日本語Remixのリリースは?

Ubuntu 16.04 LTSでも日本語Remixを提供する予定です。開発リソースの都合でリリースできるのは64ビット版のみとなるでしょう。

以前の日本語Remixは、本家には取り込まれていない日本語環境向けパッケージを提供したり、標準の日本語入力環境をより使いやすくするなどのカスタマイズを行っていました。しかしながら、最近はこれらのカスタマイズのほとんどは本家に取り込まれています。たとえば日本語入力環境はインストールした段階でFcitx・Mozcを利用します。不具合もできるだけ本家のリリース前に洗い出し、本家のイメージで修正されるように努めています。

インストールイメージの中に日本語言語パックが入っているなど、日本語Remixならではのメリットがないわけではありません。よって今後も日本語Remixをリリースし続ける可能性はあります。しかしながら普通にUbuntuをインストールするだけなら、あえて日本語Remixを待つ必要はありません。本家のUbuntuをインストールしたうえで、インストール時に日本語を選択するだけで、日本語Remixと変わらない状態になるはずです。SD

▼図3　絵文字フォントの導入

▼図4　あいまい幅の設定

第2章 デスクトップで比較する Ubuntu 16.04 LTSとそのフレーバー

Ubuntu Japanese Team
Author あわしろいくや

本章ではデスクトップ向けUbuntuの紹介と、その公式派生版であるフレーバーの特徴を、14.04との変更点を交えて紹介します。

Ubuntuデスクトップ

Ubuntuデスクトップは、その名のとおりUbuntuの中核となるデスクトップ向けLinuxディストリビューションです。正式な名称は“Ubuntu”ですが、Ubuntuには意味するものがいろいろとあるので、本稿では“Ubuntuデスクトップ”と表記します。

Ubuntuデスクトップの特徴は何といってもUnityです(**図1**)。Unity Dashは検索主体のインターフェースでアプリケーションの起動やファイル検索などが行えます。左側に表示されるLauncherではショートカットの追加や削除はもちろん、現在起動しているアプリケーションの表示や進捗アニメーションの表示といった機能まであります。インジケーターには独自のライブラリ(libappindicator)を開発していましたが、現在はKDE PlasmaやCinnamonなど、ほかのデスクトップ環境にも採用され、広く普及しています。

▼図1 Unityは下にあるアイコン(レンズ)で検索対象を切り替える

●Unityのバージョン

16.04のUnityのバージョンは7.4で、14.04の7.2からあまり大きな変更は行っていません。その理由は現在新バージョンであるUnity 8の開発を精力的に行っているからです[注1]。デスクトップとスマートデバイスで同じデスクトップ環境を使用するという大きな野望を達成するために、ディスプレイサーバ(Mir)から開発するという大掛かりなことをやっており、もともとの予定では16.04はUnity 8にバージョンアップした最初のLTSになる予定でしたが、やはり作業が遅れて、実際のリリースはもう2年後の18.04が最初のLTSになってしまいそうです。言い換えれば、16.04はUnityがデスクトップとスマートデバイスを統合する前にリリースした最後のLTSとなる予定です。

UnityのウィンドウマネージャーはCompizで、動作には3Dアクセラレーションが必須です。現在は3Dアクセラレーションが使用できない場合はソフトウェアエミュレーションでも動作しますが、非常に遅いです。また、そういった場合はハードウェアスペックが低いことが多く、いずれにしてもまったく実用的ではありません。ですから、現在3Dアクセラレーションが動作しないRaspberry Piシリーズでは、Unityは使用しません。

注1) 詳しくは158ページのUbuntu Monthly Reportをご覧ください。

●GNOMEのアプリケーション

Ubuntuデスクトップを構成するアプリケーションは、基本的にはGNOMEデスクトップ環境のものを採用しています。ファイルマネージャーである**ファイル(Nautilus)**、エディタである**gedit**、端末である**GNOME端末**、音楽プレーヤである**Rhythmbox**などが代表格です。設定ツールと設定デーモンは、GNOMEのもの(gnome-settings-daemon/gnome-control-center)をフォークし、独自にメンテナンスしています(unity-settings-daemon/unity-control-center)。Webブラウザは**Firefox**で、常に最新バージョンに更新されます。オフィススイートはもちろん**LibreOffice**です。こちらは常に最新バージョンになるわけではありませんが、マイナーバージョンアップには追随します。たとえば16.04のLibreOfficeは5.1で、最終的にはマイナーバージョンアップの最後(予定では5.1.6)までは追随しますが、5.2には上がりません。メジャーバージョンアップを行いたい場合はPPAを追加するというのが定石になっています。

GNOMEデスクトップ環境は、Ubuntuデスクトップと同じく半年に一度リリースされています。Ubuntuは可能な限り開発中にリリースされているGNOMEの安定版にアップデートするようにしています。具体的には、16.04ではGNOME 3.18相当にアップデートしています。しかし、Ubuntuで採用するのに都合が悪い点がある場合はバージョンを据え置きます。それによって不都合が発生する場合だけフォークをします。前述の設定ツールと設定デーモンはこのパターンです。しかし、そうしているものはあまり多くありません。

●UIガイドラインとその他

実例を挙げますと、**gedit**は長らくバージョンが3.10に据え置かれていましたが、このたび3.18になりました。一方**ファイル(Nautilus)**は3.14のまま据え置かれています。このようになる理由は割とシンプルで、GNOMEはUIガイドライン(GNOME Human Interface Design)があり[注2]、これに合わせるべく開発を行っているためです。しかし、このガイドラインは従来のアプリケーションとはまったく異なり、Unityには合わない部分があります。それをすり合わせる作業はUbuntu自身でやる必要があり、できた場合、あるいはやる必要がない場合はバージョンアップ、できない場合は見送りとなります。

たとえば、「**gedit**はすり合わせ作業が無事にできたものの、**ファイル(Nautilus)**はできなかった」ということになります。後者は実際試みられたものの、やっぱりできないということでバージョンが差し戻されました。パッケージのバージョンが"1:3.18.4.is.3.14.3-0ubuntu2"になっているところからもわかります(3月中旬現在)。

もちろんUbuntuデスクトップを構成するアプリケーションはバージョンごとに微調整が入り、あらかじめインストールされていたアプリケーションが追加されたり削除されたりといったことはよくあります。第1章ですでに紹介しているので繰り返しませんが、16.04では**Ubuntuソフトウェアセンター**がなくなり、**ソフトウェア(gnome-software)**になったのが大きな変更点です。

Ubuntu Japanese Teamは「Ubuntu 日本語Remix」という、Ubuntuデスクトップに日本語環境で使用する際に便利な機能を付加したインストールイメージを配布しています。とはいえ、最近は変更点が減ってきているうえ、変更点のみのインストールも可能なため、積極的に利用する必要性は減ってきています。

LTSということで、サポート期間は長めに設定されています。Ubuntuデスクトップは5年、フレーバーは3年というのが原則ですが、例外もあり、フレーバーであっても5年サポートのものもあります。

Kubuntu

Kubuntuは Ubuntu初のフレーバーであり、紆余曲折がありながら、現在も精力的に開発が継続しています(**図2**)。デスクトップ環境はKDEで、構成して

注2) https://developer.gnome.org/hig/stable/

いるアプリケーションはほぼUbuntuデスクトップとは異なります。例外はFirefoxで、これはほかのフレーバーでも同じですが、Webブラウザはセキュリティサポートが重要で、UbuntuにはFirefoxほど迅速にアップデートを行っているWebブラウザがないので、ほかに選択肢がないというのが実情です。Google Chromeのオープンソース版であるChromiumもuniverse[注3]にあるパッケージにしてはこまめに対応していますが、数日遅れならまだいいほうで、過去にはアップデート自体をスキップしてしまうことすらありました。

以前のKDEは1つのデスクトップ環境としてリリースされていましたが、4以降はKDE Plasma、KDE Frameworks、KDE Applicationの3つに分割されてリリースされ、それらを総合してKDE SC（Software Compilation）と呼んでいます。16.04のリリース時点で、KDE Plasmaはバージョン5.4.4、KDE Frameworksはバージョン5.18.0、KDE Applicationsは15.12.1です[注4]。

14.04のKubuntuはKDE SC 4だったので、16.04では大幅に変わったことになります。KDEのバージョンはツールキットであるQtのバージョンに依存しており、KDE SC 4だとQt 4、5だとQt 5となります。Qt 4は昨年末でEOL（End of Life）を迎えていますが、14.04はLTSですので5年間サポートされます。すなわち、あと3年間は使用できるので、どうしてもKDE SC 5に馴染まない場合はそちらを使用するのも手です。

残念ながらKDEの日本コミュニティは活動が活発とはいえず、メニューは英語のままのところが圧倒的に多いので、使いこなすのであればそれなりの英語力を必要とします。個人的にはKDEユーザが増え、それに伴って翻訳者も増えるといいと思うのですが、それはさておき簡単に使うのであればとっつきはさほど悪くありません。左下のメニューもカテゴリごとにわかれており、直感的に起動できます。しかし、さらに変更したい場合は、左下のアイコンを右クリック-[Alternatives]-[アプリケーションメニュー]-[切り替え]で、Windows 7のようなメニューに変更できます。[Alternatives]-[Application Dashboard]-[切り替え]にすると、メニューが画面いっぱいに表示されるようになります。

KDEを使用するのであれば、ウィジェットを活用すると便利です。これはその名のとおり、デスクトップまたはパネルにシンプルな機能のウィジェットを貼り付けます。デスクトップを右クリックして[Add Widgets]をクリックするか、左上のボタンをクリックして[Add Widgets]をクリックするとウィジェット一覧が表示されます。また、KDE Plasmaの[Get Hot New Stuff（日本語訳は[ホットな新しい物を取得]）]機能を利用して、[新しいウィジェットを入手]からインストールされていないウィジェットも使用できるようになります。

注3）Ubuntuのリポジトリはmain/universe/restricted/mutiverseに分かれており、mainとrestrictedはCanonicalによるサポートがあり、セキュリティ修正も行われます。universeとmultiverseにはなく、コミュニティによる対応がなければセキュリティ修正が行われないこともあります。
注4）リリースの前後でバージョンアップする可能性はあるので、このバージョンは目安だと思ってください。

▼図2 Kubuntuのデスクトップ。右上にあるのはメモウィジェット

Xubuntu

Xubuntuは、デスクトップ環境にXFCEを採用したフレーバーです（**図3**）。これも結構歴史があります。Xubuntu自体は活発に開発されていますが、XFCE自体は活発に開発されているとはいい難い状況です。14.04ではXFCEのバージョンは4.11という開発版だったのですが、16.04ではこれのリリース版である4.12であり、あまり大きな違いはありま

せん。14.04から16.04にアップグレードしても、壁紙が変わったことぐらいしか気づかないかもしれません。余談ですが、14.04の壁紙に変更することもできます。

変更されたパッケージもあまり多くはありません。14.04との大きな差としては、AbiWordとGnumericの代わりにLibreOffice Writer/Calcが採用されました。LibreOfficeはAbiWordとGnumericのファイルが読み込めるため、あまり大きな問題にはならないはずです。もちろんAbiWordとGnumericはリポジトリから削除されたわけではないため、必要な場合は別途インストールすることはできます。

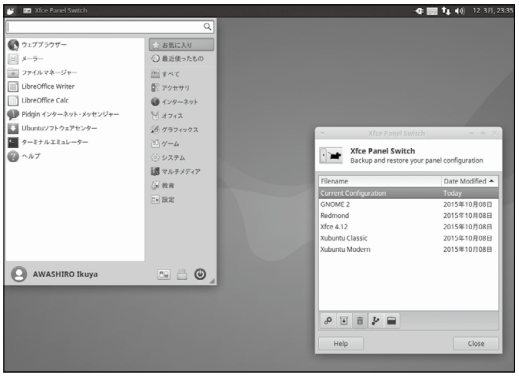
音楽プレイヤーであるgmusicbrowserと、知る人ぞ知るGIMPも削除され、使用する場合は別途インストールする必要があります。後者はXubuntuの特性を考えると削除されるのも納得できるのですが、前者は理由がよくわかりません[注5]。ただ、いずれにせよ音楽プレイヤーであればAudaciousを使ったほういいでしょう。

14.04以降新たに追加されたのはいずれも設定ツールで、LightDM GTK+ GreeterとXfce Panel Switchです。どちらも名前を見たらどういったものなのか予測できます。前者はデスクトップマネージャーであるLightDMの設定ツールであり、後者はXFCEのパネルのデザインを切り替えます。

XFCEの将来性には大きな心配があるものの、現在軽量なデスクトップ環境として使用するのであればベストな選択といえます。Raspberry Piシリーズや仮想環境で使うのであれば、軽さと機能のバランスを考えた場合、採用を検討するべきです。

注5）おそらくクラッシュバグが修正できなかったからだとは思われます。

▼図3　Xubuntuのデスクトップと[Xfce Panel Switch]

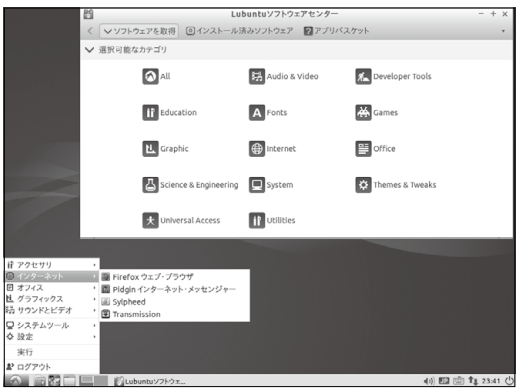


Lubuntu

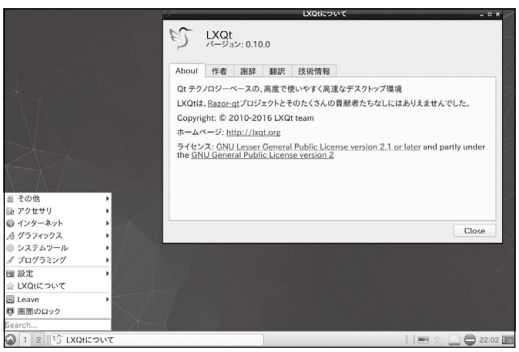
Lubuntuはデスクトップ環境にLXDEを採用したフレーバーです（図4）。LXDEは現在、その後継となるLXQtを開発中です（図5）。LXQtはその名のとおり、ツールキットをGTK+ 2ではなくQt 5に変更しています。開発の主力はそちらに移っていますが、LXDEも開発を停止したわけではなく、活発に開発されています。

LXDEはとにかくシンプルかつ軽量で、機能もプリミティブです。ウィンドウマネージャを独自に開発しないという変わった方針のデスクトップ環境であり、徹底しています。なお、これはLXQtも同様

▼図4　Lubuntuのデスクトップと Lubuntu ソフトウェアセンター



▼図5　LxQtのデスクトップとバージョン情報



であり、いずれもOpenboxと組み合わせて使われることが多く、Lubuntuもそのようになっています。

LXDEの方針を受け、Lubuntuもほかのフレーバーにはない特徴があります。それは、いまだにテキストベースのAlternativeインストールイメージを配布していることです。これはすなわちGUIインストーラ(Ubiquity)が動作しないようなPCにもインストールすることを意図しています。目安としてはメモリ512MBのPCです。インストールイメージもCDサイズを死守すべく相当な努力をしていますが、テキストベースのインストーラではCDサイズをなんとか維持しています。しかし、GUIインストーラではDVD-Rが必須となっています。

以上の理由により、Lubuntuは次のような場合にお勧めです。

・メモリが512MBしかないPC
・DVDドライブがなく、CDドライブしかないPC
・用途が極めて限定される場合

これらに該当しない場合はXubuntuがお勧めです。

古いPCだとCPUがPAE(Physical Address Extension)に対応しておらず、Lubuntu(を始めとしたUbuntuデスクトップまたはそのフレーバー)が起動しないということがあります。その場合、一部のCPUではブートオプションに"forcepae"を追加すると起動できます。詳しくはWiki[注6]をご覧ください。

LubuntuがLXQtに移行するのは16.10以降ですが、16.04でもインストールして試してみることぐらいはできます。詳しくはWiki[注7]をご覧ください。

Ubuntuデスクトップやほかのフレーバーでは**Ubuntuソフトウェアセンター**が**ソフトウェア**になったのが大きなトピックであることはすでに述べましたが、これはLubuntuには該当しません。なぜなら、独自の**Lubuntuソフトウェアセンター**があるからです。とはいえ、検索ができず、カテゴリごとに並べているだけですのでお世辞にも使いやすいとはいえません。

注6) https://help.ubuntu.com/community/Lubuntu-fake-PAE
注7) https://wiki.ubuntu.com/Lubuntu/LXQt

Ubuntu GNOME

Ubuntu GNOMEは、その名のとおりデスクトップ環境にGNOMEを採用したフレーバーです(**図6**)。比較的新顔で、14.04に続き2回目のLTSリリースです。Ubuntuのリポジトリにあるパッケージでできる限り純粋なGNOMEデスクトップを提供することを目的としており、事実そうなのですが、前述のとおりUbuntuのリポジトリではGNOMEのバージョンがそろっているわけではありません。

14.04のUbuntu GNOMEはGNOME 3.10ベースです。そして、16.04はGNOME 3.18ベースで、かなり大きな変更があります。GNOME Shellの拡張機能はバージョンアップによって動かなくなるものも多いので、アップデートする場合は事前に自分が愛用している拡張機能が新しいGNOME Shellに対応しているかを確認する必要があります。

なお、16.04のリリース時点でGNOMEは3.20が最新版になっている見込みです。すなわち、Ubuntuの開発サイクルで最新だったバージョンが採用されており、Ubuntu GNOMEの最新バージョンイコールGNOMEの最新バージョンとはなっていません。最新の純粋なGNOMEを使用したいのであれば、Fedoraをインストールするのがベストです。とはいえ、中にはGNOMEの最新版に含まれているアプリケーションがUbuntuのリポジトリにも入ることはあり、16.04のサイクルでは**ソフトウェア**と**カレンダー**

▼図6 GNOME Shellのアクティビティ画面。右側にあるのはワークスペース

とシンプルスキャンが相当します。とくにソフトウェアは、Ubuntuソフトウェアセンターを置き換えるべく、Canonical社員も精力的に開発に参加していました。

最近のGNOMEは新しいアプリケーションを続々と追加しており、それがUbuntu GNOMEでもあらかじめインストールされるようになっているので、そのあたりが14.04との違いにもなっています。具体的には**地図・天気・音楽・写真・ソフトウェア・カレンダー・ログ**です。GNOMEのアプリケーションは一般的な名前を付けるようになっているので、パッと見どんな役割なのかはわかりやすいのですが、そもそもそれがGNOMEのアプリケーションなのかどうかがわかりにくいというデメリットもあります。一方、**写真**が入ったことにより**Shotwell**はインストールされなくなりました。しかし、実際に触ってみるとわかりますが、**写真**は**Shotwell**を置き換えるにはちょっとシンプル過ぎるので、必要であれば**Shotwell**をインストールしてください。**音楽**は入っても**Rhythmbox**はそのまま残されました。

GNOME Shellの操作性は、筆者としてはすごく優れていると思うのですが、とっつきにくいのは事実です。それが影響してか、初回ログイン時にヘルプが表示され、基本的な操作はビデオで見ることができるようになっています。もう一度見たい場合は、[ヘルプ]-[GNOMEを初めて使う方へ]をご覧ください。

GNOMEはシンプルで、あまりカスタマイズできないことで有名です。しかし、これはあらかじめインストールされている**Tweak Tool**を使えば、ある程度のカスタマイズはできます。たとえばタイトルバーに最大化や最小化のボタンをつけたい場合は、[ウィンドウ]タブにある[最大化]と[最小化]をオンにしてください。

Red Hat Enterprise Linux 7あるいはCentOS 7をGUIつきでインストールすると、GNOMEクラシックになります。これはGNOME Shellに昔のGNOMEっぽい拡張機能を適用したもので、もちろんUbuntu GNOMEにもあります。ログイン時に歯車アイコンをクリックし、[GNOMEクラシック]を選択するだけです。また、ディスプレイマネージャーとしてX.orgを置き換えるべく開発中のWayland/Westonを試用してみたい場合は、"gnome-session-wayland"パッケージをインストールし、デスクトップマネージャー(GDM)を再起動してから歯車アイコンの**GNOME on Wayland**をクリックしてください(**図7**)。ただし、今のところ仮想マシンでは動作せず、実機である必要があります。また、その場合もプロプライエタリなドライバをインストールしていると動作しません。このとおり、現在はかなり制限が強いですが、いざ使ってみると割と普通に動きます。

Ubuntu MATE

MATE(マテ)はGNOME 2.xからフォークしたデスクトップ環境です(**図8**)。GNOME 2.xのルック&

▼図7　ログイン時に選択できるセッション

▼図8　Ubuntu MATEのデスクトップとWelcome

フィールや機能はあまり変えず、可能な限り古いライブラリの使用をやめ、最近のものを使用するように書き換えるべく開発を行っています。現在の最新バージョンは1.12で、次のバージョンあたりでGTK+3へのポーティングが完了しそうです。

MATEはGNOME 2.xのライブラリやアプリケーションをまるごとフォークしており、それぞれ独自の名前を付けています。たとえばファイルマネージャー(Files/Nautilus)は**Caja**、テキストエディタ(gedit)は**Pluma**、ドキュメントビューア(Evince)が**Atril**などです。**GNOME端末**は**MATE端末**と、"GNOME"がついているものはそのまま"MATE"に置き換えています。

GNOMEは3になってからすべての翻訳が見直され、今どきのポリシーで統一されました。しかし、MATEはその前にフォークされたものであり、古い翻訳と新しい翻訳が混在しています。ものによっては統一されたものもありますが、多くはそうなっていません。よって、若干見にくい部分があります。

Ubuntu MATEは、そんなMATEをデスクトップ環境にしたフレーバーです、公式フレーバーとなったのが15.04ですので、初のLTSとなります。MATEデスクトップ環境のアプリケーションは一通り収録しており、それ以外は原則としてはUbuntuと同じになるようにしています。例外的に、動画再生ソフトとしてQtアプリケーションであるにもかかわらず**VLC**が採用されています。

Ubuntu MATEには大きく2つの独自アプリケーションがあります。1つは**Welcome**で、その名のとおりログイン時に自動的に起動し、Ubuntu MATEの紹介やアプリケーションのインストールなどが(英語で)行えます。

もう1つは**MATE Tweak**で、その名のとおりMATEのカスタマイズツールです。このツールのおもしろいところは、パネルのレイアウトを簡単に変更できるところです。Windows風、OS X風はもちろん、懐かしのGNOME 2風やUnity風なんてのもあります。Unity風は、メニューを画面上部に表示することまでやってしまうという凝りようです。

Ubuntu MATEから派生したプロジェクトとして、"Ubuntu Pi Flavour Maker"というのがあります。これについては先月号のUbuntu Monthly Reportで詳しく解説したので、そちらをご覧ください。

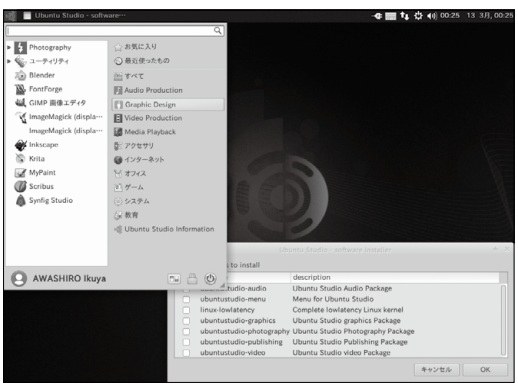
GNOME 2.xに慣れている人であればこれほど使いやすいものはないでしょう。また、3Dアクセラレーションも必要ないので、仮想環境などにも適しています。ただ、アプリケーションのチョイスからも軽量なLinuxディストリビューションを目指しているわけではないので、ハードウェアのスペックが潤沢でない場合は避けたほうがいいかもしれません。

Ubuntu Studio

これまで紹介したフレーバーはいずれもデスクトップ環境の違いが特徴になっていました。中にはそうではないフレーバーもあり、そのうちの1つがUbuntu Studioです(**図9**)。これは映像・音楽・デザインなどのクリエイター向けアプリケーションが大量に用意されています。そればかりか、カーネルもレイテンシが少なくなるコンフィグにした特別なものになっています。大量のアプリケーションがあるため、インストーラのイメージのサイズもほかのものよりかなり大きくなっており、フルインストールする場合はより多くの空きスペースが必要です。なお、デスクトップ環境はXFCEです。

レオナルド・ダ・ヴィンチのような万能人はともかく、映像・音楽・デザインなどすべてを行う人はなかなかいないわけで、すべてのパッケージを必要

▼図9 Ubuntu Studioのデスクトップと独自のパッケージインストーラ



とすることはあまり多くないのではないかと思います。そういった場合、インストールの段階で何をインストールするかを選択できます。また、あとから追加したくなった場合に簡単にインストールできるツールもあるので安心です。

何か作りたいものがあるとき、とりあえずUbuntu Studioをインストールしてみて、そのジャンルにどんなアプリケーションがあるのか確認し、実際に使用してみて手に馴染むものを探す、というのが正しい使い方だと思います。

インプットメソッド（日本語入力）

Ubuntuデスクトップとそのフレーバーでは、インストール時に日本語を選択するとFcitxとMozc一式がインストールされます。14.04ではIBusとAnthy一式でしたが、日本語Remixでは先行して16.04と同様にFcitxとMozc一式がインストールされていました。原則としては各種フレーバーでも同じパッケージがインストールされますが、Kubuntuでは“kde-config-fcitx”という専用の設定モジュールがインストールされます。

ではIBusは削除されたのかというとそのようなことはなく、Fcitxとともにインストールされています。これまでは“ibus-anthy”がインストールされていましたが、16.04からは“ibus-mozc”がインストールされるようになったため、IBusに切り替えても使い勝手が変わるようなことはありません。

IBusに切り替える方法は簡単で、**言語サポート**がインストールされている場合はこれを起動し、[キーボード入力に使うIMシステム]を[IBus]に変更して一度ログアウトし、再ログインするだけです。**言語サポート**がない場合、あるいはコマンドから行う場合は、端末を起動して“im-config -n ibus”を実行し、一度ログアウトして再ログインしてください。Fcitxに戻す場合は、前述のコマンドの“ibus”を“fcitx”にするだけです。

Ubuntu GNOMEではIBusがデフォルトになり、Fcitxに変更したい場合は別途設定が必要になる予定です。GNOMEではUnityと異なり、システム設定の[地域と言語]あるいは[入力ソース]からFcitxの入力ソース（AnthyやMozcなど）が設定できないという理由からです。また、Fcitxだとアイコンが左下に表示されてしまうという点もあります。これはFcitxがGNOME Shellのトレイアイコンに対応しておらず、またGNOME Shellがlibappindicatorに対応していないから起こることです。このままでも問題ないといえばそうなのですが、Fcitxを使用したい場合は拡張機能[注8]をインストールして右上に表示するといいでしょう。

Kubuntuの独自設定ツールを見てみましょう（**図10**）。設定できる項目はデフォルトの設定ツールと同様ですが、**スキンの管理**というFcitx用スキンのインストーラが付いており、ほかの設定ツールよりも高機能です。

Ubuntuデスクトップだけですが、接続してもいない英語キーボードを登録してしまいます。これは現在の制限事項となっているので、英語キーボードを接続していないのであれば削除してしまってもいいですし、何もしなくても問題ありません。Fcitxの仕様上、[入力メソッド]の一番上が普段使用しているキーボードでなくてはなりませんが、これを入れ替えるようなことはしないでしょうし、またMozcからほかの入力メソッドに切り替えるようなことがなければ邪魔にもならないでしょう。SD

注8）https://extensions.gnome.org/extension/1031/topicons/

▼図10　KDEのFcitx設定ツール。一番右に[スキンの管理]タブが見える

第3章 Ubuntu Server 16.04 LTSの特徴

Juju、MAAS、LXDなどの独自機能で際立つ

Ubuntu Japanese Team／(株)創夢
Author 吉田 史(よしだ ふみひと)

本章ではUbuntu Serverの紹介と、JujuやMAAS、LXDなどの紹介、16.04で更新された機能などを紹介します。

Ubuntu Server 16.04 LTSの機能

近年のUbuntu Serverは、「Debianに近似したパッケージ構成と、5年間のコミットされたサポートが提供される、無償でも利用できるディストリビューション」であるだけでなく、JujuやMAAS、LXDなどの独自の機能が提供されるようになっています。16.04で更新された機能を含め、Ubuntu Serverの特徴を見ていきましょう。

●Cloud Images

Ubuntu Serverの大きな特徴の1つが、インストール媒体としてISOイメージだけでなく、各種クラウドサービス・仮想化環境向けに「すでにインストール済みの」マシンイメージであるCloud Imagesもリリースされることです。

16.04でもこれまでと同様、Cloud Imagesがリリースされています。https://cloud-images.ubuntu.com/からリンクをたどり、https://cloud-images.ubuntu.com/releases/16.04/release/から利用してください。

これらは、Ubuntu Serverを最小構成でインストールし、cloud-initをインストールしたものに相当します。16.04では、QCOW2(QEMU・KVM用)、VHD(HyperV・Azure用)、Vagrant box・OVAと、LXD用tarball、tar(/を単純にtarballにしたもの)が提供されます。ユーザが自分でダウンロードして利用するほかに、パブリッククラウド上に準備されているイメージもあります。Cloud Image Finder[注1]で検索し、必要なテンプレート名・マシン名などを確認してください。

●IBM LinuxONEのサポート

Ubuntu 16.04 LTSのサポートアーキテクチャはi386・amd64・armhf・arm64・ppc64です。本稿の校正時点(2016年3月)では未リリースですが、16.04 LTSはIBM z13をベースにした『LinuxONE』向けにもリリースされる予定です(s390x)。これにより、メインフレーム上にUbuntuをインストールし、JujuやLXDを活用できるようになります。

基本的にすべてのバイナリは再コンパイルされ、なんらかのコンパイル上の問題でパッケージが作成できないものを除き、x64比で約95%のパッケージが利用できる予定です。

LXD/LXC

14.04から16.04への更新において、大きなポイントとなるのがLXDです。LXDを用いることで、仮想マシンのような操作感覚でコンテナを利用できます。

●LXD

LXD[注2]は、Ubuntu/Canonicalによって開発される、liblxcベースのコンテナを扱うためのソフトウェアで、しばしば「コンテナのハイパーバイザー」と表現されます。

注1) https://cloud-images.ubuntu.com/locator/
注2) 「エルエックスディー」ではなく、"lex - dee"と発音します。カタカナ表記では「レクスディー」です。

Ubuntu上で動作するデーモン(lxd)と、REST APIで操作を行うコマンドラインツール(lxcコマンド)によって構成されます。「ゲスト」にあたるコンテナは、liblxcがサポートするLinuxディストリビューションであれば、たいていのものが利用できます。

また、単なる仮想マシン的な用途だけでなく、nova-compute-lxdパッケージを利用してOpenStack Novaのcompute driverとしても利用できます。

●既存の実装との違い

LXDは、コンテナ技術を利用して「仮想マシンに近い環境」を、作り出すソフトウェアです。コンテナ技術の代名詞になりつつあるDockerとは、いくつかの点で異なります。

DockerとLXDは、どちらも「コンテナを管理する」ソフトウェアです。

Dockerは「特定のデーモンを実行するための環境を隔離して構成する」ことが主眼にあります。Dockerを利用することで、特定のサービスを実現するためのデーモンを、隔離した環境で動作させることができるのが狙いです。

これに対して、LXD/LXCは、「initデーモンを含めたOS全体を動作させる」「複数のデーモンを同じコンテナ内で動作させる」方向の実装です。カーネルを共用する、複数のマシンが動作するイメージです。FreeBSDのJailやSolarisのZoneに近い機能です。操作感覚としては、Vagrantによる仮想マシンの操作に似ています。

LXDで動作する「コンテナ」はごく普通のLinux環境です。たとえば、Dockerで走らせるコンテナに対話的にログインすることは通常はほとんどありえませんが、LXDでは珍しくありません。Dockerのように根本的な発想の転換を必要とせず、仮想化環境のメタファで操作でき、複数のマシンで構成される環境から、設計変更を伴わずに移行できる、という点がメリットです。また、Dockerと同様、必要最低限のサービスだけを動かす環境として利用することもでき、仮想化の代替からコンテナによる隔離まで、幅広く対応します。

こうした点ではLXDは、Dockerだけではなく、VMwareやESXi、KVMといった仮想化ハイパーバイザーを置き換える目的にも利用できます。既存の仮想化に比べると、ホストマシンのカーネルをそのまま利用できるため、オーバーヘッドがほとんどなく、大量の仮想マシンを動作させることができることも特徴です。LXDは基本的に「親」環境のプロセスをcgroupなどで隔離して動作させるだけですので、マシン台数を増やしてもメモリ使用量がほとんど増えないためです。8GB程度のメモリがあれば、(単にマシンが上がるだけでよければ)数百台程度のマシンを動作させることができます。

●16.04 LTSのLXD

LXDはUbuntu 15.10で搭載された新機能で、「ネットワークごしに利用できるLXC」とも言えるコンテナ環境です。LXDを使うことで、手軽にコンテナを作成できます。

16.04 LTSではmainコンポーネントに移動され、かつ、Ubuntu Server環境ではプリインストールされるパッケージに含まれるようになりました。内部的には、taskselで利用される「server」「cloud-image」タスクに含まれるようになっています[注3]。このtaskselへの登録により、CDからUbuntu Serverをインストールした場合だけでなく、Cloud Images上でもデフォルトで利用できるようになっています。ある種のプレビュー段階から、実用モードに切り替わったと言えるでしょう。

●LXCとLXD

既存のLXC[注4]との違いは次のとおりです。

- REST APIを経由して、ほかのマシンにコンテナを展開できる
- コンテナはイメージベースで管理される(LXCでは「ディストリビューションテンプレート」をもとに、コンテナ起動時にマシンを生成するアプロー

注3) taskselは、「ubuntu-desktop」や「server」「samba-server」などといった、「ある役割を果たすときに必要と考えられるパッケージ」をまとめてインストールするためのしくみです。

注4) 「LXC」という単語には、「Linux Containerの省略名」(liblxcなど)と、「LXCというコンテナ管理ソフトウェア」(以前のLXC)と、「LXDのクライアント層」という3つの異なる意味があります。

チを取っていました）
・各種プログラムがGoで書き直された

●LXDの使い方

16.04環境では、LXDは（**図1**）のように利用します。前述のとおり、Ubuntu Server環境であればLXDは初期状態でインストールされており、追加インストールは不要です[注5]。

lxd initは、LXDの初期設定を行うラッパーです。lxd initは、2016年3月時点ではストレージバックエンドとして、ディレクトリ・ZFSを利用する環境をサポートしています。

16.04以前の環境でLXDを利用することもできます。この場合は、（**図2**）のように操作してPPA経由で最新版のLXDを取り込んでからlxd initを実行します。16.04環境でもこの手順を利用できます。

初期設定が終了したら、利用するイメージを選択します。LXDでは、「イメージ」ファイルを読み込んでコンテナを起動するため、イメージを登録する必要があります。イメージの登録方法は、大きく分けて2つあります。リモートサーバからの入手と、手動インポートです。

リモートサーバから入手する場合、16.04のLXDには、linuxcontainers.org上の各種Linuxディストリビューションのイメージと、Ubuntu Cloud Images（リリース版とデイリービルド）が事前に登録されています（**図3**）。これら以外に、すでに作成され、イメージを保持しているLXDが動作するマシンを登録することもできます[注6]。

リモートサーバ上にあるイメージからコンテナを起動するには、「lxc launch ubuntu:16.04 xenial」などとします。このコマンドラインの意味は、「remote listの“ubuntu”にある16.04のイメージを用いて、“xenial”という名前のマシンを起動する」です。

イメージのダウンロードが実行され、しばらくするとコンテナが起動します。起動したコンテナは、「lxc list」コマンドで一覧できます。起動したコンテナ上でコマンドを実行する「lxc exec」や、ファイルをやりとりする「lxc file」コマンドを利用してコンテナをカスタマイズします。「lxc exec xenial -- /bin/bash」などとすることで、シェルを取得することもできます。

◆ストレージバックエンド

LXDは、ストレージバックエンド

注5） ただし、LXDは非常に進歩が早いため、PPA経由で最新版を用いる方が安全なことが多いでしょう。

注6） lxc remoete addコマンドを使います。

▼図1　LXDでの実行例

```
$ sudo lxd init
Name of the storage backend to use (dir or zfs): dir
 ※ ストレージバックエンドを選択する
Would you like LXD to be available over the network (yes/no)? no
 ※ ネットワーク経由での利用をするかどうかを選択する
    単一ホストで利用する場合はno
LXD has been successfully configured.
```

▼図2　PPA経由で最新版のLXDを取り込んでから実行する

```
add-apt-repository ppa:ubuntu-lxc/lxd-stable
apt-get update
apt-get dist-upgrade
apt-get install lxd
```

▼図3　リモートサーバのリスト表示

```
$ lxc remote list
+-----------------+-----------------------------------------+---------------+--------+--------+
|      NAME       |                  URL                    |   PROTOCOL    | PUBLIC | STATIC |
+-----------------+-----------------------------------------+---------------+--------+--------+
| images          | https://images.linuxcontainers.org      | lxd           | YES    | NO     |
+-----------------+-----------------------------------------+---------------+--------+--------+
| local (default) | unix://                                 | lxd           | NO     | YES    |
+-----------------+-----------------------------------------+---------------+--------+--------+
| ubuntu          | https://cloud-images.ubuntu.com/releases| simplestreams | YES    | YES    |
+-----------------+-----------------------------------------+---------------+--------+--------+
| ubuntu-daily    | https://cloud-images.ubuntu.com/daily   | simplestreams | YES    | YES    |
+-----------------+-----------------------------------------+---------------+--------+--------+
```

としてBtrFS・LVM・ZFSをサポートしています。いずれも利用できない場合に、ディレクトリバックエンドが利用されます。効率よく多数のマシンを格納・生成したい場合、ディレクトリ以外のバックエンドを選択する必要があります[注7]。

◆ZFSサポート

LXDで利用することも含め、16.04 LTSでは「ZFSのサポート」が追加されています。これは、ZOL(ZFS on Linux)の成果物を取り込んだものです。

これまでのUbuntuでもZOLを利用できたものの、zfs-dkmsをインストールし、カーネル更新時などに動的にカーネルモジュール(zfs.ko)をコンパイルする、という手間のかかる手順が必要でした。16.04ではカーネルパッケージの一部としてzfs.koが同梱されるため、この手間を省略できるようになっています[注8](**図4**)。

ただし、ZFSを扱うためのユーティリティ(zfsutils-linux)はuniverseパッケージとして提供されるため、Canonicalによるセキュリティアップデートなどの対象はカーネルモジュールのみとなります。

ZFSはrootdiskには利用できず、また、64bit環境専用です。

●snapパッケージとしてのLXD

LXDは、「snap」パッケージでも提供されます。「snap」パッケージは、"Snappy" Ubuntu CoreやUbuntu Phone、Ubuntu Personalで利用されるパッケージ形式です。/以下を原則として編集できない"Snappy"環境であっても、コンテナ内に自分の必要なソフトウェアを導入することで目的を達成できるでしょう。

Snappy環境でのLXDのインストール方法は、`https://linuxcontainers.org/lxd/getting-started-cli/`を参照してください。

Juju

●Juju 1.x

Jujuは、「パッケージをインストールするように」サーバ環境を準備するためのしくみです。

JujuはChefやPuppet、Ansibleなどといった構成管理フレームワークと、それをラップするフレームワークとして登場し[注9]、現時点では、AWSのCloudFormationやAzureのAzure Resource Managerに近い、クラウド環境向けのテンプレートベースのオーケストレーションツールとなっています。

Jujuの基本的な動作は、「必要なホストを起動し、そしてシェルスクリプトやChef・Puppetなどの構成管理フレームワークを用いてサービスを設定し、複数のサービスを紐付けることで1つの『アプリケーション』を立ち上げる」というものです。ここで言う『アプリケーション』は、たとえばLAMPスタックを

注7) 詳細は、https://github.com/lxc/lxd/blob/master/specs/storage-backends.mdを参照してください。

注8) ただし、ZFSのライセンス(CDDL)とGPLv2の間には非互換があり、「静的にリンクされていないとはいえ、ディストリビューションとしてLinuxカーネルとセットで配布してよいのか」といった議論が存在します。http://vlog.dustinkirkland.com/2016/02/zfs-licensing-and-linux.html

注9) このあたりの差異がわかりにくいため、FAQに「Can I use Juju with Puppet or Chef or Ansible?」(https://jujucharms.com/docs/devel/about-juju#can-i-use-juju-with-puppet-or-chef-or-ansible?)という項目が追加されています。

▼図4 zfs.koパッケージとZFS関連モジュール

```
$ dpkg -S /lib/modules/4.4.0-9-generic/kernel/zfs/zfs/zfs.ko
linux-image-4.4.0-9-generic: /lib/modules/4.4.0-9-generic/kernel/zfs/zfs/zfs.ko

$ lsmod |grep zfs
zfs                  2801664  0
zunicode              331776  1 zfs
zcommon                57344  1 zfs
znvpair                90112  2 zfs,zcommon
spl                   102400  3 zfs,zcommon,znvpair
zavl                   16384  1 zfs
```

含むWordPressや、OpenStack・Hadoopなどが該当します。

こうした動作を記載するテンプレートファイルを、「Charm」と呼びます。Charmにサービスを起動するために必要なパッケージ名や初期設定ファイルなどを定義しておき、Jujuが起動したホストに必要な設定を行う、というしくみで動作します。Charmは、「Charm Store」(https://jujucharms.com/store)と呼ばれる集積サイトに集められ、簡単に利用できます。

Jujuの特徴は次のとおりです。

- 特定のクラウドに依存せず、複数のIaaS環境で利用できる(EC2、Azure、Google Compute Engine、Joyent、Rackspace、DigitalOcean(ベータ))
- MAAS＋OpenStackを利用することで、プライベートクラウド上にデプロイすることもできる
- LXCを使うことで、ローカルマシン上にデプロイすることもできる
- サービスごとの「Relationships」を設定することで、複数のサービスを連携させて動作させることができる
- 「Bundles」と呼ばれる、複数のレシピ(Charm)をまとめた定義も行える。これにより、HadoopやOpenStackのような複雑な構成を必要とするソフトウェアスタックを数操作でデプロイできる
- Juju GUIと呼ばれる、ブラウザベースのインターフェースが提供されている

「Relationships」というのは、たとえば「WordPressのバックエンドでMySQLが動作し、動作をNagiosで監視する」といった、ソフトウェアの関係性のことです。通常であれば管理者がひとつひとつ設定を追加していく必要がありますが、Jujuではこうした「ソフトウェアを連携させる」ための設定をCharmにあらかじめ記載し、「このWordPressのインスタンスはこのMySQLを利用する」といった指定を行うだけでソフトウェアを動作させることができます。

また、Jujuには「Scaling」という概念もあります。これは、Web環境におけるエッジサーバのような、スケールアウト可能なサービスを扱うためのものです。まとめて10台のWordPressフロントエンドをデプロイし、MySQLサーバと紐付ける、といった操作が簡単に行えます。

◆Jujuの利用

Jujuをセットアップするには、https://jujucharms.com/docs/stable/reference-releasesにある手順のとおりに操作します。2016年3月時点での操作は次のとおりです。juju-quickstartにより、必要な初期設定が一括で行われます。GUIのセットアップも自動で実行されます(Juju GUIのデプロイにもJujuが用いられます)。

Ubuntuでの操作例は**図5**のとおりです[注10]。OS Xではbrewを、WindowsやUbuntu以外のLinux環境ではインストールパッケージを用いてセットアップを行います。具体的な操作方法は前述のURLを参照してください。

デフォルト設定の『ローカル』環境で動作させる場合、クラウドイメージのダウンロードが行われます。日本国内で実施する場合はダウンロードに1時間程度かかるので注意してください。iftopなどの通信状態を確認できるソフトウェアを併用して動作状態を確認するとよいでしょう。

なお、操作手順の中でadd-apt-repositoryコマンドでPPAを追加しています。これは、JujuはUbuntuのリリースよりも速いペースで更新されるため、リリース版に含まれるパッケージではなく、PPAにあるパッケージを利用するのが原則となるためです。

quickstartの実行後、**図6**のように操作すると、

注10) Jujuは進歩が非常に早いため、PPAからの利用が強く推奨されています。

▼図5 Jujuのセットアップ

```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:juju/stable
sudo apt-get install juju-quickstart juju-core
juju-quickstart
```

▼図6 バックエンドにMySQLを持つWordPressのセットアップ

```
juju deploy wordpress
juju deploy mysql
juju add-relation wordpress mysql
juju expose wordpress
```

「バックエンドにMySQLを持つWordPress」を簡単にセットアップできます。この例が示すように、Jujuの操作は、「パッケージをインストールするように」という謳い文句のとおり、APTに似たものになっています。

Jujuは、Juju GUIを用いてブラウザから操作することもできます。デモサイトが用意されているので、http://www.ubuntu.com/cloud/jujuからアクセスしてみてください。

●Juju GUI 2.0

16.04のJujuには、「Juju GUI 2.0」と呼ばれる新しいWebインターフェースが搭載されています。Juju GUI 2.0では、Charmがアップロードされたcharm Storeが統合され、必要なCharmを簡単に追加できるようになっています。また、ドラッグ&ドロップでRelationshipsを指定できるようになったこと、GUI上での操作が高速化されています。

●Juju 2.0

16.04世代でのJujuは、Juju 2.0系へのメジャーバージョンアップが行われ、以下のような大きな変更が行われる予定です。前述のとおり、Jujuのリリースタイミングは Ubuntuとは完全には同期していないため、PPA経由での提供となる見込みです。なお、「Juju 2.0」と「Juju GUI 2.0」は別物で、後者はJuju 1.x系にも対応する新世代のWebインターフェースです。

・CLIコマンドの整理
・『ローカル』環境向けにLXDをサポート
・「ノード」の名称変更

◆CLIコマンドの整理

Jujuは猛烈なスピードで開発が続けられてきたため、無数のサブコマンドとそのオプションを把握しないと操作できない状態になりつつあります。Juju 2.0ではこれらが整理され、わかりやすい形に統合される予定です。

たとえば、Juju環境のバックアップを行うコマンドは「juju backups」サブコマンドに含められていたため、「juju backups create」「juju backups restore」でした。同様の、サブコマンドを用いたさまざまな操作が存在しています。

Juju 2.0ではこれらが「juju cretae-backup」「juju restore-backup」といった形式になります。https://jujucharms.com/docs/devel/command-changes に現時点でのコマンド変更が準備されています。

◆LXDのサポート

Jujuは、IaaS的な環境を利用しない『ローカル』環境(自マシンだけを使った環境)でも動作させることができます。1.x系ではLXCをサポートしており、自マシン上でコンテナとして仮想マシンを稼働させてIaaS環境の代替としていました。しかし、この環境は「ブートストラップ」にあたるmachine-0として、Jujuを実行する自マシンそのものが提供される必要がありました。このことは、『ローカル』環境にはUbuntuマシンが必須であることを意味します。

2.x系ではLXDを使ってネットワーク越しにコンテナが利用できるため、自マシン以外のUbuntu環境を使って環境を構成できます。

◆「ノード」の名称変更

Juju 1.0では、「ブートストラップノード」と「実環境」という2つの概念が扱われていました。「ブートストラップノード」はJujuそのものの各種設定を保持し、これをIaaSなどのホスト上にデプロイしたものが「実環境」です。これはほぼ1:1の関係で、「1つのブートストラップノードにつき、1つの実環境」という構成でした。たとえばEC2にデプロイしたブートストラップノードを使うと、そのノードに紐付いたEC2上の実環境にアクセスできる、というモデルです。

Juju 2.0ではこれらが「コントローラ」と「モデル」という概念に置き換わります。「コントローラ」にJujuそのものの設定を保持することは変わりませんが、1つのコントローラから複数の「モデル」を制御できるようになります。

サーバの新機能

●MAAS 2.0

MAAS(Metal As A Service)は、「物理マシンを、IaaS的なAPIやGUIで扱う」ためのソフトウェアです。Webインターフェースからの簡単な操作を行うだけで、「Wake On LanやIPMIで物理マシンの電源を投入し、PXEでOSインストーラをブートしてKickstartでOSをインストールするとともにリモートログインできるように設定する」という一連の操作を行うことができます。

16.04では、MAAS 2.0が搭載されます。Juju 2.0と同様、Ubuntu本体のリリースタイミングとは必ずしも同期しない形で開発が行われているため、リリース時点では開発版が搭載されている可能性があります。

MAAS 2.0の新機能は次のとおりです。

- 前提となる環境をPython 3.5 + Django 1.8へ変更
- MAASの管理ノードにあたる「クラスタコントローラ」を、冗長化サポートやネットワークの分割をサポートする「ラックコントローラ」で置き換え
- APIバージョンを2.0へ更新
- DNS管理機能を搭載
- Fan[注11]を含めた、複雑なネットワーク環境のサポート
- 物理マシンのストレージのパーティショニング指定と、bcacheサポートを追加

●OpenStack Mitaka

16.04では、OpenStack Mitakaが利用できます。Mitakaのリリース時期が16.04のリリース日と近過ぎるため、リリース後のアップデートでリリース版への差し替えが行われます。

◆OpenStack Autopilot

Ubuntu Serverに搭載された特筆すべき機能の1つが、OpenStack Autopilotです。OpenStack Autopilotは、MAASとJujuの組み合わせでOpenStackをデプロイし、Landscapeによる一括管理ができます。必要な操作は、前提となるいくつかのPPAを追加し、APTでパッケージをインストールしてMAASの初期設定を行い、物理マシンを登録し、インストール指定を行うだけです[注12]。

初期セットアップさえ完了すれば、電源投入やPXEによるOSインストールを含め、ほとんどの手順をAutopilotが処理します。ハードウェアさえあれば、OpenStackベースのプライベートクラウド環境を「クラウド的な操作だけで」構築できます。

OpenStack AutopilotはCanonicalによる商用サービスですが、物理ノード10台+仮想ノード10台までは無償で利用できます。

●pagemon

16.04で追加されたソフトウェアの中で、とくにサーバのトラブルシューティングに利用できるツールが「pagemon」です。pagemonはncursesベースの(つまりターミナルで動作する)メモリモニタで、メモリページの利用種別(アノニマスページ・Dirty・Swap)を俯瞰しつつ、その中身をバイナリエディタに近いインターフェースで確認できます。メモリが逼迫(ひっぱく)したサーバの動作状態の確認や簡易デバッグに利用できます。

pagemonパッケージとして提供されます。基本的な使い方は、「pagemon -p (プロセスID)」として起動するだけです。SD

注11) Fanは、「プライベートIPアドレスを分割し、一定の規則でルーティングを構成することで、利用可能なネットワーク空間を拡張する」ための実装です。詳細はgihyo.jpのUbuntu Weekly Topics2015年6月26日号(http://gihyo.jp/admin/clip/01/ubuntu-topics/201506/26)と、https://wiki.ubuntu.com/FanNetworkingを参照してください。

注12) http://www.ubuntu.com/download/cloud/install-openstack-with-autopilot参照。MAASを利用する場合、各ホストの起動設定を「PXEのみ」に設定するなど、若干の手作業による準備が必要です。通常は数時間程度で終了するでしょう。

セキュリティ対策はまずここから！

サーバのフリーで始めるセキュリティチェック

前編「Nmapによるポートスキャン」

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

脆弱性診断というものをご存じでしょうか。あまり聞きなれない言葉かと思いますが、脆弱性診断とは、システムやアプリケーションの脆弱性（セキュリティ上の問題点）を探し、どこにどのような脆弱性があるのかということと、その検出された脆弱性に対する対策を報告するサービスです。

脆弱性診断は、一般的に2つの方法を組み合わせて実施しています。セキュリティツールを用いたチェックと、セキュリティエンジニアによる手動での診断です。セキュリティツールは有償のものや自社開発したもの、オープンソースのものなど数多くあります。セキュリティベンダが提供する脆弱性診断では検出精度を上げるために、複数のセキュリティツールを組み合わせることが多いですが、ツールは完璧ではなく誤検知や検出漏れの可能性があるため、セキュリティエンジニアによる診断を行うことで脆弱性診断の網羅性を高めています。

ただ、セキュリティベンダに脆弱性診断を依頼する場合、当然費用がかかります。また、診断対象数が多いと、診断が終了するまでに多くの時間がかかり、当初の開発スケジュールに影響が出てしまう場合もあります。筆者は、予算やスケジュールなどの理由から脆弱性診断を実施せずにリリースを迎えてしまうケースを、これまでも多く見てきました。しかし、これは脆弱性を包含したままリリースしている可能性もあり、非常に危険です。そのため、セキュリティベンダに依頼しなくても簡易的なセキュリティチェックを自分たちで行い、少しでもリスクを軽減することの一助になればと思い、今回および次回でフリーツールを用いたセキュリティチェックの方法を紹介します。

次に挙げるように、脆弱性診断は複数の種類があります。

①プラットフォーム診断
②Webアプリケーション診断
③ペネトレーションテスト
④ソースコード診断

これらの脆弱性診断がどのようなものかを簡単に説明します。①はサーバやネットワーク機器のOSやミドルウェアに存在する脆弱性や設定上の不備を確認するためのものです。②はWebアプリケーションに存在する脆弱性や設定上の不備を確認するためのものです。③はシステム全体を対象に実際の攻撃手法を用いてどこまで侵入できるかという観点で確認するもので、①や②などを組み合わせたものです。しかし、これは1つの問題に対してどこまで侵入できるかを確認するため、診断範囲や項目の網羅性の点でいうと①や②よりも低くなります。④はプログラムソースから問題の有無を確認するもので、一般的に①②③はブラックボックス[注1]

注1）内部情報を把握せずに実施するテスト手法。

での実施ですが、④はホワイトボックス[注2]での実施となります。

今回と次回で取り上げるのは、①のプラットフォーム診断を対象とした内容です。プラットフォームでのセキュリティチェックでは、ポートスキャンによる稼働サービスの確認を行い、稼働確認ができたサービスに対して脆弱性スキャンを行います。今回は、ポートスキャンによる稼働サービスの確認について記載します。

セキュリティ上、外部に対して公開するサービスは必要最小限にすることが好ましいです。そのため、ポートスキャンを実施し、不要なサービスが稼働していないかを確認する必要があります。

どんな環境が必要?

