special feature 1 Vimに強くなる!

special feature 2 受託開発の最適解

Software Design

2015年1月18日発行
毎月1回18日発行
通巻357号
（発刊291号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,220円 +税

[ソフトウェア デザイン]
OSとネットワーク、
IT環境を支える
エンジニアの総合誌

2015 1

プログラマ・インフラエンジニア・文書書きの必携

special feature 1

使うほどなじむ

「Vim使い」事始め

年末年始企画

くつなりょうすけ
「ひみつのLinux通信スペシャル」

エンジニア出世双六

special feature 2

SI崩壊を乗りきる3つの方法

ソフトウェア開発の未来

請負・受託開発は変わるべきか？

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

第73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。

Amazon Aurora

AWSの新サービス「Amazon Aurora」

パブリッククラウドサービスの先駆者としてクラウド業界をリードし続けているAmazon Web Services（AWS）ですが、そのAWSに新たに加わったサービス「Amazon Aurora」がクラウドデータベースの新しい形を示すものとして話題になっています。Amazon AuroraはMySQLと互換性があるデータベースサービスで、Amazon Relational Database Service（RDS）で選択できるデータベースエンジンの1つとして公開されました。

Amazon RDSはクラウド環境におけるリレーショナルデータベースのセットアップや運用、スケーリングを容易に行えるようにするWebサービスで、MySQLやOracle Database、SQL Server、PostgreSQLといった主要なデータベースエンジンがサポートされています。Auroraもそれらに並ぶ選択肢の1つになるわけですが、商用データベースを代替することを目指した野心的なサービスでもあります。

Auroraの強みは、最初からクラウドサービスを前提とした設計になっているため、クラウドの能力を最大限に活用できるという点です。具体的な特徴としては、次のような項目が挙げられます。

◉パフォーマンス

既存のAmazon RDS for MySQLに比べて5倍のスループットを実現

◉高可用性

データは6重化され、ストレージ側で分散処理による同時書き込みを行うことで99.99%の可用性を実現している。自動バックアップ機能や自動復旧機能を標準で備えており、障害発生時にも速やかなリカバリが可能

◉安価

データベースそのもののライセンス料は不要で、利用したストレージ容量分だけの料金を支払う従量課金制であるため、コストの最適化が可能。同等のパフォーマンスを実現するハイエンドの商用データベースに比べて10分の1程度の価格で利用できる

◉互換性

MySQL 5.6と互換性を持つように設計されており、一般的なRDBを使用している既存サービスを最小限の修正で移行することができる。とくにAmazon RDS for MySQLからであれば、数クリックの操作でマイグレーションが完了する

◉拡張性

ストレージは最小10GBから最大64TBまで、サービスを停止することなく自動的に最適なサイズにスケールさせることが可能

堅牢性を実現するアーキテクチャ

Auroraの信頼性と可用性を担保しているのは、AWSアベイラビリティゾーン（AZ）を利用した冗長化機能です。Auroraでは、SQLのリクエストを受け付けてトランザクション処理を実行するデータベースインスタンスと、データを保持するストレージエンジンが分離された設計になっています。ストレージは自動的に3つのAZに複製され、さらにそれぞれのAZ内で2つのコピーを作成します。書き込み完了の判断は6本のストレージすべての処理の終了を待つのではなく、少なくとも4本の書き込みが完了したら次の処理に進むという方式が採用されています。これによって信頼性を保ったまま高い並列度での書き込みを実現できるとのことです。

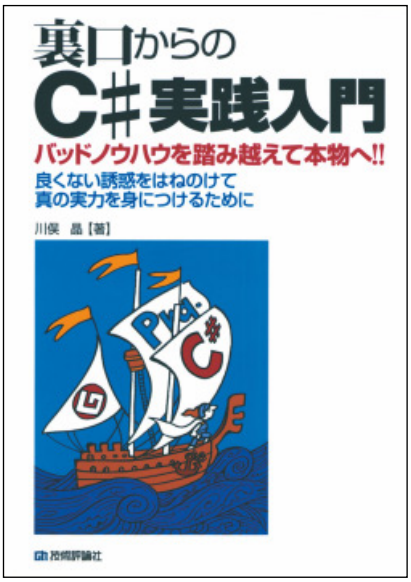
万が一障害が発生した場合でも、失われたコピーが2つまでであれば継続して書き込みが可能で、3つまで失われたとしても読み込みは継続できる設計になっているとのことです。さらに、データの状態を常に監視し、インスタンスやディスクの障害を検知した場合には自動的に復旧処理を行う機能が備わっています。バックアップはAmazon S3に対して自動的かつ継続的に実行されます。前述の分散書き込みモデルの効果で、バックアップの作成中でもデータベースインスタンスに余分な負荷を与えることはないそうです。

Auroraはエンタープライズレベルの要求に耐えられるパフォーマンスと堅牢性を備えながら、クラウドの能力を最大限に活かす柔軟性や拡張性、高いコストパフォーマンスを実現したデータベースです。テーブルの容量の増加に応じて自動でスケールするため、ストレージ増設などによるダウンタイムをなくした運用が可能で、これは従来のRDBでは実現が難しいものでした。クラウド時代のRDBは、このAuroraの登場によって新しい局面を迎えるかもしれません。SD

裏口からのC#実践入門

バッドノウハウを踏み越えて本物へ!!

●川俣晶　著



ISBN978-4-7741-6816-6
A5判／320ページ
定価（本体2680円+税）

比較的使いやすいために、現在、開発現場においては主流のひとつと言って過言ではないC#でも、入門を果たした後でコーディングの力を磨くためには、多くの場合、自ら失敗を繰り返す必要があります。しかし、そういう無駄は誰しも省きたいものです。失敗にはパターンがあり、分類可能な、類型化できる原因があります。そういった実際によく発生する事例をタイプ別・原因別に提示し、何がいけなかったのか、本当はどうすべきか、それをどう改良すればいいかなど、現役のプログラマーに必要な知恵を授けることを本書は目的としています。

15時間でわかるJava集中講座

宮下明弘
工藤雅人
原田僚　著
井上誠一郎　監修

ISBN978-4-7741-6798-5
B5変形判／416ページ
定価（本体2680円+税）

学習環境があらかじめ準備されており、すぐに学習をはじめられるJavaの教科書です。ベースとなる文法について扱う基礎編と、テスト、デバッグ、リファクタリングといった実際の業務に直結する技術や考え方を扱う実践編の2部構成で、最短で業務レベルの入口まで到達できる構成になっています。
あらかじめ用意されている仮想環境を付属のDVD-ROMからコピーすることで、細かい設定に時間をとられず、すぐに学習を開始することができます。

15時間でわかるGit集中講座

岡本隆史
織田翔
大山智之　著

ISBN978-4-7741-6575-2
B5変形判／256ページ
定価（本体2580円+税）

15時間で理解できる、Gitの教科書です。本書では、2部構成でユーザ編と管理者編に分けて、それぞれの役割の人がスムーズにGitが導入できるような構成になっています。ユーザ編では、SourceTreeを中心とした直感的でわかりやすくGitを使える方法を、管理者編では、Gitリポジトリの環境構築はもちろんのこと、Subversionなどの既存のバージョン管理ツールからの移行や大規模なプロジェクトで運用するノウハウなどを紹介しつつより進んだ運用として、チケットシステムやCIツールとの運用取り上げます。

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

Contents

Software Design Jan. 2015

1

［第1特集］

使うほどなじむ

プログラマ・インフラエンジニア・文章書きの心得

「Vim使い」事始め

技術評論社の 確定申告本

平成27年3月締切分

初めてでも大丈夫！ マネして書くだけ確定申告 平成 27 年 3 月締切分

●山本宏　監修

A4 判／ 176 ページ　定価（本体 1380 円＋税）
ISBN 978-4-7741-6789-3

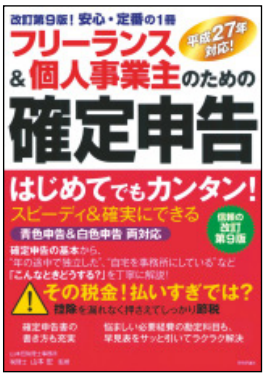
家族が働いていたり副業があるために確定申告が必要なサラリーマンのために、確定申告に必要な書類の作成方法を、簡単にわかりやすくまとめました。ほかにもパートやアルバイト、フリーランス、不動産オーナー、年金で生活している方などが確定申告を行う場合についても、個別のケースごとに詳しく解説しています。本書を参考にすれば、該当するケースを探して、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などに、おすすめしたい1冊です！

フリーランス＆個人事業主 確定申告でお金を残す！元国税調査官のウラ技

●大村大次郎　著

A5 判／ 208 ページ
定価（本体 1580 円＋税）
ISBN 978-4-7741-6771-8

税務署は申告のやり方は教えてくれても、手元により多くのお金を残す方法を指南してくれることは決してありません。経費、控除、税制、申告の「裏道」を知っている人たちだけがトクをしています。本書の著者は、元国税調査官として税金や経費のグレーゾーンを知り尽くした大村大次郎さん。「フリーランス・個人事業主の方々は、確定申告でどうやればトクになるのか」は、大村さんが十八番とするテーマの一つ。みなさんが1年間かけて得た成果、それを知恵と工夫で、より多く手元に残す確定申告攻略法をレクチャーします。



フリーランス＆個人事業のための確定申告 改訂第 9 版

●山本宏　監修

A5 判／ 240 ページ
定価（本体 1480 円＋税）
ISBN 978-4-7741-6788-6

世界一ラクにできる確定申告

全自動会計ソフト「freee」で手間なく完結！
平成 27 年版

●原尚美・山田案稜　著

A5 判／ 272 ページ
定価（本体 1580 円＋税）
ISBN 978-4-7741-6806-7

恐怖の3月15日―確定申告の締切を目前に「ただでさえ忙しいのに、もっとラクにできないの？」と思いませんか。
そんな悩みにお応えすべく、勘定科目を覚えない、領収書の整理もしない一番ラクに確定申告を終える具体的な手順を丁寧に解説！「経費にできるかどうか」という悩みどころもしっかりフォロー。自動で帳簿付けしてくれる話題の全自動クラウド会計ソフト「freee」にも対応しています。

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

Contents

Software Design Jan. 2015

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)

Contents

Software Design Jan. 2015

Column

Development

OS/Network

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

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

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

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

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

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

DIGITAL GADGET

Volume 193

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

プラスワンで価値を増すデジタルガジェット

組み合わせの妙

皆さんはお持ちのスマートフォンをそのままで持ち歩いていますか？ それとも、バンパーやケースに入れて使っていますか?

スティーブ・ジョブズはケースに入れることを嫌がり、「傷のついたステンレスは美しいと思うけどね」と、経年劣化も美しい変化の1つだととらえていました。

ケースはファッションや個性を発揮する場所でもあれば、本体を傷などから守る意味もあり、軍隊仕様のヘビーデューティーなケースから、水中で利用できる防水ケース、キャラクタやウサギなどの動物の耳がついているような、可愛いけれども持ち歩きづらそうなケースまで、多種多様なものが存在します。

最近では、スマートフォンのヘッドフォン端子や、mini USBやLightning端子、カメラ部分などに機能を付加するようなグッズが増えてきました。これらは大きく分けて次のような種類に分類できます。

- 本来スマートフォンにはなかった機能を付け加えるもの
- デジタルデバイスの操作画面のためにスマートフォンを活用するもの
- 機能やしくみを加えることで、本物志向の何かに近づこうとするもの
- 今でもある機能を拡張、拡大、延長するもの（マクロレンズや追加バッテリー）
- 端子を本来の目的ではなく単に電源を取得するためや固定するためだけに用いるもの

最近一部で広がりつつあるスマートフォンのイヤホン端子に接続するカードスキャン用のデバイスや、自撮り用のセルフィー棒なども、手軽に機能を拡張するデバイスの1つでしょう。

プラスワンで、何か新しい価値を生み出す方法

プラスワン、何か機能を加えることで、それぞれ単独で存在していたものよりも価値を生み出すということがあります。さまざまなデバイスやコンテンツなどの中で、まったく新しい、過去にまるで事例がなかったようなものが生み出されるのは稀なことで、たいていは何かの改変や、新しい組み合わせ

25種類の道具が搭載されたLEATHERMANのマルチツール。ワイヤーカッター、ナイフ、ハサミ、のこぎり、やすり、ドライバー、キリ、定規、栓抜きなどがこれ1つに

iPhoneと組み合わせる工具。リョービの「Phone Works」。レーザー測量計、湿度計、拡大鏡、レーザー距離計、方位計、温度計といった種類がある

※本記事で紹介しているものは国内未発表・未発売のものを含んでおります。

プラスワンで価値を増すデジタルガジェット

などで新しい価値を生み出す場合が多いのではないでしょうか?

新サービスや、新デバイスをブレインストーミングによって考えるときに、オズボーンのチェックリスト[注1]と呼ばれる、よく使う9つの考え方があります。

❶転用したら?
今までとは違う分野への適用。本来の使い方ではない新しい使い道

❷応用したら?
似たもの、似た役目の代わりとして使うなど

❸変更したら?
色、形、動き、意味など、定番の仕様と異なるものにする

❹拡大したら?
従来のものよりも大きくしたり、時間的スケールを大きくする

❺縮小したら?
より小さく軽くしたら、エネルギー効率など何か省略できないか

❻代用したら?
既存の素材や動力源などを何か代わりのもので満たせないか

❼置換したら?
再利用、要素の順序を変更したり、形を置き換える

❽逆転したら?
回転方向を変えたり、時間的順序を変える

❾結合したら?
何かと何かを合体させたり、ブレンドさせる

実は世の中にある「新しい」と言われるもののほとんどは、上記9つのどれか1つ、または複数の考えによって生まれたものであることがわかると思います。

これからの価値とは?

最初のページに掲載したLEATHERMANのマルチツールのように、ありとあらゆる機能が一体化し、一見便利そうに見えるなんでもできる1台よりも、実用的に使うものは状況や場合に応じてシンプルな機能へとカスタマイズしたり、パーソナライズしたりといった使い方が適しているのではないでしょうか?

一方で、大工道具のようにある目的のために専用の道具があり、何かの目的を完遂するためには状況に応じた大量の道具が必要という場合もあり得ます。大工道具にはいくつかの大きなジャンル分けがあり、さらにそれ


Equalizer Case for iPhone 5:
音楽の周波数成分を表示するLEDイコライザー

Luxi - Light Meter Attachment For iPhone:
照度計


Binocular Adapter For iPhone 5:
双眼鏡アダプタ

Nix Color Sensor:
色センサー


PocketPlug:
ケース一体型充電器


taskone:
万能ナイフ

SensePlus-Smoke & Gas Sensor:
煙検知器


FLIR ONE:
赤外線カメラ


Smartest Thermometer:
体温計

PanoPal:
パノラマ撮影


AromaCase:
20mlの香水を入れられるケース


Old-School Calculator iPhone Case:
電卓搭載ケース

注1) 引用元　http://www.mycoted.com/Osborn's_Checklist

それに細かな変種があります。同じノミという道具でも、サイズや形状、使う場所や方向、用途によって、基本的な働きは変わらないまでも道具そのものが異なります。

また、道具を専用に作る職人や、大工自身が自分の道具を作る場合もあります。同じ道具でも、押して使う用途、引いて使う用途など、使い方に工夫がある場合もあります。

道具と呼ばれるものにはほかにも、料理道具、手術道具など、コレクションするだけで楽しいものから、需要は少なくとも、ある特定の用途には必要不可欠なものが存在します。道具は常に手入れをするもので、見栄えよりも使い心地、形態が意味をなし、その結果機能的で美しいものになっているものも存在します。

スマートフォンやコンピュータは道具としての基本性能がありつつ、汎用的に利用でき、ソフトウェアやアプリによって、その機能や使い方を変化させることができます。

Googleが推し進めるProject Araでは、スマートフォンのすべてのパーツがモジュール化します。利用状況に応じてカメラモジュールを交換したり、バッテリーのサイズを変更したり、端末の大きさそのものさえも変化します。

さらにヘッドフォンやスピーカーのように、エイジングという慣れさせるための時間が必要なものもあります。そういった、新品のときには使いづらくとも、時間がたつにつれてなじんでいく道具もあるでしょう。

道具に支配されるのではなく、使いこなすための道具。適切なサイズや重さ、機能を持ち、身体の延長として使えるもの。適切な道具は言葉や言語に関係なく、世界共通で活用できるものでしょう。

さまざまなテクノロジの進化によって、人間の道具も大きく変化する時代がやってきているのかもしれませんね。

SD

GADGET 1

WOOD KEYBOARD

http://rawbkny.com/products/macbook-wood-keyboard

MacBook用の木製キーボード

WOOD KEYBOARDはMacBookのキーボードや、Mac用ワイヤレスキーボードに木片を追加することで、木の質感をもったキーボードに変更できます。キーボードバックライトの機能も活かしたままでキー入力を可能にしています。キーを取り付けるための接着剤は着脱可能で、ノリの跡も残らないそうです。装着作業には30〜40分ほどが必要で、40〜45ドル（フルキーボード用）で販売中です。刻印は英文字日本語配列キーボードにも対応しています。

GADGET 2

Ledge

https://www.kickstarter.com/projects/appliedinc/ledge-for-macbook-made-in-usa?ref=nav_search

MacBook用ハンドレスト

LedgeはMacBookの手元に装着する、丸みを帯びた拡張ハンドレストです。MacBookのハンドレスト部分は角張っており、キーボードを使う姿勢や状況によっては手首に角が頻繁に当たって痛くなることがあると思います。Ledgeは手首に当たる部分に丸みを帯びた部品を取り付けることにより、滑らかな使い心地を得られるものです。スリープランプや赤外線ポートなどを隠すことなく、通常どおり利用できます。2014年11月時点ではクラウドファンディング、Kickstarterで資金を集めている段階です。

GADGET 3

BRACKET

https://www.kickstarter.com/projects/181279175/bracket-the-macbook-pro-retina-cable-dock

MacBook Pro用の端子ドック

BRACKETはMacBook Proの各種端子をまとめてすっきりと扱うことができるケーブルドックです。コンピュータの左側にある、電源、ディスプレイポート2系統、USBポート、ヘッドフォン端子をまとめてすっきりと接続することができます。普段はMacBookを持ち歩いて利用し、帰宅時や、帰社時には数種類の端子を接続して利用する際に、まとめて取り扱えるのが便利な点です。2014年11月時点ではクラウドファンディング、Kickstarterで資金を集めている段階です。

GADGET 4

Relonch Camera

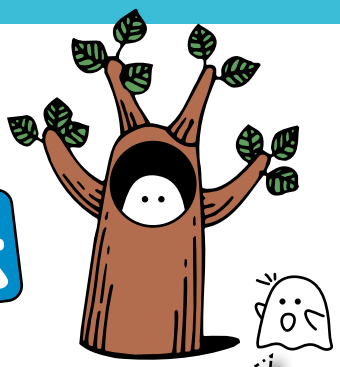
https://relonch.com

デジタルカメラジャケット

Relonch Cameraは、メモリカードを必要とせず、iPhoneと接続して利用するスマホケースタイプのデジタルカメラです。APS-Cセンサと F2レンズ級のハイエンドデジカメとしての機能を持ち、操作や画面の確認、ファインダーとしての役目はiPhoneの画面とタッチパネルが果たします。対応するiPhoneは5/5S/6の機種で、6 Plusには今のところ対応していません。iPhoneとの接続は無線ではなく、Lightning端子で行うケースタイプの製品です。499ドルで予約販売中です。

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

結城浩の再発見の発想法 20



Implement

Implement——インプリメント

インプリメントとは

インプリメント(Implement)とは、ソフトウェアを実際に作ることです。日本語では「**実装する**」と訳されることが多いですね。技術者同士ではそのまま「インプリメントする」と言うこともよくありますし、「インプリする」と省略して言う人もいます。“implement”という単語の語源は「**中を満たす**」というラテン語だそうで、おそらくそこから「定められた枠組みの中身を満たす」という意味が生まれたのでしょう(**図1**)。

インプリメント(実装)はデザイン(設計)と対比されて用いられることの多い用語です。ソフトウェアの開発で「設計はできたから、これから実装に入ろう」という使い方をしますし、できあがったものに対して「設計はいいのだけれど実装はまずいなあ」と表現することもあります。

設計とは何か、実装とは何かというのは大きなテーマなのでこのコラムで書くことは難しいですが、簡単に言うなら設計は《どのようなものを作るかを明確にすること》で、実装は《実際に作ること》と言えるでしょうか(**図2**)。

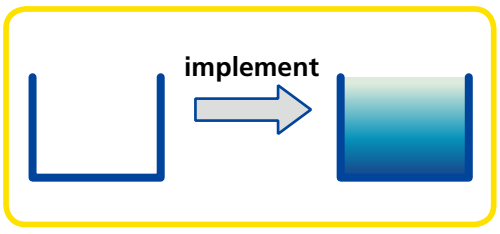
物理的なものを作る場合には、設計と実装は明確に分かれますが、ソフトウェアの場合は実装して作るものに物理的な実体はありません。ソフトウェアの「実装」を、製造業の「製造」と同じようなものとして考えるのは正しくないという意見もありますが、ここでは深入りしません。

設計と実装

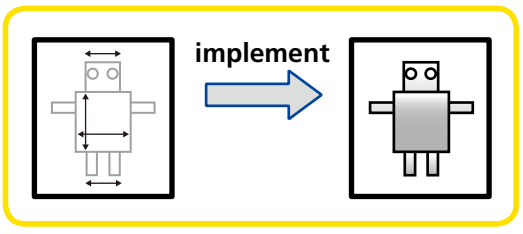
ソフトウェアは複雑なものですから、いきなり実装を始めて完成まで一気に至るのは難しいものです。あまり考えずに実装を始めると、途中で大きな問題を見つけてしまい、最初からやりなおしになることもあります。そのため、最初に設計を行って、解決すべき重要な技術的ポイントをチェックしておくことが大事になるでしょう。

設計をせずにいきなり実装してしまうと、特定の環境にべったりと依存したソフトウェアになってしまい、将来の拡張性が低くなることも

▼図1　implementの語源は「中を満たす」



▼図2　設計から実装へ



あります。設計の段階では将来の拡張性を考えておき、実装の段階ではそれを現実のコードに落とし込むことで、拡張性や移植性を高くすることもできるでしょう。

開発で、設計と実装を分けて考えると、実装のしやすさに引きずられて達成すべきゴールがねじ曲がるのを防ぐことが期待できます。ソフトウェアは、ある程度までできあがってくると「ほんの少し追加すれば実現可能な新機能」がたくさん見えてきます。しかし、もともと考えていた《どのようなものを作るか》を忘れてしまうと、余計な機能まで実装してしまう危険性があります。設計と実装を意識的に分けることは、ゴールを見失わないために重要です。

ソフトウェアの完成後に問題が生じたときも、それが実装上の問題なのか、設計上の問題なのかを分けて考えることは重要です。軽微な問題ならば実装を修正することで対処できますが、設計上の問題ならば実装の修正で対処することには無理が生じるでしょう。

設計と実装を分けて考えるためには、抽象度の高い発想が必要になります。単に「作ればいいんだろ」と短絡的に考えてはだめです。まず「このようなものを作りたい」と考え、「そのために、このようにして実現する」と考える。「設計」と「実装」を分けて考えることは、ものを作る場合でも、作ったものを改良するときでも、問題を解決するときでも重要なことなのです。

日常生活での実装

ソフトウェアに限らず、日常生活で何かを作るときにも「設計」と「実装」を分けて考えるのが有益です。

たとえば料理。いきなり料理を作り始めるのではなく、何を作るかを一通り最後まで考えるのは重要です。最後の段階で必要な食材が切れていることに気づき、ショックを受けた経験がある人は多いでしょう。これは、ソフトウェアの設計段階で、解決すべき重要な技術的ポイントをチェックするのに似ています。

たとえば会社の中でグループを作るとき。たまたまそこにいる人だけでグループを作って活動を始めるのではなく、何のためにどんなメンバーが必要かを考えておくことは重要です。達成したい目的を明確にすることで、現在はそばにいないメンバーに声を掛けるという発想も生まれるでしょう。これは、ソフトウェアの拡張性を考えるということにも似ています。

書籍執筆における実装

筆者は毎日、本を書く仕事をしています。「本を書く」という活動でも「設計」と「実装」の段階を分けて考えることができます。大まかに言えば「読者にどんなことを伝える本にしようか」を考えるのが設計段階であり、「実際にファイルの中身を文字で埋めていく」のが実装段階と言えるでしょう。

書籍の「設計」において大事なのは、自分の手に負えるテーマになりそうかどうかという見極めです。「実装」すなわち実際の執筆がかなり進んでから「このテーマは自分には大き過ぎた」となってはたいへんなことだからです。書こうとしている書籍を成り立たせる重要な技術的ポイントのチェックは最重要課題なのです。

自分が「書きやすい本だけを書いていく」ような過ちを犯さないようにすることも大事です。自分が書きやすい本というのは、実装のことだけを考えて作るソフトウェアのようなものです。完成させたいものは何かという設計をしっかりしないと、せっかく書き上げても読者に受け入れられない本になりかねません。

設計と実装。この2つを分けて考えるのは重要なことなのです。

◆ ◆ ◆

あなたの周りを見回して、「何かを作る」という場面を探してください。そこでは「設計」と「実装」は分かれているでしょうか。何を作るかを明確にする設計段階と、どのようにして実現するかという実装段階は意識されているでしょうか。

ぜひ、考えてみてください。SD

耽溺せよ電子工作

おとなラズパイリレー

第3回

「Raspberry Piで空を飛びたい（前編）」

千葉 久詞

おとなラズパイリレーは、Raspberry Piを文字どおり「リレー」し、ちょい悪ITエンジニアが電子工作をするという企画です。前編で構想を練り、後編で実装します。1年を通してどんなデバイスができあがるのか？……今回は、データホテルから社名変更も間もない、テコラス㈱によるRaspberry Piドローン開発です。

Writer 千葉 久詞（ちば ひさし） テコラス㈱ドローン倶楽部

Raspberry Piとの出会い

2012年に初めてRaspberry Piと出会ったことを今でも覚えています。「この値段で、この性能で、この消費電力!?」――これで購入を即決しました。この出会いの数年前から筆者は自宅でサーバを24時間稼働させていたのですが、一番の悩みの種は月々の電気代でした。実家暮らしだったので「電気代が高すぎる！」と、よく怒られていました。今でこそ親から怒られることはありませんが、電気代に怯える日々に変わりはありません。そんなRaspberry Piの存在は、筆者にとってうってつけのものでした。

しかし、サーバ利用ではなく、Raspberry Piで電子工作することは、なかなか手をだせずに今に至ります。無精な人間によくある「いつかやろう」症候群だったのです。

Raspberry Pi B+で何をしてみようか

電子工作といえば、なぜそこに抵抗やコンデンサが必要なのか、その原理も理解していませんし、フレミングの左手の法則やオームの法則なんて……何でしたっけそれ？――というような状態です。筆者にとってハードルが異様に高い電子工作を、1人で始めるにはあまりに心細い状況でした。そんなときに、当社でドローン倶楽部という活動が立ち上がりました（i部長の暗躍があったのかもしれませんが……）。まさに渡りに船！――入会を即決しました（**写真1**）。

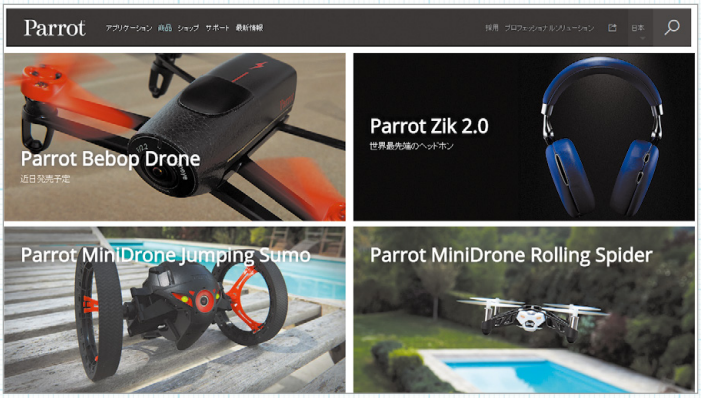
ドローン（Drone）とは無人機を意味します。遠隔操作または自律制御を用いて、機体を操作するものです。その定義によって無人機は陸海空すべてに存在します。が、本稿ではUAV（Unmanned Aerial Vehicle：無人航空機）について説明します。

UAVには航空機型とヘリコプター型があり

▼ **写真1** ドローンを手にした渡邉（左）と千葉（右）

ます。TVのニュースでよく報道されているせいで軍事用途のイメージが強く遠い世界のものであると、どうしても受け止められがちですが、現在では民生用のドローンも増えてきています。ネット上では手軽な価格帯での商品がラインナップされ、それを購入すれば家庭でも遊べるようになりました。個人でもドローンをリモコン操作して遊んだり、ドローンにGoPro[注1]のような広角レンズの小型カメラを搭載して、従来ならば困難だった映像を撮影して（多くは鳥瞰的な動画ですね）、楽しんだりする人が増えてきています。最近では、Amazonがマルチローター型ヘリコプターのドローンを用いたPrime Airという宅配サービスを構想していますし[注2]、日本でも農業用、産業用に各社が研究を重ねています。まさにこれから成長する可能性のある分野だと言えます。

▼図1　AR.Drone2.0のWebページ



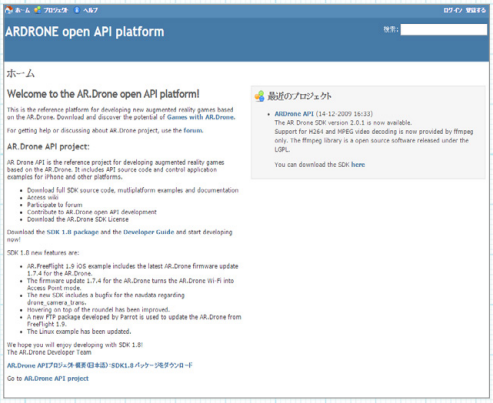
今回、我がドローン倶楽部ではAR.Drone2.0[注3]というUAVを購入しました。これを操作するには、PCかスマホといったデバイスが必要です。このUAVに、Raspberry Piを搭載してドローンを飛ばすことが、もしもできたら――それは自律制御されたドローンになるのではないか、と閃いたのです。これが完成したら、いろいろなイベントに応用できて面白いかも！――というわけです。そこでドローン倶楽部のメンバー有志と自律型ドローンの開発を検討してみることにしました。

AR.Drone2.0について

さて、話を少し戻し今回購入したAR.Drone2.0について話をします。AR.Droneは、フランスのParrot社で2010年から発売されているクワッドローター（ヘリの羽が4枚ある）のヘリコプター型ドローンです（**図1**）。当時から家庭向けに発売されています。今回ドローン倶楽部で購入したのはそのリニューアル版として2012年から発売されている商品で、世界で実に合計60万個以上の販売実績あるモデルです。その理由は、ラジコンの部類としては割とリーズナブルな価格であること、しかも高性能であることに因ります。そのサイズは、全長51.5×全幅52.5cm、重量455gと極めて小型です。基本的な飛行性能に加え、Wi-Fiで50mまで離れて操作できます。さらに標準でHDカメラを搭載しています。付属しているセンサーだけで6種類ありまして、オプションのGPSセンサーも含めれば、実に7種類ものセンサーに対応できます。航続時間は標準搭載のバッテリーならば12分程度です。操作用のアプリケーションもデフォルトで付属しています。しかも公式サイトよりSDKが配布されており、自分で操作用のプログラムも作成できます。公式のSDKはC言語でコーディングされていますが、国内外でさまざまなSDKの開発プロジェクトが進ん

注1）http://jp.gopro.com/
注2）http://www.amazon.com/b?node=8037720011
注3）http://www.parrot.com/jp/products/

▼図2　ARDRONE openAPI platform
(http://projects.ardrone.org/)



▼写真2　ラズパイを搭載したドローン

でいます(**図2**)。これによりさまざまな媒体と言語で、AR.Droneのプログラムの開発ができるようになっています。

Raspberry Pi ＋AR.Drone2.0

AR.Drone2.0は単体で使用しても十分楽しめる性能を持っていますが、少し不満点もありました。搭載カメラが水平方向に付いているので、見下ろす形の鳥瞰映像を撮ることができないことです。さらに拡張性に乏しい点も不満でした。

前述のとおり、AR.Drone2.0の操作にはスマホかPCが必要で、SDKを用いてプログラムを開発できます。前述のセンサーを状態検知に使用し、操作にRaspberry Piを適用すれば、既存の製品よりも、ずっと可能性が広がるはずです。

この計画を進めるにあたって、2つの問題点が浮かび上がりました。1つめは「消費電力量の問題」です。AR.Drone2.0に標準搭載のバッテリー容量は1,000mAh程度です。低消費電力とはいえ、Raspberry Piを起動して、さらに安定した電力を供給しつつ、持続的な飛行を維持することが可能かという問題です。

2つめは「積載量」の問題です。AR.Drone2.0は物を積載して、飛ぶようには通常は設計されていません。今回、Raspberry Piをはじめとして、その他各種パーツを積載した場合の重量が問題になりそうです(**写真2**)。果たして、これだけの装備を加えた状態で飛行が可能でしょうか。本稿ではこの2点を事前に検証してみます。

基礎データを集めよう

今回購入したドローンは、AR.Drone2.0の中でもPower Editionと呼ばれる製品です。これは同社の従来型の製品と比べて、バッテリー容量が1,500mAhで、1.5倍に増えています(**写真3**)。今回はこのモデルを基準として調査をします。

その1「消費電力量の問題」

まず、消費電力量の問題です。AR Droneの消費電流は、1,500mAhのバッテリーで18分程度飛行可能なことから推定5,000mA程度だと推測できます。

また、オプションパーツのGPSユニットとRaspberry Piを合わせた消費電流が平常時で440mA程度と計測できました。搭載する予定のパーツを接続した段階で、プログラムを動かしているときに計測してみると、平均して

▼写真3　ドローンのバッテリーパック

800mAの消費電流という結果が出ました。実際には、多めに見積もって合計6,000mA消費するとして、15分前後の継続飛行が可能という試算が出ました。これは、外部バッテリーの追加が必要になるだろうと予測していた筆者にとって嬉しい誤算となりました。実は、後で判明したことですが、今回使用したRaspberry Pi Model B+は前モデル（Raspberry Pi Model B）と比べて消費電流量が低くなっていました。この結果から、追加で外部バッテリーの搭載は不要になりそうです！

その2「積載重量の問題」

次に積載重量について調査を行います。調査方法はいたって単純です。AR.Drone2.0の中心部分に適当な重量のおもりを載せてから飛行させてみました。今回は重量の調整のしやすさのために280ミリリットルの水の入ったペットボトル（300g）をおもりとして使用し、50gずつ水を減らすことで実験を行いました。その結果、200gまでならば飛行可能で問題ないことがわりました。さらに細かく検証するならば、もう少し荷重をかけられそうでしたが、あまり積載量を多くしすぎると、今度は離陸までの動きが鈍重になってしまい、モーターに負荷が掛かる恐れがあります。そこで限界を200gまでとしました。荷重バランスについては検証していませんが、過大に負荷をかけなければAR.Drone 2.0は自動的に飛行制御してくれます。あとは機体の中心にバランスが良いようにRaspberry Pi Model B+を設置すれば大丈夫でしょう。

目標は赤外線カメラ搭載自律ドローン

今回、AR.Drone2.0を初めて見て感じたことは、「ああ、メタルギアソリッドシリーズ注4に出てくるサイファーにそっくりだな」でした。本シリーズをプレイした経験がある方はご存じかと思いますが、監視カメラを搭載したドローンが空中を決まったコースで監視していき、問題があった場合は通報します。実は、これは技術見本として90年代にシコルスキー社が開発した実在する機体だったのです（**写真4**）。

性能は限定的になってしまうかもしれませんが、Raspberry Pi B+とAR.Drone2.0の組み合わせであれば、この装置が実現可能なのではないかと考えていますので、制作目標は「自律飛行を行いつつ、赤外線カメラで撮影を行うドローンの制作」とします。次号、実際に制作を行い、動作検証をしていきます。もちろん電子工作初心者なのでできる範囲で……。SD

◀写真4　シコルスキー社のサイファー

注4）http://www.konami.jp/mg25th/truth/

軽酔対談

かまぷの部屋

第6献 ゲスト：長田 絵理子さん

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

長田絵理子（おさだえりこ）さん

ジュンク堂池袋本店勤務。コンピュータ書フロア担当。普段は書店で接客する傍ら、ITイベントに積極的に出店し、エンジニアや著者と自ら交流する。出版社と全国の書店の人的ネットワークを駆使して、在庫のない本でも探し出す。彼女が作る良書案内は、編集者の貴重なデータとして重宝される。

本を1冊1冊大切にしています。

USP研究所の鎌田広子（かまぷ）と申します。本連載が始まり、半年経ちました。気軽に飲んでいる気分で読んでいただけると嬉しいです。今回は書店でコンピュータ書を担当されている、書店員の長田絵理子さんにお話を伺います。

（鎌田）長田さんはITイベントに出展されていらっしゃることもありますよね。

（長田）はい。ITイベントでの販売も大好きです。書店内ではお客様から質問されない限り話しかけたりはしませんが、イベント販売の場合はお客さまとの距離が近いので、とても楽しいです。またイベントだと著者の方がブースに立ち寄ってくださることもあり、それも嬉しいです。

書店員はあこがれだったのですか？

十代の頃は、海外やアジアの文化に興味があって、大学ではインドのことを勉強していました。漠然と、将来はバリバリのキャリアウーマンにはならないだろうと思っていました。就職活動は、書店だけではなく、軍手の工場や出版社など、おもしろそうだと思うところしか受けなかったです。軍手の工場は職場がマレーシアだったりして、そこで働く自分を想像するのは楽しかったです。

最近の電子書籍の動向はどう思いますか？

書店が販売する本は手で触れられますが、電子の本はさわれないですよね。質感のあるリアルな本は、人の所有欲を満たしているのだと思います。本は装丁や内容に至るまで、編集さんがきちんと内容を見ています。そういったことが本の品質を高めているのだと思います。ですから書店として、きれいな本をお客様に届けられるように努力しています。また、まだ電子出版をされていない出版社さんもありますので、品ぞろえ数は書店のメリットです。

IT書籍の最近の動向や、仕事上の苦心談、裏話などありますか？

秋はJava 8系本の売れ行きがいいです。品揃えを工夫するのはもちろんのこと、店頭や電話でお客さまが必要とされている本探しをお手伝いしたり、具体的な質問に答えたりもしています。店頭での苦労と言えば、たとえば今年はiPhone 6が出ましたが、各キャリア2機種分で6種類の本が出版社ごとに出てくるので、かなりの種類になりました。そういった本をいかに陳列するか、などにも気を遣っています。それと年末期は、年賀状素材集がたくさん出ますが、実はその写真素材の中に業界関係者の家族写真が使われている場合もあって入荷が楽しみだったりもします。




へぇ、書店と出版社の関係もおもしろいですね。質問の話が出ましたが、具体的にどんなことを聞かれますか？

入門書関連では、パソコン教室などで教えるような、具体的な使い方が理解したいために来店いただく方も多く、ピンポイントな質問をされる場合もあります。コンピュータ書のフロアですから、スタッフもなるべくご希望の書籍を見つけられるようにお手伝いしますが、すべてをお答えできるわけではないのでお叱りを受ける場合もあります。IT技術は広く学ぼうと思うと果てしなくて棚作りは難しいです。でも、高度な技術書をお探しのお客様は、どういう本が欲しいというのが具体的で、場所を教えてくださいという場合が多いです。

なるほど、いろいろ大変ですね。個性的なお客様も多いのでしょうか？

うちは図書館みたいな雰囲気で、本を自由に読める椅子があって、そこで静かに読んでいる人が多いです。そうそう、311の震災の時は、建物が揺れているのに本を開いたままで読んでいるお客さんもいらっしゃいました。

地震のとき、店内は大丈夫でしたか？

コンピュータ書棚の場所の揺れもひどくて、たくさん本棚から抜け落ちてしまいました。そのとき、何よりお客様に安全に退避していただくためにがんばって声を出そうとしたのですが、声が震えてしまったのをおぼえています。実は震災の前日は、入籍日だったんです。そのため遅番出勤で14時から出版社の営業の方と商談中でした。でも翌日の午後までには、すべて本を書棚に戻してその日のうちに営業再開にしました。

そうだったんですか！　そろそろ、ほろ酔い気分になってきたので、個人的なこともお聞きしたいのですが、旦那さんはどんな方ですか？

シドニー出身で、今はCookpadで働くソフトウェアエンジニアです（英語が流暢な日本人みたい、と言われます）。ITに関しても教えてもらったり、Googleカレンダで予定を共有したりして、夫婦生活は楽しいです。パソコン買い替えの相談をしたら「なぜMacじゃダメなんだ！」と聞かれました。それ以降、自分もMacユーザになり、今や家中Macだらけです。

ところで旦那さんとはどこでお知り合いになったんですか？

RubyKaigiにジュンク堂が出店させていただいていまして、そこで彼が『インドカレー食べにいかないカ？』って声かけてくれました。それまでは、結婚願望はなかったのですが、彼と会ったとき「結婚したい！」って思いました。

ご馳走様です♡。ところで今回お誘いするときに、糖質制限されているとお聞きしましたが……。

彼がダイエットしたいということで家庭で始めました。ひとまず2ヵ月間2人で頑張ろうと糖質制限とジム通いをしています。fitbitを下着につけて（笑）。iPhoneアプリと連携してデータ集計しているのですが、たまに液晶画面がスマイルで、ニッコリ笑ってくれるんですよ。運動は生活にメリハリが出ていいです。糖質制限を始める直前に最後に食べたのはいくらごはんでした。自作のイクラ漬けです。

奇遇ですね。私も友人に触発されて、今年初めてイクラ漬けを作りました。お料理は得意ですか？

料理はストレス発散としても好きです。肉をじゅ〜っっと焼きながら「くそー！」とか。この間は鰯の梅生姜煮を作りました。あ、お酒も好きですよ。著者の“きたみりゅうじ”さんは新刊が出ると、毎回お店にPOPを描きに来てくださるのですが、その後、きたみさんと出版社の方と美味しい日本酒を飲みに行ったりもします。

美味しい日本酒いいですね。今日は楽しいお話、どうもありがとうございました。




秋葉原発!

第51回

はんだづけカフェなう

Edisonで IoTしよう

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

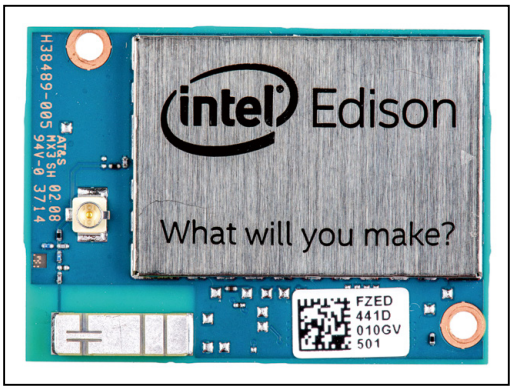
Intel Edison

Intelが10月25日に発売した、Edisonをすでにご存じの読者もたくさんいらっしゃるでしょう（**写真1**）。Edisonは、この連載でも紹介してきたGalileoよりもはるかに小さく、SDカードに近いサイズのx86コンピュータです。EdisonのメインプロセッサはGalileoのQuarkからAtomになりました。

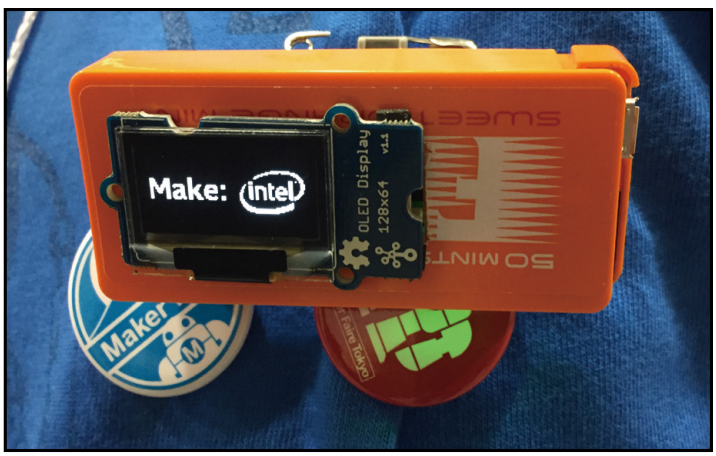
こんな小型のEdisonですが、1GBのRAMと4GBのFlash、IEEE 802.11a/b/g/nのWi-FiとBluetooth 4.0 + 2.1 EDRといった無線のインターフェース、USBインターフェースに加え、40個のGPIO（汎用入出力）を備えています。つまり、高速ではないものの、Edisonはそれ単体でコンピュータとして動作できます。Edisonは小型ながら、このようにたくさんのインターフェースが搭載されていますので、高密度の70ピンのコネクタで外部と接続するように作られています。

Edisonと同時に、「インテル Edison キット For Arduino」と「インテル Edison Breakout ボードキット」も発売されました。For Arduinoのほうは、EdisonのGPIOをArduinoフォームファクタの端子から利用できるようにする拡張ボードが同梱されています。一方、Breakoutボードキットは、フリスクより少し大きいサイズで小型な、先述の70ピンのコネクタを利用しやすいサイズに変換する拡張ボードが同梱されています。For Arduinoは、GalileoやArduinoなどのマイコンと同様に、入出力の電圧を5Vや3.3Vに変換する機能が搭載されています。Breakoutボードのほうは、電圧を変換する機能が搭載されておらず、Edisonの入出力電圧である1.8Vのままです。最近は、入出力電圧1.8Vに対応した

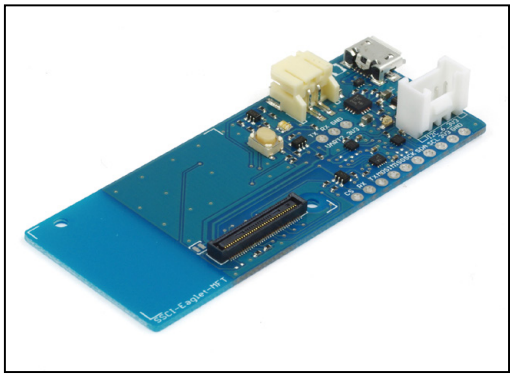
▼写真1　Edison



▼写真2　Eagretを使ったOLEDバッジ



▼写真3　SSCI-Eaglet



▼写真4　SSCI-Eagletをフリスクに組み込んだところ

センサなどのチップも出てきていますが、いまだ主流ではありません。多くのチップの入出力電圧は3.3Vないしは5Vですので、For Arduinoのほうが手に入れてすぐに使うことができます。また、ソフトウェア開発環境であるArduino IDEを使った開発でも、ターゲットとして用意されているFor Arduinoを使ったほうが調べる必要のある項目も少なく、快適です。

Eaglet

先ほどEdison Breakout ボードキットはフリスクより少し大きいサイズだと書きましたが、オフィシャルではないものの、インテル社員の方が基本設計をした、Eagletというフリスクのケースに入るサイズの拡張ボードがあります（**写真2**）。このEagletは、電圧を3.3Vに変換したGroveコネクタ形状のI²C端子が付いています。また、USBコネクタと加速度センサ、リチウムイオンバッテリの充電回路も搭載しています。Eagletは小型ながら、Edisonをバッテリで動かして手軽に使える拡張ボードです。そんな便利なEagletですが、Hackathon参加者だけに配布されるなど、入手する手段は限られています。

スイッチサイエンス版Eaglet

そんなEagletをもとに、筆者はスイッチサイエンス版Eaglet（SSCI-Eaglet）というものを設計しました（**写真3**）。Eagletのフリスクのケースに入るサイズはそのままで、3.3VのI²C端子に加えて、SPIやUARTといったほかにも一般的に使われるインターフェースを追加しました（**写真4**）。また、GroveコネクタにつながっているI²Cとは別に、Edisonに搭載されているもう一系統のI²Cも使えるようにしています。この入出力端子は、前回紹介したTesselのボードが接続できるようなピン配列にしました。使える入出力を増やしたために基板のスペースが足りなくなっていますので、皆が使うとは限らない加速度センサを廃止しました。リチウムイオン電池を利用する回路も、Edison Breakoutボード相当のものにして、より使いやすくなっています。

Eagletは、USBからの電力供給か、リチウムイオン電池からの電力供給で動かすことを前提に設計されています。リチウムイオン電池で動かしているときには、USB OTGを使おうとするときに必要となる5Vを作り出す回路が搭載されていないため、EagletではUSB OTGを使うことができません。

このスイッチサイエンス版Eagletは、11月23～24日に開催されたMaker Faire Tokyo 2014のIntelのワークショップで配布された（**写真5、6**）ほか、参加者以外の方でも入手できるようにスイッチサイエンスで量産をして販売しています[注1]。

注1）　http://ssci.to/2070

Edisonを手に入れたらやること

EdisonのOSはYocto（ヤクト）Projectを使用してビルドされた組み込み用Linuxが採用されています。Arduino IDEで開発したスケッチはelfバイナリとして実行されますし、C/C++のみならず、PythonやNode.jsなどを使った開発ができます。また、Intelから提供されている開発環境には、前述のArduino以外にもXDK[注2]という環境もあります。Linuxですので、Rubyをインストールして使っている人もいます。

Edisonに搭載されているLinuxは、かなり頻繁にアップデートがなされています。Edisonを入手したら、まず、ファームウェアを最新のもの[注3]にしましょう。“Edison Yocto complete image”という圧縮ファイルで最新のファームウェアが配布されています。IntelのWebサイトでは、シリアルコンソールを使ってファームウェアを更新する方法が紹介されています。が、筆者としては、Remote Network Driver Interface Specification（RNDIS）を使ってEdisonにSSHでログインしてEdisonの操作をする方法をお勧めします。この方法は、スイッチサイエンスのWiki[注4]で紹介されています。

Edisonの開発環境

みなさんPythonやJavaScriptを使った開発をしているようですが、ソフトウェアエンジニアとして三流かつオッサンの筆者は、C++でEdisonのI/Oを操作してみています。開発言語からEdisonのI/Oを操作するには、libmraa[注5]というライブラリを使います。先ほど簡単にEdisonの開発環境を紹介しましたが、Edisonは

注2） https://software.intel.com/en-us/html5/xdk-iot

注3） https://communities.intel.com/docs/DOC-23242
注4） http://trac.switch-science.com/Wiki/IntelEdison
注5） https://GitHub.com/intel-iot-devkit/mraa

▼写真5　Maker Faire会場でのワークショップの様子

▼写真6　Maker Faire会場でのインテルワークショップで発表する筆者（右下）

使う開発環境によって入出力に使うポートの指定方法が異なります。たとえば、Arduinoでは D13のポートはEdisonのGP40に接続されています。このGP40は、libmraaでは37番になっています。このような読み替えが少し面倒です。

先ほどのスイッチサイエンス版Eagletには、D13のポートに接続されたLEDが搭載されていますので、LEDを点滅させてみるにはリスト1のようなコードを書きます。

ちなみに、同じ処理をJavaScript(Node.js)で書くとリスト2のようになります。Pythonではリスト3のようになります。

このように、Edisonでは、ご自身が使い慣れた言語を使って開発を行うことができます。この点もEdisonの魅力の1つと言えるでしょう。

Linuxマシンとしての Edison

EdisonのOSはLinuxですので、小型のLinuxマシンとして楽しむことができます。たとえばSambaをインストールすれば、ポケットに入るNASになるでしょうし、httpdをインストールすればWebサーバとなります。VNCを使うなどして、EdisonでX Window Systemを動かしてみた人もいます。サイズの割にはEdisonのLinuxの動きはなかなか良く、ブートも待たされません。Raspberry Piよりは相当に高速に動きます。

Yoctoのパッケージはあまり豊富とは言えません。このため、EdisonをDebian化して使ってる方もいらっしゃいます。ubilinux for Edison[注6]というディストリビューションが配布されていますので、これをEdisonにインストールするのも1つの選択肢です。

まとめ

Intel EdisonはEagletと組み合わせることで、お菓子のケースに入る極小サイズでLinuxが動くという、Webエンジニアの方でも手軽に遊んでもらえるおもしろい環境です。入出力電圧が1.8Vという難点がありますが、これもEagletなどで3.3Vに変換すれば、Groveなどをつなげて、あまりハードウェアの知識がなくても何かを作ってみるという経験ができます。スイッチサイエンスのWikiにも、Physical Webのビーコンを作ってみるなど、いろいろなレシピが掲載されています。Raspberry Piよりは高いですが、もっと気軽にIoTを始めてみることのできるボードだと思います。SD