今回および次回の解説では、2つの仮想環境を使用します。

- **Kali Linux**
  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。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。「https://www.kali.org/」からダウンロード可能。今回はKali Linux 64 bitを使用する
- **Metasploitable2**
  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。このディストリビューションは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、「https://sourceforge.net/projects/metasploitable/」からダウンロードできる

使用するソフトウェアはNmapとOpenVASです。OpenVASについては次回で説明します。

- **Nmap**
  ポートスキャンを行うためのツール。最新バージョンは7.12(2016年3月末時点)で、「https://nmap.org/」からダウンロードできる。Kali Linuxにはバージョン7.01がインストール済みであるため、ダウンロードは不要
- **OpenVAS**
  脆弱性スキャンを行うためのツール。最新バージョンはOpenVAS-8で「http://www.openvas.org/」からダウンロードできる

NmapはWindowsやCentOS、Ubuntuなどでも利用できます。NmapはCUIのソフトウェアですが、GUIのソフトウェアとしてZenmapが用意されており、ZenmapはGUIの操作だけでなくNmapと同じコマンドも利用できます(**図1**)。

Nmapは、CentOSやUbuntuにはデフォルトでインストールされていません。次のように、パッケージ管理ツールからインストールします。

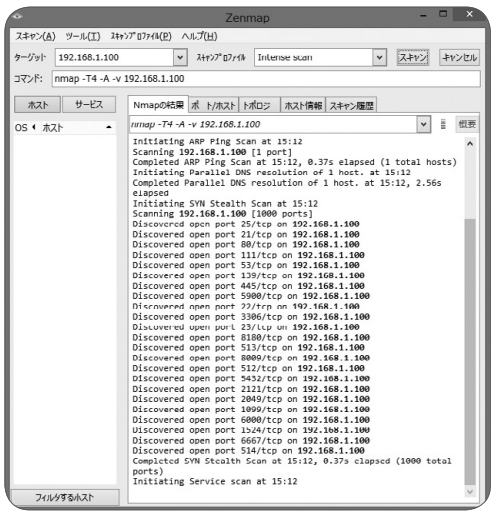
```
CentOSの場合
# yum -y install nmap
Ubuntuの場合
$ sudo apt-get install nmap
```

インストールされるNmapのバージョンは6.47[注3]ですが、本稿で使用する7.01と大きな差はありません。

注2) 内部情報を把握したうえで実施するテスト手法。

注3) CentOS 7、Ubuntu 15.10の場合。

▼図1 Zenmapでのスキャン



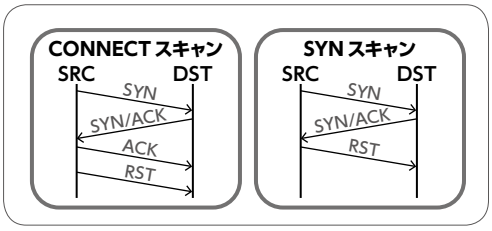
Let's ポートスキャン

ポートスキャンとは、TCPまたはUDPのパケットをスキャン対象のサーバやPCに送り、スキャン対象の挙動からポートの稼働状況を調査する行為です。

TCPとUDPではプロトコルが異なるため、ポートスキャンの方法も異なります。そのため、どちらのプロトコルでスキャンを行うのかは、スキャンタイプとしてコマンドラインオプションから指定します。また、TCPに対するポートスキャンでもいくつか方法があり、一般的によく使われるのは、CONNECTスキャンとSYNスキャンです(**図2**)。CONNECTスキャンは3 Way Handshakeを行うことで、ポートの稼働を確認します。一方、SYNスキャンはスキャン対象からSYN/ACKパケットが返ってくるかだけで、ポートの稼働を確認します。SYNスキャンはCONNECTスキャンに比べて、パケット量が少なく早く終わるため、SYNスキャンでのポートスキャンを推奨します。

一般ユーザでは権限上の理由からSYNスキャンができません。そのため、CONNECTスキャンのみとなります。管理者ユーザは両方とも利用できますが、明示的にオプションでCONNECTスキャンを指定しない限り、SYNスキャンが実行されます。

▼図2　CONNECTスキャンとSYNスキャン



nmapの書式

```
nmap［スキャンタイプ］［オプション］{スキャン対象}
```

スキャンタイプ……-sS、-sUなどで指定（省略可）
オプション……-sV、-Oなどで指定（省略可）
スキャン対象……IPアドレスなどで指定（必須）

オプションを何も指定せずに実施した結果(**図3**)と、Metasploitable2にログインしてnetstatを実行した結果(誌面での掲載は省略)を比べてみてください。いくつかのポートがNmapでは検出できていません。その理由は、オプションを指定しない場合にNmapは、一般的によく利用されている上位1,000ポートのみしかスキャンを行わないためです。すべて検出するためのオプションについては次節で紹介します。

ポートスキャンの結果はポートごとに「STATE」に出力され、open、closed、filtered、unfiltered、

▼図3　オプションを何も指定せずに、nmapを実施した結果

```
# nmap 192.168.1.100        ←Kali Linuxで、管理者権限で実行（以降の実行例も同様）
Starting Nmap 7.01 ( https://nmap.org ) at
2016-02-14 15:19 JST
Nmap scan report for 192.168.1.1
Host is up (0.00010s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
(..中略..)                          一部の結果は中略としているが、実際には23個のポートの情報が出力される
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown
MAC Address: 00:0C:29:A6:09:47 (VMware)

Nmap done: 1 IP address (1 host up)
scanned in 0.09 seconds
```

▼表1　ポートスキャンの結果

STATE	状態
open	該当ポートは稼働している
closed	ポートへアクセス可能だが、アプリケーションが稼働していない
filtered	ファイアウォールなどでフィルタされたため、該当ポートまでたどり着いていない
unfiltered	ポートへアクセス可能だが、ポートが開いているか閉じているか判別できない
open|filtered	ポートが開いているかフィルタされているか判別できない
closed|filtered	ポートが閉じているかフィルタされているか判別できない

open|filtered、closed|filteredのいずれかになります(**表1**)。

オプションを指定しないときにスキャンされる1,000ポートは、「nmap-servicesファイル」の「頻度」の高い順に実施されます。nmap-servicesには「サービス名」「ポート番号・プロトコル」「頻度」が記載されています(**リスト1**)。「頻度」は数値の大きいほうが優先度が高くなり、0から1未満の範囲で設定されています。

これを押さえれば大丈夫、Nmapのオプション

Nmapのオプションは70以上あり、すべてを紹介できませんが、有用なものをいくつか紹介します。

スキャン対象の指定

スキャン対象の指定方法はいくつかあります。スペース区切りで複数のスキャン対象を同時に指定する方法やCIDR(Classless Inter-Domain Routing)でセグメントを指定することも可能です。

```
# nmap 192.168.1.100 10.1.1.100   ←複数指定
# nmap 192.168.1.0/24   ←セグメントで指定
# nmap 192.168.1.100-110   ←範囲指定
```

▼リスト1　nmap-services(抜粋)

```
(..略..)
finger  79/tcp  0.006022
finger  79/udp  0.000956
http    80/sctp 0.000000        # World Wide Web HTTP
http    80/tcp  0.484143        # World Wide Web HTTP
http    80/udp  0.035767        # World Wide Web HTTP
(..略..)
```

▼図4　-snオプションでホストの探索を行った結果

```
# nmap -sn 192.168.1.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2016-02-15
20:30 JST
(..中略..)
Nmap scan report for 192.168.1.100
Host is up (0.00018s latency).
MAC Address: 00:0C:29:A6:09:47 (VMware)
(..略..)
```

ファイルにスキャン対象を記述し、-iLオプションでそのファイルを指定することで、一括でスキャンを行うことも可能です。

```
list.txt
192.168.1.100-254
19.1.1.0/24
```

```
# nmap -iL list.txt
```

--excludeオプションを合わせて指定することで、指定した範囲をスキャン対象から除外することも可能です。一時的に特定のサーバやネットワークをスキャン対象から除外する場合に、有用です。

```
# nmap -iL list.txt --exclude
192.168.1.200-254
```

スキャン対象の探索

-snオプションは、スキャン対象となるホストの探索に活用することが可能です(**図4**)。管理が行き届いていないセグメントに、どのくらいネットワークに接続された機器があるかを確認するときなどに有用です。特定セグメントに対してpingやTCP/80、TCP/443などによる探索を行います。TCP/80、TCP/443については、サービスが稼働していなくても、Resetパケットの有無で存在の確認を行っています。以前のバージョンでは-sPオプション(Ping Scan)でしたが、現在のバージョンでは両方のオプションが使用できます。

Nmapは通常、スキャン対象が稼働しているかを確認し、稼働している場合にポートスキャンなどを行いますが、-Pnオプションを指定すると、稼働状況の確認を行わずにポートスキャンなどを実施します。そのため、ホストが存在しない場合でもポートスキャンを

試みます。

```
# nmap -Pn 192.168.1.0/24
```

-nオプションは、スキャン対象であるIPアドレスのDNSによる逆引きを行わないためのものです。DNSの逆引きによる時間を短縮する効果が期待できます。

```
# nmap -n 192.168.1.0/24
```

スキャンプロトコルの指定

-sTオプションはTCPのCONNECTスキャンです。

```
# nmap -sT 192.168.1.100
```

-sSオプションはTCPのSYNスキャンです。

```
# nmap -sS 192.168.1.100
```

-sUはUDPに対するスキャンオプションです。

```
# nmap -sU 192.168.1.100
```

スキャン対象ポートの指定

-pオプションはスキャンするポートを指定するもので、指定方法は複数あります。ある範囲をハイフンで指定するか、ポートをカンマ区切りで複数指定することが可能です。

```
# nmap -p 1-1023 192.168.1.100   ←範囲指定
# nmap -p 21,22,23,25,53,80,443 ↵
192.168.1.100   ←複数指定
```

フルポートに対してスキャンを実施する場合、1-65535、または-p-オプションを指定する必要があります。

```
# nmap -p 1-65535 192.168.1.100
```

では、もう一度Metasploitable2に対して、フルポート指定でポートスキャンを実施してみてください。**図3**の結果よりも8ポート多く検出されています(**図5**)。先ほどは1,000ポートのみだったのに対して、今回はフルポート指定でポートスキャンを行ったために、より多く検出されるようになりました。一部ループバックアドレスやIPv6でLISTENされているサービスがありますが、netstatの実行結果と一致します。

T:1-65535,U:1-1023というように、TCPに対するポート範囲はT:を、UDPに対するポート範囲はU:を指定することで、TCP、UDPを同じタイミングでポートスキャンすることができます。ただし、-sSや-sTなどでTCPへのスキャンを明示的に指定されていない場合、TCPに対するポートスキャンは行われません。同様に、-sUで明示的に指定されていない場合は、UDPに対するポートスキャンは行われません。

```
# nmap -sS -sU -p T:1-65535,U:1-1023 ↵
192.168.1.100
```

ポートスキャンするポートを指定しない場合、頻度の高い1,000ポートが対象になりますが、--top-portsオプションで対象のポート数を変更できます。頻度の高い2,000ポートの場合は次のようになります。

```
# nmap --top-ports 2000 192.168.1.100
```

また、-Fは高速オプションですが、これは頻度の高い100ポートを実施します。

▼図5　フルポート指定で、nmapを実施した結果

```
# nmap -p- 192.168.1.100

Starting Nmap 7.01 ( https://nmap.org ) ↵
at 2016-02-15 22:28 JST
Nmap scan report for 192.168.1.100
Host is up (0.000091s latency).
Not shown: 65504 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
(..中略..)
51671/tcp open  unknown
53075/tcp open  unknown
MAC Address: 00:0C:29:A6:09:47 (VMware)
(..略..)
```

一部の結果は中略としているが、実際には31個のポートの情報が出力される

バージョンスキャン、OSの検出

Nmapには、稼働しているポートのより詳細な情報を収集するバージョンスキャンやOSの検出を行う機能があります。「nmap-service-probesファイル」には、サービスごとの特徴が保存されています。バージョンスキャンは、その特徴をもとに稼働しているサービスと照合させることで、サービスを特定することができます。そのため、一般的なポート番号から変更しているサービスも検出できます。たとえば、通常FTPはTCP/21番で稼働していますが、TCP/2121番に変更している場合も検出できます。

-sVオプションでバージョンスキャンが行われます(**図6**)。バージョンスキャンはポートスキャンを実施し、稼働しているポートに対してバージョン情報の収集を行います。そのため、ポートスキャンのみを行うよりも時間はかかってしまいます。TCP/9100～9107のポートがバージョンスキャンの対象に含まれている場合、ポートスキャンは実施されますが、バージョンスキャンは該当ポートに対しては実施されません。TCP/9100～9107のポートはプリンタで利用されており、バージョンスキャンにより大量の印刷が行われる可能性があるため、バージョンスキャンから除外されています。--allportsオプションを指定すれば、強制的にTCP/9100～9107のポートもバージョンスキャンが行われるようになりますが、よほどの必要性がない限りは実施しないことを推奨します。

OSの検出は、-Oオプションを指定します(**図7**)。ICMP、TCP、UDPの稼働ポートおよび稼働していないポートへの通信を試みて、挙動などからOSの種類を検出します。OSの検出は挙動からの推測となるため、特定されないことが多々あります。

-Aオプションは、バージョンスキャン(-sV)やOSの検出(-O)や後述するNmap Script Engineの一部(-sC)、traceroute(--traceroute)を組み合わせたものです。

▼図6　バージョンスキャンの結果

```
# nmap -sV 192.168.1.100
(..中略..)
Not shown: 977 closed ports
PORT     STATE    SERVICE     VERSION
21/tcp   open     ftp         vsftpd 2.3.4
22/tcp   open     ssh         OpenSSH 4.7p1 ➥
Debian 8ubuntu1 (protocol 2.0)
(..中略..)
2121/tcp open     ftp         ProFTPD 1.3.1
3306/tcp open     mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open     postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open     vnc         VNC (protocol 3.3)
6000/tcp open     X11         (access denied)
6667/tcp open     irc         Unreal ircd
8009/tcp filtered ajp13
8180/tcp open     unknown
MAC Address: 00:0C:29:A6:09:47 (VMware)
(..略..)
```

一部の結果は中略としているが、実際には23個のポートの情報が出力される

ポートスキャンのパフォーマンス

Nmapには、ポートスキャンのスピードなど、パフォーマンスをカスタマイズするためのオプションが用意されています。

--min_rtt_timeoutオプション、--max-rtt-timeoutオプション、--initial-rtt-timeoutオプションは、応答時間のタイムアウトをミリ秒単位で設定するものです(これらを明示的に指定しない場合、タイムアウトになる時間はそれまでのレスポンス時間をもとに算出されます)。ただ、ネットワーク環境によりますが、基本的にこれらのオプションを個別に設定する必要性は低いです。

--min-parallelismオプション、--max-parallelismオプションは、

▼図7　OSの検出の結果

```
# nmap -O 192.168.1.100
(..中略..)
MAC Address: 00:0C:29:A6:09:47 ➥
(VMware)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_ ➥
kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop
(..略..)
```

ポートスキャンやスキャン対象の探索を行うときの並列処理数を指定します。--max-parallelismを1とすると、並列処理はされなくなります。何も指定がない場合、ネットワーク状況に応じて変動します。

--max-retriesオプションは、パケットがフィルタされている場合やネットワーク上で喪失した場合に、再送する回数を指定します。デフォルトは10回で、1を指定した場合、再送されません。

--scan-delayオプションは、パケットの送信を指定された時間分遅延させます。ただし、ここで指定された遅延時間は、ポートスキャンのみに適用されます。バグのため現在のバージョンではバージョンスキャンには適用されません。--max-scan-delayオプションは、最大遅延時間を指定するもので小さい値にすることでスキャンにかかる時間は短縮されますが、その分、スキャン対象への負担は大きくなります。

```
# nmap --scan-delay 500ms 192.168.1.100
```

この場合、ポートスキャンが1,000ポートに対して行われるため、スキャンを行う側（Kali Linux）から送信されるSYNパケットが1,000個[注4]で、1パケットを送信するごとに0.5秒遅延を発生させるため、スキャンには理論上500秒かかります。実際に10回の統計を平均すると502.725秒となっており、若干の誤差はありますが理論上の時間に近い結果となっています。

このように--max-rtt-timeoutや--max-scan-delayなどで個別に値を指定し、パフォーマンスを最適化することは可能ですが、ネットワーク環境や経験に基づくため、うまく制御するのは非常に困難です。そのため、Nmapではパフォーマンス設定のテンプレートが用意されています（**表2**）。テンプレートは-T0から-T5の6段階になっており、数字が増えるほどスキャンにかかる時間は短くなりますが、その分負担も大きくなります。Nmapのデフォルト（何も指定しない場合）は-T3で、Zenmapは-T4が指定されています。一般的に-T3（デフォルトの設定値）が好ましく、-T5でスキャンをかけることはスキャン対象に影響が出てしまう可能性が高まるため、避けるべきです。また、-T0や-T1はスキャンに非常に多くの時間がかかるため、特殊な事情がない限りはあまり向かないオプションです。

スキャン結果出力

スキャン結果は標準出力に出力されますが、オプションにより特定の形式で出力することができます。

-oNオプションは指定したファイルにスキャン結果を出力します。出力結果の最初の行に実

▼リスト2　-oNで出力したファイルの例（output.txt）

```
# Nmap 7.01 scan initiated Thu Mar 3 08:48:10 ↵
2016 as: nmap -oN output.txt 192.168.1.100
Nmap scan report for 192.168.1.100
Host is up (0.00028s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
(..略..)
```

注4）パケット自体は、SYNパケットが1,000個とRSTパケットが23個（Metasploitable2で稼働しているポートが23あるため）の計1,023個送信されます。

▼表2　パフォーマンス設定のテンプレート

オプション	オプションのおもな内容
-T0	平行処理数は1で、遅延時間は5分（5分に1回送信する）、再送回数は10回
-T1	平行処理数は1で、遅延時間は15秒、再送回数は10回
-T2	平行処理数は1で、遅延時間は0.4秒、再送回数は10回
-T3	平行処理数および遅延時間はネットワーク状況に応じて変動、再送回数は10回
-T4	平行処理数はネットワーク状況により変動、応答時間の最大タイムアウトは1250ミリ秒、遅延時間は10ミリ秒、再送回数は6回
-T5	平行処理数はネットワーク状況により変動、応答時間の最大タイムアウトは300ミリ秒、遅延時間は5ミリ秒、再送回数は2回

行したコマンドが出力されるので、どのコマンド結果なのかをあとで確認できます。

```
# nmap -oN output.txt 192.168.1.100
（output.txtの内容はリスト2のとおり）
```

-oXオプションは出力形式がXMLで出力されます。

```
# nmap -oX output.xml 192.168.1.100
```

-oNや-oXに、--append-outputオプションを付与することで、出力ファイルへの追記を行えます。

```
# nmap -oN output.txt -oX output.xml ➥
--append-output 192.168.1.100
```

Nmap Script Engine

Nmapには、Nmap Script Engine(以降、NSE)というLua言語ベースのスクリプトを実行するための機能があります。この機能によりポートスキャンだけではなく、一部の脆弱性スキャンも実施できます。実行されるスクリプトは、拡張子がnseであるNSEファイルが用意されています。Kali Linuxの場合、/usr/share/nmap/scripts/以下にあり、2016年3月5日時点で515ファイル存在します。NSEファイルは**表3**のカテゴリに分類され、各NSEファイルがどのカテゴリに属するのかは、/usr/share/nmap/scripts/script.dbに記載されています。1つのNSEファイルが属するカテゴリは1つだけではなく、複数のカテゴリに属している場合もあります。

--scriptオプションにカテゴリ名を指定することで、カテゴリに属するNSEファイルが実行されます。カンマを入れることで、複数のカテゴリを指定できます。また、-sCオプションは、--scriptでdefaultを指定するのと同じ意味になります。

```
# nmap --script auth,default 192.168.1.100
```

特定のNSEファイルを指定することも可能です。たとえば、OpenSSLの脆弱性であるHeartBleed(CVE-2014-0160)をチェックするssl-heartbleed.nseを用いてスキャンを実施します。脆弱性が存在している場合、ポート情報の下に詳細情報が表示されます(**図8**)。HeartBleedが存在しない環境の場合は、脆弱性スキャンの結果は表示されません。

--script-helpは、NSEファイルの概要を確認するためのオプションです。NSEファイル名やカテゴリを指定することで、1つまたは複数のNSEファイルの概要を確認できます(**図9**)。

▼表3　NSEファイルのカテゴリ

カテゴリ	説明
auth	認証に関連するスクリプト
broadcast	ローカルネットワークで、ブロードキャストによるホスト探索をするスクリプト
brute	認証情報の推測のためのブルートフォースを行うスクリプト
default	明確な基準はないが、さまざまな観点からデフォルトで実行されるべきスクリプト
discovery	SNMPやディレクトリサービスのように、稼働しているサービスから情報を収集するスクリプト
dos	DoSを引き起こす可能性のあるスクリプト
exploit	脆弱性を試すスクリプト
external	whoisなどのデータベースやネットワークリソースから情報を取得するスクリプト
fuzzer	予期しないパケットまたはランダムなパケットを送るためのスクリプト
intrusive	ターゲットに影響が出てしまうリスクの高いスクリプト
malware	マルウェアやバックドアに感染した環境かどうかをテストするスクリプト
safe	サービスに影響を与えたり、ネットワーク帯域を大量に使用したり、脆弱性を悪用したりするようなことがないスクリプト
version	-sVのバージョンスキャンを選択したときに実行されるスクリプトで、バージョンの検出を行う
vuln	既知の脆弱性をチェックし、脆弱性があったら結果を報告するスクリプト

--script-updatedbは、NSEファイルのアップデートを行うためのオプションです(**図10**)。

運用について

いろいろなオプションを紹介しましたが、具体的にどのように使用すれば、Nmapをうまく活用できるのでしょうか。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、スキャン対象をどこまでにするのかを決めます。DMZ(DeMilitarized Zone)にあるサーバのみを対象とするのか、DMZ以外も含

▼図8　HeartBleedの脆弱性スキャンの結果(脆弱性があった場合)

```
# nmap -p 443 --script ssl-heartbleed.nse 192.168.1.215

Starting Nmap 7.01 ( https://nmap.org ) at 2016-03-03 02:51 JST
Nmap scan report for 192.168.1.215
Host is up (0.00019s latency).
PORT    STATE SERVICE
443/tcp open  https
| ssl-heartbleed:
|   VULNERABLE:
|   The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic
software library. (..中略..)
|     State: VULNERABLE
|     Risk factor: High
|       OpenSSL versions 1.0.1 and 1.0.2-beta releases (including 1.0.1f and
1.0.2-beta1) of OpenSSL are affected by the Heartbleed bug. (..中略..)
|
|     References:
|       http://www.openssl.org/news/secadv_20140407.txt
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160
|_      http://cvedetails.com/cve/2014-0160/
MAC Address: 00:0C:29:73:19:B8 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.23 seconds
```

脆弱性スキャンの結果(脆弱性の詳細情報)

▼図9　NSEファイルの概要確認の結果

```
# nmap --script-help ssl-heartbleed.nse

Starting Nmap 7.01 ( https://nmap.org ) at 2016-03-05 19:27 JST

ssl-heartbleed
Categories: vuln safe
https://nmap.org/nsedoc/scripts/ssl-heartbleed.html
  Detects whether a server is vulnerable to the OpenSSL Heartbleed bug (CVE-2014-0160).
  The code is based on the Python script ssltest.py authored by Jared Stafford (jspenguin@
jspenguin.org)
```

▼図10　NSEファイルのアップデートを実施

```
# nmap --script-updatedb

Starting Nmap 7.01 ( https://nmap.org ) at 2016-03-05 16:28 JST
NSE: Updating rule database.
NSE: Script Database updated successfully.
Nmap done: 0 IP addresses (0 hosts up) scanned in 1.04 seconds
```

めて対象にするのか、優先度に応じて実施範囲を決めます。すべてのサーバ、ネットワーク機器などを極力網羅的に実施することを推奨します。対象数が多い場合、複数のグループに分けて実施するなどして1回のスキャンにかかる負担を減らすほうがいいでしょう。TCPについてはフルポートで実施し、UDPについてはよく使われる1,000ポートを対象として行うのが、効果的です。

```
# nmap -sS -p- -sV -oX output_tcp.xml IPアドレス
# nmap -sU -sV -oX output_udp.xml IPアドレス
```

次にスキャンの実施タイミングですが、定期的な実施を推奨します。たとえば、DMZや重要なサーバは4半期に1回とし、それ以外は年に1回実施するなど、優先度に応じて実施回数を変更するのがいいでしょう。現在稼働している環境だけではなく、これから構築する環境についてもカットオーバー前にスキャンを実施し、事前に問題点に対応することを推奨します。また、スケジュールを立てる際には、スキャンやそれに対する改修も前もって計画しておくことが重要となります。

Nmapの結果は、ndiffを利用することで効率良く分析が行えます。ndiffは、Nmapの結果(XML形式)の差分を出力してくれます(**図11**)。そのため、前回の実施結果からどのような変化があったのかを簡単に把握できます。

3点目のスキャン結果の対応方針を決定するには、「どのサービスが稼働していることが正常なのか」を事前に把握しておく必要があります。そのうえで、不要なサービスは停止します。稼働しているサービスでも、不特定多数の人への公開が不要なものであれば、アクセス制限をかけることが好ましいです。また、アプリケーションによってはバージョン情報などを出力するものがあり、外部に対して公開する情報は必要最小限にしたほうがよりセキュアになるため、アプリケーションで出力されているバージョン情報などは可能な限り秘匿することを推奨します。ただし、バージョン情報を消したからといって、攻撃を受けないというわけではないので、脆弱性がある場合は適切に対応する必要があります。

今回は、ポートスキャンによるサービスの稼働状況を把握する方法について紹介しました。次回は稼働するサービスへの脆弱性スキャンについて紹介します。SD

参考サイト

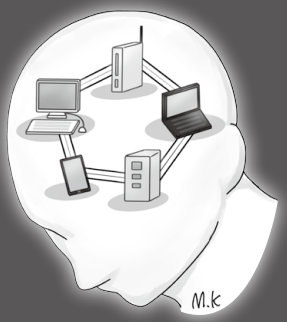
- Nmap　https://nmap.org/
- Nmap リファレンスガイド(日本語)
  https://nmap.org/man/jp/
- Kali Linux　https://www.kali.org/

▼図11　ndiffによるスキャン結果の差分出力

```
↓ポートスキャンを実施
# nmap -oX first.xml -sS -p- -sV 192.168.1.100
# nmap -oX second.xml -sS -p- -sV 192.168.1.100

↓スキャン結果の差分を出力
# ndiff first.xml second.xml
-Nmap 7.01 scan initiated Sat Mar 05 22:26:07 2016 as: nmap -oX first.xml -sV 192.168.1.100
+Nmap 7.01 scan initiated Sat Mar 05 22:36:53 2016 as: nmap -oX second.xml -sV 192.168.1.100

 192.168.1.100, 00:0C:29:A6:09:47:
-Not shown: 979 closed ports
+Not shown: 977 closed ports
 PORT     STATE    SERVICE VERSION
-25/tcp   open     smtp    Postfix smtpd
+25/tcp   filtered smtp
+80/tcp   open     http    Apache httpd 2.2.8 ((Ubuntu) DAV/2)
+3306/tcp open     mysql   MySQL 5.0.51a-3ubuntu5
```



仮想化の知識、再点検しませんか?
使って考える仮想化技術

本連載は「仮想化を使う中で問題の解決を行いつつ、残される課題を整理する」ことをテーマに、小さな仮想化環境の構築・運用からはじめてそのしくみを学び、現実的なネットワーク環境への実践、そして問題点・課題を考えます。仮想化環境を扱うエンジニアに必要な知識を身につけてください。

Author
笠野 英松(かさの ひでまつ)
有限会社 ネットワーク・メンター
URL http://www.network-mentor.com/indexj.html

第1回 仮想化の現状を見てみよう

「仮想化」とは?

定義

一般的な「仮想」とは現実ではないものを想像して作ることですが、ICT(情報通信技術)においては物理的なリソースを論理的に組み合わせて、あたかも1つの物理的なものに見せかけるしくみのことです。もう少し具体的に言うと、複数の物理的なシステムやディスクなどを、あるいは、1つの物理的なシステムやディスク、ネットワークの中のリソース/構成要素を、論理的に組み合わせて、1つのシステムやディスク、あるいは1つのネットワークとして実現し、利用するしくみです。

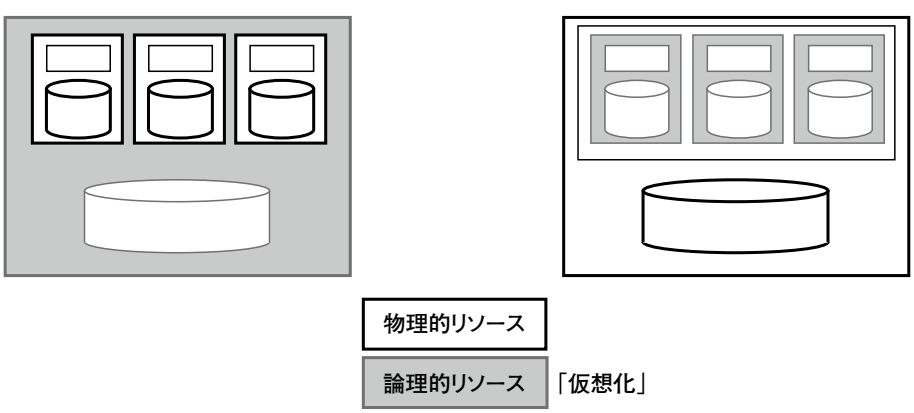
前者のように物理的なもの「それ自体」を組み合わせる形態(**図1**①)と、後者のように物理的なもの「の中のリソース」を組み合わせる形態(**図1**②)と、仮想化の実現形態には2通りあります。

歴史

1964年のIBMのSystem/360上で稼働したCP-67/CMS[注1]が仮想化の始まりです。以降、IBMは仮想化OSのVM/CMS[注2]として、最近ではz/VM[注3]となっています。

注1) Control Program-67/Cambridge Monitor System → Conversational Monitor System(CP-67/CMS)、仮想化OSはVM/CMS。
注2) VM/CMS:Virtual Machine/Conversational Monitor System、仮想機械/対話モニタシステム
注3) z/VM:zアーキテクチャ/Virtual Machine(仮想機械)

▼図1 仮想化のイメージ



現在、利用されている仮想化の初めは、1998年のVMware[注4] Workstationです（VMwareはその後サーバ製品リリースへ続く）。

2008年にはマイクロソフトのWindows Server 2008に搭載されたHyper-V[注5]が、また、Xen[注6]の開発は、2002年からケンブリッジ大学の研究プロジェクトXenoServer Projectとして始まり、最初のリリースは2003年でした。

Linuxカーネル2.6.20以降に標準添付されるようになったオープンソースのKVM[注7]は少し遅れて2006年ですが、2011年5月17日に、IBMやIntel、HP、Red Hatを中心としてオープンコンソーシアムOVA[注8]が立ち上がり、KVMの利用が広がりました。

一方、Xen Projectも2013年4月15日にAmazonやGoogle、AMD、Cisco、Citrix、Oracleなどが参加するLinux Foundation協業プロジェクトとなりました。XenはRHEL[注9]5まではOS同梱でしたが、RHEL6からは非同梱になり、一方、KVMはLinux OSに同梱となっています。

エミュレータ

仮想化と同じように、あるプラットホーム上で別システムを稼働させるしくみとして「エミュレータ」があります。エミュレータは、あるシステム（ハードウェアやソフトウェア）の機構や機能、動作などを別のシステム上で可能にするしくみです。

注4）http://www.vmware.com/jp
注5）https://www.microsoft.com/ja-jp/server-cloud/local/hyper-v-server/default.aspx
注6）http://www.xenproject.org/
注7）KVM：Kernel-based Virtual Machine
http://www.linux-kvm.org/page/Main_Page
注8）OVA：Open Virtualization Alliance
注9）RHEL：Red Hat Enterprise Linux

「仮想化」はどんなしくみで提供されるのだろうか？

完全仮想化と準仮想化

仮想化の方式には、完全仮想化（FV[注10]）と準仮想化（PV[注11]、Xenがサポート）の2つの方式があります。

完全仮想化方式では、仮想マシン上ですべてのゲストOSを通常の実ハードウェア上と同様に動作させることができますが、ハードウェア（CPU）の仮想化技術VT[注12]機構を使用しなければなりません。

一方、準仮想化技術方式では、CPUのVT機構がない場合にも可能な方式ですが、仮想マシン上で動作するためのコード修正済みのOSでなければゲストとして動作しません。Xenの準仮想化では、ゲストOSはLinuxなど一部のOSに限定されます。

なお、準仮想化方式では完全仮想化方式に比べて、ハードウェアエミュレーションを行わないためCPUの消費量が少なく、性能低下を抑えることができますが、先述のようにゲストOSの変更が必要になります。

VT機構を装備したプロセッサ

VT装備のプロセッサは、執筆現在「Intel-VT」と「AMD-V」です。これらのプロセッサは2005、6年ころ以降に出荷されたPCに搭載されています。Core 2プロセッサなどが代表的です。また、こうしたVT機構を利用する際には、BIOSでもこの仮想化機能を有効にしなければなりません。

Linux上のコマンドでCPUの仮想化機能が

注10）fullvirtualized
注11）paravirtualized
注12）Virtualization Technology（仮想化技術）

有効か無効かを判別することができます[注13]。

PAE

PAE[注14]とはX86-32ビットプロセッサ、またはホストOSが32ビットで、4GB以上(64GB)の物理メモリサイズを利用可能にするOSの拡張機能です。IntelではPentium Pro(1995年11月)以降に搭載されています。Xenの準仮想化を利用する場合には、このPAEが装備されていなければなりません。

このPAEもLinux上のコマンドで確認できます[注15]。

「仮想化」にはどのようなものがあるのだろうか?

仮想化のさまざまな形態

表1は仮想化の全体像です。仮想化は、プラットホームのタイプとその上での実現形態および技術形態で全体像が見えてきます。

仮想化の実現形態には物理的な形態と論理的な形態がありますが、最近の仮想化とはこのうち、論理的な形態でネットワーク仮想化やストレージ仮想化、システム仮想化(「システムでの仮想化[注16]」)を指します。

物理的な実現形態には利用するプラットホームに応じて、ハードウェアラックになったコンピュータボード複数で論理的にサーバ化したり、ストレージを複数で論理的にストレージ化したり、システムやストレージおよびネットワークを統合して複数論理的にクラスタ化したりといった方法があります。最初のハードウェアラックのボードがBlade Serverと呼ばれるもので、2つめがSAN[注17]やRAID[注18]、NAS[注19]です。また3番めがクラスタリングと呼ばれるサービス形態で、製品としてはLVS[注20]などがあり、物理的な多重化やNICを束ねるチャネルボンディング、IPルーティング(経路)制御、ストレージ・データ同期・共有、さらにはデータサービスなどの技術を統合して提供します。

論理的な実現形態については、**表1**でさらにそれに対応した技術形態、製品例、そして備考をまとめています(**表1**中、「論理的」欄より右の欄は論理的実現形態にのみ対する項目です)。

システム仮想化には、いわゆる「サーバ仮想化」(**表1**中、網がけ太枠内)と、単一システムOS上でアプリケーションサーバやドメインサーバを複数論理的に実現する、言わば「単体仮想システム」とがあります。

ストレージ仮想化はI/Oの物理的リソースを論理的にI/Oの単位に「再」構成する仮想化のしくみで、そのI/O単位としてボリューム(ディスク)やブロック、ファイルシステム(やファイル)での仮想化があります。代表的な製品も**表1**に挙げました。

ネットワーク仮想化は、VPN[注21]やVLAN[注22]を使用する仮想化のしくみで、製品の総称としては(VPNやVLANのほかに)SDN[注23]やOpen Flow[注24]などがあります。

多少変わったところでは、Linux Network NameSpaceというものもあります。Linuxシステム上で、複数の仮想インターフェースにIPアドレス空間を割り当てて、複数の仮想的なネットワーク空間を構築することができます。SDN/OpenFlowは物理的なネットワーク(やデバイス)の設定や管理制御などをソフトウェア

注13) コマンド= egrep -e 'vmx|svm' /proc/cpuinfo
(vmx:Intel-VT、svm:AMD-V)このコマンドで表示があれば、そのVT機構が有効になっています。何も表示されなければハードウェアのVT機構が使用できないので、Xenでは準仮想化方式しか使用できません(ゲストOSが限られてくる)。

注14) Physical Address Extension

注15) コマンド= grep pae /proc/cpuinfo
表示 = flags : fpu tsc msr PAE cx8 apic mtrr cmov pat clflush acpi mmx fxsr sse sse2 ss ht up cid

注16) ここでは便宜的に記述していて、正式な用語ではない。

注17) Storage Area Network(ストレージ・エリア・ネットワーク)

注18) Redundant Arrays of Independent/Inexpensive Disks

注19) Network Attached Storage(ネットワーク接続ストレージ)

注20) Linux Virtual Server(Linux仮想サーバ)

注21) Virtual Private Network(仮想プライベート・ネットワーク)

注22) Virtual LAN(仮想LAN)

注23) Software Defined Network(ソフトウェア定義ネットワーク)

注24) OpenFlowによるアプリケーションのネットワーク管理

で論理的に一元化処理するしくみで、ネットワーク全体の仮想化ともいえます。

さて、今回の連載のテーマは以上のような仮想化の中の「サーバ仮想化」で、仮想化基盤のVMM(仮想マシンモニタ[注25])を物理デバイス上に配置する「ハイパーバイザ型[注26]」とOS上に配置する「ホスト型」とがあります(**図2**)。

従来の「サーバ仮想化」は**表1**の網がけ太枠内の複数仮想化システムの欄の「ハイパーバイザ型」のことでした。ハイパーバイザ型は、物理システム(ホスト)上でその物理リソースを使用する複数のVM(仮想マシン[注27])を稼働させるので、そのために必要な十分なCPU能力、メモリ容量、そしてディスク容量など使用リソースを確保しておかなければなりません。

代表的な製品例に、マイクロソフトのHyper-V、VMware社のVMware ESXi[注28]、XenプロジェクトのXen、linux-kvm.orgのKVMがあります。CitrixのXenServerやOracle VMはXenベースの仮想化製品です。サーバ仮想化製品については次項で説明します。

「ハイパーバイザ型」の欄の下の「ホスト型」はクライアントPCのOS上で仮想化を実現するもので、VMware PlayerやVirtualBox、Windows 7のVirtual PC、Windows 8以降のClient Hyper-Vが代表的な製品です。

なお、「デスクトップ仮想化」とは仮想マシン上でデスクトップOSを構築し、iPadなどの携帯端末BYOD[注29]から利用する形態で、仮想マシン上にサーバOSを稼動させる利用形態を、とくに「サーバ仮想化」という場合もあります。

最新では、ハイパーバイザ上のゲスト/仮想マシン内にハイパーバイザを配置し、さらにそのゲスト/仮想マシン内にハイパーバイザを配置し、……という、ゲストをも仮想化する「ネスト化(Nested、入れ子の)ハイパーバイザ型」が出てきています。

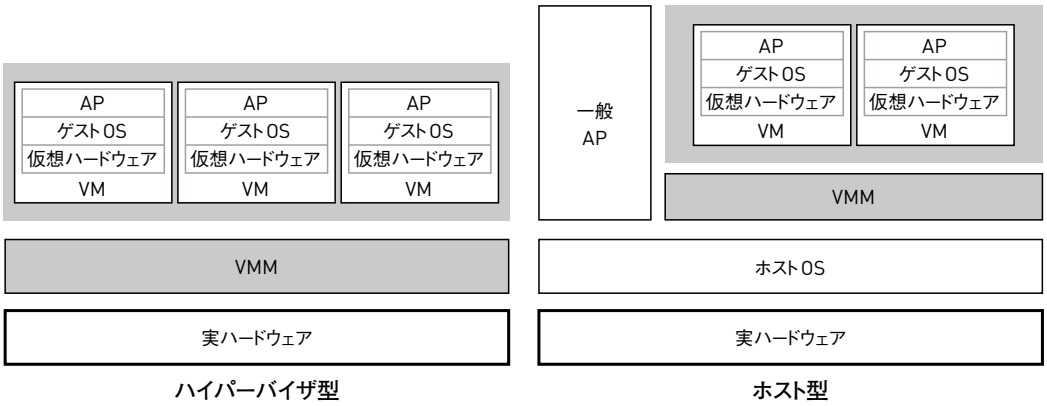
「ネスト化ハイパーバイザ型仮想化」を最初に実装したのはLinuxカーネル3.2のKVMです。

注25) VMM:Virtual Machine Monitor(仮想マシンモニタ)
注26) 「ベアメタル(ハードウェアに直結した、という意味)」ハイパーバイザ型ということもある。
注27) VM:Virtual Machine(仮想マシン)
注28) http://www.vmware.com/jp/support/vsphere-hypervisor.html
注29) Bring Your Own Device、仮想マシンの利用端末として自分の携帯端末を利用すること。

▼表1 さまざまな仮想化

プラットホーム	仮想化の実現形態		技術形態		製品例	備考
	物理的	論理的				
システム	Blade Server	システム仮想化(サーバ仮想化)	単体仮想システム	仮想ドメイン・アプリケーションサーバ	Apache仮想ホスト、Sendmail/popdom、Java VM	
				仮想ドメイン・システム	chrootバーチャルサーバ、Virtual Services	
			複数仮想システム	ハイパーバイザ型(サーバ仮想化)	Hyper-V、VMware ESXi、XEN(Citrix XenServer、Oracle VM)、KVM	利用形態(「サーバ仮想化」、「デスクトップ仮想化」-BYOD)
				ホスト型(ワークステーション仮想化)	VMware Player、VirtualBox、Virtual PC(Win 7)、Client Hyper-V	
			多重仮想システム	ネスト化ハイパーバイザ型(ゲスト仮想化)	KVM、Vmware ESXi/Player、Hyper-V、XEN	「ネスト化仮想化」
ストレージ	RAID, SAN, NAS	ストレージ仮想化	ボリューム、ファイルシステム、ブロック		HPE StoreVirtual VSA、Scale-Out File Server(SOFS)、EMC ScaleIO Node、Vmware Virtual SAN(VSAN)	
ネットワーク	クラスタリング	ネットワーク仮想化	VPN、VLAN		VPN、VLAN、Linux Network Namespace SDN(Northband/Southband API)、OpenFlow	ゲートウェイ中核

▼図2　サーバ仮想化の形態



AP：アプリケーションプログラム
VM：Virtual Machine（仮想マシン）
VMM：Virtual Machine Monitor（仮想マシンモニタ）

現在ではVMware ESXi（バージョン5.0以降）や、Hyper-V Server（2016以降）、そしてXenでも利用可能になっています。

ネスト化ハイパーバイザ型仮想化では仮想化環境内で、仮想化環境のテストや仮想化環境全体のバックアップやコピーを含めた運用管理、セキュリティ管理などを高速かつ効率的に処理することが可能になります。

ただし、もちろん、ベースとなる物理システムではネスト化ハイパーバイザ型仮想化の運用に耐えうるに十分な性能や容量などがなければなりません。

サーバ仮想化製品の概要

主なサーバ仮想化製品の、Xen、KVM、VMware ESXi、Hyper-Vについては次号以降であらためて紹介する予定ですが、簡潔にまとめておきます。

XenとKVMはLinuxベースにサポートされる仮想化プラットホームで、ほぼ同じCUI（コマンド・操作）とGUI（仮想マシンマネージャ）で運用管理を行います。リモートからはVNCなどを使います。

VMware ESXiとHyper-Vサーバはリモートクライアントからの運用管理が主となります。

KVMについてはこの連載の第2回以降の実践の中で詳しく説明していきます。

仮想化環境の運用管理

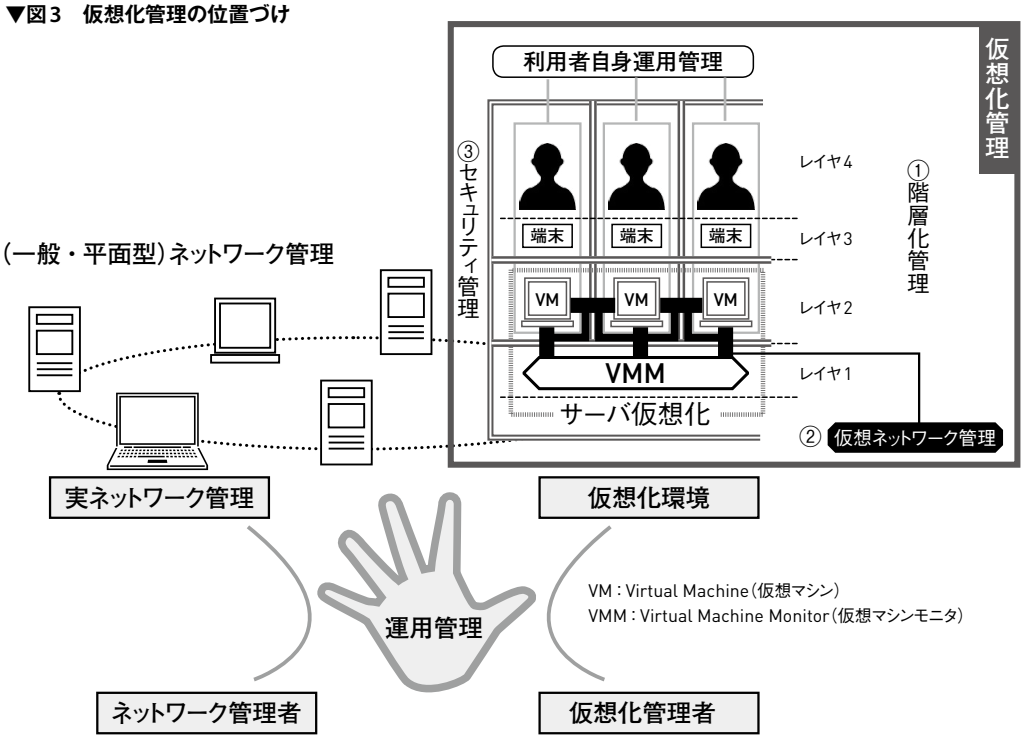
一般的な仮想化管理は「仮想化インフラと仮想マシンの、個々およびそれぞれの間の、性能や障害、リソースなどの運用管理」になりますが、実際には一般の運用管理やネットワーク管理に加えて、仮想化システム内での①階層化管理と②仮想ネットワーク管理、③セキュリティ管理、そして、④仮想環境と仮想マシンの運用管理があります。具体的なイメージは**図3**のようなものです（次回に概説します）。

階層化管理はサーバ内のVMMとその上のVM、そしてこれに接続する実端末、および、実際の利用者という4階層の階層間（縦、上下）および階層内（横）のアクセス・運用管理です。

仮想ネットワーク管理は仮想環境ネットワークと実・仮想ネットワークの運用管理です。

セキュリティ管理では、実環境と仮想環境のインターフェースと仮想環境内の各構成要素間のインターフェースの2つのインターフェースでのアクセス制御です。

▼図3　仮想化管理の位置づけ



仮想環境と仮想マシンの運用管理は、具体的には、管理者が行う仮想環境の運用管理と仮想マシン上のシステム(サーバ、デスクトップ)の運用管理です。仮想マシンを利用する者、つまり利用者自身に任せるしくみ、それが「利用者自身運用管理」です。一般には、仮想化管理者が利用者や利用者端末の運用管理を行いますが、仮想マシンについては利用者自身がよく知っているわけなので利用者自身に任せることができます。これによって仮想環境管理者の負担を軽減でき、より効率的な仮想化管理が可能になります。

次回以降:「サーバ仮想化」を実践していきましょう

次回以降の本連載は、ハイパーバイザ型サーバ仮想化を題材に、CentOS 6.7上のKVMを使って、ホストシステムや仮想環境の構築から仮想マシン/ゲストシステムの利用まで実際に使ってみながら仮想化の問題を考えてみます。

連載の前半では、デフォルト最低限の仮想環境、つまり、単一仮想マシンシステムとしての利用から進めて、後半で現実の使い方である、複数仮想マシン/ゲストシステムの仮想ネットワークと物理ホストを含む実ネットワークから構成されるネットワーク環境での仮想化を試してみます。SD

連載各回では、読者皆さんからの簡単な「ひとくち質問(QA)コーナー」や「何かこんなこともしてほしい要望(トライリクエスト)コーナー」を設けて「双方向連載」にできればと思っていますので、質問や要望をお寄せください。

宛先:sd@gihyo.co.jp
件名に、[仮想化連載]とつけてください

RDB性能トラブルバスターズ奮闘記

原案 生島 勘富(いくしま さだよし) info@g1sys.co.jp ㈱ジーワンシステム
構成・文 開米 瑞浩(かいまい みずひろ) イラスト フクモトミホ

第3回 DBサーバとAPサーバの役割分担を知っておこう

アプリケーション(AP)サーバでのループ処理は、システムの性能を急激に悪化させる一因ですが、いったいループの何が悪いのでしょうか。今回、大道君はAPサーバとデータベース(DB)サーバの役割を整理することで、ループが良くない理由と、集計はできるだけSQLでやったほうが良い理由が、見えてきたようです。

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。

「ループ」が引き起こす3つの問題

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。多くの開発現場で、「SQLをきちんと使えていない」ことによってさまざまな問題が多発しているのをあまりにもよく見かけるので、「SQLをまともに理解しよう運動」を呼びかけるために、この連載を書いています。

前回までに、浪速システムズに勤める大道君という若いエンジニアの相談を受けて、いくつか確認したことをまとめたものが、**図1**です。

たとえば、あるテーブルから一部のデータを抽出したり、集計したり、複数のテーブルを結合したりする場合、これらはいずれも「データの集合に、定型的処理を加えて、別な集合を作る」操作です。そして重要なのは、この「定型的処理」をする際にSQLなら「①：ループが不要である」ということです。

「毎回ループを強調していますけど、どうしてそんなに重要なんですか？」と大道君。

「ループを使うといろいろ困った問題が起こるんよ。ざっくり言うとこの3つやな」

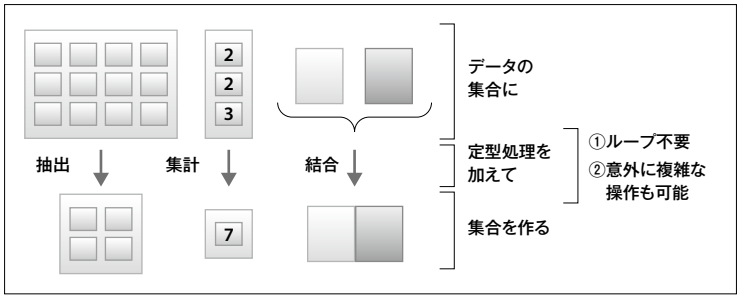
- バグが増える
- 工数がかさむ
- 性能が出ない

「それぞれどうしてか、**リスト1**を見比べてみれば想像つくやろ？」

リスト1は本連載の第1回でも載せた、JavaとSQLで似た集計処理をするコードの例です。

「Javaプログラムのほうはループなんですよね。つまり……バグが増えるのは、ループを使うと

▼図1 SQLは集合指向の言語



maxとかiのような作業変数、制御変数が必要になる分、間違えて書くリスクが増えるからで……そうすると書く量が増えてバグも出やすいから工数がかさむのは当然で……」

と、大道君は自分で考えたうえで答えを返してきます。頼もしい。

「そのとおり！　性能は？」

「性能が出ないのは……あれ？　ええと？」

おっと、これがわからないということは、大道君もコードの上だけで考えていて、実機のハードウェア上でどこをどうデータが処理されて流れていくかというイメージを持てていないのかもしれません。

DBとAPの役割分担を考える

「性能の件も含めて、こういう構造は知っといたほうがいいよ」と私は**図2**を見せました。

通常、DBサーバとAPサーバ、Webサーバは別ですので、ここではAPとWebを一緒にしてDBサーバと分けてあります。アプリケーション側のプログラムは「プレゼンテーション」「ビジネス・ロジック」「データ・アクセス」の3層に分けて考えることができます。データ・アクセス層がDBサーバにSQLを投げて取得した実行結果をデータ・セットとしてAPサーバ内に保持し、それにビジネス・ロジックで適当な加工を施してプレゼンテーションでユーザに対して表示するわけです。

DBサーバがSQL文を受け取るとパーサー、オプティマイザが解析して実行計画を作り、それを実行して結果をAPサーバに返します。

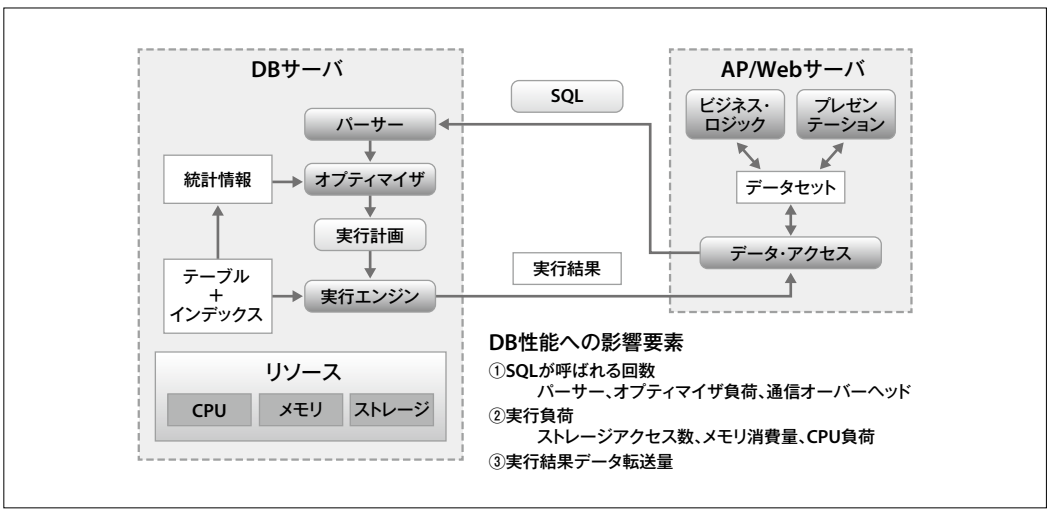
こうした構図の中でDB性能に影響する要素

▼リスト1　集計処理（本連載第1回のものを再掲）

```
手続き型言語（Java）での集計処理（orders配列のBillingの合計、最大、平均値を算出）
  int sum = 0, max = 0, avg = 0;
  for(int i=0; i < orders.length ; i++){
     sum += orders[i].Billing;
     max = (max > orders [i] .Billing) ? max : orders [i] .Billing;
  }
  if( orders.length > 0 ) avg = sum/orders.length;

集合指向言語（SQL）での集計（orderテーブルのBillingの合計、最大、平均値をcustomer_idごとに算出）
  SELECT customer_id, sum(Billing) , max(Billing), avg(Billing)
         FROM order
         GROUP BY customer_id;
```

▼図2　DBとAPの役割分担を考えるための見取り図



は大まかに3つあります。まず①SQL文が呼ばれる回数が多いと、SQL文を実行計画に変換するパーサーとオプティマイザの処理、および通信プロトコルによるオーバーヘッドがかさむため、できるだけ少ないほうが良いわけです。

「単純なSQLをループで何千回も投げてAP側で集計するなんてのは、論外ってわけですね」

「そういうこっちゃ」

次に②実行負荷に影響するのは、DBサーバの物理リソースであるストレージ、メモリ、CPUをどれだけ使うかです。CPUよりもストレージへのアクセス回数とメモリ消費量がボトルネックになる場合が多く、これを減らすためには、テーブルとインデックスの設計、およびそれをふまえた効率の良いSQL文の設計が重要です。

「ここはSQLをよく理解していないとできないところでしょうか？」

「そうなんよ。SQLが苦手なエンジニアはたいていこれができないんよね」

最後に③実行結果データ転送量ですが、ネットワークの転送速度はどうしてもメモリバスの帯域幅よりも遅いので、使いもしないデータをDB－APサーバ間で大量に転送するとその分遅くなります。

「やっぱり、集計値しか必要ないのに、明細データを全部転送してAP側で集計するなんてのは論外、と……」

「論外もええとこや。**リスト1**のコードでも、実際の業務APだとJavaのループ内でDBにクエリを投げるわけやろ？　そうするとループの回数分だけネットワーク越しにSQLを投げて大量のデータを転送することになるわけで、①と③に該当して遅くなる。SQLで集計すれば、クエリは1回で済むしネットワークを転送するのは集計結果の小さなデータだけ」

「じゃあ、この種の処理は必ずSQLでやるべきで、手続き型言語のループでやったほうがいい場合というのは存在しないんですか？」

「そんな場合があるんやったら教えてほしいわホンマ、もちろん例外はあるけどね」

「あ、でも、SQLでやれるのは『定型的な処理』だけですよね？　その処理の部分が複雑になってきたら、どうなんでしょう？」

「ところが実は『定型的な処理』と言っても、結構複雑なことができるんよ」

「え、そうなんですか」

「なのに、SQLが苦手なエンジニアはそれを知らんか、知ってもやりたがらへん。ほんで、本来SQLで書くべき処理を**図2**の『ビジネス・ロジック』に載せてたりする」

「そうすると……①、②、③の全部にひっかかって遅くなりますか……」

「そういうことやね～。大道君はそんなSQL嫌いになったらあかんで～」

ところがそんな「SQL嫌い」の開発者が私たちの前に立ちはだかる機会は意外に早くやって来たのです。

生産計画を彩る超絶技巧的ループ処理!

それから数日後のこと、大道君から私にある相談がありました。浪速システムズ社内の別チームで作っている機能がやはり遅くて困っているので、意見を聞きたいとのこと。なんでも、受注生産品の生産指示をする画面だそうで、在庫の部品で生産可能な製品のリストと生産可能個数を出すのだそうです。これもやはりSQLを使わず、ものすごく複雑なループ処理で書かれて

いました。ただ、あまりにも複雑過ぎてそのままでは本連載では書き切れませんので、バッサリ大幅に単純化してエッセンスだけ示すと概要は**図3**のようなものでした。なお通常は部品表と在庫のひもづけはIDで行い、製品名や材料名はIDでマスタテーブルから引いてくるのが普通ですが、本記事では、やはり単純化のためマスタテーブルを省略し、IDではなく名称で結合する形で示してあります。

「要するに、在庫の部品を使って生産可能な製品のリストを、生産可能個数つきで出すというわけやな。まあ、こういうのは大したことないな。SQLでやれば簡単だよ。わかる？」

「うーん……ちょっと難しいです」

「手続き型言語のロジックならわかるかな？」

「わかりますけど、ソートして集計しながら製品名が変わったのを判断して……**図4**みたいな処理が必要ですよね。めんどくさいですね」

「それがSQLきちっと使えば一発になる」

「そこなんですけど……どう書くんですか？」

ここは正解のSQL文を教えてあげるより、この機会にSQLの考え方を応用させるほうが良さそうです。

「それじゃ、前回やったみたいに、カタマリを切り出す絵を描いてみてや？」

「はい、それじゃ……えーと、……こんな、感じ？(**図5**)」

「おお！　めっちゃええやん。大道君もだいぶわかってきたで！」

「ほんとですか？　でもまだSQL書けてないんですけど」

「まあ、とりあえず流れを説明してみてや」

「じゃあ、番号順にいきますと……」

以下、大道君の説明を箇条書きにしました。

❶結果セットの製品名の部分を部品表から切り出す。製品ごとに集約して出すので、ここはGROUP BYでグループ化する

❷部品表テーブルの必要量と、在庫テーブルの在庫数を比較したいので対照表を作る。ここはINNER JOINで行う

❸その対照表から、「在庫数／必要量」を計算して小数点以下を丸めると、部品ごとに見た生産可能数が出る

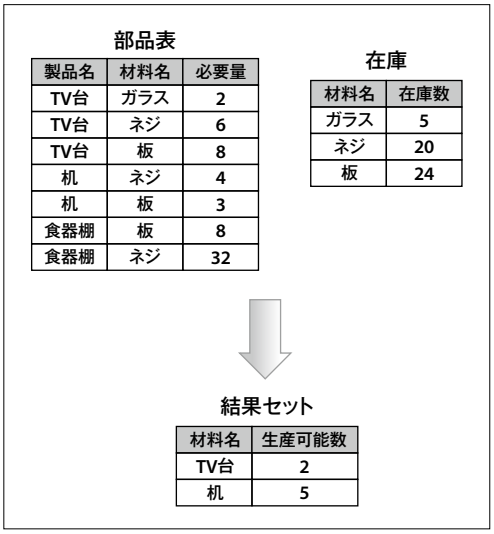
❹1つの製品についての部品ごとの生産可能数の最小値が、その製品の生産可能数になる

❺それを「生産可能数」として結果セットに入れてやる

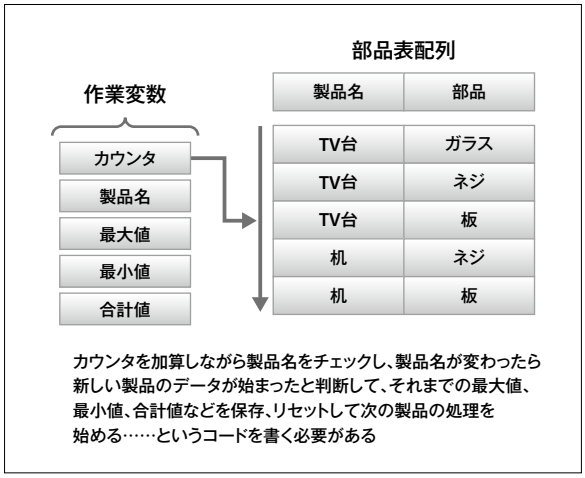
「よーし、じゃあそれをSQLにしたってや！」

「……あ、こうすればいいのかな？」と大道君が書いたSQL文が**リスト2**でした。

▼図3　生産可能品一覧



▼図4　ループ処理のためには複雑なコードを書く必要がある



「おっ、なかなかええやんか、これでもう一工夫して、生産可能数がゼロの製品は出さないようにするとしたら？」

「絞り込むなら、WHERE句を付けて……」

「WHERE句はGROUP BYした結果の絞り込みには使えないんよ」

「あ、そうか、じゃあHAVINGですか。生産可能数はTRUNCATE(MIN(…))で出るわけだから、それが1以上という条件を付けて……」と、大道君は**リスト3**を追加しました。

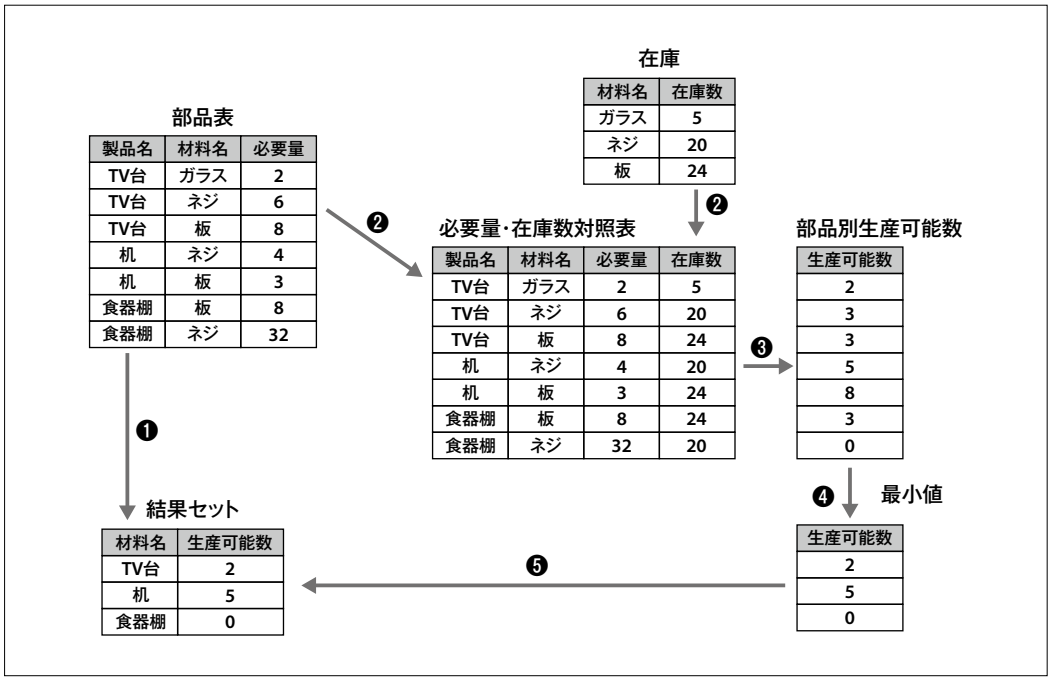
「そうそう、それで**リスト2**と**リスト3**をつなげてやればええんよ」

「あ、これでいいんですか……」

「単純やろ？」

「なんだ、単純ですね……なんなんでしょう、このシンプルさ(笑)。 ほんと、手続き型言語と

▼図5　生産可能品一覧生成ロジック(SQL版)



▼リスト2　生産可能品一覧生成SQL文

```
SELECT b.製品名, TRUNCATE(MIN(z.在庫数 / b.必要量), 0) AS 生産可能数
FROM 部品表 b   INNER JOIN 在庫 z
     ON b.材料名 = z.材料名
GROUP BY b.製品名
```

▼リスト3　生産可能品のみ絞り込むHAVING句

```
HAVING TRUNCATE(MIN(z.在庫数 / b.必要量), 0) >= 1
```

は全然違う」

「さっきのループで書くロジックのめんどくささと比べたら、月とスッポンやろ」

「何かこう、気持ちいいんですけど(笑)」

「そやろ？　だからSQLをちゃんと使おう、言うてんねん。このほうがよっぽど楽や！」

ループをなくすことはSQLの設計目標の1つだった

「実はSQLはもともとこういう処理をループを使わずに書けるように、という設計思想で作られてるから、できて当然なんよね」

「そうなんですか！」

「**図4**でそのへんがわかるけど、手続き型言語で集計処理を書こうとすると、カウンタとか最大／最小値とかの作業変数を用意して、それを判断して更新するコードを書かなあかんやろ？その分バグも入るし工数もかかるし、遅くなるしで、いいことは1つもない。ところがこういう集計処理、業務システムではよく使う。ナントカ一覧画面とか帳票とかを出そうとすると、たいていこういう『集合→定型処理→集合』のパターンが出てくるわけよ。だったらもうループを使わんで書けるようにしよう！ってんで生まれたのがSQL」

「そうなんですか……ほんと、ループ使ってるところには要注意！なんですね」

「そういうこっちゃ！」

これでこの話も一件落着……と、私たちはそう思っていたのです。SQLをまともに使わず、手続き型言語の死ぬほどめんどくさいループで書かれている処理を、SQLに変えれば楽だよ、ということで変えたものを作ってあげました。しちめんどくさいコードがシンプルになり、わかりやすくなり性能も上がる。良いことづくめじゃないか。良かった良かった、と。

SQL嫌いのボス登場?

そう思っていた数日後、五代さんがちょっと重い表情で口を切りました。

「先日の生産計画機能の件なんですけど、あれ、向こうの担当さんが、SQL見せられてもわからん、わかるように仕様書書いてくれ、と言うんですわ……それも何かすごく不愉快そうに言うんで、こら〜、向こうと直接話をしてもらうとケンカになりそうな気もしましてね、その前にご意見うかがっとこうと思いまして」

「ははあ、仕様書ですか……以前お話ししたとおり、SQL使うときは、ゴールを示す以外の仕様書はいらん場合が多いんですが」

「そうなんですがね、どうもSQL自体あまり使いたくないと思ってそうでして……」

「ああ、よくいます、そういう人。ひょっとして、SQLで処理させるとDBサーバの負荷が高くなるとか言うてませんでしたか」

「ええ、言ってました」

実際はSQLでやるべき処理をビジネス・ロジックに載せることによる負荷のほうが大きいし(**図2**)、本来帳票のような一括データ処理に向くSQLを使わず、手続き型言語のループ処理で書いていたら(**図4**)、大局的にはバグ、工数、性能を悪化させるだけです。

「下手なSQLを書いてるだけなんですけどね。あと、複雑なSQL使ったら、メンテできるメンバーがいない、とかも言うてませんか」

「ああ、それも」

「やっぱりSQLわからん人なんですね……」

「それ言ったらケンカになりますわ……」

ということで、私たちの前に難題が立ちふさがってきました。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、経験だけを積み重ねてきたタイプのベテランは、自分がわかる範囲に固執して聞く耳を持たないことがあります。一覧・帳票処理にSQLを使わない、なんて私に言わせれば風呂の水をスプーンで掻き出すようなものなのですが、それをストレートに言ってはいけないのでしょう。

それにしても、ではどう言えばいいのでしょう。私たちは3人で考え込んでしまいました。

SD

presented by
Japan Android Group
http://www.
android-group.jp/

コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

第5回 Picassoとキャッシュの上手な使いこなし

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

重村 浩二（しげむら こうじ）
日本Androidの会
中国支部長
Mail k-shigemura@
android-group.jp

画像表示の悩みを解決しよう

Androidアプリを開発していると、画面上に画像を表示したいケースがさまざまな場面で出てきます。ほかにはない使いやすさやわかりやすいコンテンツを提供しようとすると、いずれかの場面で画像表示が必要になります。

画像表示はAndroidでも標準でUIコンポーネントが用意されており、ImageViewを活用すれば比較的簡単に画像が表示できます。しかし、端末上やアプリ内のリソースとして用意した画像を表示する場合であればいいのですが、Web上に置かれた画像をHTTP通信などを介して取得しようとすると途端に難易度が上がります。

取得自体は**リスト1**のような実装をすれば可能ですが、この実装では同期処理となってしまい、ネットワークが遅延したときなどにANR（Application Not Responding）が発生し、アプリケーションが異常終了する原因となってしまいます。

このようなときの解決策は、HandlerやThread、AsyncTaskなどを利用して、UIスレッドと別スレッド上で非同期に画像を取得することで解決できます。とはいえスレッドの実装はタイミングの調整などが必要になり、バグが潜り込みやすい箇所となります。なるべく安全に、ほかのエンジニアも活用している資産を利用したい……と考えると、オープンソースで公開されている、次のような便利なライブラリの利用が望ましいでしょう。

- Picasso
- Glide
- Fresco

今回はそれらライブラリの中から国内で人気がある「Picasso」について、基礎から実践的な利用方法までを通して見ていきます。

Picassoとは

まずはじめに、Picassoについて簡単に解説しましょう。PicassoとはSquare社が提供するオープンソースの画像ライブラリです[注1]。build.gradleでライブラリの取り込みを行えば（**リスト2**）、loadメソッドで対象の画像を読み込み、

注1）http://square.github.io/picasso/

▼リスト1　ImageViewでの画像情報取得（避けたい実装）

```
ImageView imageView = (ImageView) findViewById(R.id.imageView);
imageView.setImageURI("http://example.co.jp/hoge.jpg");
```

▼リスト2　Picassoの初期設定

```
// build.gradleに追記します(2016/03/08時点の最新版)
dependencies {
    :
    compile 'com.squareup.picasso:picasso:2.5.2'
}
```

▼リスト3　Picassoを用いた実装例

```
// Activity#onCreate()で呼び出した場合の例
ImageView imageView = (ImageView) findViewById(R.id.sampleImageView);

Picasso.with(this)
    .load("http://example.co.jp/hoge.jpg")
    .placeholder(R.drawable.loading_image)  // 読込中に表示する画像
    .into(imageView);
```

▼リスト4　右方向に回転させる実装例

```
Picasso.with(this)
    .load("http://example.co.jp/hoge.jpg")
    .rotate(225)      // 画像を225度、右回転します
    .into(imageView);
```

intoメソッドで描画を行うImageViewのインスタンスを指定することで、画像を描画することができます(**リスト3**)。

Picasso自体が遅延読み込みにも対応しているため、Web上から画像が取得できるまでは**リスト3**のplaceholderメソッドで指定した画像を初期画像として表示し、取得できたら画像を差し替える、といったことも自動でやってくれます。また、Picassoを利用すれば簡単な画像の回転や加工(サイズ変更や角丸画像への変更)も可能になっていますので、アプリで実現したい表現に沿って柔軟な対応が可能です。たとえば**図1**のように画像を回転したい場合には、**リスト4**にあるようにrotateメソッドを指定することで画像を右方向に回転させることができます。

より詳細な情報が確認したい場合には、次のサイトもあわせて確認を行ってください。

・JavaDoc
http://square.github.io/picasso/2.x/picasso/

▼図1　右方向に回転した表示例

・StackOverflow
http://stackoverflow.com/questions/tagged/picasso?sort=active

では、実際にPicassoを利用するときのポイントを見ていきましょう。

いま表示している画像の情報源はどこ？

Google Playに公開するような、製品としてのアプリを開発する場合には、とくに画像の情

▼リスト5　Picassoのデバッグ設定例

```
// Picassoのインスタンスを Picasso.Builderで生成する。
Picasso picasso = new Picasso.Builder(this)
    .indicatorsEnabled(true) // ①インジケータの有効化
    .build();

// 画像を読み込み、表示を行う。
picasso.load("http://example.jp/hoge.jpg")
    .into(imageView);
```

報源がどこなのかが重要になります。おもにWeb上にある画像データを通信を通して取得して描画を行うアプリでは、毎回通信を行うと通信環境によって画像データの取得までに時間がかかってしまう場合があります。画像の表示までに時間がかかるわけですから、ユーザにとって使いにくいアプリと言えるでしょう。データサイズが小さければ許容されるケースもありますが、カメラで撮った写真などを扱おうとすればサイズは大きくなりますので、毎回Webから取得するというのは避けるべきでしょう。

そんな課題を解決するための手段として、ローカルに画像をキャッシュして保存しておき、描画したい画像が同じデータであればキャッシュから表示をするという戦略が考えられます。

Picassoでもデフォルトでキャッシュを行っており、読み込んだ画像が同一であればPicassoが自動で判別し、デバイス上にキャッシュとして保存しておいた画像を表示してくれます。現在表示されている画像がキャッシュなのか、Webから取得してきたものなのかは、Picassoのデバッグ設定を有効にすることで確認ができます。

Picassoのデバッグ機能は**リスト5**にあるように、Picasso.BuilderでPicassoのインスタンスを生成する際、**リスト5**の①にあるindicatorsEnabledメソッドにtrueを渡して有効化する必要があります。有効化すれば画像がどこから取得されているのかを確認できるようになります。

- ネットワーク経由：赤
- ディスク経由：青

▼図2　Picassoのインジケータ表示例

- メモリ経由：緑

図2の表示例のように、最初に実行したときには画像の左上角部分が赤色になります。アプリを一度終了し、再度立ち上げれば青色になり、ディスク上のキャッシュが利用されていることが確認できるという具合です。

Picassoがキャッシュを判別できるのは、Web上からデータを取得するためのURLが固定となっている場合です。言い換えると、loadメソッドで呼び出すURLが静的な、一意となるキーの場合にPicassoはキャッシュから画像を表示することが可能です。このURLが可変になると、キャッシュをPicasso任せにするわけにはいかなくなります。

URLが固定でないときの処方箋

では、キャッシュが効かない場合にはどうすべきでしょうか。

たとえばAmazon S3(ストレージサービス)では、データをダウンロードするために取得用のURLを要求する必要があります。このしくみが採用されているおかげで、発行されてから

▼リスト6　build.gradleにOkHttpClientライブラリの読み込みを追記

```
// build.gradleに追記します(2016/03/08時点)
dependencies {
    :
    compile 'com.squareup.okhttp:okhttp:2.7.4'
}
```

▼リスト7　ダウンローダ(Downloader)の設定込みを追記

```
// ダウンローダの設定
final OkHttpDownloader okHttpDownloader = new OkHttpDownloader(this);

Picasso picasso = new Picasso.Builder(this)
    .indicatorsEnabled(true)
    .downloader(okHttpDownloader)
    .build();
```

一定時間内のみデータが取得できるようなURLにすることができ、セキュアな設計となっています。しかし、データを取得するときのURLは毎回変更となるため、Picassoの標準のキャッシュ機能は利用できません。

この場合、ここまでに紹介した実装方法では毎回ネットワーク経由で画像を取得することになってしまいますが、Picassoは標準では対応できないようなケースにも対応できるように、機能を拡張するためのしくみを豊富に持っています。**リスト5**ではインジケータの有効化を行いましたが、ここで出てきたPicasso.Builderを使うことで、そのほかにもさまざまな設定が行えます。可変URLを扱う場合にはその中の1つ、Downloaderの変更を利用します(**リスト6、7**)。

OkHttpDownloaderは内部でOkHttpClientというSquare社製のライブラリを利用しているため、**リスト6**のようにbuild.gradleに追記してライブラリを読み込む必要があります。そうすれば**リスト7**のようにOkHttpDownloaderのインスタンスを生成し、downloaderメソッドに渡すことで、Picassoが取得してきたデータをプログラム上で取り扱うことができるようになります。

okHttpDownloaderは**リスト8**のキャッシュタイミングで利用します。まずキャッシュが存在するのかどうかで条件分岐を行います。キャッシュがあれば対象のFileインスタンスを元に画像を読みこんで表示を行い、キャッシュがなければURLから画像の表示を行います。

このときに呼び出すPicasso#intoメソッドには、第二引数でCallbackインターフェースを引数として渡すことができます。Callbackインターフェースには onSuccessメソッドと onErrorメソッドが用意されているので、それらをオーバーライドしてやれば、画像の読み込み処理の成功と失敗が判別できます。

画像の取得に成功したときにOkHttpDownloader#loadメソッドでレスポンスの情報を取得し、OkHttpDownloader.Response#getInputStreamメソッドを呼び出せばInputStreamのインスタンスが取得できるので、後はファイルIDをキーにしてキャッシュすればよいでしょう。**リスト8**ではgetCacheDirメソッドに指定したファイル名で保存するようにしています。

キャッシュ時の注意点とライブラリのさらなる活用

キャッシュを適切に行うことでオフライン時にもある程度は情報が閲覧できることは、ユーザの満足度を向上させることにもつながるかと思います。今回のサンプルでは触れませんでしたが、実際にユーザに提供する際には、キャッ

▼リスト8　ImageViewへの画像設定時のコールバック処理

```
/* 画像のファイルIDをキーにして、キャッシュのFileインスタンスを取得 */
if (/* キャッシュの存在チェック */) {
    picasso.load(/* キャッシュのFileインスタンス */)
        .into(imageView);
} else {
    // キャッシュが存在しなければ、URLから画像を取得し、成功したときにキャッシュしておく
    picasso.load(/* 画像取得URL */)
        .into(imageView, new Callback() {
            @Override
            public void onSuccess() {
                OkHttpDownloader.Response response = okHttpDownloader.load(Uri.parse(url), -1);
                // ファイル名をつけてキャッシュを行う（fileNameは一意に識別できるファイルIDを含むこと）
                BufferedInputStream bufferedInputStream =
                    new BufferedInputStream(response.getInputStream());
                BufferedOutputStream bufferedOutputStream =
                    new BufferedOutputStream(new FileOutputStream(new File(getCacheDir(),
                                                                    fileName)));

                int c;
                while ((c = bufferedInputStream.read()) != -1) {
                    bufferedOutputStream.write(c);
                }
                bufferedOutputStream.flush();

                bufferedInputStream.close();
                bufferedOutputStream.close();
            }

            @Override
            public void onError() {
                /* 画像が取得できなかった旨のエラーをToastなどで表示 */
            }
        });
```

シュの確保に用いるディスク容量も上限を設け、一定サイズ以上になったら古いものから削除するしくみを導入したほうが好ましいでしょう。

対応方法としては、スクラッチで書く方法もあるかと思いますが、オープンソースで提供されている「DiskLruCache」のようなライブラリを活用するのもまた一案でしょう。また、**リスト7**で設定したOkHttpDownloaderの引数には、OkHttpClientのインスタンスを渡すことも可能です。異なるレイヤーでキャッシュを実現することも一つの方法として検討してみるのも良いでしょう。

いずれにしても、キャッシュをするうえで必要なインターフェースはPicasso周辺のライブラリで補完ができることから、Picassoの主要機能を含めてさまざまな応用方法があることが読者の皆さんに伝われば幸いです。SD

COLUMN

ライブラリのバージョンにはご注意を

オープンソースで公開されているライブラリは、最新版を活用することでたいていは問題ありません。むしろ最新バージョンのほうがバグフィックスが行われていたり、安定性が増していることが多いので、積極的に検証を行い、更新をかけていくべきかと思います。しかし、ライブラリ間の依存関係がある場合には、注意が必要な場合があります。

今回のダウンロード内で使われているOkHttp Clientは、執筆（2016年3月8日）時点ではバージョン3が最新版として公開されていますが、Picasso側がそのアップデートに追いついていないため旧バージョンの2系でしか利用できません。そのため、今回のサンプルでも**リスト6**で読み込んでいるバージョンが古いものとなっています。

ライブラリ間の依存関係は見た目だけではわかりにくいものです。コンパイラから「クラスは存在しません」などとエラーが出てきた際には、ライブラリ間の依存関係も疑ってみてください。

COLUMN

コミュニティの運営ノウハウ

■勉強会の開催ノウハウを公開

これまで筆者は中国地方を拠点として、30回超の勉強会を開催してきました。これから勉強会の開催に興味がある方に向けて、開催方法と実際に開催するときの注意点を簡単ですが紹介します。

■勉強会を開催するタスクリスト

勉強会の開催のためにやることは次のとおりです。

- 日程、会場、講師決定
- タイムテーブル決定
- 勉強会アナウンス(ML、ブログ、SNSなど)
- 懇親会会場の決定&アナウンス
- 当日運営

毎月勉強会をやっているような積極的なコミュニティでしたら、大体3~4週間前(イベントが終わった直後や、その前)から講師や会場の調整などを進めているのが一般的です。日程は、可能であれば毎月いつ開催するのかを運営メンバー内で決めておくとやりやすいと思います。

■会場は有料? 無料?