注6) http://www.emutexlabs.com/ubilinux

▼リスト1 LEDを点滅させるコード(C++版)

```
#include <mraa.hpp>

int main() {
  mraa::Gpio* led = new mraa::Gpio(37);
  led->dir(mraa::DIR_OUT);
  for (;;) {
    led->write(0);
    usleep(300000);
    led->write(1);
    usleep(300000);
  }
}
```

▼リスト2 LEDを点滅させるコード(JavaScript版)

```
var mraa = require('mraa');
var led = new mraa.Gpio(37);
led.dir(mraa.DIR_OUT);
var on = false;

blink();

function blink() {
  led.write(on ? 1 : 0);
  on = !on;
  setTimeout(blink, 300);
}
```

▼リスト3 LEDを点滅させるコード(Python版)

```
import mraa
import time

led = mraa.Gpio(37)
led.dir(mraa.DIR_OUT)

while True:
  led.write(0)
  time.sleep(0.3)
  led.write(1)
  time.sleep(0.3)
```

PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは 2015 年 1 月 17 日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

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

01 アークタッチ Bluetooth マウス

1 名

本体を折り曲げると電源がオンに、平らにするとオフになるマウス。左右対称デザインなので、OS 上でクリックの左右を入れ替え、左手でも使えます。スクロール部分には、なでた際に感触と音によるフィードバックがあるタッチセンサ「タッチストリップ」を搭載。接続には Bluetooth 4.0 ワイヤレスを採用しています。対応 OS は、Windows 8/8.1/RT/RT 8.1。

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

02 裸族の一戸建て SATA6G

1 名

※製品に HDD/SSD は付属しません

6Gbps の高速データ転送に対応した、SATA HDD/SSD（25.4mm、3.5 インチ）専用ケース。工具不要で簡単に組み込むことができます。PC への接続には USB2.0/3.0、eSATA を使用します。対応 OS は、Windows Vista/7/8/8.1 および、Mac OS 10.6.8 以上 。

提供元 センチュリー **URL** http://www.century.co.jp

03 弥生会計 15 スタンダード

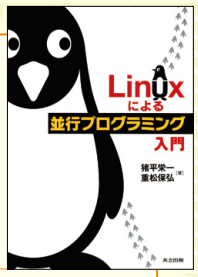
1 名

日々の記帳から集計・決算まで、1 本で行える会計ソフト。法人決算／個人決算（青色／白色申告）や、平成 26 年分の確定申告書の作成もサポートしています。Windows Vista/7/8/8.1 と各種 Windows Server に対応しています。

提供元 弥生 **URL** http://www.yayoi-kk.co.jp

04 Linux による並行プログラミング入門

2 名



猪平 栄一、重松 保弘 著／
B5 判、200 ページ／
ISBN = 978-4-320-12380-9

マルチプロセス／マルチタスク OS 上での並行プログラミングを、C 言語の演習問題を解きながら学べる本。Linux をベースに、シェルやプロセスといった OS の基礎概念も併せて解説しています。

提供元 共立出版 **URL** http://www.kyoritsu-pub.co.jp

05 新装改訂版 Linux のブートプロセスをみる

2 名

白崎 博生 著／
B5 変形判、312 ページ／
ISBN = 978-4-04-891393-5

ブートローダの起動から init の開始までの「ブートプロセス」を、ソースコード（C 言語／アセンブラ）を基に解説した 1 冊。コアなテーマながら、軽妙な文体でわかりやすく書かれています。

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

06 Git バージョン管理

2 名

松島 浩道 著／
A5 判、320 ページ／
ISBN=978-4-7973-8036-1

バージョン管理システム「Git」について、CUI ／ GUI ツールでの基本操作から実践的な活用法、Git 関連サービスまでを詳しく解説しています。サブコマンドのリファレンスも載った実用的な 1 冊。

提供元 SB クリエイティブ **URL** http://www.sbcr.jp

07 15 時間でわかる Java 集中講座

2 名

宮下 明弘、工藤 雅人、原田 僚 著／
井上 誠一郎 監修／
B5 変形判、416 ページ、DVD1 枚／
ISBN = 978-4-7741-6798-5

短時間で業務レベルの基礎知識を習得することを目指した独習形式の Java の解説書です。Vmware Player による仮想環境を付属の DVD-ROM からコピーすることで、すぐに学習を開始できます。

提供元 技術評論社 **URL** http://gihyo.jp

第1特集

使うほどなじむ

「Vim使い」事始め

プログラマ・インフラエンジニア・文章書きの心得

正直なところ、Vim は最初の敷居が高いエディタです。にもかかわらず、世界中のエンジニアに愛用され続けています。エンジニアを惹きつけてやまない魅力――その片鱗は、本特集の 2 つのコラム記事から感じ取ってもらえるはずです。

そして、Vim を常用エディタにできるかどうかは「Vim で快適に仕事ができるか」にかかっています。2 ～ 4 章は、その道で日々活用している Vim 使い (Vimmer) から、実務で効果を発揮するポイントを紹介してもらいます。

“一年の計は元旦にあり” と言います。2015 年は Vim で書き初めならぬ、エディタ初めはいかがですか？

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。

- 半角スペース……␣
- Esc キー……Esc または Esc
- Ctrl キー……Ctrl または Ctrl
- Ctrl キーを押しながら a キーを押す……Ctrl -a

第1章 犬でもわかる!? Vim導入&カスタマイズの超基本

Writer 林田 龍一(はやしだ りゅういち) URL https://twitter.com/Linda_pp Mail lin90162@gmail.com

Vimの導入はそれほど難しくありません。本章ではこれからVimを使い始めたい、Vimをちょっと使ってみようかなという方を対象に、愛犬家の筆者がVimのインストールから各基本機能の学び方、設定の基礎まで説明します。

インストールおよびセッティング

Vimの魅力を語りたいのはやまやまですが、百聞は一見にしかず、早速Vimをインストールしてみましょう。Vimは20年以上前からある有名なエディタですので、各OSやディストリビューションでパッケージが用意されており、簡単にインストールすることができるようになっています[注1]。

Windows

WindowsでVimを使うなら、香り屋さんが提供されている「KaoriYa Vim」がお勧めです[注2]。更新が活発で、便利な追加機能やスクリプトが加えられています。

香り屋さんのサイトから自分の環境に合ったバージョンをダウンロードし、お好きな場所(たとえば、C:¥vim)に展開してください。ファイルパスに空白が含まれるとうまく動かないプラグインがあるため、避けることをお勧めします。

展開したフォルダ内にあるvim.exeがCLI(Command Line Interface)版のVim、gvim.exeがGUI(Graphical User Interface)版のVimです。gvim.exeのショートカットをデスクトップに作成しておきましょう。

Mac

Macではアプリ単体として配布されている「MacVim KaoriYa」を使う方法と、Mac用パッケージマネージャのHomebrewを使って「MacVim」をインストールする方法とがあります。デフォルトの状態でも/usr/binにVimは入っていますが、Mac向けにビルドされたものではないのでMacVimをインストールすることをお勧めします。

MacVim KaoriYa

MacVim KaoriYaはKaoriYa Vimの便利な追加機能やスクリプトをMac向けのVim実装であるMacVimに適用し、MacでKaoriYa Vimの使い勝手を目指すプロジェクトで、splhackさんによってメンテナンスされています。次のURLよりDownloadsタブを選択してダウンロードしてください。

https://code.google.com/p/macvim-kaoriya/

Macのパッケージになっているため、ダウンロードしてきたファイルをダブルクリックするだけでインストールが完了します。コマンドラインから利用するには$PATHに/Applications/MacVim.app/Contents/MacOSを含め、Vimコマンドで起動します。

注1) 本稿の情報は2014年11月上旬のものです。
注2) http://www.kaoriya.net/software/vim/

Homebrew＋MacVim

MacVim KaoriYaはお手軽ですが、少々バージョンが古かったり、Lua連携などの一部機能が使えません。そういうときはHomebrewを利用します。Homebrewには多数のMac向けパッケージがあり、MacVimもHomebrewを用いてインストールすることができます。Homebrewをまだインストールしていない人は、次のURLにある説明に従って、まずはHomebrewをインストールしましょう。

https://github.com/Homebrew/homebrew/blob/master/share/doc/homebrew/Installation.md

完了したらbrewコマンドを使って、**図1**のよ

▼図1　Homebrewを使ったMacVimのインストール

```
$ brew install macvim --with-cscope --with-lua --override-system-vim
$ brew linkapps
```

コラム　ソースコードからのビルド

用意されているパッケージを使ったインストールは簡単ですが、最新のバージョンのVimが使いたいときやパッケージで提供されているVimで有効になっていない機能を利用するには、ソースコードからVimをビルドする必要があります。ここではUbuntuでソースからビルドする方法を紹介しますが、他のディストリビューションでも同様にしてビルドできるはずです。Windowsについては、KaoriYa Vimで必要になる機能がすでにほぼすべて有効になっているため割愛します。

図Aのようにして、まずはビルドに必要なパッケージと依存パッケージをインストールします。Lua連携機能を有効にしたい場合はlua5.2 liblua5.2-devも必要です。

Vimのソースコードは Mercurialで管理されているため、**図B**のようにソースコードを取得します。

続いて、ビルド設定を行います(**図C**)。詳しいビルドオプションの一覧は`./configure --help`で見ることができます。LuaやRuby、Pythonの連携機能を使いたい場合は`--enable-rubyinterp`、`--enable-pythoninterp`、`--enable-luainterp`オプションをそれぞれ追加します。

最後にビルドし、インストールします(**図D**)。これで`/usr/local/bin/vim`にVimがインストールされます。

▼図A　ビルドに必要なパッケージをインストール

```
$ sudo apt-get build-dep vim
$ sudo apt-get install libxmu-dev libgtk2.0-dev libxpm-dev mercurial
```

▼図B　Vimソースコードの取得

```
$ hg clone https://vim.googlecode.com/hg/ vim
```

▼図C　ビルド設定

```
$ cd vim/
$ ./configure --with-features=huge --enable-gui=gtk2 --enable-fail-if-missing
```

▼図D　ビルド＆インストール

```
$ make -j
$ sudo make install
```

うにインストールします。CLI版のVimはvimで、GUI版のVimはLaunchpadからMacVimを選択するか、コマンドラインからopen -a MacVimで起動します。

Ubuntu

UbuntuではVimのパッケージが標準のパッケージマネージャaptによって提供されています。

```
$ sudo apt-get install vim
```

コマンドライン内ではvimで、GUI版はコマンドラインからvim -gで起動できます。

CentOS

CentOSではVimのパッケージが標準のパッケージマネージャyumによって提供されています。

```
$ sudo yum install vim-X11
```

これでGUI版がインストールされます。GUI版を使わない場合は、vim-enhancedパッケージを利用できます。

ファイル構造

Vimのインストールが完了したら、最初にVimの各設定ファイルの置き場所を把握しておきましょう。

- **~/.vimrcまたは~/.vim/vimrc** …設定を書くためのファイルで、Vim scriptという言語で書くことができる(Windowsの場合は$HOME/_vimrcまたは$HOME/vimfiles/vimrc)
- **~/.gvimrcまたは~/.vim/gvimrc** …GUI版のみで読み込まれる設定ファイル(Windowsの場合は$HOME/_gvimrcまたは$HOME/vimfiles/gvimrc)
- **~/.vim** …プラグインなどのVim関連のファイルを置くためのディレクトリ(Windowsの場合は$HOME/vimfiles)

プラグインはVimの機能を拡張するためのスクリプトファイルで、プラグインの設定やVim本体の設定を.vimrcで行います。また、.gvimrcではGUI版のみで読み込みたい設定(たとえばツールバーの設定)を行います。

Vimチュートリアル「vimtutor」

Vimには「vimtutor」というチュートリアルが付属しています。このチュートリアルがとても良くできており、Vimの基本操作やVimの独特なしくみである**モード**に慣れることができます。まずはチュートリアルから始めてみましょう。

コマンドラインから次のように起動できます。

```
$ vimtutor
```

コマンドを入力するとVimが立ち上がり、そのVimの中でチュートリアルが開きます。KaoriYa VimではVimを起動した後、`:Tutorial`と入力して Enter キーを押すとチュートリアルを開始できます。

Vim内でチュートリアルを読みながら、直接チュートリアルの文章を編集してVimの各機能に触れてみましょう。h、j、k、lによるカーソル移動やVimの起動／終了といった最も基本的なことから順番に説明されていきます。

Vimはモードの存在など、ほかのエディタに比べて取っ付きにくいところがありますが、チュートリアルで丁寧に基本操作が説明されているため、ひととおりこなした後は基本的な操作で困ることはほとんどなくなるはずです。テキストの編集(削除、挿入、カーソル移動など)の重要な操作がたくさん含まれているので、たまに時間を見つけて操作が身につくまで繰り返すことをお勧めします。

最初から最後までかかった時間を記録しておいて、どれだけ早く正確に編集できるようになっ

たかを見てみるのもおもしろいかもしれません。

Vim設定の超基礎

ひととおりチュートリアルを終えたら、早速Vimを自分好みにカスタマイズしてみましょう。Vimはファイル構造の節で説明した.vimrc(Windowsなら_vimrc)に設定を書くことで、Vimの各種機能の設定を行ったり、拡張したりすることができます。この節ではVimに標準添付されているデフォルト設定ファイルvimrc_example.vimから、いくつか例を取り上げてVimを設定する方法の超基礎を紹介します。

vimrc_example.vimはVimのインストール先の中にあるランタイムディレクトリ(Vim実行時に読み込まれるファイル群が入ったディレクトリで、Ubuntuなら/usr/share/vim/vim74/)にあります。

コメントの書式

まず最初に、設定ファイル内のコメントは"から行末までです。設定の意味を忘れないように適宜コメントを書いていくことをお勧めします。

```
" 行末までコメント
```

オプションを設定する書式

基本的な各オプションの設定はsetを使って行えます。

```
set nocompatible
set backspace=indent,eol,start
set undofile
```

有効/無効で設定するオプションはset {オプション名}で有効に、set no{オプション名}で無効になります。よって、set nocompatibleはcompatibleを無効にするという意味になります。compatibleはVimとviの互換機能を有効にするかどうかのオプションです。

また、set undofileではundofileというオプションを有効にしています。undofileはVimを終了してもアンドゥ情報が保存され、次回同じファイルを開いたときにアンドゥを行える便利機能です。

文字列を指定するオプションの場合は、=で値を指定します。set backspace=indent,eol,startでは、backspaceオプションに"indent,eol,start"という値を設定しています。backspaceはバックスペースキーでの削除の挙動を決めるためのオプションです。それぞれindentは自動挿入されたインデントを超えて削除できるように、eolは改行を超えて削除できるように、startは挿入モード開始位置を超えて削除できるようになります。

キーマップの書式

次にキーマップのカスタマイズ方法について説明します。

```
map Q gq
inoremap <C-U> <C-G>u<C-U>
```

map Q gqはQというキー入力がされたときに、それをgqというキー入力にマッピングします。つまり、Qを入力するとgqを入力したのと同じ扱いになります。gqがさらに別のキー入力にマッピングされていた場合は、それがさらに適用されます。gqがほかにマッピングされていなければ、Vimのデフォルトの機能である"行の整形機能"が使われます。

なお、マッピングがどのモードに対して行われるかはマッピング用コマンドの頭1文字で決まっており、**表1**のようになっています。

次にinoremap <C-U> <C-G>u<C-U>について説明します。

まず、inoremapについてです。mapでなくnoremap(= no recursive mapping)を使うと再帰的なマッピングを行いません。再帰的なマッピングと言われてピンとこない場合は、試しに

Vimを開いて:nmap w wwと入力してみましょう。このあとwを押すと一気にファイル末尾までカーソルがジャンプしてしまったはずです。これはマッピングによってwがwwに置き換えられ、置き換えられたwwがさらにまた置き換えられてしまうためです。これを禁止するのがnoremapで、nnoremap w wwとすると2単語分しかカーソルが移動しなくなります。

また、Ctrlキーを含むキー入力は<C-文字(表記には大／小文字の区別なし)>と書きます。したがって、上記の例は"挿入モードでCtrl+uを入力した際に、Ctrl+g→u→Ctrl+uにマッピングする"という意味になります。再帰的なマッピングを行わないため、<C-G>u<C-U>はそれ以上別のマッピングが適用されず、Vimのデフォルト機能が実行され、undoの変更点を作った後undoを実行します。.vimrcでは基本的にこのnoremapを使ってマッピングすると事故が起きにくいのでお勧めです。なお、このほかにもEnterキーを表す<CR>、BackSpaceキーを表す<BS>などがあります。

マッピングについての詳細は:help key-mappingで見ることができます。

auto commandの書式

最後に、auto commandについて説明します。Vimには特定のタイミングで実行されるイベントに対して、任意のコールバックを指定できるauto commandというしくみがあります。

```
autocmd FileType text setlocal ➥
textwidth=78
```

この場合ではFileType textがイベント、setlocal textwidth=78がコールバックになります。具体的にはファイルタイプがtextになったとき(.txtなファイルを開いたときなど)に、textwidthを78に設定しています。textwidthは指定された文字数で自動的に行を折り返す機能です。また、setlocalは現在編集しているファイルに対するローカルな設定を行います。

以上、Vimに関する基本的な設定について紹介しました。これらはすべてVim scriptの機能の一部なので、本来は変数や制御構文といった言語機能も存在しますが、最初の時点では本体の簡単な設定方法を把握していれば十分です。

ヘルプの引き方

では早速設定していきましょう！と言いたいところですが、設定を書いているとわからないことが次から次へと出てくると思います。まずはその調べ方について知っておいたほうが良いでしょう。

Vimでわからないことがあったとき、Googleで検索するよりも前に試してみるべきことがあります。Vimにはしっかりメンテナンスされている膨大なドキュメントがあり、:helpコマンドによってVimの中からドキュメントを調べることができます。

ヘルプの基本的な使い方

Vim内でヘルプを閲覧するには:helpコマンドを使います。では、早速やってみましょう。Vimを立ち上げて、次のように入力してください。

```
:help help
```

:helpコマンドのヘルプが表示されたはずです。このように:help {検索ワード}でヘルプ

▼表1　マッピング用コマンドとモードの対応表

マッピングコマンド	マッピングするモード
map	ノーマル・ビジュアル・モーション待ち
imap	挿入
nmap	ノーマル
vmap	ビジュアル
cmap	コマンドライン
omap	モーション待ち

※モーション待ち……dやyなどのオペレータ(後述)直後の範囲指定用マッピング

を検索できます。

調べている中でわからないオプション名やコマンド名、マッピングなどはどんどんヘルプを引きましょう。ヘルプ中ではWebページのリンクのようにほかの用語へのリンクが貼られており、Ctrl-]でカーソル下のリンクにジャンプできます(Ctrl-tで元の場所に戻ってこられます)。

多くのプラグインのドキュメントもヘルプとして書かれており、:help {プラグイン名}で検索できます。プラグインの導入の仕方については次の節で説明していきます。

項目別ヘルプの引き方

前節で出てきたundofileという設定値について、もっと詳しく知りたいと思い、次のコマンドを実行したとしましょう。

```
:help undofile
```

しかし、実際にはundofile()という関数のヘルプが表示されてしまいます。これは、オプションundofileと同名の組み込み関数があり、:helpは最初に検索にヒットしたヘルプを表示するためです。このようなときどうすれば良いでしょうか。

ヘルプには一定の書式があり、それに従った検索ワードを用いることで、オプションやコマンド、キーマッピングなど対象を絞ることができます。**表2**に例をあげます。

また、キーマッピングについて、Ctrlキーを含んだマッピングはCTRL-{キー}で検索します。たとえば次は、Ctrl-oを検索したい場合です。

```
:help CTRL-o
```

この場合、“ノーマルモード”のCtrl-o(カーソル位置の履歴をたどる)が検索されます。

では、“挿入モード”のCtrl-o(一時的にノーマルモードのマッピングを使う機能)をどう検索すれば良いでしょうか?

マッピングのモードを指定するには、各モードのプレフィックス(**表3**)を使います。

```
:help i_CTRL-o
```

最後に、複数の文字入力でCtrlキーを含む場合は_を付けてつなげます。たとえば、挿入モードのCtrl-x Ctrl-o(オムニ補完実行[注3])は次のように検索します。

```
:help i_CTRL-x_CTRL-o
```

なお、この時点ではヘルプページは英語で表示されており、読むのに少し苦労したかもしれません。後ほどプラグイン導入の節で日本語ヘルプを導入します。

便利なまとめヘルプページ

残念ながらヘルプには逆引きはありませんが、情報がまとめられているヘルプがいくつかあります。

注3) カーソル位置の文脈にあわせた補完候補リストが表示される。

▼表2　ヘルプの書式例

検索したい種類	検索方法	検索例
オプション	''で囲む	:help 'undofile'
コマンド	:を頭につける	:help :write
組み込み関数	()を末尾につける	:help empty()

▼表3　各モードのプレフィックス

	ノーマル	挿入	ビジュアル	コマンドライン
プレフィックス	付けない	i	v	c

- `:help quickref` …クイックリファレンスマニュアル。チートシート的に操作の簡単な説明が載っている
- `:help usr_toc` …ユーザが順に読んでいくことを想定した、Vimのユーザマニュアルの目次
- `:help reference_toc`…Vimのリファレンスマニュアルの目次。各機能ごとにまとめられている
- `:help key-mapping`…マッピングのコマンドや書き方などのまとめ
- `:help function-list`…組み込み関数一覧

ほかの人の「.vimrc」を参考にする

ヘルプを順番に見て.vimrcをいちから書いていくのは大変です。代わりに、最初はほかの人が公開している.vimrcを参考にして、設定ファイルを書いていく方法がお勧めです。

GitやGitHubは設定ファイルを管理するのにも非常に便利なため、.vimrcはほかのさまざまなツールの設定ファイルと共に、GitHubで多く公開されています。慣例的に、dotfilesという名前のリポジトリを使っている人が多いようです。Vimを使っているエンジニアで気になる人がいれば、その人のdotfilesリポジトリを探してみるのも良いかもしれません。

また、Lingr(P.28のコラム参照)ではvimrc読書会という.vimrcを読むイベントが毎週開催されています。vimrc読書会で読む対象に選ばれたものを使ってみるのも良いかもしれません。一覧は次のURLにあります。

http://vim-jp.org/reading-vimrc/archive/index.html

Vimに詳しい人の設定を参考にするのは、自身のVimの設定を行ううえでかなり近道になります。

いろんなところから設定を持ってくる場合の注意点

ほかの人の.vimrcを参考にするのはとても有効な設定の書き方ですが、注意点もあります。

まず、よくわからないまま設定をコピー&ペーストして使うのは避けるべきです。わからないオプションやコマンド、マッピングなどが出てきた場合には、前章で説明した方法でヘルプを確認しましょう。設定の意味がわからないまま書いてしまうと、Vimの挙動がおかしくなった場合などにどの部分に問題があるか見当すらつかなくなってしまいます。

また、Vimを使い込んでいる人はVim scriptでかなり高度な設定をしている場合があります。しかし、最初からそういう設定を真似るよりは、基本的な部分(`set`によるオプション設定やマッピング、変数による各プラグイン設定など)から始めるほうが無難です。複雑な設定を書いてしまうと、今後メンテナンスしていくのが大変になってしまいます。

多くの人はプラグイン管理用のプラグインを使って、ほかのプラグインを管理しています。プラグイン管理プラグインの使い方を先に知っておくことで、ほかの人がどのようなプラグインを使っているかを知ることができます。プラグイン管理プラグインについては次の節で解説します。

設定ファイルをバージョン管理する

Vimの設定ファイルはバージョン管理システムで管理することをお勧めします。設定を書いていると、思わぬところでVimの挙動がおかしくなってしまったり、前の設定に戻したくなるときがあります。そのようなときに、今までの変更履歴を保存していると元の状態に戻したり、どの時点からVimの挙動がおかしくなったのかを二分探索したりすることができ、非常に便利です。

たとえば、Gitを使ってdotfilesというリポジトリで設定ファイルを管理するときは**図2**の

ようにします。

シンボリックリンクによって、dotfiles内で複数ある設定ファイルを管理しつつ、ホームディレクトリに設定ファイルを配置します。設定ファイル内に公開できない情報などが含まれていない場合は、このリポジトリをGitHubで管理しておくと、git cloneを使うだけでほかの環境にもお手軽に設定ファイルを持ってくることができます。

オススメのvimrc

最後に、GitHubで公開されている.vimrcのうち、設定に対するコメントが丁寧であるなど、筆者がお勧めするものを紹介します。かなり長いものもありますが、まずは基本的な設定を行っている個所(setによるオプション設定やマッピングなど)を読んで参考にさせてもらいましょう。

- https://github.com/cohama/.vim/blob/master/.vimrc
- https://github.com/rhysd/dotfiles/blob/master/vimrc
- https://github.com/deris/dotfiles/blob/master/.vimrc
- https://github.com/daisuzu/dotvim/blob/master/.vimrc

プラグイン管理プラグイン

Vimプラグインは本来は~/.vimディレクトリ(Windowsなら$HOME/vimfiles)以下に直接展開しますが、最近では"Vimプラグインを管理するためのVimプラグイン"を使って管理するのが一般的となっています。

プラグイン管理プラグインはpathogen.vim、Vundle.vim、vim-plug、neobundle.vimなど多数ありますが、ここでは筆者も利用しているneobundle.vimの使い方を説明していきます。neobundle.vimの詳しい使い方や最新の情報はインストール後にヘルプページ(:help neobundle)で参照することができます。

基本的な使い方

gitコマンドが必要なので、Xcode(Mac)やGit for Windows(Windows)、各種パッケージマネージャ(Linuxなど)でGitをインストールした後、**図3**のコマンドでneobundle.vimをローカルにダウンロードします。Windowsでは~/.vimを適宜$HOME/vimfilesに読み替えてください。

次に、Vimでneobundle.vimを利用するための最小限の設定を書きます。.vimrcの先頭に**リスト1**のように書いてみましょう。この時点でVimを起動してみてエラーなどが出なければ、次のステップに進みましょう。

▼図2　Gitでdotfilesリポジトリを管理する

```
$ mkdir ~/dotfiles
$ cd ~/dotfiles

... (.vimrcをdotfiles内に書く)

$ git init . && git add .vimrc && git commit -m "first commit"
$ ln -s .vimrc ~/
```

▼図3　neobundle.vimのダウンロード

```
$ mkdir -p ~/.vim/bundle
$ cd ~/.vim/bundle
$ git clone https://github.com/Shougo/neobundle.vim.git
```

インストールするプラグインの指定には:NeoBundleコマンドを使います。書き方はリスト2のようになります。

▼リスト1 neobundle.vimを使うための設定例(.vimrc)

```
if has('vim_starting')
    set nocompatible
    set runtimepath+=~/.vim/bundle/neobundle.vim
endif

call neobundle#begin(expand('~/.vim/bundle'))

" neobundle.vim自身をneobundle.vimで管理する
NeoBundleFetch 'Shougo/neobundle.vim'

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" ここにインストールしたいプラグインの設定を書く
"   :help  neobundle-examples
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

call neobundle#end()

filetype plugin indent on

" プラグインがインストールされているかチェック
NeoBundleCheck

if !has('vim_starting')
    " .vimrcを読み込み直した時のための設定
    call neobundle#call_hook('on_source')
endif
```

▼リスト2 インストールするプラグインの指定方法

```
❶ https://github.com/{user}/{repo} で公開されているプラグイン
NeoBundle '{user}/{repo}'

❷ http://www.vim.org/scripts/ で公開されているプラグイン名
NeoBundle '{プラグイン名}'

❸ プラグインのリポジトリをgit:// プロトコルで直接指定
NeoBundle 'git://...'
```

▼リスト3 ヘルプの日本語化プラグインのインストール指定(.vimrc)

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

▼図4 インストールの確認メッセージ

```
Not installed bundles:  ['vimdoc-ja']
Install bundles now?
(y)es, [N]o:
```

▼リスト4 ヘルプの日本語化プラグインの設定

```
set helplang=ja,en
```

ここで1つ具体例として、ヘルプページを日本語化するプラグインをインストールしてみましょう。Vimのヘルプは有志によって日本語に翻訳されており、次のリポジトリに置かれています。

https://github.com/vim-jp/vimdoc-ja

これはリスト2の❶の書式ですから、.vimrcに書いたneobundle.vimの設定(リスト1)のうち、call neobundle#end()の1つ前の行に、リスト3を追記してください。保存したら一度Vimを終了し、再度起動しようとすると、インストールするかどうかの確認メッセージが表示されます(図4)。

ここで「y」を選択すると、neobundle.vimが自動でGitHubから未インストールプラグインをダウンロードして配置してくれます。また、起動時の未インストールプラグインチェック時以外でも:NeoBundle Installコマンドを使ってインストールすることもできます。

プラグインのインストール自体はこれだけで完了です。次にプラグインの設定をします。.vimrc内にリスト4のように書いてみましょう。

これにより、日本語ヘルプのほうが英語ヘルプよりも優先して表示されます。試しに、:help 'undofile'などを試してみましょう。なお、明示的にヘルプの言語を指定したいときは、リスト4の末尾に@jaや@enを付けます。

このようにして、気になったプラグインを:NeoBundleコマンドを使って気軽に試してみたり、管理することができます。各プラグインのアッ

プデートは`:NeoBundleUpdate`、不要になったプラグインのアンインストールはプラグインの`:NeoBundle`の行を削除した後、`:NeoBundleClean {プラグイン名}`を実行します。

初心者向けお勧めプラグイン

プラグインはGitHubやvim.orgなどで公開されています。最近では、VimAwesome[注4]という、GitHubに公開されている設定ファイルから使われているプラグインを抽出して集計するWebサービスもあります。この節では、初心者にお勧めだと思うプラグインを3つほど簡単にご紹介します。

vim-quickrun

https://github.com/thinca/vim-quickrun

現在開いているファイルをVim内で直接実行し、結果を表示できるプラグインです。ちょっとした動作の確認やプログラミング勉強中のコード片の実行などを非常に手軽に行えます。このタイプのプラグインはいくつかありますが、拡張性の高さや対応言語の多さからvim-quickrunを一番お勧めします。

seoul256.vim

https://github.com/junegunn/seoul256.vim

柔らかい色合いで視認性の良いカラースキームです。カラースキームはVim全体の色合いを決めるもので、人によってかなり好みが出ますが、目が疲れにくいと感じるものを選ぶのがベストです。ぜひお好みのものを見つけてください。

unite.vim

https://github.com/Shougo/unite.vim

Vim内で「候補のリストを表示して絞り込み選択して実行する」という一連のインターフェースを提供するプラグインです。たとえば、“カレントディレクトリのファイル一覧から目的のファイルを絞り込み検索して開く”といった操作ができます。もちろんファイルだけでなく、さまざまなリストを対象にすることができます。どのようなものがあるかはunite.vimのヘルプを参照してください。

オペレータとテキストオブジェクト

最後に、少し応用的な機能の紹介をします。

Vimには「ある範囲に対して特定の操作をする」という一連の処理を行えるしくみがあります。このしくみの基本的な部分だけでも知っておくと、編集がとても楽になります。このしくみは**どの範囲に対する操作か**を示すテキストオブジェクトと、**その範囲に対して何をするか**を示すオペレータに分かれており、それぞれマッピングで指定します。

たとえば、ノーマルモードで単語の上にカーソルを移動させてから`diw`と入力してみてください。カーソル下の単語が削除されたはずです。これは、オペレータ`d`とテキストオブジェクト`iw`から成っています。`d`は「対象範囲を削除する」というオペレータで、`iw`は「カーソル下の単語」を範囲とするテキストオブジェクトです。

次に、選択ではなくカーソル下の単語をコピーしたい場合はどうすれば良いでしょうか？　範囲は変えず、オペレータを「対象範囲をコピーする」というオペレータである`y`に変更し、`yiw`とすれば可能です。

このように、オペレータとテキストオブジェクトを自由に組み合わせることでさまざまな編集を行うことができます。さらに、一連の操作は`.`で繰り返すことができます。なお、テキストオブジェクトには`i`で始まるものと`a`で始まるものがあり、対象の「内部」を選択するか、「全体」を選択するかの違いがあります。`i`は英語の“inner”の略、`a`は英語の“a”と覚えるとわかりやすいでしょう。たとえば、`iw`は“inner word”、`aw`は“a word”の略です。

注4）http://vimawesome.com/

表4、5にそのほかの代表的なオペレータとテキストオブジェクトを示します。

プログラムの文字列中でdi"や、C言語のifブロック内でyi{など、いろいろ試してみてください。慣れてくると、選択したい範囲に対してほとんど考えることなく適切なテキストオブジェクトを選べるようになり、リファクタリングなどの編集作業がとても速くなります。

オペレータやテキストオブジェクトの種類は多数あるため、詳細は:help operatorや:help text-objectsを参考にしてください。

▼表4　代表的なオペレータ例

オペレータ	操作の説明
c	対象を削除した後、挿入モードに入る(change)
y	対象をyank(コピー)する
>	対象のインデントを深くする
<	対象のインデントを浅くする

▼表5　代表的なテキストオブジェクト例

テキストオブジェクト	範囲の説明
i" または a"	"..." で囲まれた内部または全体
i(または a(	(...) で囲まれた内部または全体
i{ または a{	{...} で囲まれた内部または全体
it または at	HTML などのタグの内部または全体
ip または ap	段落の内部または全体

まとめ

Vimの導入方法について基本的なことを駆け足で紹介しました。最初はモードに慣れず戸惑うかもしれませんが、普段のコーディングにVimを使って慣れてくると、どんどん効率的な編集ができるようになっていくことが体感できるはずです。この入門記事があなたのコーディングの生産性を上げるきっかけになればと思います。SD

コラム　Vimコミュニティについて

● Lingr
http://lingr.com/signup?letmein=vim

Vimを使っていたり設定しているときにhelpで調べてもGoogleで検索してもわからないことが出てきた場合、よく知っている人に聞くのが一番の近道です。そういった疑問や問題を気軽に相談／共有する場として、Lingrというチャットサービスを利用できます。Lingrにはvim-jpというチャットルームがあり、日々Vimに関する雑談や相談などで賑わっています。また、vimrc読書会などのオンラインイベントも開催されています。

ここにはVimプラグイン開発者などといったVimを使いこなしている人たちがいて、質問すると丁寧に教えてくれます。ぜひ気軽に質問してみてください。筆者もLindanという名前でたまにログインしています。

● vim-jp
https://github.com/vim-jp/issues

GitHub上で日本のVimコミュニティが運営するvim-jpというorganizationがあります。Vim本体の挙動でバグを見つけたり、機能の要望がある場合はvim-jpが管理しているissuesというリポジトリが利用できます。

上記のリポジトリのissueにバグや要望を登録すると、Vimに長年コントリビュートしている一流のVimmer達からアドバイスがもらえたり、バグを修正するパッチを書いてもらえたりします。ここで出された成果はVim本体に還元されるため、Vim本体の改善にもつながります。

● 各地もくもく会

日本各地で、Vimユーザが集まって交流したり、(おもにVimで)作業をしたりする「もくもく会」が開催されています。Sapporo.vim、Aizu.vim、TokyoVim、momonga.vim、nagoya.vim、Osaka.vimなどが不定期で開催されていますので、お近くにお住まいの方は参加してみてください。Vimmer達にアドバイスをもらったり相談したりできる良い機会になると思います。

コラム Vimの正規表現

vimtutorの中で`/{文字列}`を使って検索する例が出ていたと思います。これだけでも便利なのですが、ファイルを編集していると単純な文字列検索では不十分なことがあります。たとえば、「num」という変数を検索したいが、「number」という変数には検索がヒットしてほしくない場合などが典型例でしょうか。`/`による検索では、ほかの多くのテキストエディタが実装しているように、正規表現によるパターン検索を行うことができます。

```
/\w\+          ←任意の1単語を検索
/^\s*$         ←空行を検索
/\(foo\|bar\)  ←'foo'か'bar'のどちらかを検索
```

正規表現の詳細については`:help regex`で参照できます。一般的な正規表現について説明しようとするととても長くなってしまうため、ここでは一般的な正規表現については紹介していません。正規表現ってそもそも何？という方は、まずは一般的な正規表現について調べてみると良いでしょう。

● Vimの正規表現の注意点

非常に便利な正規表現ですが、Vimの正規表現はRubyなどの正規表現と比べてパターンの書き方が異なっているものがあります。たとえば、Rubyで1回以上の繰り返しを表すのは`+`ですが、Vimの正規表現では`\+`となります。こういった違いはVimで正規表現を初めて使うときに戸惑う原因になりますので、よく使うものについて**表A**にまとめてみました。

表に載っているもののほかにも“否定／肯定先読み”や“肯定／肯定後読み”などがあります。

なお、PerlやRubyの正規表現を利用できるようになるeregex.vim※というVimプラグインもあるので、本文で説明しているプラグインのインストール方法を参考にしながらインストールしてみるのも良いかもしれません。

※https://github.com/othree/eregex.vim

● Vimの正規表現で便利なパターン

まずは単語の始まりにマッチする`\<`と単語の終わりにマッチする`\>`です。たとえば、前述の「num」にはマッチさせたいけれど「number」にはマッチさせたくない場合などは、`/num\>`と検索します。こうすると、numのすぐ後ろで単語が終了していなければならないため、numberにはマッチしなくなります。検索対象を絞るのに役立ちます。

次に、マッチを開始する位置を指定する`\zs`と`\ze`を紹介します。たとえば、「foo_bar」という変数名の内で、「bar」だけにマッチする正規表現を書きたいとします。このとき、Rubyなどでは肯定後読みを用いて`(?<=foo_)bar`などとすると思いますが、Vimでは`foo_`の後の`bar`からマッチが始まるという意味で、もっとシンプルに`foo_\zsbar`というふうに書けます。

これは、たとえば変数の一部を書き換えたいときに非常に便利です。`.`による繰り返しを利用して1ヵ所ずつ置換する方法もありますが、ここでは置換用のコマンド`:substitute`の省略形`:s`を使います。ファイル内を一斉置換するためのコマンドの書式は`:%s/{置換候補パターン}/{置き換え文字列}/g`です。`:%s/foo_\zsbar/hoge/g`と入力すると、「foo_bar」の「bar」を「hoge」に置き換えます。このほかにも、非空白行頭にマッチしたい場合は`^\s*\zs`で書けるなど、`\zs`や`\ze`はマッチさせたい個所の前後のパターンを指定したいときにいろいろ役立ちます。ぜひ試してみてください。

▼表A　Vim特有の正規表現例

Ruby	機能	Vim
+	1回以上の繰り返し	\+
?	0 or 1回のマッチ	\? または \=
{n}	n回の繰り返し	\{n}
\b	語境界	\<(単語始まり)、\>(単語終わり)
(...)	グループ化	\(...\)
(?:...)	キャプチャなしグループ化	\%(...\)

コラム1

「とっつきにくい変態エディタ」だったVimが「私の素敵な相棒」に変わるまで

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

なんでVimを使おうと思ったの？

今はRuby on Railsの開発がメインになっている私ですが、前職、前々職はそれぞれ、.NETやJavaの開発がメインでした。当時はVisual StudioやEclipseをほぼデフォルトの状態で使っており、特殊なキーバインドはまったく使っていませんでした。IDE（統合開発環境）を使わないときはサクラエディタをメインのテキストエディタとして使っていました。

そんな私がVimを使おうと思った理由は2点あります。

1つめの理由はLinuxサーバにログインすると、viぐらいしかデフォルトで入っているテキストエディタがなかったためです。自分の業務上、Linuxサーバを使って作業する機会はそれほど多くありませんでした。とはいえ、「viが使えないために簡単な設定ファイルの編集をするだけで右往左往してしまうのはちょっと恥ずかしい」という気持ちがありました。

2つめの理由は「Vimって便利ですよ！」と言いながらVimを使いこなす同僚のエンジニアがいたからです。どこからどう見ても扱いづらい変態エディタであるVimを「便利！」と絶賛する同僚が私は不思議でなりませんでした。しかし、同僚の言葉だけでなく、ネット上でも「Vimはすごい」「Vim最高！」といった声をよく目にしました。なので、「一度だまされたつもりでVimを使ってみよう。同僚の言葉が正しいのか、それともやはりただの変態エディタなのかはじっくり使い込んでから結論を出そう」と考えました。

このような理由から私はVimを本格的に使い始める決心をしました。

どんなふうにVimを始めたの？

前述のとおり、Vimは私にとって本当にとっつきにくい変態エディタでした。ですから、「Vimを使い始める」イコール「一時的に生産性が落ちる」ということを覚悟しました。また、「いつものキーバインドに戻れる」という逃げ道を作ってしまうと、なかなかVimの学習曲線が上昇しません。そこで、「テキストを編集するときは絶対にVimしか使わないぞ!!」という「Vim縛り」を自分に課すことにしました。

しかし、それだけでもまだ不十分です。Vimはインサートモードに入るとほとんど「メモ帳」のように使えてしまいます。それに頼ってしまうとVimを使う利点がなくなってしまうので、チートシートを手元に用意し（**図1**）、できるだけノーマルモードとVimコマンドの組み合わせでカーソルを移動させたり、テキストを編集したりすることを心がけました。

こういった練習を毎日繰り返すと、だんだんとVimが手に馴染んできます。2～3日もすれば「あ、ちょっと慣れてきたかも」という実感がわいてきました。その感覚がつかめたら最初の大きな壁は越えたことになります。そこからはどんどん学習曲線が上がり、Vimを使うことが楽しくなってきました。

で、Vimに対する結論はどうなったの？

はい、Vimはやっぱり便利でした!!

たとえば、Vimを使えばマウスや十字キーを使わなくてもカーソルをびゅんびゅんと好きなように移動できます(例：w、e、bやH、M、L)。「この行やこの単語を削除したい」と思ったときもコマンド一発です(例：dd、dw)。ほかにもカーソルの下にある数字を増減させる(Ctrl-a、Ctrl-x)など、「メモ帳系エディタ」にはマネできない操作を実現できる点もVimの大きな強みです。

Vimには数多くのコマンドがあり、確かに覚えるのにはちょっと時間がかかりますが、一度覚えてしまえば「頭の中にあるやりたいこと」をとてもすばやく実現できるようになります。

まとめ

というわけで、このコラムでは私がVimを使おうと思ったきっかけと、Vimを使い始めたころの練習方法、それにVimが使えるようになってから感じているVimの利点について、経験をふまえて書いてみました。数年前の私と同じように、「Vimを使いこなしたいがどうすればいいかわからない」という方の参考になると幸いです。

なお、Vimの練習方法やVimの便利なコマンドについては、私のブログやQiita記事に詳しく書いていますので、よかったらそちらも参考にしてみてください。SD

- **僕がサクラエディタからVimに乗り換えるまで**
  http://blog.jnito.com/entry/20120101/1325420213
- **脱初心者を目指すなら知っておきたい便利なVimコマンド25選(Vimmerレベル診断付き)**
  http://qiita.com/jnchito/items/57ffda5712636a9a1e62

▼図1　私のチートシート

第2章 IDE並みの機能を軽快な動作で! 実用Tips&対策［プログラマ編］

Writer mattn Twitter @mattn_jp

Vimにも興味はある、けれどもIDEの便利さはなかなか手放せない。あるいは逆に、Vimを使っているけれどもIDEへのあこがれもある。そんなプログラマの方に向け、本章ではIDEの良さをVimに取り込む、プログラマ向けカスタマイズのポイントについて解説します。

IDEとテキストエディタのいいとこ取りをしたい!

職業であっても趣味であっても、プログラミングをしている人であれば必ずお世話になるのがテキストエディタです。IDE(統合開発環境)を使って体系的にプログラミングを学ぶのも楽しいですが、IDEの多くは起動が遅く、イライラするうちにせっかく思いついたアイデアを失ってしまう場合もあります。いったんテキストエディタでプログラミングする快適さを覚えてしまった人達にとって、テキストエディタは切っても切れない存在になってしまうのです。

IDEが不便だと言っているのではありません。IDEの使い心地の良い機能を一度味わってしまうと、テキストエディタの非力さを感じてしまうことも多々あります。そしてどうしても同じ使い勝手を使い慣れたテキストエディタに求めてしまうのです。Vimmer(Vim使い)も例にもれず、IDEの操作感を模倣したvimrc(vimの設定ファイル)やプラグインをよく見かけます。

一般的なIDEの主な機能は次のようなものだと思います。

・入力補完
・GUIの作成
・コンパイラとの協調動作
・デバッガ
・プロジェクトファイルの管理

ネイティブなGUIの作成はさすがにIDEには敵わないのですが、Webアプリケーションのようにhtmlを書くのであればVimの強力な編集能力はIDEに勝ることもあるのです。

Git対応の強化

最近はGitでソースが管理されることも多くあります。Vimでも、リポジトリ内のファイルをプロジェクト管理されたファイルと見立ててファイル検索する便利なプラグインがあります。Unite[注1](**図1**)、CtrlP[注2](**図2**)などが有名です。

プロジェクトのルートディレクトリを検出する方法も指定できるため、リポジトリ内に複数のプロジェクトが存在し、特定のディレクトリ配下をプロジェクトと見立てて使う場合にも使用できます。これらのプラグインはバッファ／ファイルの検索だけでなく、最近使用したファイルやタグ一覧などの絞り込み検索もできます。

UniteやCtrlPはプログラマブルなインターフェースを提供しており、いろいろな拡張をユーザ側で作ることができます。たとえば、ローカ

注1) URL https://github.com/Shougo/unite.vim
注2) URL https://github.com/kien/ctrlp.vim
CtrlPは現在、次のリポジトリでforkという形で開発が進められています。
URL https://github.com/ctrlpvim/ctrlp.vim

ルでのGitHubのリポジトリ管理を楽にするghqをVimから簡単に扱えるようになるUnite/CtrlP用のプラグインもあります[注3]。

ソースファイル検索

Vimには標準で:grepコマンドや:vimgrepコマンドが用意されていますが、ag(the_silver_searcher)と連携するag.vim[注4]を使うことで高速にファイル検索を行うことができます。

agはスレッド、mmap、正規表現のJITなどを使って高速にファイルを検索できるプログラムです。Windowsでも動作します。検索結果はQuickfixウィンドウに表示されるので、一覧から選択とジャンプが可能です(図3)。

また、Vimの:grepコマンドは実行に外部プログラムのgrepを使用しますが、grepコマンドはeuc-jpやshift_jis、utf-8など、それぞれ別々のエンコーディングで書かれたファイルを扱うことはできません。そういった場合、jvgrep[注5]コマンドを使うことで解決できます。これもWindowsでも動作します。

タグジャンプ

Vimを使ってソースコードを編集する場合に役立つのがタグジャンプです。Vimプラグイン

注3) URL https://github.com/sorah/unite-ghq
URL https://github.com/mattn/ctrlp-ghq

注4) URL https://github.com/rking/ag.vim

注5) URL https://github.com/mattn/jvgrep

▼図3 agによる検索結果の一覧表示

```
    static int
mch_system(char *cmd, int options)
{
    DWORD       ret = 0;
    UINT        wShowWindow;
    UINT        h_module;
    MSG         msg;
os_win16.c                                                    224,1
os_win16.c|224 col 1| mch_system(char *cmd, int options)
os_win16.c|308 col 6| x = mch_system(p_sh, options);
os_win16.c|334 col 7| x = mch_system((char *)newcmd, options);
os_win32.c|4028 col 1| mch_system_classic(char *cmd, int options)
os_win32.c|4320 col 1| mch_system_piped(char *cmd, int options)
os_win32.c|4606 col 1| mch_system(char *cmd, int options)
os_win32.c|4610 col 9| return mch_system_piped(cmd, options);
os_win32.c|4612 col 9| return mch_system_classic(cmd, options);
os_win32.c|4618 col 1| mch_system(char *cmd, int options)
os_win32.c|4633 col 11| #  define mch_system(c, o) system(c)
[Quickfix List] :ag --column mch_system                       1,1
```

▼図1 Unite

```
> mby
  runtime/doc/mbyte.txt
  src/testdir/mbyte.vim
  src/mbyte.c
  src/proto/mbyte.pro
  src/objects/mbyte.o
~
~
~
```

▼図2 CtrlP

```
> runtime/indent/ruby.vim
> pixmaps/tb_copy.xpm
> runtime/compiler/ruby.vim
> runtime/doc/mbyte.txt
> src/proto/mbyte.pro
> src/testdir/mbyte.vim
> src/mbyte.c
 prt  path  <mru>={ files }=<buf> <->                  /home/mattn/dev/vim
>>> mby_
```

に頼っているユーザの中にはタグジャンプの使い方を知らないままの人もいますが、言う人に言わせれば「タグジャンプあってのVim」と言っても良いというほどにVimとtagsファイルは切っても切れない存在となっています。

tagsファイルを生成するctags[注6]は40を越えるプログラミング言語をサポートしています[注7]。この40を越えるプログラミング言語のタグジャンプを、Vimを使えば同じ操作感で扱えるのです。

一般的な使い方はソースツリーのルートディレクトリで次のコマンドを実行します。

```
$ ctags -R
```

これでtagsファイルが生成されます。Vimはtagsオプションで指定したパスリストにtagsファイルが存在すると、そのファイルを読み込んでタグジャンプを行います。

```
:set tags=./tags,tags,../tags
```

この例ではカレントディレクトリおよび親ディレクトリにあるtagsファイルが検索され、参照されます。また、

```
:set tags=/lib/**/tags
```

このように**を用いることで階層をくだって検索したり、

```
:set tags=/lib/**2/tags
```

このように数値を付与することで2階層限定の検索を行うこともできます。

タグジャンプはジャンプしたいシンボル上でCtrl-]をタイプします。ジャンプリストを戻るにはCtrl-tをタイプします。タグが複数存在するときは:tag fooのようにコマンドをタイプするか、g Ctrl-]をタイプしてタグを選択できます。

前述したUniteとCtrlPにも、このタグを扱える拡張が存在します。

注6）Exuberant Ctags。ソースやヘッダ内にある名前のタグ（インデックス）ファイルを生成するプログラム。

注7）URL http://ctags.sourceforge.net/languages.html

ファイルブラウザ

Vim標準でnetrwというファイルブラウザが搭載されています。これを画面左側に表示することでIDEのファイルブラウザが実現できます。

リスト1の例は画面左側にツリー状のファイルブラウザを開き、その右側でファイルを開くという設定を<leader>e（何も設定していない状態であれば\e）というキーに割り当てています。

▼リスト1　netrwを開きIDE風に配置する設定

```
let g:netrw_liststyle = 3
let g:netrw_browse_split = 4
let g:netrw_altv = 1

function! ToggleVExplorer()
  if !exists("t:netrw_bufnr")
    exec '1wincmd w'
    25Vexplore
    let t:netrw_bufnr = bufnr("%")
    return
  endif
  let win = bufwinnr(t:netrw_bufnr)
  if win != -1
    let cur = winnr()
    exe win . 'wincmd w'
    close
    exe cur . 'wincmd w'
  endif
  unlet t:netrw_bufnr
endfunction
map <silent> <leader>e :call ➥
ToggleVExplorer()<cr><c-w>p
```

▼リスト2　NERDTreeを使ったIDE風配置設定

```
map <silent> <leader>e :NERDTreeToggle<cr>
```

▼リスト3　シンボルブラウザ（taglist）を右に表示する設定

```
let Tlist_Show_One_File = 1
let Tlist_Use_Right_Window = 1
let Tlist_Exit_OnlyWindow = 1
map <silent> <leader>E :TlistToggle<cr>
```

また、NERDTreeというファイルブラウザを使う場合は**リスト2**のように設定します。

NERDTreeはnetrwよりもより高機能でカスタマイズ性が高く、キャッシュを使用することで高速化が図られています。

シンボルブラウザ

前述のタグジャンプをIDEのようにUIから行いたいのであれば、taglist.vim[注8]とtagbar[注9]の2つがお勧めです。

ファイルブラウザは**リスト1または2**で左側に表示したので、シンボルブラウザはエディタ画面の右側に表示してみます(**リスト3**)。キーは`<leader>E`に割り当てています。これでIDEのような見栄えになりました(**図4**)。

あとはCtrlPやUniteなどでファイルを開く際に、中央のウィンドウで開かれるように、**リスト4**の設定を追加します。ほぼIDEと同じ見栄えになったのではないでしょうか。