会場は、有料の会場と無料の会場ではどちらを選ぶべきでしょう? 20~30名程度集まるイベントであれば、大学などに協力してもらい、無料で開催することを模索すると良いでしょう。難しいようであれば、半日4,000円程度で貸し出してくれている会議室を活用するのも手です。筆者が開催したときには1人あたり200~300円の参加費とし、会場費を参加者全員で割り勘していました。継続して開催していく中で、勉強会でつながった方から、大学などとのネットワークを作っていくのが良いかと思います。

■一番の悩みは講師

勉強会を開催するときに一番悩むのは講師の調整です。日程や会場はある程度しくみが出来上がってくると簡単に回るようになりますが、講師だけは毎回同じ人が登壇していると、徐々に疲弊してきますし、内容がマンネリ化してしまうことにつながりかねません。筆者の場合は、基本的に土曜日の午後1時からの開催で、講師が3~4名登壇するというスタイルで1人50分の枠で話してもらっています。たとえ筆者が1人分の枠をやるとしても、残りの2~3人の講師を見つけてくる必要があり、時には同じ人にたびたびお願いすることもありました。講師の選定は今も大きな課題の1つです。

筆者がこれまで実践してきた解決策としては、参加してくれている方の中から話してくれそうな方を見つけつつ、全員参加型のイベント(ハッカソンやハンズオン)を行ったり、近隣で開催されている別のコミュニティと合同で勉強会を開催するなどして、ネットワークを広げていきました。1人でも多く、講師を担当してくださる方とつながっていく努力を積み重ねることが大事かと考えています。

■継続していくために

講師の問題を解決できれば、勉強会を開催する課題はほぼほぼ解決した状態かと思います。しかし、継続していくのは思った以上にパワーを使うものです。長く勉強会を続ける秘訣は、一緒に勉強会の開催をしてくれる仲間を見つけることでしょう。参加者の中には、継続して参加してくれる人が必ず1人はいるはずです。その人を巻き込んで、一緒にコンテンツを考え、勉強会を作っていくことができれば、うまくしくみが回り始めるかと思います。

そしてぜひ参加者の方へのアンケートを取ってみて、どんな人が来てくれているのかを定量的に確認していくと良いでしょう。筆者が開催していたときには、初めて来てくれた方が毎回2~3割はいる状態を維持するようにコンテンツの調整をしていました。同じ方だけが集まってスキルを高めていくというのもやり方の1つだと思いますが、常に新しい発見をもたらしてくれる方が参加し続けてくれることを考えると、新しい人の流入が続いている状態がコミュニティとしては健全であると考えています。回を繰り返せば参加者のレベルは高くなり、難易度の高い内容が求められるようになっていくかと思いますが、ぜひその中に初心者向けのコンテンツなども含めるようにして、新しい人を取り込むようにしていってください。

コミュニティの運営はとても地味で、細かい調整事ばかりが発生し面倒なものです。いろんな雑用を一手に引き受けることの労力はありますが、それ以上に新しい知見が得られることと、さまざまな方とコミュニケーションを取ることで主催者側も大きく成長できることを思えば、決してやって損はありません。ぜひ運営にチャレンジしてみてください。

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://rubikitch.com/

第25回 シェルコマンドを活用しよう（前編）

今月来月と、Emacs上で使えるシェルコマンドについてみていきます。端末をいちいち起動しなくても高度な機能が呼び出せるのが強みです。今回はM-x shellとM-x eshellの違いから、region（リージョン）に関する便利コマンドまで紹介します。

どーも、るびきちです。新年度が始まりましたね。というわけで、Emacsの基本的な内容を取り上げていこうと思います。

今回のテーマは「シェルコマンド（外部プログラム）」です。Emacsの優れた点の1つとして外部プログラムとの連携があります。ただ呼び出すだけでなく、バッファの内容を外部プログラムの入力にしたり、出力をバッファに挿入したりできます。それだけでなく、実行時間のかかるプログラムを並列で動かしたり、対話的プログラムを実行したりもできます。さらに推し進めたものとして、特定のプログラム専用インターフェースもたくさん存在します。「Emacsは環境」と言われるのは、elispでできることが多いこともそうですが、外部プログラムによるところも大きいです。

シェルバッファ

Emacsでシェルコマンドを実行するもっとも簡単な方法は、専用のシェルバッファを使用することです。既存のシェルを動かすM-x shellとelispで書かれたシェルを動かすM-x eshellがあります。

M-x shell

M-x shellを実行すれば、お使いのシェルを*shell*バッファで動かします。シェルのプロンプトが現れるので、端末と同様にそのままシェルコマンドを入力・実行してください（**図1**）。

複数のシェルバッファを立ち上げるには、C-u M-x shellを実行するか、*shell*バッファをリネームしてからM-x shellを実行します。バッファ名をリネームするには、M-x rename-bufferやM-x rename-uniquelyを使います。前者は任意の名前にリネームし、後者は「*shell*<2>」のような数字サフィックスが付きます。

M-x shellでは、シェルがEmacsのバッファで動いているので一長一短です。次の長所があります。

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

もっとも、①〜③はGNU ScreenやtmuxというUNIX端末マルチプレクサを使えば実現できますが、Emacsで完結しているのは快適です。

M-x shellはシェルの位置付けを変えました。元来テキストエディタはシェルのプロンプトから実行され、対象ファイルの編集が終わればシェルに戻ってきます。この使い方では「まずはシェ

ルありき」です。一方、M-x shellはEmacsから実行されるので、Emacsの管理下にあります。Emacsを立ち上げている限り、いくつでも内部でシェルを実行できることになります。

とはいえ、端末上のシェルにかなわない点もあり、次のような短所があります。

❶画面志向のプログラムを実行できない
❷補完が弱い
❸コマンドライン履歴が共有できない
❹シェル専用のコマンドが使えない

画面指向のプログラムとは、画面全体を使うプログラムのことです。top（プロセスモニター）やw3m（テキストブラウザ）やpeco（helmの端末バージョン）などが該当します。これらのプログラムをM-x shellで実行すると、動作がおかしくなります。実行したい場合は、urxvt -e topなどと端末エミュレータを立ち上げて実行してください。

そもそも、対話的シェルは端末で実行されることが前提となっているので、端末ではないEmacsでシェルを実行しても、シェル本来の力を発揮しきれないのです。とくにzshはEmacsを思わせる拡張性を持ち、テトリスも実装されているほどです。zshはそれ自体が画面指向のプログラムなのです。

M-x shellの補完は基本的なプログラム名・ファイル名補完＋αしかできないため、bash/zshで作り込まれた補完設定よりも貧弱です。シェル独自の補完設定はEmacsからは使えません。コマンドライン履歴についても同じです。

筆者は、初級者時代はM-x shellを愛用していましたが、これらのような欠点があるため、いつしか使用をやめてしまいました。

M-x eshell

M-x eshellはフルelispで書かれたシェルで、M-x shellの利点をすべて継承しています。フルelispですのでプラットフォームを選ばないうえ、シェルそのものの挙動を思いどおりにカスタマイズできます。もちろん、足りない機能を付け足すこともできます。また、M-x shellと同様の方法で複数のeshellを立ち上げられます。

▼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)
```

M-x shellの欠点もいくつか克服しています。画面指向のプログラムは、プログラム名を指定して端末（M-x term）で実行でき、eshell専用のコマンドもあります。

筆者はzsh/eshellのコマンドライン全履歴にhelm/anythingでアクセスできるようにしたうえで、eshellを使っています。補完が弱いという欠点は長年溜めてきた履歴でカバーできるので気にしていません。

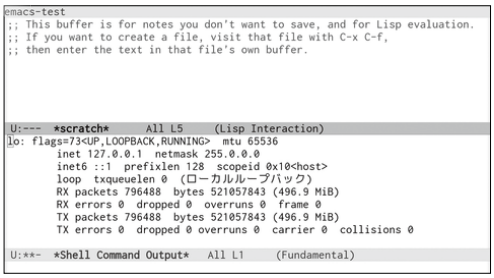
単発シェルコマンド呼び出し

実行結果をエコーエリア／別ウィンドウで表示する

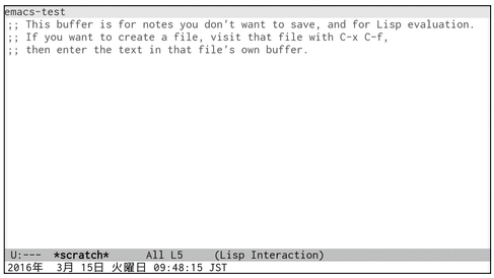
その場でシェルコマンドを実行するのに、わざわざM-x shell/M-x eshellを使うのは面倒に思うことがあります。そういうときは、その場でシェルコマンドを実行するM-!（shell-command）を使いましょう。

実行結果がある程度長い場合は、別ウィンドウ（*Shell Command Output*バッファ）で表示され（図2）、短い場合はエコーエリアで表示されます（図3）。数行の結果を表示するためにウィンドウ構成が崩されないのはうれしいですね。エコーエリアで表示される場合でも*Shell Command Output*バッファを見れば実行結果が格納されています。

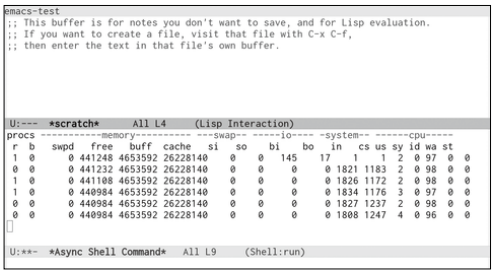
▼図2　M-! ifconfig eth0



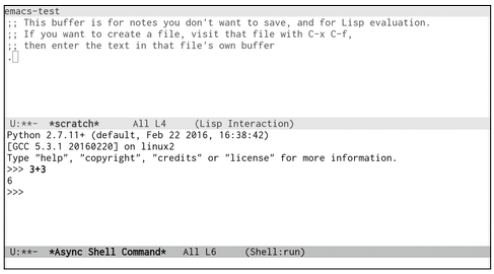
▼図3　M-! date



▼図4　M-& vmstat 1で1秒ごとにリソースの状況を表示



▼図5　M-& python -iで対話モードのPythonを実行



M-!ではユーザ入力を伴うプログラムと画面指向のプログラムは実行できません。前者は次項のM-&(async-shell-command)を使い、後者は端末エミュレータで実行してください。

並列(バックグラウンド)実行する

M-!はシェルコマンドの実行終了を待つので、実行中はEmacsの動作が停止します。ですので、時間がかかるプログラムを実行すると、その間待たされてしまいます。

M-!の代わりにM-&を使うと、シェルコマンドを実行してもEmacsの動作は継続されます。プロンプトは、M-!の「Shell Command:」から「Async Shell Command:」になります。実行すると別ウィンドウで*Async Shell Command*バッファがポップアップし、そこに実行結果が表示されます。

たとえば、vmstatプログラムは一定時間ごとにメモリやCPUの使用状況を表示し、C-cで強制終了するまで実行され続けます。これをM-!で実行するとEmacsがフリーズ(C-gで解除可能)してしまうので、M-&の出番です(**図4**)。停止させるときは、*Async Shell Command*バッファにてC-c C-cを押してください。

M-&ではユーザ入力を伴うプログラム(シェルなど)も実行できます。入力するときは、*Async Shell Command*のウィンドウを選択してください(**図5**)。

なお、M-!でもシェルコマンド末尾に&を付ければM-&と同じ挙動になります。ちょうど、シェルのバックグラウンド実行を連想させます。

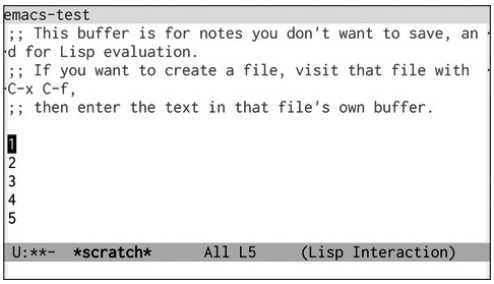
実行結果をバッファに挿入する

Emacs上でシェルコマンドを実行する意義は、それ自体を編集作業の一環とすることにあります。たとえば、C-u M-!はシェルコマンドの実行結果をバッファの現在位置に挿入します。seqプログラムは連番を出力するのでテキストエディタと相性が良いです(**図6**)。

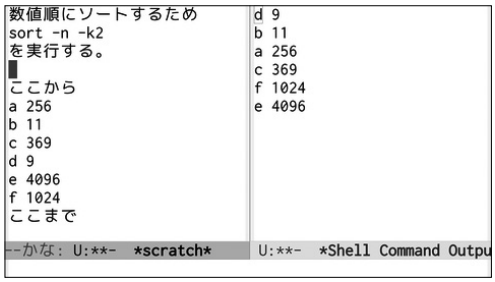
regionを標準入力にする

M-!の亜種M-|(shell-command-on-region)はregionを標準入力にしてシェルコマンドを実行します(**図7**)。

▼図6　C-u M-! seq 1 5



▼図7　regionを指定し、M-| sort -n -k2



regionを置き換える

C-u M-| はregionを標準入力にしてシェルコマンドを実行し、実行結果に置き換えます。図7においてC-u M-| sort -n -k2を実行すれば、regionは数値順にソートされた結果に置き換わります。C-u 2 M-x sort-numeric-fieldsでも同じ結果が得られますが、こちらはシェルコマンドさえ知っていれば実現できることに意味があります。

筆者はC-u M-| はもっと知られるべきコマンドだと思っています。シェルコマンドの習得が重要であることは今さら言うまでもありませんが、応用範囲の広いシェルコマンドの知識をそのままEmacsの世界に持ち込めることに意味があります。たとえばsortプログラムは、Emacsのソートコマンドよりも混み入った条件でのソートができます。

特定の加工をするEmacsのコマンドが存在しなくても、シェルコマンドがあればそれを呼び出してEmacsで加工できます。たとえば、標準入力で渡されたテキストをアスキーアートで装飾して出力するboxesというプログラム[注1]があります。「Hello world!」という文字列をregionにしてC-u M-| boxes -d shellを実行すると、次のように四角で囲んでくれます。

```
#################
# Hello world! #
#################
```

実行結果が意図するものではない場合は、C-/（undo）で元に戻してください。ほかにもたくさんの装飾方法が定義されているので、興味がある方はM-! boxes -lを実行してたしかめてみてください。

Emacsでできるテキスト処理方法を増やすには、elispをインストールしたり自作したりするのが一般的です。それに加えて、フィルタプログラム（標準入力を加工して標準出力に出力するプログラム）を他言語で記述し、それをC-u C-|で呼び出す方法もあるということを覚えておいてください。

Emacsを拡張する方法は、elispに限らないのです。

おわりに

筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指しています。手元でgrep検索できるよう全文をGitHubに置いています。またEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでもお答えします。「こんなパッケージ知らない？」「挙動がおかしいからなんとかしてよ！」はもちろんのこと、自作elispプログラムの添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

注1）URL http://boxes.thomasjensen.com

書いて覚える Swift 入門

第14回　型にまつわるプロトコルとLiteral Convertible

Writer 小飼 弾（こがい だん）　twitter @dankogai

本稿を書いている真っ最中の3月22日未明、iPhone SEや9.7-inch iPad Proの発表直後にApple製品用の春のアップデート祭りが始まりました。Xcodeも7.3に、そしてSwiftも2.2にアップデートされました。「このクソ忙しいときになんてことを」という気持ちが2割、「Swift 2.2が本稿に間に合ってよかった」が8割と言ったところでしょうか。

Xcode 7.3 with Swift 2.2

Swift 2.2と2.1の違いは、ソースコード無変更で警告を出しつつも、なんとかコンパイルしてくれる程度の違いではありましたが、Xcode用のSwiftがオープンソース版になったという意味で感慨もひとしお。リリース版のバージョンがLinuxも含めてそろったのは、これが初めてということになります(**図1**)。

しかし無変更でコンパイルが通るとはいえ、手元のプロジェクトではXcodeが警告でかなり黄色くなりました:-p。Swift 2.2では単なる警告でも、Swift 3.0では廃止予定のものが、わんさかdeprectedと警告されます。たとえば「PONS (Protocol Oriented Number System)」では、

```
typealias UIntType:POUInt
```

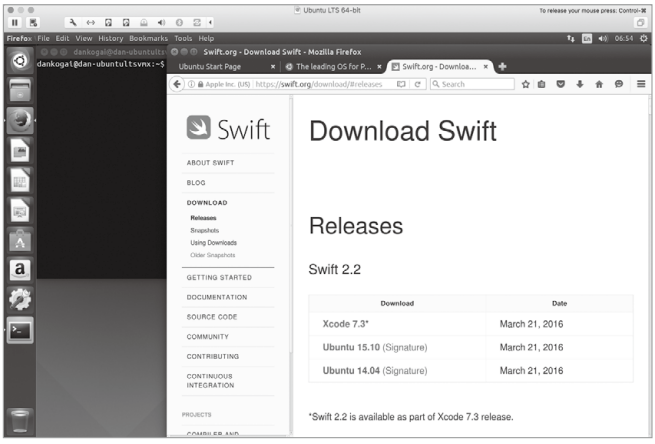
が、

```
associatedtype UIntType:POUInt
```

になったり、「enum Bitが廃止予定なのでIntに変更しろ」とあれこれ20ヵ所前後警告されましたが、Linux版が先行して2.2だったこともあり、Xcodeを7.3にアップグレードして30分足らずで、2.2への移行を完了できました。

性能、とくにArrayまわりの速度が改善されたのは非常にうれしい点で、とくにPONSでは任意精度数値を実装するのに配列を使いまくっているだけあってその効用は大きく、64ビットでもオーバーフローしない20!/19!の計算で、任意精度のBigIntとの速度比較で500倍近くあった速度差が100倍を余裕で切るところまできました。PONSのもくろみはあくまでProtocol-Orientedな数値型を実現することにあり、任意精度数値は「とりあえず動けばOK」だったのですが、これでかなり実用性が増したのではないでしょうか。

▼図1　Swift 2.2リリース



Literal Convertible

Swift 2.2の紹介はこれくらいにして、前号の続きです。さっそくですが問題です。iの型は何でしょうか?

```
var i = 1
```

正解はIntです。なぜ? 1はIntですから。では次のdは?

```
var d = Double(1)
```

正解はもちろんDouble。1はIntでもDouble()に食わせているのですから、最終的にDoubleになるのはごく自然です。では、次のxは?

```
var x:Double = 1
```

やはりDoubleです。print(x)してみると、確かに1.0と表示されます。1ではなくて。これはどうでしょう?

```
var y:Double = "1"
```

今度はerror: cannot convert value of type 'String' to specified type 'Double'というエラーを出して止まります。しかし、

```
var z = Double("1")
```

とすると……、 zには無事、Doubleの1.0が代入されるではありませんか。これはいったいどういうことなのでしょう?

「DoubleはIntegerLiteralConvertibleプロトコルに準拠しているが、StringLiteralConvertibleプロトコルには標準では準拠していない」というのが、その答えになります。

え? 答えになっていない?

では実際にvar d:Double = "1"を受け付けるようにしてみましょう。**リスト1**のようなコードをvar d:Double = "1"の前にペーストしてみてください。するとあら不思議。今度はエラーにならずにdに42.195が代入されています(**図2**)。

▼リスト1 StringLiteralConvertible

```
extension Double:StringLiteralConvertible {
    public init(stringLiteral: String) {
        self.init(stringLiteral)!
    }
    public init(unicodeScalarLiteral: String) {
        self.init(stringLiteral: "¥(unicodeScalarLiteral)")
    }
    public init(extendedGraphemeClusterLiteral: String) {
        self.init(stringLiteral: extendedGraphemeClusterLiteral)
    }
}
```

▼図2 Doubleの変数をStringで初期化

```
extension Double:StringLiteralConvertible {
    public init(stringLiteral: String) {
        self.init(stringLiteral)!
    }
    public init(unicodeScalarLiteral: String) {
        self.init(stringLiteral: "\(unicodeScalarLiteral)")
    }
    public init(extendedGraphemeClusterLiteral: String) {
        self.init(stringLiteral: extendedGraphemeClusterLiteral)
    }
}

var d:Double = "42.1950"                                    42.195
```

リスト1中のLiteral Convertibleとは、その型からの暗黙的な変換をサポートしているという意味なのです。IntegerLiteralConveribleなら整数リテラルからの、StringLiteralConvertibleなら文字列リテラルからの、という風に。「暗黙的」というのがポイントです。暗黙的ですので、初期化以外の目的にも使えます。たとえば、

```
42.0 + "0.195"
```

はエラーとはならずに、42.195になってしまうのです。

便利といえば便利ですが、濫用するとコードがわかりにくくなってしまいます。標準状態ではDoubleはIntegerLiteralConvertibleであっても、StringLiteralConvertibleでないという仕様は、賢明な判断と言えるでしょう。おかげで、

```
42 + 0.195
```

はエラーとならずに42.195になる一方、

```
var i = 42 + 0.195
```

▼リスト2　整数型リテラルでは表現できない数値を初期化

```
var bi:BigInt =
"933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536
979208272237582511852109168640000000000000000000000000"
```

はエラーとなるわけです。

ちなみに、Swift 2.2には現在で次のLiteralConvertibleプロトコルが存在します。

- ArrayLiteralConvertible
- BooleanLiteralConvertible
- DictionaryLiteralConvertible
- ExtendedGraphemeClusterLiteralConvertible
- FloatLiteralConvertible
- NilLiteralConvertible
- IntegerLiteralConvertible
- StringLiteralConvertible
- StringInterpolationConvertible
- UnicodeScalarLiteralConvertible

PONSでは、任意精度整数であるBigIntとBigUIntはIntegerLiteralConvertibleとStringLiteralConvertibleに、実数であるBigFloatとRationalはIntegerLiteralConvertibleとFloatLiteralConvertibleにそれぞれ準拠しています。おかげで**リスト2**のように、整数型リテラルでは大き過ぎて表現できない数値の初期化も可能ですし、

```
var bq:BigRat = 42.195
// (5938418321153065/140737488355328)
```

という具合に小数で分数を初期化することも可能になっています。

ちなみに**リスト2**のbiですが、実は100の階乗です。import PONSされた状態であれば、

```
(1...100).reduce(BigInt(1),combine:*)
```

として手軽に生成できます。もしIntegerLiteralConvertibleでなければ、

```
(1...100).map{ BigInt($0) }.reduce(BigInt(1)
,combine:*)
```

のように、ひとつひとつのIntを明示的に変換したうえで.reduceしなければならなかったでしょう。

▼リスト3　フィボナッチ数を計算する総称関数

```
func fib<T:POInteger>(n:T)->T{
    if n < 2 { return n }
    var (a, b):(T, T) = (0, 1)
    for _ in 2...n {
        (a, b) = (b, a + b)
    }
    return b
}
```

フィボナッチ数を計算する総称関数も、**リスト3**のように書けます。最初のif文のn < 2の2は、Intではなく Tなのですが、わざわざT(2)と書く必要はないのです。

Swiftの'ミステリー'

リテラルといえば、記号が欠かせません。たとえばStringのリテラルは、""(double quotation mark)で囲まれた部分ですし、{}で囲まれた部分がブロック＝関数リテラル、[]で囲まれた部分が添字またはArray/Dictionaryリテラルという具合に。「`(backtick)」ですら「予約語(keyword)を普通の識別子(identifier)として扱う」ための引用符としての役割を担っています。

その中にあって異彩を放つのが、「'(Single quotation mark)」です。PerlやRubyをはじめ、多くの言語では「変数展開なし(non-interpolating)引用符」としての役割を与えられているのですが、Swift 2.2時点では単なる文字にすぎません。将来使われるようになるのでしょうか？

そうやって見てみると、あらゆる言語の良いとこ取りをしてきたように見えるSwiftにも、まだまだ一層の発展の余地があるように見えます。どうなっていくのか、お楽しみはまさにこれからでしょう。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）

2015年11月号

第1特集
すいすいわかるHTTP/2
HTTP/1.1から変わること・変わらないこと

第2特集
攻撃を最前線で防ぐ
ファイアウォールの教科書

特別企画
・SMB実装をめぐる冒険
　File System for Windowsの作り方

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15回 mackerel-agentのカスタムメトリックプラグインを書いてみよう

Mackerelの監視エージェントである「mackerel-agent」はプラグインによる拡張ができます。プラグインは公式で提供しているものもありますが、もちろん自分で書くこともできます。今月と来月の2回で、mackerel-agentのプラグインの書き方について説明します。

プラグインは2種類

mackerel-agentのプラグインは、カスタムメトリックプラグインとチェックプラグインの2種類に分類されます。

カスタムメトリックプラグインはホストのメトリックを取得するメトリック監視用のプラグインであり、チェックプラグインはチェック監視用のプラグインです。

プラグインはいずれも、単なる実行可能なファイルやスクリプトです。プラグインはmackerel-agentから1分おきに実行され、その標準出力や終了ステータスがプラグインの実行結果として利用されます。公式のプラグインはGo言語で書かれていますが、BashやPerlやPythonやRubyなど、あらゆる言語で記述ができます。シンプルでUNIX的な考え方に即していると言えるでしょう。この考え方はMackerelオリジナルのものではなく、多くの監視ツールで採用されてきたもので、mackerel-agentのプラグインはそれらのツールのプラグインとも互換性があります。実際、カスタムメトリックプラグインはSensuのメトリックプラグインと互換性があり、チェックプラグインはNagiosやSensuのチェックプラグインと互換性があります。

カスタムメトリックプラグインの作り方

今回は、カスタムメトリックプラグインの仕様と作り方について解説していきます。

カスタムメトリックプラグインの仕様

カスタムメトリックプラグインは、標準出力の各行に次のフォーマットの出力をすることが期待されます(\tはタブ文字)。

```
{metric name}\t{metric value}\t{epoch seconds}
```

詳細

- 名前の最後のドットまでが共通するメトリックがMackerel上で1つのグラフにまとめられ、ホスト詳細で閲覧できる
- mackerel-agentにより、メトリック名の先頭には自動的に"custom."という文字列が付与される
- メトリック名に使える文字は英数字もしくはハイフン(-)、アンダースコア(_)、ドット(.)のいずれか(/[-a-zA-Z0-9_.]/)

たとえば、example.fooとexample.barという名前のメトリックを投稿した場合、custom.example.*と名付けられたグラフが、ホスト詳細に現れます。このグラフにはexample.fooとexample.barの系列データが描画されます。

シェルスクリプトによる例

たとえば、1〜6のランダムな数値を返すカスタムメトリックプラグインは、**リスト1**のようにたった1行で書けます。実際にこのスクリプトを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
random.dice     2       1458477767
```

この場合、2の目が出ていることになります。これだけでもれっきとしたプラグインと言えます。

グラフ定義の指定(任意)

Mackerelのカスタムメトリックプラグイン独自の追加仕様として、任意でグラフ定義を指定できる機能があります。これは、投稿した時系列データをMackerel上でどのように表示したいかをJSONで指定するものです。これによって、カスタムメトリックの表示設定をWeb上で行わずに、あらかじめ指定しておけます。

mackerel-agentは起動時に、MACKEREL_AGENT_PLUGIN_META環境変数を「1」に設定した状態でプラグインを実行し、その出力をグラフ定義として利用します。出力の1行目は、#mackerel-agent-pluginである必要があります。最初の行がこの内容ではなかった場合、mackerel-agentはこのプラグインがグラフ定義を出力しないものとして、グラフ定義の設定を行いません。2行目以降に続けてJSONフォーマットでグラフ定義を出力します。プラグインのグラフ定義出力は**リスト2**のようになります。それぞれの項目は、**表1**のような意味を持ちます。**表1**におけるメトリック定義は、**表2**のようなキーを持ちます。

▼リスト1　1〜6のランダムな数値を返すカスタムメトリックプラグイン

```
#!/bin/sh
echo "random.dice\t$(($RANDOM%6) + 1))\t$(date +%s)"
```

▼リスト2　プラグインのグラフ定義出力の例

```
# mackerel-agent-plugin
{
  "graphs": {
    {graph}: {
      "label": GRAPH_LABEL,
      "unit": UNIT_TYPE
      "metrics": [
        {
          "name": METRIC_NAME,
          "label": METRIC_LABEL
        },
        ...
      ]
    },
    GRAPH_NAME: ...
  }
}
```

メトリック名のワイルドカード

あるメトリックの階層に不特定のキー名でメトリックが出力され、それらをMackerelに投稿したいといった場合、ワイルドカードを用いてグルーピングできます。

ワイルドカード(*または#)は2つのドット(.)

▼表1　グラフ定義における各要素

項目	説明
graphs.{graph}.label	ユーザ定義メトリック{graph}.*に対応するグラフの表示名。{graph}にはドット(.)を含むことができる
graphs.{graph}.unit	ユーザ定義メトリック{graph}.*に対応するグラフの値の種類。可能な値は"float", "integer", "percentage", "bytes", "bytes/sec", "iops"のいずれか
graphs.{graph}.metrics	ユーザ定義メトリック{graph}.*に対応するメトリック定義の配列

▼表2　メトリック定義におけるキー

キー	説明
name	このメトリックがユーザ定義メトリック{graph}.{name}に対応することを表す。この値にドット(.)を含むことはできない。使用できる文字は英数字もしくはハイフン(-)、アンダースコア(_)のいずれか(/[-a-zA-Z0-9_]/)。また、ワイルドカード#, *を使用することもできる
labal	ユーザ定義メトリック{graph}.{name}に対応する時系列の表示名
stacked	ユーザ定義メトリック{graph}.{name}に対応する時系列を積み上げ表示するかどうか。たとえばfalseなら線分で表示する

▼図1　6つの項目をワイルドカードでグルーピング

の間、または最後のドット(.)の後ろに単独で使用できます。先頭には使えません。ワイルドカードはドットを除く文字の連続[-a-zA-Z0-9_]+にマッチします。

ワイルドカード#は1つまでしか使えません。#を使った場合は、メトリック名の#にマッチした部分でグラフの凡例がグループ化されます。たとえば、custom.docker.cpu.#.user、custom.docker.cpu.#.systemという2つの定義があり、

```
- custom.docker.cpu.f5240a.user
- custom.docker.cpu.f5240a.system
- custom.docker.cpu.e866aq.user
- custom.docker.cpu.e866aq.system
- custom.docker.cpu.e552ad.user
- custom.docker.cpu.e552ad.system
```

のように6つのカスタムメトリックを送信した場合の凡例は**図1**のようになります。ちなみに、ワイルドカードを含むグラフ定義のカスタムメトリックは一定時間(およそ6〜8時間以上)送信がない場合、自動的に削除されます。

グラフ定義の出力例

公式プラグインのmackerel-agent-uptimeのグラフ定義は**図2**のように出力させることができます。手元で出力を確認するために、MACKEREL_AGENT_PLUGIN_META=1を付けてプラグインを実行しています(実際には2行以降のJSONの部分は1行で表示されます)。

公式のヘルパーライブラリを使う

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。公式プラグインでも利用しているGo言語でのカスタムメトリックプラグイン作成用のヘルパーライブラリである、**go-mackerel-plugin-helper**[注1]を利用したプラグインの開発方法を説明します。Go言語の基本的な文法を理解していることを前提として解説しますが、それほど難しい構文は出てきませんのでご安心ください。

▼図2　mackerel-agent-uptimeのグラフ定義を出力

```
% MACKEREL_AGENT_PLUGIN_META=1 ./mackerel-plugin-uptime
# mackerel-agent-plugin
{
  "graphs": {
    "uptime": {
      "label": "Uptime",
      "unit": "float",
      "metrics": [
        {
          "name": "seconds",
          "label": "Seconds",
          "type": "",
          "stacked": false,
          "scale": 0
        }
      ]
    }
  }
}
```

このヘルパーの作法に従って開発することで、グラフ定義の出力や前回取得した値との差分値計算などを簡単に行うことができます。またこのヘルパーを使えば、自ずと公式プラグインと同じ作法でプラグインを書くことになるので、公式プラグインとして採用される可能性もあります。

go-mackerel-plugin-helper利用時の構成

go-mackerel-plugin-helperを利用した場合、プラグインのソースコードは次の5つの部分で構成されます。

❶package宣言とimport文
❷プラグイン用structの定義
❸グラフ定義出力メソッドGraphDefinitionをstructに定義
❹メトリック取得用メソッドFetchMetricsをstructに定義

注1) URL https://github.com/mackerelio/go-mackerel-plugin-helper

❺main()関数の定義

ここではmackerel-agent-uptime[注2]を例にとって、それぞれ見ていきましょう。

❶package宣言とimport文(リスト3)

packageはmainで宣言します。また、go-mackerel-plugin-helperではmpというエイリアスでインポートすることが慣例となっています。

❷プラグイン用structの定義(リスト4)

プラグイン用のstructを定義しています。このstructにはPrefixというフィールドが定義されています。これは、グラフ定義出力時に、そのメトリックの名前空間の先頭を決めるためのものです。uptimeプラグインの標準ではPrefixはuptimeであり、uptime.secondsというキーでメトリックを出力しますが、この中のuptimeを、たとえばuptime2に変更したいといった場合に利用するフィールドです。

uptimeプラグインではPrefixフィールドはとくに有益ではありません。ただ、たとえばミドルウェア用のプラグインの場合、1台のホストの中で同じミドルウェアを複数起動してそれぞれのメトリックを取得したい場合に、メトリックの名前空間を分ける必要が出てくるので、このフィールドを定義しておくことが推奨されています。

uptimeプラグインでは、このPrefixフィールドのみが定義されていますが、一般的なミドルウェアのプラグインであれば、PortやHostといったフィールドも必要になるでしょう。

また、このプラグイン用structはmp.Pluginのinterfaceを満たす必要があります。interfaceの定義はリスト5のようになっています。これらが、グラフ定義出力メソッドとメトリック取得用メソッドです。

❸グラフ定義出力メソッドGraphDefinitionをstructに定義(リスト6)

この、GraphDefinitionを定義することで、グラフ定義のJSONやメトリックが正しく出力されるようになります。uptimeプラグインでは、u.Prefixをキーとした1つのグラフ定義しか返していませんが、多くのプラグインはu.Prefix + "runtime"、u.Prefix + "memory"といったよう

▼リスト3　package宣言とimport文

```
package main

import (
    "flag"
    "fmt"
    "strings"

    mp "github.com/mackerelio/go-mackerel-plugin-helper"
    "github.com/mackerelio/golib/uptime"
)
```

▼リスト4　プラグイン用structの定義

```
    type UptimePlugin struct {
        Prefix string
    }
```

▼リスト5　interfaceの定義

```
    type Plugin interface {
        GraphDefinition() map[string]Graphs
        FetchMetrics() (map[string]interface{}, error)
    }
```

▼リスト6　グラフ定義出力メソッドGraphDefinitionをstructに定義

```
func (u UptimePlugin) GraphDefinition() map[string](mp.Graphs) {
    labelPrefix := strings.Title(u.Prefix)
    return map[string](mp.Graphs){
        u.Prefix: mp.Graphs{
            Label: labelPrefix,
            Unit:  "float",
            Metrics: [](mp.Metrics){
                mp.Metrics{Name: "seconds", Label: "Seconds"},
            },
        },
    }
}
```

注2) URL https://github.com/mackerelio/mackerel-agent-plugins/tree/master/mackerel-plugin-uptime

▼表3　mp.Metricsに指定できるフィールド

フィールド	型	説明
Name	string	必須事項。メトリックの名前。FetchMetrics()で取得するmapのキー名と対応する
Label	string	Mackerel上での表示名
Diff	bool	plugin上で差分値計算をするかどうか(Default：false)
Type	strint	"float64", "uint32"もしくは"uint64"。おもに整数値のカウンターでDiff計算が必要な場合に、上限値の決定のために用いられる(Default：float64)
Stacked	bool	Mackerel上で積み上げ表示されるかどうか(Default：false)
Scale	float64	指定された場合、取得した値にこのScaleの値を乗じてから出力を行う。たとえば、KBで取得した値をByteに補正したい場合は「1024」を指定する

なキーで複数のグラフ定義を返します。

Labelは Mackerel上で表示されるグラフ名、Unitはグラフの単位で、グラフ定義API同様に"float", "integer", "percentage", "bytes", "bytes/sec", "iops"が指定可能となっています。

Metricsにはそのグラフ内に描画する複数のMetrics定義を指定します。mp.Metricsに指定できるフィールドは**表3**のとおりです。

❹メトリック取得用メソッドFetchMetricsをstructに定義(リスト7)

Fetchmetrics()は map[string]interface{}の形式で値を返します。interface{}になってますが、実際は何らかの数値となります。uint64とfloat64を統一的に扱うために、このようになっています。ここでは、secondsをキーにuptimeの値が格納されたmapを返しています。

❺main()関数の定義(リスト8)

プラグインのメインの処理です。コマンドラインオプションのパース、プラグインヘルパーオブジェクトの作成、そしてプラグインの実行を行っています。

helperに指定されているTempfileは、差分値計算用に前回取得した値を保持しておくためのファイルです。ほかのプラグインや、同じプラグインであっても引数が異なる複数の設定がある場合などに重複しないように指定する必要があるので、気を付けてください。

おわりに

これで、mackerel-agentのカスタムメトリックプラグインの構成を一通り説明しました。みなさんも実際に作ってみて、動作をたしかめてみてください。

今回は、おもにカスタムメトリックプラグインの作成方法について解説しました。次回はチェックプラグインの作成方法などについて解説していきます。SD

▼リスト7　メトリック取得用メソッドFetchMetricsをstructに定義

```
func (u UptimePlugin) FetchMetrics() (map[string]interface{}, error) {
    ut, err := uptime.Get()
    if err != nil {
        return nil, fmt.Errorf("Faild to fetch uptime metrics: %s", err)
    }
    return map[string]interface{}{"seconds": ut}, nil
}
```

▼リスト8　main()関数の定義

```
func main() {
    optPrefix := flag.String("metric-key-prefix", "uptime", "Metric key prefix")
    optTempfile := flag.String("tempfile", "", "Temp file name")
    flag.Parse()

    u := UptimePlugin{
        Prefix: *optPrefix,
    }
    helper := mp.NewMackerelPlugin(u)
    helper.Tempfile = *optTempfile
    if helper.Tempfile == "" {
        helper.Tempfile = fmt.Sprintf("/tmp/mackerel-plugin-%s", *optPrefix)
    }
    helper.Run()
}
```

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第27回 宮田さん

犯罪者プロファイリングならばご飯三杯いけるくつな先生に、愛の未解決事件を！

to be Continued

システムに問題が発生した場合、ログなどを見て状況を把握したあと、発生した原因などを推測・検証し、再発しないための対策を実施して運用再開につなげます。発生原因などが不明な場合や新手の攻撃方法が使われた場合などは推測・検証が難しくなりますが、こういうときには先人が実は良きアドバイザーになりますよ。先人のアドバイスには踏んだ場数や経験がこもっています。相談は大事ですよ。先人がまわりにいないなら書籍に頼りましょう。Webもいいですが、書籍も先人の経験と叡智の集合と言えます。このタイミングなら言えそうだ！　これからLinuxを使う新人さんは『改訂3版Linuxコマンドポケットリファレンス』が超便利だそうですよ（ステマ）。

Sphinxで始めるドキュメント作成術

清水川 貴之 SHIMIZUKAWA Takayuki Twitter @shimizukawa

第14回 Sphinxで楽々ドキュメント翻訳

翻訳もSphinxを使って

オープンソースソフトウェア(OSS)のライブラリやツールを使っていると、多くの英語ドキュメントに触れる機会があります。日本人であれば、日本語で読めたほうが理解が早いので、翻訳されたドキュメントがあれば楽なのに……と、筆者もいつも思ってしまいます。

そこで今回は、ドキュメントを翻訳する方法の1つとして、Sphinxの国際化機能を紹介します。翻訳そのものは人間が行う必要がありますが、翻訳を進めるうえでSphinxを使うと楽になること、うれしいこと、について紹介します。

Sphinxをインストールしていない場合は、Sphinxユーザ会の手順[注1]を参照してインストールしておいてください。

翻訳しよう

日本語に翻訳されたドキュメントが必要になるのは、次のような場合だと思います。

- 自分のため：英語のドキュメントをそのまま読んでも、正確な意図が読み取りづらい
- みんなのため：多様な人に使ってもらうには、英語では読んでもらえない。日本語が必須

これ以外にもいろいろな動機があると思いますが、「よっしゃ、自分が翻訳しよう！」と思い立って翻訳を始めてみても、「勝手に翻訳して公開しても良いんだろうか？」「みんなはどうやって翻訳してるんだろう？」といった心配も出てきます。

OSSのドキュメントは多くの場合、ソースコードと同様にオープンなライセンスが適用されています。ライセンスはGPL、BSD、MIT、Apacheなどがよく採用されていますが、このようなオープンなライセンスであれば、翻訳して公開するのに作者の許可などは必要なく、翻訳した文章を自由に公開して大丈夫です。翻訳しようとしているドキュメントがどのようなライセンスで公開されているのか確認しておきましょう。また公開したら、原作者に連絡するととても喜ばれます。

本文書き換え翻訳の良いところ、つらいところ

ドキュメントを翻訳する手順として一番始めやすい方法は、原文のパラグラフとその翻訳が対になるように、原文の直後に翻訳文を書き込んでいく方法です。翻訳が完了した原文を順次コメントアウトしていけば、ドキュメントが徐々に翻訳された文章に置き換わっていく、という流れです。原文がWebサイトで公開されている場合には、ブラウザの機能でHTMLを保存して手元で書き換えてしまうという方法が、「とりあえず翻訳したら、すぐにその結果を見たい」というときには良さそうです(**リスト1**)。

この方法は、ブラウザとエディタだけあれば

注1) http://sphinx-users.jp/gettingstarted/index.html

手軽に始められるうえに、元のドキュメントと同じ体裁で結果を確認できます。ただ、HTMLタグだらけなので、翻訳したいテキストを判別するのに手間がかかりそうですね。

原文がreStructuredText(以下、reST)やMarkdownの場合も、同じアプローチで翻訳できます。元のドキュメントと同じ体裁で確認するためには、ドキュメントソースをHTMLなどに変換してください(**リスト2**)。

HTML書き換えと、ソース書き換え、どちらの方法も手軽に始められますが、このように書き換えて翻訳していく方法は、あとあとの手間が増える原因になります。とくに、OSSのドキュメントのように原文がどんどん更新されていく場合、追従する手間が大きいのです。その結果、更新が面倒になって放置されてしまったり、手間のかかる作業を誰かに引き継ぐのもたいへん、ということになってしまいます。また、複数の言語に翻訳したい場合、各言語の翻訳者がそのようなたいへんな作業をそれぞれで行うのは時間の無駄です。

これに対して、国際化のための共通フォーマットを使う方法であれば、原文への追従がしやすく、共同作業や引き継ぎはスムーズに行えます。そのような共通のフォーマットとして有名なものの1つに、GNU gettext[注2]で扱う"メッセージカタログ"形式があります。メッセージカタログは、msgid(一意なID)とmsgstr(表示するメッセージ)の集まりを記録したファイルです。Sphinxはこのフォーマットを使って、msgidに原文の各パラグラフの文章を割り当て、msgstrを翻訳文として利用します(**リスト3**)。

注2) https://www.gnu.org/software/gettext/

▼リスト1　Requestsライブラリ(後述)のHTMLをコメントアウト方式で翻訳する例

```
<!--
<h1>Requests: HTTP for Humans<a class="headerlink" ...(中略)
-->
<h1>Requests: 人間のためのHTTP<a class="headerlink" ...(中略)

<!--
<p>Release v2.9.1. (<a class="reference internal"
href="user/install/#install"><span>Installation</span></a>)</p>
-->
<p>リリース v2.9.1. (<a class="reference internal"
href="user/install/#install"><span>インストール</span></a>)</p>
```

▼リスト2　Requestsライブラリのindex.rstをコメントアウト方式で翻訳する例

```
..
   Requests: HTTP for Humans
   =========================

Requests: 人間のためのHTTP
==========================

.. Release v¥ |version|. (:ref:`Installation <install>`)

リリース v¥ |version|. (:ref:`インストール <install>`)
```

▼リスト3　gettextのメッセージカタログ(.po)の例

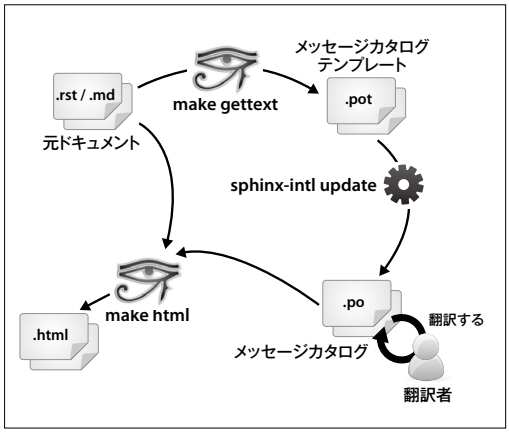
```
msgid "Requests: HTTP for Humans"
msgstr "Requests: 人間のためのHTTP"

msgid "Release v¥¥ |version|. (:ref:`Installation <install>`)"
msgstr "リリース v¥¥ |version|. (:ref:`インストール <install>`)"
```

Sphinxの国際化機能(**図1**)は、Sphinxドキュメントからgettextのメッセージカタログテンプレートファイル(拡張子.pot)を生成する機能と、翻訳文が書かれたメッセージカタログファイル(拡張子.po)を取り込んでドキュメントを生成する機能の2つで構成されています。

図2はSphinx国際化機能を使った翻訳の流れです。はじめにドキュメントの原文から.potファイルを作成し、このテンプレートから各言語の翻訳文を書き込むための.poファイルを用意して、.poファイルに翻訳文を書き込んでいきます。.poファイルを用意する作業は、サポートツールsphinx-intlで簡単に行えます。翻訳者は、翻訳が適用されたHTMLをブラウザで確認しながら翻訳を進め、原文に更新があれば、.poファイルに差分を適用して再度翻訳を進めます。

▼図1　Sphinxの国際化機能



ドキュメント翻訳クイックスタート

それでは、Sphinxを使ってドキュメントの翻訳を行ってみましょう。今回は、実際にOSSライブラリのドキュメントを翻訳する例として、Pythonの「Requests」[注3]のドキュメントを使用します[注4]。

Requestsのドキュメントをmake html

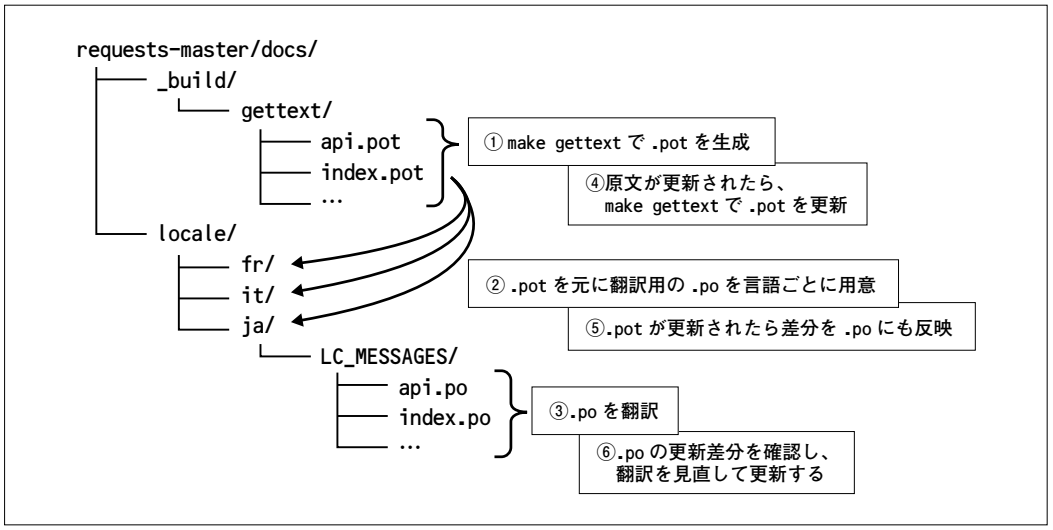
Requestsのサイト(注3参照)のDownload ZIPボタンからソースコードをダウンロードして展開してください。取得したディレクトリ「requests-master」には、「docs」というSphinxプロジェクトのディレクトリがあります(**図3**)。

コマンドラインでdocsディレクトリに移動し、make htmlを実行してHTMLを生成してみましょう(**図4**)。

生成されたHTMLは「_build/html」ディレク

注3) https://github.com/kennethreitz/requests
注4) Sphinxの国際化機能は、Python用ライブラリのドキュメントでなくても使用できます。

▼図2　Sphinxドキュメントとメッセージカタログによる翻訳の流れ



トリ以下に出力されます。「_build/html/index.html」をブラウザで開いてみてください。**図5**のように、Requestsの公式サイト[注5]と同じ内容が表示されました。

それでは、このドキュメントを国際化していきましょう。

国際化機能の設定と補助ツールのインストール

翻訳文を書き込んだ.poファイルを置くディレクトリの場所と構造をSphinxプロジェクトの設定ファイル「conf.py」に指定します。次の内容をファイルの末尾に追加してください。

```
locale_dirs = ['./locale']
  ↑.poファイルを置くディレクトリ
```

これで、.poファイルはconf.pyからの相対パスでlocaleディレクトリ以下から読み込まれます。

次に、国際化機能の補助ツールsphinx-intl[注6]を次のようにインストールします。

```
$ pip install sphinx-intl
```

これで国際化機能を使う準備ができました。

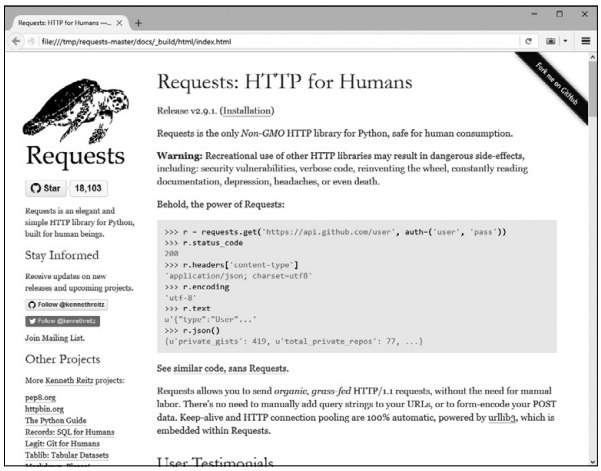
メッセージカタログの準備

それでは、翻訳を行うためのメッセージカタログ(.po)ファイルを作成しましょう。ここで使用するコマンドは次の2つです。

▼図3 Requests用のSphinxプロジェクトディレクトリ

```
requests-master/
├── docs/                  ……Sphinx プロジェクトのディレクトリ
│   ├── _static/           ……ロゴや css などを格納
│   ├── _templates/        ……カスタム HTML テンプレートを格納
│   ├── _themes/           ……HTML テーマディレクトリ
│   │   ⋮                  …….rst を含むいくつかのディレクトリ
│   ├── api.rst            ……api ページ
│   ├── conf.py            ……Sphinx プロジェクトの設定ファイル
│   ├── index.rst          ……トップページ
│   ├── make.bat           ……Windows 用 make コマンド
│   └── Makefile           ……Linux/Mac 用 Makefile
```

▼図5 Requestsのドキュメントから生成したページ



▼図4 make html

```
$ make html
Running Sphinx v1.3.6
making output directory...
loading pickled environment... not yet created
loading intersphinx inventory from http://urllib3.readthe...（中略）
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 18 source files that are out of date
（..中略..）
dumping search index in English (code: en) ... done
dumping object inventory... done
build succeeded.

Build finished. The HTML pages are in _build/html.
```

注5) http://docs.python-requests.org/en/master/

注6) https://pypi.python.org/pypi/sphinx-intl

- make gettextで、メッセージカタログテンプレート(.pot)を生成
- sphinx-intl updateで、.potからメッセージカタログ(.po)を生成

はじめにSphinxのmake gettextでメッセージカタログテンプレート(.pot)ファイルを生成します(**図6**)。実行が完了すると、**図7**のように「_build/locale」ディレクトリ以下に.potファイルが生成されます[注7]。

注7) 出力先はMakefile(またはmake.bat)に書かれているため、異なる場合があります。

▼図6　make gettext

```
$ make gettext
Running Sphinx v1.3.6
making output directory...
loading pickled environment... not yet created
loading intersphinx inventory from http://urllib3...（中略）
building [gettext]: targets for 3 template files
（..中略..）
writing message catalogs... [100%] dev/philosophy
build succeeded.

Build finished. The message catalogs are in _build/locale.
```

▼図7　生成されたgettextのメッセージカタログテンプレートファイル

```
requests-master/docs/
└── _build/
    ├── locale/              ……gettext ビルダーの出力ディレクトリ
    │   ├── ⋮                ……いくつかのディレクトリと *.pot ファイル
    │   ├── api.pot          ……api.rst ドキュメントのメッセージカタログテンプレート
    │   └── index.pot        ……index.rst ドキュメントのメッセージカタログテンプレート
    └── html/
```

▼リスト4　index.pot

```
#: ../../index.rst:7
msgid "Requests: HTTP for Humans"
msgstr ""

#: ../../index.rst:9
msgid "Release v¥¥ |version|. (:ref:`Installation <install>`)"
msgstr ""
```

生成された.potファイルの内容を見てみましょう。**リスト4**はindex.rstから抽出されたindex.potの内容です。msgidとmsgstrがパラグラフごとにペアで出力されていますね。この.pot

COLUMN

MarkdownでSphinxの国際化機能を利用する

Sphinxは、ドキュメントソースのフォーマットとしてreSTとMarkdownのどちらでも利用できます。Markdownの場合、国際化機能に一部制約があります。

リンク先URLがパラグラフ外で定義されている場合、国際化機能を使うとリンクされないため、翻訳文側でパラグラフ内にURLを書いてあげる、という一時的な回避策で対処しなければなりません(**リストA**)。将来的にはこの制約も解消していきたいと考えています。

▼リストA　翻訳文でリンク先URLを明示する

```
msgid "CommonMark [spec][the spec] and implementations."
msgstr "CommonMark [仕様](http://spec.commonmark.org/) と実装。"
```

ファイルはテンプレートですので、このファイルのmsgstrに翻訳文を書き込んでいってはいけません。もう少しお待ちください。

翻訳を書き込むための.poファイルをsphinx-intl updateコマンドで生成します(**図8**)。

sphinx-intl updateを実行すると、.potファイル群をもとに、日本語用の.poファイル群が**図9**のようなディレクトリ構造で作成されます。

これでメッセージカタログが準備できました。それでは実際に.poファイルを書き換えて翻訳を進めていきましょう。

翻訳してビルド

index.poには、ドキュメント内のパラグラフごとにmsgidとmsgstrのセットが生成されています。翻訳する際はmsgidと対になるように、msgstrに翻訳文を書き込んでいきます。この際、msgidの行は変更する必要はありません。また、強調やロールなど、reSTによるマークアップが行われている場合は、そのマークアップを維持したまま翻訳を進めると良いでしょう(**リスト5**)。

それでは、翻訳した.poを適用したHTMLを生成しましょう。localeディレクトリには、言語ごとに別々のディレクトリでメッセージカタログを用意できるので、ドキュメント生成時にはどの言語でビルドしたいのかを指定する必要があります。そのために、conf.pyの末尾に、次のように表示言語指定を追加してください。

```
language = 'ja'
```

これで、再度make htmlコマンドを実行すれば、「locale/ja」ディレクトリ以下の.poファイルが自動的に適用され、翻訳されたHTMLが生成されます(**図10**)。

▼図8　sphinx-intl updateで日本語用の.poファイルを生成

```
$ sphinx-intl update -l ja
Create: ./locale/ja/LC_MESSAGES/api.po
Create: ./locale/ja/LC_MESSAGES/index.po
Create: ./locale/ja/LC_MESSAGES/community/faq.po
Create: ... (以下略)
```

▼図10　翻訳を適用したindexページ

▼図9　国際化ディレクトリの構成

```
requests-master/docs/
└── locale/                          ……国際化の言語ごとのカタログを置くディレクトリ
    └── ja                           ……日本語のディレクトリ
        └── LC_MESSAGES              ……国際化のメッセージを置くディレクトリ
            ├── ⋮                    ……いくつかのディレクトリと*.poファイル
            ├── api.po               ……api.rstドキュメントのメッセージカタログ
            └── index.po             ……index.rstドキュメントのメッセージカタログ
```

▼リスト5　index.poの翻訳例

```
#: ../../index.rst:7
msgid "Requests: HTTP for Humans"
msgstr "Requests: 人間のためのHTTP"

#: ../../index.rst:9
msgid "Release v¥¥ |version|. (:ref:`Installation <install>`)"
msgstr "リリース v¥¥ |version|. (:ref:`インストール <install>`)"
```

元ソースが更新されたら

翻訳中に、ドキュメントの原文が更新されることがあります。Sphinxの国際化機能を利用している場合、手間をかけずに原文の更新に追従できます。

まず、元のドキュメントファイルを更新します。たとえば「docs/index.rst」が更新されたとします(ここでは説明のために、文中のfor Humansをfor Catsに変更しました)。次に**図11**のようにコマンドを実行します。これによって、ドキュメントソースから.potファイルが再生成され、.potファイルをもとに.poファイルが更新されます。

原文が更新されると、**図11**のように各ファイルの更新状況と更新行数が表示されます。この例では、index.po +1, -1となっているので、index.poの翻訳メッセージが1行減って1行増えたようです。

そこでindex.poファイルを見てみると、**リスト6**のようにfuzzyとなっている行が見つかります。これは、原文の更新によってこの翻訳メッ

▼リスト6　更新されたindex.po

```
#: ../../index.rst:7
#, fuzzy
msgid "Requests: HTTP for Cats"
msgstr "Requests: 人間のためのHTTP"
```

▼図11　sphinx-intl updateで.poを更新

```
$ make gettext   ←.potファイルを再生成
(..以下略..)

$ sphinx-intl update                                  ←言語無指定ですべての既存カタログを更新
Not Changed: ./locale/ja/LC_MESSAGES/api.po           ←カタログ変更なし
Update: ./locale/ja/LC_MESSAGES/index.po +1, -1       ←カタログが1行増減している
Not Changed: ...(以下略)
```

COLUMN

conf.pyを編集しない方法

今回の例では、conf.pyに3つの行を追加しています。しかし、ソースコードの所有者が自分でない場合、このようにconf.pyを書き換えてしまうと管理しづらくなってしまいます。そこで、conf.pyを編集せずにコマンドラインから設定値を指定する方法を紹介します。

SphinxプロジェクトのMakefile(またはmake.bat)には、Sphinxの実行オプションを設定するための変数が用意されています。この変数にコマンドラインから値を指定して、conf.pyの設定を**図A**のように変更してください。

sphinx-intlもconf.pyを参照しています。ここでは環境変数での指定方法を紹介します。UNIX系ではexport、Windowsではsetで指定してください。

```
UNIX系の場合
$ export SPHINXINTL_LOCALE_DIR=./locale
$ export SPHINXINTL_POT_DIR=./_build/locale
$ sphinx-intl update -l ja
```

▼図A　コマンドラインから設定値を指定する

```
UNIX系の場合
$ make html SPHINXOPTS='-D language=ja -D gettext_compact=0 -D locale_dirs=./locale'

Windowsの場合
> set SPHINXOPTS=-D language=ja -D gettext_compact=0 -D locale_dirs=./locale
> make html
```

セージの原文(msgid)が変更になったけれど、翻訳文(msgstr)が正しい状態かどうかわからない、ということを示しています。

原文を見直してみて、翻訳に影響がある変更の場合は、msgstrを修正してください。修正してもしなくても、確認が終わったらfuzzyと書かれた1行を削除してください。翻訳メッセージにfuzzyが付くのは、誤字の修正や、複数形のsを変更するような文法上の修正、URLのちょっとした修正などです。それらの中には変更に気づきにくい場合もあります。

なお、原文が半分近く修正された場合、fuzzyマークは付かず、新しいmsgidとして追加されます。

翻訳がどのくらいできているかを確認するには、sphinx-intl statを実行します(**図12**)。fuzzyがいくつあるかもこのコマンドで確認できます。

注意点

Sphinxの国際化機能ではできないこともあります。パラグラフやコラムの追加といった、原文にない文章構造の追加はできません。これは、原文と1対1で翻訳文を用意するしくみでは実現できない機能です。

画像ファイルを言語ごとに差し替える機能はSphinx-1.4から利用できます[注8]。

gettext関連の設定

Sphinxのconf.pyには、ここまでに紹介した以外に、gettext関連の設定もあります。ここで少し紹介します[注9]。

今回の例のRequestsのドキュメントは、devディレクトリ以下に4つの.rstファイルを持っています。しかし、メッセージカタログファイルとしては、devディレクトリ以下のすべての文章がdev.potにまとめられます。gettext_compactオプションをFalseに設定すれば、ドキュメントソースがサブディレクトリを持つ場合に1ファイルに集約せず、ソースと同じディレクトリ構造を維持します。

ほかにも、gettext_additional_targetsオプションでソースコードサンプル内のコメントや画像の代替文字列を翻訳するかどうかを指定できます。翻訳の対象にするには次のようにconf.pyに設定します。

```
gettext_additional_targets = ['literal-block']
```

次回予告

今回は、Sphinxの国際化機能について紹介しました。.poファイルの翻訳支援ツール、サービスを併用すれば、さらに煩雑な手作業から解放されてドキュメント翻訳そのものに集中できると思います。

次回は、Sphinxの国際化機能を使った翻訳の手順を自動化して、複数人で翻訳を行う方法を紹介します。SD

注8) http://www.sphinx-doc.org/ja/master/config.html のfigure_language_filenameの説明を参照。

注9) 各オプションについての詳細は、http://www.sphinx-doc.org/ja/stable/config.htmlを参照。

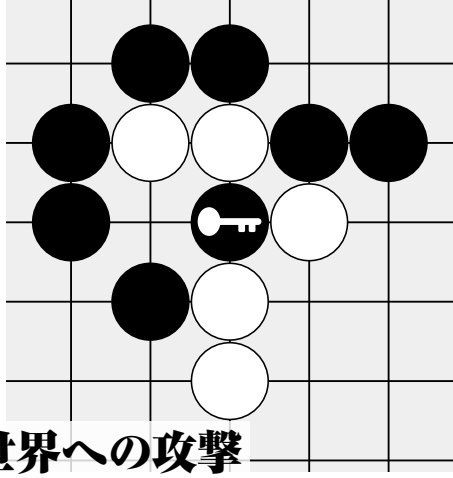
▼図12 sphinx-intl stat

```
$ sphinx-intl stat
./locale/ja/LC_MESSAGES/api.po: 198 translated, 0 fuzzy, 65 untranslated.
./locale/ja/LC_MESSAGES/index.po: 7 translated, 1 fuzzy, 38 untranslated.
(..以下略..)
```

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



【第二二回】BlackEnergyによるリアルな世界への攻撃

サイバー攻撃で社会インフラが物理的な被害を受けるということは、フィクションで描かれることはあっても、現実の世界ではそう起こるものではありません。しかし、2015年末に、マルウェアの攻撃により電力システムが麻痺し、大規模な停電を引き起こした事件が発生しました。今回は、その事件の背景を取り上げます。

物理的な被害をもたらしたマルウェア

これまでマルウェアによる攻撃は情報を盗む、あるいは情報を破壊するといった、情報という形のないものへの攻撃でした。しかし、2015年12月23日、ウクライナの電力会社がマルウェアを使った攻撃にさらされ、地域への電力供給が停止し、140万人の地域住民の半分が停電被害にあうという事態が起こりました。

変電所を運用するシステムにマルウェア「BlackEnergy」が感染したために、送電が停止し(**図1**)、イヴァーノ＝フランキーウシク地域の住民世帯の半分へ電力供給が停止、停電しました[注2]。電力会社のエンジニアが変電所を手動で操作し、電力を復旧させるという手順をふみました。

ウクライナだけの問題ではない

BlackEnergyはトロイの木馬タイプのマルウェアです。侵入後、C&Cサーバ(Command and Control Server)と通信し、目的に合わせたマルウェアを呼び込むリモート制御の役割を果たします。ボットネットを構築するために、2007年ごろから使われていることが知られています。また、2011年当時の報道[注3]で、ロシアのサイバー犯罪市場で生み出され

◆**図1　停電がマルウェアBlackEnergyによるものであることを伝える報道**[注1]

注1) First known hacker-caused power outage signals troubling escalation (Jan 5, 2016)
http://arstechnica.com/security/2016/01/first-known-hacker-caused-power-outage-signals-troubling-escalation/
注2) Ukraine to probe suspected Russian cyber attack on grid (Dec 31, 2015)
http://www.reuters.com/article/us-ukraine-crisis-malware-idUSKBN0UE0ZZ20151231
注3) 進化するDDoSボットネット 第1回：BlackEnergyボット(2011年5月20日) http://blogs.mcafee.jp/mcafeeblog/2011/05/1215.html

ていた可能性があると指摘されています。

工業生産現場に導入されている制御システムは、広くはICS（Industrial Control System）と呼ばれますが、そのICSを対象としたマルウェアとしても使われています。

とくに2011年以降は攻撃が顕著となり、米国の制御向けCERTであるICS-CERTからも繰り返し警告が出ています。今回のウクライナの攻撃にもBlackEnergyが使われたため、再度情報をアップデートした形で警告が出ています[注4]。細かいことを言うと、ICS-CERTのドキュメントには「攻撃」という意味よりさらに軍事色の強い「軍事行動」という意味の“Campaign”という単語が使われています。

ウクライナの事例は、電力網という社会インフラへの攻撃が成功するという深刻な事態であったため、米国のCERTチームであるICS-CERTとUS-CERTが、Ukrainian CERTと組んで共同で解析を行っています。

これまでの連載で何度も取り上げてきましたが、ICS-CERTやUS-CERTといった組織は、米国の国家安全保障の観点から作られた米国国土安全保障省傘下のCERTチームです。このようなCERTチームが前面に出てきているということは、ウクライナの事案は非常に深刻であり、将来において重要な意味を持つものであるという視点で見なければいけないでしょう。

BlackEnergyは、人間が使うコンソールであるHMI（Human-Machine Interface）システムの部分に影響を与える形になっています。今回のウクライナの場合は、HMIは外部とインターネットで接続していることを条件としていると報告されています。BlackEnergyも世代が進み、ウクライナで使われたものはBlackEnergy 3と呼ばれています。

通常のPCと変わらない

1世代前のBlackEnergy 2は、標的型攻撃としてMicrosoft Wordのアタッチとして送られてきて、感染するようになっていました。そのような攻撃をしかけてくるということは、プラントなどの制御用端末として使っているPCは、「インターネットに接続されており、Microsoft Officeも同時に扱えるような通常のデスクトップPCと変わらない構成のものを、HMIのコンソールとして使っている」と想定しているようです。それがメール経由なのかブラウザ経由なのかはわかりませんが、そこからBlackEnergy 2が感染することになります。

これはイランの核開発施設を狙ったマルウェア「Stuxnet」と比べるとレベルは低いと言えるかもしれませんが、逆にこのようなインターネットに接続している端末を制御系のコンソールに使っているものは、いつでも餌食（えじき）になる可能性があることを意味しています。

現在使われているメジャーな制御システムには、GE Cimplicity、Advantech/Broadwin WebAccess、SIMATIC WinCCなどのシステムがありますが、いずれもHMIはPCベース（Windowsベース）なので、理屈のうえでは今回と同じような攻撃が可能です。

たとえば、2016年3月25日時点で、Siemens社の日本語サイトで紹介している「SIMATIC WinCCプロセスビジュアライゼーションシステム 日本語版カタログ」を参照してみると、SIMATIC WinCC V7.0の推奨動作環境は次のとおりです。

- Windows Vista 32-bit Ultimate、Businessおよび Enterprise
- Windows XP Professional
- Windows Server 2003およびWindows Server 2003 R2

つまり、これらのプラットフォームを麻痺させることができれば、制御システムのHMIが麻痺してしまい、HMI経由でコントロールできなくなるということにつながります。

具体的には、たとえば2014年10月にJPCERT/CCから告知された「Microsoft OLEの未修正の脆弱性に関する注意喚起（JPCERT-AT-2014-0043）」

注4）Ongoing Sophisticated Malware Campaign Compromising ICS（Update E）（March 2, 2016）
https://ics-cert.us-cert.gov/alerts/ICS-ALERT-14-281-01B

（CVE番号ではCVE-2014-4114）[注5]を使った攻撃が行われれば、極めて危険です。

これは攻撃側がMicrosoft Officeファイルをユーザに送り付けるなり、ダウンロードさせるなりしてドキュメントを読み込ませると（ファイルを開かせると）、任意のコードを実行させられる脆弱性です。しかも、この脆弱性に対する攻撃はベンダから修正版が出る前に確認されました。いわゆる「ゼロデイ攻撃」です。OLEは、Windowsファミリではサーバでもデスクトップでも共通に搭載されていますので、この脆弱性は多くのWindowsのバージョンに影響を与えていました。

HMIのコンソールとなっているPC上で、ドキュメントファイルを開けるような環境であることを知っているならば、狙うのは難しいことではありません。また、HMIのコンソールだと知らなくても、たまたまオペレータが不注意でドキュメントを開いてしまうかもしれません。その瞬間にマルウェアに感染してしまいます。

もちろんこのような運用をしているところは極めて少数でしょう。ですがゼロではないはずです。

ウクライナのケースはもっとひどい

ICS-CERTのレポートによれば、被害にあったウクライナの電力会社はGE Cimplicity HMIを利用しており、さらに直接インターネットに接続していました。直接インターネットに接続していて、しかも外部から直接アクセス可能だなんて、そんなバカなと思うでしょう。しかし、本当のようです。そして、GE Cimplicity HMIのコンポーネンツであるCimplicity CimWebServerの10212/tcpポートに細工をしたメッセージを送ると、任意のコードが実行できるという危険性の高い脆弱性を放置していました。

この脆弱性に関しては、2014年1月にCVE-2014-0751（ICS-CERTからはICSA-14-023-01として公開）の脆弱性情報が公開されています。そして、GE Cimplicity HMIは少なくとも2012年1月から脆弱性を抱えていることがわかっています。CVSS v2（共通脆弱性評価システム）での深刻度の値は6.8となっています。電力会社はシステムのアップデートをしていなかったようで、この脆弱性を狙った攻撃が有効となりました。

BlackEnergyはC&Cサーバと通信を行い、命令に従って外部からマルウェアをダウンロードします。必要なマルウェアはもちろんのこと、外部からファイルをダウンロードするのも自由自在です。たとえば、GE Cimplicity HMIの設定スクリプト（拡張子は.cim）を外部からダウンロードし入れ替えるようなこともできます。つまり、実質的にそのコンソールから制御している電力システム全体を乗っ取ることも可能です。

もちろん、その制御システム全体がどのような構成になっているかの知識がなければ、コントロールするためのスクリプトは作成できないので、ハードルは高いでしょう。しかし、ハードディスクの中をきれいに消去してシステムを麻痺させることは簡単です。いくつかのセキュリティベンダによると、今回の事件では、KillDiskをダウンロードし、それを使ったとあります。KillDiskという名前から簡単に機能は想像がつきますが、きれいさっぱりハードディスクの中身を消去してしまえば、システムが麻痺してしまうのは当然です。

電力会社だけではなかった攻撃対象

トレンドマイクロ㈱のセキュリティブログには、たいへん興味深いことが書かれています[注6]。ウクライナの大手鉱業会社や大手鉄道会社での感染情報を調べると、電力会社への攻撃に使われていたものと同じものが検出されたそうです。

つまり、特定の電力会社を集中的に狙ったのではなく、CVE-2014-0751の脆弱性をターゲットにして、かなり広範囲に10212/tcpポートをスキャン

注5） Microsoft OLEの未修正の脆弱性に関する注意喚起（JPCERT-AT-2014-0043） https://www.jpcert.or.jp/at/2014/at140043.html
注6） エネルギー業界だけが標的ではなかった「BlackEnergy」の攻撃（2016年2月12日） http://blog.trendmicro.co.jp/archives/12828

し、一斉に攻撃した可能性も高いということです。

それがたまたまウクライナでたくさん症状が出たのか、それともウクライナを集中的に狙ったのかは、現状では判断がつきません。なぜならば、まともな運用をしているところならば、ICSのシステムをファイアウォール的な防御もなく、あるいはVPNを使ってネットワーク的に閉鎖空間にすることもなく、直接インターネットにつなぎ、簡単に第三者が10212/tcpポートにアクセスできるような環境にはしないからです。その意味ではウクライナの電力会社は、ネットワークセキュリティに対して無頓着であったと言わざるを得ません。

……と、言ってしまいましたが、ウクライナをそんなふうに言いきれるほど日本は安全なのかどうか。たまたま運がいいだけなのかもしれません。

一歩間違えば大惨事

12月23日という冬の季節に、長時間停電したのはさぞかしたいへんだったと思いますが、爆発や火災といった惨事が発生したり、死者が出たりはしていないようですので、その点に関しては不幸中の幸いかと思います。

これまで、「情報」=「形のないもの」への攻撃だったものが、今や物理的なダメージに直結した危険な攻撃が出てきました。ウクライナのケースは極端な例のように思えますが、必然的に、このような攻撃は今後も増えるでしょう。また、これまでコンピュータセキュリティ、ネットワークセキュリティなど考慮していなかった制御系システムを、意図せず攻撃が可能となる形でネットワークに接続して運用されてしまうことも出てくるでしょう。完全にネットワークから切り離されていても、StuxnetのようにUSBメモリなど外部から持ち込まれるものもあるでしょう。

今回のウクライナの停電は、けっして遠い国のお話ではないのです。SD

新連載

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

今年の夏に実施予定のWindows 10 Anniversary Updateにて、“Bash on Ubuntu on Windows”の提供計画がMicrosoftから発表されて注目の集まるShell環境。本連載は、UnixのShellの実践的な操作とさまざまなコマンドを通じて、OSへの理解を深めていきます。第1回目は、端末操作の基本とオンラインマニュアルの使い方を紹介します。

第1回 Unixコマンドを探す旅

はじめに

今号から新連載Unixコマンドライン探検隊が始まります。我らがSoftware Designは、コマンドラインインターフェース(CLI)をとても重視しているコンピュータ専門誌です。これまでにも、CLIを取り上げた「開眼シェルスクリプト」などの名連載がありました。この連載は、これらの偉大な業績を継承してUnixのCLIを使ったコマンド操作を中心に紹介し、それらを通してOSのしくみも楽しみながら理解しようという欲張り企画です。

本連載での中心ターゲットは初心者です。執筆にあたって、入門者ができるだけふるい落とされないように、記事の全体が難しくなり過ぎないように注意します。加えて、熟練のエンジニアにとっても、知識の再確認や、ちょっとした作業効率改善に役立つものにしたいと考えています。Unixは長きにわたって**発展を続けている**OSです。筆者自身が、CLIを操作していてまだに不思議に思うことや知らなかったこともたくさんあります。この連載を通して、みなさんと一緒にUnixへの理解をさらに深めていきたいと思っています。

記事の一部分は深く掘り下げた内容になるかもしれませんが、わからなければ気にせず斜めに読み飛ばしてください。本当には理解していなくても、使われている言葉や対象を眺めて**知った気になる**のも対象を本当に理解するための第一歩になります。ただ可能なら**読んで知る**だけでなく、動作を確認するための環境も手近に用意し、実際に**キーボードを叩いて確認して**ください。安全な環境[注1]を用意して、たくさんの試行を繰り返しましょう。成功はもちろん、**失敗の経験がとても大切**です。

実践してみると、記事に書かれているのとは違う動作をする環境もあるかもしれません。そうした場合は、何が同じで何が異なっているかも考えてみるようにしてください。きっと理解が深まり、技術は向上するはずです。**実践は、エンジニアリング力強化の王道です**。

「Unixって何?」という比較的若い?方、「いまさら」という古のころからコンピュータを知っている方もいらっしゃることでしょう。Unixは、1969年にAT&Tのベル研究所で開発が始まったオペレーティングシステムです。現在に至るまでさまざまな経緯を経て、LinuxやMacintoshのOS XなどのUnixの流れを汲むオペレーティングシステムが開発されました。UNIXはAT&Tの登録商標ですが、本連載では、これらのUnixの流れを汲むオペレーティングシステムをまとめてUnixと呼ぶことにします。本連載で対象とする環境は、主としてOS XとLinuxです。

注1) ファイルを消してしまったり、システムを落としてしまったりと、失敗しても大丈夫なように、プライベートなマシンや仮想環境が安心です。

まずは端末アプリケーション

CLIをご存じでない方もいらっしゃるかもしれませんね。とりあえず、画面キャプチャをご覧ください(**図1**)。

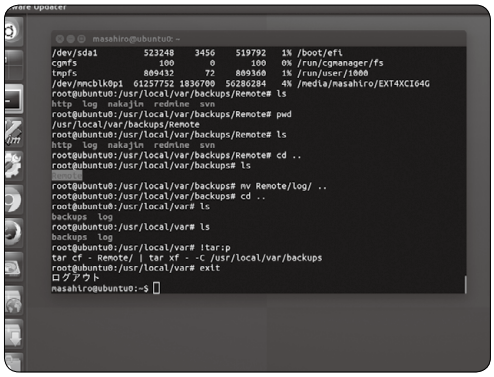
OSや使っている言語によってさまざまな端末アプリケーションがありますが、**端末**、**ターミナル**、**Terminal**などと呼ばれているアプリケーションを起動した状態です。**図1**では黒いウィンドウですが、いろいろとカスタマイズして色やフォントを変更することができる端末もあります。みなさんもOS XやLinux上で端末アプリケーションを実行してみてください。

> ウィンドウシステムが動いていない状態でも、(端末アプリケーションではありませんが)このコマンドラインインターフェースは使うことができますので、クラウドやサーバ環境でも同様の操作が可能です。

図2の状態で、**入力プロンプト**の後ろにさまざまなコマンドを入力しコンピュータを操るのです。

よくこの画面を揶揄(やゆ)して「黒い画面」とか呼ばれて素人筋からは敬遠されていますが、GUIのきらびやか?なインターフェースに対して、男らしく突き放した感じが素敵じゃないですか;-)。みなさんも、CLIを操る上級コンピュータ職人を目指してください。

▼図1　CLIの画面

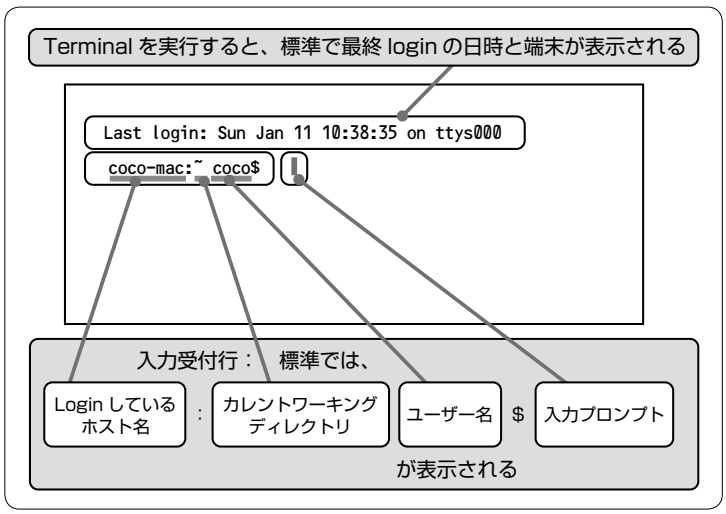


CLIは魔法の呪文!?

CLIとスクリプト言語環境は、ソフトウェアエンジニアにとっては**プログラミングから実行テストまでが俊敏に実施できる軽量性**、また熟練したシステム管理技術者にとってはボタンを押すだけの定型的な操作よりも**システムの深淵まできめ細かく操作できること**に実践的な価値があります。1つ1つの機能を簡潔に見通しよくすることで、長い期間安全・安定的に使い続けることができます。また、軽量なツールであれば、CPU、メモリなどのリソース消費が少なく、限られたリソース環境においても最大の性能を引き出すことが可能です。

このように、小さな機能を組み合わせて問題を解決する手法は、Unixの基本的な考え方です。この考え方は、Agile(アジャイル)やXPなどのソフトウェアプロセスで重視される簡潔さ、つまり「ニーズを満たす解決策の中で最も簡潔なものを開発すること」そして「簡潔であることを維持すること」と符合します。

▼図2　Terminal画面の説明(OS X)



CLI上の短い文によって人が一生かかってもできないような仕事をコンピュータが一瞬でやってしまうので、まるで魔法の呪文のように見えるかもしれません。ですが我々探検隊は、それらのしくみとその秘密を科学の視点[注2]で探っていきましょう。

さあ、旅立とう!

Unixコマンドを知る冒険に出発するにあたり、冒険の対象となる世界を知ることが重要です。これから私達が体験するコマンドは、どこにあるのでしょう。コマンドだけでなく、関連するファイルの場所も知っておくべきです。

実行可能なコマンドの多くは、/bin、/usr/bin、/usr/local/bin、/sbin、/usr/sbin、/usr/local/sbin、/opt[注3]などのディレクトリ[注4]に入っています。

試しに、lsコマンドでどのようなファイルがあるのか確認してみてください(**図3**)。たくさんのコマンドが表示されました。みなさんの環境でも、同様に多くのファイルが表示されたことと思います。さすがに連載の中で、これだけたくさんのソフトウェアをすべて紹介することはできません。重要なコマンドや便利なコマンド、楽しいコマンドを紹介していこうと思います。

これらのコマンドは、単純な仕事をこなすだけのものも、スクリプト言語やコンパイラ、DBMS、コンパイラやスクリプト言語を記述することもできるすごく強力なツールも含んだ、たくさんの英知の蓄積です。Unixの、小さなツールを組み合わせて使うという考え方にそって、これらのツールを組み合わせて使えば、さらにできることは広がります。

注2) OSやプログラムの動作原理や背景を知ること。

注3) /で始まる名前がディレクトリです。ディレクトリは階層的な構造になっているので、/usrの中に/binなどのサブディレクトリを含むことができ、その場所を示すには/usr/binと続けて記述します。

注4) Unixではディレクトリ、Mac OS＝OS Xなどではフォルダと呼んだりしますが同じ意味です。

Googleの前にman

ところで、コンピュータの操作をしていてわからないことがあった場合、どうしていますか。最近ではほとんどの人が「Google先生」に尋ねるのではないでしょうか。

使っている環境の体系だった最も確からしい情報源はマニュアルです。そのシステムに付属しているものですから、バージョンの微妙な違いによる情報の食い違いなども心配する必要はありません。もちろん、外部の動作だけを見て「なんとなくうまく行ったよ」などという頼りない情報でもありません。

経験の長いエンジニアであれば、以前はUnixの印刷されたマニュアルを手にしながら作業していたかもしれませんね(**写真1**)。最近では、これらのほとんどのドキュメントがデジタル化されてPDFやHTMLで読めるようになり、冊子になったものは見かけなくました。

Unixでは、エンジニアリングの現場ですぐに必要な情報はオンラインマニュアルで参照できます。オンラインマニュアルは、manというコマンドで参照します。manはとても便利で、個別のコマンドなど後に紹介する情報のほとんどを確認することができます。

早速、使ってみましょう。**図4**はmanコマン

▼図3　lsの出力例

```
$ ls /bin /usr/bin /usr/local/bin /sbin /usr/sbin /usr/local/sbin

/bin:
[   df    launchctl pwd   tcsh
bash  domainname  link  rcp   test
cat   echo  ln    rm    unlink
chmod   ed    ls    rmdir   wait4path
cp    expr  mkdir   sh    zsh
...中略...

/sbin:
autodiskmount fstyp   mount   mount_ntfs  pfctl
disklabel fstyp_hfs mount_acfs  mount_smbfs ping
dmesg   fstyp_msdos mount_afp mount_udf ping6
...中略...

/usr/bin:
2to3      mdls
...後略...
```

ドの使い方をOS X 10.11.3で参照した例の一部分です。表示された情報をmanページといいます。

「おいおい、英語じゃないか。」……そうなんです。環境によって異なりますが、標準では英語のマニュアルしか入っていないシステムがほとんどです(日本語マニュアルについては後述します)。

コンピュータ関係の情報全般に言えることですが、最も迅速にアップデートされる正確な情報はたいてい英語ですので、一歩上のレベルのエンジニアを目指すべく、できるだけ英語のものも読みましょう。学校の英語の授業のように翻訳する必要もなければ、辞書を片手に悶絶する必要もありません。筆者もそうでしたが、学生時代英語が苦手だった人でも技術英語であれば、専門用語はわかりますし、文も曖昧さのない形式がほとんどですので、慣れてしまえばスラスラと読めるようになります。日本語マニュアルがあれば、それも参照しつつ(とはいえ自動翻訳はあまりお勧めできませんが)実践的に、必要な情報を読み取ってください。日本語のmanページに書かれていないことが、英語のmanページには書かれていることが意外とあるものです。

▼写真1　印刷されたUnixのマニュアル。System VとBSD

manを読む

manで参照する際、どのような情報が記載されているのか知っておく必要があります。マニュアルの章立て(セクションの構成)は、Unix系のOSではおおむね共通です。

1. 一般的な実行可能コマンドとシェル・コマンド
2. システムコール
3. ライブラリファンクション
4. スペシャルファイル(/devの下のデバイスドライバーなど)
5. ファイルフォーマットとその約束事
6. ゲームとスクリーンセーバー
7. その他の情報
8. システム管理コマンドとデーモン
9. カーネル開発者向けのマニュアル

個々の環境でどのような章立てになっているかは、次のmanページで確認できます。

▼図4　manの実行例(OS X)

```
$ man passwd↵

PASSWD(1)                 BSD General Commands Manual                 PASSWD(1)

NAME
     passwd -- modify a user's password

SYNOPSIS
     passwd [-i infosystem [-l location]] [-u authname] [user]

DESCRIPTION
     The passwd utility changes the user's password.  If the user is not the super-
     user, passwd first prompts for the current password and will not continue unless
     the correct password is entered.
...後略...
```

```
Ubuntu
$ man man-pages

CentOS、Ubuntu(先の方法に加えて)共通で
$ man man

OS X
$ man manpages
```

コマンドとして実行できるものは、おおむねセクション1に書かれています。管理系のコマンドはセクション8、ゲームのマニュアルはセクション6にあります。

このように、manがカバーしているのはコマンドだけではなく、プログラミングやシステム管理に必要な「システムコール(2)」や「ライブラリファンクション(3)」など多くの情報を含んでいます。同じ名称で異なるセクションに情報がある場合には、明示的にセクションを指定してページを参照します。

例として、/usr/bin/passwdコマンドと/etc/passwdの形式について調べてみましょう。manでセクションを指定しない場合には、セクション1のpasswdコマンドの情報が表示されます。

セクション1については、先出の「**図4　manの実行例(OS X)**」をもう一度見てください。これは、次のように明示的にセクション1を指定した場合と同じです。

```
$ man 1 passwd
```

/etc/passwdの形式を確認したい場合はセクション5を指定します(**図5**)。

▼図5　セクション5のpasswdマニュアル

```
$ man 5 passswd↵

PASSWD(5)                 BSD File Formats Manual                PASSWD(5)

NAME
     passwd, master.passwd -- format of the password file

DESCRIPTION
     The /etc/passwd file is a legacy BSD 4.3 format file.  It is mostly unused, but
     is updated by some utility programs.  Its format is similar to the /etc/mas-
     ter.passwd file, except that it does not contain the class, change, and expire
     fields described below.
...後略...
```

manの探し方

どのセクションか、明確なファイルやコマンドの名前がわからない場合、本文中に含まれているキーワードを指定することもできます。

aproposコマンドは、manページのユーティリティです。引数に正規表現を指定してmanページ名と"説明(DESCRIPTION)"中を検索することができます。また、manに次のオプションを指定して、manページ内のワードを検索することもできます。

キーワード検索のオプション

-k……オプションに続くキーワード(正規表現；別の回であらためて解説します)を、manページ名と"説明(DESCRIPTION)"の中から検索する。一致するすべてのmanページが表示される。aproposコマンドを使うのと同じ意味
-K……オプションに続くワードがmanページに含まれているかページの全文を検索するのですごく遅い。見つかったmanページを表示するか、次を検索するかなどを対話的に確認しながら検索する。Linux系では**--regex**オプションを指定すれば、正規表現も使える

STEP UP!

aproposコマンドの実体は、Ubuntuでは**whatis**コマンドへのシンボリックリンク、OS Xではシェルスクリプト[注5]です。OS Xの**whatis**は、これもまたほとんど**apropos**と同じ内容のシェルスクリプトです。

たとえば、"file open"というキーワードを探してみましょう(**図6**)。これは、次と同じことです。

注5) 興味のある人は中をのぞいてみましょう。

```
$ apropos "file open"
```

次は、-Kオプションで全文検索を試してみます(**図7**)。

Linux(Ubuntu/CentOSなど)では、一度manページが開いてから、PAGERをぬけた後に、対話オプションを選択します(**図8**)。

見出しについて

ここで、manページの形式を確認しておきましょう。もう一度「**図4　manの実行例(OS X)**」を見てください。大文字の見出し+内容という書式体裁になっていますね。ほかのコマンドを調べても、ほぼ似たような形式です。

STEP UP!

OS Xのmanページは、いずれもヘッダ部分に"BSD ... Manual"という記述があると思います。OS Xは、FreeBSDというUnixのディストリビューションをベースにして作られたOSですので、マニュアルもBSDのものが流用されているのですね。

見出しに注目すれば、だいたい目的の情報が見つかるはずです。コマンド(セクション1)についての主要な見出しを紹介しておきましょう。

- NAME　名前
- SYNOPSIS　書式
- DESCRIPTION　説明
- EXAMPLES　例
- SEE ALSO　関連項目

今回のまとめと次号について

今回の確認リスト

【manで調べるもの】

man、apropos、more、less、ls、groff、nroff、zcat、printenv、env、apt-get(Ubuntuなどの Debian系Linux)、yum(CentOSなどRedHat系Linux)

今回は、Unixとコマンドラインインターフェースについての紹介と、manコマンドについて紹介しました。本編では十分に紹介しきれませんでしたが、1つのコマンドに見えるmanは、groffやless、apropos など外部のコマンドなどとも連携して動いています。興味のある方は、これらについても調べてみましょう。次回は、ファイルシステムのお話と、実際にファイル操作を体験します。SD

▼図6　キーワード検索の例(OS X)

```
$ man -k "file open"↵
TIFFSetField(3tiff), TIFFVSetField(3tiff)
- set the value(s) of a tag in a TIFF file
open for writing
opensnoop(1m)            - snoop file
opens as they occur. Uses DTrace
```

▼図7　全文検索の例(OS X)。実行に時間がかかります

```
$ man -K "file open"↵
masa-air:~ masahiro$ man -K "file open"↵
/usr/local/share/man/man1/gnutls-serv.1? [ynq] n
/usr/local/share/man/man3/TIFFSetField.3tiff? [ynq] y
/usr/share/man/man1/authopen.1? [ynq] n
/usr/share/man/man1/fuser.1? [ynq] q
```

▼図8　全文検索の例(Ubuntu)。実行に時間がかかります

```
$ man -K "file open"↵
masahiro@ubuntu0:~$ man -K "file open"↵
--Man-- next: pager(1) [ view (return) | skip (Ctrl-D) | quit (Ctrl-C) ]
next
--Man-- next: x509(1ssl) [ view (return) | skip (Ctrl-D) | quit (Ctrl-C) ]

--Man-- next: HTML::TreeBuilder(3pm) [ view (return) | skip (Ctrl-D) | quit (Ctrl-C) ]
--Man-- next: IO::InnerFile(3pm) [ view (return) | skip (Ctrl-D) | quit (Ctrl-C) ]
--Man-- next: PerlIO::gzip(3pm) [ view (return) | skip (Ctrl-D) | quit (Ctrl-C) ]
上記で、manページの表示と入力文字は割愛しています。
```

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第30回 ❖bhyveでOpenBSDファイアウォール on FreeBSDを構築（その5）

OpenBSD on FreeBSD ——NAT／双方向変換／ポートフォワーディング

FreeBSDハイパーバイザでOpenBSDを動作させ、そこでOpenBSDネイティブのパケットフィルタリング／ファイアウォール機能の最新版pf(4)を使ってみよう、というシナリオでここ数回にわたってpf(4)を取り上げてきました、NATとポートフォワーディングを取り上げる今回で、pf(4)シリーズもファイナルです。

企業でも仮想環境でも家庭でもそうですが、インターネットにサービスを提供しているサーバ以外のマシンに直接IPv4のグローバルアドレスを振るといったことは、それだけ費用がかかるのであまりしません。たいていはISPから単一のグローバルIPv4アドレスがランダムに割り当てられ（実際にはだいたい同じIPv4アドレスが割り当てられることが多いみたいですけれども）、これを複数のマシンやスマートフォン、タブレットデバイスなどで共有しています。

こうした場合に使われる技術がNAT（Network Address Translation）と呼ばれる技術です[注1]。NATではネットワークに所属しているマシンから送付されるIPパケットを単一のIPアドレスにマッピングし、あたかも単一のマシンからIPパケットが送信されているのように見せかける技術です。通常はルータでこの処理を行っています。

たとえば、LANの中ではグローバルIPではなく次のようなプライベートIPが使われます。

- 10.0.0.0～10.255.255.255
- 172.16.0.0～172.31.255.255
- 192.168.0.0～192.168.255.255

仮にLANで192.168.1.*といったアドレスが使われていて、ルータはLAN側が192.168.1.1、WAN側が特定のグローバルIPになっているとします。ルータはLANから送られてくるパケットの次の情報をルータのグローバルIPのものに書き換えてからインターネット側に送信します。

- 送信元IPアドレス
- 送信元TCP/UDPポート番号

書き換わったIPパケットが送信されていきますので、外のマシンからはこのルータの存在しか見えません。ルータは戻ってくるIPパケットを監視してIPアドレスとポート番号をチェックし、送付したパケットの返信パケットに一致した場合には記録しておいた送信元IPアドレスとTCP/UDPポート番号に戻し、ファイアウォールを華麗にスルーしてLANの中に戻します。LAN内のマシンには変換後のIPパケットが届きますので、こいつはIPパケットが途中で書き換わったことは知りません。

これがNATの基本的な動作です。ping(8)で使われるICMPも、NATでの扱いはポート番号の変換が

注1　参照：RFC 1631 The IP Network Address Translator（NAT）および以降の改訂版

◉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。

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。

ない以外は基本的に同じです。pf(4)はNATの機能を提供していますので、ルータで設定するようなことはそのままpf(4)で指定できます。

IPフォワーディングを有効化

NATを使うということは、OpenBSDのネットワークインターフェースの間をIPパケットが行ったり来たりすることを意味しています。こうした機能はルータやゲートウェイで使われる機能で、OpenBSDで利用するにはオペレーティングシステム側の設定を変更して、IPフォワーディングと呼ばれる機能を有効化してあげる必要があります。

設定は/etc/sysctl.confファイルに記述します。IPv4とIPv6でそれぞれ設定する項目値が異なります(**リスト1、2**)。

設定を/etc/sysctl.confに追加したらシステムを再起動するか、**図1および2**のようにsysctl(8)コマンドを実行して手動でIPフォワーディングの機能を有効化します[注2]。

NATルールセット

NATの設定は外側に向かうpassパケットルールにおいてnat-toパラメータを使って指定します。原理的にはpassで指定することになりますが、これだとすべてのpassにnat-toを付けることになって煩雑です。ですから、通常はmatchというルールのほうでnat-toを指定し、これを自動的にpassルールのほうに適用させます(**リスト3**)。

matchに一致するとそこで指定されているオプションが記憶され(今回はnat-toがここに該当します)、passに一致したときにmatchに指定されている内容が自動的に適用されるようになります。

注2　こうした設定はOpenBSDに限らず多くのOSで似たような設定が必要です。

ルールで記述する中身を説明すると次のようになります。

- match……ここに一致したオプションパラメータはpassで一致した場合などに適用されるようになる
- pass……通過許可。matchに一致している場合にはそこで指定されたオプションパラメータがこちらにも適用される
- out……外方向の指定。nat-opはoutが設定されたルールでのみ使用可能
- log……一致したパケットに関する情報をログに出す指定。logは通常通信の最初のパケットのみ記録するので、一致したすべてのパケットのログを取りたければlog allと指定する
- interface……パケットが通過するネットワークインターフェース名またはそのグループ名
- af……アドレスファミリーを指定。IPv4の場合はinet、IPv6の場合はinet6を指定するが、通常はpf(4)が自動判定するため不要
- protocol……プロトコルtcp/udp/icmpを指定。ポート番号を指定した場合にはプロトコルも必ず指定する必要がある
- src_addr……送信元アドレス

▼リスト1　/etc/sysctl.conf IPv4のIPフォワーディング有効化設定

```
net.inet.ip.forwarding=1
```

▼リスト2　/etc/sysctl.conf IPv6のIPフォワーディング有効化設定

```
net.inet6.ip6.forwarding=1
```

▼図1　IPv4 IPフォワーディングを有効化するコマンド

```
sysctl net.inet.ip.forwarding=1
```

▼図2　IPv6 IPフォワーディングを有効化するコマンド

```
sysctl net.inet6.ip6.forwarding=1
```

▼リスト3　match/passを使ったNATルールセット

```
match out on vio0 from 192.168.1.0/24 to any nat-to 外側のIPアドレス
pass on vio0 from 192.168.1.0/24 to any
```

チャーリー・ルートからの手紙

- src_port……送信元TCP/UDPポート番号
- dst_addr……送信的アドレス
- dst_port……送信先TCP/UDPポート番号
- ext_addr……送信元アドレスはここで指定したアドレスへ書き換え(static-portの指定がなければポート番号はランダムな値に変換される)
- pool_type……変換に使用するアドレスプールのタイプを指定
- static-port……この指定があるとTCP/UDPポート番号は変換されずにそのまま使われる

アドレスやポート番号の記述はこれまで紹介してきたものと同じです。かなり柔軟に指定することができます。

matchを使わないでpassだけで書くとすれば**リスト4**のようになります。

ただし、この書き方はIPアドレスの変更があるとルールも書き換えないといけないので、あまり推奨されていません。たいていの場合は**リスト5**のようにインターフェースを指定してpf(4)に自動的に設定してもらうようにします。これでLAN側のIPアドレスの変更を気にしなくてもよくなります。

外側のインターフェースのIPアドレスはpf.confの読み込み時に決定されますので、外側のインターフェースのアドレスがDHCPで振られていてときどき変わるといった場合には**リスト5**の書き方だと対処できません。その場合は**リスト6**のように記述します。

これでだいたい設定に手を入れることなく、通常のNATに求められる機能はこなせると思います。現在アクティブになっているNAT変換については、pfctl(8)を**pfctl -s state**のように実行すれば一覧表示させることができます。

双方向マッピング

LAN内部で運用しているサーバを外部から直接利用できるようにしたい、といった場合にもこのNATの機能が利用できます。指定するパラメータはbinat-toにかわります。たとえば**リスト7**のように設定します。

たとえばこの場合であれば、LANの中にある192.168.1.50は設定した外部のIPアドレスに相互に変換されることになるので、あたかも非武装地帯に設置しているマシンのように振る舞うことになります。似たような機能は、名前はいろいろありますが、ルータの設定画面などで見たことがあると思います。

ポートフォワーディング

双方向マッピングのようにマシンごと外側に設置されているように見せるのでなく、特定のポート(つまりWebサーバだけといったように特定のサービス)だけをLAN内のマシンに変換して通したいということがあります。この機能はリダイレクションまたはポートフォワーディングと呼ばれ、**リスト8**のように指定します。

fromで指定する内容をanyから特定のIPアドレスに設定すればアクセスしてくる適用対象を絞り込むことができますし、**port 番号:番号**のように記述すればポート番号を範囲指

▼リスト4　passだけで書いたNATルールの例

```
pass out on vio0 from 192.168.1.0/24 to any nat-to 外側の IP アドレス
```

▼リスト5　推奨されるNATルールの書き方の例

```
pass out on vio0 inet from vio1:network to any nat-to vio0
```

▼リスト6　外側のIPアドレスがDHCPで振られている場合のNATルールの書き方の例

```
pass out on vio0 inet from vio1:network to any nat-to (vio0)
```

▼リスト7　LAN内部のサーバを外部から利用できるようにするルールの例

```
pass on tl0 from 192.168.1.50 to any binat-to 外側の IP アドレス
```