ビルド

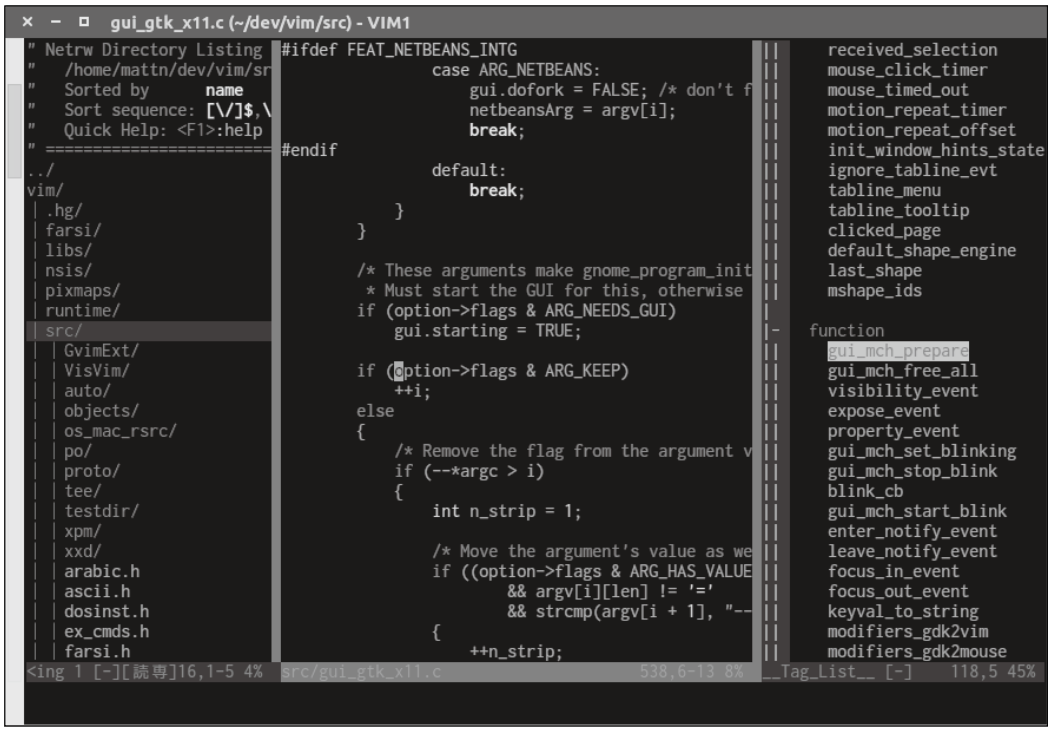
`:make`を実行すると`makeprog`というオプショ

注8) URL https://github.com/vim-scripts/taglist.vim
注9) URL https://github.com/majutsushi/tagbar

▼リスト4　CtrlPでファイルを中央のウィンドウで開く設定

```
function! CtrlP_OpenAtCenter(action, line)
  let cw = bufwinnr('.')
  for n in range(0, bufnr('$'))
    let bw = bufwinnr(n)
    if bw == cw && buflisted(n)
      exe bw . 'wincmd w'
      break
    endif
  endfor
  call call('ctrlp#acceptfile', [a:action, a:line])
endfunction
let g:ctrlp_open_func = {'files': 'CtrlP_OpenAtCenter'}
```

▼図4　ファイルブラウザを左に、シンボルブラウザを右に表示させたVim



ンで指定されたコマンドが実行されます。Vimにはこのmakeprogを言語ごとに切り替えられる:compilerコマンドが存在します。標準で59個ものcompilerプラグインが含まれており、コンパイルだけではなくスクリプト言語の構文チェックを実行することもできます。結果はQuickfixに出力されるため、エラーの一覧から直したい個所を選んでジャンプできます。

なお、ファイルの書き込みと同時に:makeを実行するflymake[注10]というプラグインもあります。

また、フルスクラッチからプログラミングする際には、現在のバッファを簡単に実行できるプラグインquickrun[注11]が便利です。

コマンドウィンドウ

IDEの中には小ウィンドウ内でコマンドラインシェルを実行できるものがありますが、Vimならtmuxを使うことで同様の機能を得ることができます。Vim本体の機能ではありませんが、UNIXでは個別のプログラムを組み合わせることで目的の機能を作り上げるのが一般的です。

また、vim-dispatch[注12]というプラグインを使うことで、Vimでシームレスなコマンドウィンドウを実現できます。

vim-dispatchをインストールすると、:Dispatchや:Make、:Startといったコマンドが使えるようになります。外部コマンド名を付与して実行することで非同期にコマンドが実行され

注10) URL https://github.com/kana/vim-flymake
注11) URL https://github.com/thinca/vim-quickrun
注12) URL https://github.com/tpope/vim-dispatch

コラム あなたは常駐派？　それとも毎回起動派？

誰しもがVimをテキストエディタだと思ってはいますが、昨今のVimの進化によりVimを環境として使おうとする人が増えてきました。作業中はずっとVimを起動したままにしておき、ファイル操作やコマンド実行などのすべてをVimから行っている人もいるようです。さらにscreenやtmuxを使ってサーバ上でVimを何日も起動したままにしておくという人も見かけます。

こういった使い方はどちらかというとEmacs使いの人達に多く見られたのですが、昨今のこういった状況を見ると、Vimもだんだんemacs化してきたなぁと感慨深くなります。彼らはVimをめったに終了させません。代わりにVimプラグインをとにかくたくさんインストールします。Vimの起動に数秒かかるという人もいるようです。

それに比べて筆者はどちらかというと、shellとvimを行き来する使い方をします。:wqでVimを終了してshellを操作したり、Vimから:shellを実行してシェルを操作したりします。Vimが1秒以内に起動しないとイライラします。そのためにもプラグインはできるだけインストールせず、起動が遅くなってきたら使っていないプラグインを消したり、vimrcから必要ない設定をどんどん消していきます。

どちらかというと古い世代なのかもしれませんね。Vimを起動したままにするという使い方は筆者が知る限りなかったように思います。

こういった新しい使い方をする人達によって、未だ見ぬバグが発見される事例が最近いくつか出てきています。我々では気付かないバグです。いろんな人達によっていろんな使われ方をすることで、バグが発見されたり新しい要望が生まれたりします。とても良いことだと思います。

Vimはまだまだ変われるな、そう思ったりもします。バグを見つけた方はぜひ次のURLからissueを登録してください。私達vim-jpのメンバがお待ちしております。

https://github.com/vim-jp/issues/issues

ます。:Dispatchコマンドでは出力結果を分割ウィンドウに表示することもできます[注13]。たとえば、実行時間の長いmakeコマンドを実行中にVimでほかの作業をするといった用途に使えます。

VimShellやConqueのようにVim上でインタラクティブシェルを実行できるプラグインがいくつかありますが、vim-dispatchはインタラクティブ性よりもリアルタイム性が優先されています。インタラクティブシェルを模擬するプラグインはVimの内部タイマ(updatetime)を短くすることで実現しているため、あまりシームレスとは言えません。

vim-dispatchはtmuxのほか、iTerm、screen、Windows上でのremote API、X11のWindow Manager Controlなどを扱うことができ、各プラットフォームで同じ動作となるように実装されています。makeだけでなくgrep、デプロイスクリプトの実行など、編集のかたわら別のコマンドが実行できるのはとても便利です。

入力補完

最近はVimの入力補完もどんどん便利になってきており、IDEに引けを取らない入力補完が実現できています。Vimは標準でも次にあげる補完をサポートしています。

- 行補完
- キーワード補完
- 辞書補完
- シソーラス補完
- 参照ファイルを含むキーワード補完
- タグ補完
- ファイル名補完
- 定義補完
- マクロ補完
- Vimのコマンドライン補完
- スペル補完

Vimではプラグラマブルな機能を実装しており、次の補完拡張機能を用いることで未対応のプログラミング言語の入力補完機能を作成することができます。

- ユーザ定義補完
- オムニ補完

なお、標準では上記で紹介した補完機能は個別のキーアサインとなりますが、neocomplete[注14]

注13) Vimがtmux上で実行されている場合に限ります。

注14) URL https://github.com/Shougo/neocomplete.vim

▼図5 clang_complete

```
#include <iostream>
#include <string>
#include <algorithm>

int
main(int argc, char* argv[]) {
  std::string s;
  s.r
  r replace f std::basic_string<char> & replace(iterator __i1, iterator __i2, c
}   replace f std::basic_string<char> & replace(iterator __i1, iterator __i2, c
~   replace f std::basic_string<char> & replace(iterator __i1, iterator __i2, i
~   replace f std::basic_string<char> & replace(iterator __i1, iterator __i2, c
~   replace f std::basic_string<char> & replace(iterator __i1, iterator __i2, i
~   reserve f void reserve()
~   rend    f reverse_iterator rend()
~   rbegin  f reverse_iterator rbegin()
~   resize  f void resize(size_type __n, char __c)
~   resize  f void resize(size_type __n)
~
```

というプラグインを入れることで統合的な操作感を得ることができ、標準のキーアサインを用いない自動補完を行うことができます。

C言語系プログラマ向け

入力補完プラグインを入れることでC/C++のコーディングがかなり捗ります。

clang_complete

https://github.com/Rip-Rip/clang_complete

C/C++コンパイラであるclangのライブラリlibclangをバックエンドとして使用しています。構造体のメンバを補完したり、ネームスペース内の関数を補完したりできます(**図5**)。

YouCompleteMe

https://github.com/Valloric/YouCompleteMe

同じくlibclangを使い、バックグラウンドデーモンと通信することで高速な入力補完を行います。海外で絶大な人気のあるプラグインです。

marching.vim

https://github.com/osyo-manga/vim-marching

バックエンドとしてlibclangだけでなく、Wandboxという Webサービスを利用することができます。上記2つのプラグインは補完動作に入るとVimがブロックしてしまいますが、marching.vimは外部プロセスに補完候補の抽出を依頼した後は非同期に結果を待つことができるため、補完を始めた直後でも文字のタイプを邪魔されることはありません。

Java言語系プログラマ向け

javacomplete[注15]を使うことで補完が行えます。javacompleteは内部でclassファイルをコンパイルし、リフレクションを使って候補を作り出しています。若干、構文解析が弱いため、補完が得られるシーンに限りがあります。

Web系プログラマ向け

HTMLのコーディングであればemmet-vim(旧zencoding-vim)[注16]が便利です。CSSセレクタに似た構文からHTMLを一気に生成できます。たとえば`html:5>div#content>ul>li*3>a{お知らせ$}`といった式を展開すると**リスト5**のHTMLが得られます。

そのほか、haml、slim[注17]でもHTMLと同じ記法で展開できたり、css、less、scss、sass内での展開入力もサポートしています。

注15) URL http://www.vim.org/scripts/script.php?script_id=1785
注16) URL https://github.com/mattn/emmet-vim
注17) HTMLのテンプレートエンジン。

▼リスト5　emmet-vimで生成されたHTMLの例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <div id="content">
    <ul>
      <li><a href="">お知らせ1</a></li>
      <li><a href="">お知らせ2</a></li>
      <li><a href="">お知らせ3</a></li>
    </ul>
  </div>
</body>
</html>
```

▼表1　言語別入力補完プラグイン

言語	プラグイン
C/C++	clang_complete、marching.vim、YouCompleteMe
Python	jedi-vim
Perl	perlomni.vim
Ruby	RSense
golang	gocode
Java	javacomplete
JavaScript	tern.vim
Clojure	neoclojure
C#	OmniSharp
HTML	emmet-vim

言語別入力補完プラグイン

各言語の代表的な入力補完プラグインを**表1**にまとめました。

特筆したいのがjedi-vim[注18]です。Pythonの入力補完はjedi.vimの一人勝ちと言ってよいでしょう。内部でPython拡張(if_python)を使い、Python自身の構文解析を行うことで補完を実現しています。ファイルを開く関数openの戻り値を.で補完するとfileオブジェクトのメンバが補完候補に現れますが、openを実行しているわけでもないのに動的言語の型推論がきちんと行えています。

なお、Vimでcssを編集する際は、CSS3に対応したシンタックスプラグインを入れておくべきです[注19]。

注18) URL https://github.com/davidhalter/jedi-vim
注19) URL https://github.com/hail2u/vim-css3-syntax

▼表2　Vimライクな操作感を実現するIDE／エディタ拡張プラグイン

IDE/エディタ	拡張
Emacs	Evil
Atom	vim-mode
Eclipse	Vrapper
Visual Studio	VsVim
IntelliJ IDEA	IdeaVi

IDEをVimライクに

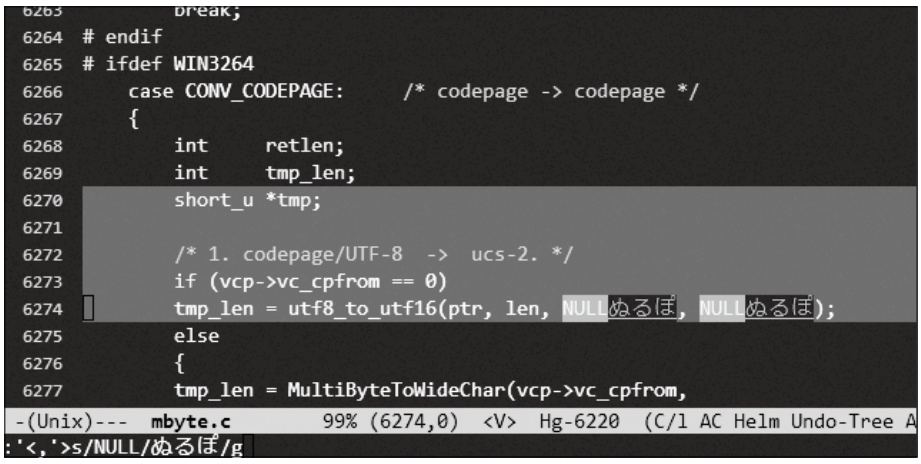
使い慣れたIDEを逆にVimライクにする方法も存在します(**表2**)。拡張可能なIDEやテキストエディタではVim風のキーバインドを提供しているものもあります。

多くの拡張はどちらかというと「なんちゃってVimモード」と言わざるを得ませんが、EmacsのEvilに関してはEvil上の拡張も存在し、vim-railsをポーティングしたevil-railsや、vim-surroundをポーティングしたevil-surroundなど、多くのVimプラグインがEvil上にポーティングされています(**図6**)。

まとめ

IDEにあこがれを抱きつつ、それでもやっぱりVimが好きで使い続ける、そんなあなたも一手間入れるだけでいつも使っていたVimをIDEのように変身させることができるのです。Vimは高度な拡張ができるため、プラグインを導入したり設定を行うことで自分だけのIDEを作り上げることができます。ぜひやってみてください。SD

▼図6　Evilを使ったVimライクなEmacs



第3章 運用作業であわてないために 実用Tips&対策 [インフラエンジニア編]

Writer LINE㈱ 佐野 裕(さの ゆたか) Twitter @sanonosa

日頃サーバ運用を行う場面では、必要に迫られて直ちにエディタを開いて操作するという場面が多いものです。vi系エディタはたいてい、どのUNIX系OSでも必ず標準でインストールされていて、動作が軽く消費ハードウェアリソース量が少ないので、インフラエンジニアの業務特性に合っていると言えます。本章では、著者が考えるインフラエンジニアのVimとの付き合い方について述べます。

Vimを使う場面

インフラエンジニアがエディタを使う場面として、おおよそ次のものが挙げられます。

・設定ファイルを編集する
・バッチやスクリプトを作る
・ログファイルをじっくりみる

どのサーバやOSであっても、これらの操作をすばやく行えるVimはインフラエンジニアの業務特性に合ったエディタと言えます。

障害対応などの場面において、各種設定ファイルを編集するなどの操作が日常的に発生します。その際いちいち検索エンジンなどで操作方法を検索していては迅速な障害対応が行えません。お医者さんが医学書を見ながら手術を行えないのと同様、インフラエンジニアは最低限操作感に慣れていなければなりません。Vimの操作についてはほかの章を参考にしてぜひ練習しておきましょう。

カスタマイズはあえてせず使う

Vimは~/.vimrcを書き換えることでさまざまなカスタマイズを行うことができます。しかし著者の場合は、日々さまざまなサーバを扱っている関係上、カスタマイズされた環境に慣れてしまうと、カスタマイズされていない環境のサーバでは即時対応力が落ちるので、カスタマイズして使ったほうが明らかに便利な設定があったとしても、あえて標準のままで使うようにしています。

Vimは初心者に敷居が高い

Vimの習得は、初級インフラエンジニアにとって負担が大きいと言えます。ただでさえLinuxなどのUNIX系OSのCUI操作に慣れていないのに、そのうえでVimの独特な操作感はますます敷居を高くしています。この件について筆者も過去、いろいろ考えてみたことがありますが、どうしても慣れてもらうほかに方法がないようです。

初級インフラエンジニアがVimを扱う際によく見られる失敗事例を見てみましょう。

場面1 Vimから抜ける方法がわからずCtrl-Zで抜けてしまう

Windows上などで動くエディタであれば、通常は「ファイル」のドロップダウンリストから「終了」を選ぶとエディタから抜けることができます。それに対してVimでは`:q!`(編集中のファイルを破棄して抜ける)や`:ZZ`(編集中の

ファイルを保存して抜ける)などの方法でVimから抜けることができます。この操作はあらかじめ知識がないと絶対思いつかない操作と言えます。

経験の浅い初心者は、いろいろ試行錯誤をしているうちにCtrl-ZでVimから抜けられる(ように見える)ことがわかり、以降Vimを使うたびにCtrl-Zで抜けるようになります。ご存じのとおりCtrl-Zはプロセスを中断するだけですので、Vimから抜けたように見えても、実際はプロセスが残っています。先輩エンジニアは、初級エンジニアがこういった悪習慣を行っていることに気づいたら直ちにやめさせなければなりません。

場面2　ほかにVimで編集中のファイルがあるにもかかわらずVimで開く

先ほどのCtrl-Zの話にもつながる部分がありますが、Vimでは、ほかにVimで編集中のファイルを編集しようとした場合、**図1**のような警告が出ます。初級エンジニアは「意味がわからない」「英語なので読めない」「読めても読みたくない」ため、この警告を無視してファイルを編集して保存しようとします。しかし当然のことながらいくら保存しようが、あとからもともとのファイルを編集中のプロセスが保存したら内容が上書きされることになります(この警告が出た場合は、素直にQもしくはAで抜けるのが安全です)。

♠

ここで紹介したことは、誰しも一度は同じような経験をしたことがある話です。ただし、インフラ運営においてはこれらのミスが致命的な障害を引き起こす可能性もあるため、単なる笑い話で済ませないように気をつけましょう。

知っておくと便利な操作

次に、インフラ運用の場面で知っておくと便利なVimの使い方の例を紹介します。

UndoとRedo

基本的な操作ではありますが、UndoとRedoは知っておくとたいへん便利です。設定ファイルを書き換える途中でいろいろ試行錯誤するときに便利に扱うことができます。

ここでは実際にUndo/Redoの操作を試して

▼図1　Vimで編集中のファイルを別のVimで編集しようとしたときのメッセージ

```
E325: ATTENTION
Found a swap file by the name ".sample.txt.swp"
          owned by: root   dated: Thu Nov 20 14:57:20 2014
         file name: ~root/sample.txt
          modified: no
         user name: root   host name: TESTSVR01
        process ID: 71381 (still running)
While opening file "sample.txt"
             dated: Thu Nov 20 14:57:19 2014

(1) Another program may be editing the same file.
    If this is the case, be careful not to end up with two
    different instances of the same file when making changes.
    Quit, or continue with caution.

(2) An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r sample.txt"
    to recover the changes (see ":help recovery").
    If you did this already, delete the swap file ".sample.txt.swp"
    to avoid this message.

Swap file ".sample.txt.swp" already exists!
[O]pen Read-Only, (E)dit anyway, (R)ecover, (Q)uit, (A)bort:
```

みましょう。

①文章の中でpenと入力します(図2)。
②ノーマルモードにして、そこでUndoをします。Undoはuです(図3)。
③さらにRedoをします。RedoはCtrl-rです(図4)。

直感的な操作方法とは言えないですが、VimにおけるUndoとRedoの操作は覚えておいて損はありません。

ウィンドウ分割

たとえばログファイルを見ながら設定ファイルを編集したいといったようなことがあるとします。この場合いちいちファイルを開いて、閉じて、また別のファイルを開いて、というようなことをやるのではなく、Vim内のウィンドウを分割してそれぞれ別のファイルを操作／編集できます。

このウィンドウ分割機能の存在自体は知っていても、実際に使っていないという方も多いと思います。しかし使い慣れておくと、とくに障害対応やちょっとしたスクリプト編集などのときに少しだけ便利なので、この機会に練習しておくと良いと思います。

それでは実際に試してみましょう。まずはVimで1つめのファイルを開きます(図5)。

```
$ vim sample.log
```

続いて2つめのファイルを開きながらウィンドウを分割します(図6)。

```
:split sample.conf
```

するとウィンドウが分割され、1つのVimウィンドウ中に2つのファイルの内容が現れます(図7)。

▼図2　penと入力(This is a .はすでに入力されている場合です)

▼図3　ノーマルモードにして、uを押してUndoする

▼図4　Ctrl-rを押してRedoする

さらにCtrl-W +という操作を行うと、操作中のウィンドウを1行広くできます(**図8**)。逆はCtrl-W -です。

ウィンドウの移動についてはいろいろなキーバインドがありますが、一番よく使われるのはCtrl-W Ctrl-Wです。この操作を繰り返すことで求

▼図5　sample.logファイルを開く

```
server
Jul 29 14:16:23 Updated: kernel-firmware-2.6.32-431.20.5.el6.noarch
Jul 29 14:16:28 Installed: kernel-2.6.32-431.20.5.el6.x86_64
Jul 29 14:16:36 Installed: kernel-devel-2.6.32-431.20.5.el6.x86_64
Jul 29 14:17:18 Installed: xfsprogs-3.1.1-14.el6.x86_64
Jul 29 14:17:18 Installed: xfsdump-3.0.4-3.el6.x86_64
Jul 29 14:17:34 Updated: glibc-common-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:35 Updated: glibc-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:36 Updated: 1:net-snmp-libs-5.5-49.el6_5.1.x86_64
Jul 29 14:17:36 Updated: glibc-headers-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:36 Updated: glibc-devel-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:36 Updated: 1:net-snmp-5.5-49.el6_5.1.x86_64
Jul 29 14:17:36 Updated: 2:irqbalance-1.0.4-9.el6_5.x86_64
Jul 29 14:17:36 Updated: 1:dmidecode-2.12-5.el6_5.x86_64
Jul 29 14:17:37 Updated: glibc-utils-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:37 Installed: 1:mcelog-1.0pre3_20120814_2-0.13.el6.x86_64
Jul 29 14:17:37 Updated: nscd-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:38 Updated: glibc-2.12-1.132.el6_5.2.i686
Jul 29 14:17:42 Updated: openssl-1.0.1e-16.el6_5.14.x86_64
Jul 29 14:17:58 Installed: libsmbios-2.2.27-4.12.1.el6.x86_64
Jul 29 14:17:58 Installed: python-smbios-2.2.27-4.12.1.el6.x86_64
Jul 29 14:17:58 Installed: smbios-utils-python-2.2.27-4.12.1.el6.x86_64
Jul 29 14:17:58 Installed: yum-dellsysid-2.2.27-4.12.1.el6.x86_64
Jul 29 14:18:07 Updated: libxml2-2.7.6-14.el6_5.2.x86_64
Jul 29 14:18:07 Updated: libxml2-python-2.7.6-14.el6_5.2.x86_64
                                                          1,1           Top
```

▼図6　:split sample.confと入力(splitはspと省略可能)

```
server
Jul 29 14:16:23 Updated: kernel-firmware-2.6.32-431.20.5.el6.noarch
Jul 29 14:16:28 Installed: kernel-2.6.32-431.20.5.el6.x86_64
Jul 29 14:16:36 Installed: kernel-devel-2.6.32-431.20.5.el6.x86_64
Jul 29 14:17:18 Installed: xfsprogs-3.1.1-14.el6.x86_64
Jul 29 14:17:18 Installed: xfsdump-3.0.4-3.el6.x86_64
Jul 29 14:17:34 Updated: glibc-common-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:35 Updated: glibc-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:36 Updated: 1:net-snmp-libs-5.5-49.el6_5.1.x86_64
Jul 29 14:17:36 Updated: glibc-headers-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:36 Updated: glibc-devel-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:36 Updated: 1:net-snmp-5.5-49.el6_5.1.x86_64
Jul 29 14:17:36 Updated: 2:irqbalance-1.0.4-9.el6_5.x86_64
Jul 29 14:17:36 Updated: 1:dmidecode-2.12-5.el6_5.x86_64
Jul 29 14:17:37 Updated: glibc-utils-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:37 Installed: 1:mcelog-1.0pre3_20120814_2-0.13.el6.x86_64
Jul 29 14:17:37 Updated: nscd-2.12-1.132.el6_5.2.x86_64
Jul 29 14:17:38 Updated: glibc-2.12-1.132.el6_5.2.i686
Jul 29 14:17:42 Updated: openssl-1.0.1e-16.el6_5.14.x86_64
Jul 29 14:17:58 Installed: libsmbios-2.2.27-4.12.1.el6.x86_64
Jul 29 14:17:58 Installed: python-smbios-2.2.27-4.12.1.el6.x86_64
Jul 29 14:17:58 Installed: smbios-utils-python-2.2.27-4.12.1.el6.x86_64
Jul 29 14:17:58 Installed: yum-dellsysid-2.2.27-4.12.1.el6.x86_64
Jul 29 14:18:07 Updated: libxml2-2.7.6-14.el6_5.2.x86_64
Jul 29 14:18:07 Updated: libxml2-python-2.7.6-14.el6_5.2.x86_64
:split sample.conf
```

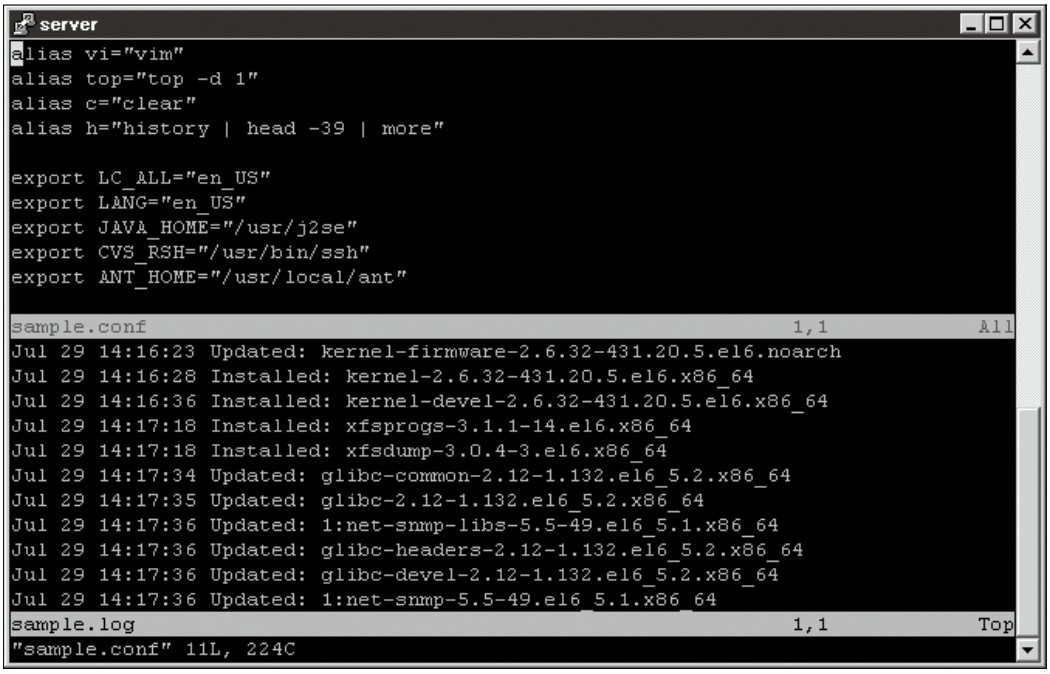
めるウィンドウにフォーカスを移動できます。

表1にウィンドウの基本的な操作について整理しましたので参考にしてみてください。

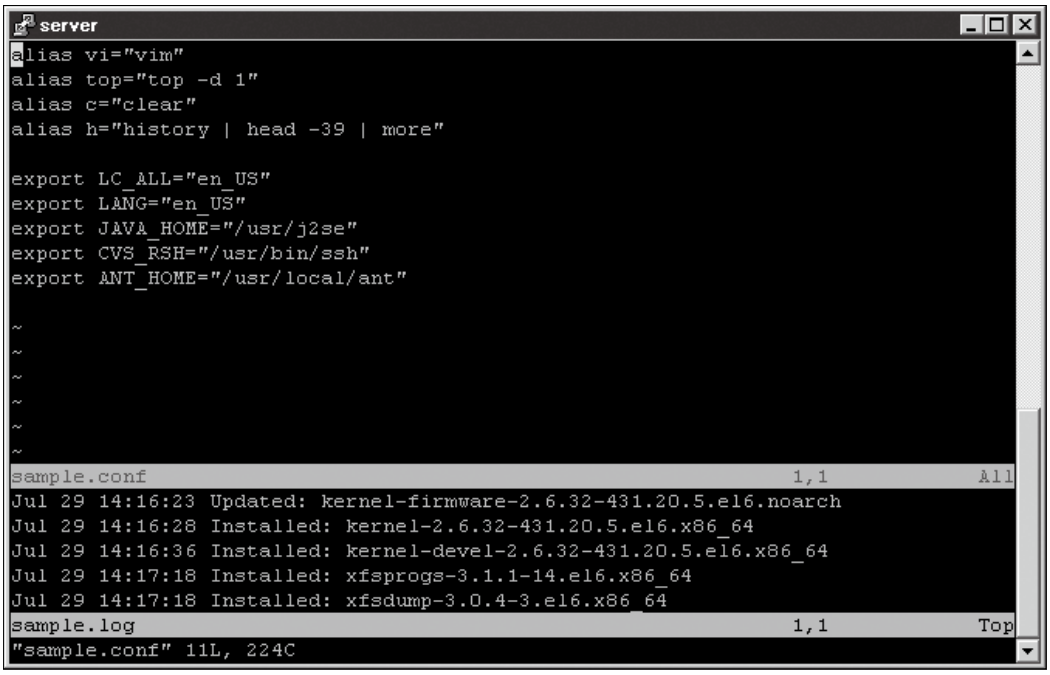
インデントの編集

設定ファイルを編集している際、きれいにインデントさせたいときがあります。もちろんカー

▼図7　上半分がsample.confで下半分がsample.log



▼図8　5行広げた例



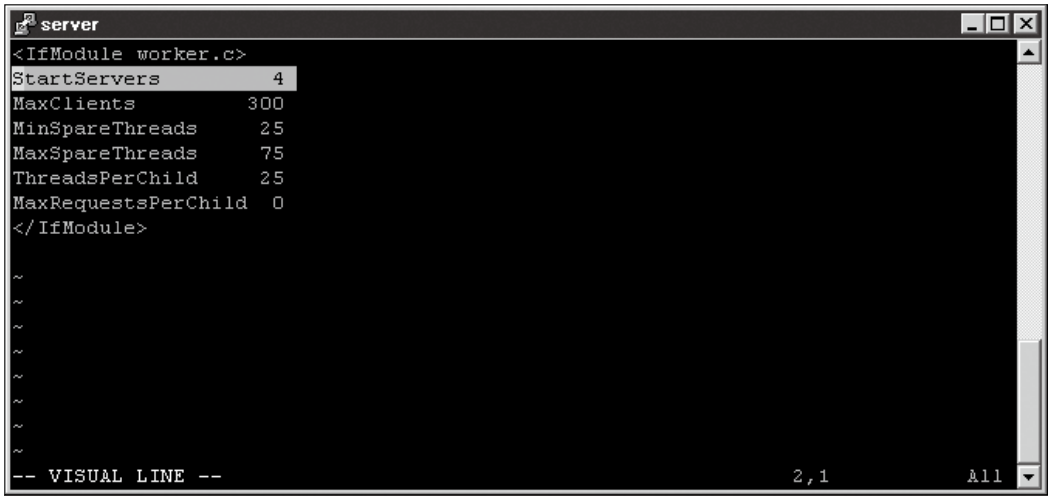
▼表1　ウィンドウの基本的な操作

Ctrl-W Ctrl-W	次のウィンドウの移動
Ctrl-W k	上のウィンドウに移動
Ctrl-W j	下のウィンドウに移動
Ctrl-W +	操作中のウィンドウを1行広くする
Ctrl-W -	操作中のウィンドウを1行狭くする
:q!	操作中のウィンドウを閉じる
:qa!	すべてのウィンドウを閉じる
:w	操作中のウィンドウ内のファイルを保存する
:ZZ	操作中のウィンドウ内のファイルを保存して閉じる

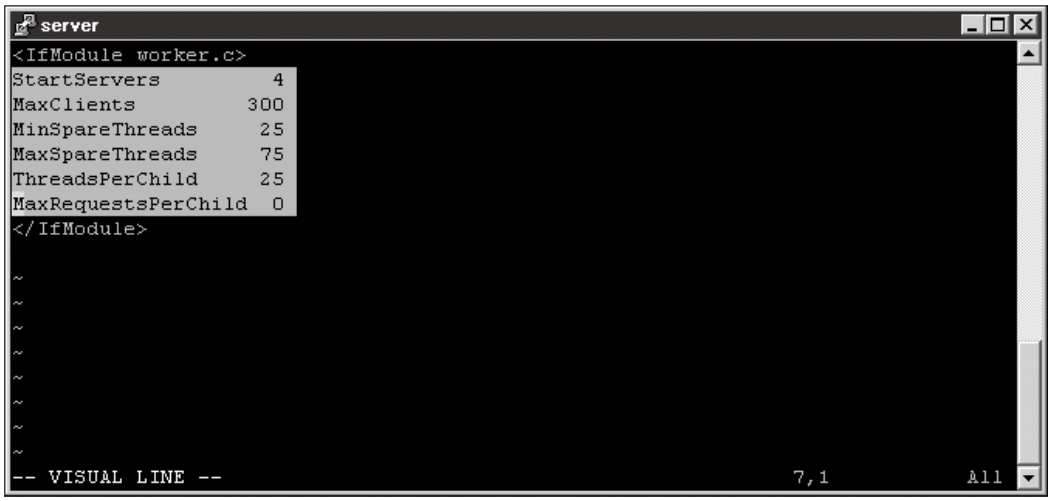
ソルキーとスペースキーを駆使することでインデントさせることもできますが、対象の行数が多いと、その方法は結構手間がかかります。また手作業でインデントさせるとスペースとタブが混在する場合があり、あまりきれいに仕上がらないときがあります。こんなときはビジュアルモードを使って複数行を一括でインデントできます。

まずはインデントを行いたいファイルをVimで開き、Vでビジュアルラインモードにします(**図9**)。次にインデントさせたい行をkやjで選択します(**図10**)。そして>を押すとイン

▼図9　ビジュアルラインモード



▼図10　インデントする行を選択する



デントされます(**図11**)。<を押すと戻ります。

この操作はそこそこ直感的ですし、ビジュアルモードを使いこなす最初の取っ掛かりとして活用してみるのも良いと思います。

選択部分の文字数をカウント

たまに作成しているファイル中の一部分の文字数をカウントしたいときがあります。こんなときにもビジュアルモードが活用できます。

文字数をカウントしたいファイルをVimで開き、Ctrl-v(WindowsはCtrl-q)でビジュアルブロックモードにします。ほかのモードでもかまいません。

次に文字数をカウントしたい範囲を移動キーを使って指定します(**図12**)。

そしてg Ctrl-gを押すとウィンドウの下部に文字数だけでなく行数、単語数、そして文字数(バイト数)が表示されます。**図12**の例では、22行中7行選択し、単語数98中62個、文字数703バイト中422バイトとなっています。

まとめ

最後にこういうことを言うのは何ですが、も

▼図11　インデントされた

```
server
<IfModule worker.c>
        StartServers         4
        MaxClients         300
        MinSpareThreads     25
        MaxSpareThreads     75
        ThreadsPerChild     25
        MaxRequestsPerChild  0
</IfModule>

~
~
~
~
~
~
~
~
~
6 lines >ed 1 time                                  2,1-8         All
```

▼図12　下から2行目に文字数などが表示されている

```
server
NAME
       vim - Vi IMproved, a programmers text editor

SYNOPSIS
       vim [options] [file ..]
       vim [options] -
       vim [options] -t tag
       vim [options] -q [errorfile]

       ex
       view
       gvim gview evim eview
       rvim rview rgvim rgview

DESCRIPTION
       Vim  is  a  text editor that is upwards compatible to Vi.  It can be used to
       edit all kinds of plain text.  It is especially useful for editing programs.

       There  are  a  lot of enhancements above Vi: multi level undo, multi windows
       and buffers, syntax highlighting, command line editing, filename completion,
       on-line help, visual selection, etc..  See ":help vi_diff.txt" for a summary

Selected 7 of 22 Lines; 62 of 98 Words; 422 of 703 Bytes
```

し「Vimが好きか」と問われたら、おそらく「とりわけ好きというほどでもないけれども、ほかに代替手段もないし、それなりに必要な機能がそろっているので使っている」と答えると思います。日常的に使うエディタですので好き嫌いがあって当然かと思います。しかしインフラエンジニアにとってVimは避けて通れない必須ツールですので、使わざるを得ないのであればさりげなく使いこなしたいものです。

Vimは最低限の操作(カーソル移動、ファイル保存、終了)さえ覚えれば、とりあえずは使えますし、それでも実運用上は多少不便であっても大きな問題はありません。しかしこの手のツールは使いこなせば使いこなすほど日々の仕事が楽になることは明確ですので、積極的に使いこなしていきましょう。

とはいえ、Vimの豊富な機能を無理して使いこなそうと考えなくても良いです。著者が大事だと考えるのは、日常不便と感じたことが出たら、それをVimの機能やプラグインで解消できないかと発想することだと思います。人は新しいことに挑戦するのは面倒に思うものですが、ちょっとした不便はちょっとした手間で解消できるものです。この機会にぜひVimを活用して日頃の不便を少しずつでも解消していきましょう。SD

コラム 緊急時の対応例

Vimとは直接関係ないですが、Linuxサーバを運営する際たまに遭遇するVim操作関連の問題と対処方法を紹介します。

①rootアカウントでログインしているのに重要な設定ファイルが編集できない

とくに重要なファイルはimmutable(ファイルの変更を許可しない。削除もリネームもできない)属性がついている可能性があります。

```
# whois
root

# vim /etc/hosts
"/etc/hosts" [readonly] 3L, 206C
```

この場合はchattrコマンドを使って属性を無効にすることでファイルの編集が可能となります。

```
# lsattr /etc/hosts
----i---------e- /etc/hosts

# chattr -i /etc/hosts

# lsattr /etc/hosts
-------------e- /etc/hosts
```

ファイルを編集し終えたら元に戻しておきましょう。

```
# chattr +i /etc/hosts

# lsattr /etc/hosts
----i---------e-- hosts
```

②Linuxをシングルユーザモードで起動後に、ファイルの編集ができない

Linuxをシングルユーザモードで起動する際、/を読み込み専用でマウントしているためファイルの編集ができません。

そこで/を読み書きできるようにするために、下記の要領で再マウントする必要があります。

```
# mount -o remount,rw /
```

③日本語ファイルをVimで開くと文字化けする

文字化けの原因にはターミナルソフトの設定、OSの言語設定、vimの言語設定などさまざまな切り分けポイントがありますが、Vimの言語設定に絞ると、日本語ファイルをVimで開くと文字化けする際は、次のように設定すると解決する可能性があります。

```
~/.vimrcに、自動判別の設定を追加します。
.vimrcがない場合は作成します。

:set encoding=utf-8
:set fileencodings=utf-8,euc-jp,sjis,
iso-2022-jp
```

第4章 vim-markdownという選択 実用Tips&対策［文書作成編］

Writer mattn Twitter @mattn_jp

業務別Vimの使い方、最後のパートは日本語文書の作成編です。インプットメソッドまわりのVimの日本語入力は、これまでさまざまな改善がなされてきました。その実装の歴史をふりかえります。また、VimにおけるMarkdownの文法・シーン別の編集方法とそれを助けるプラグインを紹介します。

Vimはどちらかというとプログラマ向けのテキストエディタです。では、文章を書くのにはどうでしょうか？　プログラミング言語は英語ベース(1バイト文字)なので英文を書くにはとても適しています。しかし日本語文章を書くにいたっては残念ながら、巷の評判は良いものではありません。文章を書くときだけはほかのテキストエディタを使うという方もチラホラ目にします。本当にVimは日本語入力に適さないテキストエディタなのでしょうか？　ご存じのようにVimは拡張できるテキストエディタです。工夫しだいでは日本語の取り扱いも十分に可能なのです。ただし、Vimで文章を入力する際にはいくらか知識と準備が必要です。

Vimの日本語入力で困る点

Vimはモードを持ったテキストエディタです。ノーマルモードに戻る際、ユーザのほぼ全員がインプットメソッドがオフになっていることを期待しているでしょう。しかし残念ながら、インプットメソッドが持っている入力モードをVimから変更することは現状できません。その結果として、意識せずにEscキー(もしくはCtrl-[)をタイプすると、Vimは漢字変換モードのままノーマルモードになってしまいます。

Windows上での日本語入力

WindowsのGUI版(gvim.exe)の場合は次節で説明するiminsertというオプションによりこの「ノーマルモードでインプットメソッドがオンのままになる」問題を解決できています。CUI版(vim.exe)の場合は、後述のようにインプットメソッドを制御できません。

Linux上での日本語入力

Vimの開発を援助している側からこういうことを言うのはとても心苦しいのですが、Vimのインプットメソッドまわりはお世辞にもよくできているとは言えません。Vimはモードを扱うテキストエディタですが、そのモード切り替えとインプットメソッドの連携ができていないのです。なお、インプットメソッドを扱う実装には3者の立場を考慮しなければなりません。

❶インプットメソッドをまったく使わない(使いたくない)人
❷挿入モードでは常にオンを期待する人
❸挿入モードでオン／オフを切り替えたい人

英語圏の人は❶、日本人は❸にあたります。ハングルのインプットメソッドを使う人たちが❷にあたると聞いたことがあります。

これらをふまえVimには2つの入力機構が実装されています。1つはインプットメソッド、もう1つはlmapです。インプットメソッドは昔で言えばkinput2、現代で言えばanthy、uim、mozcがそれにあたります。lmapは日本語のかな入力のようなものと思ってください。

もう少し説明すると、日本語入力は英字を数文字タイプしてひらがなを入力し、それを漢字に変換することで行われます。しかし世界には英字を数回タイプすることで特殊文字を入力する人々もいます。一般的にはdigraphが有名ですが、Vimではdigraphのほかにマルチバイトに特化した実装としてlmapがあるのです。

しかしながらlmapは前述のように、かな入力に値するものであり漢字変換は含まれません。lmapを使用してひらがなを入力することはできますが、それを漢字に変換する方法がないのです。結果としてVimで日本語入力を行うにはインプットメソッドを使うほかありません。Vimのインプットメソッドは❶〜❸の要求を満たすために次のオプションを用意しています。

・imdisable
　インプットメソッドを無効にし、オンにできないようにするオプション
・iminsert
　インサートモードに入った際にインプットメソッドまたはlmapを切り替えるオプション
・imsearch
　検索モードに入った際にインプットメソッドまたはlmapを切り替えるオプション
・imactivatekey
　インプットメソッドをアクティブにするキーをVimに教えるオプション(Linux GUI向け)

WindowsのGVimではimactivatekeyを除くすべてが期待どおりに動作します。しかしながらLinuxのGUI上ではすべて期待どおりには動作しません。正確には正しく動作しなくなりました。インプットメソッドは今では各GUIコンポーネントの部品の1つとしてレイヤ状に構築されており、近代ではXIM(X Input Method)はほとんど使われないものになってしまいました。

XIMにはプログラムからインプットメソッドのオン／オフを切り替えるAPIが存在しました。しかしgtk_im_moduleなど、UIの入力機構には現状存在しません。つまり先の4つのオプションのすべてが正しく動作しなくなったと同時に、使いにくい状態に逆戻りしてしまいました。

前述のとおり、インサートモードでインプットメソッドをオンにしたあと、Esc キー(Ctrl-[)をタイプすると、日本人Vimmerであればインプットメソッドがオフとなりノーマルモードに遷移することを期待してしまいます。

しかしこれらの理由で、Vimからインプットメソッドをオフに切り替えることができません。結果、ノーマルモードでインプットメソッドがオンのままとなってしまい、たとえば「nn」をタイプすると「ん」が入力され、期待しない動作となってしまいます。これに痺れを切らしてか、Linux上のインプットメソッドのいくつかにはEsc キーを押したときにインプットメソッドをオフにする、いかにもVimmerのための設定が用意されている場合もあるので設定しておいた方が無難です。もちろんEsc キーでなくCtrl-[を使う人は別途設定が必要です。

このような状況から抜け出そうと、Vimmerたちはいろいろなハックを行ってきました。

skk.vim

SKK入力方式を模擬できます。L辞書を使って漢字変換をし、~/.skk-jisyoにユーザ辞書を保存できるしっかりしたものです。Vimの中で動作するインプットメソッドですのでsshでログインしたサーバ内でも問題なく動作します。

vim onlineでは開発が止まってしまいましたが、現在はtyruさんのリポジトリ[注1]で管理されています。

注1) URL https://github.com/tyru/skk.vim

eskk

skk.vimに触発され、もっとカスタマイズ性に優れたSKK入力方式がほしいというVimmerたちの願いの中、tyruさんが開発を始めたプラグインがeskk[注2]です。skk.vimよりも細かい設定変更が可能であり、skkserverとも通信できるようになっています。

◆◆◆

なお、Vimのインプットメソッドまわりの状況が現在どのようになっているのかというと……実は以前から何も進化していません。これはおそらくユーザが現状に慣れてしまい、Esc キー(Ctrl-[ )のタイプ前にはきちんと、インプットメソッドをオフにするという習慣ができてしまったのが理由だと推測しています。現に筆者もこの記事をLinux上のVimで書いていますが、文句も言わずにインプットメソッドをオフにしながらノーマルモードに戻る操作を繰り返しています。

autofmt

日本語の扱いで発生する問題はインプットメソッドだけではありません。エンコーディングに関しては最近はutf-8で決め打ちになりつつあるという、とても良い時代になってきましたが、日本語には良くも悪くも禁則処理が存在します。行を折り返した際、次の行が「、」で始まらないしくみや設定が必要になります。

また、ノーマルモードでJをタイプすると次の行がカーソル行の行末に連結されますが、英語圏の仕様により空白が挿入されます。この辺をうまくフォーマットしてくれるのがautofmt[注3]です。

行の折り返しで「、」が行頭に来ることがないように、また行連結で空白が入らないように動作します。

注2） URL https://github.com/tyru/eskk.vim
注3） URL https://github.com/vim-jp/autofmt

Markdownの入力に便利なカスタマイズ

最近のREADMEはMarkdownで書かれていることが多くなりました。ブログのエントリもMarkdownで書くという人も多いようです。

筆者も仕事や趣味でメモを取るときはMarkdownを使っています。とくにGitHubを使っているエンジニアであれば、Markdownは必須スキルと言っていいでしょう。

先に挙げた日本語まわりの設定を施しているならば、VimとMarkdownは実は非常に相性の良いテキストフォーマットであると言えます。

Markdownの文法と、そのシーンごとの編集方法を説明していきます。まずMarkdownをシンタックスハイライトするために、vim-markdown[注4]というプラグインをインストールしておきます。

ほかにもVim上でMarkdownをハイライトできるプラグインがいくつかありますが、筆者が知る限りTim Pope作のこのプラグインが一番よくできていると思います。

以降、Markdownの文法に対してどのようにVimを使っていくかを説明します。代表的なMarkdownの文法には次のものがあります。

・見出し
・箇条書き／リスト
・水平線
・リンク／画像
・コード／引用
・表

見出し

見出しは行の先頭に#を書き、その個数で段落のレベルを決めます。Vim使いであれば/^#で検索して見出しを行き来することができます。

注4） URL https://github.com/tpope/vim-markdown

箇条書き／リスト

筆者の経験上、箇条書きの多くはほかの資料からコピー&ペーストして作成することが多いように思います。ペーストされた行を箇条書きにする場合でもVimであれば簡単です。

```
aaa
bbb
ccc
```

このような行をビジュアルモードにおいて選択した状態で、

```
:'<,'>s/^/*␣/
```

を実行すると、行頭が*で置換され、箇条書きができあがります。

```
* aaa
* bbb
* cc
```

リストは既存の行に連番を作る必要があり箇条書きほど簡単ではありません。そこでリストの作成にはマクロ(P.55コラム参照)を使います。次は一例にすぎませんが、連番付きの行を作る方法を紹介します。

```
0i1.␣Esc0qqvf␣yjP0 Ctrl-a q
```

マクロ登録開始　レジスタ(a〜z)　マクロ登録終了

と入力してください。分解して説明すると、

①始めに0i1.␣Esc0で最初の行の行頭に「1.」を足す。この数字が連番の開始番号となる
②次にqqvf␣yjPで「1.」の部分をyank(コピー)し、次行の先頭にペーストする
③最後に0 Ctrl-a qで行頭の数字をインクリメントする

この②と③の操作が「q」というレジスタに格納されるので、10個の連番付き行を作りたいのであれば10@qとタイプすることで既存の行に連番が作られます。

```
1. aaa
2. bbb
3. cccc
```

マクロはとても便利な機能ですが、慣れていないと結局手動で連番を書く方が早く終わってしまうこともあります。どうしてもマクロが慣れない人は、次のようなスクリプトを使って連番付きの行を作るという方法もあります。

```
function! s:vnr() range
  let n = 1
  for i in range(a:firstline, a:lastline)
    call setline(i, n . '. ' . getline(i))
    let n += 1
  endfor
endfunction
vnoremap <leader>nr :call <SID>vnr()<cr>
```

ビジュアル選択した状態で<leader>nr(何も設定していない状態であれば\nr)をタイプすると自動的に連番付きの行が生成されます。vimrcなどにコピーして使ってください。また、連番を作るという目的に特化したVimプラグインも存在します[注5、注6]。Markdownだけでなく、プログラミングにもたいへん便利です。

水平線

水平線は数種類あるのですが、筆者は----を使います。Vim使いであれば5i-Esc xなどと入力して一気に作ってしまいましょう。

リンク／画像

Markdownのリンクは[タイトル](URL)の形式で記述します。そのままタイプしても良いのですが、次のように既存のURLからMarkdown記法に変換するマッピングをvimrcに書いておくと便利かもしれません。

注5) URL https://github.com/deris/vim-rengbang
注6) URL http://deris.hatenablog.jp/entry/2013/06/16/174559

```
vnoremap <leader>mdu ygvs[]
(<c-r>")<esc>?[]<cr>a
```

URLをビジュアルモードで選択した状態で<leader>mdu(何も設定していない状態であれば\mdu)をタイプすると

```
[](http://www.google.com)
```

このようなMarkdown記法のリンクが生成されます。[]の中にカーソルが移動するので、あとはタイトルを書けば完成です。

```
[Google](http://www.google.com/)
```

ちなみに筆者作のemmet-vim[注7]の最新版ではURLの末尾にカーソルを移動してCtrl-y aをタイプすると、自動でリンク先のタイトルを調べてMarkdown記法を生成してくれます。

コード／引用

Markdownでコードを書くには次のように記述します。

````
```ruby
  [1, 2, 3].each do |x|
    puts x
  end
```
````

注7) URL https://github.com/mattn/emmet-vim

▼リスト1　シンタックスハイライトの設定

```
let g:markdown_fenced_languages = [
\  'coffee',
\  'css',
\  'erb=eruby',
\  'javascript',
\  'js=javascript',
\  'json=javascript',
\  'ruby',
\  'sass',
\  'xml',
\]
```

rubyと書かれた部分にはそのコードがどのプログラミング言語で書かれているのかを表します。GitHubではこのプログラミング言語名を基にコードのシンタックスハイライトが行われています[注8]。

Vimは多くのプログラミング言語のシンタックスハイライトに対応していますが、Markdownに埋め込まれたコードに対しても色付けすることができます(**図1**)。

vimrcに**リスト1**を書いておくと、指定したプログラミング言語の識別が現れると自動で埋め込みシンタックスハイライトを行ってくれます。ここではjs=javascriptのように別名を設定することもでき、短い表記に省略できます。

引用は正規表現を用いた置換を使います。引用行は先頭に>␣を挿入すればいいので、引用表記したい部分をビジュアルモードで選択して、

```
:'<,'>s/^/>␣/
```

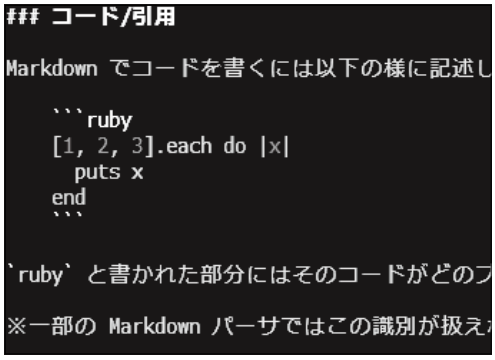
とするだけです。簡単ですね。

表

Markdownの表はASCII文字で書きます。Markdown専用のプラグインを使ってもよいの

注8) Markdownパーサの中にはこの識別が扱えないものもあります。

▼図1　マークダウン中のプログラミング言語へのハイライト



ですが、筆者はAligntaというプラグインを使ってテーブルを整形しています。

まず体裁を無視して表を書ききります。

```
品名| 値段
コーラ|120
ハンバーガー|200
スマイル|0
```

次に、表部分をビジュアルモードで選択して、

```
:'<,'>Alignta_|
```

のようにAligntaコマンドを「|」(パイプ)を指定して実行すると選択部分が次のように整形されます。

```
品名          | 値段
コーラ        | 120
ハンバーガー  | 200
スマイル      | 0
```

あとは2行目に罫線を入れて完成です。19i-Escのように幅分の-を入力し、交差部分を|に書き換えます。数値など、右寄せ／左寄せを行うには罫線部分に:を指定します。

```
品名          | 値段
--------------|----:
コーラ        | 120
ハンバーガー  | 200
スマイル      | 0
```

これで、**図2**のように表示されます。

注9) URL https://github.com/h1mesuke/vim-alignta

▼図2　表への整形

品名	値段
コーラ	120
ハンバーガー	200
スマイル	0

なお、ほかにも整形プラグインはあるのですが、上記のようなマルチバイト文字が混じった場合でも正しく整形できるのは筆者が知る限りAligntaだけです。

Markdownのプレビュー

Markdownを書いていると、実際にブラウザにレンダリングされた際にどのような見栄えになっているか気になってきます。

Markdownのプレビューを行えるプラグインはいろいろあるのですが、筆者はprevimを使っています[注10]。previmはkannokannoさんによって開発されており

- Webサーバのような外部プログラムが不要
- デフォルトではVimの操作感に影響を与えない
- 見た目のカスタマイズが可能
- リアルタイム編集も可能
- Markdown/reStructuredText/textideに対応

という聞いただけでも便利そうなプラグインです。ブラウザを自動で開く場合だけopen-browser[注11]が必要です。

使い方は、markdownファイルを開いたあと、次のコマンドを実行します。

```
:PrevimOpen
```

すると、ブラウザが起動して現在編集中の内容がプレビューで表示されます(**図3**)。バッファを変更し、保存するたびにプレビューが更新されるようになっています。

使用するブラウザを変更したい人は次のように変更します(Mac OS Xでブラウザを Firefoxに変更する場合)。

```
let g:previm_open_cmd = 'open -a Firefox'
```

注10) URL https://github.com/kannokanno/previm
注11) URL https://github.com/tyru/open-browser.vim

またデフォルトの見た目が気に入らない人は次のようにカスタムCSSを指定します。

```
let g:previm_disable_default_css = 1
let g:previm_custom_css_path = '/Users/mattn/public_html/style.css'
```

本記事もこのpreviを使いながら執筆しましたが、とても重宝しました。

メモ取りとしてのMarkdown

前述のように筆者は業務でメモを取る場合にMarkdownを使用しています。その際、memolistというプラグインを使っています[注12]。

注12) URL https://github.com/glidenote/memolist.vim

▼図3　previm

▼リスト2　memolistの設定

```
nnoremap ,mf :exe "CtrlP" g:memolist_path<cr><f5>
nnoremap ,mc :MemoNew<cr>
nnoremap ,mg :MemoGrep<cr>
let g:memolist_memo_suffix = 'md'
let g:memolist_path = '~/memo/_posts'
let g:memolist_template_dir_path = '~/memo'
```

筆者の場合はファイル名で検索したい場合が多いので、memolistのフォルダを指定してCtrlP[注13]を起動しています(**リスト2**)。

また次のMarkdownテンプレートファイルを~/memo/md.txtとして配置しています。

```
---
title: {{_title_}}
date: {{_date_}}
layout: post
---
```

業務中に何か思いついた場合や、気になったことなどをすべてMarkdownのメモとして蓄積していきます。自宅で作業する場合も同じです。

なぜこんなテンプレートなのか、勘のいい方は気づいたかと思います。Jekylやmiddleman、その他多くのブログエンジンがサポートしている形式なのです。~/memoで`jekyll_new_.`とすると_config.ymlなど、必要なファイルが生成されるので、あとは_postsディレクトリにメモ記事を書き溜めていくといった具合です。_postsディレクトリをDropboxやbtsyncなどで共有しておくと便利ですね。

ちなみに筆者はお手製のjekylクローン、jedie[注14]を使っており、高速なmarkdown/html変換を行っています。

このほかにもVimを使ってMarkdownを書く際に便利なプラグインがたくさん存在します。

まとめ

Markdownのような文章の入力には難があると思われがちなVimですが、やり方しだいではとても便利に、かつ強力に編集することができます。筆者が挙げたプラグインや編集方法は、数ある方法の一部にすぎません。ぜひ自分に適した自分向けの編集方法を探してみてください。SD

注13) URL https://github.com/ctrlpvim/ctrlp.vim
注14) URL https://github.com/mattn/jedie

コラム マクロの活用方法

マクロとは一連のキーボードを記録し、再生することで単純な.による繰り返し操作以上の高度な操作を行える機能です。マクロは「記録開始」「操作」「記録終了」「再生」で構成されます。

記録の開始はqと**アルファベット文字**(a から z)をタイプして行います。記録の終了はqをタイプします。この間の操作が記録対象の操作となります。記録を行うと、記録開始でタイプしたアルファベット文字(たとえばx)が示すレジスタに操作内容が登録されます。あとは@のあとにレジスタをタイプ(例では@x)するとマクロが再生されます。また@@をタイプすると前回再生したマクロが再実行されます。

マクロは、ほぼすべてのキーボード操作を記録できます。マクロ実行中のマークおよびジャンプ、マクロ実行中のマクロ記録および再生もできます。またマップされたキーも記録の対象ですので、複雑な機能をマップしておきマクロと組み合わせることもできます。undoと.によるやりなおし操作、繰り返し操作も記録されます。簡単な例を示します。たとえば以下のテキストがあったとします。

```
apple
banana
strawberry
```

そして分割されたバッファに次の形式で英単語の和訳が書かれたテキスト(例はGENE95辞書の一部)があったとします。

```
apple
リンゴ,りんご,リンゴの木
```

そして、apple、banana、strawberryのすべてを和訳に置き換えたいとします。単純にすべての単語で同じ動作を繰り返しても良いのですが、こういった場面でマクロが活躍します。 ではqqをタイプして「q」レジスタへの記録を開始しましょう。

(1)英単語を消して無名レジスタに放り込む

ノーマルモード中、appleの上でdawをタイプします。これによりappleが削除されると同時に無名レジスタ「"」にappleが格納されます。

(2)辞書バッファに移動して検索する

Ctrl-w kで上部に分割されたバッファに移動、appleという単語の行を探すためにgg/^Ctrl-r "$<cr>をタイプします。Ctrl-r "で無名レジスタ「"」を貼り付けていますので実際には /^apple$ に置き換えられます。冒頭では検索単語を1行目から検索するためにggをタイプしました。

(3)1行下の最初の単語をyank(コピー)して元のバッファに戻ってpaste

辞書バッファでjをタイプして1行下に移動、ywで単語をyank、Ctrl-w pで元のバッファに戻り、Pでpasteします。

(4)連続実行するために次の行に移動しておく

jで次の行に移動します。これをしておかないと、100@qと実行しても同じ行に対して実行されてしまいます。

ここまでできたらqをタイプして記録を終了します。これで「q」というレジスタには、カーソル上の単語を和訳し、1行下に移動する操作が格納されたことになります。 では残りの2行に対しても実行するために2@qをタイプします。