定でフォワーディングさせることもできます。この機能もたいていのルータには搭載されているので、Webの管理画面などで似たような設定を見たことがあると思います。

もっと知りたいpf(4)のルールの記述方法

これまで数回にわたってpf(4)の機能を紹介してきましたが、ここで取り上げた内容は本当に基礎の部分です。アドレスプールやロードバランシング、ポリシーフィルタリング、ファイアウォールリダンダンシ、リンクアグリゲーションなど、pf(4)はほかにもさまざまな機能を提供しています。

そういった機能はどうやって設定すればよいのか(いくつかの機能はpf(4)のルールだけではなくOpenBSD側でいくつかの設定も必要になったりするのですが)知りたくなると思うのですが、ドキュメントを使う以外に、ちょっとばかり慣れない方法ですが、pfSenseやOPNsenseのルールを真似するという方法があります。

■pfSense
https://www.pfsense.org/
■OPNsense
https://opnsense.org/

pfSenseやOPNsenseはFreeBSDベースのルータ/ファイアウォールソリューションです。pf(4)がベースになっており、どちらも開発が活発に続いています。OPNsenseがフォークしたあとはOPNsenseに開発が流れるかと思いましたが、どちらもバックグラウンドに企業があり、現在でもともに活発に開発が続いています。

これらソフトウェアアライアンスでは、WebのUIを経由してさまざまな設定を行うことができます。pfSenseやOPNsenseをインストールしてブラウザ経由で設定を行い、その設定結果をサーバにログインして確認します。こうすることで設定内容を見ることができますので、書き方を学んで自分のルールセットに適用するといったことができます。それならpfSenseやOPNsenseをそのまま使えばよいのではないかということになりますが、たしかにそうだと思います。

ただ、とくにOPNsenseは開発が活発で頻繁にリリースがでています。安定した環境で運用したいなら、アップデートなどが自由に管理できる環境で、自分でpf(4)ルールを書いて運用するというのは1つの方法だと思いますし、インターフェース任せではなく実際の書き方を知っておくと技術の幅が広がってよいので、いったんは書き方を学んでみることをお勧めします。

広がるpf(4)ワールド

pf(4)はOpenBSDのみならずFreeBSDやほかのOSでも使われているので、ファイアウォールのルール記述のスキルを共有しやすいという特徴もあります。代表的なところでは次に挙げるOSで利用できます。

- Mac OS X 10.7(Lion)およびこれ以降のバージョン
- FreeBSD 5.3およびこれ以降のバージョン
- FreeBSDをベースに開発されているファイアウォール/ルーティングソリューション pfSense/OPNsense
- NetBSD 3.0およびこれ以降のバージョン
- DragonFly BSD 1.2およびこれ以降のバージョン
- iOS
- QNX

OpenBSD以外のOSで使用できるpf(4)の機能は常に最新版と同じ(つまり、最新のOpenBSDと同じ)というわけではありませんが、基本的な書き方は通じますので、共通技術として使い回しが効くという利点もあります。SD

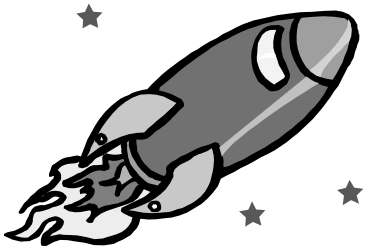
▼リスト8　特定のポートをLANの中の特定のマシンにフォワードする設定の例

```
pass in on vio0 proto tcp from any to any port 80 rdr-to 192.168.1.50
```

35

Debian Developer　やまねひでき　henrich@debian.org

10年かけて解決 DebianのFirefox問題



Debian Hot Topics

Debian開発の近況

squeezeはLTS終了、WheezyはLTSへ

Debian 6「squeeze」のLTS(Long Term Support)が2月29日に終了し、squeezeのパッケージはミラーサイトから削除されてarchive.debian.orgへ移動しました。いまだにsqueezeを使っているという方は、一刻も早く現在の安定版Debian 8「Jessie」への移行を推奨します。

Debian 7「Wheezy」・Debian 8「Jessie」のポイントリリース(7.10および8.4)が4月2日に行われました。内容としては、いつもどおりのこれまでのセキュリティ修正+バグ修正で、格段変わった点はありません。

なお、Debian 7「Wheezy」のリリースは2013年5月4日でしたので、そろそろ通常のサポートは終了を迎え、LTSへ引き継ぎとなります。関係者間の調整の結果、1つ前のsqueezeとは違い、Wheezyは設定を変更する必要なくLTSサポートを受けられるようになっています。詳細な設定の確認については、Debian Wikiの該当のページ[注1]を参照ください。

Debian 9「Stretch」のリリース

そして、現在開発中のDebian 9「Stretch」のリリース作業ですが、LTSであるカーネル4.10を採用するためにフリーズ開始時期を遅らせるようです[注2]。2016年11月5日から徐々にフリーズを開始し、2017年2月5日には完全フリーズ。そして、そこからバグを潰してリリースとなります。

Debianの場合、事前にリリース日は決定されず、バグの収束速度から直前にリリース日が決まります。代わりに新しいバージョンの流入を止める「フリーズ」の日程が決まっていて、だいたいのスケジュールが読めるようになっています。前回のリリースなどから推測すると、フリーズは5ヵ月半から6ヵ月ほどかかっていたので、「Stretch」のリリースは2017年7月末か8月になるものと思われます。

お疲れ様Iceweasel、お久しぶりFirefox

ロゴやアートワークのライセンスの問題が解決され、FirefoxがDebianに戻ってきました[注3]。不安定版(unstable、sid)あるいはテスト版を使っている場合、インストールされているiceweaselは、firefox-esrパッケージで置き換えられています[注4]。なお、firefox-esrで採用されているものよりも新しいバージョンのFirefoxを使

注1) URL https://wiki.debian.org/LTS/Using

注2) URL https://lists.debian.org/debian-devel-announce/2016/03/msg00000.html　ここでのLTSとは「Linux KernelのLTS」を指します。Upstreamの長期メンテナンスサポート期間に歩調を合わせることで、Debian側での作業負担を軽減しようという意図ですね。

注3) URL https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=815006

注4) ESRは、Extended Support Releaseの略。おもに法人企業などでの利用を念頭に置いて、Debianでの安定版のように、バグ修正のみを中心に適用するリリース。
URL https://www.mozilla.jp/business/参照

いたい場合は、不安定版(unstable、sid)では、firefox-esrではなくfirefoxパッケージのほうを利用すれば良いなど、選択肢の幅が広がりました(安定版リリースではfirefoxパッケージは利用できず、firefox-esrのみになる予定です)。

さて、今回はなぜDebianにはFirefoxではなくIceweaselが入っていたのか(**図1**)、という歴史的な経緯を振り返ってみましょう。

Iceweasel誕生までの経緯

まずこの話の前提として、Firefoxのアイコン・ロゴなどのアートワークのライセンスが、本体とは違いプロプライエタリなものであったという点が挙げられます[注5]。ライセンスが違っていたのは、Mozillaが「商標としてのFirefoxを守るための措置」でした。

よって、Debianではこの部分がディストリビューションのポリシーであるDFSG[注6]に合致しないため、アートワークを削除したFirefoxパッケージとして配布すると、2005年半ばに決定していました。このアートワークを削除した形でのFirefoxの配布は、Mozilla側の担当者と話し合って「OKだよ」という言質を得ていました。

ところが、半年ほど過ぎた2006年2月に突如Mozilla CorporationのMike Connor氏が、「このような形での配布は認められない」と、"Uses Mozilla Firefox trademark without permission"というバグ[注7]を登録したところから話はこじれてきます。

このバグに対して、「Firefoxという名前を使うのに(ロゴは使わなくてかまわないと)Mozilla Foundationの人から以前に許可もらってるよ」とDebian関係者が返します。確かに当時のMozilla Foundation側の関係者であるGervase Markham氏は、「Firefox/Thunderbird trade marks: a proposal」というメールで始めたスレッドの中で(@mozilla.orgのアドレスで)、次のように述べています[注8]。

- Mozilla FoundationはMozilla製品について、Debianおよび公式Debianパッケージの再配布者に対して、そのパッケージについて商標を含む名称を付ける権利を与える
- FirefoxとThunderbirdの公式ロゴは、異なるライセンスでカバーされており、これはDebianにとって制約が大き過ぎる。ロゴはダウンロード可能なソースコードには含まれておらず、ロゴ削除の作業は必要ない。ここまでの議論はこのロゴについてではなく、ロゴを使わないことが双方にとって幸せであるという考えを共有している(と少なくとも私は認識している)

注5) URL http://lxr.mozilla.org/mozilla/source/other-licenses/branding/firefox/LICENSEによると、
「貴方はFirefoxの名称あるいはロゴに留まらず、Mozilla Foundationあるいはその他の団体の商標の利用権またはライセンスを得てはいません。詳細はhttp://www.mozilla.org/foundation/licensing.htmlを参照のこと」
(You are not granted rights or licenses to the trade marks of the Mozilla Foundation or any party, including without limitation the Firefox name or logo. For more information, see: http://www.mozilla.org/foundation/licensing.html)
とあります。

注6) Debian Free Software Guidlineの略。Debianがフリーだ、と考えるソフトウェアの基準で「オープンソースの定義」のもとになった。

注7) URL http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=354622

注8) URL https://lists.debian.org/debian-legal/2005/01/msg00503.html

▼図1　IceweaselとFirefox

これで一見落着……かと思いきや、Mike Connor氏は強硬でした。彼はこう続けます。

「確かに彼はそう言った。そのときは、我々はあまり深く考えてなかったんだ。そして今は、OKではないと言っている」[注9]

つまり、合意をひっくり返したというわけです。

さらに「Firefox」の配布条件として、次のような項目を出してきました。

- ディストリビューションが加えたすべての変更について、なぜ変更が必要なのかの説明をきちんと付けたうえで、Mozilla側に提出してレビューを受けること
- リリースするのは、CVSのタグあるいはリリースされたtarballに許可されたパッチのみにすること
- ビルド設定(configure)についても許可を得ること
- ロゴと商標(名称)は不可分なので必ず合わせて使うこと

そもそもFirefoxのソースコードは、ロゴとアイコンを除いてはOSSライセンスで配布されているものですから、どのようにパッチを当てようが、ビルド設定を変更しようが自由……ではあるものの、この際のMike Connor氏の言い分は、ソフトウェアのライセンスではなく商標権(トレードマーク)を盾にしたものであると考えられます。

商標権とは「事業者が、自己の取扱い商品(サービス)を他人のものと区別するために使用するマークで独占的に使用できる権利。商標を他人が許可なく使った場合はその商品を排除することが可能」というものです。よって、ソースコードのライセンスが何であろうと、それとは別の次元での制約である「名称」=「Firefoxという名前」でMozillaが意図しないソフトウェアを配布しようとするのであれば、Mozilla側は異議を申し立てて配布差し止めが可能になる、というわけです。かなり強力な権利ですね。

これを受け、「ロゴを含めないと『Firefox』としては配布してはいけない」というのに、「『ロゴ』などのアートワークは改変不可の別ライセンスとして配布されている」ため、結局Debianとしては、Firefoxという名称での配布をあきらめざるを得ませんでした。そこで選ばれたのが「Iceweasel」——炎ではなく氷、キツネではなくイタチ、というわけです(ここでFirefoxに似た名前を採用すると、商標に抵触するということで訴えられる可能性があります)。

この問題、最初から「ロゴを含んで公式として認められたものはFirefoxと呼ばれ、ロゴを含んでないものは○○と呼ばれます」というように名称を分けておけば良かったのではないか……と筆者は思います(同様の事柄はChromeにもありますが、こちらはきちんとオープンソース版はChromiumとして名称が分かれていますね)。

「Firefox」の復活

それから時は流れて2014年、Debian開発者でもあるSylvestre Ledru氏[注10]がFirefoxのリリースマネージャに就任し、2016年2月にIceweaselへのバグとして「Renaming Iceweasel to Firefox」[注11]を投稿したところから話は再度大きく動き出します。

このバグでSylvestre氏は(@mozilla.comのアドレスで)次のことを述べています。

- 問題となったロゴのライセンスがDFSGに合致したもの(本体と同じMPL[注12])になったので、ライセンス問題は解決したこと

注9) Fair enough, he did make that statement. At the time, we obviously weren't taking that part seriously. We are now, and we're saying its not ok.

注10) gccではなくclang(LLVM)でDebianの全パッケージを再ビルドして問題を見つけるclang.debian.netの人、というとおわかりになる方もいるかもしれません。

注11) URL https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=815006

注12) Mozilla Public Licenseのこと。現在のFirefoxはMPL2.0のもとで提供されている。

- Mozilla側が提議していたパッチなどのクオリティ問題については、Iceweaselのメンテナ Mike Hommey氏はMozilla内でもトップ10に入るコントリビュータ、かつMozillaの被雇用者であり、その作業クオリティに疑念の余地はないこと
- Debianで適用されているパッチについては、常にMozilla側へ可能な限り修正をフォワードしていて、Debianとして必要なものであると認識していること

このように、以前のバグ登録で述べた結論はもはや適切でなくなり、現在はFirefoxの名称を使うのに支障はなくなったと宣言し、IceweaselからFirefoxへ名称を戻すことを提案したのです。

そして、以前のバグでFirefoxの名称を使うのに反対したMike Connor氏も「当時を振り返ってみると、DFSGの観点からは、FirefoxはEULA[注13]を持つなどロゴマークのライセンス以外も問題だった。幸いなことに、少し前に双方とも問題を解決できている」と、以前の反対姿勢とは打って変わった発言を寄せます。

これらのコメントを受け、複数のDebian開発者からIceweaselをFirefoxに改名することについての歓迎の言葉が寄せられます。しかし、慎重派からは「これはDFSG第8項『ライセンスはDebianに限定されない』[注14]に抵触しないのか？」という質問も出ました。これについてもSylvestre氏が「これは例外と呼ばれる類のものではない、という一点を明確にしておきたい」と回答。Debian派生ディストリビューションについての扱いはどうなるかというと、「Debianのパッチを利用している限りは、Firefoxのブランドを使っても問題ないと見なす（Ubuntuは別のパッチを適用しているのでここには含まれない）」と明言しました。

さらに「Debianやほかのディストリビューションが行っている変更と配布は、現状のMozillaの商標ポリシーとは矛盾しているけど？」という質問に対しても、Sylvestre氏は、「確かにそれを商標ポリシー内で明確にしたほうが良いと思う。ただ、ご想像のとおり、（Debianでも同様だけれども）この類の変更については時間がかかる。その変更が行われるまでは、Mike Connor氏のコメントと私のバグレポート（@debian.orgのメールアドレスではなくて@mozilla.comのアドレスを使ってバグレポートをしているのに注意）で、Debianのftpmaster[注15]に対しては十分な説明になっているだろう」と返答して、疑問点をクリアにしてくれました。

これで、Debianは、晴れてFirefoxという名称のもとでDebianパッケージとして適切なもの[注16]を配布できるようになったのです。

終わりに

IceweaselとFirefoxの問題は、著作権を根拠とする「ライセンス」と商標権「トレードマーク」についての取り扱いの難しさ[注17]、そして両団体を取り巻く人間関係など、いろいろと興味深い一件だったように感じます。解決までに10年もかかってしまったものの、問題の1つが終わりを迎えうれしい限りです。関係者の努力に感謝します。SD

注13) End User License Agreementの略で、利用許諾条件のこと。多くのソフトウェアに付随しているが、OSSの場合はOSS自体のライセンスと二重になり、矛盾が生じる可能性がある。

注14) 「プログラムに付随する権利は、プログラムがDebianシステムの一部であるかどうかに左右されてはいけません。プログラムがDebianから取り出されDebianとは別に使用または配布されるとしても、その他の点でそのプログラムのライセンス条項を満たしているならば、プログラムが再配布されたすべての当事者はDebianシステムにおいて付与されたのと同じ権利を与えられなければなりません」……要は「Debianだけが特別扱いされているならダメだよ」ということ。

注15) Debianのパッケージリポジトリの管理者。どのパッケージを入れる／削除するのかについての作業権限を持つ。

注16) ちなみにMozillaの配布するFirefoxパッケージとDebianのFirefoxパッケージではいくつかの点が違います。たとえば、CiscoのOpenH264ビデオコーデックアドオンを、「ソースがない不明瞭なバイナリblobである」との理由から、Debianではデフォルトで無効にしています。
URL https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=769716参照。

注17) 現状でも、Firefoxを改変したうえでFirefoxの名称で配布した際に、Mozilla側がこれを不適切だと考えた場合には、商標権をもとにMozillaから配布差し止めを受ける可能性が残っている点にはご注意ください。

第73回 Ubuntu Monthly Report

UbuntuとIoT・スマートデバイスの状況

Ubuntu Japanese Team
(株)創夢 柴田 充也(しばた みつや)　mail：mty.shibata@gmail.com

Ubuntu 16.04 LTSがリリースされたことで、Ubuntu開発者は本格的に16.10の開発に携わることになります。この16.10やそれ以降の最も大きな目標が、通常のUbuntuとIoT・スマートデバイス向けのUbuntuの統合です。

Covnergence：あらゆるデバイスで同じUbuntuを

本誌の第2特集でも水野氏やあわしろいくや氏が書いていたように、Ubuntu(とその母体であるCanonical)は、一般的なPCだけでなく、スマートフォンやタブレット、IoTといったあらゆる情報機器でUbuntuを利用できるようにしようという「野望」を抱いています。

ただ「Ubuntuをインストールできる」というだけではありません。GUIやCUIを含めてすべてのデバイスで「同じような操作性」を提供することが重要です。ディスプレイが接続されたデバイスであればUbuntuデスクトップと同様にUnityをベースにしたデスクトップシェルを提供しますし、ディスプレイのないデバイスならUbuntuサーバと同じようにログインして操作できるCUIを提供します。しかもただ提供するだけではなく、利用者の環境に合わせて適切なユーザ体験をもたらす必要があります。

たとえばデスクトップシェル1つとっても、サイズの大きなモニタにつなげばPCのようにマルチウィンドウモードで使いたいでしょうし、スマートフォンのようなサイズは小さいけれども高解像度なディスプレイならAndroidやiOSのようにシングルウィンドウモードの方が使いやすいでしょう。さらにスマートフォンを外部モニターにつなぐ可能性も考えると、このモードは動的かつスマートに切り替わらなくてはなりません。

「Convergence(集中・収斂など、一点に集まっていくという意味)」は複数のデバイスに渡るユーザ体験を1つにまとめ、「Ubuntuの使い方がわかればどのデバイスでも使える」ことを目指しています。言ってしまうとUbuntu Phoneは、このConvergence戦略の最初の一手でしかないのです。

スマートフォン・タブレットのUbuntu

Convergenceを最も実現しているのが、スマートフォン・タブレット向けの「Ubuntu Touch」です。もともとはタッチデバイス向けだったのでこのような名前が付いていますが、「Ubuntu Phone」の方が有名かもしれません。

Ubuntu TouchもUbuntuベースです。Linuxカーネルの上でsystemdやそのほかのプロセスが動いていますし、sudoコマンドで管理者権限を取得できます。Ubuntuの最小環境で入っているパッケージ、vimやpythonはもちろんejectコマンドもあります。通常のUbuntuと大きく異なるのは、ルートファイルシステムが読み込み専用でマウントされていることでしょう。初期状態ですとホームディレクトリといくつかの設定ファイルのみ書き込みできます。

Touchでは「イメージベースのアップデート」を採

用しています。Androidなどと同じく、ベースシステムを更新する場合はカーネルやルートファイルシステムを固めたイメージをダウンロードし、そっくりそのまま置き換えるというわけです。このためaptコマンドを使ってパッケージをインストールすることはできません[注1]。

アプリはUbuntu独自のアプリストアを用意しています。アプリは通常のルートファイルシステムとは隔離された領域に保存されるため、イメージベースのアップデートの影響は受けません。将来的にはデスクトップでもこのアプリをインストールできる予定です。逆にデスクトップアプリをスマートデバイス上で動かすこともできるようになります。こちらは今のところ開発版では実現しているものの、まだまだ解決すべき問題が多いという状態です。デスクトップとスマートデバイスで同じアプリを使える、これが1つ目のConvergenceです。

もうひとつのConvergenceは「スマートフォン・タブレットをデスクトップPCとして使える」ことです。前述したように画面サイズの小さなデバイスはアプリは全画面で表示した方が何かと便利です(**図1**)。しかしながら外部ディスプレイにミラーリングした場合はその限りではありません。とくに画面の半分を専有するソフトウェアキーボードは邪魔ですし、外部キーボードで直接入力した方が何かと便利でしょう(**図2**)。

注1) 開発用という位置づけではありますが、ルートファイルシステムを書き込み可能にすることでaptコマンドで追加パッケージをインストールできます。

図1　シングルウィンドウモード

図2　同じスマートフォンがマルチウィンドウにもなる

そこでTouchでは「Ubuntu Pocket Desktop (ubuntu-pd)」というプロジェクトを立ち上げました。これによりUbuntu PhoneにBluetoothキーボード・マウスをつないだらデスクトップ版のUbuntuのようなUIになるように調整を進めています。現在のUbuntuデスクトップはX.orgとUnity7を使用していますが、TouchはMirとUnity8です。Pocket Desktopとは、Mir・Unity8の組み合わせでマルチウィンドウモードにしたりデスクトップアプリを起動するための作りこみを行うプロジェクトなのです。

イメージベースのアップデートではバージョン名を「OTA-xx」としています。3月上旬時点の最新版はOTA-9で、本誌発売までにOTA-10がリリースされる予定です。OTA-10では待望の日本語入力が導入される予定です[注2]（**図3**）。OTA-10までは15.04ベースでの開発で、OTA-11からは16.04ベースになる見込みです。

注2）実装しました。ただし足りない機能がまだまだたくさんあります。このあたりの苦労話は将来のMonthly Reportで書きたいな、と思っています。

図3　日本語入力も可能

IoT向けのUbuntu

UbuntuではIoT・クラウド向けの小さなイメージとして「Snappy Ubuntu Core」を開発しています。SnappyではTouchでの知見をもとにしたイメージベースのアップデートやアプリ単位での隔離環境に加えて、組み込みでは一般的なロールバック可能なルートイメージの冗長化やミドルウェアの導入に便利なフレームワーク機能も実装しました。しくみや機能そのものはDockerでよく使われているCore OSのUbuntu版といった感じです。

Snappyではパッケージングシステムも一新し、aptコマンドではなくsnappyコマンドを使うようになりました。Snappy上で普通のUbuntuシステムを使用したい場合は、LXDパッケージを導入してその上にUbuntu環境を構築します。現在はsnappyパッケージの作成環境の開発が活発で、とくにパッケージ作成ツールであるsnapcraftは、debファイルからだけでなく、GitHub上のプロジェクトなどからもsnappyパッケージを簡単に構築できるようになりました。

組み込み向けとしてはロボット用OSとして一般的になりつつある「ROS」がSnappyをサポートしています。今年の2月にスペインで開催されたMWCでは、Snappy＋ROSで動作するヒューマノイドロボットがサッカーをするデモが、非常に注目を集めていたようです。

これだけだとただの組み込み向けのOSなのですが、このSnappyシステムは実はデスクトップユーザにも影響します。なぜならUbuntuは将来的にデスクトップもTouchもSnappyベースに移行する予定だからです。「Ubuntu Personal」とも呼ばれるプロジェクトでは、来年10月にリリースされる16.10を目処に、Snappy化したデスクトップイメージを提供する予定になっています（**図4**）。

Ubuntu PersonalではスマートデバイスむけのTouchも統合し、デスクトップ環境はMirとUnity8を採用します。また半年ごとのリリースではなく、Snappyと同じくローリングリリースを採用する見込みです。少なくともアプリとベースシステムは独立

したリリーススケジュールとなるでしょう。

ちなみにKubuntuを始めとするほかのフレーバーは、従来どおりの環境を提供する予定です。またUbuntu Personal上でもLXDなどを利用して普通のシステムを構築できるようにするため、通常のパッケージリポジトリはそのまま維持されます(**図5**)。

将来はサーバも?

今のところConvergenceの対象になっているのは、GUIに関わるコンポーネントのみです。しかしながら、Ubuntu Personalのようにシステムイメージレベルでの統合が進んでくれば、サーバのようなCUI環境もSnappy化の波が押し寄せる可能性もあります。とくにUbuntuサーバはJuju・MAASというデプロイツールが存在するので、そちらとの統合がポイントとなってくることでしょう。開発の進み具合によっては、「現在のUbuntu」のLTSリリースは16.04が最後ということになるかもしれませんね。SD

図4　Ubuntu Personalの画面切り替え

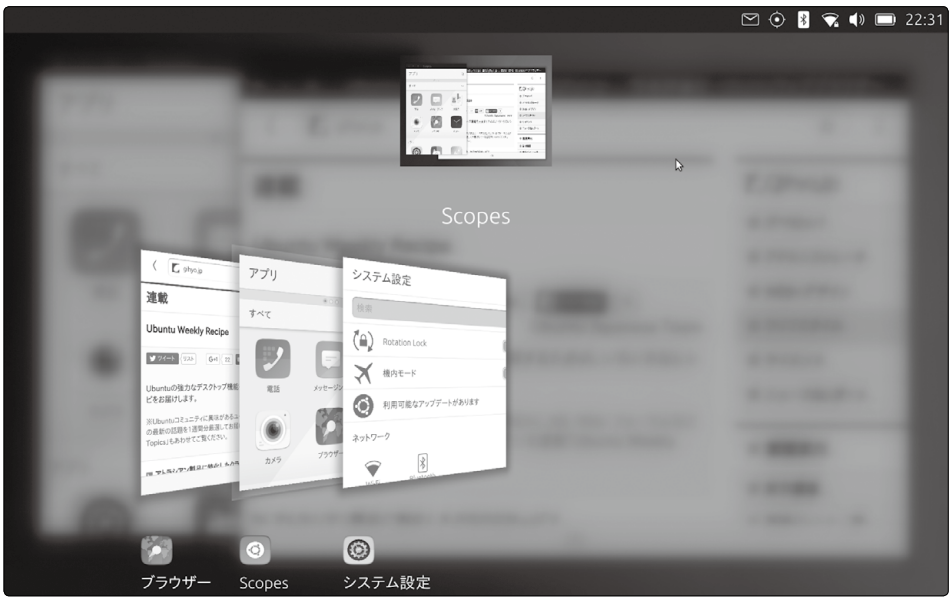
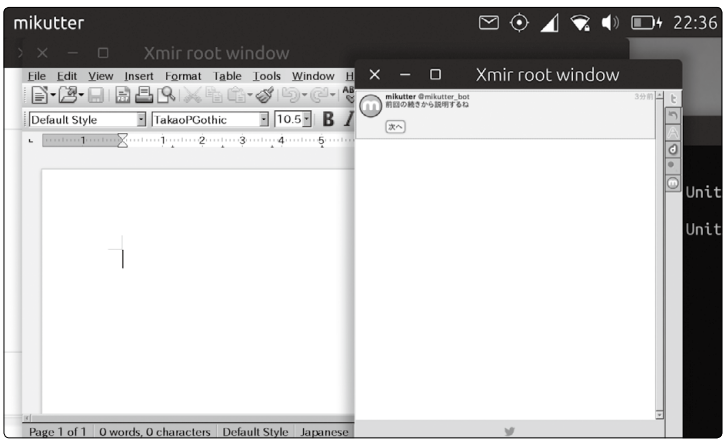


図5　Ubuntu PersonalによってMikutterやLibreOfficeも起動できる



第50回

仮想マシン用の準仮想化デバイスドライバのフレームワーク

virtioドライバのしくみ

Text：青田 直大　AOTA Naohiro

3月14日にLinux 4.5がリリースされ、Linux 4.6の開発が始まっています。3月末ごろまでMerge windowが開いていて、さまざまな新機能が導入されているところです。今回はvirtioドライバがどのようにロードされ、どのようにデータをやりとりするかについて紹介します。

virtioとは

仮想マシンをセットアップしたことがある方は、virtioの名を見かけたことがあるかと思います。virtioというのは仮想マシン用の準仮想化デバイスドライバのフレームワークとみることができます。

仮想マシンのカーネルで実際のデバイス用のドライバを使う場合を考えてみましょう。この方法では仮想マシン側はただ物理デバイスで動いているように動作すればよいのでカーネルの変更を必要としません。しかし、ハイパーバイザ側では実際のデバイスの動きをエミュレートすることになります。これは複雑かつ非効率です。

実際にやりたいことはホスト・ゲスト間でデータをやりとりし、ネットワーク通信やディスクの読み書きなどを行うことだけです。ホスト・ゲストが共同して動作すれば、ホスト側のデバイスの実装はよりシンプルにできます。virtioはこういった仮想デバイスにどういったものがあるのか、どのようにデバイスは振る舞うのか、そしてゲストのドライバはデバイスとどのようにデータをやりとりするのかを定めています[注1]。

virtioデバイスにはネットワークカードやブロックデバイスといったお馴染みのものから、コンソールや9pプロトコルを使ったファイル共有、それぞれLinux 4.1や4.2で導入された入力デバイス(キーボードやマウス)・GPUなどさまざまなものがあります。こうしたデバイスがどのように初期化され、どのようにデータをやりとりし、動作しているのかを見ていきましょう。

virtio PCI

まずはvirtio deviceの初期化を見ていきましょう。ゲストにとってvirtio deviceはPCIのデバイスとして見えます[注2]。lspciコマンドで見てみると、"00:02.0"にネットワークカードが、"00:04.0"と"00:06.0"にブロックデバイスが、"00:05.0"にメモリバルーン用のデバイスが、そして"00:08.0"と"00:09.0"とに9pfsによるファイル共有用通信経路とな

注1）http://docs.oasis-open.org/virtio/virtio/v1.0/csprd01/virtio-v1.0-csprd01.pdf

注2）PCIがないような場合にはMMIOなど、ほかの手段が使われます。

るデバイスが挿さっていることがわかります(**図1**)。

PCIデバイスにはconfiguration spaceというものがあり、ここの情報をもとにデバイスを識別し、適切なドライバを読み込んでいます。virtioのネットワークデバイスとブロックデバイスについて、どのようなconfigurationになっているかと、どのドライバが読み込まれているかを"lspci -kx"で見てみましょう(**図2**)。

最初の2byteがVendor IDでどちらも0x1af4となっており、次の2byteがDevice IDでネットワークデバイスでは0x1000で、ブロックデバイスでは0x1001となっています。このようにVendor IDが0x1af4でDevice IDが0x1000から0x103fまでのデバイスがvirtio deviceとして認識されます。さらに、Device IDによってそのデバイスがどのような種類のデバイスであるか、すなわちネットワークデバイスであるのか、ブロックデバイスであるのかなどが識別されます。

図2の"Kernel driver in use"の行を見るとわかるように、どちらのデバイスに対しても"virtio-pci"のドライバが読み込まれています。では、2つのデバイスの違いを処理するはずの"virtio_net"ドライバと"virtio_blk"ドライバはどこでロードされているのでしょうか。virtio-pciのprobe関数を追って、固有のドライバのロード部分を見ていきましょう。

virtio driverのロード

PCIサブシステムからvirtio pciのprobe関数が呼び出されるところから見ていきましょう。PCIのドライバはid_tableというものを持ち、ここにそのドライバがサポートするデバイスのVendor IDとDevice IDとを記述しています。PCIサブシステムはこのテーブルを参照して、PCIデバイスに

▼図1　PCIデバイス一覧

```
# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 Ethernet controller: Red Hat, Inc Virtio network device
00:03.0 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #1 (rev 03)
00:03.1 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #2 (rev 03)
00:03.2 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #3 (rev 03)
00:03.7 USB controller: Intel Corporation 82801I (ICH9 Family) USB2 EHCI Controller #1 (rev 03)
00:04.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:05.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:06.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:07.0 VGA compatible controller: Cirrus Logic GD 5446
00:08.0 Unclassified device [0002]: Red Hat, Inc Virtio filesystem
00:09.0 Unclassified device [0002]: Red Hat, Inc Virtio filesystem
```

▼図2　PCIのconfiguration spaceのダンプ

```
# lspci -kx -s 00:02.0
00:02.0 Ethernet controller: Red Hat, Inc Virtio network device
        Subsystem: Red Hat, Inc Device 0001
        Kernel driver in use: virtio-pci
00: f4 1a 00 10 07 05 10 00 00 00 00 02 00 00 00 00
10: 01 c1 00 00 00 00 bd fe 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 f4 1a 01 00 # Subsystem Vendor ID, Subsystem ID
30: 00 00 b8 fe 40 00 00 00 00 00 00 00 0a 01 00 00

# lspci -kx -s  00:04.0
00:04.0 SCSI storage controller: Red Hat, Inc Virtio block device
        Subsystem: Red Hat, Inc Device 0002
        Kernel driver in use: virtio-pci
00: f4 1a 01 10 07 05 10 00 00 00 00 01 00 00 00 00
10: 01 c0 00 00 00 20 bd fe 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 f4 1a 02 00 # Subsystem Vendor ID, Subsystem ID
30: 00 00 00 00 40 00 00 00 00 00 00 00 0b 01 00 00
```

対応するドライバを発見しprobeを呼び出します。virtio-pciの場合、**リスト1**のように"Vendor ID = 0x1af4"でDevice IDはどんなものでも対応するというテーブルになっています。先ほど見たようにVendor IDがテーブルに適合するので、virtio-pciのvirtio_pci_probe()関数が呼び出されます。

virtio_pci_probe()はVirtio PCIデバイスの情報を管理する"struct virtio_pci_device *vp_dev"をallocateします(**リスト2**)。

virtio_pci_modern_probe()関数(またはvirtio_pci_legacy_probe()関数)で、データの設定を行います。virtio_pci_modern_probe()ではDevice IDが前述したVirtIOデバイスの範囲であるかを確認し、"vp_dev->vdev.id"を設定しています。ここで"vp_dev->vdev"は"struct virtio_device"でvirto deviceの情報を管理する構造体です。設定されている"subsystem_vendor"と"subsystem_device"とは先ほどのPCIconfiguration spaceの0x2cからの2byteと0x2eからの2byteとにそれぞれ対応しています。すなわち、"id.vendor"にはどのタイプのvirtioデバイスでも0x1af4が設定される一方で、"id.device"にはデバイスの種類によって違う値が設定されます。たとえば、ネットワークデバイスでは"1"が、ブロックデバイスでは"2"が設定されることになります。

"vp_dev->vdev"の設定が終わったところで、register_virtio_device()を呼び出してVirtioデバイスをシステムに登録していきます(**リスト3**)。この関数の中では、Virtioデバイスのbusをvirtio_busに設定して、device_register()を呼び出します。device_register()から先は一般的なデバイス登録の関数になります。

device_register()はごく小さな、device_initialize()で

▼リスト1　VirtIO PCIのIDテーブル(linux/drivers/virtio/virtio_pci_common.c)

```
static cons
t struct pci_device_id virtio_pci_id_table[] = {
    { PCI_DEVICE(0x1af4, PCI_ANY_ID) },
    { 0 }
};
```

▼リスト2　virtio-pciのprobe関数(linux/drivers/virtio/から)

```
static int virtio_pci_probe(struct pci_dev *pci_dev,
                const struct pci_device_id *id)
{
    struct virtio_pci_device *vp_dev;
    int rc;

    /* allocate our structure and fill it out */
    vp_dev = kzalloc(sizeof(struct virtio_pci_device), GFP_KERNEL);
…
        rc = virtio_pci_modern_probe(vp_dev);
        if (rc == -ENODEV)
            rc = virtio_pci_legacy_probe(vp_dev);
        if (rc)
            goto err_probe;
…
    rc = register_virtio_device(&vp_dev->vdev);
    if (rc)
        goto err_register;

    return 0;
…
}

int virtio_pci_modern_probe(struct virtio_pci_device *vp_dev)
{
    struct pci_dev *pci_dev = vp_dev->pci_dev;
…
    /* We only own devices >= 0x1000 and <= 0x107f: leave the rest. */
    if (pci_dev->device < 0x1000 || pci_dev->device > 0x107f)
        return -ENODEV;

    if (pci_dev->device < 0x1040) {
        /* Transitional devices: use the PCI subsystem device id as
         * virtio device id, same as legacy driver always did.
         */
        vp_dev->vdev.id.device = pci_dev->subsystem_device;
    } else {
        /* Modern devices: simply use PCI device id, but start from 0x1040. */
        vp_dev->vdev.id.device = pci_dev->device - 0x1040;
    }
    vp_dev->vdev.id.vendor = pci_dev->subsystem_vendor;
…
}
```

データの初期化を行ったあと、device_add()を呼び出しているだけの関数です。device_add()の中では/sys下のファイルの作成を行う関数などが呼び出されています。抜粋している2行がvirtioデバイスのドライバが読まれる処理に関連するコードです。

kobject_uevent()は登録されたデバイスについて"uevent"をuserlandに通知します。virtioデバイスでは(virtio busによって)"add_uevent_var(env, "MODALIAS=virtio:d%08Xv%08X", dev->id.device, dev->id.vendor);"のコードが実行され、uevent に"MODALIAS=virtio:dXXXXXXXXvXXXXXXXX"といった形式の値が設定されます。"dev->id.device"と"dev->id.vendor"が引数にとられているように、この値はドライバの識別に使うことができます。userlandのプログラムは、このイベントを受け取り、この値とリストのような/lib/modules/<kernel version>/modules.alias(**リスト4**)を参照して、デバイスに対応するモジュールを発見し、ロードします。

一方のbus_probe_device()は、デバイスとドライバとを結びつける役割を果たします。virtio_netなどのvirtioドライバもPCIドライバと同じようなIDテーブルを持っています。virtio busはこのテーブルに指定されたIDと、先ほどの"virtio_pci_modern_probe()"で設定したIDが一致するかを調べて、一致する場合ドライバのprobe関数を呼び出します。ここまででやっとvirtio_netやvirtio_blkなどの固有のドライバのコードにたどり着くわけです。

▼リスト3　virtio デバイスの設定と追加

```
Descriptor Table
Desc Indirect  Desc Indirect
Available Ring

int register_virtio_device(struct virtio_device *dev)
{
    int err;

    dev->dev.bus = &virtio_bus;
…
    /* device_register() causes the bus infrastructure to look for a
     * matching driver. */
    err = device_register(&dev->dev);
…
}

int device_add(struct device *dev)
{
…
    kobject_uevent(&dev->kobj, KOBJ_ADD);
    bus_probe_device(dev);
…
}
```

▼リスト4　module.alias

```
alias virtio:d00000001v* virtio_net
alias pci:v00001AF4d*sv*sd*bc*sc*i* virtio_pci
alias virtio:d00000005v* virtio_balloon
alias virtio:d00000012v* virtio_input
```

virtioドライバのprobe

virtioドライバによるデバイスの初期化について見ていきましょう。ここでドライバがやるべきことはもちろんドライバの種類によって異なっていますが、設定の読み込みと、データの通信経路であるvirtqueueの作成の2つになります。

設定の読み込みは、たとえばネットワークデバイスにおけるMACアドレスなどを設定します。これらの値はI/O空間からロードされます。

virtqueueはその名のとおりデータ通信に使われるキューです。ドライバは、デバイスのI/O空間からキューのサイズや数といった設定を読み込み、ゲストのメモリ空間にキューのための領域を確保します。

virtqueueの構造

virtqueueのメモリ領域は3つの領域に分かれています。1つめはDescriptorTableという領域で転送データがあるバッファのアドレスや長さを記述する部分です。2つめはAvailable Ringでデバイスが使うことができるバッファをDescriptor Tableのエントリを参照して指定しています。3つめはUsed Ringでデバイスが使用したバッファ

をAvailable Ringと同様に指定している領域です。

デバイスとドライバはそれぞれこれらの領域を読み書きし、また適宜I/O空間への書き込みや、割り込みによってRingの更新を相手に通知することで、データのやりとりを行います。

inputの送受信

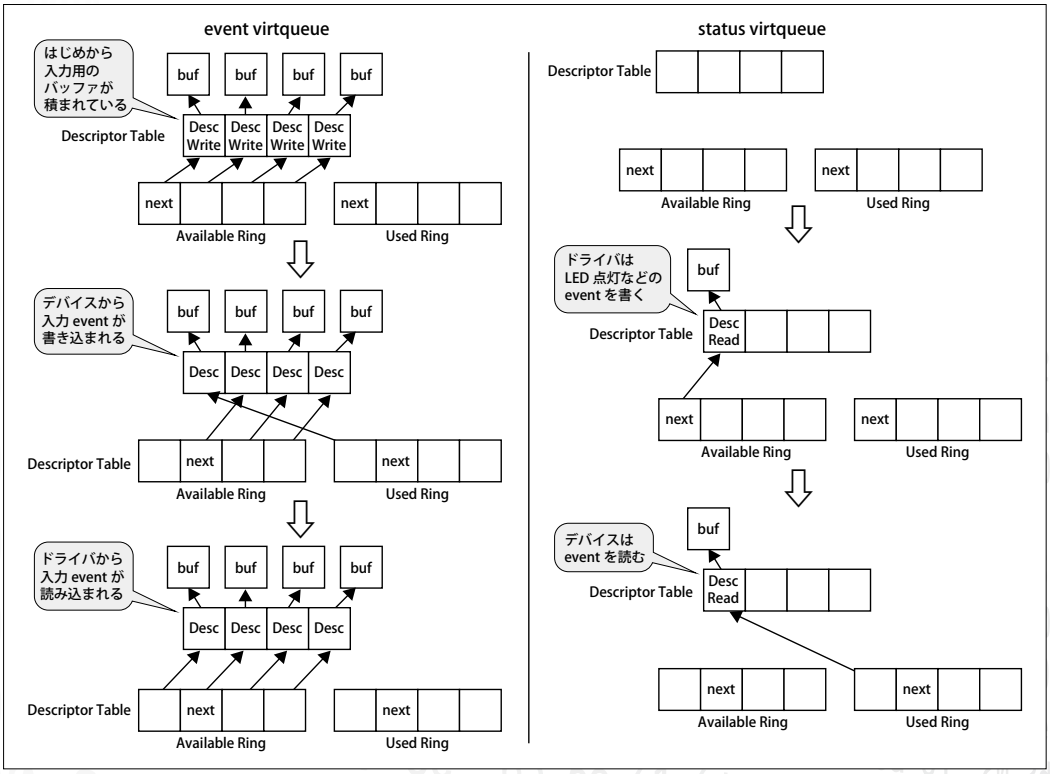
具体的にvirtio inputでのデータのやりとりを見ていきましょう(**図3**)。virtio inputはキーボードやマウスといった入力デバイスのためのvirtioデバイスです。Linuxでの汎用的な入力デバイスのフレームワークであるevdevのイベントをそのままvirtioで転送するので、evdevに対応しているデバイスであればなんでもエミュレートできますし、evdev対応デバイスのパススルーも可能となります。

virtio inputは2つのvirtqueueを使います。1つはevent用のvirtqueueで、デバイスからドライバへと入力のevdevが流れます。もう1つはstatus用のvirtqueueで、ドライバからデバイスへのevent(LEDの点灯やサウンドの再生など)が流れます。

まずはevent virtqueueのほうから見ていきましょう。こちらはデバイスからデータが送られてくるほうですので、まずすべてのバッファをデバイスからの書き込み専用にしてAvailable Ringに登録します。キーボードのキーが押されるなどのeventが発生すると、デバイスはAvailable Ringからバッファを1つ使ってeventを格納し、Used RingにDescriptorを登録して、割り込みを発生させます。割り込みが起きると、ドライバのコールバック関数が呼び出されます。コールバックの中では、Used Ringを参照してeventが書き込まれたバッファを取り出し、ゲスト内にevdevのeventを流してから、バッファを再びAvailable Ringへと戻します。

次に、読み書きが逆になるstatus virtqueue

▼図3　virtio inputでのデータのやりとり



を見てみましょう。LEDの点灯などのstatus eventが発生するとゲストのevdev subsystemからドライバのコールバックが呼び出されます。コールバックはバッファにデータを埋めて、デバイスからの読み込み専用のバッファとしてAvailable Ringに登録し、デバイスに新たなイベントが入ったことを通知します。通知を受けたデバイスはAvailable Ringを参照して、バッファからデータを読み込みLEDを点灯するなどの処理をして、Used RingにDescriptorを登録します。ここでstatus virtqueue用のコールバックも呼ばれますが、こちらはただバッファを読み捨てるだけの関数です。

chainとindirect descriptor

virtio inputではDescriptor 1つに対してバッファを1つ使用していました。virtio blkでは、より高度なDescriptorの機能であるbuffer chainとindirect tableを使用しているのでそれについても紹介します(**図4**)。

virtio blkはブロックデバイスの入出力を行うvirtioデバイスです。ブロックの入出力においては、最低でも読み書きしたいセクタなどの情報と、読み書きデータ用バッファと、I/Oの状態を返してくるバッファと3つのバッファを一度のI/Oリクエストでやりとりします。このような複数のバッファを一度にやりとりしたい場合にbuffer chainが使われ、さらにDescriptor Tableの節約のためにindirect descriptorが使用されることもあります。

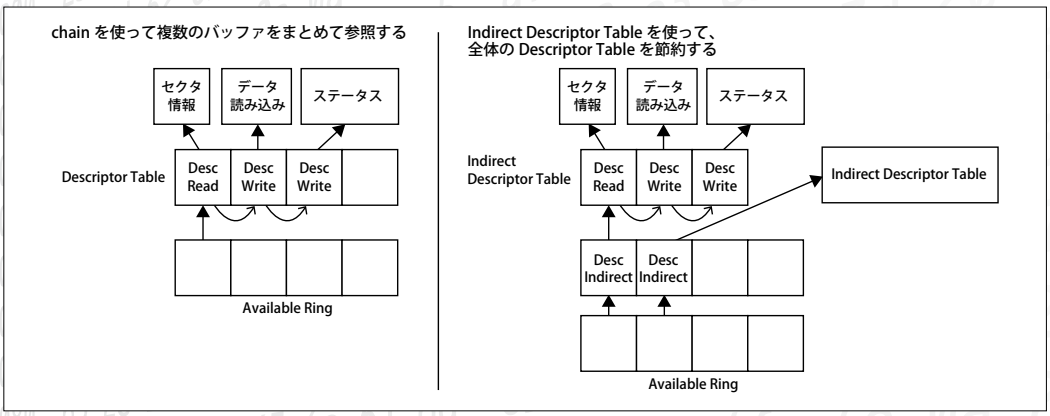
chainとはDescriptorのnextフィールドでほかのDescriptorを参照し、複数のバッファをまとめる機能になります。Ringは、こうしたchainの先頭を参照していて、デバイスやドライバはnextをたどって複数のバッファを認識することになります。

indirect descriptorは、Descriptor Tableから直接データをやりとりするバッファを指定するのではなく、もう一段Descriptor Tableとなるバッファを指定して、一段目のDescriptor Tableを節約する機能です。Descriptor Tableが埋まるとそれ以上の通信はできなくなるので、複数のバッファを高頻度にやりとりするような場合にはこの機能が役に立ちます。

まとめ

virtioはOASISで標準化もされてOSやハイパーバイザに依存しないホスト・ゲスト間の通信経路としてネットワークやブロックデバイスを超えて入力デバイスやGPU、乱数発生源などさまざまな用途で使われるようになっています。今回はvirtioデバイスがどのようにゲストに見えて、どのようにデータをやりとりしているかについて紹介しました。SD