```
リンゴ
バナナ
イチゴ
```

大量に翻訳が必要な場合にとても有用な操作となりました。 この説明でわかるとおり、マクロを使うためには編集前の状態から編集後の状態に変更するにはどのキーをタイプしたら良いのかを把握しておく必要があります。ここがマクロが難しいと思われる由縁でもあります。

コラム2

Vimの真のチカラを引き出すパラダイムシフト

Vimは編集作業をプログラムにする

Writer MURAOKA Taro (a.k.a. KoRoN) Twitter @kaoriya

まるでロボットを操縦して文字を書くかのよう!?

Vimとそのほかのテキストエディタ やIDEとの大きな違いを、一言で説明するのは簡単ではありません。しかしあえて挑むのであれば「ロボットを操縦しているようだ」と言うのが、Vimにおけるテキスト編集を説明付けてくれるでしょう。

ここで言うロボットとは、人間が行うべき作業を完全に代行してくれる専門用語としてのロボットではなく、ガンダムなどのSF作品にでてくるような、人が乗って操縦するロボットをイメージしています。土木作業に用いるショベルカーのイメージでも良いでしょう。

この比喩をもう少し具体的に掘り下げてみましょう。普通の紙に文字や絵を書くときには、自分の指でボールペンなどの筆記用具を持ち、とくに意識せずとも肩から腕、指を連動させて文字を書いていることでしょう。これならば非常に直感的で、自分が書きたいように書けます。これがまさにVim以外のエディタを扱うときの感覚です。

一方、Vimを使う場合には話が違ってきます。ショベルカーの先端にボールペンを付け、コントロールスティックを操作して文字を書くのです。もしくはロボットハンドにボールペンを持たせ、そのロボットに指示をして文字を書かせるのです。突拍子もなく感じるかもしれませんが、長年Vimを愛用している私にすれば、これがVimを使っているときの感覚をもっとも良く説明できています。

操作すべてがプログラムである意義とは

さらに大袈裟に言えば、このときの指示内容には各アームの角度調整や動作速度の指定、力加減の設定などの細かな情報、すべてを含んでいます。これでは慣れないうちは絵はおろか、ひらがな1文字を書くのにも大変苦労します。それこそがVimを使い始めたときに感じる、使いにくさの本質です。当然、文字を書くという目的だけの行為としては、しくみが大掛かりなうえにやるべきことが多くて、細か過ぎて、馬鹿げています。

それでもVimを使うことの意味とはなんでしょう。この問いへの答えも、この比喩の中にあります。実はこのVimというロボットハンドへの操作・指示は、すべてプログラムでもあるのです。漢数字の「十」という文字を書くのに、必要になるはずの最初の一本の横棒を書くためのプログラムは、少しパラメータを変えてあげれば、次に書くべき縦棒にも使えそうです。

プログラムであるということは、すなわち**再利用可能である**ことを示し、また**自動化できる**ということにほかなりません。これは非常に強力な概念です。日常行うテキスト編集作業が全部、プログラムであり、再利用可能であり、自動化の対象となりうるのです。

ですから熟練Vimユーザが Vimで作業をしているときは、単に文章を書いている・編集している以上の意味を持っています。それは、ある種のプログラミングです。すなわち文章を書くための、編集するためのプログラムをライブで書いている

のです。しばしばVimがプログラマ向けのエディタだと考えられるのは、Vimのこのような性質に基づきます。

もしもVimを使ってプログラムを書いているならば、それは今風に言うと、プログラムを書くためのメタプログラムをしている、というのも過言ではありません。また、一部の熟練Vimユーザが Vimを使ってある仕事をしているとき、気が付くと、その仕事をより効率的にこなすためのVimプラグインを作り始めている、トータルではプラグイン作りのほうにより多くの時間をかけてしまった、などということがあるのも、この性質を考えれば納得できるというものでしょう。

プログラムとシームレスなVimの機能

Vimの機能に話を戻しましょう。

Vimにおいて「FOO」という文字を入力するためにタイプすべき`iFOO`Escは、それ自体がプログラムです。だから、このプログラムの直前に「3回繰り返す」という意味の「3」を付け加えて、`3iFOO`Escとするだけで、「FOOFOOFOO」と入力できてしまいます。Vimはモーダルなテキストエディタではありますが、その機能はプログラムとモードレス＝シームレスなのです。

ですから熟練Vimユーザは、Vimが「モーダル」であることを指摘され、批難されてもあまりピンとは来ません。Vimは、**インタラクティブなエディタとしてはモーダルですが、プログラマブルなエディタとしてはこのうえなくシームレス**だからです。そして一般的にはエディタはインタラクティブなモノとの認識があるかもしれませんが、熟練Vimユーザはプログラマブルなモノとして認識しているので、そもそもの指摘自体が誤っているように感じられてしまうのです。

このシームレスにプログラマブルなテキスト編集ロボット＝Vimを構成する重要な機能には、操作を記録できるマクロ・レジスタや、制御構造を添加するVim script、それらをトリガーするキーマッピングやユーザコマンドやイベントを挙げることができます。これらの機能はとても地味ですが、Vimを効果的に使うつもりであれば早晩避けては通れません。

Vimを使い始めの頃は、とかくシンタックスハイライトやウィンドウ分割、それにタブ表示などの派手でわかりやすい、ほかのエディタにもあるような機能に目が行くことでしょう。しかし、本当にVimを使うのであれば、**Vimによるテキスト編集行為自体がプログラミングである**という前提をふまえて、それに関する前述のような機能を積極的に学んでみてください。

Vimの真のチカラを得るために

話をまとめましょう。

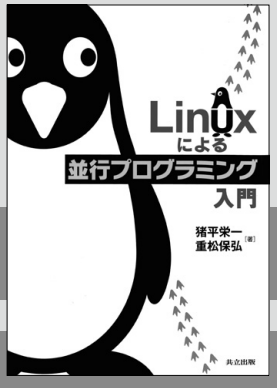
Vimはシームレスにプログラマブルであることが、ほかのエディタやIDEとは異なる最大の特徴です。ゆえにその特徴を活かすことで、Vimを使う真のチカラが享受できます。そのためには、普段行うすべての編集作業がプログラムであることを意識し、どうすれば再利用・自動化できるか常に考え、それにかかわる機能を利用・実践し続けることが、とても重要になってきます。

ただし、なんでもかんでも再利用・自動化すれば良い、というものでもありません。再利用や自動化にこだわり過ぎて、膨大な時間を投資したあげくに、実質的にメンテ不能なマクロやスクリプト＝独自システムを作りあげてしまう、そんな危険性があることは常に気に留めましょう。

この危険性はVimに限りません。プログラミング的な傾向の強いソフトウェアを使う際には、「なにかのタスクを完了させる」といった具体的な目標を掲げ、それを達成することがなにより重要です。そのうえで、次回以降の類似タスクに向けて、何を資産として残すべきか、それを考えることこそが真の再利用・自動化のメリットです。

とくに意識しないでも、普段の編集作業をほどほどに再利用・自動化できるようになったとき、そのときこそ熟練Vimユーザになったと、胸を張って言えるでしょう。SD

BOOK no.1 Linuxによる並行プログラミング入門



猪平 栄一、重松 保弘【著】
B5判、200ページ／価格＝2,600円＋税／発行＝共立出版
ISBN＝978-4-320-12380-9

本誌2014年8月号で「Linuxカーネルのしくみを探る」という特集があったが、その先の内容を教科書的にしたイメージなのが本書だ。forkとプロセスについて実験をしながら体験的に並行処理について学ぶことができる。説明はUbuntu上でC言語を用いて行われている。プロセス、シェル、ファイル入出力、パイプ、メッセージキュー、プロセス間通信などからスレッドやより高度な内容についても解説している。体裁はやさしい入門書というよりは教科書ふうなので、とっつきにくいかもしれない。本書のまえがきにも「C言語の中級課程のテキスト」「オペレーティングシステム」「組込みシステム」の授業の教科書としての利用を勧めている。

BOOK no.2 新装改訂版 Linuxのブートプロセスをみる

白崎 博生【著】
B5変形判、312ページ／価格＝2,800円＋税／発行＝KADOKAWA
ISBN＝978-4-04-891393-5

Linuxカーネルにおいて、ブートローダが起動するところからinitが開始するまでの「ブートプロセス」の処理をソースコードを基に解説した1冊。改訂にあたり、対象のLinuxカーネルを2.4.17/32bit版から3.3.4/64bit版に引き上げている。ソースコードはアセンブラ、C言語で書かれているので、深く理解するためには事前の勉強が必要だろう。また、理解を助けるため、第1章ではIntel CPUのアーキテクチャについても説明がされている。

おもな対象読者をカーネルハッカー志望の情報学生としているが、軽やかな文体で、基礎的な部分を省くことなく書かれているので、取り上げているテーマほどハードルは高くない。

BOOK no.3 デザイナーからプログラマーまで絶対わかる Gitバージョン管理

松島 浩道 【著】
A5判、 320ページ／価格＝2,500円＋税／発行＝SBクリエイティブ
ISBN=978-4-7973-8036-1

バージョン管理ソフト「Git」の機能について、GUIソフト「SourceTree」での操作も併せて1つ1つ解説している。Gitの関連サービスとして、GitHubを始めSourceForge.JPやBitbucketを紹介しており、とくにGitHubについては基本的な使い方から開発スタイル「GitHub Flow」の紹介まで、ページを多く取って解説がされている。Git環境の構築方法、リポジトリの各種設定や、代表的なサブコマンドを網羅したコマンドリファレンスもついており非常に実用的な一冊である。また、「Gitの内部構造を理解する」では「git add」「git commit」を打ったとき内部で何が起きているかといった内部構造の説明もあり、一歩踏み込んだ理解を助けてくれる。

BOOK no.4 15時間でわかる Java集中講座

宮下 明弘、工藤 雅人、原田 僚【著】／井上 誠一郎【監修】
B5変形判、416ページ、DVD1枚／価格＝2,680円＋税／発行＝技術評論社
ISBN＝978-4-7741-6798-5

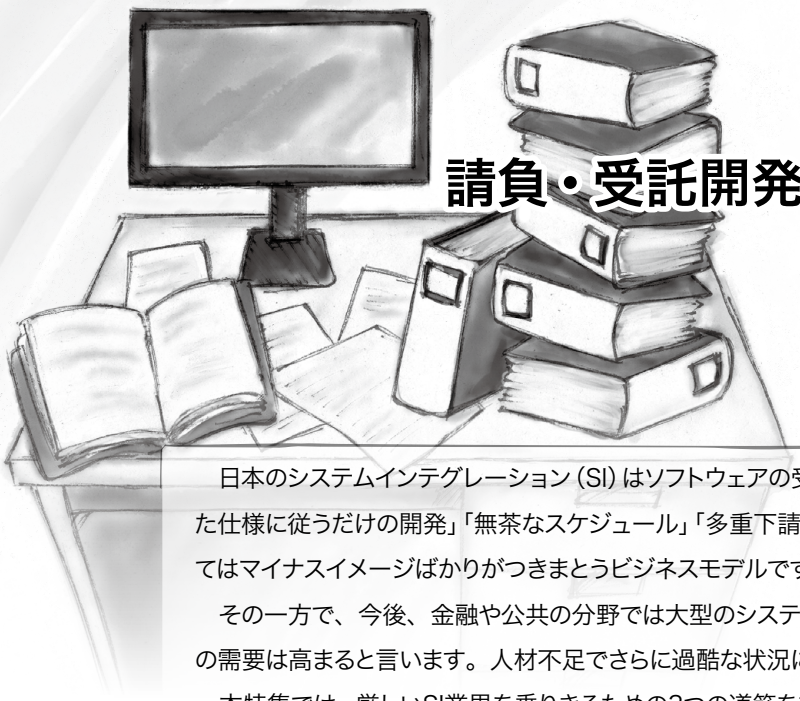
実際の新卒向けプログラミング研修をベースにしたJavaの独習本。文法や周辺技術の解説に注力した従来の入門書とは異なり、実務に役立つ「良いプログラム」を書けるようになるための方法、実践的な考え方を身につけるための内容を中心に、15時間で学べるよう段階的に扱っている。付属のDVD-ROMに収録されているVmware Playerによる仮想環境（Java 8、Eclipse 4.4）はEclipseなどの設定がすでに整っており、設定に時間をとられずにすぐ学習に入ることができる。文法のルールからテスト、リファクタリング、セキュリティの考え方まで一通りそろった本書を理解すれば、良いエンジニア／プログラマの基礎が身につくことだろう。

第2特集

SI崩壊を乗りきる3つの方法

ソフトウェア開発の未来

請負・受託開発は変わるべきか



日本のシステムインテグレーション（SI）はソフトウェアの受託開発が中心で、「決められた仕様に従うだけの開発」「無茶なスケジュール」「多重下請け」など、ITエンジニアにとってはマイナスイメージばかりがつきまとうビジネスモデルです。

その一方で、今後、金融や公共の分野では大型のシステム開発案件が控えており、SIの需要は高まると言います。人材不足でさらに過酷な状況になることも予想されます。

本特集では、厳しいSI業界を乗りきるための3つの道筋を紹介します。受託開発契約そのものを見直すか。今の受託開発スタイルの中に活路を見いだすか。いっそのことユーザ側に立ってみるか。あなたはどの道を選択しますか？

第1章

ソフトウェア受託開発の新しいビジネスモデルへの挑戦

木下 史彦(きのした ふみひこ)　㈱永和システムマネジメント
f-kinoshita@esm.co.jp　URL http://fkino.net

受託開発はITエンジニアの間ではネガティブに語られることの多いビジネスモデルです。その一方で、国内IT業界で一番多いモデルでもあります。受託開発の課題解決に目を背けていては、ITエンジニアの明るい未来はなさそうです。本章では、受託開発の課題を改善すべく、「価値創造契約」という新しいビジネスモデルに取り組んだ㈱永和システムマネジメントの事例を紹介します。

はじめに

「ソフトウェア受託開発」という言葉を聞いて、読者のみなさんはどんな印象を持たれるでしょうか？　3K、デスマーチ、オワコンといったネガティブな印象を持たれる方も多いのではないかと思います。どうしてこのような印象を持たれるようになったのでしょうか？

受託開発と自社開発

ソフトウェア受託開発(以下、受託開発)はユーザのシステムを開発会社が依頼を受けて開発します。これに対して、自社開発という形態があります。自社開発とは、自社が販売するパッケージや提供するサービスを自社内で開発する形態です。

自社開発では会社間の契約の壁がないため、プロジェクト開始後も作るものの仕様やスケジュールを柔軟に調整できます。

それに対して、受託開発では契約行為をするために、あらかじめ仕様を合意する必要が出てきます。そのため、プロジェクト開始後に仕様やスケジュールの調整が難しく、何が何でも契約時に決めた仕様どおり、スケジュールどおりに完成させなければならないという状況に陥りがちです。それがネガティブな印象の元になっているように思えます。

また、受託開発はビジネスモデルが労働集約的であり、エンジニアの稼働率を上げることが売上向上につながります。逆に言うと、作業環境の効率化やエンジニアのスキルアップのために時間を使うことは短期的には売上減になり、開発会社はそういったところに投資がしづらいというのが実状ではないでしょうか。こういったところも、受託開発に魅力がない一因になっていると考えます。

ネガティブな側面ばかり述べましたが、受託開発には次のような魅力もあります。

- 受託開発では基本的に数ヵ月～数年でプロジェクトが変わっていくため、多くの顧客(ユーザ)と出会い、人脈を広げることができる
- 1つのシステムの運用や特定の技術に縛られることなく、プロジェクトが変わるごとに新しい技術にチャレンジできる

アジャイル開発から新しいビジネスモデルへのチャレンジ

筆者は1998年にソフトウェア業界に入り、さ

まざまな現場を経験してきました。仕様が契約の範囲内かどうか、言った言わないということを繰り返しているうちに、これを続けても「誰も幸せにならない」と思うようになりました。このような誰も幸せにならない開発から脱却したいという思いを強くし、アジャイル開発への興味を持ちました。そして、2005年頃から実際にアジャイル開発に取り組むようになりました。

しかし、アジャイル開発だけですべてがうまくいくようになったかというと、そうではありませんでした。アジャイル開発で解決できた問題も多くありますが、受託開発の契約やビジネスモデルに潜んでいる問題もありました。

筆者は2010年から管理職として永和システムマネジメントのアジャイル開発を行うグループのグループ長になりましたが、労働集約的でエンジニアの稼働率に依存するビジネスモデルの限界はますます身近な課題となりました。

このような背景をふまえ、既存のビジネスモデルの延長ではなく、1つ上の段階にジャンプするために考えたのが、「価値創造契約」です。

「価値創造契約」の概要

当社では、2010年11月に「価値創造契約」をリリースしました。まずはじめに、従来型の受託開発の契約の問題点と「価値創造契約」の概要を紹介します。

従来型の受託開発契約の問題点

ソフトウェア開発の契約形態には大きく分けて次の2つがあります。

- **請負契約**
  あらかじめ定めた成果物を完成することが目的であり、成果物に対して対価が発生する
- **準委任契約**
  知識や労働力といったサービスを提供することが目的であり、提供したサービスの割合(作業した時間など)に応じて対価が発生する

これまでのソフトウェア業界での受託開発における一括請負契約では、納品時にユーザから開発会社に、システム開発費用を全額支払うというビジネスモデルをとってきました。しかし、受託開発における一括請負契約は要件定義が完了してから開発見積もり／契約するというやり方が当たり前となっており、仕様やスケジュールを固定せざるを得ず、ユーザにアジャイル開発のメリットを実感いただくのが難しいという課題がありました。

一方、準委任契約では発注者側に一方的にリスクを押し付ける形となり、発注経験の浅いユーザには採用しづらいという課題がありました。

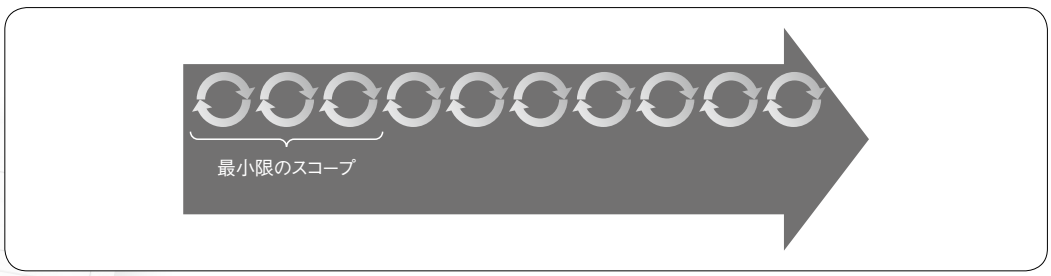
従来型の受託開発契約の創意工夫

そういった状況の中でも、当社ではさまざまな契約形態を創意工夫してやってきました。

●**パターンA　最小限のスコープを約束する請負契約**

契約段階では大まかな全体像と最小限(絶対必須)のスコープを約束します(**図1**)。最小限のス

▼図1　最小限のスコープを約束する請負契約



コープを超える部分は開発開始後に調整可能なこととします。全体像ははっきりしていないが、最低限作りたいものははっきりしている場合に有効です。

発注者側からは最低限何ができるかがクリアになるため、発注の決裁を通しやすいというメリットがあります。

●パターンB　機能ごとの請負契約

システム全体を0.5〜3人月くらいの機能に分割し、優先度の高いものから契約していきます(**図2**)。五月雨式に着手していくため、複数の契約が同時に走ります。

この契約をするうえでのポイントは、開発チームの人数をできる限り増減させず、固定の人数で対応できるように同時並行させる契約のボリュームをコントロールしていくことです。

●パターンC　短期の請負契約

1〜3ヵ月単位で成果物を規定して契約します(**図3**)。全体像ははっきりしていないが、短期的に作りたいものははっきりしている場合に有効です。

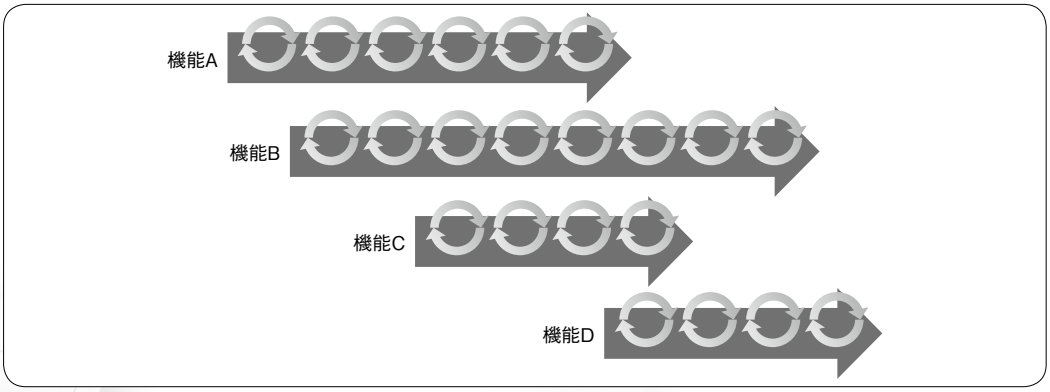
この契約をするうえでのポイントは、要件を先に固定するのではなく、先にタイムボックス(たとえば3ヵ月)を決めて、その期間に入る要件を決めていきます。

単発で終わるケースはほとんどなく、継続が基本です。

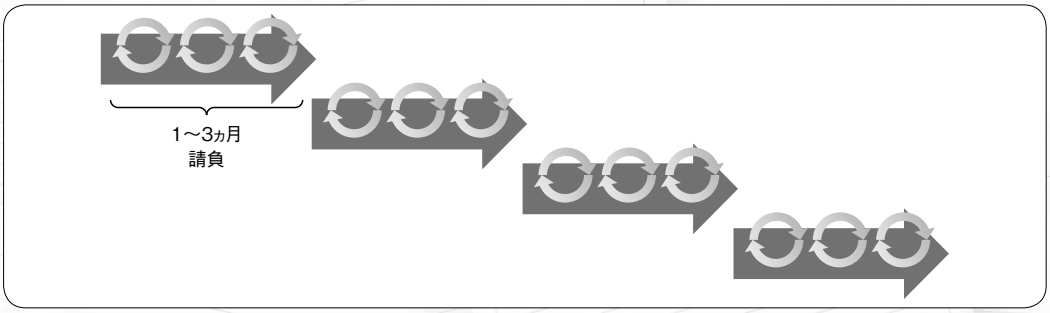
●パターンD　短期の準委任契約

1〜3ヵ月単位で準委任契約を締結します(**図4**)。たとえ1ヵ月でも成果物を明確に規定することが難しいプロジェクトに有効です。契約期間を短くすることでユーザのリスクを最小化しています。

▼図2　機能ごとの請負契約



▼図3　短期の請負契約



B2CやB2Bのサービス開発や初期開発が一通り落ち着き、メンテナンスフェーズに入ったプロジェクトで多く採用されています。随時上がってくるユーザからの要望に応えていくことが可能です。

このパターンも、単発で終わるケースはほとんどなく、継続が基本です。

◆ ◆ ◆

いずれの契約形態も実際のプロジェクトに適用し、うまくいっています。

永和システムマネジメントのビジネスの現状

図5は契約形態別に見た当社のアジャイル事業部[注1]の売上高の比率です。この中で「コンサル／教育」を除いた部分が受託開発の売上です。

全体の3分の2が準委任契約（前述の「パターンD　短期の準委任契約」）になっています。準委任契約でも、作業場所はお客さま先に常駐ではなく、お客さまのところに行ったり、自社に持ち帰ったり、プロジェクトチームの裁量で作業場所を選択できるようにしています。

請負契約も15%ありますが、ほとんどが前述のようなさまざまな契約上の工夫を行い、アジャイル開発を適用しています。

図6は当社のアジャイル事業部が受託開発を担当しているシステムの種類です。

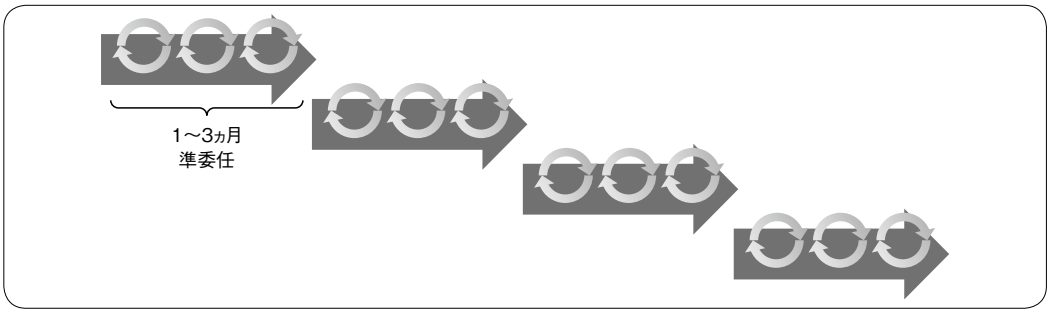
8割がB2CないしはB2Bのサービスです。このようなサービス開発はリリースしてみてユーザからのフィードバックを得て、継続的にサービスを改良していくスタイルになるため、アジャイル開発との相性が良いです。そのため、お客さまからの相談も多く、ビジネスのボリュームが大きくなっています。

図7は営業ルートの比率です。

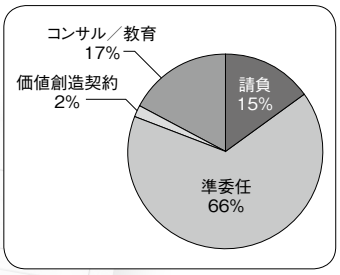
現在、アジャイル事業部で開発を担当しているプロジェクトの8割がエンジニアのコミュニティの人と人とのつながりでいただいている仕事です。ほかの2割もコーポレートサイトからのお問い合わせであったり、プライベートセミナーに参加いただいたのがご縁でお声がけいただいたり、他事業部から紹介を受けたりという

注1）当社の中でRubyとアジャイル開発手法を用いた受託ソフトウェア開発とコンサルティングを専門に担当する組織です。

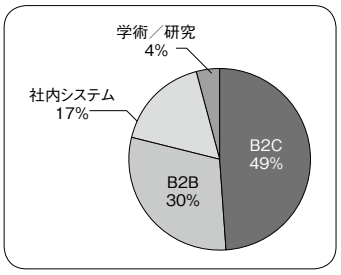
▼図4　短期の準委任契約



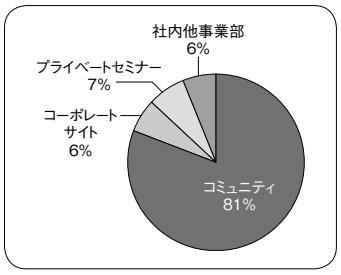
▼図5　契約形態別売上比率



▼図6　対象システム種類別売上比率



▼図7　営業ルート別売上比率



形で、100%がプル型の営業で案件を取得しています。

これらのグラフを見ると、ビジネスは非常に順調なように見えますし、実際に順調です。しかし、前述の契約形態(4つの工夫)はあくまでも既存の労働集約的なビジネスモデルの延長であり、稼働率に依存するビジネスモデルの限界から抜け出すことはできないと考えています。従来型の受託開発のビジネスモデルから脱却するための新しいチャレンジが必要だったのです。

このように筆者が考えていた折、当社は2010年に創業30周年を迎えました。創業30周年の社内イベントとして、「新規ビジネス創生事業」と名付けられた社内のインキュベーション企画が実施されることになりました。企画の応募条件は「『売上』と『コスト』が比例しないビジネスモデルであること」。B2C向けのスマートフォンアプリやB2C向けのパッケージ製品の企画などが出される中に、筆者は「価値創造契約」を提案し、採択されました。

「価値創造契約」とは

「価値創造契約」とは従来型の受託開発のビジネスモデルから脱却し、開発したシステムを初期費用0円で提供するサービスです。お客さまにはサービス利用料という形で月々の費用をお支払いいただきます。

サービスの価値は納品した瞬間ではなく、お客さまがサービスを利用しているあいだ継続的に提供されます。このことから、サービスを利用している期間に対してサービス利用料をお支払いいただく形をとっています。

「価値創造契約」と従来型の契約を比較したものが**表1**です。

お客さまにとっては、次のようなメリットがあります。

- 初期投資が不要なため、まとまった資金を調達する必要がない
- 一定量の追加開発についてはサービス利用料の範囲で対応できるため、追加開発のたびに社内決裁を通して、契約するという面倒な手続きを踏む必要がない[注2]
- 継続してメンテナンスをし続けるため、短期的にリプレイスを繰り返すことなく、システムを長く使える
- いつでも解約手数料なしで解約できる(「システムができてみたら思っていたものと違った」というリスクをユーザ企業が取る必要がない)
- レンタルという形態になるため、ユーザの会社で資産化する必要がない(バランスシートの健全化に寄与する)

「価値創造契約」のより詳細な情報は、Webサイト[注3]を参照ください。

注2) サービスチケットというしくみがあり、契約期間中、毎月一定数のチケットをお客さまに発行します。お客さまは追加開発の際にチケットを使うことができます。
注3) http://www.esm.co.jp/new-agile-contracts-service.html

▼表1　「価値創造契約」と従来型契約の比較

	価値創造契約	一括受託	SaaS	パッケージ
支払い	月額払い	納品時に一括払い	月額払い	納品時に一括払い、もしくは、月額払い
カスタマイズ可能性	オーダーメイド(ユーザごとにカスタマイズ可能)	オーダーメイド(ユーザごとにカスタマイズ可能)	可能な場合もあるが、基本的には難しい	可能な場合もあるが、基本的には難しい
顧客要望による機能追加	月額費用の中で対応、追加費用を払うことで大きな機能追加も可能	追加費用を払えば可能	基本的に未対応	基本的に未対応
保守／サポート	月額費用の中に保守／サポートも含まれる	別途、保守契約を結ぶ必要がある	月額費用の中に保守／サポートも含まれる	別途、保守契約を結ぶ必要がある
用途／分野	特殊な業務／サービス	特殊な業務／サービス	汎用的な業務／サービス	汎用的な業務／サービス

●「価値創造契約」のターゲット

B2C、B2Bのサービス開発は前述の準委任契約でうまくいく(お客さまにも満足いただける)ことはわかっていましたので、これまでアジャイル開発のメリットを実感していただくことが難しかった社内システム(とくに前述の工夫が適用できないような請負契約を採用していたような受託開発)をターゲットにすることにしました。

B2C、B2Bのサービス開発ではレベニューシェア[注4]という形態がありますが、社内システムはその性格上、そのシステムによって得られたインカムを数値化することが難しいため、シンプルに毎月定額を支払っていただくという契約形態にしました。

●「価値創造契約」の自信と覚悟

「価値創造契約」の最大の特徴は解約手数料なしでいつでも解約できる点にあります。このような契約になっているのには背景があります。

永和システムマネジメント(以下、文脈によって「私たち」と表記)ではRubyとアジャイル開発の組み合わせで直近8年間に150近くのプロジェクトを実施し、そのうち「使えない」「思っていたものと違う」という理由でリプレイスされたものは、ほとんどないという自信があります。

しかし、私たちにお声がけいただいた初めてのお客さまの提案/コンペに参加する際に、アジャイル開発の実績を示して、「これまでちゃんとやってきました」「今回もちゃんとやります」ということをアピールし、お客さまに信じていただこうとしても、どうしても限界がありました。失注したときに、少しでも一緒に開発をやらせてもらえれば私たちの良さがわかってもらえるのに、と歯がゆい思いをしたことは一度や二度ではありません。筆者はこの「ちゃんとやります」ということを契約の形で表現したかったのです。

●「価値創造契約」のポイント

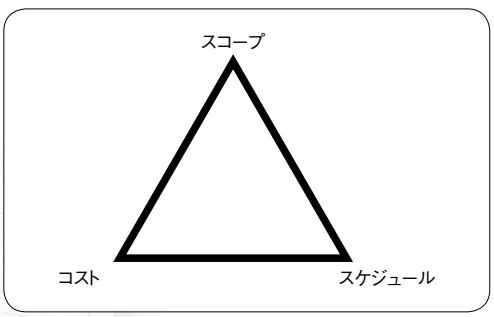
受託開発において開発を成功させるためには、発注者と受注者という関係を超えて、お互いが同じゴールを目指して協力してソフトウェア開発にあたることが不可欠です。

図8はIron Triangleと呼ばれるものです。ソフトウェア開発ではスコープ、スケジュール、コストの綱引きになります。発注者が作りたいもの(スコープ)が増えるとスケジュールやコストも増加していきます。逆にコストのキャップや決まった納期があると、スコープを一定量以下に制限しなければなりません。

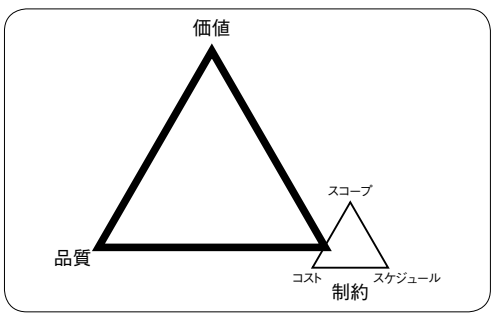
アジャイル開発のIron Triangle(Agie Iron Triangle、**図9**)では、従来のIron Triangleよりも、もう少し広い視野でソフトウェア開発をとらえています。スコープ、スケジュール、コストを1つの変数(制約)とみなし、価値、品質、制約の綱引きであると考えます。つまり、スコープ、スケジュール、コストの制約の中で、いかに価値や品質を高めるかを考えるのです。

注4) 複数企業による共同事業において、得られた利益をあらかじめ決めておいた配分率で分け合う方式。

▼図8　Iron Triangle



▼図9　Agile Iron Triangle



価値と言っても人によってとらえ方はさまざまです。B2CやB2Bのサービスであれば、アクセス数やコンバージョン率のようなKPIと呼ばれるわかりやすい指標を使って効果測定をすることができます。

しかし、「価値創造契約」がターゲットとしてる社内システムとなるとそうもいきません。社内システムはその性格的に、使ってみて効果を測定するのではなく、使う前にこのシステムを使えば業務がどれだけ効率化できるかとか、人件費がこれだけ削減できるという予測をしたうえで開発されるものです。よって、社内システムは使いながら効果を高めていくというよりも、社内業務の変更にも柔軟に対応でき、リプレイスの手間がかからず、長く使えるシステムであることが価値のあるシステムなのではないかと考えました。

開発期間中に、ユーザから要望された機能を入れるかどうかという場面で、従来の受託開発では追加で予算を取っていただくか、ほかの機能を削るかしか選択肢がありませんでした。「価値創造契約」では、要望された機能を入れることによって、ユーザ価値が高くなって、その機能を入れない場合よりもシステムを長く使ってもらえるのであれば、追加予算を取ることもほかの機能を削ることもなしに、要望された機能を作るという選択肢が出てきます。

「価値創造契約」は私たちにとっても納品して終わりではなく、ユーザに長く使っていただかないと、開発にかかったコストを回収できませんので、発注者と受注者が「長く使えるシステム」を作るという共通ゴールを持てる点が最大のポイントであると言えます。

「価値創造契約」の現状

2010年11月の「価値創造契約」のニュースリリース後に続けて問い合わせをいただき、合計3件の案件を受注しました(うち1件はすでに解約)。

お客さまからは次の点を褒めていただきました。

- 初期費用0円、月払いのサービス利用料、また使用開始後も利用年数によらず解約自由という、クライアント側としてはリスクの少ないサービス形態
- 「納得のいくシステムをできるだけ長く使ってほしい」という姿勢に共感した
- 開発開始までの着手期間が短く、開発の進捗状況を動くシステムで確認でき、要望の追加や修正を途中でも受け入れてくれる開発手法にメリットがある
- サービス利用料に見合う形で開発することが初めにはっきりしていたため、要望がその枠内に収まることを確認した時点で、ある程度安心できた
- できあがったシステムは自社内で使用する以外に使い道はなく、よく考えると所有していることにあまり意味がない。解約時にデータしか残らないが、逆にデータさえあれば次のシステムに移行する場合でも何とかなるだろうという思いで割り切ることができた

このようにお客さまから評価いただけたところがある[注5]反面、実際に「価値創造契約」を提案したり、サービスを提供したりしていく中で、さまざまな失敗を経験し、お客さまから厳しい評価をいただくこともありました。

また、2010年の「価値創造契約」リリース後は多くの引き合いをいただきましたが、数ヵ月でお問い合わせが落ち着いたため、プル型の営業スタイルをプッシュ型に変更しました。前述したように、私たちは基本的にプル型の営業で仕事をいただいてきたため、プッシュ型の営業というのは慣れない試みでした。2013年から2014年にかけて産業交流展への出展と約800社へのテレアポを行い、12社とコンタクトを取りました。そのうち、2社から具体的な案件の相談を

注5) http://www.esm.co.jp/new-agile-contracts-service/576/573.html

受けました。しかし、目立った成果はなく、受注に至ることができませんでした。このような状況からさまざまな課題が見えてきました。

「価値創造契約」の問題[注6]

サービス利用検討段階

①「いつでも解約できます」はメリットにならない

システム開発プロジェクトの6割は失敗しているか何らかの課題を抱えているというレポート(**図10**)があり、失敗したときに解約できることはお客さまにとってメリットになると考えました。しかし、実際にお客さま(ユーザ企業)と会って話をすると「失敗する前提でシステム開発をしない」ため解約はメリットとして社内で説明できないという声が多く聞かれました。

解約すれば金銭的な損害は受けないかもしれませんが、再び、別の会社に発注してシステムを作り直さなければならない事態になることを軽視していたことが反省点です。

ユーザ企業の情報システム部や担当者は、たとえ失敗したとしても失敗を認めることが許されないといった声も聞かれました。

②サービス利用料の設定が高い

「価値創造契約」は初期費用がかからないことがウリの1つです。よって、初期費用を準備できないような中小企業がターゲットになってきます。しかし、もっとも低価格のプランでもサービス利用料として月々15万円の費用がかかります。これはターゲットユーザ(初期費用の準備できないような中小企業)の手に届かないような価格設定であることがわかりました。

「初期費用が準備できないような会社はそもそもシステム開発なんて考えない。業務システムの開発は儲かっている会社がすることだ」という声すらいただいたことがあります。

③怪しいと言われる

お客さまのメリットとして「初期投資不要」「解約手数料なし」を強調してきましたが、私たちの自信と覚悟、それからリスクとそのリスクをどのように担保しているかを説明してきませんでした。そのため怪しいサービスという印象を持たれてしまうことがありました。

本記事の「『価値創造契約』の自信と覚悟」に書いたようなことをもっとお客さまに知っていただく必要があったのです。

初期開発コストを当社が回収する前に、途中解約された場合のリスクヘッジはどのようにしているのかと思われるかもしれません。前述のとおり「価値創造契約」は「新規ビジネス創生事業」という当社内のインキュベーション企画から始まったものです。この公募で「価値創造契約」が採択され、途中解約のリスクヘッジ(担保)として、1千万円を獲得しました(会社が1千万円を用意しました)。

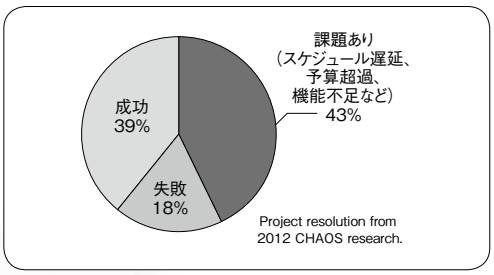
サービス提供開始後

④社内のシステム化方針の変更による解約

「価値創造契約」で開発したシステムをリリースした直後、経営者(CIO)が交代し、「価値創造

注6) これ以降は「XP祭り2014」で発表した失敗事例を再構成し、紹介します。
XP祭り2014　http://xpjug.com/xp2014/
発表資料「俺の価値創造契約」　http://www.slideshare.net/fkino/ore-no-new-agile-contracts-in-action

▼図10　システム開発プロジェクトの成功／失敗の割合[注7]



注7) 調査資料の出典：The Standish Group International, Incorporated. "CHAOS MANIFESTO 2013",p1　http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf

契約」で運用しているシステムをERPに入れ替える方針となったことにより解約に至ったケースがあります。

お客さまの担当者とは少なくとも3年程度は運用する前提でスタートし、ユーザ(システムの実際の利用者)にも高い評価をいただいたのですが、経営者の交代は想定していませんでした。解約には「価値創造契約」の規定どおり、手数料なしで応じました。

ユーザが満足するものを作れば長く使ってもらえるという考えは、作る側の勝手な思い込みであることを知らされました。

⑤チケット使用に承認が必要なため、あまり使われない

「価値創造契約」では一定量の追加開発についてはチケットの範囲で対応できるため、追加開発のたびに社内決裁を通していただくという面倒な手続きが必要ないというメリットを挙げています。しかし、実際にやってみると、チケットの使用にお客さまの社内決裁が必要となり、追加開発のために予算を確保する場合と同じフローになるという状況が発生しました。

⑥「最終的な仕様は開発会社が決める」は受け入れられにくい

「価値創造契約」では開発スコープをお客さまと合意したサービス利用料のプランの範囲内に収めるために、最終的な仕様は開発会社が決めるとしています。しかし、対象としているシステム(お客さまの社内システム)の性質上、お客さまに判断を仰ぐ場面が多くなっているのが実状です。仮に開発会社で仕様を判断したとしても、使いものにならないシステムができあがってしまいます。

また、仕様は開発会社で決めると言っておきながら、知っていて当たり前の業務知識についてまったく不勉強でした。知識／勉強不足により、お客さまに確認すべきことが確認できていないことがありました。

⑦お客さまに理解いただけるドキュメントの不足

「価値創造契約」で開発したシステムは納品せず、当社の資産になるため、当社の開発者がわかるレベルの仕様書しか残してきませんでした。開発中は良かったのですが、運用が始まり、チケットを使ってシステムをより良いものにしていこうという段階になって、お客さまと共通認識を持つために必要な情報が不足しました。

上記と同様の理由で、お客さまが自分たちのシステムとして運用していくために、必要な情報が不足しました。お客さまは、自分たちの資産ではない(レンタルのような形態)とはいえ、システムを運用していく中で発生する課題を、「自分たちのシステム」として可能な限り自ら解決して使いこなしていきたいと考えていることがわかりました。しかし、お客さまに提供できるような資料を作っていませんでした。

⑧担当者変更に伴う進め方、考え方の擦り合わせの難しさ

「価値創造契約」はこれまでにない新しい契約であるため、途中でお客さまのシステム担当者が交代した場合、新しい担当者から理解を得るのが難しいです。

開発時は新しいものが段階的にできていく姿を見せながらお客さまの要望を取り入れていくため、加点法で評価していただけます。しかし、運用に入ったシステムは障害発生や過去のシステムと比較して使いづらくなったなど、減点法で評価されることが多いです。システムが運用に入ってからお客さまのシステム担当者が交代した場合、信頼を得るのが難しいことを実感しました。

⑨システム利用料に対する成果を求められる

「価値創造契約」ではシステムを初期費用0円で提供し、月々のサービス利用料という形で対価をいただいています。しかし、エンジニアの稼働に対する対価を支払うという考え方が抜け

きらず、システムの運用に入ってから、対価に対してエンジニアの稼働が少ないという不満を抱かれるケースがありました。

⑩仕様確定や決裁に時間がかかるとチケットが失効する

チケットを使った対応を行う際に、仕様確定や決裁に時間がかかって開発開始が遅れたことによって、チケットが失効したことがありました(チケットの有効期限は1年)。

大きめの要望が発生したときには、仕様確定や決裁に時間がかかりチケットが失効する可能性が出てくるため、お客さまには要望を細かく区切って段階的な対応をお願いしました。しかし、段階的な対応を取った場合、受入テストや移行作業、システムの利用者への説明が何度も必要になり、お客さまに負担をかけるため、敬遠されることがありました。

「価値創造契約」の教訓

このような失敗を通して「価値創造契約」から多くのことを学びました。課題は大きく次の3つに分類されます。

(1)ニーズとサービスのミスマッチ
(2)ユーザから新しいサービスへの理解を得ることの難しさ
(3)私たちのレベルアップの必要性

このうち、(1)の「ニーズとサービスのミスマッチ」が一番の失敗だったととらえています。当社のビジネス上の制約(解約されない長く使われるシステムをターゲットとすること)と「価値創造契約」に魅力を感じるターゲット(スタートアップのような事業)がミスマッチであり、受注につながっていない、つまりスケールしていないということです。前述のように「価値創造契約」の売上比率は全体のわずか2%です。

(2)、(3)については「価値創造契約」に限定した話ではないと考えています。これらについては、これまでの経験を糧にすでに改善を実施しています。

なお、収益面では受託した3案件のトータルで黒字になっています(途中解約された案件は赤字ですが、3案件トータルすると黒字です)。従来の受託開発の案件と比較しても、遜色のない利益を上げられるビジネスモデルであることは実証できたのですが、スケールしないことが課題であると言えます。

「価値創造契約」のこれから

サービス自体の見直しとコンセプトを守ること

「『価値創造契約』の教訓」に記載したとおり、「価値創造契約」にはニーズとサービスのミスマッチが発生しています。ニーズを見極め、サービス自体を改良していく必要があります。

サービスの改良を考えるうえで、「価値創造契約」は受託開発に対する私たちの姿勢や覚悟を示しており、「長く使い続けられるシステムを育てていくことが、お客さまとサービス提供者である私たち双方にとっての価値であり、そういった価値をお客さまと私たちが共同で創り出していく」というコンセプトをぶらさないことが重要であると考えています。広く売れるよりも、私たちのやり方に共感していただき、本当にサービスを必要としている人(これまで請負契約を採用するしかなく、アジャイル開発のメリットを実感いただけなかったお客さま)に利用いただけるサービスにします。

そして、受託開発でストックビジネス[注8]を作っていきたいと思います。稼働率を上げることでしか売上の向上がない労働集約的なビジネスモデルから脱却し、エンジニアがクリエイティブな活動により多くの時間を使えるようにしたいと考えています。

注8) 電気料金、電話の通話料金、Webサービスの月額利用料などのように、一度作ったしくみを利用して継続的に利益を得るビジネスモデル。

サービスづくりよりもファンづくり

ふりかえってみると、「お客さまに価値を提供したい」という思いの発端には、「アジャイル開発で」という手段へのこだわりがありました。開発をアジャイルにしたい、Rubyを使いたい、ストックビジネスを作りたいという自分たちのやりたいこと、思いからスタートしていました。「価値創造契約」をマーケットイン[注9]で確実に売れるサービスにするというやり方もありますが、それによって自分たちのやりたいことができなくなったのでは意味がありません。

そして、「価値創造契約」のような新しいチャレンジをする際には、どんなチーム(開発者だけではなく、お客さまも含めたチーム)で対応するのかが重要になってきます。したがって、一番大事なことは自分たちのやり方やコンセプトに共感してもらえるお客さまと出会うことだと思います。

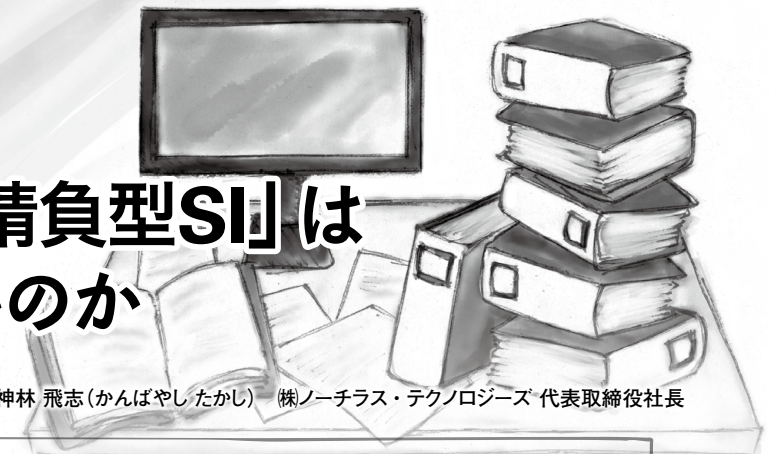
失敗事例を「XP祭り2014」で初めて発表したとき、「ここまで公表してしまっていいのか」という反応をいただきました。アジャイル事業部のビジネスは私たちにお声がけいただくお客さまの存在に支えられています。「永和システムマネジメントのビジネスの現状」にも記載したように、そのお客さまの8割がエンジニアのコミュニティからのつながりです。「価値創造契約」の事例や「価値創造契約」を通して得た経験/ノウハウをコミュニティの場で発表することで、支えていただいているコミュニティの人たちに現状を報告し、受託開発のこれからを考えるうえで議論の種になればという思いで、発表をしました。

今後もこのような事例や経験/ノウハウを発表していくことで、口コミや人のつながりを広げ、「価値創造契約」を必要としているお客さまに出会うことを目指していきます。SD

注9) なによりも顧客ニーズを優先し、そのニーズに応えるための製品開発を行うこと。

第2章

なぜ「受注請負型SI」はなくならないのか



神林 飛志（かんばやし たかし）　㈱ノーチラス・テクノロジーズ 代表取締役社長

日本のIT業界において、受注請負型SIはなくならない―― その理由は、人月商売型のビジネスの悪循環に陥っているからといえるでしょう。しかし悪循環に依存しながら抜け出せないのが、今のSI業界です。そんな状況が変化しないならば、いっそのこと撤収すべきか否か。SI屋稼業を貫くための処方を最後に示します。

ノーチラス・テクノロジーズという会社

小職の所属するノーチラス・テクノロジーズという会社は、実質的な設立から3年を経過した独立ベンチャーです。もともとは㈱EC-ONEのクラウド事業部とウルシステムズ㈱のミドルウェア事業部が本体から分離独立し合弁した会社に、Hadoopなどの分散処理に興味があるメンバーが参加して出来上がった会社です。現在は、Hadoop上で業務系のバッチ処理を開発・運用するためのAsakusaというフレームワークを開発、OSS（Open Source Software）として公開し、そのサポートライセンス・コンサルティング・インテグレーションを生業（なりわい）としています（**図1**）。

ミドルウェアを売るという商売とSI

現在、当社はAsakusaを軸にしたビジネスを展開していますが、その一方で、前身であるウルシステムズのミドルウェア事業部からの事業も引き継いで、特定業務向けのミドルウェアの販売・運用も行っています。結果としてミドルウェアの開発・販売を、ウルシステムズ時代から足かけ10年にわたって行ってきました。この一方で、当社はシステム構築そのものへも強い“こだわり”をもってビジネスを展開しています。すなわち、「**コア技術を持つためにはシステムの開発・構築は小なりといえども必ず行う**」という方針です。

通常、ミドルウェアを専業にしている会社はシステム構築は行いません。これは、何かのはずみで過剰にシステム構築を請け負った場合、会社のリソースが構築の仕事に専有されてしまい、本業であるミドルウェアの開発が滞ってしまうからです。

▼図1　Asakusa Framework（http://www.asakusafw.com/）

当社で、そのようなリスクを背負ってまでSIを仕事として受けるというのは、大きくは次の2つの理由からです。

①自分の作っているものは自分でちゃんと食べる

ミドルウェアに10年近くかかわった経験からわかったことは、ユーザやSI屋さんから当社で販売しているプロダクトへの適切なフィードバックを受けることは、極めて難しいということです。もちろん、機能要求やパフォーマンス要求、バグ報告や使い方への質問はどんどん来ます。**「実際に触っている人間がシステムの構築時点でなんとなく感じる違和感」**は言語化がなかなか困難な場合が多く、ミドルウェアへのフィードバックとして受け取ることは大きな課題です。しかし、自社で実際の製品を使って構築をしてみると、その製品の隠れた欠点や**「微妙な違和感」**が見つかりやすく、対応策を次の製品の機能として取り込んだり、製品の今後の方向性として検討できます。その意味で自社製品を利用したシステム構築を自分たちのリソースで行うことは、非常に重要と考えています。

②食べていけることが非常に重要

エンタープライズでのミドルウェアの販売では、長期のサポートを行えるという体制が非常に大事です。そのミドルウェアが長期にわたり使えるかどうか?――はユーザ企業がミドルウェア製品を選択するときの基準の1つです。このため形はどうであれ、製品としてサポートを長期間提供し続けるということを、ミドルウェアベンダーは行う必要があります。加えて、日本のエンタープライズ市場は、販売しているソフトウェアを使ってもらうためにかかる新規参入の時間的なコストが非常に高い。これはソフトウェアを利用してもらうために、SI屋さんとユーザという大きな2種類のステーク・ホルダーに納得してもらうことが必要なためです。とくに独立ベンダーのミドルウェアであれば、エンドユーザへのシステムの新規導入までは普通に数年はかかります。つまり日本でミドルウェアをちゃんと商売にしていくには、立ち上がりにしろ、そのあとの展開にしろ、かなり時間を稼ぐ必要があります。

さらに、仮にうまくビジネスとして立ち上がったとしても、そもそも日本でのミドルウェアのビジネスは、DBといったミドルウェアの本流がグローバルソフトウェアベンダーに席巻されているため、どうしてもニッチな部分に限定されがちです。このようにマーケットが狭いうえに、ソフトウェアに高い金額を払うというカルチャーが日本企業にはどうしても薄いため、結果、日本でのミドルウェアの商売はかなり厳しいのが現実です。

ミドルウェアは儲からない――これは日本の実態でしょう。現にミドルウェア本業だけで食えている独立ITベンダーは数えるほどです。当社も例外ではありません。たしかに既存のミドルウェアでの商売があったため、純粋にゼロからスタートするベンダーよりは圧倒的に有利ではありました。とはいえ、十分ではありません。ある程度の糊口をしのぐために、ある程度の構築の仕事を受ける――ということは選択として視野に入れて、実際にシステムのコアな部分の実装は請け負っています。

ただし、SI屋にならないために歯止めを設けるべき

SIの仕事は簡単に増えますし、売上を上げることができます。その結果、多くのソフトウェアベンダーや有名なWebサービスの会社が、食うに困ってシステム構築に手を出しました。結果は、本業のソフトウェアの消滅やサービスの新規開発ができない状態になってしまっています。ミドルウェアベンダーが構築の仕事をするときには、かならず歯止めを設けるべきです。当社では「規模」と「内容」と「顧客」について、制限を設けています。

その制限とは、構築の金額・難易度・見通し

については自社で制御できる範囲とすること、内容はあくまでAsakusaにかかわるものであること、お客さんは自社のメンバーにちゃんとした敬意を払ってくれる会社であることを、システム構築の仕事を受けるときの目安にしています。また、社内でも常にSIのあるべき姿、自分たちのかかわり方を常に議論しています。

AsakusaというOSS

さて、当社が本業として提供しているAsakusaですが、Hadoopでの業務系バッチ処理を記述するOSSです。これはHadoopのエコシステムの中では、“SI(System Integration)”を非常に意識しているという点で、かなり特異な存在になっています。通常、Hadoopのようないわゆるビッグデータ系のフレームワークは、SI、とくに大規模SIはあまり意識していません。データサイエンティストなどの少数の人員が分析の道具として利用することを想定しているからです。このようなフレームワークと異なり、Asakusaは、普通の企業の一般的な業務バッチ処理の開発、すなわちSIの対象となるような業務処理を開発することを目標とし、SIで必要になる、テスト機構、開発手法、フール・プルーフを織り込んだDSL(Domain Specific Language)を提供しています。

なぜAsakusaがSIを意識しているかというと、これは筆者たち自身の経験やキャリアが深く影響しています。筆者はもともとユーザ企業出身で、ITのキャリアのスタートはユーザ企業でのシステムの内製化から始めました。当時のIT業界は汎用機からオープン化への移行が進んでいる時期で、ちょうどJavaが出始めたころでした。勤務していたユーザ企業は、大手のSIベンダーに依存する状態で、コストの高止まりの中、ガバナンスの確保と次世代へのオープン系への基盤の移行が急務でした。そこでコアシステムの内製化を進めつつ、できない部分はベンダーのリプレースで対応するという方針で臨みました。結果として、ユーザ企業におけるITの位置づけというものを、内製化・外部委託としてSIの両面から再検討するというバランスの難しい仕事を経験することになりました。

Asakusaの誕生

このときの筆者の経験がAsakusaの基本思想に大きく影響しています。Asakusaはユーザがシステムの開発に積極的にかかわるツールとしての一面も持っています。「ユーザは基本設計は可能であれば行ったほうがよい、無理であっても少なくともレビューは必ず行うべき」という考え方を採用しています。そのため、Asakusaは、ユーザとSI屋の接点として設計を重要視し、設計指向のフレームワークになっています。**設計をちゃんとやらない限り実装ができない**、そういう性格を持っています。

次に筆者が参加した企業は、ウルシステムズという会社でした。現在はITコンサルティング会社として運営されていますが、自分が加わったスタート当初は、実態としてはいわゆる「土方SI屋」でした。徹夜続きのデス・マーチSIもご多分に漏れず、ガッチリ行っていました。土方のデスマSIは、かかわるSEの能力を著しく減衰させ、さらにプロジェクト運用の段取りの悪さが個々人への負担を倍増させていきます。Asakusaがテストまわりを充実させていたり、フール・プルーフのしくみを導入しているのは、このような経験に基づくものです。可能な限り現場の負担を減らす、ということを主眼の1つにしています。

なお、ウルシステムズの上場のためには「ソフトウェア」が必要だ、ということで、自分たちのチームが編成され、特定業務向けのミドルウェアを開発・販売するということになりました。これがそもそも今の当社の母体になります。このようなスタート地点を持っているため、参加メンバーはもれなくSI経験者ということになります。

たしかに現在の当社は、Asakusaの提供とい

う点で言えば、一ツールベンダーの立ち位置ですが、その出自はユーザ・ベンダーの双方の立場から、とくにSIに深くかかわっています。これが結果として、AsakusaというOSSがSIを軸に考えたしくみを持ち、それが基本になっている背景になっています。

SIビジネスのあり方

以上、筆者たちの経験をふまえたうえで、現在のSIが「どういう意味」を持つかということを、筆者なりに描いていきます。現状のSIの問題点、今後の方法、今話題の「新型SI」の課題をスケッチしていきましょう。

SI市場の現状

まず現在のSI市場というものを、俯瞰してみましょう。現在のSI市場では、ベンダーが提供すべき技術の高難度化が進む一方になっています。プログラミング言語や習得すべきフレームワークの多様化・複雑化は止まることを知りません。これはハードウェアが低コストで高いパフォーマンスを叩き出し、多少の無理な処理でも力技で解決してしまっていることも一因です。結果、一昔前であればパフォーマンスから見ると非現実的だった、言語・フレームワークも十分利用に耐えるものとなってしまっています。これに加え、最近急速に一般化しつつあるクラウド技術はインフラの多様化をもたらし、さらに分散処理のような、従来であればスーパーコンピュータの技術に分類されるものまで、一般に利用できるようになっています。結果として、習得すべきIT技術は拡大化の一途です。

また、SI市場への良質な人員の供給がボトルネックになり、今はさらに顕著になってきています。現在は、どこのSIプロジェクトでも人手が足りません。これは、ベンダーサイドが景気の停滞時に人員採用を絞ったツケです。しかも稼働率優先で人の移動を優先させた結果が相まって、設計・PM・繊細な部分の実装ができる人員が、従来にも増して不足しています。マクロの視点でみれば、日本全体での若年就労人口の大幅な減少も、SI市場のような労働集約的な市場への人員不足に大きな影を落としています。

これに加えて、ユーザサイドのIT人員への投資の長年にわたる低減が、ユーザサイドで細かい要件を決めきれる人材の不足へとつながっています。ここ10数年の「持たざる経営」のトレンドはバックエンドの人員の削減を大きく進めました。ユーザ企業での、自社の業務内容を正確に把握し、システム化要件に落とし込める人材は姿を消しつつあります。

この結果として近年の大規模SIの失敗率は、かなり上がってきています。複雑さが増し、仕様がわかり、適切な判断をくだせる人間が減少すれば、これは当然の結果です。大型案件の失敗をめぐる訴訟が大きく取り上げられるようになって久しいです[注1]。実際に、我々のまわりに見える大型案件も、従来よりも失敗するケースが多くなっています。

受注請負型SIはなくなるのか?

このような現状を一見すると、受注請負型SIはもはや成立しないように見えます。では、このような受注請負型のSIは、今後なくなるのでしょうか?——結論から言うとNOです。受注請負型のSIは今後、かなりの市場として残るでしょう。その理由はSI市場でのSI屋サイドとユーザサイドの各プレーヤのマーケット維持のモチベーションが強いこと、要するに「SI屋はSIが売りたいし欲しいし、ユーザも受注請負型のSIが欲しい」というに尽きます。

まずSI屋サイドから見ましょう。IT稼業で「まず売上を上げる」とするのであれば、人月商売の営業をすることが圧倒的に近道です。もっとも売りづらい業務系ソフトウェアと比較すると、体感的には人月商売のSIビジネスのほう

注1) 日本IBMに野村が33億円賠償請求　なぜ訴訟相次ぐ(http://www.nikkei.com/article/DGXMZO77218190Y4A910C1000000/)

が、営業効率でざっくりと10〜50倍は高いです。これはSIビジネスのほうがソフトウェアなどの販売に比べて、価格根拠が説明しやすいということに尽きます。この機能は5人がかりで2ヵ月かかります、という説明は、そのソフトウェアの価値はだいたい1,000万円ですという説明よりは、残念ながら圧倒的に説明しやすい。また、以前はハードウェアも売りやすい商材の1つではありましたが、値段の下落・ハードウェアのコモディティ化・クラウドサービスの登場もあり、大きな数字を簡単に上げるのはなかなか難しくなりました。よって現在のところ、数字が欲しいITベンダーの営業は、人月商売のSIを仕事として普通に取りにいきます。

◆ ◆ ◆

次に、ユーザサイドを見てみましょう。ユーザにおいては、「受注請負型SI」は極論すると「いくら金がかかってもいいから絶対にやってくれ」というスタンスに近いです。このニーズは非常に強力です。

なぜユーザは「受注請負型SI」を必要とするのか

ユーザが「受注請負型SI」の依存から抜け出せない大きな原因が2つあります。

●原因その1「ユーザの意思決定プロセスの問題」

最終的にユーザサイドでIT投資を決定するのは情報システム部ではなく、その上位の経営陣です。ユーザ企業においてはとくにIT案件は、たいていの場合「大型の投資」になります。その意思決定はユーザ企業の経営陣が可否を判断することになります。そして、通常その経営陣はITの専門家ではありません。

ユーザ企業の経営陣にとって、多くの場合でITは必要悪以外の何者でもありません。一般企業においては、別にITに投資をしても売上は上がりません——驚くなかれ、いわゆるWeb系でも実は同じです。もちろん例外はありますが——そんなお金があれば、売上に直結する人員／設備／プロモーションに投資します。短期で見れば当たり前のことです。ただITは「業務をまわす」うえでは必要不可欠なところはあるので、可能な限り最小限にしたいというのが普通の感覚です。結果、役員各位はおっしゃられるわけです。「それで、結局いくらかかるの？　この金食い虫は？」……これが普通です。

さらにユーザ企業の経営陣にとって、ITはブラックボックス以外の何者でもありません。

まずもって「いや、最近のITは複雑だからわからんよ」「いや？……さすがにITはよくわかりませんわ」と言わない経営者には会ったことがありません。実際、自社のシステムの細かいところまでわかっている経営者は絶無でしょう。前述のように最近のIT技術の多層化・複雑化・多様化は拡大の一方です。IT本業の我々ですら、すべてを知っているということはありません。いわんや専門家でもない経営陣にとっても、むべなるかなです。

上記の2つを合わせるとどうなるかというと、非常に簡単で、まともな経営者であれば、こんな状況では確定しない金額でのIT投資の意思決定を行うことはできません。

「いや、だからいくらかかるかはっきりさせろ、こっちも予算があるし、そのうえで削れるものは削れ、金は無限にあるわけないでしょ」。

似たようなことを、情報システムの部長であれば、役員会で何度も言われているでしょう。挙げ句に「**プロジェクトが失敗した場合のリスクは、ベンダーではなくユーザさんにありますよ**」という委任型の請負は完全に却下されます。「請負・金額確定型」がユーザサイドの「経営陣」にとっては、絶対に必要なのです。

●原因その2「IT人員の不足の補完としてのSIの役割」

受注請負型SIは、基本的にいわゆる「**丸投げ**」というスタイルになりがちです。なるほど、IT投資の透明性の確保やITガバナンスの維持、ひいては失敗リスクの高いSIをやめるために、丸

投げをやめて、内製化に進む動きが一部のユーザ企業に見られるようになりました。しかし、うまく行っているところは少数にとどまっています。大きな理由は「**人が採用できない**」ということに尽きます。

背景は日本のITへの従事者の企業間、とくにユーザ企業とITベンダー間の人の移動の少なさです。コアな業務システムを構築できる人間がユーザ企業に転職したり、またユーザ企業で業務の設計・運用ができる人間がSI屋に移籍したりすることは、現状の日本のIT関係者の労働市場では極めて少数です。これは仮に転職できたとしても、転職後のキャリアパスが、それぞれの転職先でまったく考慮されていない、ということが大きな理由です。

ITベンダーからユーザ企業に移った場合、基本的にはその企業のIT部門に配置されるわけですが、そのあとの昇進・出世ルートは基本的に準備されていません。理由は単純で、普通の企業においてITは"**金をかせぐ花形部署**"ではなく"**地味で金ばかり食うバックエンド部署**"だからです。よって出世ルートからは原則外れます。例外的にエリート的な人員が配置されることもありますが、これは出世が約束された人間が「一応、形だけでもITも知っておけよ」という人事の配慮で、腰掛で異動されるだけです。このような部署に"**外から**"転職してきたところで、昇進ルートは制限されます。

逆にユーザからITベンダーに移った場合、やはりその能力不足が顕著です。ITベンダー稼業は、やはり基本はSI・システム構築です。したがって、PM・実装・設計能力がどうしても必要です。ユーザ企業の情報システム部にいると、ベンダーへの発注能力は磨かれますが、実装や設計(とくに詳細設計以降)といった手足を動かす能力はどうしても磨かれません。したがって、ITベンダーに移籍したとしても、実働部隊の戦力になかなかなりません。

要するに、人員の交流が現実的にはほとんど行われていないのが現状ですし、このようなそれぞれの組織的な問題があるため、今後も交流する見通しは少ないでしょう。この「人の移動の少なさ」は、ユーザ企業においては、必要なときにITの人員を確保するということも事実上不可能にしています。

いや、最近はSI屋からWeb系ユーザ企業への人材の移動がトレンドだろ——という声も聞こえますが、現実には、業務設計をシステム計画に落とし込めて、SIのPMやアーキテクチャ設計ができるうえに、必要な場合は自分の手足を動かすことを厭わないというコアの人材はまったく動いていません。その証拠に、たいていのWeb系企業の企業運営に必要なバックエンドのクリティカルなシステムは、内製どころか、SI屋さんがデスマで作っているのが実態です。Web系ユーザ企業が採用している、元SIな人材では開発することはできていません。さらにそもそもWeb系企業では転職組に対して、既存SI屋で準備されている以上の明快なキャリアパスは準備していません。結果、次々と転職を繰り返す——いわゆるワンダーフォーゲル化になります。人材の移動は失敗していると言わざる得ないでしょう。

現状では「システム構築のプロ」をユーザ企業が独自に確保することは、非常に困難です。ではどういう対処があるのか？　ということで、結局は「SIの丸投げ」という選択をせざるを得ません。これは一種の脱出できない、人員のアウトソーシングへのロックインになってしまっています。

では「受注請負型SI」はなくならないのか?

上記のような底堅いユーザニーズと、数字が欲しいIT企業の営業スタンスが強固にある限り、SI市場は均衡し、大きなマーケットとして存在し続けます。我がSIは永遠に不滅なわけです。

しかしながら、前述のように、SIをめぐる環境は悪化する一方です。このままでは、マーケッ

トは存在するが、個々の取引がうまくいかず、結果としてハードランディングするということになりかねません。個々のSIの結果リスクは増大し、大型案件は常に火を噴くことになるでしょう。SI失敗リスクの顕在化の最たるものは、ユーザ・ベンダーが相互にいがみ合い、最後は感情論になってしまう訴訟合戦です。もうその萌芽はそこかしこで見て取れます。こうなってくると、そもそも開発の開始前の契約条項の交渉が無意味にハードになっていきます。またリスク回避のためだけの作業やドキュメンテーションも増大します。SIのオーバーヘッドがさらに上がります。うまく行かないものが、ますますうまく行かなくなります。

◆ ◆ ◆

さて、では、この「SI」というリスク商品に今後我々IT屋がどうかかわっていくか？――ということが問題ですが、基本的には次の2つの方策しかないでしょう。すなわち、**やめるか？続けるか？**です。

いっそのことSIから撤収する

これは悪くはないアイデアです。そもそもSIから撤収しても、我々ベンダーサイドは結局、なんとかなってしまうような気がします。困るのはむしろユーザサイドでしょう。そもそもユーザ事情で成立している側面が強いマーケットです。SIマーケットを維持するにはユーザサイドの働きかけも必要でしょう。そのようなスタンスがないようであれば、マーケットの崩壊というハードランディングもやむなし、でしょう。

SIを止めることによるベンダーサイドのデメリットは非常にシンプルで、要するに食えなくなるということです。これは解決案は2つしかありません。食う量を減らすか、別に食い扶持を探すかです。

SI屋さんの経営陣としては、要するに人を減らすというリストラが選択肢になります。むろん、別のサービスを見つけて、そちらに人員を異動させるという方法もあるとは思いますが、とはいえ、SIの売り上げに代わる数字をたたき出すサービスというものはなかなか見つけ出せないのが実情です。見つけていたらとっくの昔にSIから撤退しているでしょう。しかたなくSIをやっているとはいえ、現実にかかわるSIが訴訟多数・不採算案件が乱立という状態になれば、事業的には撤収または事業縮小という形で選択せざるを得ないでしょう。

また、実際にSIをやっている人間から見れば、これは転職ということになります。デスマ確定・いがみあい確定の仕事に人生の時間を費やす必要はありません。IT以外にもあなたを必要とする職場はいくらでもあります。多少ネットワーク技術の知識があって、ちょこっとしたSQLが書けて、Excelを手足のごとく使いこなせるのであれば、どこでも食えます。鬱病にかかるリスクも減るでしょう。個人的には超お勧めではあります。

SIでがんばる

ニーズはある。ただし提供する商品が爆発するリスクが高い。ということであれば、やはり商品を「変えていく」ということは有効な手段だと言えるでしょう。「従来型のSIはもはやうまくいかない。時代は価値創造型契約だ、納品のないSIだ」というメッセージとともに、新しい形のSIの手法が模索・実践されていますが、これらもSIマーケットから見れば、その1つと言えるでしょう。ただ、私見ですが、このような**“新型SI”**（価値創造型契約・納品のないSI）な方々が完全に見逃している観点は、ユーザの意思決定プロセスです。「受注請負」という性格がユーザニーズそのものの根幹にあるのであれば、商品を変えても、ユーザのニーズは満たすことはないため、結果として普及は困難でしょう。

仮にこのような**“新型SI”**をSI市場でちゃんと成立させるには、受注請負型SI以上に、まずITベンダーの営業が売りやすい商材に仕立て上げる必要があることと、ユーザの意思決定プロセスを覆すだけのユーザメリットを提供する必要

があります。現在のところは、そのどちらにも成功していないように見えます。

では、「受注請負」がSI商品の必須要件だとして、どのようにSIという爆弾商品を、安心安全商品に変えるか?――ということですが、傍から見ていて、かつまた実際にやっていて次の処方が有効なようです。

●処方その1「困難は分割せよ」

まずは、マネージメントサイド(とくにPMや営業の方々)では大規模な案件をできるだけ細かくし、かつ単純にしていくことが肝要でしょう。

SIの失敗リスクの1つは、人員の問題です。大規模案件でのリスクは人員の不ぞろいから発生する、さまざまなノイズのコストが大きくなっていることです。可能な限り、少数精鋭で臨むという方針をまずは原則とすべきです。受注請負が原則ですので、アジャイルという形ではなく、小さいチームでのウォーターフォールが望ましいでしょう。ただし、できることには限界があるので、問題を分割して、把握可能な粒度まで落とすということが基本になると思います。開発の範囲も「ここまでは絶対にできる」・「頑張ればできるとは思うがタイムアウトの可能性が高い」・「何がどうであろうと絶対無理」にわけて、ユーザと交渉すべきでしょう。

●処方その2「シンプルな技術を選ぶ」

次に検討すべきは技術の問題です。まずは過度に複雑なアーキテクチャは採用しないということにつきます。これは技術の新旧の問題ではありません。過去に習熟した技術でも、ごてごて新機能やレイヤーが追加され複雑になってしまっては、単純なしくみの新技術に結果として劣るということにはなりがちです。当たりまえですが、「必要にして十分」なアーキテクチャ・スタックを検討すべきです。時代はクラウドという流れもありますが、複雑怪奇な多層スタックのクラウドを利用するよりも、シンプルなベアメタルを採用した方が吉ということはあります。

また、実際に動く側のSEサイドですが、まず原則として「大規模SIにはかかわらないほうがよい」というスタンスを貫くべきです。この業界を見ていると優秀な人材ほど責任感が強く、「よしやったるぜ」というリーダーが多いのですが、それは「逆効果」ということを自覚すべきです。営業が不用意に膨らませた案件は、とくに分割して細かくするということが重要です。無意味にデカイな……と思ったら、即刻警告を発すべきです。

●処方その3「設計する力こそすべて」

次に、徹底して設計力を上げるべきです。SIの案件の失敗の多くは、設計の段階で破綻しています。「**システムは実装したら負け**」です。もっとも品質の高いコードは、もっとも単純なコードであることは論を待たないでしょう。“実装しない技術”は“設計する技術”とイコールです。とにかく動くモノをというアジャイル的な発想はSIにおいては、まったく効果がありません。大事ですのでもう一度書きます。

「まったく役にたちません」

ぎりぎりまで考えて設計し、机上で不明確な部分を徹底的につぶすことがSIの失敗リスクを低減します。何度も設計を事前に見直しても、実際の実装が始まってみると見落としが発見されるのがSIです。近年のアジャイルブームと、実装技法の多様化は、無責任なUML至上主義への反発をテコに、設計軽視の風潮に拍車をかけました。結果として現場の設計力はどんどん落ちています。

よくSIは一品モノの建築にたとえられることが多いですが、であれば、昔の大工の棟梁が何をしていたか、参考にすべきでしょう。材木にノコギリを入れるまえに、徹底してどの木材をどのように使い、どう組み上げるか、頭の中

で何度もシミュレーションをしていたそうです。すなわち、設計の基本は「**制約**」の明確化です。システムの挙動を頭の中で想定し、I/Oとシステムの状態を何度もシミュレートすることも十分な設計です。別段、決まったフォーマットに書き出す必要はありません。設計とはExcelシートを作ることではありません。

おわりに

最後ですが、まとめとして言いたいのは、「**ユーザとちゃんと向き合うべき**」です。「俺はおまえに金を払っているから言うことを聞け」というユーザがいたのであれば、即刻、上役・上司に相談して関係の再検討を迫るべきです。そしてこれは最後の手段ではありません。「最初の手段」です。

結局のところ、SIマーケットで、我々ベンダーができることは、「普通にできるSIを、普通にできるように準備し、奇をてらわずに普通に行う」ということに帰着するように見えます。

まぁ、これが一番難しいのですが。Hadoopのような最先端の分散処理屋が言う台詞ではないのですが、IT屋は目新しいものに無意味に飛びつき過ぎですよ。

うろたえる前に足下を見つめ直しましょう。

SD

第3章

ユーザ企業の「一人情シス」という選択肢

湯本 堅隆(ゆもと みちたか)　Twitter @gothedistance

第1、2章では、システム開発／運用を事業とするIT企業の立場からIT業界を見てきました。本章では視点を変えて、ユーザ企業の情報システム部門(以下、情シス)に所属する著者から、IT業界の展望やエンジニアの働き方について語ってもらいます。ユーザ側に身を置くと、IT企業側とはまた違うエンジニアとしての役割や存在意義が見えてくるようです。

はじめに

2003年に㈱アイ・ティ・フロンティア(現在は日本タタ・コンサルタンシー・サービシズ㈱)に新卒で入社し、プログラマ、開発リーダー、プロジェクトマネージャ(PM)、コンサルタントというキャリアを歩み、やむにやまれぬ事情から中小企業の一人情シスに転身しました、湯本と申します。インターネットでは「ござ先輩」という呼称でいろいろやらせていただいています。

周りを見渡しても、受託開発を生業としていた会社から転職して、誰もエンジニアがいない会社で業務システムを内製して運用しているエンジニアは全然いません。特殊な環境ではありますが、一人情シスも結構おもしろいぞという話をさせていただき、さらにそこで見た景色から今後のIT業界を占っていきます。

SIer時代の疑問

1円も産まないシステムに1,000万円?

これは筆者がSIerにいたときの話です。とある企業向けの受託開発案件でリーダーを任されまして、PMの代わりにプロジェクト計画書を作成していました。「フェーズごとに必要な人員はどのような人で、人月単価を積み上げてコストを出して……」みたいなことをします。その結果、マスタメンテナンスに毛が生えたようなシステムの見積額が1,000万円近くになりました。

「これ……別に1,000万円かけて作っても顧客の売上は1円も産まないな……。高いけど、世の中そういうもんなのかな」。

そのときは「金持ちはいるもんだな」くらいの認識でしたが、1,000万円払わないと手に入らない価値が説明できませんでした。自分で組めば1,000万円も払う必要はないという極論がずっと頭の中に残りました。システムの価値っていったいなんだろう、と。

工数をベースに価格を算出することはまったく問題のない話ではありますが、工数に比例してソフトウェアの価値が上がるなんてことはありません。仕様変更で改修する際に、大量の修正工数を必要とするやっつけ仕事で作られたシステムが、洗練された設計で作られた修正工数が小さく済むシステムより価値が高いわけがないからです。

技術は進化の一途をたどり、今では小中規模のシステムならプログラムレスで開発できるよ

うな時代になっています。先ほどの例ですと、1,000万円を100万円にする方法が多く提供されるようになっています。そのような技術を乗りこなせれば不必要なお金を使わずに済むけれど、技術的知見のある人が発注側にいなければソフトウェアの値段や価値がどれほどのものかを計り知ることができません。ソフトウェアの価値を測る物差しってなんだろうと、漠然とした疑念を抱き始めました。

言われた仕様に従う虚無感

ソフトウェアの価値が自分の中でよくわからなくなってきたと同時に、エンジニアとして最もつまらない時間を過ごすことになりました。その1,000万の案件のあと、元請けとして取れる仕事がなかったので某大手通信会社のシステム構築の案件に孫請けとして入りました。期間はだいたい半年です。委任契約ですので基本的には言われたことを粛々とこなしていれば良いのですが、自分で仕様を考えることはなく、与えられた仕様に沿ってコードを書くのが非常につまらなかったことを覚えています。

その仕様も結構な頻度で変わります。設計もお粗末で、DBからこの値を取ってこいと書いてあるけれどそれに該当するカラムがない、という有様でした。このような作業を行うだけなら確かに委任で十分。言い方は悪いですがバカでもできる。でも、何も身につかない。コードを実装しても誰がどのように使ってくれるかもわからない。そのような時間を過ごした反動からか、コード自体は何も産まないというエンジニアの存在意義を自分で否定するような気持ちが芽生えました。

どんなに優れた技術を持ってきても、ゴミからはゴミしか作ることができません。腐った仕様からは腐ったシステムしか生まれません。システムを構築するIT技術が発展してもシステムには自浄作用がないのだから、プログラムを活かすも殺すも仕様がすべてになります。だからこそ、「きちんとした仕様を策定したい」「自分ならこうやって組むのにという思いを現実にしたい」という気持ちが強くなりました。

当時在職していたSIerで、仕様を決めることができる立場の人間はプロジェクトの最高責任者、PMです。仕様の策定に責任を持つということは、プロジェクト運行のほぼすべてに責任を持つことに等しい。上流工程が狂ったら下流工程が破綻するのは言わずもがな。優れたプログラマがいても、そのプログラマ自身が仕様を決定できないのなら何の意味もない、と前述の委任案件で強く思っていました。プログラマが仕様を決定できないのもおかしな話なんですけどね……。

そのときの自分は社会人4〜5年目、2007〜2008年ごろです。そのころから今のブログを始めたのですが、最初のころは「プログラマって言われたことをやってればいいだけじゃん。いわゆるスーツの人間が産んだカネに文句があるなら同じフィールドに立てよ」なんてことを思っていたことを思い出します。

仕様を決定できても課題は解決できない？

運良く前職の会社は元請けとしてやれる体力もあったので、小さな案件でPMをやらせてもらえました。そのときに感じたことは、「動くものがない状態で、技術的背景がない発注者の方々と仕様を決めることがこんなにも難しいものか」ということ。この仕様で動くかどうかは我々で判断できるのですが、この仕様が業務上／システム上正しいかどうかの判断が、ユーザにもできないことが多くありました。システム屋とクライアントの情シスがToBe（あるべき姿）を描いて、エンドユーザの要望とAsIs（現状の姿）に照らし合わせてFit & Gap（適合する部分とずれている部分を把握）します。「でも、自分たちのシステムや業務をどうすべきかあいまいなのに、大丈夫なんだろうか」と不安になりました。

システムの素人には正しいソフトウェアの仕様などわかるわけがないから、我々がシステムの仕様を決める。それはOK。でも、トマトソー

スのパスタをくれというような大まかな要件しか示されないのなら、味の違いのような細かな仕様の違いはわからない。がんばって細かく詰めていこうとしても枝葉の話ばかりが膨らんでしまう。

筆者も決められた仕様をもとにシステムを作らないといけない立場ですので、スケジュールの遅延は許されません。最終的には外堀を埋めるように要件を固めにいき、「これ以外やりません」と言うしかなくなりました。これでは、「与えられた課題を解決する最適なシステム」を作ることが目的ではなく、「決められた仕様を満たすシステム」を作ることが目的になります。仕様を決める立場になっても、その仕様が与えられた課題を解決する最適解なのかどうかは判断できない、というもどかしさが募ってしまいました。

SIerからユーザ企業へ

コード自体は何も産まないという経験と、仕様を策定できても間違いが生まれるという経験が相まって、「いったいなんでこんなすれ違いが産まれるのか」「課題を解決するソフトウェアを手に入れることがこんなにも難しいのはなぜなのか」を考えるようになりました。

いろいろと考えた結果、このようなすれ違いが発生する原因の多くは発注者側にある。それが筆者の答えでした。

経営者や発注者(非エンジニア)が理解できるITの粒度と、エンジニアが考えるITの粒度はまったく違います。システムを構築するために必要なステップはエンジニアのほうがより細かくイメージ／策定できるから当然のことです。大きな粒度で正しくモノを語れるのは小さな粒度で細かく問題を認識できる人間だけです。どのような過程を経てシステムが構築されるのかわからないが故に、正しい仕様が策定できずソフトウェアを正しく手に入れることができない。コードを書いて仕様の策定ができるようになるには時間もかかりますし、ユーザ企業ではまずそんな人材はいない。そこが問題の根幹にあるように感じました。

ならば、「人材育成の意味も込めて自分たちで作ればいいじゃないか」「内製すればいいじゃないか」と単純に考えました。初めてそう感じたのは6年ぐらい前ですが、今でもそう感じています。

「システムの開発にはRFP、見積もり、提案、要件定義などさまざまな工程を踏む必要があるので非常に時間がかかってしまい、業務変革に求められるスピードに追いつかない」とみなさんおっしゃいますが、自分でシステムを組めばいくらでもスピードは出るでしょう。

自分たちに最適なものがほしい、システムと一緒にビジネスの進化を成し遂げたい。それが目的であれば、自分たちで管理するしかない。管理できないものを改善できるわけがない。丸投げしたら何も手に入らない。前述の1,000万円の話に戻りますが、同じお金をかけるならプログラマを直接雇うなりして作りながら考えていけば絶対良いシステムができるはず。

そんな内製への慕情が生まれ始め、「ソフトウェアに携わるプロとしての自分自身の進化で会社の進化を促していけたらいいな～」と考えていた矢先に、叔父が営んでいる生活雑貨の輸入製造卸の会社、㈲エフ・ケーコーポレーションに入社しました。

事業規模が拡大してきて手管理が限界に達していたこと、システム活用で成功している同業他社に刺激を受けたことなどが理由で、筆者を誘ってくれました。あまりにも何もない環境だったので最初は断ったのですが、何度も誘ってくれたうえに内製への慕情が滅茶苦茶強かったことも重なり、「失敗してもいいや」と飛び込んでみました。それが2009年のことです。

一人きりのエンジニアとして学んだこと

現職ではコードを書くだけではなく、業務設計、経営管理、新たなサービス開発などをやっ

ています。前述のとおり、弊社のエンジニアは筆者一人です。ここではエンジニアとして学んだことにフォーカスします。一人情シスをやって一番良かったことは、頭から尻尾まで自分でやれたことです。ふわっとした要望を自分で要件に落とし込んで、自分で考えて実装して、ユーザに使ってもらってフィードバックをもらって改善する。そうすることで、何が最適な仕様なのかを常に考える習慣ができ、社内システム構築は当然のことながら、対外的な開発でも非常に役に立っています。

仕様決定における落とし穴

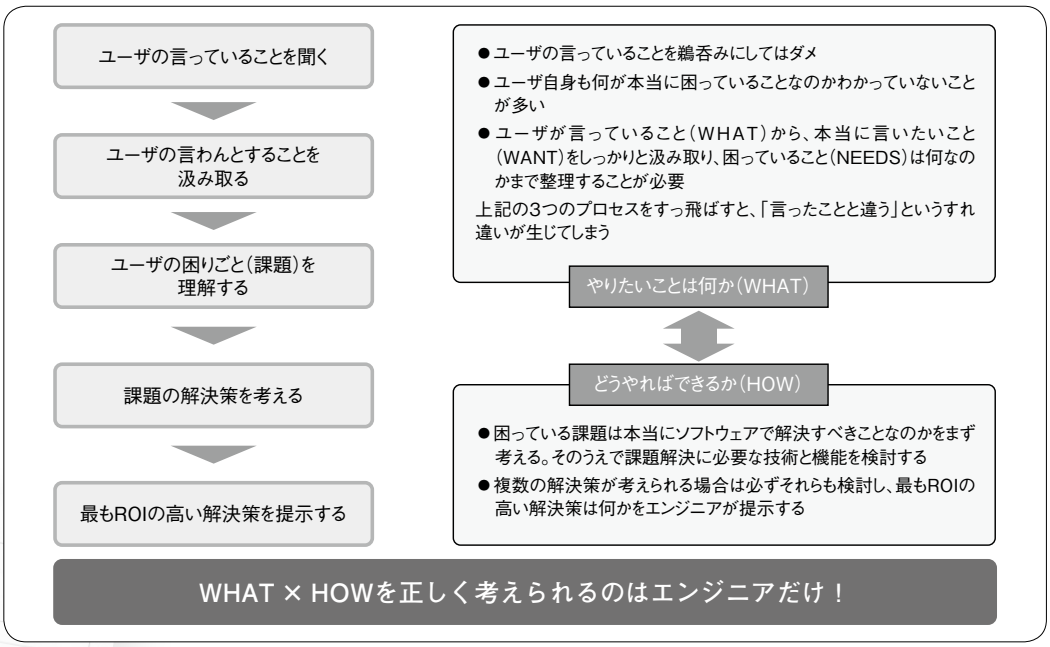
実は仕様を決定するプロセスは、**図1**のようなステップを踏む必要があるものです。

仕様を決めるとは、「ユーザが言っていることは何か」→「なぜそんなことを言っているのか」→「何に困っているのか」→「それをソフトウェアで解決するためには何が必要なのか」→「最もROI（Return On Investment：投資利益率）に合う解決策は何か」ということを順番に詰めていく作業です。

多くの方が第1ステップであるユーザの言っていることをそのまま鵜呑みにしてしまい、言っていることの向こう側にあるニーズを確認せず、間違った要求を実現しようとしてしまいます。そのような場合によくあるのが「言ったものと違うぞ」という反応です。ユーザが言っていることから確実に判断できるのは、やりたいことが満たされていないということだけ。「とくに根拠はないけど、こういう機能があったほうがええんちゃうか」というレベルの話も結構ありますので、本当にこの人は困っていることがあるのか、切実な課題があるのかをシビアに判断していきましょう。欲しかったものは要求の奥に潜む「困っていること」に隠れています。

そして、困っていることを解決する手段は実装することだけではありません。機能の実装は、課題解決の一手段でしかありません。実装するのではなく、ユーザが行っている作業を変えたほうが良い場合もあります。また、新たに機能を実装すれば、ユーザはその機能を使うという

▼図1　仕様を決めるプロセス



作業が増えます。新機能で得られるメリットよりも作業が増えることによるコストのほうが大きければ、実装しないほうが良いということもあります。さまざまな手段を検討したうえで、最もROIに合う解決策を提示することで全体最適を図る。これが仕様を決めることだと考えています。

ユーザを導くのがエンジニアの役目

仕様を検討し正しく決定するには、最後のROIに合う解決策の提示までを「エンジニア」が全部用意する必要があります。経営者／ユーザにはどうしたら良いかという方法がわからないので、仕様を決めるために最適な選択ができません。WHATの検討は誰でもできます。しかし、素人がソフトウェアを作れない以上、HOWを検討し妥当性を判断できるのは、エンジニアだけです。ですので、ユーザの困っている背景を理解したうえで解決策をエンジニアが用意してあげないと、使い手と作り手の間で仕様を決めることはできません。

これが社外の一般顧客相手だと、「ここをあまり細かく言うと、作業が増えてやぶへびになる」「いいや、もう投げちゃえ。どうしたら良いですかね？　決めてもらえます？」とある意味、相手に判断を投げることもできるのですが、社内だと「どうしたら良いかわからないから聞いてるんだろ。それを考えるのがお前の仕事だろ」と返り討ちにあいます(笑)。

ですので、相手に答えを求めるのではなく、相手が決断できるように導いてあげるという視点に切り替えました。プロである以上は、素人の目利きとして先へ先へと導いていきましょう。

高い技術力は必要ない

求められる技術について、システムを内製して自社で使うという前提で話をします。最低限、自分一人でWebアプリケーションを組む力は必要になります。自分で考えて自分で実装するわけですから、LAMPとHTML/CSSがわかっていないと一人では何もできません。

とはいえ、初心者レベルでもいいと思います。高い技術力が求められるのは、工学的／数学的な知見がないとモデル化できないような非常に高度な問題解決に挑戦するときぐらいです。一人情シスではそのようなケースはまずないので、入門書を買ってきてトライして動いた、というレベルでも全然問題ありません。しょせんは自分の会社だけが使うもの。気楽に考えていいです。

しかし、気楽に作ってしまったものが社内で広まってしまうと、さまざまな改善要望が寄せられることになります。ここからが勝負です。「動けばいいや」というやっつけ仕事をしますと、自分が保守運用するので、改修時に自分の首を絞めることになります。

例外に耐え得る設計のセンス

業務システムの仕様は単純なものが多いので、正常系だけならすぐに実装できることが多いです。が、問題になるのは例外処理です。正常系から外れる事態がとても多く起こります。ですので、技術力というよりも「正常系から外れるケースがどれくらいあるのか」を常に考えて、その例外を吸収できる設計にする発想力が求められます。

実装経験が浅い場合は、正常系のみを動かすことで手一杯になります。しかし、業務系システムは手戻りと例外の宝庫です。「注文をもらったけどキャンセルが入った」「このお客さんだけ単価を別に提示したい」「通常は実棚を在庫数として表示したいが、取り置き分の商品に限っては取り置きの残数を表示したい」「間違えて出荷したものを差し戻したい」「在庫があることになっているけど、実際にはなかったのでこの商品を自動的に発注に含み入れたい」……。筆者のところにはこのような例外対応の依頼がたくさんやってきます。

このような例外に柔軟に対応するためには、オブジェクト指向型の言語での開発経験と最低限GoFのデザインパターンは理解している必要

があると考えます。その知見がないと、どうすれば変更に強い設計になるかを考えられず、場当たり的な対応に終始してしまい、技術的負債が積み上がってしまうからです。

機能追加はLess is Moreで

弊社で利用している販売管理システムは筆者が内製したものですが、このバージョン1.0をリリースしたのが2012年8月です。それからさまざまな機能追加を行ってきました。振り返ってみるとシステムに求められる機能というのは「Less is More」である必要があるようです。

より少ない労力で、より多くのことをできるようにする。複数の作業を1つにまとめたり、ある作業を行ったら自動的に次の作業を行ったり、今までと同じ労力でより多くの顧客の注文を受けられるようにしたり、そのようなベクトルに向かわなければなりません。

使っている方の立場にたてば、「機能追加＝ムダな作業の発生」であっては意味がないのです。システムを使うことで作業が増えてしまっては本末転倒です。人的資源が乏しい会社だったので、「その機能があっても、それを使うために別の苦労があったら使わないだけだ」とよく言われました。「なかなか無茶を言うな」と思いましたが、振り返ってみると機能の本質を教えてもらったように思います。

技術を極めたいヒトには向かない

技術を極めたいという方は、ユーザ企業の情シスに入るべきではありません。ユーザ企業でコードを書き続ける時間をもらえることは少ないからです。スクラッチで組むまでもないケースも多く[注1]、実装と言ってもWordPressが使えればたいていのことは解決できるでしょう。自分の周りだけで使うちょっとした便利システムを実装する機会は山ほどありますが、それ以上の規模になると難しくなります。自社の売上向上に貢献するシステムでなければ、継続的に改善をする理由がないからです。

その代わり、技術的制約はありません。あなたの技術的挑戦を邪魔する人はいません。会社の何かしらの課題解決に寄与するアプリケーションやサービスを作るのに必要な技術に果敢にトライしてください。失敗しても誰にも迷惑はかかりません。自分たちだけで完結しますから。今の会社に入ってから、Windows Form、WPF、Objective-C、Android、HTML5+CSS3などを覚えました。全部独学ですから限界はありますけれど、誰にも文句を言われずに新しい技術を使えるのは楽しいことです。

要素技術をひたすら極めるという志向の方にはまったく向いていない環境ですが、平凡かもしれないけど自分の持っている技術で組織に貢献して居場所を作って充実した時間を過ごしたい方には、向いている環境です。

一人情シスをやる意味

一人で大丈夫なの？

筆者がみなさんに聞かれる一番多い質問がこれです。「一人って怖いよね」。

でも、今は一人でもすごく多くのことが達成できる時代になっています。「一人情シス」がユーザ企業の究極のIT活用の形ではないかとすら考えています。組織論で言えば誰か特定の個人に依拠するのはNGです。しかし、今はクラウド技術のおかげで運用作業はほとんど自動化できるようになっています。そのしくみをちゃんと構築して、それを引き継げるようにしていれば別に誰がやってもいいわけです。

弊社は規模が小さいので、日々の業務の中で2人もエンジニアはいらないと感じています。クラウドのおかげで、サーバ運用が本当に楽になりました。

業務システムは筆者一人で内製しましたが、コードもドキュメントも開発環境も検証環境も、

注1） ASP/SaaSで十分な業務もたくさんあります。

システムに手を入れるために必要な情報と環境はすべて自社にあります。内製すれば情報や環境のみならず運用のノウハウが残りますが、外部に丸投げしたものは何も残りません。内製したシステムが会社の経営インフラを支えている以上、システムの主体性を失うことが最大のリスクだということは、経営者も100%同意してくれています。

何かを始めるにも政治的制約はほとんどありません。自分で考えて自分で実装することに集中して取り組んでいることからもわかるように、かなり自由にやらせてもらっています。実装して思うように動かないのは、一人だろうが複数人だろうが等しく起こることです。

そのような作業負荷よりも、社内で居場所を作るまでがきつかったです。自分以外に自分のやっている仕事の内容がわかる人はいないという状況はとても孤独です。筆者が何を考えておりどういう時間が必要だからという説明をしても、非エンジニアの方には理解されません。自分の居場所を築くためには、ただ作るだけではダメ。「このシステムがなければこの仕事は取れなかった。この事業は展開できなかった」というレベルの影響力を出さなければ、内製している意味がありません。それができなければ、筆者がここにいる意味を失うことにもなります。そのプレッシャーとの戦いが、最もハードな部分でした。

内製化の果実とは

これは単純で、コストを低く抑えられることと、ビジネスの外部環境の変化に対応できるスピードを得られることです。

当初、筆者が内製に踏み切った理由は、新たな事業環境に適合するためにオリジナルで組むしかなかったからです。やむにやまれず、です。受注～出荷～売上～請求～回収を自社の都合で連携できる販売管理システムがどこにもありませんでしたし、開発を外注するにしてもべらぼうに高くつきました。そんな予算はありませんし、できるまでに半年も待っていられませんでした。大企業であれば、業務と業務の橋渡しを行う人の人件費をかけることができます。しかし、悲しいかな中小零細企業である弊社ではそのお金すら捻出するのがはばかられました。

もう1つの理由は、改善のスピードが必要だったこと。限られた人的資源で社内のほぼすべての業務を完結する業務システムを作る必要があったので、改善のスピードが求められました。今聞いた要望を、最低でも翌日までには実装して提供するスピードが必要でした。そこで培ったノウハウがさらにムダを省いた業務設計へと反映され、同じ労力でより多くのことができるようになっていく好循環が生まれていきます。

「すべての業務を1つのシステムで完結でき、かつ改善ができて運用費が安いならそれに勝るものはない」というのが、社員の中にエンジニアがいる会社ならではの経営判断です。

弊社の販売管理システムの開発費はゼロ(有料ソフトウェアはいっさい使っていません)、運用費は月々980円です。そんなシステムが年商数億円の中小企業の屋台骨を支えています。システムを使うことで顧客からいただいた発注に関する大部分の事務作業から解放され、注文が増えても付随する事務作業の量はほとんど変わらない状態です。その結果、営業活動や顧客のフォローアップに専念できるようになり、億単位の売上に貢献しています。

BtoB、BtoC問わず売上に貢献しているITシステムやサービスがあり、それらを自分たちで開発／運用するか、もしくはエンジニア経験者を雇用してさらなる売上向上やサービス向上のために改善したい、と考えている会社ならば、内製化の果実(低コストとスピード)を得ることができるでしょう。

一人情シスの醍醐味

一人情シスの醍醐味は、大きく分けて2つあります。

1つは、技術をテコに組織を動かせることで

す。エンジニアのいない企業に入ってポストを作ることができれば、あなたは必要不可欠な人材となります。ブルーオーシャン戦略とでも言いましょうか。技術は、技術を活用できていない環境でこそ最大の効果を発揮しますので。

はじめは自分の作業が何なのかもわかってもらえないし、モノができて動いて成果が出るまでに時間がかかります。しかし、1年経てば自分の作ったシステム／サービスで多くの仕事が生まれ組織が動いていること、成果をあげていることを実感できることでしょう。技術をテコに会社組織を動かし、改善させてより良い状態を築く。ある意味、エンジニアの矜持（きょうじ）を見せつけることにもなります。

もう1つの醍醐味は、やればやるだけ結果が出ることです。自社で運用しているものは、「ここで終わり」という枠がありません。システムや会社のITに手を入れたら入れただけ、自分にダイレクトに結果が返ってきます。良い結果もあれば悪い結果もありますけれど、「○○を解決するためにこのITを導入したら△△になった」というフィードバックを得られるのはエンジニアとして勉強になります。

受託開発でよくある「契約内容さえ満たせていればOK」というような他人行儀な考えは不要です。仕様を満たすだけの目的でプロダクトを作ることはありません。実装段階で気づいたアイデアや欠陥を対応しないままにすると、自分で運用するときに自らの首を絞めます。それがプレッシャーなのかもしれませんが、出し惜しみせずに自分のベストを出すことで、周りを取り巻く環境が良くなる。そして、自分に付加価値をつけていく。おもしろいと思いますよ。

一人情シスとエンジニアのキャリア

事業会社の内製担当という立ち位置は、多くの場合素人の集団に放り込まれることになります。どんなエンジニアでも、素人の中にいれば大先生。IT技術という魔法を正しく使えば、ポストを作ることは難しくありません。世の中の会社が等しくF1を乗りこなせるヒトを求めているわけではありません。高度な技術は高度な問題を解くときだけに有効になります。ほとんどの事業会社は、ITシステムという軽自動車が運転できるようになりたいけれど免許がないし、免許を取得できる教習所もないというような状態です。平たく言えば誰でも活躍できるフィールドがあります。

ですが、先ほども述べたように、一人情シスでは専門性を磨くことが難しいです。LINEのようなトラフィックをさばけるようになりたいと思っていても、LINEのようなインフラはありません。ありふれたインフラでありふれたコードを書くことになります。筆者は個人的には何の問題もありませんが、高度な技術的課題を解決することが望みの方は、重々承知のうえで一人情シスへの転換を考えてください。

ピラミッドの頂点を目指しますか？

筆者が好きなエッセイでこんな一文があります。筆者のブログの記事[注2]でも引用しています。

「私は新しい技術を学ぶことを重視しているが、新しい技術に没頭するというのは、同じ場所に居続けるために全力で走っているにすぎない。どこか別な場所に行こうと思ったら、2倍速く走らなければならないのだ。それは5年たっても陳腐化しないようなトピック――人間的側面とデザイン――について学ぶということだ。」(Jeff Atwood／青木靖 訳)[注3]

この一文が示すことは非常に示唆的です。技術は陳腐化するのが宿命ですから、常に棚卸しをして新しい技術を学ばなければエンジニアでい続けることは難しい。それができたとしても、プログラミングだけを武器に35歳以降を戦っていくのはすごくたいへんだし、それができるのは一握りだけ、というのが筆者の実感です。ピ

注2）http://gothedistance.hatenadiary.jp/entry/20100725/1280066810
注3）http://www.aoky.net/articles/jeff_atwood/everything_you_know_will_be_obsolete_in_five_years.htm

ラミッドの頂点を目指し続け、そこから落ちないように全力で走り続ける。できたらカッコいい生き方ですが、普通の人にはキツ過ぎます。

筆者自身、SI業界からもWeb業界からも離れてしまったことへの不安はゼロではありませんが、エンジニアの居場所はソフトウェアハウスやWeb制作会社の現場だけではないことを身をもって証明できたと思っています。技術的刺激は少ないかもしれませんが、自分のやっていることが会社の事業貢献につながったほうが充実感を得られるのではないか、と感じています。なぜなら、確固たる自分の居場所が作れるからです。弊社の事業は平たく言えば、モノを仕入れて利益分を上乗せした価格をつけて売るだけの商売ですけれど、軌道に乗せるためにはいかにITの手助けが必要かを感じています。その手助けをしたら、自分の居場所がそこにできます。

技術の競争の最前線にい続ける努力をしても、F1はF1の世界でしか生きられない。F1に乗れるヒトも絶対数が決まっている。でも、視野を広げて見ればF1を必要としない企業も多くあります。「コモディティになったもので十分だから使い方を教えてくれないか」という感じの会社のほうが圧倒的に多いです。必要とされているところで力を発揮すればいいだけです。Webサービスをやっている会社であれ、受託開発の会社であれ、ユーザ企業であれ、どこでも同じです。

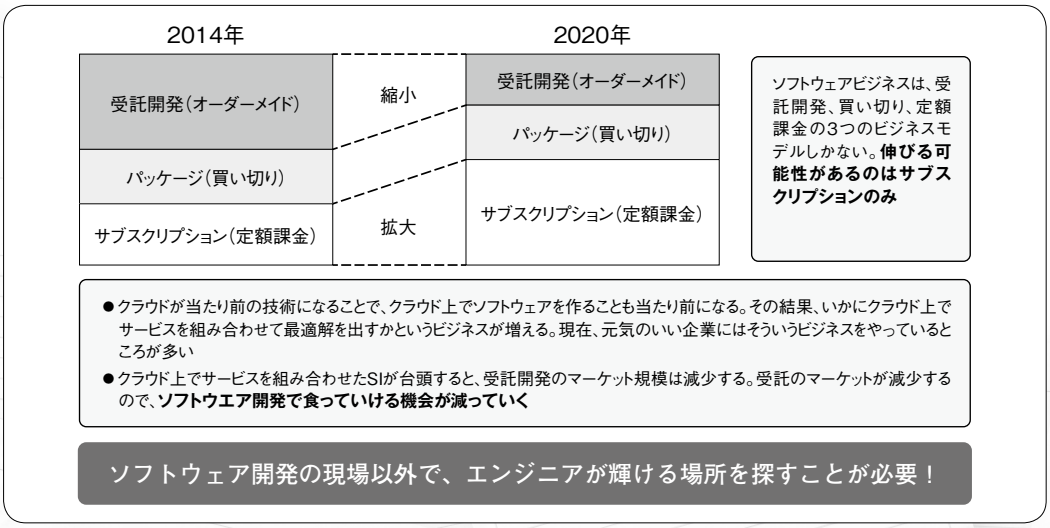
これからのIT業界

まずは**図2**の筆者の考える業界予測図をご覧ください。

断言しますが、今後のIT業界の成長は鈍化します。理由は単純で、成長ドライバが見当たらないからです。ソフトウェアのビジネスは、大きく分けて3つしかありません。オーダーメイド(受託開発)、買い切り、サブスクリプション(月額課金)の3つです。伸びるとしたらサブスクリプションですが、最も金額としては少ない。最も高額な報酬が得られる受託開発のビジネスがIT業界の成長ドライバとなっていないうえに、買い切りやサブスクリプションといったパッケージのビジネスは受託開発に比べて市場規模が小さく参入障壁も低いので、業界内での競争も激しくなります。

要は共食いが起きてしまいWebサービスの世界のようにコモディティ化が促進され価格が下落していき、生き残れる業者が減ります。その

▼図2　クラウドの普及による業界の変化



ような状態で業界全体の成長が加速することは、非常に考えにくい。

クラウドビジネスの台頭でIT産業の売上額はUPするかもしれませんが、業界人口は減る可能性が滅茶苦茶高い、ということです。クラウドが発展したら俺達は死にかけるかもしれないという記事[注4]を2011年1月、今から約4年前に書きました。ITのサプライサイドにとっては、本当に難しい時代に入ります。クラウドの時代は、私たち業界にいる人間にとってみれば「多産多死の時代」と言えると考えています。

ふつーのエンジニアだから活きるキャリアパス

そう考えていくと、ソフトウェア開発の現場以外でエンジニアが輝ける場所が必要になってくる。それはどこだろうか。そのうちの1つの答えが、ユーザ企業に転身するというものです。

「ITシステムを導入したいけど、どうやれば自社に最適な方法で導入できるのかまったくわからない」「山のようにあるソフトウェアから、どのソフトウェアが自社に最適なのか判断できない」というユーザ企業の悩みは永久不変の課題です。「車の運転をしたことがないのに車の乗り方がわかるわけないだろ」という話なのですが、ソフトウェアの乗り方を教えてくれる教習所はどこにもありません。

運転において、車を作る知識は不要ですが、車を制御する知識は絶対に必要になります。それと同じで、ITシステム導入において、ソフトウェアの開発知識は不要ですが、ソフトウェアを乗りこなす知識は絶対に必要です。乗りこなすとは、自社の業務や仕事に合わせてソフトウェアを最適化させることを意味しています。

なぜ乗りこなす必要があるか。事業の変化に伴いシステムに変化を求めるときが必ずやってくるからです。システムを使って自社のビジネスモデルに変革を促したいのであれば、ソフトウェアを乗りこなすための知識が必要です。それがなければ、何をどのように変えたらいいのかがわかりません。

ですので、ソフトウェア開発を生業としていないユーザ企業にも、ソフトウェアの構造に関する理解／知見が必要だと考えています。もしも自分たちの会社を遠くへ運びたいのであれば、ITシステムを活用する以外の方法はありません。速く走ることだけなら、マンパワーで可能です。でも遠くへ行くためには、人に依拠しないしくみが必要です。

そのような「しくみ」を構築するために必要なのは、高度な技術ではありません。何が正しいのかを真摯に考える姿勢です。どうすれば自分の周りにいる人たちがITシステムを活用して全体最適が図れるのか。それを考えるのに必要なのは執念であって、技術ではありません。

世の中のソフトウェアが一般化／コモディティ化しているので、高度なプログラミング技術を求められるケースがどんどん減っており、普通のアプリケーションやWebシステムを作る技術で自社に必要なシステムを作ることができます。ふつーの技術があれば、ふつーにできます。何の問題もありません。

顧問プログラマ

車の免許を取るときにみなさんがお世話になる「教習所の先生」のような立場のエンジニアが、もっと脚光を浴びても良いと思います。単なるITヘルプデスクではなく、仕事を改革し組織の事業運営に貢献していくために。ソニックガーデンの倉貫義人さんが「顧問プログラマ」という言葉を使っていますが、この働き方はもっと増えてほしいです。ふつーのエンジニアの力が必要とされている会社で、多くのエンジニアが顧問として、教習所の先生として、活躍するような事例が増えてほしいな、と顧問プログラマである筆者は強く思います。SD

注4）http://gothedistance.hatenadiary.jp/entry/20110112/1294798106

Jamesのセキュリティレッスン

短期集中連載

パケットキャプチャWiresharkの新展開

最終回

pcapとpcap-ngのファイル形式の違いを知ろう!

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

はじめに

皆さん、こんにちは! James Blakeの「The Wilhelm Scream」を聴きながら寝たら溺れる夢を見たEiji James Yoshidaです。前回はpcap-ngのファイル形式について具体的に解説したので、今回は実際にpcap-ngファイルをバイナリエディタで読んでみましょう。そして最後はpcapとpcap-ngのファイル形式の違いを説明します。

pcap-ngファイルの解析

本連載第1回でも説明したように、使用するツールは次のとおりです。

・hexedit 4.0
http://www.hexedit.com/

インストール方法はお任せしますが、とくにこだわりがない場合はデフォルトでインストールしてください。

さらに筆者のブログからarp.pcapngをダウンロードしてください。

・Eiji James Yoshidaの記録
http://d.hatena.ne.jp/EijiYoshida/20140907/1410071296

ダウンロードが終わったらhexeditを起動してarp.pcapngをドラッグ&ドロップすると、ファイルの内容が16進数で表示されます。

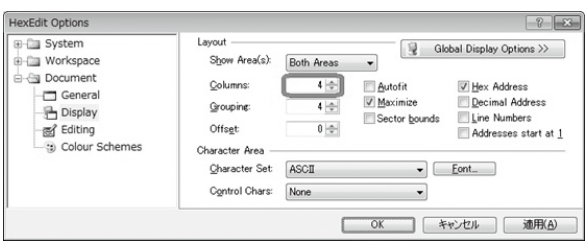
このままでは解析しずらいので、4バイトで折り返すように設定を変更します。16進数が表示されているペインの上で右クリックをして、[Options]をクリックします。[HexEdit Options]ウィンドウが表示されたら、右側ペインにある[Layout]の[Columns:]の値を[4]に変更して、[OK]ボタンをクリックします(**図1**)。これで4バイトで折り返すようになります。

目指すは某アニメに出てくる飛行石を片手に「読める! 読めるぞ!」といったキャラみたいに、皆さんも前号の本連載第2回や本稿を片手にpcap-ngファイルを読むことです。それでは始めましょう!

セクションヘッダ・ブロックの解析

まずはブロックタイプを特定するために先頭4バイトの値を確認すると0x0A0D0D0Aなので、**表1**から最初のブロックはセクションヘッダ・

▼図1 HexEdit Optionsウィンドウ



ブロックということがわかります。

前号でも使った**図2**を参照しながらセクションヘッダ・ブロックをフィールドに分けると**図3**と**図4**になります。

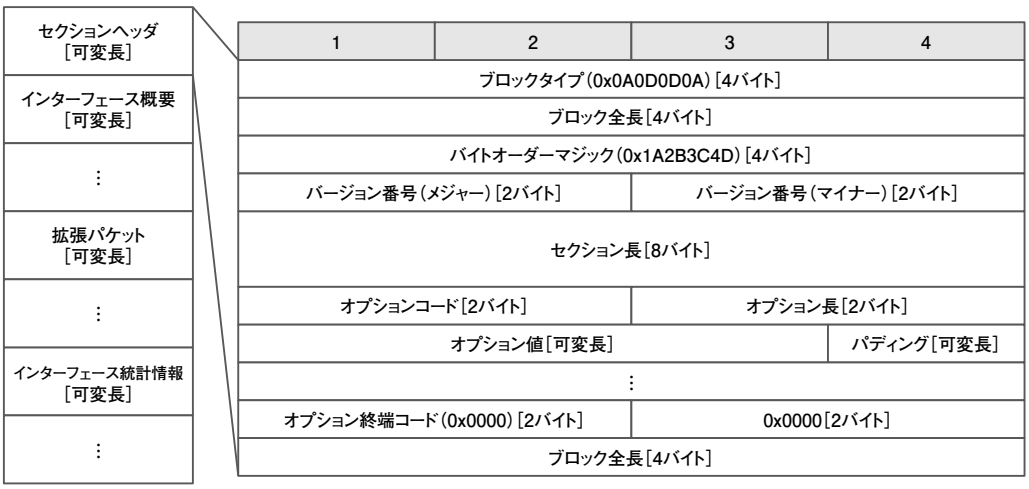
● バイトオーダーマジック・フィールドの解析

バイトオーダーマジック・フィールドは0x1A 2B3C4Dではなく0x4D3C2B1Aと並びが逆になっていることから、バイトオーダーがリトルエンディアンであることがわかります。以降、

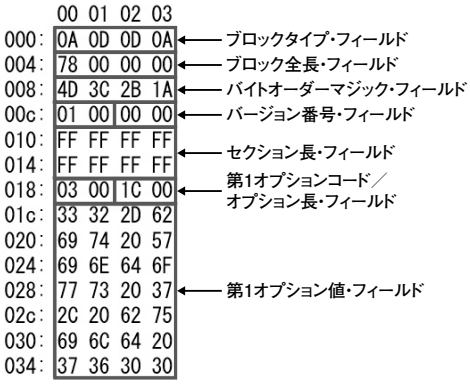
▼表1　代表的なブロックタイプ

ブロックタイプ	ブロック名	説明
0x00000001	インターフェース概要	キャプチャに使用したインターフェースについての情報が記録される
0x00000004	名前解決	DNSの名前解決についての情報が記録される
0x00000005	インターフェース統計情報	キャプチャに使用したインターフェースの統計情報が記録される
0x00000006	拡張パケット	キャプチャされた1つのパケットが記録される
0x0A0D0D0A	セクションヘッダ	セクションについての情報が記録される

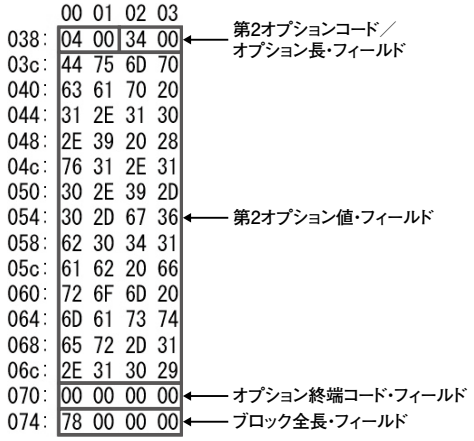
▼図2　セクションヘッダ・ブロックの構造



▼図3　arp.pcapngのセクションヘッダ・ブロック（1）



▼図4　arp.pcapngのセクションヘッダ・ブロック（2）



パケットやOSの名前といったデータを除いてリトルエンディアンと判断します。

● ブロック全長・フィールドの解析

ブロック全長・フィールドは0x00000078ですので、セクションヘッダ・ブロックの全長は120バイトということがわかります。

● バージョン番号・フィールドの解析

ブロックの中身となるデータが入るフィールドです。ブロックタイプによってはブロックボディがさらに複数のフィールドに分かれます。

● セクション長・フィールドの解析

セクション長・フィールドは0xFFFFFFFFFFFFFFFFですので、「スキップしない」が設定されています。

● 第1オプション部分の解析

第1オプションコード・フィールドは0x0003ですので、**表2**からshb_osということがわかります。

第1オプション長・フィールドが0x001Cですので、この後のオプション値の長さは28バイトということがわかります。

第1オプション値・フィールドには、このセクションを作成したOSの名前として「32-bit Windows 7, build 7600」が記録されています。バイトオーダーはビッグエンディアンです。

● 第2オプション部分の解析

第2オプションコード・フィールドは0x0004ですので、**表2**からshb_userapplということがわかります。

第2オプション長・フィールドが0x0034ですので、この後のオプション値の長さは52バイトということがわかります。

第2オプション値・フィールドには、このセクションを作成したアプリケーションの名前として「Dumpcap 1.10.9 (v1.10.9-0-g6b041ab from master-1.10)」が記録されています。バイトオーダーはビッグエンディアンです。

● オプション部分の終端

オプションコード・フィールドとオプション長・フィールドが両方とも0x0000ですので、ここがオプション部分の終端になります。

● セクションヘッダ・ブロックの終端

このブロックの最後の4バイトがブロック全長・フィールドの値と同じ0x00000078ですので、ここがセクションヘッダ・ブロックの終端になります。

▼表2　セクションヘッダ・ブロックの代表的なオプションコード

オプションコード	オプション名	説明
0x0002	shb_hardware	このセクションを作成したハードウェアについての説明がUTF-8で記録される
0x0003	shb_os	このセクションを作成したOSの名前がUTF-8で記録される
0x0004	shb_userappl	このセクションを作成したアプリケーションの名前がUTF-8で記録される

▼表3　代表的なデータリンクタイプの値

リンクタイプコード	リンクタイプ
0x00000001	IEEE 802.3 Ethernet
0x00000009	PPP
0x00000069	IEEE 802.11 Wireless
0x00000071	Linux cooked socket capture（SLL）

インターフェース概要・ブロックの解析

セクションヘッダ・ブロックの次の4バイトが0x00000001ですので、**表1**からインターフェース概要・ブロックということがわかります。

図5を参照しながらフィールドに分けると**図6**と**図7**になります。

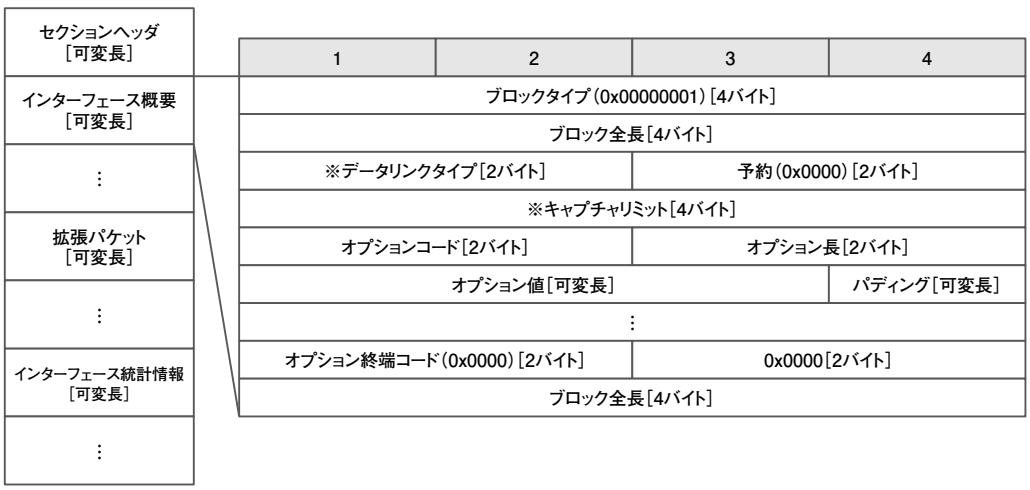
● ブロック全長・フィールドの解析

ブロック全長・フィールドは0x00000094ですので、インターフェース概要・ブロックの全長は148バイトということがわかります。

● データリンクタイプ・フィールドの解析

データリンクタイプ・フィールドは0x0001ですので、**表3**からIEEE 802.3 Ethernetということがわかります。

▼図5　インターフェース概要・ブロックの構造



▼図6　arp.pcapngのインターフェース概要・ブロック（1）

```
     00 01 02 03
078: 01 00 00 00 ←― ブロックタイプ・フィールド
07c: 94 00 00 00 ←― ブロック全長・フィールド
080: 01 00 00 00 ←― データリンクタイプ・フィールド
084: 00 00 04 00 ←― キャプチャリミット・フィールド
088: 02 00 32 00 ←― 第1オプションコード／オプション長・フィールド
08c: 5C 44 65 76
090: 69 63 65 5C
094: 4E 50 46 5F
098: 7B 42 43 38
09c: 37 39 36 36
0a0: 38 2D 38 30
0a4: 43 43 2D 34 ←― 第1オプション値・フィールド
0a8: 45 38 34 2D
0ac: 41 44 37 41
0b0: 2D 46 32 36
0b4: 30 46 35 31
0b8: 31 31 41 44
0bc: 46 7D 00 00 ←― パディング
```

▼図7　arp.pcapngのインターフェース概要・ブロック（2）

```
     00 01 02 03
0c0: 09 00 01 00 ←― 第2オプションコード／オプション長・フィールド
0c4: 06 00 00 00 ←― 第2オプション値・フィールド
0c8: 0B 00 17 00 ←― 第3オプションコード／オプション長・フィールド
0cc: 00 61 72 70
0d0: 20 61 6E 64
0d4: 20 68 6F 73 ←― 第3オプション値・フィールド
0d8: 74 20 31 39
0dc: 32 2E 30 2E
0e0: 32 2E 32 00 ←― パディング
0e4: 0C 00 1C 00 ←― 第4オプションコード／オプション長・フィールド
0e8: 33 32 2D 62
0ec: 69 74 20 57
0f0: 69 6E 64 6F
0f4: 77 73 20 37 ←― 第4オプション値・フィールド
0f8: 2C 20 62 75
0fc: 69 6C 64 20
100: 37 36 30 30
104: 00 00 00 00 ←― オプション終端コード・フィールド
108: 94 00 00 00 ←― ブロック全長・フィールド
```

● キャプチャリミット・フィールドの解析

キャプチャリミット・フィールドは0x00040000ですので、キャプチャするパケットの最大長は262,144バイトということがわかります。

● 第1オプション部分の解析

第1オプションコード・フィールドが0x0002ですので、**表4**からif_nameということがわかります。

第1オプション長・フィールドが0x0032ですので、この後のオプション値の長さは50バイトということがわかります。

第1オプション値・フィールドには、キャプチャに使用したインターフェースの名前として「￥Device￥NPF_{BC879668-80CC-4E84-AD7A-F260F5111ADF}」が記録されています。バイトオーダーはビッグエンディアンです。

● 第2オプション部分の解析

第2オプションコード・フィールドが0x0009ですので、**表4**からif_tsresolということがわかります。

第2オプション長・フィールドが0x0001ですので、この後のオプション値の長さは1バイトということがわかります。

第2オプション値・フィールドには、タイムスタンプの分解能として6つまり10^6が記録されています。

● 第3オプション部分の解析

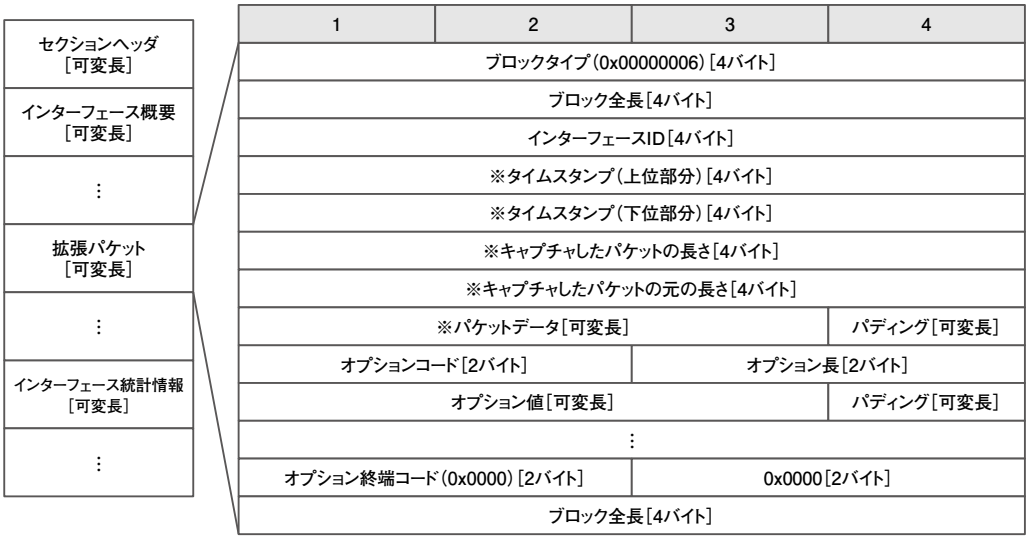
第3オプションコード・フィールドが0x000Bですので、**表4**からif_filterということがわかります。

第3オプション長・フィールドが0x0017ですので、この後のオプション値の長さは23バイトということがわかります。

▼表4　インターフェース概要・ブロックの代表的なオプションコード

オプションコード	オプション名	説明
0x0002	if_name	キャプチャに使用したインターフェースの名前がUTF-8で記録される
0x0009	if_tsresol	タイムスタンプの分解能が記録される。デフォルトの値は「6」で10^6を表す
0x000b	if_filter	このインターフェースに設定されたキャプチャフィルタが記録される（先頭に0x00が追加される）
0x000c	if_os	このインターフェースがインストールされたOSの名前がUTF-8で記録される

▼図8　拡張パケット・ブロックの構造



第3オプション値・フィールドには、このインターフェースに設定されたキャプチャフィルタとして「arp and host 192.0.2.2」が記録されています。バイトオーダーはビッグエンディアンです。

● 第4オプション部分の解析

第4オプションコード・フィールドが0x000Cですので、**表4**からif_osということがわかります。

第4オプション長・フィールドが0x001Cですので、この後のオプション値の長さは28バイトということがわかります。

第4オプション値・フィールドには、このインターフェースがインストールされたOSの名前として「32-bit Windows 7, build 7600」が記録されています。バイトオーダーはビッグエンディアンです。

● オプション部分の終端

オプションコード・フィールドとオプション長・フィールドが両方とも0x0000ですので、ここがオプション部分の終端になります。

● インターフェース概要・ブロックの終端

このブロックの最後の4バイトがブロック全長・フィールドの値と同じ0x00000094ですので、ここがインターフェース概要・ブロックの終端になります。

▼図9　arp.pcapngの拡張パケット・ブロック

```
      00 01 02 03
10c: 06 00 00 00 ←― ブロックタイプ・フィールド
110: 4C 00 00 00 ←― ブロック全長・フィールド
114: 00 00 00 00 ←― インターフェースID・フィールド
118: 74 02 05 00 ←― タイムスタンプ(上位部分)・フィールド
11c: 15 EA 66 2F ←― タイムスタンプ(下位部分)・フィールド
120: 2A 00 00 00 ←― キャプチャしたパケットの長さ・フィールド
124: 2A 00 00 00 ←― キャプチャしたパケットのもとの長さ・フィールド
128: FF FF FF FF ←― パケットデータ・フィールド
       (省略)
150: 02 02 00 00 ←― パディング
154: 4C 00 00 00 ←― ブロック全長・フィールド
```

拡張パケット・ブロックの解析

インターフェース概要・ブロックの次の4バイトが0x00000006ですので、**表1**から拡張パケット・ブロックということがわかります。

図8を参照しながらフィールドに分けると**図9**になります。

● ブロック全長・フィールドの解析

ブロック全長・フィールドは0x0000004Cですので、拡張パケット・ブロックの全長は76バイトということがわかります。

● インターフェースID・フィールドの解析

インターフェースID・フィールドは0x00000000ですので、最初のインターフェース概要・ブロックにあるif_nameに記録されたインターフェースを指しています。

● タイムスタンプ・フィールドの解析

タイムスタンプ・フィールドは上位部分の0x00050274と下位部分の0x2F66EA15を結合して10進数に変換すると1,410,072,918,288,917ですので、あとはインターフェース概要・ブロックにあるタイムスタンプの分解能(if_tsresol)で割るとUNIX時間は1410072918.288917秒になります。

●キャプチャしたパケットの長さ・フィールドの解析

キャプチャしたパケットの長さ・フィールドは0x0000002Aですので、パケットデータ・フィールドの長さは42バイトということがわかります。

●キャプチャしたパケットのもとの長さ・フィールドの解析

キャプチャしたパケットのもとの長さ・フィールドも0x0000002Aですので、パケットのもともとの長さも42バイトになります。

● パケットデータ・フィールドの解析

パケットデータ・フィールドにはキャプチャしたパケットが記録されています。バイトオーダーはビッグエンディアンです。パケット自体の解析は省略します。

● 拡張パケット・ブロックの終端

このブロックの最後の4バイトがブロック全長・フィールドの値と同じ0x0000004Cですので、ここが拡張パケット・ブロックの終端になります。

インターフェース統計情報・ブロックの解析

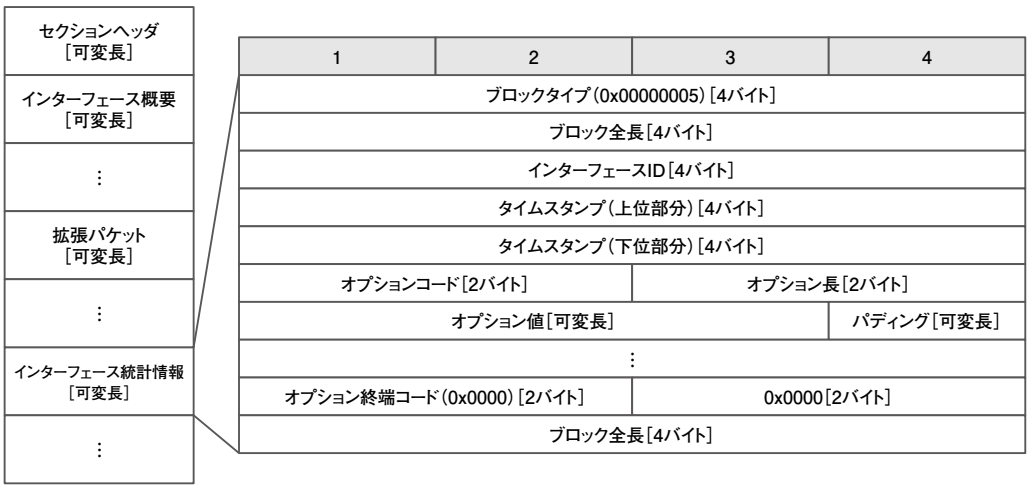
拡張パケット・ブロックの次の4バイトが0x00000005ですので、**表1**からインターフェース統計情報・ブロックということがわかります。

図10を参照しながらフィールドに分けると**図11**と**図12**になります。

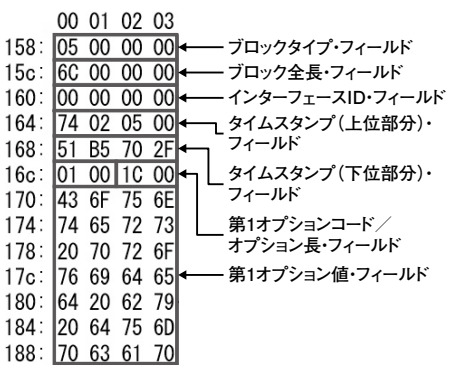
● ブロック全長・フィールドの解析

ブロック全長・フィールドは0x0000006Cですので、インターフェース統計情報・ブロックの全長は108バイトということがわかります。

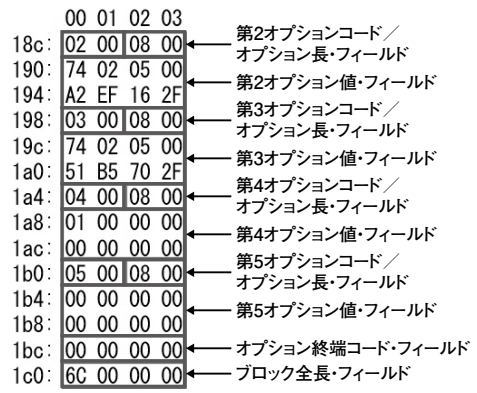
▼図10　インターフェース統計情報・ブロックの構造



▼図11　arp.pcapngのインターフェース統計情報・ブロック（1）



▼図12　arp.pcapngのインターフェース統計情報・ブロック（2）



▼表5　共通のオプションコード

オプションコード	オプション名	説明
0x0000	オプション終端	オプションフィールドの終端を表す
0x0001	コメント	ブロックについてのコメントがUTF-8で記録される

▼表6　インターフェース統計情報・ブロックの代表的なオプションコード

オプションコード	オプション名	説明
0x0002	isb_starttime	キャプチャを開始した時刻がUNIX時間で記録される。フォーマットは拡張パケット・ブロックのタイムスタンプと同じ
0x0003	isb_endtime	キャプチャを終了した時刻がUNIX時間で記録される。フォーマットは拡張パケット・ブロックのタイムスタンプと同じ
0x0004	isb_ifrecv	キャプチャ開始後にインターフェースが受信したパケットの数が記録される
0x0005	isb_ifdrop	キャプチャ開始後にリソース不足でインターフェースがドロップしたパケットの数が記録される

● インターフェースID・フィールドの解析

インターフェースID・フィールドは0x00000000なので、最初のインターフェース概要・ブロックにあるif_nameに記録されたインターフェースを指しています。

● タイムスタンプ・フィールドの解析

タイムスタンプ・フィールドは上位部分の0x00050274と下位部分の0x2F70B551を結合して10進数に変換すると1,410,072,918,930,769ですので、あとはインターフェース概要・ブロックにあるタイムスタンプの分解能(if_tsresol)で割るとUNIX時間は1410072918.930769秒になります。

● 第1オプション部分の解析

第1オプションコード・フィールドが0x0001ですので、**表5**からコメントということがわかります。

第1オプション長・フィールドが0x001Cですので、この後のオプション値の長さは28バイトということがわかります。

第1オプション値・フィールドには、コメントとして「Counters provided by dumpcap」が記録されています。バイトオーダーはビッグエンディアンです。

● 第2オプション部分の解析

第2オプションコード・フィールドが0x0002ですので、**表6**からisb_starttimeということがわかります。

第2オプション長・フィールドが0x0008ですので、この後のオプション値の長さは8バイトということがわかります。

第2オプション値・フィールドには、キャプチャを開始した時刻(UNIX時間)がタイムスタンプと同じフォーマットで記録されています。0x000502742F16EFA2は、10進数に変換すると1,410,072,913,047,458となり、タイムスタンプの分解能(if_tsresol)で割るとUNIX時間は1410072913.047458秒になります。

● 第3オプション部分の解析

第3オプションコード・フィールドが0x0003ですので、**表6**からisb_endtimeということがわかります。

第3オプション長・フィールドが0x0008ですので、この後のオプション値の長さは8バイトということがわかります。

第3オプション値・フィールドには、キャプチャを終了した時刻(UNIX時間)がタイムスタンプと同じフォーマットで記録されています。0x000502742F70B551は、10進数に変換すると1,410,072,918,930,769となり、タイムスタンプの分解能(if_tsresol)で割るとUNIX時間

は1410072918.930769秒になります。

● 第4オプション部分の解析

第4オプションコード・フィールドが0x0004ですので、**表6**からisb_ifrecvということがわかります。

第4オプション長・フィールドが0x0008ですので、この後のオプション値の長さは8バイトということがわかります。

第4オプション値・フィールドには、キャプチャ開始後にインターフェースが受信したパケットの数として1が記録されています。

● 第5オプション部分の解析

第5オプションコード・フィールドが0x0005ですので、**表6**からisb_ifdropということがわかります。

第5オプション長・フィールドが0x0008ですので、この後のオプション値の長さは8バイトということがわかります。

第5オプション値・フィールドには、キャプチャ開始後にリソース不足でインターフェースがドロップしたパケットの数として0が記録されています。

● オプション部分の終端

オプションコード・フィールドとオプション長・フィールドが両方とも0x0000ですので、ここがオプション部分の終端になります。

● インターフェース統計情報・ブロックの終端

このブロックの最後の4バイトがブロック全長・フィールドの値と同じ0x0000006Cですので、ここがインターフェース統計情報・ブロックの終端になります。

pcapとpcap-ngのファイル形式では何が違うのか

pcap-ngファイルの解析は以上です。

本連載第1回から本稿までの解説でわかるようにpcapとpcap-ngのファイル形式は大きく違うので、記録できる情報も当然違います。違いをわかりやすくするために、pcap-ngファイル形式のうちpcapファイル形式でも同様の情報が記録できるフィールドについては、名前の頭に「※」を付けてみました(**図2、5、8、10**)。明らかに「※」が付いているフィールドが少ないことがわかります。つまり、pcapファイル形式よりpcap-ngファイル形式のほうが記録できる情報は多くなります。

また、pcap-ngファイル形式はオプション部分にコメントといった任意の情報が記録できるため、pcapファイル形式にはない拡張性があります。

このような違いによって、pcap-ngファイル形式ならpcapファイル形式だと記録できない次のような情報を記録できます。

- キャプチャに使用した環境やインターフェースについての情報
- パケットをキャプチャしたインターフェースのID
- ナノ秒単位のタイムスタンプ(現時点はデフォルトでマイクロ秒単位)

インシデント対応での使用を考えると、pcapファイル形式にインターフェース関連の情報を記録するフィールドがまったくないのは不安です。なぜなら「どのインターフェースでキャプチャされたパケットなのか」「どのようなキャプチャフィルタがインターフェースに設定されていたのか」といった情報がまったく残らないので、複数インターフェースを搭載したPCの場合だとpcapファイル形式では情報不足で困る場合が考えられるからです。pcap-ngファイル形式では上記のほかにもキャプチャに使用し

たアプリケーションの名前や、インターフェースごとのパケットのドロップ数なども記録できるので、pcapファイル形式より安心です。

pcap-ngファイル対応のツールが少ない

pcap-ngファイルでの保存を考えたときに大きな問題となるのは、対応ツールの少なさです。とくにpcap-ngファイルの読み込みはできるのに保存はできないというツールは今でも多いです。よく使われているtcpdumpも比較的新しいバージョンであればpcap-ngファイルの読み込みはできますが、保存は現時点でもできません。今のところpcap-ngファイルで保存するにはWiresharkか、Wiresharkに同梱されているtsharkやdumpcapを使うことになります。

ファイル形式をpcap-ngからpcapに変換

ツールがpcap-ngファイルの読み込みに対応していない場合は、pcapファイルに変換する必要があります。一般的な方法としてはWiresharkに同梱されているeditcapを使います。次のようにeditcapのオプションに**-F pcap**を設定することで、ファイル形式をpcap-ngからpcapに変換できます。

```
editcap -F pcap file.pcapng file.pcap
```

また比較的新しいバージョンのtcpdumpでも変換できます。

```
tcpdump -r file.pcapng -w file.pcap
```

もちろんpcap-ngのファイル形式にしかないフィールドの情報をpcapのファイル形式に保存することはできないので、pcap-ngからpcapにファイル形式を変換すると記録されている情報が減ることに注意してください。

おわりに

筆者が本連載のようなニッチな内容を、なぜ書くことにしたのか不思議に思うかもしれませんが、セミナーなどで講師をしていると、

「ファイル形式のpcapとpcap-ngって何が違うの？　どっちがお勧め？」

――みたいな質問をされることはけっこうあります。

同じ質問を知り合いの技術者数人にしたところ、

「違いは知らないけど、とりあえずpcap-ngにしておけば良いんじゃない？」

といった返事ばかりでしたので、これは今も知らないまま使っている人が多いのかもしれないと思い、書くことにしました。

ちなみに上記質問への筆者の回答は、

「ファイル形式が大きく異なっていて、情報を記録するフィールドの数はpcap-ngのほうが多いです。またpcapにはなかったオプションを付けられるので拡張性も高く、どのインターフェースでキャプチャされたパケットなのか、どのようなキャプチャフィルタがインターフェースに設定されていたのか、といった情報もpcap-ngなら記録できます。インシデント対応での使用を考えると、こういった情報が記録できるpcap-ngのほうがお勧めです」

――となります。

今回で「Jamesのセキュリティレッスン」は最終回になりますが、いかがでしたでしょうか。バイナリエディタでキャプチャファイルを読むという少々面倒なやり方でしたが、これで皆さんにはpcapとpcap-ngのファイル形式の違いといった知識のほかにも、バイナリエディタでキャプチャファイルを作成したり修正したりできる技術が身についたと思います。

本連載の内容が少しでも皆さんのお役に立てば幸いです。SD

書いて覚える Swift 入門

第 1 回 関数型プログラミングを試す

Writer 小飼 弾（こがい だん）　twitter @dankogai

関数という名の（定|変）数

連載第1回目（もちろん0から数えています[注1]）の今回は、前回予告どおり関数型言語としてのSwiftを見ていきます。ところで「関数型言語」とはいったいなんでしょう。実のところあいまいな用語ではあるのですが、「最低限文化的な関数型」の関数は第一級オブジェクトであることに異議のある読者はあまりいらっしゃらないと思います。「第一級オブジェクトってなんぞや」おさらいしてみるとこういう感じでしょうか。

・変数に代入できる
・関数の引数にできる
・関数を返す関数が書ける

JavaScript、Lua、Perl、Python、PHP、Rubyといった今日日のスクリプト言語はその意味において「最低限文化的な関数型言語」の定義を満たしています。一例としてJavaScriptを見てみましょうか。前回のFizzBuzzを普通に書くとこんな感じでしょうか（**リスト1**）。

▼リスト1　FizzBuzzの例

```
var fizzbuzz = function(n) {
    if (n % 15 == 0) { return "FizzBuzz"; }
    if (n %  5 == 0) { return "Buzz"; }
    if (n %  3 == 0) { return "Fizz"; }
    return n;
}
for (var i = 1; i <= 100; i++) {
     console.log(fizzbuzz(i))
}
```

見てのとおり、fizzbuzzは変数名で、そこに関数を代入しています。Swiftではどうでしょうか？　こうかな？

```
var fizzubzz = func(n:Int)->String {
    if n % 15 == 0 { return "FizzBuzz" }
    if n %  5 == 0 { return "Buzz" }
    if n %  3 == 0 { return "Fizz" }
    return String(n)
}
```

いいえ、ちょっと違います。正しくはこうです。

```
var fizzbuzz = { (n:Int)->String in
    if n % 15 == 0 { return "FizzBuzz" }
    if n %  5 == 0 { return "Buzz" }
    if n %  3 == 0 { return "Fizz" }
    return String(n)
}
```

あるいはこうです。

```
var fizzbuzz:(Int)->String = { n in
    if n % 15 == 0 { return "FizzBuzz" }
    if n %  5 == 0 { return "Buzz" }
    if n %  3 == 0 { return "Fizz" }
    return String(n)
}
```

まとめるとこう。

・funcは不要
　→{ 引数 in 定義 }という形をしている

・型指定は必要
　→前者は{型定義付き引数 in 定義}
　→後者は変数名:型 = { 引数名 in 定義 }

注1） 前回は、編集担当が習性的に「0回」を「1回」としてしまったのですが（汗、仕切り直して今回から第1回とします。

変数指定を{}の外ではなく中でやるあたり、JavaScriptよりむしろRubyっぽいですね。実はもっとRubyっぽいことをこれから見ていきます。FizzBuzzをちょっと変えて、「1から100に対応した結果を出力」するのではなく、「1から100まで入った配列をFizzBuzz変換」することを考えてみましょう。JavaScriptではこうかな(**リスト2**)。

ここで、fizzbuzz()を無名化するとこうなります。

```
var a = range(1, 100).map(function(n) {
    if (n % 15 == 0) { return "FizzBuzz"; }
    if (n %  5 == 0) { return "Buzz"; }
    if (n %  3 == 0) { return "Fizz"; }
    return n;
}))
```

動くことは動きますが、()の中に入ったfunctionはなんとも不格好です。Rubyならどうでしょうか?

```
a = (1..100).map { |n|
    if    n % 15 == 0 then "FizzBuzz"
    elsif n %  5 == 0 then "Buzz"
    elsif n %  3 == 0 then "Fizz"
    else                   n.to_s
    end
}
p a
```

「メソッドの最後の引数がブロックの場合、()の外に書いてよい」というルールのおかげでずいぶんとエレガントです。それではSwiftでは?