▼図4　buffer chainとindirect table



May 2016

NO.55

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp
りゅうちてつや　RYUCHI Tetsuya　ryuchi@ryuchi.org

寒中シェル芸勉強会

今回は、2015年12月と2016年2月に行った、シェル勉強会の模様をお伝えします。

2015年12月開催　シェル芸勉強会

■jus・USP友の会共催シェル勉強会
【日時】2015年12月26日（土）10:00～19:00
【会場】㈱KDDIウェブコミュニケーションズ

昨年12月26日に、USP友の会と共催でシェルスクリプトおよび周辺技術についての初心者向け勉強会を開催しました。年末の時期にもかかわらず、46名の参加がありました。午前はおもに初心者向けにテーマを絞ったセミナーを2本、午後はシェル芸勉強会という構成でした。

■「もっと簡単にAWKを使おう - 10文字プログラミング」

斉藤博文（日本GNU AWKユーザー会）

最初は斉藤さんによるAWKに関するセミナーでした。AWKはシェルスクリプトだけでなく、日常的によく使われるコマンドです。今回はこのAWKの使い方を基本から解説されました。また、恒星や星座のデータをサンプルとして使用し、AWKのさまざまな技法を実際のプログラム例を見せながら紹介されました。斉藤さんの資料は次のURLにあります。

- http://gauc.no-ip.org/20151226_usptomo_presentation_handout.pdf

■「第1回man bash読書会 展開（expansion）編 - シェル芸を支える7つの展開-」石井久治

続いて石井さんが、bashの機能をmanを使いながら調べる方法について紹介されました。はじめにbashコマンドのマニュアルの構成を説明したあと、今回は展開（expansion）の章に絞って解説されました。展開は、コマンド行に書かれた文字列を単語に分割したあとに行うもので、たとえばa{d,c,b}eはade ace abeに展開されます。展開にはブレース展開やパラメータ展開など全部で7種類あります（図1）。一部の展開については演習問題を用意し、参加者が実際にプログラムを書いて動作を確認しました。石井さんの資料は次のURLにあります。

- http://bit.ly/manbash20151226

■「シェル芸勉強会」上田隆一（USP友の会）

午後の部は、USP友の会がいつも開催している

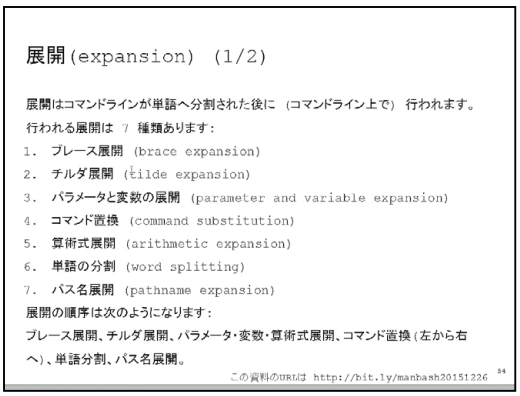


図1　展開の種類（石井久治氏のスライド資料 p54 より）

シェル芸勉強会を行いました。講師の上田会長による眠気覚ましを含んだスライドから始まり、ご自身の近況についても報告がありました。そのあとシェル芸についての説明や進め方、問題を解くときの着目点などについて一通りの説明がありました。その後、実際に問題に取り組みましたが、今回は全部で8問の課題がありました。数字のソート、端末の操作、最大公約数を求める、漢数字からアラビア数字への変換などが出題されました。上田さんの資料は次のURLにあります。

● https://blog.ueda.asia/?p=7332

■懇親会

午後の部が終了したあとは、同じ部屋で有志による懇親会を行いました。懇親会はビアバッシュ形式で、用意されたビール、ソフトドリンクやピザ、スナック菓子などをつまみながら、自己紹介が行われました。また、ライトニングトーク形式の発表も行われ、お互いに楽しい時間を過ごすことができました。

2016年2月開催　シェル芸勉強会

■jus・USP友の会共催シェル勉強会
【日時】2016年2月13日（土）10:00～19:00
【会場】㈱KDDIウェブコミュニケーションズ

12月に引き続き2月にも、USP友の会と共催でシェルスクリプト関連の勉強会を開催しました。今回は44人が参加しました。構成は前回と同じで、午前は初心者向けセミナーを2本、午後はシェル芸勉強会を行いました。

■「正規表現」今泉光之（USP友の会）

セミナーの1本目は今泉さんによる正規表現の講座でした。sed、grep、trなどのコマンドでUNIXの初期から利用されている基本正規表現と、awkやegrepなどで利用される拡張正規表現について、それぞれで使えるメタキャラクタを一通り説明されました。その後、実際によく使われる正規表現の実例を、単純な文字列にマッチする簡単なものから、任意のURLやメールアドレスにマッチする長大な正規表現に至るまでを解説されました。

■「シェルがコマンドを実行する前にしていること」
鳥海秀一（USP友の会）

2本目のセミナーは、シェルのコマンド処理の様子を鳥海さんに解説していただきました。シェルがコマンド行に入力された文字列を解釈して実行するまでの処理を、トークン分割、エイリアス展開、その他の展開（ブレース展開やチルダ展開など）、コマンド検索に分けて順番に解説されました。また、講師からは「手を動かすことが技術習得の近道である」というメッセージがあり、それを実践するためにコマンド検索の優先順位を確認する課題に取り組みました。

■「シェル芸勉強会」上田隆一（USP友の会）

午後の部は恒例となっている上田さんによるシェル芸勉強会です。今回も8問の課題が用意され、1問あたり15分ぐらいで解答案を考えたあとに、上田さんによる問題解説と解答例が示されました。今回の問題は、PDFからのテキストの抽出、Shift JISの固定長データをUTF-8に変換、日曜日の列挙、シェルスクリプトのデバッグ、拡張正規表現を基本正規表現に変換、添付ファイルの抽出と画像の復元、などでした。上田さんの資料は次のURLにあります。

● https://blog.ueda.asia/?p=7608

■終わりに

シェル勉強会は毎回好評で、参加登録ページを公開すると早々に多数の申し込みがあります。継続的に開催していることにより、知名度が上がっている効果だと思います。こうした活動を続けることで、シェルに慣れ親しむ人が増えるのは喜ばしいことです。jusとしても引き続き協力していく予定です。SD

Hack For Japan
エンジニアだからこそできる復興への一歩

Hack For Japan

第53回 NPOが抱える課題にITはどこまで協力できるか

●Hack For Japanスタッフ
清水 俊之介　SHIMIZU Shunnosuke
Twitter @donuzium

2月20日から21日にわたって、仙台市にて日本NPOセンター主催の「情報・ICT利活用を通じた組織基盤強化・課題解決型研修」が行われました。技術を人に届けるためにはどうすればいいのか。エンジニアとして問答した2日間をレポートします。

まずは課題と向き合う

おもに東日本大震災の被災三県である宮城・福島・岩手を中心に、その他全国各地の団体で活動している方々が、ICTの分野でできること・気をつけることなどを理解しながら、2日間にわたって課題解決方法を探ることを目的とした今回の研修。この研修は武田薬品工業株式会社の寄付により日本NPOセンターが行っている「タケダ・キャパシティ・イニシアチブ」の一環として開催されました。このプログラムでは東日本大震災の復興を目的に、NPOなどの組織基盤強化のための支援を行っています。

日本NPOセンターとはNPOの基盤強化の支援を行う中間支援組織で、本研修では、ICTを活用してNPOの支援を行う「NPOのためのICT支援者ネットワーク」と岩手県、宮城県、福島県の県域のNPO支援センター、発災後ICTを使って現地を支援されていた方からのアドバイスをとりまとめ、練られた企画とのことです。

1日目はそれぞれの団体が抱える問題を見つめなおすため、マンダラート[注1]などの手法を用いながらディスカッションを行いました（**写真1**）。まず現状の問題点を書き出し、その問題に対処できず放置するとどうなるのかを広げていく形でシートを埋めていきます（**写真2**）。

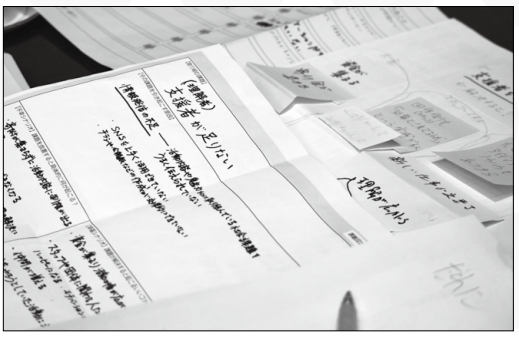
筆者と同じグループになったのは、日本NPOセンターの中川馨さん、南三陸町の一般社団法人さとうみファームの金藤克也さん、そして一般社団法人ワタママスマイルでの菅野芳春さん。金藤さんはおもにスタッフ間の「情報共有」を、菅野さんは団体の「情報発信」をそれぞれ課題として挙げていました。

共通して話題になったのはICTに強い人材がいないということ。2団体ともにWebサイトは持っていますが、管理できるスタッフがもっと多ければより頻繁に更新できると話していました。過去にボランティアでブログやサイトを作ってくれた人はいたものの、継続して情報発信することの難しさを日々実感しているようでした。

注1　3×3に9分割したマスを用意し、中央に主となるテーマを書き込み、その周りにそれに起因したものを書き込んでいくルールで発想を広げていく手法。

◆写真1　会場の様子

◆写真2　課題を整理するためのシート



情報共有

そのほかのグループでも課題として多く挙げられていた「情報共有」という問題。組織を運営して時間が経つとともに、どうしても皆さんこの問題にぶつかってしまうようです。NPOや一般社団法人という形態に限らず、どんな組織においても障壁になる大きな問題だと思います。そして多くの情報共有ツールはネット利用を前提としたものであり、1人1台PCを持たない団体では導入が難しいという話も。

そんな話を聞かせてくれたのはNPO法人にじいろクレヨンで働く中田心さん。この団体は石巻市で活動をしており、最初は避難所や仮設住宅に住む子どもたちのケアからスタートし、現在では自分たちの施設を持って、いわゆる学童クラブのような事業も行っています。中田さんは団体の中では「ITに強い」ということで広報としてメールマガジンの配信なども担当されているそうですが、本人曰く「あまり得意ではない」とのこと。

にじいろクレヨンはパートを含めて17名規模の団体ということですが、事業と拠点が増えていくに連れコミュニケーションに齟齬が生まれてきているようです。たとえばスタッフのシフト管理にはExcelを用い、そのファイルをDropboxで共有していたため、オンラインで複数人が同時編集できるGoogleスプレッドシートを提案したところ「PCを全員持っているわけではない」「セキュリティが不安」という2つの問題が挙がりました。ただ話を聞いてみるとスタッフはほぼスマートフォンを持っているようで、実際にスマートフォンからスプレッドシートを使っている様子をその場で見せると、少し興味を持ってもらえたようです。しかし同時に「難しい」という印象も持たれたようで、今後ICTの分野で支援を続けるに当たっては、こういった障壁を取り除いていくことも課題の1つです。

羊が死ぬ

これはアイデアソンで一緒のグループになった金藤さんが、自分たちが抱える情報共有の問題が放置されると最終的にどうなるかという結果の例として挙げたものです。筆者も同じ問題で想像してシートを埋めていきましたが、これほど具体的な結果は出てきません。本当に現場で必要とされている声には重みがあります。

待ちに待った夕食の時間になり、仙台ではどんな美味しいものが食べられるのかと考えながら指定された場所に集合すると、エンジニアチームの目の前にごっそりと置かれた紙の束が。アイデアソンでの課題の掘り下げやアイデアがまとめられたシートで、参加された方の数だけ紙はあります。この課題やアイデアを元にハッカソンでアプリケーションを作ることが目的です。そんなことで少しだけ楽しみだった時間を忘れ、食事をしながら1日目の成果を確認していく時間になりました。その後夜に改めてラーメンを食べに行ったのはさておき。

求められるアプリ

石巻市から来たイトナブの津田恭平さんと一緒に内容を見ていくと、各団体が抱える切実な問題や必死に練られたアイデアなどがびっしりと。エンジニアとして期待されていることがとても伝わり、紙をめくるごとに気合が入ります（**写真3**）。

そしてシートを見ている間、筆者らにはわからないNPOなどが持つ特有の事情を、横について丁寧に解説してくださったのは一般社団法人ICTCカウンシルあおもりの三澤章さん。三澤さんからは問題を解決できそうな、すでにあるNPO向けのサービ

◆**写真3　悩める津田さん**

スなどのガイドもしてもらいました。

資金面の課題

シートに書かれたアイデアはとても現実的で、すぐに実装できそうなものが多くありました。当初はこの中から実現できそうなアプリを開発しようとしていましたが、課題のほうには挙げられていた「資金」という問題についてのアイデアがあまり見つけ出せず、エンジニアとして資金問題をサポートするアイデアを捻り出すことに方針を変えました。

今回はメンターとして参加していたHack for Japanスタッフでもある小泉勝志郎さんも、自身のクラウドファンディングの経験から資金集めの方法論や重要性をアドバイスしていました。小泉さん曰く「クラウドファンディングを成功させる秘訣は、資金集めのために新たなプロジェクトを立ち上げるよりも、すでにやっていることに対して支援してもらうという考え方が大切」とのこと。本来行っていないプロジェクトを無理やり立ち上げてしまうと、手に余るばかりか失敗したときに余計に苦しくなってしまうというアドバイスを送っていました。

ありがとうまでの時間

現場ではさまざまなことが起こり、人手も充分でない。するとデスクワークは後でまとめてやることになり、自分たちの団体を訪ねてくれた人々へのフォローに時間がかかってしまう。この問題を解決することで資金集めの問題に取り組もうと、津田さんと株式会社セカイネットの中園良慶さんとの3人でチームになり、2日目の朝はどんなアプリケーションを作るかの話し合いでスタートしました（**写真4**）。

まずは一概に「支援を多く集める」と言っても、どのように行うかという方向性はいろいろあるということを話し合いました。広報を充実させてより自分たちを周知し、支援の輪を広げていく方法もその1つです。一方で支援者になり得る人々とのコネクションを太くしていくことで、より多くの支援を集めることもまた1つだと考えました。

前者の広げていく方法を考えると、人材が充実していれば広報担当者を専任に置き、今よりも多くの人とのコミュニケーションを取ることができるかもしれません。しかしこれまで課題として挙がっていた“人手不足”という状況を鑑みると、適切な方法とは思えませんでした。ブログを書く十分な時間がなく、広報の知識を持った人材も少ないという声を実際に聞いていたこともあり、後者の方法を模索することに。

そこで筆者らのチームでは、名刺をスキャンするとすぐにメールを送れるアプリを作ることになりました。受け取った名刺をスキャンをすることで、自動でメールアドレスを判別し、定型文とともにメールフォームが立ち上がり、編集・送信を行うことができます。できる限りスタッフのタスクを増やさずに、今まで不便だったことを解消することを目的としています。新しいツールを作ったとしても、それが逆に手間を増やす結果になっては本末転倒です。

援助をたくさん受けるために自分たちのところに来てくれた人は、今後大切な支援者となる可能性が非常に高く、現地を訪れたばかりで体験をまだ共有しているタイミングでコミュニケーションを取ることができれば、より自分たちに興味を持ってもらえるかもしれません。そういったタイミングを逃さないうちに、まずその組織のファンになってもらうことをサポートできるツールになればという想いとともに開発が始まりました。

◆**写真4　悩める中園さん**

名刺からメールを瞬時に送る

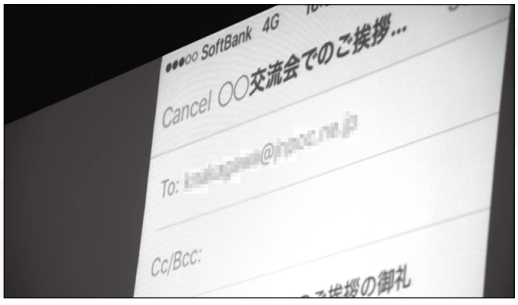
メインの会場でICTの研修が行われている中、別室でエンジニアチームは黙々と作業を続けていました。

アプリのしくみは、まず津田さんが開発を担当するiOSアプリで画像を撮影し、中園さんが担当するWebサーバに投げます。Webサーバでは画像を受け取り、筆者の担当するOCRの処理に投げて返却された解析済みの文字からメールアドレスを抽出し、iOSアプリ側に戻すという流れです。開発が始まるとそれぞれの担当に分かれて静かに作業が進んでいきます。

筆者はOCRの処理を担当するにあたり、いくつかのAPIを使用して精度を比べていきました。無料で使える海外製のものから、国内の企業が開発者向けに提供しているものまで。どれも一長一短で、アルファベットに強いけれど漢字に弱かったり、あるいはその逆だったり。今回はメールアドレスのみの抽出ということで英数字のみ解析できればよかったのですが、ピリオドが日本語の読点や中黒として認識してしまったりと試行錯誤を繰り返しているうちにお昼になってしまいました。

もう時間がないなと思ったときに使い始めたのがGoogleのVision APIでした。Vision APIは非常に優秀で、一般的な名刺であればほぼすべての文字（半角カナも！）を認識していました。テストに用いた名刺でも、認識できなかった個所は6ピクセル程度のピリオド1つのみ。Vision APIは今年に入ってベータ公開されたまだ新しいサービスです。

◆写真5　メールアドレスのスキャンに成功した様子



完成はラスト5分で

メイン会場から駆けつけてくれた小泉さんの力を借りながら、粘って不慣れなiOSと格闘していた津田さん（普段はAndroidの開発がメイン）。終了5分前でなんとかひととおりの流れを確認できました。ハッカソンが終わると、メイン会場へ移動して成果発表に。

デモで読み込んだ最初の名刺では、メールアドレスのアカウント名が途中で切れてしまいましたが、2枚目の名刺では無事成功。会場でお借りした名刺から、メールアドレスだけ抽出し、定型文の入ったメールフォームが立ち上がるところまで無事にデモをすることができました（**写真5**）。

挙がった手

「このアプリを使ってみたい人！」

最後に会場にそう問いかけてくれたのは日本NPOセンターの三本裕子さん。同センターの山本朝美さんと一緒に、かなり内容の深いものとなったこの研修を最初から最後まで支えていました。お陰さまで多くの方々に笑顔で手を挙げていただくことができました（**写真6**）。普段接する機会もなく得体が知れなかっただろうエンジニアという職業の人間が、会場の中で共に課題を解決するための仲間であると受け入れられたような気がしました。

まだ寒い2月の仙台で、笑顔とともに挙げられた手を思い出しながら、一刻も早く皆さんに使っていただけるようにと開発を続けています。SD

◆写真6　笑顔で手を挙げてくれた皆さん

温故知新

IT むかしばなし

第54回

MASM～x86のアセンブラ障壁を越えられるか～

速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami

はじめに

1980年代中盤は、8bitマイコンの黎明期が過ぎ、PC-9801を代表とする16bitのパソコンが主流になりつつ性能アップしましたが、まだ実行速度とメモリの容量に不満が残る時代でした。

MS-DOSが使われるようになると、そのうえで動作するC言語やPascalのコンパイラでプログラミングすることが普通にできるようになり、初期のBASIC言語+マシン語のプログラム環境は廃れていきました。しかし、より高速性やメモリの効率を図るため、マシンを密接に制御できるアセンブラ・プログラムは、まだ歩みを続けていました。そこに大きく立ちはだかったのが16bit CPUであるIntel 8086(以下i8086)のわかり難さの壁でした。今回は、この壁に立ち向かうためのツールであるMicrosoft Macro Assembler(以下MASM)についてのお話をしましょう。

16bitCPU i8086

1978年にインテル社から登場したi8086は、8bit CPUであるIntel 8080との互換性を有することを目指して、同じ名前のレジスタを8bitから16bitに拡張しています。

その1年後にザイログ社から発表されたZ8000は、Z8001/Z8002の2種類があり、Z80とのレジスタの互換性を排除して理想的な16bit CPUを目指していました。Z8002は割り切って、アドレスは16bitのままで64KBのアクセスしかできないものの、Z8001は、7bitのセグメントを上位に加えることで、16bitのオフセットと併せて23bit、8MBのメモリアクセスが可能になっていました。

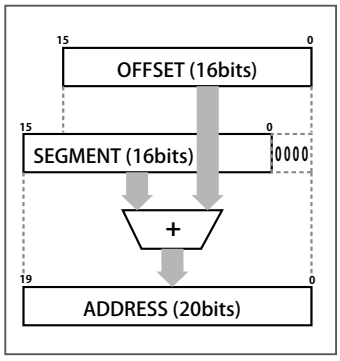
また、モトローラ社は1979年末にMC68000を発表し、データバスは16bitなのですが、データ／アドレスレジスタを4倍の32bitに拡張し、24本のアドレスバスにより当時としては広大な16MBのメモリのアクセスが可能でした。

当時8bitのCPUのアドレスバスは16bitであり、2本の8bitレジスタをペアにするなどして16bitのアドレスレジスタとして使い、メモリアクセスを行っていました。

i8086は、16bitのセグメントと16bitのオフセットを4bitずらして足し合わせることで20bitのアドレスを示し、1MBのメモリ空間にメモリアクセスできるようになっています(**図1**)。しかし、これがくせ者で、1つのセグメント区間は64KBですが、4bitのずれにより16byte配置でセグメントを指定できることで、4,096個(64KB/16)のセグメント区間が重なり合ってしまうのです。セグメントの設定を間違えると、大きなバグが発生することになります。また、i8080から引き継がれた4個のレジスタは、命令により使えるレジスタが制限されるなど汎用レジスタとは呼べないものでした。4つのセグメントレジスタ[注1]をいか

注1) CS：コードセグメント、DS：データセグメント、SS：スタックセグメント、ES：エクストラセグメント

▼図1　セグメントと実アドレス



MASM～x86のアセンブラ障壁を越えられるか～

に管理するかが、i8086のアセンブラ・プログラムのテクニックでした。そのセグメントのテクニックをサポートするツールがMASMだったのです。

MASMとは

MS-DOS上で動作するMASMは、CP/M上のi8080/Z80用マクロアセンブラMACRO-80の延長上にあり、リンク、ライブラリなどの機能を引き継いでいます。

さまざまな疑似命令(いわゆるディレクティブ)を使って、セグメントの設定を自動化します。ただ、その設定状況を把握しておかないと思わぬ動作になってしまいます。例として、PC-9801のVRAMに文字(A～E)を直接転送するプログラムを作ってみました(**リスト1**)。

アセンブラを使う意味

1991年3月号パソコン情報誌「The BASIC」創刊100号(**写真1**)、から筆者の「Advanced Assembler」の連載が始まりました。当時はすでにコンパイラ環境が十分に普及し、「アセンブラでわざわざ苦労してまでプログラミングを行うのか」という風潮もでてきた時代です。しかし、マシンの立場で会話できるアセンブラ言語(以下、アセンブラと略)は、次の点において力を発揮できると考えていました。

❶速度を追及する

アセンブラでは1命令がマシン語の1命令に対応するため、命令のマシンクロックを考えながらスピードのアップをねらったプログラム作成が可能になります。ただ、1993年にPentiumが登場するとスーパースケーラ構造により、1クロックで1つ以上の命令が実行可能となり、そのようなテクニックは難しくなりました。

❷コードサイズの適性化

命令のサイズがわかるため、コードサイズを適切に調整することが可能になります。とくに常駐プログラムやROM内のコードのようにコードサイズを追及するような、プログラム作成では欠かせなかったのです。

❸独特なCPU命令のサポート

i8086→V30→i80286→i80386→i486→Pentiumと続く新しいCPUが持つ新規に加わった命令、機能を利用するためには、アセンブラが必要不可欠でした。

❹データとコードのアドレス設定

デバイスドライバのように、特定のアドレスにデータやコードを配置しなければならない場合においてアセンブラの力を借りる必要がありました。

現在でも、組み込みの分野などでは、上記の点からアセンブラが必要なところもあるはずです。また、コンピュータ科学の分野における学習において、CPUの基本的な理解を深めるためにアセンブラはかかせないものとなっているはずです。ただ、そこで使用するCPUは、わかり難いx86であってはいけないと思います。SD

▼リスト1　PC-9801のVRAMにA～Eを直接表示するアセンブラプログラム

```
TITLE PC9801
 0000                           .MODEL small  ;SMALLメモリモデル
 0000                           .STACK 100h   ;スタックエリアを256バイト確保
 0000                           .DATA         ;データセグメントエリア設定
 0000  41 00 42 00 43 00 44 + msg db 'A',0,'B',0,'C',0,'D',0,'E',0
       00 45 00
 000A                           .CODE         ;コードセグメントエリア設定
 0000  B8 0000s          start: mov     ax,@DATA
 0003  8E D8                    mov     ds,ax     ;データセグメント設定
 0005  B8 A000                  mov     ax,0a000h ;PC9801 VRAMセグメント
 0008  8E C0                    mov     es,ax
 000A  BE 0000r                 mov     si,OFFSET msg
 000D  33 C0                    xor     ax,ax
 000F  8B F8                    mov     di,ax
 0011  B9 0005                  mov     cx,5      ;文字数
 0014  FC                       cld               ;増加方向
 0015  F3> A5                 rep movsw           ;ブロック転送
 0017  B8 4C00                  mov     ax,04c00h ;0:エラーレベルで終了
 001A  CD 21                    int     21h
                                END     start     ;スタート位置ラベル設定
```

▼写真1　TheBASIC

Service

Bluetooth SIG、2つの新サービスと2016年技術ロードマップを発表

3月30日、Bluetoothの規格策定・技術利用に対する認証を行う団体であるBluetooth Special Interest Group（以下Bluetooth SIG）は、新サービスおよび2016年の技術ロードマップを発表した。

Transport Discovery Service

Transport Discovery Service（以下、TDS）は、使用されている無線技術の種類にかかわらず、IoTデバイス間の認識と接続を可能にする共通フレームワーク。デバイスの休止状態中は低電力消費のBluetoothで接続しておき、高消費電力や高帯域幅を特徴とするその他の無線技術が必要になればそちらに切り替える、というしくみを実現できる。本技術は現在、他の標準化機関および提携団体に対して提案がなされている。

Bluetoothインターネットゲートウェイアーキテクチャ

BluetoothゲートウェイはBluetoothセンサーからクラウドへ、そしてクラウドからセンサーへデータを中継するしくみ。本アーキテクチャは、Bluetoothゲートウェイを迅速に作成できる方法を開発者に提供する。本アーキテクチャにより、遠隔地からのBluetoothセンサー監視制御などを簡易に実現できる。

Bluetoothゲートウェイのためのスターターキットは次のURLから入手できる。

・https://www.bluetooth.com/develop-with-bluetooth/developer-resources-tools/gateway

2016年技術ロードマップを発表

Bluetoothは2016年、IoT向けの機能拡張を目的としたアップデートを行う。おもな機能強化は「通信範囲の拡大」「通信速度の向上」「メッシュネットワーク」の3つで、スマートホームや位置情報サービスといった分野での活用が見込まれる。現在のBluetoothの最新バージョンは「4.2」だが、これら機能強化は「4.3」以降で実現されていくとのこと。

CONTACT
Bluetooth Special Interest Group
URL https://www.bluetooth.org

Topic

独立行政法人情報処理推進機構、「つながる世界の開発指針」を公開

独立行政法人情報処理推進機構（IPA）は3月24日、IoT製品の開発者が開発時に考慮すべきリスクや対策を指針として明確化した、「つながる世界の開発指針」を公開した。

本開発指針は、IoTの普及とそれに伴うセキュリティリスクの高まりという背景を受け、2015年8月に産業界や学界の有識者で発足された「ワーキンググループ」が策定した。特定の製品分野・業界に依存しないことを念頭に策定されており、IoTに関連するさまざまな製品分野・業界においての分野横断的な活用が想定されている。特徴は次のとおり。

- IoT製品を開発する企業全体の「方針」の策定、つながる場合のリスクの「分析」、リスクへの対策を行うための「設計」、製品導入後の「保守」や「運用」といった製品の開発ライフサイクル全体において考慮すべきポイントを全17の指針として明示
- それぞれの指針ごとに、取り組むための背景や目的、具体的なリスクと対策の例を解説
- 指針一覧はIoT製品開発時のチェックリストとしても活用が可能。またIoT製品を調達する利用者側においても自社の要件確認時のチェックリストとしても活用が可能
- 開発者に限らず、経営者層がIoT製品に想定されるリスクや対策を自社が取り組むべき課題と認識し、理解を深めてもらうためのガイドとしても活用が可能

資料は以下のURLでPDFとして公開されている。

「つながる世界の開発指針～安全安心なIoTの実現に向けて開発者に認識してほしい重要ポイント～」
・http://www.ipa.go.jp/files/000051411.pdf

▲「つながる世界の開発指針」表紙

CONTACT
独立行政法人情報処理推進機構 URL https://www.ipa.go.jp

Hardware

センチュリー、8インチUSB接続サブモニター「LCD-8000U2B」発売

㈱センチュリーは8インチのUSB接続サブモニター「LCD-8000U2B」を2月23日に発売した。

LCD-8000U2Bは、USBケーブル1本で手軽にデュアルディスプレイができるUSB接続サブモニター「plus one」シリーズの新製品。液晶には、写り込みや反射しにくいノングレア（非光沢）パネルを採用。解像度はSVGA 800×600ピクセルとなっている。電源はUSBバスパワーで、持ち運びの際に液晶を傷付けないための「専用液晶保護カバー」も付属するので、外出先など、持ち運んで使うのにも便利な製品となっている。また、上下左右反転、ミラーリング、回転など、多彩な表示が可能となっている。参考価格は19,224円（税込）。

▲LCD-8000U2B

CONTACT
㈱センチュリー URL http://www.century.co.jp

Report

「攻殻機動隊 REALIZE PROJECT the AWARD」開催

2月11日、渋谷ヒカリエ（東京都渋谷区）にて、人気アニメシリーズ「攻殻機動隊」の近未来テクノロジーを具現化した作品を展示、表彰する「攻殻機動隊 REALIZE PROJECT the AWARD」が開催された。

「攻殻機動隊 REALIZE PROJECT」は「攻殻機動隊」に描かれている数々の近未来テクノロジーの実現を追究するプロジェクト。今回のイベントでは、2015年10月～11月に「義体部門：東京大会」「電脳部門：神戸大会」、「都市部門：福岡大会」の各大会から選出された「攻殻×コンテスト」優秀作品6チーム、「攻殻×ハッカソン」優秀作品4チームの中から、最優秀の2チームが選ばれた。

◎攻殻×コンテスト the AWARD
「攻殻機動隊の義体を支える臓器設計技術」
：横浜市立大学　小島伸彦研究室

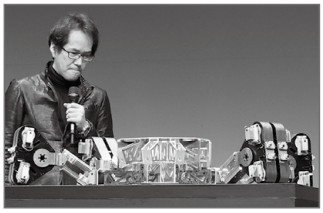
◎攻殻×ハッカソン the AWARD
「空圧式人工筋肉身体防御スーツCyber Protection Suit」
：チームShift

▲左：チームShift、右：小島伸彦研究室

▲攻殻ユニバーシティと攻殻シンポジウムのメンバー（情報通信研究機構 井上 大介氏、神戸大学 塚本 昌彦氏、筑波大学 岩田 洋夫氏、はこだて未来大学 松原 仁氏、小説家 冲方 丁氏、ほか）

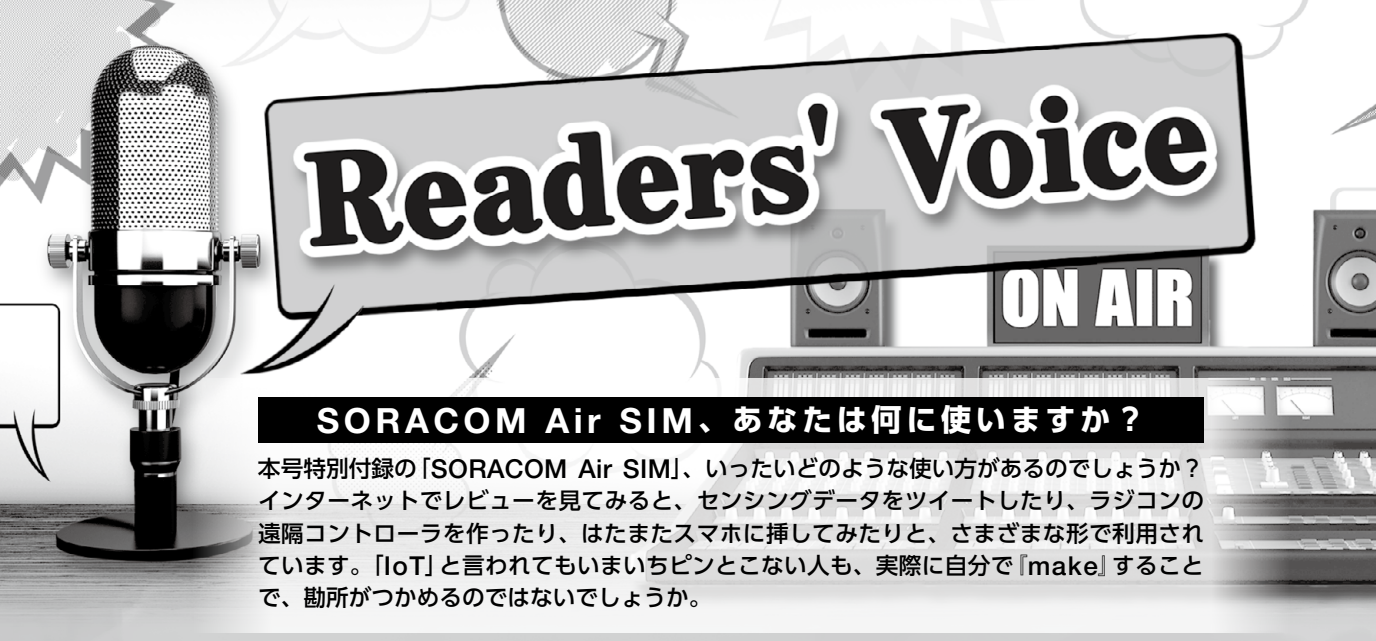
また、当日は大会審査に関わった各大学・大学院の教授陣による、「日頃の研究テーマと攻殻機動隊」をテーマとした特別講義（攻殻ユニバーシティ）および、映画監督の神山健治氏（『攻殻機動隊S.A.C.』シリーズ監督・脚本）と小説家の冲方丁氏（『攻殻機動隊 新劇場版』脚本、「攻殻機動隊ARISE」シリーズ構成・脚本）を迎えたトークショー（攻殻シンポジウム）も催された。

イベントの最後には、作品に登場する多脚思考戦車「タチコマ」を開発する2つのプロジェクトが発表された。1/2スケールのタチコマを開発するのは海内工業㈱×karakuri products。スマホと連携する1/10スケールのタチコマを開発するのは㈱Cerevoだ。



▲1/2タチコマ（プロトタイプ）と神山 健治氏

CONTACT
攻殻機動隊 REALIZE PROJECT
URL http://www.realize-project.jp



SORACOM Air SIM、あなたは何に使いますか？

本号特別付録の「SORACOM Air SIM」、いったいどのような使い方があるのでしょうか？インターネットでレビューを見てみると、センシングデータをツイートしたり、ラジコンの遠隔コントローラを作ったり、はたまたスマホに挿してみたりと、さまざまな形で利用されています。「IoT」と言われてもいまいちピンとこない人も、実際に自分で『make』することで、勘所がつかめるのではないでしょうか。

2016年3月号について、たくさんの声が届きました。

第1特集 チーム開発を回す現場のアイデア

Webサービス／スマホアプリを高頻度でデプロイするには、どのようなチーム作り、環境づくりが必要なのか。SUUMO、Retty、Qiitaを開発・運営するチームに、それぞれの現場で培われたアイデアを訊きました。

現場から出る改善提案って大きいよねっ、て印象。 tekitoizmさん／東京都

IT雑誌と言う枠を超えて役立つスキル。
ジャンボ団さん／長崎県

Qiitaは最近よく利用しているので、興味深い内容で楽しめた。
山添さん／東京都

今までの失敗など、かなりぶっちゃけたことが書いてあって期待以上でした。
G_famiさん／島根県

テスト自動化による開発スピードアップはどこの現場でも必要だと思う。
ほまれさん／千葉県

今回は技術というよりは、システム工学に近いお話でしたね。華やかなイメージのあるWeb業界やアプリ業界ですが、エンジニアが最大限の力を発揮するために、草の根のルール作りがなされていました。

第2特集 あなたの知らないCOBOLの実力

広く使われている割には苦手意識が持たれている印象のCOBOL。そもそもCOBOLとはどのようなニーズから生まれ、どんな特徴を持つ言語なのか。「COBOLを知る」ことからはじめる特集でした。

元COBOLerとしてはとても興味深く読めました。実は手元にCOBOLの環境があるので、たまには触ってみようかな、と思ってます。
ROMEOSHEARTさん／長崎県

連綿と生き残っているんですね。
杉浦さん／愛知県

若者はなんのことかと思うでしょうが、COBOLを特集するなんて斬新です。
目からウロコさん／埼玉県

まさかのCOBOLか、と読む前は思いましたが、筆者の方々の熱い思いが伝わってくる良い記事でした。
山下さん／東京都

第3章は、COBOLだけではなく今でも一般的な開発に通じることではと感じました。目的のための手段であるはずが、手段を適切に変化させていくことができないのは対応が難しいですね。
出玉のタマさん／大阪府

最古のプログラミング言語にも数えられるCOBOLですが、金融を支えるシステム開発においてはまだまだ現役。表紙の「COBOL」の文字に動揺された方が多かったようですが、読んでみるとCOBOLの印象が変わった、という声が多く寄せられました。

一般記事 iPad Proのさきに見えてくるもの

「enchantMOON」の開発者がペンコンピューティングについて、誕生から隆盛、iPad Pro・Apple Pencil発売までの歴史を紐解き、そしてその先の展望を論じた読みもの記事でした。

個人的には、よく考えたら手書きペンを使うことってなかったですね。
クラウド1号さん／京都府

歴史系はおもしろい。大好き。なかなかネットで得られない情報で重宝する。
田中さん／大阪府

3月号のプレゼント当選者は、次の皆さまです

①洗える防水キーボード 「SKB-BS3W」
佐久間捷矢さま（東京都）

②GitHub Tシャツ＆ステッカー
藍ちゃんさま（兵庫県）、ダイオウグソクムシさま（東京都）、清水邦晃さま（兵庫県）、中村夏実さま（大阪府）、binaさま（東京都）

③McAfee LiveSafe
オミオさま（宮城県）

④『Docker 実践ガイド』
匿名希望さま（山形県）、木村勇太さま（鳥取県）

⑤『ネットワーク・デザインパターン』
鈴木浩さま（熊本県）、tomato360さま（東京都）

⑥『リモートチームでうまくいく』
西村駿人さま（神奈川県）、さくらますさま（東京都）

⑦『ITインフラ監視［実践］入門』
横田敏一さま（静岡県）、田代勝久さま（埼玉県）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

ある技術を概観するような内容は個人的に好みでした。 ぴんぐさん／東京都

iPad Proは全然注目していなかったが、記事を読んでほしいと思ってしまいました。 英輔さん／東京都

技術の歴史は、さまざまなエンジニアの試行錯誤を見ることができて興味深いですね。マウス・キーボードに続く次のインターフェースとして、「ペン」が台頭する日は来るのでしょうか？

一般記事　サイトオーナがとるべき行動と注意点

「運営するWebサイトが突然改ざんされた！」というシナリオの下、被害を広げない・再発を防ぐために何をすべきかを考える特別企画。どのような手順を踏むべきか、各手順で注意することは何かを詳細に解説しました。

好きな話題の記事。もう少しボリュームがほしい。 嶋田さん／三重県

この記事をふまえ、社内の運用マニュアルを再検討したい。 加納さん／千葉県

セキュリティ関係の記事は防御や攻撃についてのものが多い印象で、事後の対応についての記事は新鮮でした。 虹コン大好きマンさん／神奈川県

うちの会社ではこういうものをプロジェクトごとに作成させられるが、活字になっていると周囲を説得するのに効果的だ。 わっしゃさん／宮城県

情報セキュリティの管理者として参考になりました。セキュリティ対策はもちろん重要ですが、改ざん被害に遭った際に適切な対応をすることで被害を最小限に抑えることができるので、事前の対策にさっそく取り掛かろうと思います。 サファイアさん／茨城県

読者の間でもセキュリティに対する意識が高まっているようで、「たいへん参考になった」という声が多く寄せられました。Webサイトは企業、あるいは個人の"顔"となる重要なものなので、とくに注意する必要がありますね。

短期連載　クラウド時代のWebサービス負荷試験再入門【4】

ITインフラの中心がオンプレミスからクラウドへ移るに伴って、システムに対する負荷試験も変える必要があります。本連載ではクラウドに載せたWebサービスの「スケーラブル」を担保するための負荷試験について見ていきます。最終回は2月号に続いて、負荷試験の具体的な段取りについてみていきました。

クラウド環境ではオンプレミス以上に見えにくいボトルネックがある。それらを確実に切り分けてテストを実施する手法がよくわかった。 tack41さん／愛知県

だんだんと難易度が上がってきましたが、その分学びがいがあります。 羊毛布団さん／神奈川県

ほかの試験に活かせそうな話が多くためになります。 モモンガさん／滋賀県

「クラウドファースト」という言葉も出現し、クラウドサービスは当たり前の基盤技術となりました。負荷試験に限らず、クラウドの特徴をふまえたうえで、従来の手法をアップデートする必要がありそうです。

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。

SORACOM Air サービス契約約款

(2016年3月31日時点)

第1章　総則

第1条　約款の適用

株式会社ソラコム(以下、「当社」といいます。)は、SORACOM Airサービスに関する本契約約款及びこれに関連する個別規約(以下、総称して「本約款」といいます。)を定め、本約款に基づき締結されるSORACOM Airサービス契約(以下、「本契約」といいます。)に基づき、SORACOM Airサービスを提供します。

第2条　約款の変更

当社は、本約款を変更することがあります。かかる変更を実施する場合、当社は、当社のウェブサイト又は当社が別途定める方法で契約者に対して告知するものとし、当該告知が行なわれた後に契約者がSORACOM Airサービスを利用した場合には、契約者は、かかる変更に同意したものとみなします。本約款が変更された後のサービスに係る料金その他の提供条件は、変更後の約款によります。

第3条　用語の定義

本約款においては、次の用語はそれぞれ次の意味で使用します。

用語	用語の意味
電気通信設備	電気通信を行うための機械、器具、線路その他の電気的設備
電気通信回線	送信の場所と受信の場所との間を接続する伝送路設備
IMEI	International Mobile Equipment Identifier：国際移動体装置識別番号(端末識別番号)
VPG	閉域網等接続サービスに係る電気通信回線との接続を行うために当社が設置する接続点である仮想ゲートウェイ

第2章　サービスの種類等

第4条　サービスの種類

SORACOM Airサービスには、次の種類があります。

種類	内容
s1プラン	端末と当社間の上下の通信速度が対称なデータ通信サービス

第5条　サービスの提供区域

SORACOM Airサービスの提供区域は、日本国の全ての地域とします。ただし、個別規約において別段の定めが規定されている場合にはこの限りではありません。また、その提供区域内であっても電波の伝わりにくいところでは、SORACOM Airサービスを利用することができない場合があります。

第3章　本契約の締結

第6条　契約の単位

当社は、1の申込者と1の本契約を締結します。

第7条 アカウント

1. SORACOM Airサービスを利用するためには、契約者は、有効な電子メールアドレスに関連づけたアカウント(以下、「ソラコムアカウント」といいます。)を作成しなければなりません。サービス条件で明示的に認められている場合を除き、契約者は一つの電子メールアドレスにつき、一つのソラコムアカウントのみ作成することができます。

2. 当社は、契約者に対し、前項に基づき作成されるソラコムアカウントに当社が提供するシステムにログインするためのIDであるログインID(以下「本ログインID」といいます。)及びログインパスワード(以下「本ログインパスワード」といいます。)を付与します。

3. 契約者は、自己の責任において本ログインID及び本ログインパスワードを管理するものとし、本ログインID及び本ログインパスワードを第三者に貸与、譲渡若しくは使用許諾又は第三者の利益のために使用してはならないものとします。また、契約者は、ソラコムアカウントの不正使用若しくはそのおそれを認識した場合又はソラコムアカウント情報の紛失若しくは盗難があった場合、直ちに当社にその旨通知するものとします。

4. 契約者は、自らのソラコムアカウントに基づき生じるあらゆる事象につき、かかる事象が契約者、契約者の役員若しくは従業員、又は第三者による不正使用若しくは誤使用のいずれによるものかを問わず一切の責任を負うものとし、かかるソラコムアカウントの使用に基づき当社に損害が発生した場合、当社は、契約者に対し、当該損害の賠償を請求できるものとします。また、当社は、かかるソラコムアカウントの使用に基づき契約者に損害が生じた場合であっても何らの責任も負担しないものとします。

第8条　申込の方法

SORACOM Airサービスの利用申込者(以下、「申込者」といいます。)は、本約款を承認した上で、当社所定の手続に従ってオンラインサインアップによる申込(以下、「申込」といいます。)行うものとします。

第9条　申込の承諾

1. 当社は、申込があったときは、これを承諾するものとします。

2. 当社は、前項の規定にかかわらず、通信の取扱上余裕がないときは、その申込みの承諾を延期することがあります。

3. 当社は、申込者に対して、申込者がSORACOM Airサービスの提供に関し負担すべき金額の支払いを怠るおそれがあるか否かを当社が判断するために必要な情報の提出を求めることがあります。

4. 前項の規定により当社が提出を求める情報のうち、貸借対照表及び損益計算書等財務の状況を示すものとして当社が別途定める情報の提出を求められた申込者は、その情報を書面により速やかに当社に提出することを要するものとします。

5. 当社は、第1項の規定にかかわらず、次に掲げる事由に該当する場合には、当該申込を承諾しないことがあります。

(1) 申込者が本契約上の債務の支払を怠るおそれがあると当社が判断したとき。

(2) 申込者に対するSORACOM Airサービスの提供により、当社の事業運営上支障が生じるなど当社の信用又は利益を損なうおそれがあると当社が判断したとき。

(3) SORACOM Airサービスに係る他の契約者の利益を損なうおそれがあると当社が判断したとき。

(4) 申込者に対するSORACOM Airサービスの提供により、当社若しくは第三者の知的財産権、所有権、その他法令等により保証された権利を害するおそれがあると当社が判断したとき。

(5) 申込者に当社との信頼関係を著しく損なう行為があったとき又は申込者若しくはその役員等が反社会的勢力に該当する等当社が不適当と判断したとき。

(6) 申込者が第18条(利用停止)各号の事由に該当するとき。

(7) 申込者が、申込より以前に、当社が提供するサービスにつき当社と契約を締結したことがあり、かつ、当社から当該契約を解除したことがあるとき。

(8) 申込者が当社に対し虚偽の事実を通知したとき。

(9) 申込に際し、申込者が支払手段として正当に使用することができないクレジットカードを指定したとき。

(10) 申込者がSORACOM Airサービスを適切に利用する意思が無いと当社が判断したとき。

第10条　契約の成立

申込者の申込を当社が第9条(申込の承諾)に基づき承諾した時点で本契約が成立するものとし、以降は申込者を契約者と称するものとします。

第11条　契約者識別番号

1. 契約者識別番号は、当社が定めることとします。

2. その契約者識別番号については、契約者がSORACOM Airサービスを継続的に利用できることを保証するものではありません。

3. 当社は、技術上及び業務の遂行上やむを得ない理由があるときは、SORACOM Airサービスの契約者識別番号を変更することがあります。

第4章　契約者の変更等

第12条　契約者の氏名等の変更の届出

1. 契約者は、氏名、名称、住所若しくは居所又は当社に届け出たクレジットカードその他の当社が指定する事項に変更があったとき又はかかる変更の予定を認識したときは、当社に対し、直ちに当該変更の内容について通知するものとします。 2. 前項の届出があったときは、当社は、その届出のあった事実を証明する書類を提示していただくことがあります。