```
var a = (1...100).map { n in
    if n % 15 == 0 { return "FizzBuzz" }
    if n %  5 == 0 { return "Buzz" }
    if n %  3 == 0 { return "Fizz" }
    return String(n)
}
println(a)
```

見てのとおり、mapの後の無名関数を()でくくらなくても動きました。無名関数の引数と戻り値のほうも指定していません。このあたりは、Swiftの父の1人であるChris Lattnerもホームページで[注2]「アイデアを拝借した」と率直に答えているとおりです。

▼リスト2　JavaScriptの場合

```
function range(start, end) {    // ないので作る
    var ret = [];
    for (var i = start; i <= end; i++) ret.push(i);
    return ret;
}
var a = range(1, 100).map(fizzbuzz)
console.log(a)
```

▼リスト3　Rubyのアイデアを拝借

```
var a = (1...100).map {
    if $0 % 15 == 0 { return "FizzBuzz" }
    if $0 %  5 == 0 { return "Buzz" }
    if $0 %  3 == 0 { return "Fizz" }
    return String($0)
}
println(a)
```

しかし、Lattnerが言っていない、拝借されたアイデアがもう1つあります。コードで見てみましょう(**リスト3**)。

なんと、inが消えてしまいました。その代わりnがあった位置に$0という未宣言の変数が存在します。この変数のことをプレイスホルダー(placeholder)変数というのですが、$0が最初の引数、$1が次の引数といった具合です。これ、どう見てもPerl 6のアイデアなんですが……。

{}は全部関数!

ここで今まで見てきたSwiftのコードの{}をじーっとよく見てみてください。何か見えてきませんか? ヒントを1つ。ifの後ろの{}は、Swiftでは**省略不可能**です。ifの後ろの条件には()がないのに……。

そう。Swiftでは、意味論的(semantically)には{}は**例外なく**ブロックで、つまり関数なのです!

注2) http://nondot.org/sabre/

本当かどうか、確かめてみましょう。どうやって？——ifを再発明して！（**リスト4**）

動いてます（**図1**）。動いちゃってます。IFは関数で、しかも中に条件分岐がいっさいないのに。それにしても、関数IFはずいぶん奇妙きてれつな形をしています。ちょっと冗長に書き直してみましょう（**リスト5**）。

きてれつではありますが、筋は通ってるのがおわかりいただけたでしょうか。

Swiftが「筋を通しやすい」のは、辞書（dictionary）のリテラルに、Perlによって一般化しJSONによって不朽の地位を得た{}を使っていないこともあります。Swiftでは辞書リテラルは[key:value]の形式をとるので{}が辞書リテラルなのかブロックなのかあいまいになることはありません。{}を見たら例外なくブロック、つまり関数リテラルだと言い切ってしまってよいのです。

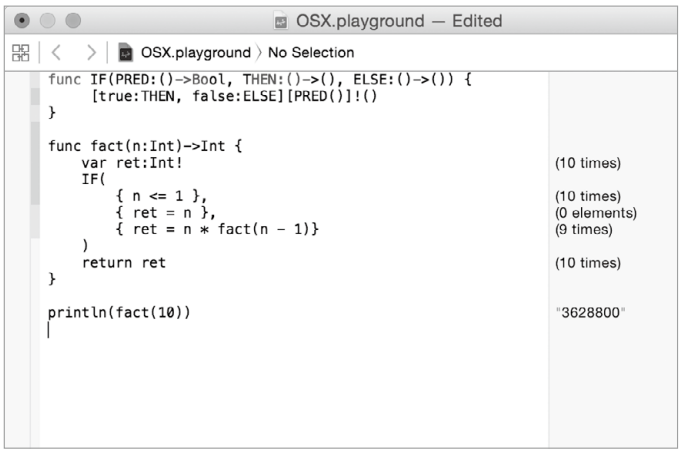
▼リスト4　ifの再発明

```
func IF(PRED:()->Bool,THEN:()->(),ELSE:()->()) {
     [true:THEN, false:ELSE][PRED()]!()
}

func fact(n:Int)->Int {
    var ret:Int!
    IF( {n <= 1},
        { ret = 1 },
        { ret = n * fact(n - 1)}
    )
    return ret
}

println(fact(10))
```

▼図1　ifの再発明



では func は不要かというと……

ここまで見てきたとおり、Swiftにおける関数の本質は{}にあり、func(){}という構文糖衣は無用の長物に思えます。しかしこれが必要になるケースが2つほど存在します。

1つは、再帰関数。先ほどの例のfactも再帰的に定義されています。ここでもう一度「ふつう」に書き直してみましょう。

```
func fact(n:Int)->Int {
    return n <= 1 ? 1 : n * fact(n - 1)
}
```

▼リスト5　関数IFの解説

```
func IF(
    PRED:()->Bool,  // 引数なしでBoolを返す関数
    THEN:()->(),    // 引数なし、戻り値なしの関数
    ELSE:()->()     // 引数なし、戻り値なしの関数
) {
    let dict = [    // [Bool:()->()] な辞書
        true:THEN,  // true  には THEN を
        false:ELSE  // false には ELSE を紐付け
    ]
    let which = dict[PRED()]   // PRED()の結果で辞書引き
    which!()                   // それを実行
}
```

これを、こう書き直してみましょう。

```
let fact:(Int)->Int = { n in
    return n <= 1 ? 1 : n * fact(n - 1)
}

println(fact(10))
```

一見動きそうですが、こんな感じに怒られてしまいます。

```
<EXPR>:9:29: error: variable used within its own
initial value
    return n <= 1 ? 1 : n * fact(n - 1)
```

これを防ぐためには、先にvar、つまり変数として宣言だけしておいたうえで、それに定義を代入しなければなりません。

```
var fact:(Int)->Int
fact = { n in
    return n <= 1 ? 1 : n * fact(n - 1)
}
```

なんとも冗長であるうえに、これではfactを上書きできてしまいます。残念ながら(?)、Swiftには("use strict"されていない)JavaScriptのarguments.callee()やR言語のRecall()相当の、自己再帰のための構文はありません。素直にfuncしましょう。どうしてもという方のために、

```
func recall<T,R>(f:((T->R),T)->R)->T->R {
     var r:(T->R)!
     r =  { n in f(r,n) }
     return r
}
let fact = recall { $1 <= 1 ? $1 : $1 * $0($1-1)
}
```

という手法も一応紹介だけしておきます。何を意味するかは、次回のお楽しみということで。

ただし、ここでfuncが必要になる実例がもう1つ出てきました。総称関数(Generic Functions)です。総称関数とは何か？　型も「変数」になっている関数です。型が変数とはどういうことか？これまた実例で見てみましょう。

```
func add(x:Int, y:Int)->Int {           // 0
     return x + y
}
func add(x:String, y:String)->String {  // 1
     return x + y
}
println(add(4, 2))      // 6
println(add("4", "2"))  // "42"
```

まったく同じ名前、まったく同じ定義の関数が2つあります。違いはなんでしょうか？　そうです。型です。Swiftでは「関数のフルネーム」は名前と引数の型の組み合わせであり、引数の型に応じて違う関数が呼ばれています。実にありがたい機能ですが、定義が同じなのに型ごとに別の関数を定義しなければならないとなるとずいぶん面倒です。もっとDRY[注3]な方法はないでしょうか?

そこで総称関数です。

```
func add<T>(x:T, y:T)->T {
     return x + y
}
add(4, 2)      // 6
add("4", "2")  // 42
```

C++のテンプレートやJavaのジェネリクスに相当するこの機能は、Swiftにもしっかり実装されています。add(4, 2)では4と2からTはIntと推論され、結果add(x:Int, y:Int)->Intがコンパイラーによって生成され、add(4,2)ではadd(x:String,y:String)->Intが生成されるというわけです。

しかし、こう書くことはできません。

```
let add:<T>(x, y)->T = { x, y in
  return x + y
}
```

なぜそう書けないのか。次回はSwiftにおける総称関数を詳しくみていきます。SD

注3) DRY：Don't repeat yourself.『達人プログラマー』アンドリュー・ハント、デビッド・トーマス(著)、ピアソン・エデュケーション刊(2000年)を参照のこと。

サポートエンジニアのクラウドワークスタイル

Heroku女子の開発日記

第5回 用心棒とともに Herokuでアプリ運用!

今回は、Herokuにデプロイしたアプリを運用するうえで頼りになるTipsを紹介します。スケールの方法、監視に便利なアドオン、エラーの対処方法など実践的な内容でお送りします。「サンフランシスコだより」では、前回に引き続きサンフランシスコでのイベント事情をお届けします。

Heroku　織田 敬子(おだ けいこ)

Herokuでアプリを運用する

前回までで、Herokuのメイン機能については一通り解説しました。今回は、実際にHeroku上にデプロイしたアプリをどうやって運用していくかについて話していきます。

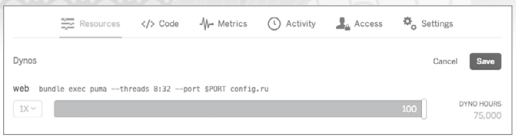
本連載を今回から見始める人は、HerokuのGetting Startedページ[注1]に行き、自分の好きな言語で最後のNext Stepまでやってみましょう。

Herokuは Platformを提供しているベンダですので、なるべく利用者がインフラあたりを意識しなくてもいいように作られています。しかし、自分のアプリがどうなっているか、うまく動いているかどうかは自分でしっかりと監視しておきたいところですね。

アプリをスケールする

Herokuの大きな利点と言ってもいいのが、スケールがとても簡単なことです。たとえば、有名なブログやテレビで取り上げられることで、アクセスの急激な伸びが期待されるとき、その期間のみdynoをスケールして対応できます。しかも、Herokuは使った分のみ料金を支払うので、思ったよりアクセスが来なかった場合でも、すぐにスケールダウンすれば非常に安く上がります。CLIもしくはダッシュボード(**図1**)からスケールができます。

▼図1　dynoを100にスケールする




```
heroku ps:scale web=100
```

アプリをスケールする際に陥りがちなミスとして、データベースなどのアドオンのプランが、スケールしたdyno数に対応していない場合があります。それぞれのアドオンは、コネクション数やストレージなどがプランによって異なりますので、事前にしっかりと負荷テスト(負荷テスト用のアドオンもあります[注2])をして、スケールアップした場合にもアプリが動くことを確認しましょう。

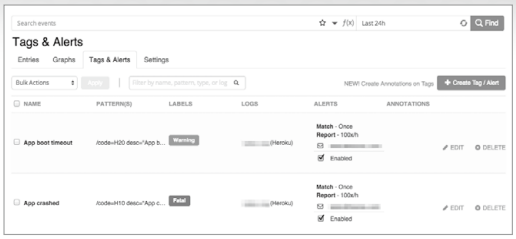
アドオンを使って監視する

Loggingアドオン

たいていのLoggingアドオンには通知機能がついています。この通知機能を使用し、HipChatやPagerDutyなどの好きなチャネルに通知を送るようにしましょう。たとえば、LogEntriesアドオン[注3]ではデフォルトで、各

注1) URL https://devcenter.heroku.com/start
注2) Loader.io URL https://addons.heroku.com/loaderio
注3) URL https://addons.heroku.com/logentries

▼図2　LogEntriesのアラート設定画面



▼図3　LibratoのHeroku Runtime Metrics画面

種Herokuエラーについての通知が有効になっています。頻度によってどう通知するのかなどを変更できるので、非常に強力です(**図2**)。

Librato アドオン

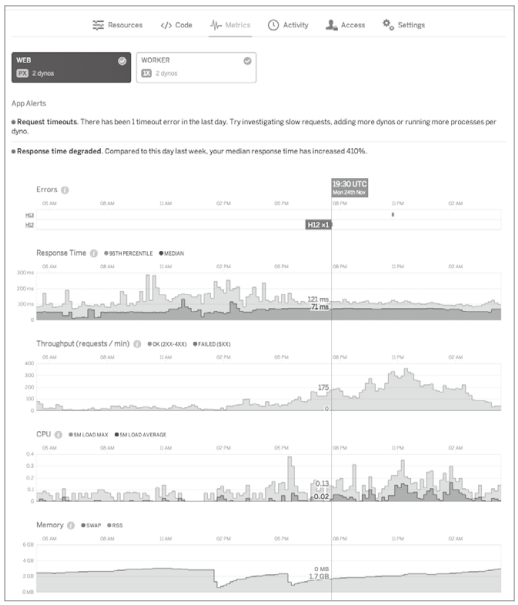
リクエスト数の推移や平均レスポンス数などは、Webアプリを管理しているなら、必ず監視しておきたいメトリクスです。Librato[注4]はそれだけで非常に強力なモニタリングツールでHeroku内部でも多用されていますが、Herokuアドオンで使うとデフォルトでHerokuのログを読んで、さまざまなメトリクスを表示してくれてさらに便利です。たとえば、Heroku Runtime Metrics(**図3**)ダッシュボードではそれぞれのdynoのload averageやmemory usageを見ることができます。これにはHeroku Labsのlog-runtime-metrics[注5]を追加する必要があるので注意してください。また、Librato を使っても各種通知ができます。

Heroku dashboardを使って監視する

Herokuの新しいDashboardでは、2個以上のdynoを走らせているアプリにおいて、「Metrics」という名前のタブが利用できます(**図4**)。このMetricsタブもアドオンに負けず劣らず良い出来となっており、アプリが今どんな状態で動いているのかを確認できます。

各種エラー、レスポンスタイム、スループット、

▼図4　Heroku dashboardのMetricsタブ



CPU/Memory使用量と、最低限カバーすべきところはカバーされています。このMetricsも非常に有用なのですが、データが24時間前までしか遡(さかのぼ)れないなどの制約もあるので、やはりアドオンと組み合わせて使うのが良いでしょう。

このようにいろいろな手法で監視をすることにより、適切なタイミングでスケーリングやトラブルシュートができるようになります。

Heroku エラーコード

Herokuでアプリを運用していると、Heroku

注4) URL https://addons.heroku.com/librato
注5) URL https://devcenter.heroku.com/articles/log-runtime-metrics

独自のエラーコード・メッセージを見る機会があると思います。各エラーコードについての詳細はDev Centerのページ[注6]を参照してください。ここでは、よく見るエラーコードを2つ取り上げて解説したいと思います。

H12 - Request timeout

「H12」は、Herokuのルータがdynoからタイムアウト値である30秒を超えてもレスポンスを受け取らなかった場合に発生します。「H12」が発生すると、クライアント側ではHerokuのエラーページ(もしくは指定されたエラーページ)が表示されますが、サーバ／dyno側では処理がそのまま続けられます。もし慢性的に時間のかかるリクエストがある場合、これに続くリクエストもこのリクエストを待ち続けることになるので、レスポンスタイムが雪だるま式に増え続け、アプリがダウンする恐れがあります。「H12」の対策としては、

・時間のかかるリクエストがないかどうか、アプリをチェックすること(New Relicアドオンが役立ちます)
・アプリケーションサーバのworker数(プロセス数)を増やしてリクエストを同時に捌けるようにすること
・アプリケーション側[注7]もしくはアプリケーションサーバ側[注8]でタイムアウトを設けて、時間がかかり過ぎているリクエストをkillしてしまうこと

などが挙げられます。Dev Centerにも「H12」に関する記事[注9]があるので、今現在時間のかかるリクエストがなくとも一度目を通してみることをお勧めします。

R14 - Memory quota exceeded

Herokuではdynoのサイズによりメモリの上限が変わります。この決められたメモリの上限を超えて使用すると、「R14」というエラーがログに吐き出されます。「R14」が発生している場合はメモリがスワップしている可能性が高く、これはアプリケーションのパフォーマンスに大きく影響してきます。「R14」の対策としては、

・dynoのサイズを変更すること
・アプリサーバのworker/thread数を調整すること
・アプリでメモリリークがないかを確認すること

が挙げられます。Heroku上ではメモリまわりのデバッグがローカルに比べて難しいので、ローカル環境でも再現可能かチェックしてデバッグしていくのもいいでしょう。Javaアプリに関してはDev Centerの記事[注10]にメモリまわりのトラブルシュートが載っています。

サンフランシスコだより

なんと！　ここでしつこくオフィスに遊びに来てねと言っていた甲斐があり、本連載を見てHerokuに遊びに来てくれた方がいらっしゃいました。ありがとうございます。懲りずに言いますが、筆者はいつでもwelcomeですのでTwitterやFacebookなどでpingいただければ対応します。

日本では冬に向かって寒さも増していると思いますが、サンフランシスコは年中あまり気温が変わらないので、夜に自転車に乗っていると寒いなぁと思うことはありながらも、冬本番という感じではありません。今回はサンフランシ

注6) URL https://devcenter.heroku.com/articles/error-codes
注7) URL https://github.com/heroku/rack-timeout
注8) URL http://unicorn.bogomips.org/Unicorn/Configurator.html#method-i-timeout
注9) URL https://devcenter.heroku.com/articles/request-timeout
注10) URL https://devcenter.heroku.com/articles/java-memory-issues

スコの市民性を交えながら2つのイベントについて話したいと思います。

サンフランシスコジャイアンツの優勝

サンフランシスコには陽気な人が多いのですが、それも筆者がサンフランシスコを好きな理由の1つです。今年はサンフランシスコジャイアンツがワールドシリーズで優勝したため、本当にお祭り騒ぎで、もうみんなクラクションを鳴らしながら車を運転、道端では誰も彼もがハイタッチ状態でした。筆者の住んでいるミッションという地区はとくにこういったときに荒れがちで(東京で言えば渋谷スクランブル交差点をイメージしてもらえれば)、2年前に優勝したときなどはテンションが上がり過ぎて暴徒と化したファンたちが市バスを燃やすほどでした。今年もミッションストリートが歩行者天国状態となり、たくさんの人が優勝を祝っていました。

後日優勝パレードが盛大に行われ、筆者はちょうど優勝パレードの前日に「DMV」と呼ばれるアメリカの運転免許センターに行っていたのですが、職員の方が「明日なんて誰も来ないわよ、あー明日なんか急病になって休んじゃいそう」と言っていたのが印象的でした。陽気ですよね。パレードは金曜に行われましたがHerokuでも、とくにジャイアンツファンの社員はパレードを見に行っていました。

こんなノリのいい人が多く、エネルギーが溢れるサンフランシスコですが、土地勘のない人はこのようなイベントはとくに気を付けてほしいところです。みんなが浮かれているからこそ犯罪もまた起きやすく、その日は爆竹の音か銃声かもわからないような音が夜遅くまで鳴り響き(発砲も実際に何件かありました)、家の近くには病院もあるのですが救急車の音も普段より多く聞こえました。サンフランシスコ在住の友人のSNSでは、暴徒化したジャイアンツファンに向けて「私たちのサンフランシスコをいじめないで」といった書き込みも多数見られました。

ハロウィン

日本でもハロウィンは年々盛り上がりを見せていて、今年も大盛り上がりだったようですね。サンフランシスコでも、ハロウィンは楽しむべきイベントの1つとなっていて、Herokuでもハロウィンパーティーをしました。Herokuだけではなくほかの会社でもこういったハロウィンパーティーというのは行われていて、従業員の子供なども交えてパーティーを行っています。筆者のルームメートも気合たっぷりで素敵なコスチュームを着ていました(**写真**)。

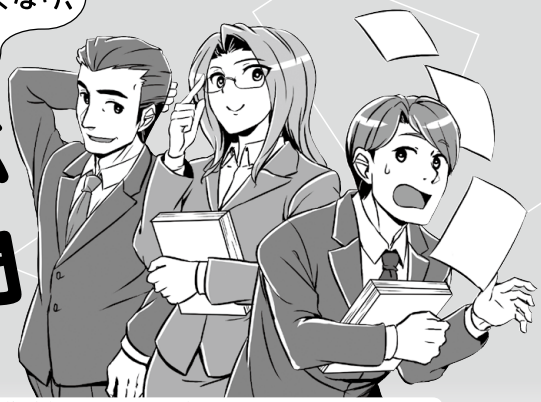
ちなみに日本ではあまり子供がトリック・オア・トリートと言って家を回ることは(筆者が知る限りでは)しないと思うのですが、スティーブ・ジョブズの家にトリック・オア・トリートをしに行くとなかなか豪華なお菓子がもらえたらしいです。あのあたりにはそれこそ億万長者がたくさん住んでいるので、筆者も一度は小さくなってトリック・オア・トリートをしに行ってみたいものです。SD

▼写真　ハロウィンパーティー

帰宅が5分早くなり、休出もなくなる!?

目指せ！定時帰りのエンジニア！

Hinemosで学ぶジョブ管理超入門



㈱NTTデータ 基盤システム事業本部　茶納 佑季（ちゃのう ゆうき）　chanouy@nttdata.co.jp

cronによる運用自動化の反省をふまえ、いよいよ「Hinemos」を使い始めた藤井君。Hinemosの基本を学びながら、ジョブの作成、LinuxとWindows Serverが混じった環境でのジョブの実行に挑戦します。　イラスト（高野涼香）

第4回　Hinemosで構築するジョブシステム！

登場人物紹介

藤井　今年SI企業に入社した新人SE。運用自動化のために日々奮闘中。

上司　軽いノリで仕事を依頼してくるが、藤井君の成長を考えている。

定時先輩　藤井君の先輩社員。華麗に仕事をこなし定時に颯爽と帰っていく優秀な女性。

前回までのあらすじ

今年SI企業に入社した新人SEの藤井君は、社内の「勤怠管理システム」の運用を上司から任されました。そのシステムを運用するためのシェルスクリプトを夜間の不在時にも実行できるよう、cronでのスケジュール実行について学びました。そんな折、勤怠情報が転送されていないというトラブルに見まわれ、何とか解決したものの、現在のシェルスクリプトとcronによる運用に限界を感じ始めたのでした。

Hinemosってなんだっけ？

上司「勤怠管理システムのトラブル、どうやら解決したみたいだな」

藤井「はい……今回は何とか解決しましたが、やっぱり今の状況だとトラブルが起こったとき、cronの設定を変更したり、いろいろなログを参照したりしなければならないので、問題の解析は本当にたいへんですね。しかも依頼のあったWindows Serverと連携するほうはまだ解決案がなくて……」

上司「そうだな。この先、こんな運用を続けていっても、スケジュールを急遽変更したり、実行結果をすぐに確認したりするなんて、とてもじゃないが対応できないからな。今動いているシステムのしくみや課題もわかったところだし、そろそろ前に言ってたツールを試してみるか？」

藤井「ついにHinemosを使えるんですね！　さっそく調べてみます！」

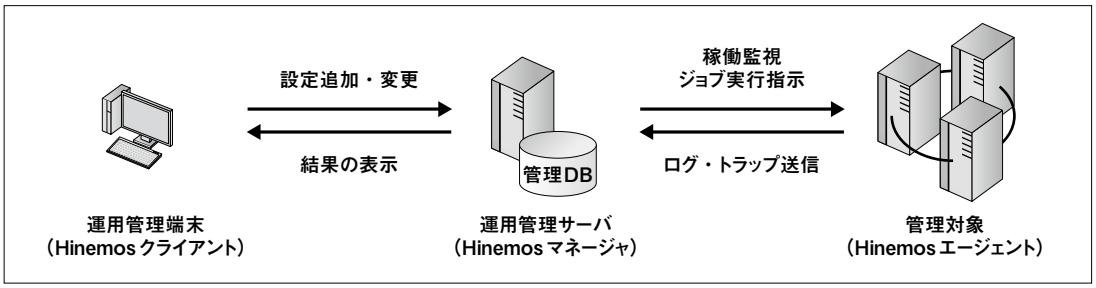
Hinemosとは

藤井「よし、今の課題を整理して、Hinemosでちゃんと課題を解決できるのか調べよう」

藤井「今の課題は大きく2つ。『実行結果はログで確認しているので、すべてを把握するためには複数のログを見る必要があること』と『Windows Serverと連携する必要があること』だな」

藤井「まず1つめはHinemosクライアントっていうアプリケーションから各処理の実行結果を確認できるみたいだからクリアかな。2つめはLinux

▼図1　Hinemosの3つのコンポーネント



に対する処理もWindows Serverに対する処理も実行できるみたいだし、Hinemosポータルサイトには『サーバを跨がる一連の処理をジョブネットとして実行』できるって書いてあるからクリアできそうだ！」

さて、Hinemosについておさらいをしましょう。

Hinemosはオープンソースの統合運用管理ツールです。今、藤井君が目指している運用業務の自動化を実現できるもので、システム運行に必要な処理（＝ジョブ）の自動化や監視の効率化を行えます。Hinemosでは物理環境に加え、仮想化・クラウド環境が混在するマルチクラウド環境も単一の操作画面で管理できます。

Hinemosは3つのコンポーネントから構成されています（**図1**）。

・運用管理サーバ（Hinemosマネージャ）

Hinemosの運用管理機能を提供するサーバ。Hinemosマネージャをインストールし、ジョブ定義などの各種設定内容を保持したり、実行を指示したりする。また、その結果をHinemos内部のデータベース（PostgreSQL）に蓄積する

・運用管理端末（Hinemosクライアント）

運用管理者やオペレータが操作するコンソール端末。Hinemosクライアントをインストールし、その画面から設定を投入したり、ジョブの実行結果を確認したりできる

・管理対象（Hinemosエージェント）

管理対象となる機器。Hinemosエージェントを導入し、ジョブを実行。なお、監視機能だけを用いる場合は、大半の監視機能がOS標準のパッケージを利用するためHinemosエージェントを導入しなくても使用できる

Hinemosで構築！ジョブ管理システム！

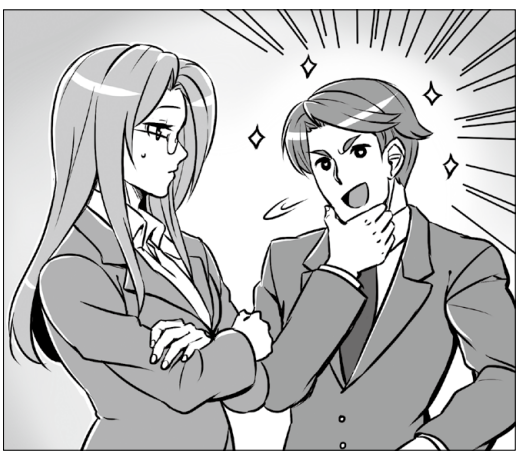
藤井「ふーん。3つのコンポーネントを導入するだけでいいのか。よーしHinemosでジョブ管理の自動化に取り組むぞー！」

定時「おーい、藤井くーん。ついにHinemosを導入するんですって？」

藤井「あ、定時先輩！　そうなんです。ついにHinemosでジョブ管理システムを構築するんです（キリッ」

定時「（ジョブ管理システム……？）あぁ、ジョブ管理を自動化するのね。Hinemosのインストールは簡単で、5分で入れちゃう人もいるみたいね」

藤井「え？　そうなんですか！？」



定時「ええ、この前Hinemosが出展されているイベントで見たわ」

Hinemosインストール

まずはSourceForge.JPのHinemosプロジェクト[注1]から必要な資材をダウンロードしましょう。インストーラと同じくSourceForgeからダウンロードできるインストールマニュアル（install.pdf）に従い、Hinemosマネージャ、Hinemosエージェント、Hinemosクライアントをインストールします。

①Hinemosマネージャを運用管理サーバにインストール
②Hinemosエージェントを管理対象機器にインストール
③Hinemosクライアントを運用管理端末にインストール

藤井「ほとんど、『yes』って入力したり『次へ』を選択したりするだけで入っちゃった！　あとはそれぞれを起動してっと。よし、起動したぞ」

定時「次はHinemosクライアントからHinemosマネージャに接続して、管理対象をHinemosに登録してみましょう。Hinemosでは、管理対象を登録するデータベースをリポジトリと呼んでいるわ」

藤井「はい。ん、管理対象の登録もこのピンク色のところにIPアドレスと必要な情報を打ち込むだけでいいのか」

定時「ピンク色のところが必須入力項目ね。それと、管理対象のことをHinemosではノードと言うわ。あと、Hinemosには『Find By SNMP』っていう機能があってね、リポジトリのノードの作成・変更ダイアログにIPアドレスを入力してFindボタンを押すと、ユーザが任意に入力する必要のある、ファシリティID（管理対象を識別するためのID）とファシリティ名（管理対象を識別するための名前）以外は、自動的に設定してくれるの[注2]」

藤井「この機能で簡単に入力できますね！　ところで、このリポジトリに登録したノードにHinemosエージェントがちゃんと入っているかどうかってわからないんですか？」

定時「リポジトリパースペクティブの、エージェントビューで確認できるわよ」

藤井「あ、本当だ。登録したノードの最終接続時刻が表示されてる」

定時「Hinemosエージェントを監視するっていう方法もあるけど、それはまた今度ね」

Hinemosのジョブ機能

定時「Hinemosのインストールから、管理対象の登録、疎通の確認までできたわね」

藤井「はい！　これでHinemosでジョブを管理するための準備が整いました！」

ジョブ管理ツールでは、ジョブの定義や実行はもちろん、複数ジョブ間の先行・後続といった流れの制御やスケジュールに則ったジョブの制御が行えます。Hinemosのジョブ機能では、ジョブはジョブユニットという大きなフィールド上に定義します。また、ジョブを実行する方法として、「手動による実行」「スケジュールによる定期実行」「ファイルの生成などを検知しての実行」「監視機能と連動した実行」を行えます。制御についても**図2**のように多種多様なジョブの実行条件の制御をすることができます。

Hinemosでジョブを動かそう！

ジョブの定義

藤井「まずはジョブを定義してみます！」

定時「そうね。まずは、ジョブを1つ作成して、手動実行してみましょう」

HinemosのジョブやジョブネットはP.111のコラ

注1） URL http://sourceforge.jp/projects/hinemos/または、Hinemosポータルサイト URL http://www.hinemos.info/からもリンクされています。

注2） Linux環境の場合はsnmpd、Windows環境の場合はSNMP Serviceが起動している必要があります。ノードの作成など、各種機能に関する詳細はSourceForge.JPからダウンロードできるユーザマニュアル（user.pdf）をご覧ください。

ムにあるとおりジョブユニットの要素として定義されます。そのため、ジョブ、ジョブネットを作成するためには、ジョブユニットを作成する必要があります。ジョブパースペクティブを開き、左側のジョブツリー上で、最上位のジョブを選択して、ジョブユニットの作成ボタンを押します。ジョブユニットのジョブID、ジョブ名を設定し、OKボタンを押すことでジョブユニットが作成されます。

続いて、ジョブを作成します。ジョブユニットを選択してジョブの作成ボタンを押します。ジョブユ

Column

Hinemosのジョブの基礎

ジョブユニット・ジョブネット・ジョブ

Hinemosのジョブは大きく、ジョブユニット・ジョブネット・ジョブから構成されています。

・ジョブユニット
　ジョブ階層の最上位要素。すべてのジョブネットとジョブは、このジョブユニットの要素として設定する。このため、ジョブを登録する際には、まずジョブユニットを作成する必要がある
・ジョブ
　Hinemosジョブ管理における最小の実行単位。管理対象ノード上で実際に起動するコマンド（シェルスクリプトなど）に相当する。待ち条件には、時刻と、同階層にあるジョブネットもしくはジョブの終了を条件にできる
・ジョブネット
　複数のジョブをひとまとめにして扱うことのできる要素。**図3**のようにジョブネットの中にジョブネットを作ることもできる。よって、ジョブネットはジョブネットとジョブから構成され、複数のジョブネットとジョブを登録できる。待ち条件には、時刻と、同階層にあるジョブネットもしくはジョブの終了を条件にできる

ジョブユニットやジョブネットを実行すると、そのジョブユニットやジョブネットに登録された下位階層のジョブ（もしくはジョブネット）が実行されます。下位階層のすべてのジョブ（もしくはジョブネット）の実行が終了することがジョブユニットやジョブネットの終了条件となります。

終了状態・終了値

Hinemosのジョブユニット・ジョブネット・ジョブは、それぞれ終了値と終了状態を持っています。

終了値は、終了状態によって決定され、どの終了状態のときに、どの終了値とするのかを設定できます。終了値はジョブネットとジョブユニットの終了状態を決定するために使われるほか、待ち条件にも使われます。

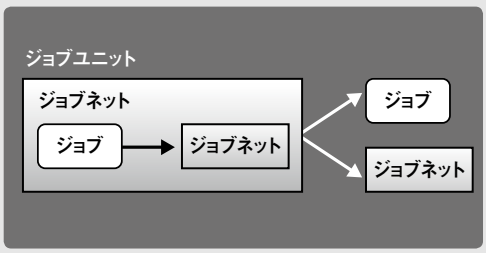
終了状態には正常、警告、異常の3つの状態があります。終了状態の決まり方は、ジョブとジョブネット（もしくはジョブユニット）で異なります。

ジョブの場合、終了状態はジョブ実行時に実行されるコマンドのリターンコードの範囲で決定します（たとえば、リターンコードが0の場合は「正常」。リターンコードが1～9の場合は「警告」。それ以外は「異常」のように設定できます）。

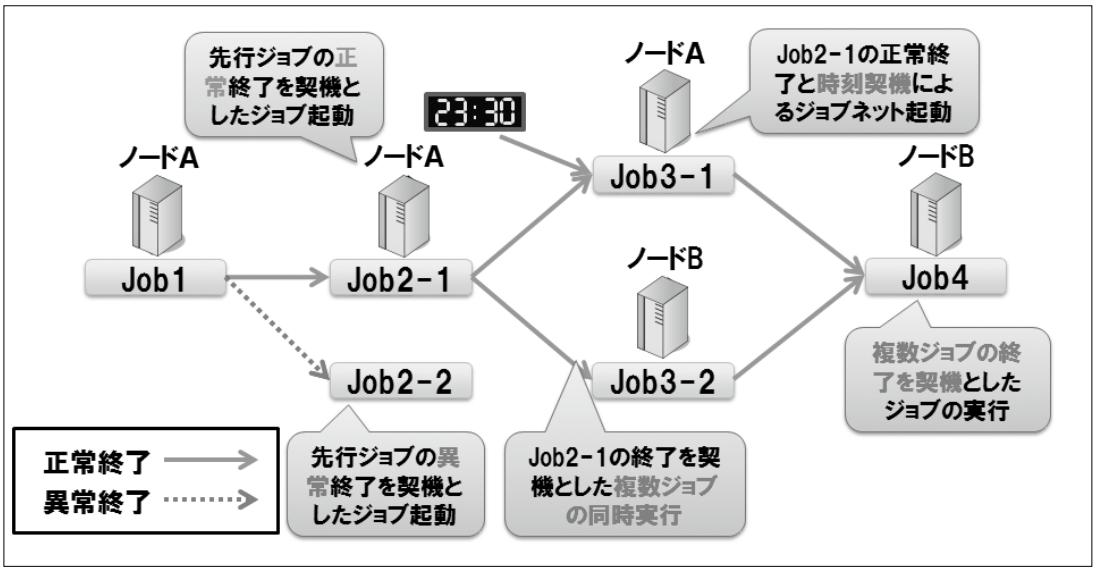
ジョブネットの場合は、そのジョブネットの実行時に実行されるすべてのジョブ（もしくはジョブネット）の中で、後続ジョブを持たないジョブ（もしくはジョブネット）の終了値の範囲で決定します。

この終了状態は待ち条件に使えるほか、監視機能またはメールにて結果を通知できます。

▼図3　ジョブユニットの構成



▼図2　Hinemosのジョブ機能の制御



▼図4　ジョブの作成

ニットのジョブID、ジョブ名を設定し、コマンドタブで、スコープの欄に処理の実行先、起動コマンドに実行するコマンドを入力します（**図4**）。

定時「起動コマンドには、動かしたいスクリプトやコマンドを入力するんだけど、今回は、日付をtest.logに出してみましょうか」

藤井「わかりました」

定時「OKボタンを押して確定させたら、登録ボタンを押してジョブを登録してみて」

ジョブを動かそう！

藤井「さっそく実行してみます！」

定時「ええ。test.logはエージェント側のターミナルで表示させておきましょうね」

藤井「はい。ええっと、ジョブを実行するためには、実行したいジョブを選択して、実行ボタンを押すっと」

```
# tail -f /tmp/test.log
ジョブ1:2014年 12月 18日木曜日 12:31:22 JST
```

藤井「しっかり動きました！」

定時「やったわね。Hinemosクライアントからも確認してみて」

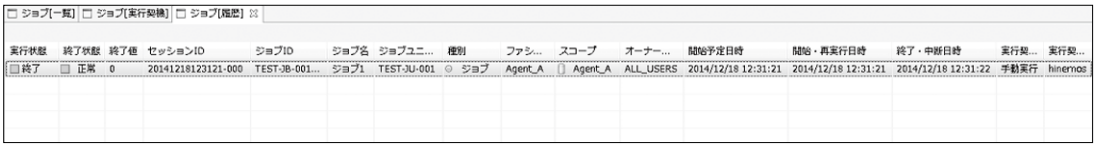
藤井「はい！　ジョブ[履歴]ビューを見てみると……しっかり動いた記録が残っています！　正常に終了していることもわかります！（**図5**）」

挑戦！ 処理の連携

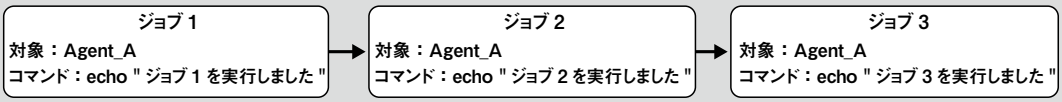
藤井「簡単に実行できました。確認も簡単です」

定時「よかったわね。次は、この前トラブルのあった処理の連携を試してみましょう。まずは1つ

▼図5　ジョブ[履歴]ビュー



▼図6　ジョブの例



のサーバで処理を連続して実行させるために**図6**のようなジョブを組んでみて」

藤井「わかりました。まずはジョブを編集するために、ジョブユニットを編集モードにするんですね」

定時「ええ。今回はすべてHinemosクライアント上で確認するために、標準出力で出してみて。ジョブ1～3の内容はほとんど同じでいいから『ジョブのコピー』をすると楽よ」

藤井「はい！　ジョブ1を右クリックして『コピー』、ジョブユニットの上で貼り付けっと。……ジョブID、ジョブ名、コマンドを変えてジョブ2、ジョブ3を作りました」

定時「それじゃ、待ち条件を設定しましょう。ジョブ2を開いて、待ち条件を追加してみて（**図7**）」

藤井「できました。同じようにジョブ3にも設定しました」

定時「ジョブを登録して、ジョブユニットを実際に動かしてみましょう」

藤井「実行できました！　すべて正常に終了して、標準入力も表示されています（**図8**）」

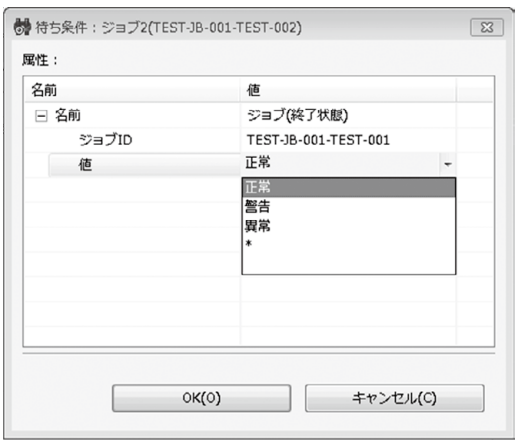
定時「[ノード詳細]ビューにもメッセージとして標準出力が出ているわね」

挑戦！ サーバを跨るジョブ実行

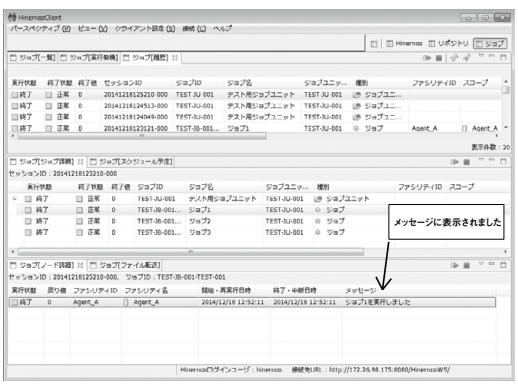
藤井「処理を順番に実行するためには待ち条件を考えればいいんですね。ジョブの流れがわかる図があるとよくわかります」

定時「そうね。今のは簡単な例だけど、複雑になっていった場合は、マップで示すことが多いし、わかりやすいわ。Hinemosでは、Hinemosジョブマップオプションを導入することで、マップで操作・確認できるわよ」

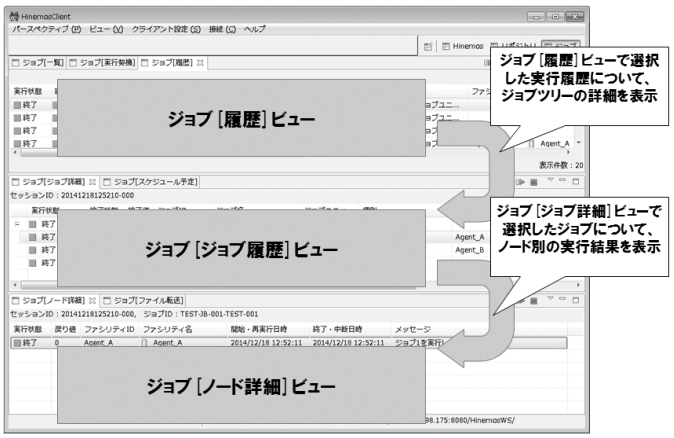
▼図7　待ち条件の指定



▼図8ジョブの実行結果



▼図9　各ビューの連携



藤井「いいなぁ、使ってみたいなぁ」

定時「そうねぇ……考えておいてあげる。ともかく、次はジョブ2をAgent_Bで実行するように変更してみましょう。Agent_BはWindows Serverだからコマンドプロンプトで実行するようにコマンドを書くのよ」

```
$ CMD /C echo ジョブ2を実行しました
```

藤井「わかりました。当然ですけど、対象のOSが変わったら、それに合うように書き換える必要がありますね」

定時「えぇ、ちゃんと意識しておきましょうね。さぁ実行してみましょう」

藤井「実行できました！　これならサーバが複数ある環境でWindowsとLinuxが混ざっていてもジョブの実行が簡単にできます！」

定時「[ジョブ詳細]ビューにも、どのノードでどのジョブが実行されたかがわかるようになっているから、次からはジョブの結果はHinemosクライアントから確認すればOKよ（**図9**）」

今月の時短ポイント

前回までは各スクリプトの結果の確認はいろいろなログに出力させ、確認していました。これからはHinemosを導入したことで、Hinemosクライアントに出力される実行結果を確認するだけでよくなり、一元管理を実現できます。また、ジョブも「スコープ（どこ）」で「実行するコマンド（何をする）」を意識すれば簡単に設定できます。次回からもHinemosのジョブの便利な使い方でさらなる時短を目指していきましょう！

次回予告

上司「藤井君、今日は順調そうだね」

藤井「はい！　Hinemosの導入にも、設定にも全然時間がかからなかったので今日は定時で帰れそうです」

上司「それはいいね。今までcronで動かしていたけど、それらをHinemosで実現するための残り項目は何だい？」

藤井「定時実行のためのスケジュールの設定ですね。それはこれからやるつもりです！」

上司「おお、仕事のスケジュールも立てられるようになって何よりだ。移行にむけた事前調査、頑張ってくれよ。はっはっはー」

今回、Hinemosによるジョブの手動実行と簡単な制御を学びました。

次回は今までcronで実現していた定時実行を、Hinemosの機能を使って実現し、ジョブの自動化をさらに進めるようです。次回「便利な機能で運用を自動化しよう」SD

To Be Continued...

サーバーワークスの瑞雲吉兆仕事術

最終回　エンジニアに求められる技術の変化

クラウドの登場によって、情報システムのあり方、そしてエンジニアのキャリアそのものが大きく変わろうとしています。本連載では、クラウドの登場とそれによって情報システムがどのように変わろうとしているのか、そしてエンジニアのキャリアがそれによってどのように変わろうとしているのか――すでに起こりつつある変化を読み取ります。皆さんが自分のキャリアを考えるための材料を提供します。

Writer　㈱サーバーワークス　代表取締役　大石 良（おおいし りょう）／ http://blog.serverworks.co.jp/ceo/

エンジニアに必須のスキルとは何か?

本連載の前半では、今までコンシューマ向けだと思われていた製品やサービスを積極的に活用することで、閉塞的だった情報システムを民主化しようという筆者たちの取り組みを「社内LAN撲滅運動」などの具体的な取り組みとともに紹介しました。最終回では、これまでの流れから、エンジニアに求められるスキルがどのように変化するのか、筆者の考えを示します。

その1「T字型の知識」

本誌2014年12月号で「アンケートシステムを作ろう」というニーズに対して「Survey Monkey[注1]を提案する」という例を示しましたが、細部を知っているかどうかはとにかく、エンジニアにとって幅広いサービスと技術を薄く広く知っていることが、とりわけ重要になります。

本稿でも再三述べているように「作ることが最悪の選択」になりつつある現在、「作る」という選択肢は「世の中に同じようなものが存在しない」か「作ることによって競争優位を確立できる」か、いずれかの場合だけ選ばれることになるでしょう。そうなると、「世の中に存在するのか否か」を知っているかどうかは、適切な設計・実装になるかどうかのキーファクターになるわけです。

「大量にメールを配信してその開封率をチェックしたい」と言われたときに、スゴ腕プログラマがゴリゴリ組んだ1万行のコードより、SendGrid[注2]を筆者は信用します。

ただし、薄く広くという横軸の知識だけでは、技術者として仕事をしていくことは難しいと思います。横軸の知識はあくまで「インデックス」です。そうしたインデックスが必要とされるのは設計フェーズ、もしくはIT戦略立案のフェーズが中心です。エンジニアとしての成功には、やはり縦軸、つまり「深掘り」された知識が絶対に欠かせません。

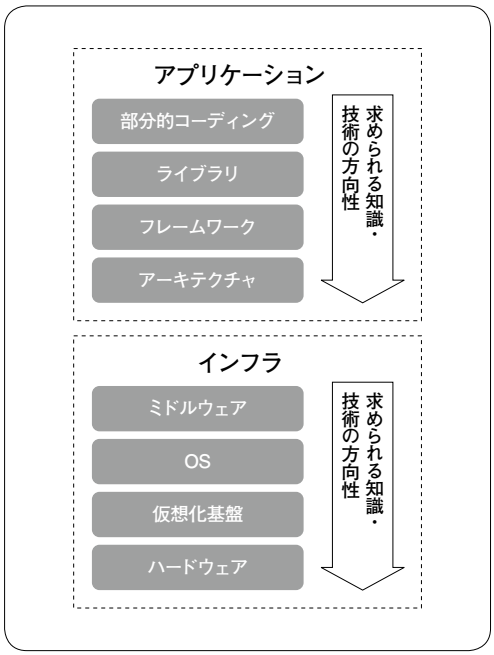
これまでのエンジニアは「I字型」の知識、つまりプログラマならJavaだけ、とか、インフラエンジニアならインフラだけ、といった知識のストラクチャが中心でした（**図1**）。これはインフラとアプリケーションがきれいなブロック型のアーキテクチャになっていて、インフラとアプリケーションの連携が限定的であったことが原因です。

ところが、クラウドの世界ではインフラがコードでコントロールできるようになったり、サービスによってはインフラそのものが隠蔽されて

注1）https://jp.surveymonkey.com/

注2）https://sendgrid.kke.co.jp/

▼図1　これまでの「I字型」エンジニア知識セット



いたりします。そうなると、インフラとコードの相互作用を理解することが重要になってきます。

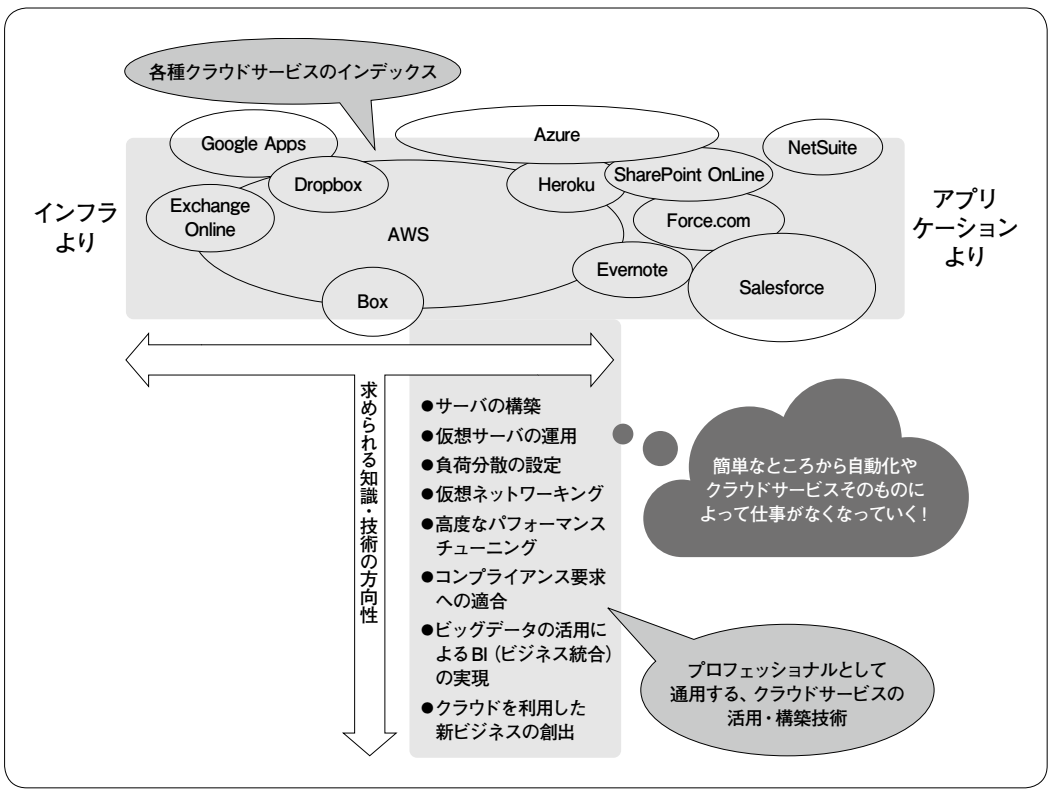
Apacheの設定をなんとなく知っているだけのなんちゃってサーバエンジニアは、Amazon S3とCloudFrontの組み合わせの前にやることはありません。

スクリプト言語でちょろっとバッチ処理が書けますというなんちゃってプログラマは、AWS Lambdaを前にして何が起きているのか理解することすら難しいかもしれません。

クラウドサービスが進化するにつれ、サービスそのものがインフラやアプリケーションの簡単な仕事をどんどん奪っていきます。そうなると、人間のやるべきことは「クラウドサービスの横連携」か「クラウドサービスによる自動化が追いつかない領域」のいずれかに絞られてきます。

筆者が「これからのエンジニアにはT字型の知識セットが必要だ(**図2**)」と言っているのは、こうした理由によります。

▼図2　「T字」型の知識セット(AWSの場合)



その2「説明能力」

もうひとつ、クラウドには特有の難しさがあります。それは「意志決定者に、正しく、理解可能な表現で伝えなくてはいけない」ということです。

たとえばあなたがSEで、提案先の(それほどAWSに明るくない)お客さまがストレージの採用を検討しているとして、あなたが「これはAmazon S3が適切だ」と考えたとします。そのときに、次のいずれの説明が適切でしょうか？

説明1 Amazon S3はHTTP(S)でアクセスできるオブジェクトストレージで、99.999999999%の耐久性を誇っています。ブロックストレージではないことからそのままファイルシステムとしてマウントしたりすることはできませんが、Storage Gatewayを利用することでiSCSIボリュームをS3上に作成することができ、このボリュームを仮想サーバからマウントすることでNASとしての利用も可能です。

説明2 Amazon S3は、容量の制限なく利用できるクラウド型ストレージです。書き込まれたファイルは自動的に3ヵ所のデータセンターに複製されますので、非常に高い堅牢性を保っています。S3そのままではファイル共有には利用できませんが、AWSの別サービスと組み合わせることで、複数人でアクセスするファイルサーバのように利用することも可能です。

どちらの説明も、技術的な誤りはありません。文脈によっては「説明1」が適切なケースもあるでしょう。ですが、ほとんどのケースで「説明2」のほうが、お客さまが選択しやすく、また正しく理解してくださるものと思います。

この問題は非常にやっかいです。**AWSは、どこから買っても、AWSなのです！**

それにもかかわらず、説明する側の説明能力によって、ユーザに適切な判断がなされるかなされないかが決まってしまう。本来クラウドを使うべきところで適切な説明がなされないために、クラウド以外の選択肢が選ばれたり、最悪の「作る」という選択肢が選ばれたりしてしまう。そうした不幸が起きてしまうわけです。

技術的に正しいと思っていても、それを理解可能な形で説明できなければ導入に至らず、設計や実装がそれに引きずられてしまうということが起きかねないのです。

こうした理由で、エンジニアといえども、意志決定者が理解可能な表現で正しくクラウドの価値を伝えることがとても重要になってくるのです。

その3「コーディング能力」

もはや説明は不要でしょう。インフラもコードで操作する時代です。クラウドの世界では、コードが書けなければインフラを触れないことと同義です。

その4「自制心」

これは半分冗談ですが(笑)半分は本気です。クラウド時代のエンジニアは「作りたい」というエンジニアが持つ根源的な欲求と戦い、自制心を持ってそれをコントロールしなければいけません。

筆者たちも、2007年に初めてAWSに出会ったとき、「コピーを作りたい！」という衝動に実は駆られました。まだSimpleDB、S3、EC2くらいしかありませんでしたから、ベンチャーキャピタルから相応のお金を集めればコピーができるかもしれない、と考えていたのです。

その後、冷静に分析を進めるにつれ、「これ

COLUMN　説明・プレゼン能力を磨くには何が必要か

筆者たちは、説明能力・プレゼン能力が、クラウドインテグレータで働くエンジニアとして極めて重要であるとかなり早い段階から考えていました。そのため、セールスだけでなくエンジニアもプレゼン能力のスキル向上に取り組んでいます。

その中でもとくに成功している取り組みが「金曜LT（Lightning Talk）」です。

これは、社内から2名のエンジニアを選んで、金曜日の夕方に10分間のプレゼンを行うというものです。プレゼンはUstreamで配信されていますので、当社の大阪や福岡、仙台のメンバーも見ています。場合によっては大阪からリモートLTを行う場合もあります。そして、見ているメンバーはSlackで「体が揺れている」とか「画面を見過ぎている」などのツッコミを入れます。終わると投票です。必ず、どちらか一方に投票しなければいけないルールです。

この取り組みのポイントは2つあります。1つは「リアルタイムにフィードバックする」ということ。まず、フィードバック自体がとても大切です。それがないまま100回プレゼンをやるより、フィードバックのある10回のプレゼンのほうが確実に上達します。そしてフィードバックも「後でまとめて」ではなく「リアルタイム」がベターです。まとめてフィードバックすると、具体性に欠ける指摘になりがちですが、リアルタイムのフィードバックですと「このスライドはもっと文字が大きいほうが良い」など、具体的な指摘になるからです。

そしてもう1つは「Ustreamで配信している」という点です。USTで流すことになると、お客さまの情報やプロジェクト固有の情報がプレゼンに入れられません。ですから、技術の話をする場合でも、きちんと抽象化して、再利用可能なコンテンツにしてプレゼンしなくてはいけません。プロジェクト固有の話のほうが内輪にはウケますが、それでは説明能力の向上にはつながりません。あえてUST配信することによって、メタ情報をくみ取る能力の向上につなげているのです。

この取り組みの効果は抜群で、最初はプレゼンが苦手だったエンジニアでも、何回かこれをやれば確実に上達します。プレゼン能力の向上に関心がある方は、ぜひ試してみてください。

は壮大な戦略と、とんでもない規模で周到に準備されたものだ」ということがわかり、「追いつく見込みがないものをムリに作るよりも、徹底して使う側に回ろう」という判断に至りました。この判断は――少なくとも今の時点では――正しかったものと考えています。

この姿勢について、DevLOVE代表の市谷さんがとても適切な表現を教えてくれました。それは「Outputより Outcome」というものです。これまでのエンジニアは作ることが前提でしたから、当然仕事の結果としてコードやドキュメントなどのOutputがあったわけです。ところが、前に例示したSurveyMonkeyの話にOutputはありません。Outputはありませんが、大きなOutcome（成果）は得られたわけです。

これが「これからのエンジニアはOutputを出す欲求と戦う自制心が必要だ」という理屈です。

まとめ

いかがでしたでしょうか？　全6回で筆者たちが経験してきたクラウドとの出会いから、筆者たち自身がクラウド専業のインテグレータになる道筋を伝えるとともに、筆者たちの身に起きたことを通じて、これからのエンジニアにはどのような技術が求められるのか、どうやって自分のキャリアを構築していくのか、筆者の考えを示しました。

もちろん、ここで述べたことは1人の意見でしかありません。私が述べたことを信じるも信じないも、みなさんの選択しだいです。

ですが、1つだけ確実に言えることがあります。それは「この流れは止められない」ということです。AWSが毎年ラスベガスで行っているカンファレンスre:Inventで、AWS事業のトップAndy Jassy氏がこのようなことを言っていました。

「クラウドは重力のようなもので、逆らうことはできない大きな流れである。それであれば、私は流れを作るほうに回りたい。そんな想いでAWSを始めたのだ」

ほぼすべての大手ITベンダーがクラウドに大きなリソースを割いている以上、これからのイノベーションはクラウドが中心になることは間違いありません。こうした流れに逆らうのではなく、積極的に先頭に立ってみませんか？

それこそが、これからのエンジニアが変化の激しい業界で生き残るためにベストな生存戦略だと思います。

一緒にこの波に乗って、自分自身の手で未来を作っていきましょう！（完）SD

㈱サーバーワークス
代表取締役
大石 良（おおいし りょう）

- 昭和48年7月20日新潟市生まれ
- コンピュータとの出会いは10歳の頃・当時はPC-8001にベーマガのプログラムを入力する日々
- コンピュータの購入は11歳／SHARP X1
- 中2の時に初めてプログラムが書籍に掲載
- 高校入学記念にX68000を購入
- 大学生の時にパソコン通信開始。本格的にシェアウェアを販売
- 総合商社でインターネットサービスプロバイダー事業に携わる
- 2000年にECのASPを立ち上げるべく起業

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9回 すぐに使える! 便利な11のパッケージ

ファイル・バッファ切り替え編、カーソル移動編、編集編に分けて、それぞれの操作が格段に便利になる定番パッケージを紹介します。気に入ったモノがあれば、設定してご自分の環境をカスタマイズしてください。

便利なパッケージたち

ども、るびきちです。今回は日常的に使える便利なパッケージたちを紹介していきます。

今ではここで紹介しているパッケージよりも強力なインターフェースが存在しますが、物事には順序があります。それを使うにはEmacsの定番機能を知ってからでも遅くはありません。もちろん本連載でも取り上げますので、楽しみにしていてください。

ファイル・バッファ切り替え

本節ではファイルとバッファの切り替えをカイゼンするパッケージを紹介します。おもに標準添付のパッケージですが、知るだけで使い勝手が劇的によくなります。

ido：バッファ・ファイルを選択して開く

バッファを切り替えるには`C-x b`を使いますが、使い勝手はイマイチです。バッファ名入力が面倒だからです。確かに昔よりは便利になっています。もともとは先頭一致でしかバッファ名を選択できませんでしたが、今は部分文字列が使えます。ただし、バッファ名の先頭が一致するとそれが優先されます。たとえば、*scratch*バッファを選択するのに「`rat`」と入力して tab キーを押せば「*scratch*」と補完されます。しかし、ratpoisonというバッファがあるときはそれで確定されてしまいます。

`C-x C-f`でファイルを開く場合もデフォルトではファイル名の先頭から入力する必要があります。

idoはバッファ切り替え・ファイル名入力をカイゼンします(**リスト1**、**図1**)。`C-x b`や`C-x C-f`で入力時に文字列をタイプすると、候補が`C-s`と`C-r`で選択できるようになります。部分文字列を数文字タイプするだけで数個にしぼれてきます。

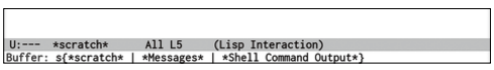
また、idoではバッファ・ファイルを連続して削除する機能もあります。idoではRETキーを押すことで選択しますが、RETキーの代わりに`C-k`を押せばそのバッファ・ファイルを削除します。一連の削除が終わりましたら、`C-g`を押してください。もはや`C-x k`で別なバッファを削除する必要がなくなります。

さらに、ido内で`C-x C-f`を押すとidoをとりやめてファイルを開くことができます。バッファを選択しようとしたけど存在しなかった場合、`C-g`でキャンセルすることなく、流れるように

▼リスト1 idoの初期設定

```
(ido-mode 1)
(ido-everywhere 1)
```

▼図1 idoでバッファ選択



望みのファイルを開けるのです。

bs：手軽にバッファ選択

C-x C-b(list-buffers)は、バッファ一覧を表示しますが、そこから選択するにはわざわざそのウィンドウにフォーカスする必要があります。Emacsの初期からあるこのコマンドは、極めて原始的なインターフェースです。そこで、この操作性を改善するいくつものパッケージが登場してきました。

今回紹介する標準添付のbsもそのひとつです(**図2**)。M-x bs-showで起動します。ですが、元のM-x list-buffersを完全に置き換えるので、C-x C-bに割り当ててしまいましょう(**リスト2**)。

M-x bs-showを起動すると、*buffer-selection*バッファが適切に調節された高さでポップアップします。おかげさまで、バッファリストの使用目的に合わせてほどよい操作性を実現しています。バッファリストの目的は大きく分けて3つです。

・どんなバッファが開かれているかを見る
・バッファを選択する
・バッファを連続して削除する

バッファリストを閲覧するだけならば、見終わったらC-gで閉じられます。C-gはもともと中止を意味するコマンドですので、この割り当ては直感的です。バッファを選択するのは、普通にRETキーです。バッファを削除するのはdを押すだけで、わざわざ質問したりはしません。

M-x bs-showはもともとファイルと関連付けられたバッファしか表示しませんが、「C」(大文字のc)を押してから表示されるミニバッファにallと入力すれば隠しバッファ以外のすべてのバッファが表示されます。

ffap：C-x C-fに機能を追加する

ffapは、C-x C-fを強化する標準パッケージです(**リスト3**)。ffapとは「Find File At Point」の略で、カーソル位置のファイル名を認識して、ファイルを開くときのデフォルトにしてくれます。もしカーソル位置がファイル名ならばC-x C-f RETでそのファイルが開けます(**図3**)。

また、URLにも対応していて、同じ方法でカーソル位置のURLをもブラウザで開けます。開きたいURLを直接入力することもできます。

また、ffapはファイルの存在確認にも使えます。もしカーソル位置のファイルが存在するときはffapのデフォルトになりますが、存在しないときはデフォルトにはなりません。存在を確認したらC-gで中断してください。

ffapにより、C-x C-fはファイルを開くだけでなく、

・URLを開く
・カーソル位置のファイル・URLを開く
・ファイルの存在確認

と多目的に働けるようになりました。ぜひとも入れておきたい設定です。

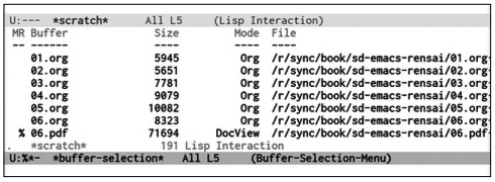
recentf-ext：最近使ったファイルを開く

最近使ったファイルというのは、再び使われる可能性が高いです。recentfは、最近開いた

▼リスト2 C-x C-bを置き換える

```
(global-set-key (kbd "C-x C-b") 'bs-show)
```

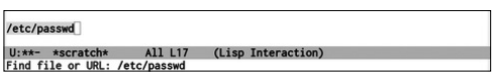
▼図2 M-x bs-show



▼リスト3 ffapの初期設定

```
(ffap-bindings)
```

▼図3 C-x C-fで現在位置のファイルがデフォルトに



ファイルを開きやすくする標準パッケージです。しかも、最近開いたファイルのリストはファイルに保存され、Emacs再起動時に復元されます。

ファイルを開くときに、もう長ったらしいフルパスを入力する必要はありません。最近開いたファイルであれば、C-x C-fを使う必要はないのです。

M-x recentf-open-filesは、最近開いたファイル一覧をバッファに表示します(**図4**)。直近10個までのファイルならば、そのあとに番号を押せばそのまま開けます。もし、そこから見つからなくてもisearchを使えばすばやく目的のファイルを探し出せます。GUIのダイアログボックスとは違い、Emacsのバッファは検索できるのがうれしいですね。

拙作**recentf-ext**を導入すればさらにパワーアップします。recentfが扱うのは、厳密には「最近使ったファイル」ではなく「最近開いたファイル」です。Emacsを長時間起動していれば、最初に開いたファイルに再びアクセスしても、奥のほうに隠れてしまいます。recentf-extでは、そのファイルバッファを表示した時点でrecentfの先頭に持っていくので、「最近使ったファイル」としてアクセスしやすくなります。

また、オリジナルのrecentfではディレクトリ(dired)を除外していますが、recentf-extではディレクトリも「最近使ったファイル」として扱えるようにします。

どれくらいのファイルを記憶するかはrecentf-max-saved-itemsで制御できますが(**リスト4**)、デフォルト値の20は少な過ぎて本領発揮できません。検索することも考えれば100や500など大きめの値にすればいいです。

▼図4 M-x recentf-open-files

```
Click on a file, or type the corresponding digit key, to open it.
Click on Cancel or type 'q' to cancel.