第13条　名義変更(契約上の地位の移転又は承継)

1. 契約者はSORACOM Airサービスの提供を受ける権利を第三者に譲渡、承継、名義変更、質権その他担保に供する等の行為をすることはできません。ただし、SORACOMシステムの利用状況が「利用開始前」の状態のSORACOM Airサービスの回線に限り、ソラコムコンソールから所定の操作を行うことでSORACOM Airサービスの提供を受ける権利を第三者に譲渡できるものとします。

2. 前項の規定にかかわらず、契約者が死亡した場合、その契約者の法定相続人(相続人が複数あるときは、最初に申し出た相続人)は、当社が定める手続きに従い当社に届け出ることにより、引き続き当該契約に係るSORACOM Airサービス(当社が別途定めるものに限ります。)を受ける権利を承継することができます。この場合、当該相続人は、元契約者の当該契約上の地位(元契約者の当該契約上の債務を含みます。)を引き継ぐものとします。

第5章　利用の制限、中断、中止及び停止等

第14条　利用の制限

1. 当社は、電気通信事業法(昭和59年法律第86号。その後の改正を含みます。)第8条に基づき、天災、事変その他の非常事態が発生し、又は発生するおそれがあるときは、災害の予防若しくは救援、交通、通信若しくは電力の供給の確保、又は秩序の維持に必要な通信その他の公共の利益のために、緊急を要する通信を優先的に取り扱うため、SORACOM Airサービスの利用を制限することができます。

2. 当社は、帯域を継続的かつ大量に占有する通信手順又はアプリケーションを用いて行われる当社所定の電気通信を検知し、その電気通信に割り当てられる帯域を制御すること等により、その電気通信の速度や通信量を制御することができます。

3. 当社は、契約者が当社所定の基準を超過したトラヒック量を継続的に発生させることにより、SORACOM Airサービス用に使用される設備又はシステムに過大な負荷を生じさせる行為、その他その使用若しくは運営に支障をきたす行為、又は契約者若しくは第三者による迷惑メール等送信行為があった場合又はこれらの行為が相当な確度をもってなされる可能性を当社があらかじめ察知した場合には、通信の利用を制限し、SORACOM Airサービスの利用を制限することができます。

4. 当社は、児童買春、児童ポルノに係る行為等の規制及び処罰並びに児童の保護等に関する法律(平成11年法律第52号。その後の改正を含みます。)において定める児童ポルノを閲覧又は取得するための通信を制限することができます。

第15条　通信の切断

当社は、SORACOM Airサービスの通信に関して、次の措置をとることがあります。

(1) セッション(データ通信を行うことができる契約者回線の状態をいいます。以下この条において同じとします。)の設定が長時間継続されたと当社が認める場合において、その通信を切断することがあります。

(2) 同一セッション内に大量の通信があったと当社が認める場合において、その通信を切断することがあります。

第16条　利用の一時中断

当社は、契約者から請求があったときは、SORACOM Airサービスの利用の一時中断(その契約者識別番号を他に転用することなく一時的に利用できないようにすることをいいます。)を行います但し、一時中断の期間は1年を超えることはできず、かかる期間経過後は、当社は契約者のソラコムアカウントその他の契約者情報を保管、維持又は提供する義務を負いません。

第17条　利用中止

1. 当社は、次の場合にはSORACOM Airサービスの提供を中止することができます。

(1) 当社の電気通信設備又はシステムの保守上又は工事上やむを得ないとき。

(2) 当社が契約している電気通信事業者(以下「通信キャリア」といいます。)が当社への携帯電話サービスの提供を停止するとき。

(3) 当社が契約しているクラウド提供業者が当社へのクラウドサービスの提供を停止するとき。

(4) 第11条(契約者識別番号)第3項の規定により、契約者識別番号を変更するとき。

2. 当社は、前項の規定によりSORACOM Airサービスの利用を中止するときは、あらかじめそのことを当社のウェブサイト等において掲示します。ただし、緊急やむを得ない場合は、この限りではありません。

第18条　利用停止

当社は、契約者が次に掲げる事由に該当するときは、当該契約者の利用に係る全てのSORACOM Airサービスについてその全部若しくは一部の提供を停止又は利用を制限することができます。

(1) 料金その他の債務について、支払期日を経過してもなお支払わないとき、又は支払いを怠るおそれがあることが明らかであるとき。

(2) SORACOM Airサービスに係る契約の申込みに当たって、事実に反する申込みを行ったことが判明したとき。

(3) 第37条(禁止行為)の規定に違反したと当社が認めたとき。

(4) 第9条(申込の承諾)に定める申込の拒絶事由に該当するとき。

(5) 契約者が指定したクレジットカードを使用することができなくなったとき。

(6) 前各号に掲げる他、当社が不適切と判断する態様においてSORACOM Airサービスを利用したとき。

第19条　SORACOM Airサービスの廃止

当社は、技術上及び業務の遂行上やむを得ない場合は、SORACOM Airサービスの全部又は一部を廃止することがあります。

第6章　本契約の解除

第20条　契約者が行う契約の解除

1. 契約者は、当社に対し、当社所定の方式により通知をすることにより、本契約を解除することができます。この場合において、当該解除の効力は、当社が当該通知を受領した日からSORACOM Airサービスの種類毎に定める日を経過する日又は契約者が当該通知において解除の効力が生じる日として指定した日のいずれか遅い日に生じるものとします。

2. 前項の規定にかかわらず、第14条(利用の制限)、第15条(通信の切断)、又は第17条(利用中止)第1項の事由が生じたことによりSORACOM Airサービスを利用することができなくなった場合において、本契約の目的を達することができないと認めるときは、契約者は、当社に通知することにより、当社が当該通知を受領した日をもって本契約を解除することができます。

3. 第19条(SORACOM Airサービスの廃止)の規定によりSORACOM Airサービスの全部が廃止されたときは、当該廃止の日に本契約が解除されたものとします。

第21条　当社が行う契約の解除

当社は、契約者が次に掲げる事由に該当するときは、本契約を解除することができます。その場合、当社は、合理的な時期に契約者にその旨を通知します。

(1) 第18条(利用停止)の規定によりSORACOM Airサービスの利用を停止された契約者が、なお当該利用停止の原因事実を解消しないとき。

(2) 第18条(利用停止)各号の規定のいずれかに該当する場合

で、その事実が当社の業務の遂行に特に著しい支障を及ぼすと当社が判断したとき。
(3) 当社と通信キャリアとの契約に基づき、当社への携帯電話サービスの提供に関する契約が通信キャリアによって解除されたとき。
(4) 当社とクラウド提供業者との契約に基づき、当社へのクラウドサービスの提供に関する契約がクラウド提供業者によって解除されたとき。

第7章　責務等

第22条　守秘義務

当社及び契約者は、第8条(申込の方法)に基づく申込以降、相互に知り得た当社又は契約者の技術上、経営上及びその他一般に公表していない一切の事情に関する秘密を厳守し、本約款又は本契約に定める場合を除き、これをSORACOM Airサービスの提供又は使用の目的以外に使用しないこととします。ただし、法令等上必要とされる場合、相手方の書面による同意を得た場合又は主務官庁より報告を要請された場合は、この限りではありません。なお、本条は本契約の締結に至らなかった場合又は本契約が解除された場合若しくは終了した場合であっても有効に存続するものとします。

第23条　信用の維持

契約者は、SORACOM Airサービスの提供又は使用にあたり、当社の信用を損なう行為を行わないように努めるものとします。

第24条　必要事項の通知

1. 契約者は、次の各号に定める事項について、当該事項発生後速やかに当社に対して書面により通知することとします。
(1) 名称、住所若しくは居所、請求書の送付先又は法人の代表者の変更
(2) 第42条(期限の利益喪失)第(2)号乃至第(5)号に定める事由のいずれかが発生した場合においてはその事実
2. 当社は、次の各号に定める事項について、当該事項発生後速やかに契約者に対して通知することとします。
(1) 電気通信事業の休止若しくは廃止又は法人の解散
(2) 電気通信事業の登録、届出又は変更登録の取消し
(3) 電気通信事業法第8条第2項に規定する電気通信業務の一部停止
(4) 提供条件に影響がある電気通信設備の変更、増設又は廃止
3. 第1項第(1)号に規定する変更の通知があったときは、当社は、その届出のあった事実を証明する書類の提示を求めることができます。
4. 契約者において第1項第(1)号に規定する変更があったにもかかわらず、当社に届出が無いときは、第21条(当社が行う契約の解除)に規定する通知については、当社が届出を受けている名称、住所若しくは居所への郵送あるいはソラコムアカウントへの電子メール等の通知をもってその通知を行ったものとみなします。

第25条　情報の提出

当社は、契約者に対して、契約者がSORACOM Airサービスの提供に関し負担すべき金額の支払いを怠るおそれがあるか否かを当社が判断するために必要な情報の提出を求めることがあり、この場合は第9条(申込の承諾)第4項の規定を準用します。

第8章　再提供

第26条　再提供の前提条件

契約者は、SORACOM パートナースペースへの登録その他当社が指定する契約の締結及び手続の履行を行った上でSORACOM Airサービスに基づく電気通信サービスを自己の電気通信サービスとして第三者に提供することができます。ただし、その場合、かかるサービスの提供に関する一切の責任は契約者が負担するものとします。

第27条　利用者数等の報告

契約者は、SORACOM Airサービスに基づく電気通信サービスを自己の電気通信サービスとして提供する場合において、当社が必要とする場合は、かかる電気通信サービスの利用者との間で締結しているSORACOM Airサービスに基づく電気通信サービスに関する契約の数を、当社が定める方法により報告を行うことを要します。

第28条　商標の使用

契約者は、SORACOM Airサービスに基づく電気通信サービスを自己の電気通信サービスとして提供する場合において、当社の登録商標又は商標の使用を希望するときは、当社の承諾を得るものとし、当社が別途定める条件を遵守するものとします。

第29条　本人確認

契約者は、自らの責任により、SORACOM Airサービスに基づく電気通信サービスを自己の電気通信サービスとして提供するときは、その電気通信サービスの申込者に対して、本人確認(携帯音声通信事業者による契約者等の本人確認等及び携帯音声通信役務の不正な利用の防止に関する法律(平成17年法律第31号。その後の改正を含みます。)第3条で定める本人確認をいいます。)及び利用者に係る本人確認(同法第9条で定める契約者確認をいいます。)を行うことを要し、当社はその違反等に基づく一切の責任を負いません。

第30条　提供条件等の説明等

1. 契約者は、自らの責任により、SORACOM Airサービスに基づく電気通信サービスを自己の電気通信サービスとして提供するときは、かかる電気通信サービスの利用者に対して、その電気通信サービスに係る提供条件等の説明を行うことを要し、当社はその不順守等に基づく一切の責任を負いません。
2. 契約者は、前項の規定によるほか、自らの責任により、SORACOM Airサービスに基づく電気通信サービスを自己の電気通信サービスとして提供するときは、かかる電気通信サービスの利用者その他当社又は契約者以外の者からの契約者への通信料金若しくはサービス内容に関する問合せ、SORACOM Airサービスに基づく電気通信サービスに係る故障修理の請求等又はその他の苦情の受付及び対応等を行うことを要します。

第9章　SIMカードの貸与等

第31条　SIMカードの貸与

1. 当社は、契約者が自ら使用するため、又は契約者が第8章(再提供)に従って再提供する電気通信サービス利用者へ転貸与するために、契約者へSIMカードを貸与します。この場合において、貸与するSIMカードの数は、1のSORACOM Airサービス回線につき1とします。
2. 当社は、技術上及び業務の遂行上やむを得ない理由があるときは、当社が貸与するSIMカードを変更することがあります。この場合は、あらかじめそのことを契約者に通知します。この場合において、契約者は、自らの責任により、変更後のSIMカードを契約者が提供する電気通信サービスの利用者へ転貸与するものとします。
3. 契約者は、当社が契約者に対して提供するSIMカードにつき通信キャリアが当社に対して課す管理義務その他の義務を遵守するものとし、また、契約者が提供する電気通信サービスの利用者をしてこれらの義務を遵守させるものとします。また、契約者又は契約者が提供する電気通信サービスの利用者によるSIMカードの管理不十分、使用上の過誤等による損害は契約者が負担するものとし、当社は一切責任を負わないものとします。

第32条　SIMカードの返還

当社からSIMカードの貸与を受けている契約者は、次の場合には、当社が別に定める方法によりそのSIMカードを当社が指定する場所へ速やかに返還していただきます。
(1) 契約者が提供する電気通信サービスの利用者が、当該サービスに係る契約の休止若しくは解除し又は契約を終了したとき。
(2) 第31条(SIMカードの貸与)第2項の規定により、当社がSIMカードを変更するとき。
(3) 当社が契約者識別番号を変更するとき。
(4) 本契約が解除又はその他の理由により終了した場合。
(5) その他SIMカードを利用しなくなったとき。

第10章　通信

第33条　通信時間等の測定

1. SORACOM Airサービスに係る課金対象(契約者回線との間において伝送されるデータをいいます。以下同じとします。)の情報量は、当社の機器により測定します。この場合において、回線の故障等発信者又は着信者の責任によらない理由により、課金対象データ(当社が定めるものを除きます。)が通信の相手先(その通信が相互接続点への通信であるときは、その相互接続点を通信の相手先とします。)に到達しなかった場合には、そのデータについては、情報量の測定から除きます。
2. SORACOM Airサービスに関する課金対象については、前項の規定により測定した情報量を、それぞれの1料金月(各月1日の日本時間午前9時から翌月1日の午前8時59分までの間をいいます。以下同じとします。)における総情報量について、1のSORACOM Airサービス回線契約ごとに、1メガバイトまでごとに1の課金対象として算出します。
3. SMSに係る通信回数は、当社の機器により測定します。
4. SORACOM Beamサービス(当社が第57条(SORACOM Beam)に基づき提供する、契約者からの請求により、当社が設置した電気通信設備において、通信の暗号化や当社宛通信を契約者の指定する送信先に変更して送出するサービスをいいます。以下同じとします。)に係る課金対象(当社と契約者回線との間及び当社と契約者が設定した送信先の間においてそれぞれ要求されるリクエスト数をいいます。以下同じとします。)の情報量は、当社の機器により測定します。この場合において、回線の故障等発信者又は着信者の責任によらない理由により、課金対象データ(当社が定めるものを除きます。)が通信の相手先に到達しなかった場合でも、そのデータは課金対象として算出します。
5. SORACOM Beamサービスに関する課金対象については、前項の規定により測定した情報量を、それぞれの1料金月における総情報量について、1のSORACOM Air サービス回線契約ごとに、1リクエストごとに1の課金対象として算出します。
6. SORACOM Funnelサービス(当社が第62条(SORACOM Funnel)に基づき提供する、契約者からの請求により、当社が設置した電気通信設備において、当社宛通信を契約者の指定する送信先クラウドサービスに変更して送出するサービスをいいます。以下同じとします。)に係る課金対象(当社と契約者が設定した送信先の間において要求されるリクエスト数をいいます。以下同じとします。)の情報量は、当社の機器により測定します。この場合において、回線の故障等発信者又は着信者の責任によらない理由により、課金対象データ(当社が定めるものを除きます。)が通信の相手先に到達しなかった場合でも、そのデータは課金対象として算出します。
7. SORACOM Funnelサービスに関する課金対象については、前項の規定により測定した情報量を、それぞれの1料金月における総情報量について、1のSORACOM Air サービス回線契約ごとに、1リクエストごとに1の課金対象として算出します。

第11章　SORACOMシステムの利用

第34条　ソラコムコンソールの提供

当社は、契約者に対し、SORACOM Airサービスのために、ソラコムアカウントにより使用可能となる同サービスのコンソールシステム(以下、「SORACOMシステム」といいます。)を、SORACOMシステムに係るWEBサイト(以下、「SORACOMサイト」といいます。)を通じて提供します。

第35条　ソラコムコンソールへの接続

SORACOMサイトへの接続は、契約者が自らの費用と責任で行うものとします。SORACOMサイトへの接続中、回線・無線LANの環境等の都合で接続が中断した場合であっても当社は一切の責任を負いません。

第36条　ソラコムコンソールの利用条件

1. 契約者は、法令等を遵守し、善良な管理者の注意をもって通常の用法に従って、SORACOM Airサービス使用の目的の範囲でのみSORACOMシステムを利用するものとします。
2. 当社は、契約者に事前に連絡することなく、SORACOMサイトの内容、SORACOMシステムにより提供する情報(以下、「SORACOM提供情報」といいます。)の内容その他のSORACOMシステムの内容を変更することができます。当該変更が重要なものである場合は、当社は、契約者に対して契約者に事前に通知します。
3. SORACOMシステムの所有権及びSORACOMシステムに関する発明、考案、意匠、商標、著作物等に係る一切の知的財産権(著作権法(昭和45年法律第48号。その後の改正を含みます。)第27条及び第28条の権利を含む。)その他の権利は当社に帰属します。また、SORACOMシステム上のテキスト情報及びデジタル情報はすべて当社の著作物であり、当社は、契約者によるテキスト情報及びデジタル情報の利用行為で当社が不適当と判断する行為を禁止することができます。
4. SORACOM提供情報に係る一切の権利は当社に帰属します。

第12章　禁止行為

第37条　禁止行為

契約者は、次の各号に掲げる行為を行うことはできません。
(1) 当社所定の基準を超過したトラヒック量を継続的に発生させることにより、SORACOM Airサービス用に使用される設備又はシステムに過大な負荷を生じさせること
(2) 迷惑メール又はSMS等の送信
(3) SORACOM Airサービス又はSORACOMシステムの利用者資格を第三者に販売、譲渡し若しくはその再利用権を設定し、又はSORACOM提供情報の全部若しくは一部を第三者に販売、譲渡、転貸すること。
(4) 第三者の使用に供する目的でSORACOM提供情報の全部若しくは一部を複製すること。
(5) 第三者にSORACOM提供情報を取扱わせ、又はその占有を移転すること。
(6) 第三者にSORACOM Airサービス若しくはSORACOMシステムの利用者資格又はSORACOM提供情報を担保として提供すること。
(7) SORACOM提供情報を改変又は改竄すること。
(8) 第三者が販売する商品又はサービスに対してSORACOM提供情報を活用すること。
(9) 当社の知的財産権を侵害する商品又はサービスに対してSORACOM提供情報を活用すること。
(10) SORACOM提供情報を基にして特許その他の知的財産権を取得すること。
(11) 不正なアクセス、コンピューターウィルス等を用いて当社がSORACOM提供情報を格納するサーバーに対して攻撃を行うこと。
(12) SORACOMシステムに対し、リバースエンジニアリング、逆コンパイル、逆アセンブルその他一切の解析を行うこと。
(13) 前各号の行為を第三者に行わせること。

第13章　料金等

第1節　サービス利用料及び支払義務

第38条　サービス利用料

当社がSORACOM Airサービスの料金(以下、「SORACOM Airサービス料金」といいます。)は、基本使用料、通信料、付加機能使用料及びその他の手続に関する料金とし、その額及び計算方法は、料金表第1表(料金)(以下、「本料金表」といいます。)に定めるところによります。

第2節　サービス利用料の支払義務

第39条　基本使用料等の支払義務

1. 契約者は、本契約に基づいて当社が契約者回線の提供を開始した日から、本料金表に規定するSORACOM Airサービス料金を支払う義務を負います。
2. 契約者が、付加機能の提供を受ける場合、かかる付加機能の提供開始日から、本料金表に規定する料金を支払う義務を負います。
3. 契約者は、本契約に基づいて当社が契約者回線の提供を開始して以降は、第16条(利用の一時中断)、第17条(利用中止)又は第18条(利用停止)によりSORACOM Airサービスを利用することができない又は利用しない状態が生じたときであっ

ても、基本使用料（ユニバーサルサービス料を含みます。）及び付加機能使用料を支払う義務を負います。

第40条　SORACOM Airサービス料金の支払方法

契約者は、SORACOM Airサービス料金を、当社が指定する日までに、当社が指定する方法により支払うものとします。

第41条　延滞利息

契約者は、SORACOM Airサービス料金その他の債務（延滞利息を除きます。）について支払期日を経過してもなお支払いがない場合には、支払期日の翌日から支払いの日の前日までの日数について、年14.5%の割合で計算して得た額を延滞利息として支払っていただきます。

第42条　期限の利益喪失

契約者は、次の各号に定める事由のいずれかが発生したとき（第(4)号、第(5)号又は第(6)号に該当する場合においては、契約者が負担すべきSORACOM Airサービス料金その他の債務の支払を怠るおそれがないことを契約者が明らかにしたときを除きます。）は、当社に対して負担するSORACOM Airサービス料金その他の債務の全てについて、当然に期限の利益を失い、当社に対して直ちにそのSORACOM Airサービス料金その他の債務を弁済しなければならないものとし、以後発生する債務については、その事由が解消されない限り、期限の定めのないものとします。

(1) 契約者が負担する債務の全部又は一部について履行不能状態に陥ったと当社が認めたとき。

(2) 契約者について、破産手続開始、会社更生手続開始又は民事再生手続開始その他法令に基づく倒産処理手続の申立てがあったとき。

(3) 契約者に係る手形又は小切手が不渡りとなったとき。

(4) 契約者の資産について、法令に基づく強制換価手続の申立てがあったとき、契約者を債務者とする差押え若しくは仮差押え、金銭債権保全のための仮処分又は税等の滞納処分があったとき。

(5) 契約者について電気通信事業の登録又は届出が取り消されたとき。

(6) 契約者が電気通信事業の全部を廃止したとき。

(7) 契約者の所在が不明なとき。

(8) その他契約者の業務継続に重大な支障を及ぼすと認められる状態が発生した場合であって、契約者がその負担すべき債務を履行すると認められないとき。

第14章　保守

第43条　当社の維持責任

当社は、当社の設置した電気通信設備を事業用電気通信設備規則（昭和 60年郵政省令第30号）に適合するよう維持します。

第44条　修理又は復旧

1. 当社は、当社の設置した電気通信設備又はシステムが故障し又は滅失した場合は、速やかに修理し又は復旧するものとします。ただし、24時間未満の修理又は復旧を保証するものではありません。

2. 当社は、当社の電気通信設備又はシステムを修理又は復旧するときは、契約者識別番号を変更することがあります。

第15章　知的財産権

第45条　権利等の非許諾

SORACOM Airサービス、SORACOMシステム及びこれらに付帯するサービスに関する特許権、実用新案権、意匠権、著作権等の知的財産権及びノウハウ等の一切の権利並びに実証実験のデータその他の記録は当社に帰属するものであり、本約款、SORACOM Airサービス、SORACOMシステム又はこれらに付帯するサービス提供の過程での当社による契約者に対する情報の開示は、明示、黙示を問わず、いかなる意味においても、当社による契約者に対する、特許権、実用新案権、意匠権、著作権、ノウハウ等に基づく実施権その他のいかなる権利の許諾、付与、又は譲渡を構成するものではありません。

第16章　保証の否認

第46条　保証の否認

契約者は、SORACOM Airサービス、SORACOMシステム及びこれらに付帯するサービスは現状のままで提供されることに合意するものとします。当社は、提供されるSORACOM Airサービス、SORACOMシステム及びこれらに付帯するサービスが中断されないこと、誤りがないことの保証を含め、明示であると黙示であるとを問わず、いかなる種類の表明も保証も行いません。また、法令等により禁止される場合を除き、当社は、SORACOM Airサービス、SORACOMシステム及びこれらに付帯するサービスに関し、品質、特定目的への適合性に関する黙示の保証ならびに取引過程又は取引慣行により生じる保証を含め、一切の保証を行いません。

第17章　損害賠償

第47条　損害賠償

本約款に別段の定めがある場合を除き、当事者は、本約款に定める義務に違反したことにより相手方に損害を与えた場合には、本約款に別途定める場合を除き、当該義務違反により相手方が被った損害を賠償する責任を負うものとします。

第48条　責任の制限

1. 当社は、法令等で求められる場合を除き、第三者の帰責事由による本サービス利用不能の場合、責任を負わないものとします。

2. 当社は、SORACOM Airサービスを提供すべき場合において、当社の責に帰すべき事由によりSORACOM Airサービスが全く利用できない状態（全く利用できない状態と同程度の状態となる場合を含みます。以下同じとします。）にあることを、当社が認知した時刻から連続して24時間以上の時間（以下「利用不能時間」といいます。）当該状態が継続したときで契約者から請求があった場合、当社は、契約者に対し、その請求に基づき、利用不能時間を24で除した数（小数点以下の端数は、切り捨てます。）に応じた日額のSORACOM Airサービス料金額を減額しますが、当社はそれを超えては責任を負いません。ただし、契約者が当該請求をし得ることとなった日から3ヶ月を経過する日までに当該請求をしなかったときは、契約者は、その権利を失うものとします。ただし、当社の故意又は重大な過失による場合はこの限りではありません。

3. 事由の如何を問わず、当社が契約者に対して損害賠償責任を負う場合、当該損害が発生した日に属する月の月額のSORACOM Airサービス料金を上限とします。ただし、当社の故意又は重大な過失による場合はこの限りではありません。

4. 前各項の規定にかかわらず、通信キャリア・クラウド提供業者の帰責事由によるSORACOM Airサービスの利用不能の場合には、当社は、通信キャリア・クラウド提供業者から受領した損害賠償額を限度として契約者に生じた損害（但し、現実に発生した通常損害に限られ、逸失利益、間接損害は含みません。）につき責任を負います。

5. 当社は、SORACOM Airサービスの提供が行われなかったことによる逸失利益及び契約者の顧客、契約者が提供する電気通信サービスの利用者その他の当社又は契約者以外の者から契約者への問合せ対応、故障修理の請求等その他の苦情の受付又は対応等に要した費用等について一切責任を負わないものとし、契約者はかかる逸失利益又は費用等を当社へ請求しないものとします。

第49条　免責

電気通信設備又はシステムの修理、復旧等に当たって、その電気通信設備又はシステムに記憶されている内容等が変化又は消失することがあります。当社はこれにより損害を与えた場合に、それが当社の故意又は重大な過失により生じたものであるときを除き、その損害を賠償しません。

第18章　雑則

第50条　約款の掲示

当社は、本約款（変更があった場合は変更後の約款）を当社のウェブサイトにおいて掲示することとします。

第51条　プライバシーポリシー

当社は、契約者に関する個人情報の取扱いに関する方針（以下「プライバシーポリシー」といいます。）を定め、これを当社のウェブサイト等において公表します。

第52条　通信キャリアへの情報の通知

契約者は、SMSの送信を行った場合であって、そのSMSの接続先の電気通信回線を設定した通信キャリアが、その電気通信回線に係る利用者からの申出に基づき、そのSMSの送信をその通信キャリアが規定する禁止行為に該当すると判断したときは、その通信キャリアが当社及び当社以外の通信キャリアへ、SMSの送信を行った契約者回線に係る契約者識別番号、SMSの受信時刻及びSMSの内容等の情報を通知することに予め同意するものとします。

第53条　反社会的勢力の排除

1. 当社及び契約者は、自己が反社会的勢力（「企業が反社会的勢力による被害を防止するための指針（平成19年6月19日犯罪対策閣僚会議幹事会申合せ）」において、暴力、威力又は詐欺的手法を駆使して経済的利益を追求する集団又は個人である旨定められている「反社会的勢力」、以下同じとします。）又は次のいずれかに該当する者（以下併せて「反社会的勢力等」といいます。）に該当しないことを表明及び保証し、現在及び将来において反社会的勢力又は次の事項に該当しないことを確約するものとします。

(1) 役員等（役員のほか、支配人、営業所の代表者その他いかなる名称によるかを問わず役員と同等以上の職権又は支配力を有するものをいい、非常勤の者を含みます。）に、暴力団員による不当な行為の防止等に関する法律（平成3年法律第77号。その後の改正を含みます。）第2条第6号に規定する暴力団員（以下「暴力団員」といいます。）又は同条第2号に規定する暴力団（以下「暴力団」といいます。）と関係を持ちながら、その組織の威力を背景として同条第1号に規定する暴力的不法行為等を行なうおそれがある者（以下「暴力団関係者」といいます。）がいること。

(2) 暴力団、暴力団員又は暴力団関係者（以下これら三者を「暴力団等」と総称します。）が経営に関与していること。

(3) 暴力団等から名目を問わず資金提供、出資などの便益を受けていること。

(4) 暴力団等に対し名目を問わず資金の供給などの便益を供与していること。

(5) 反社会的勢力との間に、利用、協力、交際など社会的に非難されるべき関係を有していること。

2. 当社又は契約者が、相手方が第1項の規定に反すると疑う事実のあるときは、相手方に対し当該事項に関する報告を求めることができ、報告を求められた相手方は指定された期日までに報告書を提出するものとします。

3. 当社又は契約者は、相手方が次の各号のいずれかに該当した場合は、即時本契約を解除し、解除によって生じた損害を相手方に請求することができるものとします。

(1) 第1項の表明、保証又は確約に反し、又は反すると疑うに足る相当の理由があるとき。

(2) 第2項の規定に違反して報告書を提出せず、又は虚偽の記載をした報告書を提出したとき。

第54条　分離可能性

本約款のいずれかの条項が何らかの理由により無効又は執行不能とされた場合であっても、本約款の他の条項が無効又は執行不能となるものではなく、また、かかる場合には、当該規定は、有効かつ執行可能となるために必要な限度において限定的に解釈されるものとします。

第55条　合意管轄

本契約に起因し又は関連する一切の紛争については、東京地方裁判所を第一審の専属的合意管轄裁判所とします。

第56条　準拠法

本約款の成立、効力、解釈及び履行については、日本国法に準拠するものとします。

第19章　その他のサービス

第57条　SORACOM Beamサービス

1. 契約者は、SORACOM Airサービスにおいて、SORACOM Beamサービスを利用することができます。

2. 当社は、SORACOM Beamサービスに関する契約者の損害については第48条（責任の制限）の規定に該当する場合に限り、その規定により責任を負うものとし、端末設備又は通信内容に係る情報の変化若しくは消失、動作不良又は第三者との紛議により生じた損害その他の損害については、一切の責任を負いません。

3. SORACOM Beamサービスの利用方法その他の提供条件については、当社がウェブサイトに掲示するところによります。

第58条　カスタムDNSサービス

1. 契約者は、SORACOM Airサービスにおいて、独自にDNSサーバを設定することができるカスタムDNSサービスを利用することができます。

2. 当社は、カスタムDNSサービスに関する契約者の損害については第48条（責任の制限）の規定に該当する場合に限り、その規定により責任を負うものとし、端末設備又は通信内容に係る情報の変化若しくは消失、動作不良又は第三者との紛議により生じた損害その他の損害については、一切の責任を負いません。

3. カスタムDNSサービスの利用方法その他の提供条件については、当社がウェブサイトに掲示するところによります。

第59条　メタデータサービス

1. 契約者は、SORACOM Airサービスにおいて、SORACOM Airサービスを利用しているデバイスの情報（IMEI）の取得及びデバイス自身が使用しているSORACOM Airの情報を取得、更新することができるメタデータサービスを利用することができます。

2. 契約者は本サービスの利用にあたり、利用者から適切な許諾を得るものとします。

3. 当社は、メタデータサービスに関する契約者の損害については第48条（責任の制限）の規定に該当する場合に限り、その規定により責任を負うものとし、端末設備又は通信内容に係る情報の変化若しくは消失、動作不良又は第三者との紛議により生じた損害その他の損害については、一切の責任を負いません。

4. メタデータサービスの利用方法その他の提供条件については、当社がウェブサイトに掲示するところによります。

第60条　端末情報取得サービス

1. 契約者は、SORACOM Airサービスにおいて、SORACOM Airサービスを利用しているデバイスの情報（IMEI）を取得することができる端末情報取得サービスを利用することができます。

2. 契約者は本サービスの利用にあたり、利用者から適切な許諾を得るものとします。

3. 当社は、端末情報取得サービスに関する契約者の損害については第48条（責任の制限）の規定に該当する場合に限り、その規定により責任を負うものとし、端末設備又は通信内容に係る情報の変化若しくは消失、動作不良又は第三者との紛議により生じた損害その他の損害については、一切の責任を負いません。

4. 端末情報取得サービスの利用方法その他の提供条件については、当社がウェブサイトに掲示するところによります。

第61条　SORACOM Canalサービス

1. 契約者は、SORACOM Airサービスにおいて、SORACOM Canalサービスを組み合わせて利用することができます。その場合、SORACOM Air VPG利用オプションの契約が必要となります。

2. SORACOM Canalサービスの利用方法その他の提供条件については、閉域網等接続サービス契約約款および当社がウェブサイトに掲示するところによります。

第62条　SORACOM Directサービス

1. 契約者は、SORACOM Airサービスにおいて、SORACOM

Directサービスを組み合わせて利用することができます。その場合、SORACOM Air VPG利用オプションの契約が必要となります。
2. SORACOM Directサービスの利用方法その他の提供条件については、閉域網等接続サービス契約約款および当社がウェブサイトに掲示するところによります。

第63条　SORACOM Endorseサービス
1. 契約者は、SORACOM Airサービスにおいて、SORACOM Endorseサービスを利用することができます。
2. 契約者は本サービスの利用にあたり、利用者から適切な許諾を得るものとします。
3. 当社は、SORACOM Endorseサービスに関する契約者の損害については第48条(責任の制限)の規定に該当する場合に限り、その規定により責任を負うものとし、端末設備又は通信内容に係る情報の変化若しくは消失、動作不良又は第三者との紛議により生じた損害その他の損害については、一切の責任を負いません。
4. SORACOM Endorseサービスの利用方法その他の提供条件については、当社がウェブサイトに掲示するところによります。

第64条　SORACOM Funnelサービス
1. 契約者は、SORACOM Airサービスにおいて、SORACOM Funnelサービスを利用することができます。
2. 当社は、SORACOM Funnelサービスに関する契約者の損害については第48条(責任の制限)の規定に該当する場合に限り、その規定により責任を負うものとし、端末設備又は通信内容に係る情報の変化若しくは消失、動作不良又は第三者との紛議により生じた損害その他の損害については、一切の責任を負いません。
3. SORACOM Funnelサービスの利用方法その他の提供条件については、当社がウェブサイトに掲示するところによります。

第65条　クーポン
1. 契約者は、次の各号に定める内容を当社所定の条件、方法により利用することができます。なお、ご利用にはSORACOM Airサービスの契約が必要です。
(1) クーポンの設定金額にて、当社の基本使用料、通信料、各サービス料などの月々の料金を支払うための電子データを登録すること
2. 契約者は1料金月あたり2つまで、有効なクーポンを同時に登録しておくことが可能です。
3. クーポンの適用は1料金月単位に実施します。なお、無料利用枠がクーポンに優先して適用されます。
4. クーポンの適用状況等は、当社指定のサイト上でご確認いただくことができます。
5. クーポンはコンソールからの登録後、譲渡することができません。
6. クーポンの有効期限は発行日から180日を超えることは無く、資金決済法の対象外であることを確認します。
7. 契約者が、クーポンを盗難、滅失、毀損、紛失した場合や、その他いかなる理由であっても、当社はクーポンを再発行できず、また、その義務を負わないものとします。
8. 当社は、クーポンの残高の払い戻し、換金を行わないものとします。

料金表
通則
(料金の計算方法等)
1. 当社は、この料金表において、消費税相当額を含まない額(以下「税抜額」といいます。)で料金を定めるときは、その額に消費税相当額を加算した額(以下「税込額」といいます。)を併記します。この場合において、当社は税抜額により料金を計算することとします。
(注)この料金表に規定する税込額は消費税法(昭和63年法律第108号。その後の改正を含みます。)第63条基づき表示するものであり、税込額で計算した額は実際に支払いを要する額と異なる場合があります。
2. 当社は、契約者がその契約に基づき支払う料金について、1料金月単位で計算します。なお、日額で課金される料金については、当日の日本時間午前9時を起算時とする24時間以内に1回の通信が生じた場合は当日の利用があったものとみなします。
ただし、当社が必要と認めるときは、料金月によらず随時に計算します。
(注)料金月に従って通信量を計算する場合において、通信又はセッションを開始した料金月と終了した料金月が異なるときは、当社が定める方法により計算するものとします。
3. 当社は、当社の業務の遂行上やむを得ない場合は、料金月に係る起算日を変更することがあります。
(端数処理)
4. 当社は、料金その他の計算において、その計算結果に1円未満の端数が生じた場合は、その端数を切り上げます。
(料金等の支払い)
5. 契約者は、料金について、第7項に規定する場合を除き、所定の支払期日までに支払っていただきます。
6. 料金は支払期日の到来する順序に従って支払っていただきます。
(料金の一括後払い)
7. 当社は、1料金月の料金が50円に満たない場合及び当社に特別の事情がある場合は、2月以上の料金を当社が指定する期日までにまとめて支払っていただくことがあります。また、全回線の解約を行った場合等で1料金月の料金が50円に満たない場合、料金を50円に切り上げて支払っていただくことがあります。

第1表　SORACOM Airサービス料金
第1　基本使用料
1. 料金プラン

1契約ごとに

料金プラン	日額料金の額　次の税抜額(カッコ内は税込額)
s1プラン	10円　(10.8円)

基本使用料にはユニバーサルサービス料を含みます。
ユニバーサルサービス料が値上がりする場合、又はキャリアが約款にて規定するMVNO向け料金における基本料等を値上げする場合には、当社はかかる値上げに対応して、基本料を値上げ(又は新たな利用料金を設定)することができるものとします。

2. SORACOMシステムの利用状況に応じた割引
SORACOM Airサービスの基本使用料については、別途当社が定めるSORACOMシステムの利用状況に応じて、次表の日額料金を適用します。

1契約ごとに

利用状況	日額料金の額　次の税抜額(カッコ内は税込額)
利用開始前	5円　(5.4円)

第2　通信料
1. 料金プラン
本約款第33条(通信時間等の測定)に基づき測定された課金対象の情報量に応じて以下の通信料を適用します。

1契約・1MBごとに

料金クラス	通信速度	料金額(上り・下り) 次の税抜額(カッコ内は税込額)
minimum	32kbps	0.2円・0.6円　(0.216円・0.648円)
slow	128kbps	0.22円・0.7円　(0.2376円・0.756円)
standard	512kbps	0.24円・0.8円　(0.2592円・0.864円)
fast	2Mbps	0.3円・1円　(0.324円・1.08円)

なお、上りは契約者の端末から当社、下りは当社から契約者の端末への通信を意味します。
ただし、午前2時から午前6時までに行われた通信は以下のとおりとします。

1契約・1MBごとに

料金クラス	通信速度	料金額(上り・下り) 次の税抜額(カッコ内は税込額)
minimum	32kbps	0.2円・0.2円　(0.216円・0.216円)
slow	128kbps	0.2円・0.2円　(0.216円・0.216円)
standard	512kbps	0.2円・0.2円　(0.216円・0.216円)
fast	2Mbps	0.2円・0.2円　(0.216円・0.216円)

なお、本項における通信速度の数値は実際の伝送速度の上限を示すものではありません。通信の伝送速度は通信の状況等により変動します。

2. 無料利用枠
ソラコムアカウントを新規に作成した月を含む作成後12ヶ月間は、通信料に対し、1ソラコムアカウントあたり各月30円(税込み32.4円)分を無料通信分として減算するものとします。なお、基本使用料の減算は行わず、また、月末に残った未使用分の無料利用枠が翌月に持ち越されることはありません。
本無料利用枠の有効期間が終了した後のSORACOM Airサービスのご利用に対しては、1.「料金プラン」に定める通信料が課金されます。また、本無料利用枠を超えた通信に対しても、1.「料金プラン」に定める通信料が課金されます。
本無料利用枠の項目が新たに追加された場合は、有効期間が終了するまでの間ご利用いただけます。ただし、新しい無料提供項目が追加されても、既に無料利用枠をご利用いただいている、又はソラコムアカウントを新規に作成した月を含む作成後12ヶ月間の有効期間を経過したSORACOM Airサービスの既存のサービス利用者の有効期間が延長されることはありません。

第3　付加機能使用料
1. SMS機能

項目	料金　次の税抜額(カッコ内は税込額)
SMS機能利用料	1契約(SIMカード)あたり日額5円(税込み5.4円)
SMS通信料	本約款第33条(通信時間等の測定)に基づき測定された通信回数及び通信文字数に対し、ドコモが定めるFOMAサービス契約約款及びXiサービス契約約款においてショートメッセージ通信モードに係る料金として定められた額と同額(国外への送信においては、消費税は課税されません)の通信料を適用します。

2. SORACOM Beamサービス

項目	料金　次の税抜額(カッコ内は税込額)
Beam料金	1リクエストあたり0.0009円(税込み0.000972円)

3. カスタム DNS サービス

項目	料金　次の税抜額(カッコ内は税込額)
カスタム DNS 料金	カスタムDNS機能を有効にしたグループに所属する1契約(SIMカード)あたり日額3円(税込み3.24円)

4. 端末情報取得サービス

項目	料金次の税抜額(カッコ内は税込額)
端末情報取得サービス料金	無料

5. VPG利用オプションサービス

項目	料金　次の税抜額(カッコ内は税込額)
VPG利用オプション料金	SORACOM Canal / SORACOM Directを利用するにあたり、SORACOM Airにおいて VPG利用オプション機能を有効にしたグループに所属する1契約(SIMカード)あたり日額5円(税込み5.4円)

6. SORACOM Endorseサービス

項目	料金　次の税抜額(カッコ内は税込額)
Endorse料金	Endorse機能を有効にしたグループに所属する1契約(SIMカード)あたり日額5円(税込み5.4円)

7. SORACOM Funnelサービス

項目	料金　次の税抜額(カッコ内は税込額)
Funnel料金	1リクエストあたり0.0018円(税込み0.001944円)

8. 無料利用枠
ソラコムアカウントを新規に作成した月を含む作成後12ヶ月間は、Beam料金に対し1ソラコムアカウントあたり各月50,000リクエスト分を、Endorse料金に対し1ソラコムアカウントあたり各月155円(税込み167.4円)分を、Funnel料金に対し1ソラコムアカウントあたり各月50,000リクエスト分をそれぞれ無料通信分として減算するものとします。なお、基本使用料の減算は行わず、また、月末に残った未使用分の無料利用枠が翌月に持ち越されることはありません。
本無料利用枠の有効期間が終了した後のSORACOM Beamサービス、SORACOM Endorseサービス及びSORACOM Funnelサービスのご利用に対しては、2.「SORACOM Beamサービス」、5.「SORACOM Endorseサービス」及び6.「SORACOM Funnelサービス」に定めるBeam料金、Endorse料金、Funnel料金がそれぞれ課金されます。また、本無料利用枠を超えた通信に対しても、2.「SORACOM Beamサービス」、5.「SORACOM Endorseサービス」及び6.「SORACOM Funnelサービス」に定めるBeam料金、Endorse料金、Funnel料金がそれぞれ課金されます。
本無料利用枠の項目が新たに追加された場合は、有効期間が終了するまでの間ご利用いただけます。ただし、新しい無料提供項目が追加されても、既に無料利用枠をご利用いただいている、又はソラコムアカウントを新規に作成した月を含む作成後12ヶ月間の有効期間を経過したSORACOM Airサービスの既存のサービス利用者の有効期間が延長されることはありません。

第4　手続きに関する料金
1. 手続きに関する料金の種別

種別	内容
契約事務手数料	SORACOM Airサービスの申込みをし、その承諾を受けた時に支払いを要する料金
SIM再発行手数料	(自然故障であるか否かを問わず)故障・紛失等の場合、SIMカードのサイズ変更を行う場合、SORACOM Airサービスの種別変更に伴うSIM再発行を行う場合にあって、1SIMカードあたりSIM再発行手数料として支払いを要する料金

2. 料金額

料金種別	単位	次の税抜額 (カッコ内は税込額)
契約事務手数料	1契約ごとに	560円　(604.8円)
SIM再発行手数料	1枚ごとに	2,000円　(2,160円)

次号予告

Software Design
June 2016

2016年6月号
定価（本体1,220円+税）
192ページ

5月18日発売

［第1特集］新人エンジニア養成特別企画
使いこなしていますか?

Bash入門+再入門
エンジニアの道具を磨こう

エンジニアにとってUnixシェルは道具の中の道具です。しかし、皆さん使いこなしていますか? シェルのなかで使用頻度の高いbashを取り上げ、多面的にエンジニアの道具を見直します。もちろんシェルスクリプトの書き方から、シェル芸の極め方までみっちりサポート。さらに発表されたばかりのbash on Windowsについても速報解説!

［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RDBであるMySQLをベースに、SQLの基礎の基礎をしっかりおさえ、一生使える技術を習得しましょう!

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「Vimの細道」（第8回）は都合によりお休みさせていただきます。

SD Staff Room

●今月は入稿中にインフルエンザA型に罹患。地獄を味わう。人生も半分を過ぎようというのに初めてインフルエンザにかかった。風邪とは違うヤバイ「雰囲気」のする病気で、これでは老人はひとたまりもないと思った。しかしながらイナビルという薬であっというまに治療は終わった。これはスゴイことだ。（本）

●自宅玄関に新しいネットカメラ（2台目）を設置したら1週間もしないうちに警察官が来た。「何か困ったことがありましたか?」と聞かれ「庭に入るネコの動向を調べるため」と回答し、笑われた。1台目の時は、登録して事件が起きた時に協力して欲しいと言われたっけ。リアル「ねこあつめ」。お巡りさんまで集める。（幕）

●マンションが大規模修繕中。14階建てでも結構短期間に骨組み作っちゃうんですね。こんな高所でお仕事をする職人の皆さんには恐れ入ります。しかし平日8:30から17:00まではベランダに洗濯物が干せず、工事の人がいつ来るかわからないからカーテンは閉めっぱなし。こりゃストレスたまるわ。（キ）

●今どきは、新たな知り合いができたときに「携帯のメールアドレスを教えてください」と言うと驚かれるそうです（みんなLINEだから）。そして、アドレスを教えてもらう際には、赤外線通信がないスマホも多いので、アドレスを口頭で読み上げ、手打ち入力することになるそう。ひと昔前に戻ったの?（よし）

●最近ハンバーグにハマってるんだ。会社の近くに『らいむらいと』とか『ABO』みたいなおいしい店が結構あって、食べ比べを楽しんでいるよ。食事は1人よりも2人のほうが楽しいんだが、ハンバーグは1人で食べるに限るね。なぜかってそう、逢引（合い挽き）は好きじゃないから。ハンバーーー（ry（な）

●編集部のあるビルと本社との間にある小路の脇には樹木がたくさん植わっていて、この時期、春の陽気で一気に芽吹いた花々が美しくていつも見惚れてしまいます。先日、植込みの陰に小さなタンポポの花を見つけました。それも珍しいニホンタンポポ。こんな都会の片隅で出逢えるなんて! ちょっと嬉しい出来事なのでした。（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］
03-3513-6173

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2016年5月号

発行日
2016年5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。