 [1] /r/sync/emacs/init.d/50persistent-timer.el
 [2] /r/sync/memo/mokuhyo.org
 [3] /r/sync/memo/daily-projects.org
 [4] /r/sync/memo/wp/rubikitch/1411150557.helm-cmd-t.org
```

なお、recentf-extはMELPA(**リスト5**)から次のようにインストールできます。

```
M-x package-refresh-contents
M-x package-install recentf-ext
```

カーソル移動

カーソル移動を強化するパッケージたちを紹介します。これらは筆者も現役で使っているものばかりです。

point-undo：カーソル位置を戻す

Emacsを使っていて、迷子になったことはありますか？　つまり、操作ミスによりカーソル位置が意図しない場所に向かってしまったことです。筆者もたまにあります。あわてふためく前にワンタッチでカーソル位置を戻せたらいいですね。そこで拙作**point-undo**を次のようにインストールして使ってみてください(**リスト6**)。

```
M-x package-refresh-contents
M-x package-install point-undo
```

確かにisearchやM-<といった大域移動コマンドは自動でマークされるためC-u C-SPCで戻れますが、そうじゃないコマンドもあります。M-x point-undoは、自動マークとは関係なく働いてくれます。

▼リスト4　recentfの設定

```
;; 最近のファイル500個を保存する
(setq recentf-max-saved-items 500)
;; 最近使ったファイルに加えないファイルを
;; 正規表現で指定する
(setq recentf-exclude
      '("/TAGS$" "/var/tmp/"))
(require 'recentf-ext)
```

▼リスト5　パッケージを使うための初期設定

```
(package-initialize)
(add-to-list 'package-archives '("marmalade" . "http://marmalade-repo.org/packages/"))
(add-to-list 'package-archives '("melpa" . "http://melpa.org/packages/") t)
```

また、同じカーソル位置であってもウィンドウの表示範囲が異なっていると、別な場所に移動したような錯覚になります。そこでカーソル位置だけでなくウィンドウ表示開始位置も記憶しているので、画面上の見た目も復元されます。

M-x point-redoはM-x point-undoで戻し過ぎたカーソル位置を修正できますので、戻し過ぎてしまっても安心です。**リスト6**で、それぞれ f7 とM-<f7>に割り当てています(M-<F7>でOSのコマンドが優先実行される場合は、適宜設定を変更してください)。

goto-chg:編集個所の履歴をたどる

テキストエディタは、カーソルをいろいろな場所に移動して書き換えます。よくある挙動として、ほかの場所に移動したあと元の位置に戻ったり、別な場所を編集してから元の場所を編集したりします。たとえば文章を書いていて、ふと誤字脱字が目について、そこを修正したあと、再び元の位置に戻って文章の続きを書くといったケースです。そんなときに便利なのが**goto-chg**です(**リスト7**)。インストール方法は次のとおりです。

```
M-x package-refresh-contents
M-x package-install goto-chg
```

マークを多用している人ならば、そういうときはC-SPCでマークして、C-u C-SPCで戻るでしょう。しかし、Emacsはどこを書き換えたかという足跡情報をすべて記憶しているので、その必要はありません。

M-x goto-last-changeは、変更履歴という足跡をたどり、過去の変更個所にカーソルを移動します。戻り過ぎたときはM-x goto-last-change-reverseで戻します。

前項のpoint-undoはすべてのカーソル移動を追跡しているのに対し、goto-chgは変更個所を追跡します。よって、編集位置をたどることに関してはgoto-chgのほうが早く目的の位置に到達します。両者をうまく使い分けましょう。**リスト7**でM-x point-undoとM-x point-redoは、それぞれ f8 とM-<f8>に割り当てています(M-<F8>でOSのコマンドが優先実行される場合は、適宜設定を変更してください)。

bm:現在位置をハイライト付きで永続的に記憶させる

バッファの現在位置を記憶する標準的な手法はマークとレジスタです。しかし、マークした位置は目に見えません。レジスタもいまいち操作性がよくありません。そこで**bm**を使うと記憶した行がハイライトされてわかりやすくなります(**リスト8**)。しかも、Emacsを終了しても再起動時に記憶した位置を復元できます。筆者も長らくお世話になっているパッケージです。

次のようにインストールしましょう。

```
M-x package-refresh-contents
M-x package-install bm
```

▼リスト6 point-undo初期設定

```
(require 'point-undo)
(global-set-key [f7] 'point-undo)
(global-set-key [M-f7] 'point-redo)
```

▼リスト7 goto-chg初期設定

```
(require 'goto-chg)
(global-set-key [f8] 'goto-last-change)
(global-set-key [M-f8] 'goto-last-change-reverse)
```

▼リスト8 bm初期設定

```
(setq-default bm-buffer-persistence nil) (setq bm-restore-repository-on-load t) (require 'bm)
(add-hook 'find-file-hook 'bm-buffer-restore) (add-hook 'kill-buffer-hook 'bm-buffer-save)
(add-hook 'after-save-hook 'bm-buffer-save) (add-hook 'after-revert-hook 'bm-buffer-restore)
(add-hook 'vc-before-checkin-hook 'bm-buffer-save) (add-hook 'kill-emacs-hook '(lambda nil
(bm-buffer-save-all)
(bm-repository-save))) (global-set-key (kbd "M-SPC") 'bm-toggle) (global-set-key (kbd "M-[")
'bm-previous) (global-set-key (kbd "M-]") 'bm-next)
```

使い方は簡単で、M-x bm-toggleでbmを付けたり解除したりします(**図5**)。

M-x bm-previousとM-x bm-nextでbm間を移動します。基本的にはこれらのコマンドだけで便利です。それぞれM-SPC、M-[、M-]に割り当てています。

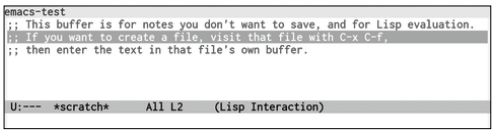
M-x bm-showでカレントバッファでのbm一覧(**図6**)、M-x bm-show-allで全バッファでのbm一覧を表示します。RETキーでその位置に移動し、qで一覧を閉じます。

imenu：見出し・関数定義にジャンプする

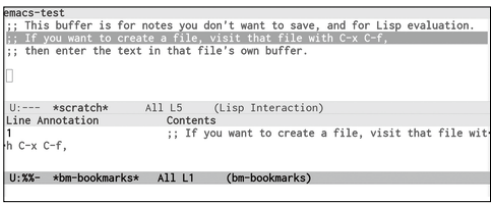
構造化されたテキストファイルは見出し行を見るだけで全体像がわかります。見出し行とは、そのファイルに目次を作るときに載せる行のことです。プログラミング言語の場合は関数(クラス・メソッド)定義、マークアップ言語の場合は見出しです。

関数定義に移動したいために関数名をisearchしようとすると、関数呼び出しなどにも止まってしまいます。そこでM-x imenuを実行するとバッファの目次を作成し、そこで関数名や見出しを指定すれば見出し行にすばやくジャンプします(**図7**)。もちろんTABキーによる補完も効きます。**imenu**はEmacs標準コマンドであるうえ、あらゆるメジャーモードに対応しているので役立つ機会はとても多いです。

▼図5　現在位置を目に見えるようにマーク



▼図6　M-x bm-showでbm一覧



▼リスト9　visual-regexp初期設定

```
(global-set-key (kbd "M-%") 'vr/query-replace)
```

編集

ここでは基本的な編集コマンドを強化するパッケージを紹介します。

visual-regexp：正規表現置換を対話的に行う

正規表現置換は強力な編集機能ですが、扱いが難しいです。代替案として10月号ではM-x occurの結果を編集することで正規表現置換と同等のことができることを紹介しました。置換される行を絞り込んでから編集するので、安心できるというメリットもあります。

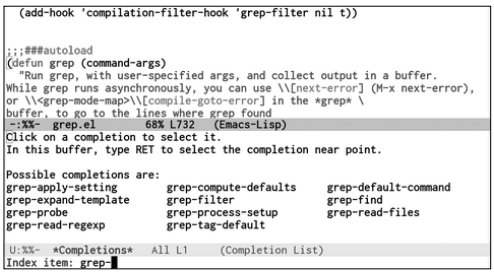
visual-regexpはこれとはアプローチを変えまして、正規表現置換を対話的に行うのが狙いです(**リスト9**)。

```
M-x package-refresh-contents
M-x package-install visual-regexp
```

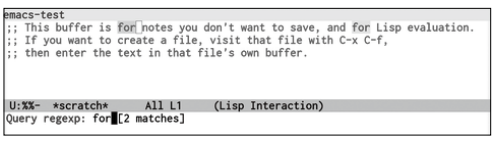
と、インストールします。置換のための正規表現を組み立てるときに、視覚的なフィードバックがあると劇的に使いやすくなります(**図8**)。

標準コマンドのM-x re-builderは対話的に

▼図7　M-x imenuを実行し候補を一覧表示させる



▼図8　正規表現入力時点でハイライト



正規表現を組み立て、あとの正規表現置換やEmacs Lispプログラミングに活かすのですが、visual-regexpはre-builderと置換を合体させたものです。**リスト9**でM-x vr/query-replaceをM-%に割り当てると手軽に正規表現置換の恩恵を受けられます。

rectangle-mark-mode：Emacs 24.4の新矩形編集

Emacs 24.4では矩形編集がとても使いやすくなりました。C-x SPCは矩形選択コマンドです(**図9**)。このあとにC-w、M-w、C-yを使うと、矩形のカット／コピー&ペーストになります。もう煩わしい旧来の矩形コマンドを無理して使う必要はありません。

C-x SPCを押すタイミングは2とおりあります。マーク開始時点かマーク終了時点です。C-x SPCをC-SPCの代わりに使うと、regionは矩形方向に広がります。C-SPCでマークしたあとにregionを設定してC-x SPCを押すと、その間の矩形を選択してくれます。そのため、使い勝手はかなり良いです。

C-x SPCの後でC-oを押すと、その矩形の部分を空白にしてくれます。C-x r oと同じです。

C-x SPCのあとでC-tを押すと、矩形の各行を同じ文字列で置き換えます。とくに横方向がゼロの場合、同じ文字列を各行に挿入することになります。筆者は旧来のコマンドC-x r tで多用しています。

これはEmacs 24.1から使えますが、C-x r Nは各行に番号を付けるコマンドです。C-u C-x r Nでは開始番号と書式を指定できるので、こちらのほうが便利でしょう。

browse-kill-ring：kill-ringをフル活用する

EmacsではC-wでカット、M-wでコピー、C-yでペーストができます。Emacsでのコピー&ペーストは現在のGUIのそれよりもはるかに進化しています。なぜなら、Emacsでのクリップボードに相当するkill-ringには、複数の文字列を記憶できるからです。実際、C-yの直後にM-yを1回以上押すと、過去に保存したテキストを呼び起こせます。

しかし、M-yでは過去にkill-ringに保存したテキストを一覧できません。過去60個(kill-ring-max変数で調節可能)ものテキストを記憶できるのに、もったいないです。そこで、**browse-kill-ring**でkill-ringのテキストを一覧・選択できるようにしましょう(**リスト10**)。

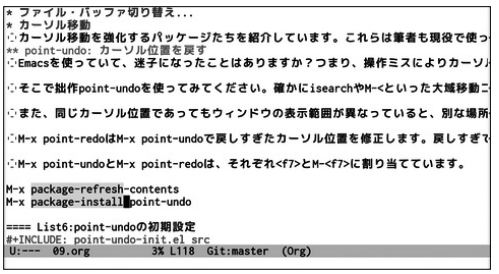
```
M-x package-refresh-contents
M-x package-install browse-kill-ring
```

でインストールしてから、M-x browse-kill-ringを実行すると、*Kill Ring*バッファにkill-ring一覧が表示されていて、pとnで選択できます(**図10**)。RETキーで貼り付け、qで中止です。一覧はバッファですので、isearchで目的のテキストにすばやく到達できます。**リスト10**でM-yを置き換えましょう。SD

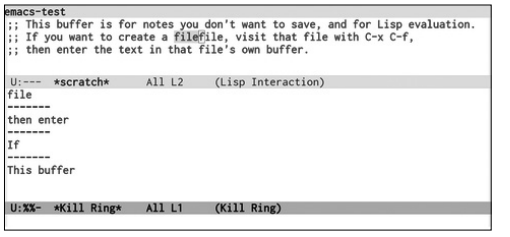
▼リスト10　browse-kill-ring初期設定

```
(global-set-key (kbd "M-y") 'browse-kill-ring)
```

▼図9　C-x SPCで矩形選択

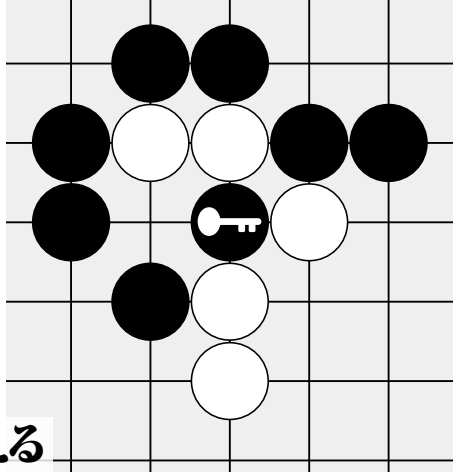


▼図10　kill-ringの履歴を表示



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

【第十六回】IoTのセキュリティについて考える

ここ数回の本連載ではOpenSSLやbashの脆弱性など緊急性が高く、すぐにでも解決しなければいけない目の前の問題を取り上げました。今回はもっと先の、つまり将来課題となるようなテーマとしてInternet of Thingsのセキュリティについて考えます。

Internet of Thingsとは

Internet of Things（IoT）とは、最近バズワード化しているきらいがありますが、もともとは「これまでPCやサーバといったコンピュータ類だけがつながっていたインターネットに、多種多様な機能やサービスを持った機材がつながれていくこと」を意味します。

たとえば、世の中にたくさんセンサーを用意し、それをインターネットで接続するのもIoTですし、我々が普段使っている家電などをインターネットにつなぐこともIoTです。あるいは、家庭のガスメータや電力メータをインターネットにつなぐのもIoTです。

IoTはインターネットの新しい使い方であり、インターネットの可能性の拡大であり、インターネットの新しい革命だと言われています。これらについての論文や報告書、あるいは宣伝は世にあふれています。

しかし、一歩後ろに引いて見たときに確実に言えるのは、「これまでインターネットに接続していなかったものが接続し始めるということは、同時にインターネットの新しいセキュリティを考えなくてはいけない」ということです。

筆者がACM（Association for Computing Machinery）[注1]のACM Digital Libraryで“Internet of Things”で検索してみたところ、3,877件の論文がヒットしました。ACMは2012年からはSecurITという国際会議を開催しており、学術系でも本格的にIoTのセキュリティについての議論が始まっている様子がうかがえます。

事件は現場で起こっている

IoTという言葉が広まる前から、インターネット接続の家電はすでに登場していました。そして、やはりセキュリティ上の問題を抱えていました。

今から10年前の2004年に、東芝のハードディスクレコーダRDシリーズに匿名HTTPプロキシサーバとして使える問題があることがわかりました。実際に、インターネット側からアクセスできる状態にしていたハードディスクレコーダが踏み台になり、ブログに大量のスパムコメントが書き込まれるという事件が起こりました[注2]。これは世界的にみても初めてのケースで、この報告は後に各国のCSIRTが参加している団体FIRSTが主催する国際会議でも発表されました[注3]。

注1）世界で最も大きいコンピュータ学会。www.acm.org
注2）「インターネットセキュリティの歴史 第23回『ハードディスクレコーダを踏み台にしたコメントスパム』」
https://www.jpcert.or.jp/tips/2008/wr084701.html
注3）Keisuke Kamata & Masaki Kubo,JPCERT/CC,Japan “Vulnerabilities in Consumer Electrics -- DVD Players, Cell Phones Attack Your System?”
http://www.first.org/conference/2005/schedule.html

また、インターネットに接続するタイプの監視カメラも、外部から乗っ取られる脆弱性が何度も発見されています。2012年のことですが、おもにアメリカで発売されているIPカメラ（Foscam社製およびWansview社製）には制御用のWebインターフェースに認証を回避できる脆弱性があり、外部からコントロールを乗っ取ることができました。すでにお馴染みの危険度を示すファクタであるCVSS v2の値も含め、関連する危険度を示す数値はすべて10.0という最高値をつけられています[注4]。

このIP接続可能なカメラをネットワーク的に無防備な状態、つまり第三者がネットワーク経由で自由にアクセスできる場所に接続していたならば、簡単に操作されてしまう、ということです。その第三者はカメラがとらえる映像を自由に入手できてしまいます。

映画やテレビで、街角にあるカメラを自由に操れる天才ハッカーといった話が出てきますが、実際には天才でもなんでもなく、何の才能もいらず、PCを扱えるスキルさえあれば、同様のことができます。インターネット上のあちらこちらの掲示板に懇切丁寧に書いてある手順どおりにやればいいだけのレベルです。

これは海外だから、という話でもありません。日本国内でも㈱アイ・オー・データ機器製のネットワークカメラ「Qwatch（クウォッチ）」シリーズにおいて認証を回避できる脆弱性が報告されています。こちらのCVSS v2の値は6.4ですので、警告レベルです[注5]。もし該当の機種をお持ちであれば、ファームウェアを最新版へバージョンアップすることをお勧めします。詳しくはアイ・オー・データ機器社の告知ページをご覧ください[注6]。

これらはサポートするベンダがはっきりしているから、まだ良いほうです。問題は、ベンダ名も聞いたことがない、サポート先もさっぱりわからない、だけれども価格的に魅力的な格安製品をあちこちの店頭で見かけることです。もうこうなれば内部で何をやっているかは誰にもわからず、サポートもされず、完全にお手上げです。

冷蔵庫は無実

2014年1月、イギリスBBCが「冷蔵庫がインターネット経由で大量のスパムメールを送っていた」という報道をしたため[注7]、「とうとう冷蔵庫までもか」ということであちらこちらのニュースサイトで話題になりました。

これはProofpoint社から出されたリリースをベースとして報道された内容です。そのリリース内容とは、2013年12月23日から2014年1月6日までの期間に10万台のコンシューマ機器からスパムメールが総計75万通送られていたことを同社が検知し、分析した結果、その中の1台は冷蔵庫であった、というものです[注8]。

ハードディスクレコーダですら匿名HTTPプロキシサーバとして使われてしまう時代ですから、冷蔵庫がスパムメールの踏み台になってしまうのも、それほど不思議なことではありません。ただ筆者はその機種に脆弱性があるのに、なぜかその1台だけが使われているという不自然さが気になりました。

後に、Symantec社から冷蔵庫からスパムが送られたというのは誤りで、Windows経由の典型的なボットネットであるという分析が公開されました[注9]。

注4）National Cyber Awareness System "Vulnerability Summary for CVE-2012-3002"（2013年1月3日）
http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-3002
注5）「JVNDB-2014-000087　アイ・オー・データ機器製の複数のIPカメラにおける認証回避の脆弱性」（2014年8月1日）
http://jvndb.jvn.jp/ja/contents/2014/JVNDB-2014-000087.html
注6）「ネットワークカメラ「Qwatch（クウォッチ）」シリーズご愛用のお客様へお知らせ」（2014年7月29日）
http://www.iodata.jp/support/information/2014/qwatch/
注7）"Fridge sends spam emails as attack hits smart gadgets"（2014年1月17日）
http://www.bbc.com/news/technology-25780908
注8）"Proofpoint Uncovers Internet of Things (IoT) Cyberattack"（2014年1月16日）
http://www.proofpoint.com/about-us/press-releases/01162014.php
注9）「冷蔵庫によるスパム送信は誤報」（2014年1月27日）
http://www.symantec.com/connect/ja/blogs-334

家庭内の同一ネットワークにあったPCが原因だったわけです。通常、家庭内からのインターネット接続はいったんNATルータに収容されます。そこから外部のインターネットには1つのIPアドレスを持つものとして見えますから、もしIPアドレスだけを頼りに判断すると、NATルータ以下が1つのものと見えてしまいます。

組込みデバイスでのマルウェア

以前は、組込みデバイスといえばコスト的に最小限の資源しか持てなかったためハードウェア資源が乏しく、スクリプト言語(インタプリタ言語)を処理系に使ったWebインターフェースを持つのは難しいことでした。しかし今では、ハードウェア資源がどんどんと低価格化し、ついこの間までインターネットに接続されて動作していたサーバと同じレベルの資源を積んで動作するようになってきています。

組込みデバイスでもWebインターフェースの処理系にPHPを搭載しているものも増えているようです。そんな状況で、PHPの脆弱性をついてくるワームLinux.Darllozが現れています[注10]。

これはPHPの持つ複数の脆弱性(CVE-2012-1823、CVE-2012-2311、CVE-2012-2335、CVE-2012-2336)をついてくるワームです。組込みデバイスではIntelのCPUアーキテクチャ以外にARM、PPC、MIPSなども使われていますが、LinuxにPHPさえ搭載されていれば、これらのCPUアーキテクチャの違いに関係なく感染します。

もちろんPHPだからダメだということではなく、これがRubyであろうとPythonであろうと脆弱性があれば同じことになります。これらの抽象度の高い言語は、プログラマにハードウェアのアーキテクチャを意識させることなく柔軟なプログラミング表現を提供するところに、その価値があります。しかし、その利点にいったんほころびができれば、これまでのバイナリーコードであったような「CPUが違えば感染しようがない」という壁がなくなります。ハードウェアの性能の向上、および低価格化は、いい意味でも、悪い意味でも大きな影響を与えていることがわかります。この方向性は、今後はさらに加速度的に進むでしょう。

Raspberry Pi

最近はRaspberry Piのようなカードサイズのコンピュータ・プラットフォームが現れています。Raspberry Piは、32ビットARMアーキテクチャベースのCPUコアやGPU、その他必要な機能を1つのチップに収めたBroadcom BCM2835システムオンチップ(SoC)を使ったカードサイズで動く本格的なコンピュータです。メモリは256MB、もしくは512MB、外部記憶装置にはSDカード、100Mbps Ethernet、HDMIビデオ出力、そしてUSBインターフェースを持っています。価格は数千円です。

今でこそ小さく安価なコンピュータシステムにしか感じませんが、これが20年前だと、ギガ単位容量の外部記憶装置、512MBの主メモリ、100Mbpsのネットワーク・インターフェース、そして解像度が1920×1200のフルカラーのビデオ出力を持つ最高級クラスのワークステーション・コンピュータに匹敵します。

OSもGNU/LinuxのDebianベース・ディストリビューションであるRaspbianや、同じくGNU/LinuxのFedoraベース・ディストリビューションのPIDORAなどが用意されていて、その上には通常のDebianやFedoraで扱うような各種アプリケーションが用意されています。さらにはWolfram Research社の数式処理システムMathematicaも用意されているなど、既存のGNU/Linuxシステムと遜色のない利用が可能です。

もちろんUNIXベースですので、ユーザの使うデスクトップシステムとして使うだけではなく、サーバマシンとしても申し分なく使えます。I²CやGPIOといった外部インターフェースも用意されているの

注10) Linux.Darlloz http://www.symantec.com/ja/jp/security_response/writeup.jsp?docid=2013-112710-1612-99

で、電力消費の少なさと、カードサイズという利点を活かして組込みシステムやセンサーシステムのプロトタイプや実験のプラットフォームとしても活用可能です。

Raspberry Piは将来現れるIoTのプロトタイプとして、現在一番身近にとらえることができるコンピュータシステムです。最近ではIntelでも、Raspberry Piより若干大きめのサイズで、32ビット・デュアルコアAtom CPUをベースにしたSoC、主記憶に1GB LPDDR3、外部記憶装置に4GB EMMCを搭載したシステムをアナウンスしています。このスペックは80年代のスーパーコンピュータレベルだと言っても、けっして言い過ぎではないでしょう。近い将来、このスペックのコンピュータがIoTとして大量に入ってくるわけです。

さて、Raspberry Piのような低価格のプラットフォームをIoTのベースシステムとして使っている実験記事があちらこちらに紹介されており、そのこと自体は、さほど珍しいものではなくなっています。

しかし、筆者には非常に気になることがあります。というのも、家庭用ブロードバンドルータの設定を変更し、外部ネットワーク、つまりインターネット経由でパケットが通るようにしたうえで、家電を制御するRaspberry Piを用意し、そこにアクセスし、「これがIoTである」といった紹介をしている記事を見つけたからです。ドメイン名でアクセスするためにダイナミックDNSの利用のしかたまで説明がありました。しかも、ソフトウェアの品質は、サンプルかつ実験的なシステムとはいえ、セキュリティ的には極めて懸念を抱かざるを得ないものです。

これを実験するならばローカルネットワーク上で行うべきで、インターネット側からアクセスできる環境で行うべきではありません。

IoTデバイスの開発コストが下がると同時に、誰でも開発ができるようになり、ハードディスクレコーダやWebカメラで起こったようなことが、今度は機材の電力オン／オフといった物理的に負荷を与えるようなものでも発生するでしょう。

そうなったときに、どんな悪意を持った攻撃が起きるか。これまでのインターネット・セキュリティの歴史をふり返って考えてみれば、みなさんだいたい想像がつくのではないでしょうか。

IoT時代への提案

Windows 95/98時代の教訓

話は変わりますが、Windows 95/98にはすでにフォルダをLAN上に公開する機能がありました。ネットワークの共有設定をする必要がありますが、とくにパスワードを設定しなくても公開できるので、ネットワークにつながっているWindows 95/98のコンピュータと簡単にファイルを共有することができました。

Windows 2000は最初からネットワーク機能を搭載してフルに活用できるLAN対応のシステムでした。ファイルおよびプリンタ共有のクライアント／サーバ機能が、デフォルトで利用できるようになっており、Ethernetに接続すれば、そのままLANとして使えるというたいへん便利なものでした。

一方で、当時、インターネットにCATVやADSL経由で接続する場合、現在のようにNATやIPマスカレードのルータを介して接続するということはせず、直接、接続するケースが多くありました。CATVやADSLの端末はEthernetの口が1つあるタイプで、それは、建物や地域単位のLANに接続するようなしくみになっていました。

家庭内でインターネット接続する機材はせいぜいPCが1台ある程度です。家庭内で複数のコンピュータを使っていて、相互にネットワークで接続し、さらにNAT機能を持ったルータを経由してインターネットに接続するということも、あまりありませんでした。そのため、たった1台のコンピュータのために、わざわざお金を出してNATルータを購入するといった発想はありませんでした。

そんな状況で何が起こったかというと、CATV経由でインターネットに接続すると、隣の家のPCのフォルダが見えました。もう16～17年前になりま

すが、筆者の実家のマンションでは、ものの見事にマンション中のいくつものコンピュータのフォルダが見えていました。

その後、業者側のIPフィルタリング対応以外にも、複数のコンピュータが接続したり、それ以外の家電もインターネットに接続したりするようになって、ISPが提供するものも必然的にNATルータへと変化していったこともあり、前述したようにフラットにインターネットに接続するようなことは一般的ではなくなっていきました。

ここから学べることは、極めて簡単です。システムを提供する側もインターネットとは何かを十分に理解してから作っているわけではなく、「とりあえずやってみよう」というレベルからサービスが始まるということです。

◆ ◆ ◆

IoT時代は多種多様な機材がインターネットに接続されるという時代になるわけですが、それだけいろいろなベンダがいろいろなものを開発し、提供する時代になるはずです。

そうなると、歴史を振り返れば当面は、新しく現れるIoTが、どれだけ品質が高いか、セキュリティ的に安全であるか、はたまた、どれだけサポートが行われるかということは未知数なわけです。そのことからも、そのシステムの安全性とはべつに安全性を確保する必要があるということになります。

小さなセキュリティゲートウェイ

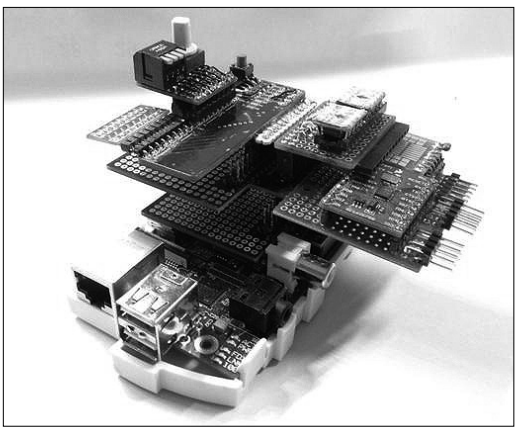
IoTとして入ってくる機材が信用できないわけですから、それをネットワーク的に監視するものが必要となってきます。IDS(Intrusion Detection System：侵入検知システム)やIPS(Intrusion Prevention System：侵入防止システム)も含めたファイアウォールのような機能を家庭内や建物内、あるいはもっと小さなエリア、たとえば茶の間とか自分の部屋といったレベルのサイズのネットワークに設置する時代がくるかもしれません。

この節だけ読むと、多くのみなさんは、ファイアウォールやIDS/IPSが極めて特殊なシステムで、性能のいい機材を必要とし、複雑で高価なソフトウェアの組み合わせで、さらに設定に特殊な知識を必要とするものである、と思うかもしれません。

しかし、私たちは、昔の高性能ワークステーションの能力を数千円で購入できることをすでに知っています。そのハードウェアに少し追加し、さらにオープンソースソフトウェアで構築した簡便なセキュリティゲートウェイを付けてあげることも可能ではないでしょうか。そして、また、このセキュリティゲートウェイもIoTの1つなのではないでしょうか。

現在、筆者は家庭や学校といった身近なIoT環境でのセキュリティゲートウェイが考えられないものかと、Raspberry Piをハードウェア・プラットフォームにしたオープンソースベースのプロトタイプを作成する共同研究を行っている最中です[注11](**写真1**)。

◆写真1　I/Oなどを満載した実験用Raspberry Pi
(写真提供：大野浩之氏)



i2cGPIO(16bit)、フォトカプラ(4bit×2)、UART(絶縁型)、SPI高速シリアルを搭載済。今後、RTCと電源制御系を追加予定とのこと。

注11)【参考文献】
大野浩之、鈴木裕信、「電子工作愛好者向けセキュリティゲートウェイの構築 第四報：Raspberry Guardianの実証実験へ向けて」、『マルチメディア、分散協調とモバイルシンポジウム2014論文集』、p211-p218、2014年7月
大野浩之、北口善明、鈴木裕信、「電子工作愛好者向けセキュリティゲートウェイの構築 第三報:構成要素の検討と性能評価」、『情報処理学会研究報告．マルチメディア通信と分散処理研究会報告』、情報処理学会、2014-07-17、2014年

NFCによる公開鍵交換

最近では、スマートフォンからIoT機材をコントロールするといったものが現れています。通信路はSSLなどの暗号通信を行っていますが、エンド・ツー・エンドの認証に関しては、今もってパスワードのものが多いようです。しかし、この連載で何度も繰り返し説明してきたとおり、現在はパスワードによる認証で安全性を確保できる時代ではなくなっています。

今はスマートフォンにNFC（近距離無線通信）が付いていてBluetoothのペアリングをする時代です。これをもう一歩発展させれば、NFCで公開鍵を交換し、お互いを認証するときは、パスワードではなくデジタル署名による認証を行うことができるようになるでしょう。こうすれば認証に関して暗号学的な強さでの安全性が確保できます。

認証のための鍵を入手しようとすると、実際に家に入って、クーラーなり、冷蔵庫なり、テレビなりに触るか、スマートフォンの中身を盗まない限り、鍵を入手できません。絶対に不可能とは言えませんが、パスワードに比べれば極めて安全になります。ぜひ、このようなしくみを持ったシステムを開発してほしいと思います。

終わりに

今回は、今現在大きな問題にはなっていないけれども、近い将来必ず大きな問題となってくるIoTの話題と、それを考察するという内容でした。問題が起こってから考えるのではなく、一歩先に思いを馳せなければ、そのときが本当に来てしまったときに右往左往するか、あるいは問題に目をつぶってしまうことになるでしょう。そうならないように、今日（今回）は、明日のことを少し考えてみました。SD

第54回

日本Androidの会とは ～コミュニティにより広がるAndroid創作活動

モバイルデバイス初のオープンソースプラットフォームとして、エンジニアから高い関心を集めるGoogle Android。いち早くそのノウハウを蓄積したAndroidエンジニアたちが展開するテクニックや情報を参考にして、大きく開かれたAndroidの世界へふみだそう！

嶋 是一　SHIMA Yoshikazu
NPO法人日本Androidの会 理事長

Android好きが集まるユーザコミュニティ「日本Androidの会」をご存じでしょうか。すでに6年におよぶ活動を続けており、参加いただいている方は2万人を超えています。Androidコミュニティ規模としては全世界から見ても稀有な存在と言われています。この連載も日本Androidの会のメンバーがリレー執筆を行っており、気づいたら4年目に突入しました。今回は日本Androidの会の紹介とともに、コミュニティ活動の楽しみ方や運営について紹介したいと思います。

日本Androidの会とは

日本Androidの会はコンソーシアムや業界団体ではなく、Androidをネタにして、好きなことを行い楽しもうというユーザコミュニティです。目指しているのは、この楽しみから広がるAndroidの普及を促進することです。

その活動母体はメーリングリストですが、それ以外にも日本最大級のAndroidの祭典Android Bazaar and Conference（以下ABC）を年2回のペースで開き、毎月東京エリアで開く無料勉強会はもとより、全国36ヵ所にわたる地方支部で開催される集いを通した活動を行っています。

入会するには、会のWebページ[注1]でメーリングリスト（Googleグループ）にメールアドレスを登録します。2014年11月27日現在22,080名の方に登録いただいており、会のイベント告知や案内を配信しています。また、会員同士のQ&Aをはじめ、技術の紹介や告知も行われています。

会で主催するイベント「ABC」は毎回掲げるテーマに沿って、カンファレンス（セミナー会場）とバザール（展示会場）の2本柱で行っています。前回の「ABC 2014 Spring[注2]」はABCの開催として10回目、5年目の記念イベントだったので、テーマを「Android REBORN! ～次の5年へのクロスポイント」とし、会場の秋葉原ダイビルとUDXに3,000人を超す方々が来場しました（**写真1**）。

このときは、開催の前々日にGoogleのスマートウォッチ開発環境「Android Wear SDK」が発表されるなどのサプライズがあり、皆がこぞって「我日本一也」とばかりにWearアプリ開発の発表が行われました。公開から1日ちょっとしかない状態なのにアプリの展示が多数行われるなど、まさに最先端を体感し、その熱気と興奮が忘れられないイベントとなりました。当日の目玉はGoogleから登壇した、Google Playのエンジニアリング・ディレクターChris Yerga（クリス・ヤーガ）さんであり、ここでもAndroid Wearの紹介がされるなど、まさにREBORN（生まれ変わり）するAndroidにぴったりなイベントとなりました。偶然のサプライズが開発意欲に火をつけ、それがコミュニティの中で飛び火し、多くの方の気持ちをあおり、盛り上げ、一つのムーブメントとしてイベント会場でお祭りになる。このような事件が起こるのも、コミュニティのおもしろいところです。

注1）http://www.android-group.jp/

注2）http://www.android-group.jp/conference/abc2014s/

▼写真1　ABC 2014 Springの様子

WGなどの取り組み

メーリングリストを通じて情報交換するだけでなく、特定のテーマを掲げてメーリングリストよりも少人数で活動するWGや部活動があります。WGは独自で活動を行い、イベントで活動報告や成果発表などの出展を行っています。もし自分の興味と一致するならば、このようなWGグループ活動から入るのがおもしろいでしょう。

「OBFT（Open Beacon Field Trial）[注3]」は、Bluetooth Low Energyの技術を用いて近接無線による位置検出を行う技術を用いた、参加者が自由に使えるフィールドトライアルを実施するグループです。AppleのiOSではiBeaconと呼ばれるしくみです。2014年8月には、神奈川県六角橋商店街でフィールドを設営するため、OBFTメンバーが商店街にBeaconデバイスを設置し、商店街としては世界初の取り組みを実現しました（**写真2**）。もしビーコンを用いたアイデアがあるのならば、この六角橋商店街でお試しください。

「学生部[注4]」は大学生が構成メンバーとなり、活動を通じて知識やスキルの向上など、切磋琢磨できる場所を提供することを目的に活動を行っています。

「福祉部[注5]」は“福祉の世界にもAndroidを”というコンセプトのもと、「福祉」に特化したアプリを制作することを目的としたグループです。

▼写真2　Beaconを設置するOBFTの活動のひとコマ

技術関連だとほかに、「組み込み部」はAndroidの組み込み開発に特化した活動を行い、「ものづくり部」はロボットや新しいAndroidデバイスを考案試作する活動を行っています。また新しい活動として「VR部」、そして「ドローン部（仮称）」などが今後立ち上がる予定です。

WG以外の取り組みも行っています。一つは教育の取り組みとして「Tech Institute」という、アプリ開発者を育てるプログラミングスクールの実施があります[注6]。高校生から社会人まで幅広い層の方が受講されており、最終目標は未経験者でもアプリをGoogle Playで公開することです。これは、㈱角川アスキー総合研究所と、会場である早稲田大学エクステンションセンター、そしてサムスン電子ジャパン㈱で運営されており、日本Androidの会として、半年で75回におよぶ講義の教材作成や企画協力を行っています。また、この全9巻にもわたるAndroidアプリの開発方法や企画手法の書かれた教材は、Creative Commons[注7]としてどなたでも見られるように公開します。この取り組みから、アプリ開発者のすそのが広がることを期待して活動しています。

教育分野ではこのほかにも、文化服装学院とインターンシップの取り組みも行っています。ファッションデザイナーを多数輩出する国内最大手の服飾学校ですが、こちら学生のインターンシップ先として、日本Androidの会が受け入れています。目的はAndroid技術と異文化の融合で、Androidアクセサリ（周辺装置）とファッ

注3）http://openbeacon.android-group.jp/
注4）http://student.android-group.jp/
注5）http://fukushi.android-group.jp/
注6）http://techinstitute.jp/
注7）http://creativecommons.jp/licenses/

ション文化の融合を目指して取り組んでいます。実際にLEDの点滅をスマートフォンからコントロールできるドレスやアクセサリを制作し、卒業発表として開催したファッションショーで、それらデバイスの披露がされました(**写真3**)。

コミュニティを運営するために

「自律分散」が日本Androidの会のコミュニティ運営の基本思想です。

人が集まり活動を行う場合、運営の方針が必要になります。もし業界団体や標準化を行う団体であった場合、参加者の利益や思いを実現するために発言力を高める必要があります。そのためには中央集権を行い、トップダウンで活動するほうが効果的でしょう。しかし、コミュニティ運営は異なります。とくに日本Androidの会はユーザコミュニティであり、参加している人たちが楽しむことのほうが、会のプレゼンスを保つよりも大切なことです。コミュニティ活動が継続的に盛り上がるためには、「おもしろい！」「取り組みたい！」と思った人が、思いどおり、存分に活動できる環境があることが大切と考えます。そのためには自分から立ち上がり、自分から動き始めやすくする必要があります。

何か行おうとしたときに、毎回中央の組織に問い合わせ、許可、指示を仰ぐのは、さまざまな足かせとなります。仰ぐ形で問い合わせても多くの場合は「何々がだめ、こうやるべき」としか返ってきませんので、やる気が削がれ、小さい小粋な提案があっても取り組むのが億劫(おっくう)になってしまいます。また、いちいち仰いでいたら時流に追いつけません。何にしろモバイルの世界で起こっている技術革新は、1日1日と進化します。問い合わせるよりは、新しいことに取り組んだほうがよっぽどおもしろいことができますし、自分のモチベーション向上にもなります。

そのため支部を立ち上げても、冷たい話かもしれませんが、日本Androidの会とは別組織として動いてもらいます。自分たちで企画して、自分たちで運営することが、活動の基本原理です。新しい支部を作るときは活動したい人がいれば、すぐに設立することができます。これが「自律分散」です。日本Androidの会としてのイベントや取り組みのときには、本会[注8]から一緒に活動する打診を行い、参画を促すような形で運営しています。

中央の組織に問い合わせることは、不必要とは考えませんが、これを最小にして各個の自由な活動を広げられる方向にバランスを取りながら運営しています。

▼写真3　Android技術を融合した作品が発表されたファッションショー（卒業発表）

日本Androidの会の組織

前述のとおり、日本Androidの会は参加者には無料で参加いただけるイベントを多く開催しています。これができているのも、スタッフのみなさんには(自発的な意味で)ボランティアとして集まっていただき、そして活動対価も無償で行っていただいているところに大きく依存しています。とはいえ、実際には会場の費用もかかれば、イベントでの掲示物やノベルティなどの制作物、工作セミナーなどでの部品代など、運営費は必ずかかります。このような運営費は企業などの賛助会員からの賛助費で賄われています。

日本Androidの会にはコミュニティ活動とは別に、特定非営利活動法人日本Androidの会[注9]

注8) 会社の組織だと本部、とかの呼称になりますが、自律分散だとそぐわないため、このような呼び方になっています。

注9) http://npo.android-group.jp/

というNPO法人があります。こちらはコミュニティの運営を支援・補助するために運営されています。賛助会員からの賛助費はNPO法人で受け取り、コミュニティ活動からの要請に応じて年間計画を立案し、活動の支援を行う形を取っています。

コミュニティ活動の計画立案や実際の活動は「コミュニティ運営委員」で行われており、現在筆者も含む87名のメンバーが加わっています。Androidの普及に向けて精力的に最先端の活動をされている方々であり、運営委員の一定数の推薦により加わることができます。もし世の中で行ってみたい普及活動やネタがあるならば、会の運営に加わっていただけると幸いです。

このような組織体制は長年の取り組みによって構築されてきました。当然コミュニティの活動なので、内部で揉めて運営が炎上することも多々あります。しかし、それら1つ1つを悩んで考えて解決し、時には身を切るような思いで、より良いしくみやルールを組み上げてきたため、なんとか今日まで運営が継続できているものと思っています。

アンテナを持つ人の集まり

気が付けばAndroidが発表されて7年目に入り、これ自体は新技術とは言えなくなりました。また、Androidも全世界のスマートフォンの8割に搭載されており、普及活動の役割は終わったのではないかと指摘されることもあります。

しかしスマートフォン以外のAndroidの普及は、まだまだこれからです。そして、Androidが新技術でなくなったかというと、そうではなく、新しい技術はAndroidの周辺で多く起こっています。そういう情報に鋭いアンテナを持ち、興味を持つことに長けたメンバーが運営委員に数多くいます。そもそも7年前に、モバイルのオープンOSが登場したことを、アンテナに引っかけて集まったメンバーです。それがたまたまAndroidであっただけで、Androidだけに興味あるわけではないと思っています。しかし、このAndroid登場のおかげで、そのような方々が一度でも一点に集まり、コミュニティ活動を行うときにふたたび集う場所となっている、これが日本Androidの会として存在する意義の一つだと思っています。

おもしろいのが、新しい技術の勉強会やイベントなどに行くと、必ずと言っていいほど日本Androidの会のメンバーに会うことです。またそういう人たちと、一緒に将来を予測、体感、普及していく活動も、筆者にとってとても刺激的なことです。

12月21日、ABC 2014 Winter開催!

本文でも紹介したABCですが、次回は2014年12月21日(日曜日)に品川駅から徒歩で行ける東海大学高輪キャンパスで開催します[注10]。ほとんど本誌本号発売と同時になりますが、もし興味持っていただき、間に合いましたらご来場ください。今回のテーマは「Android Link to Next Generation!」です。Android Lバージョンは、スマートフォンだけでなく、スマートウォッチ、テレビ、自動車のOSとしてリリースされました。Androidはまさに前回のABCのテーマであったREBORNを行い、異なるプラットフォームのOSとして次世代のスマートデバイスへの橋渡しを行うところです。それを体感するのが今回のイベントです。

このようなイベントをきっかけに、さまざまなコミュニティ活動に興味を持っていただけると幸いです。SD

注10) http://abc.android-group.jp/2014w/

嶋 是一(しま よしかず) 特定非営利活動法人日本Androidの会 理事長

モバイルに関連する技術開発を行うかたわら、モバイル技術の普及活動行に取り組む。KDDIテクノロジーに所属。著書に『Google Android入門 ～携帯電話開発の新技術 2008』(技術評論社)など。Mail sim@android-group.jp Twitter @shimay

Red Hat Enterprise Linuxを
極める・使いこなすヒント

.SPECs

ドット・スペックス

第9回 Red Hat Enterprise Linux Atomic Host登場!

Red Hat Enterprise Linux Atomic Hostは昨今注目を集めているDockerに最適化されたRed Hat Enterprise Linux 7として開発が進められています。今回はAtomic Hostの概要と特徴を紹介します。

レッドハット(株)サービス事業統括本部プラットフォームソリューション統括部
ソリューションアーキテクト部長　藤田 稜 (ふじた りょう)

Atomic Hostの Public Betaを開始

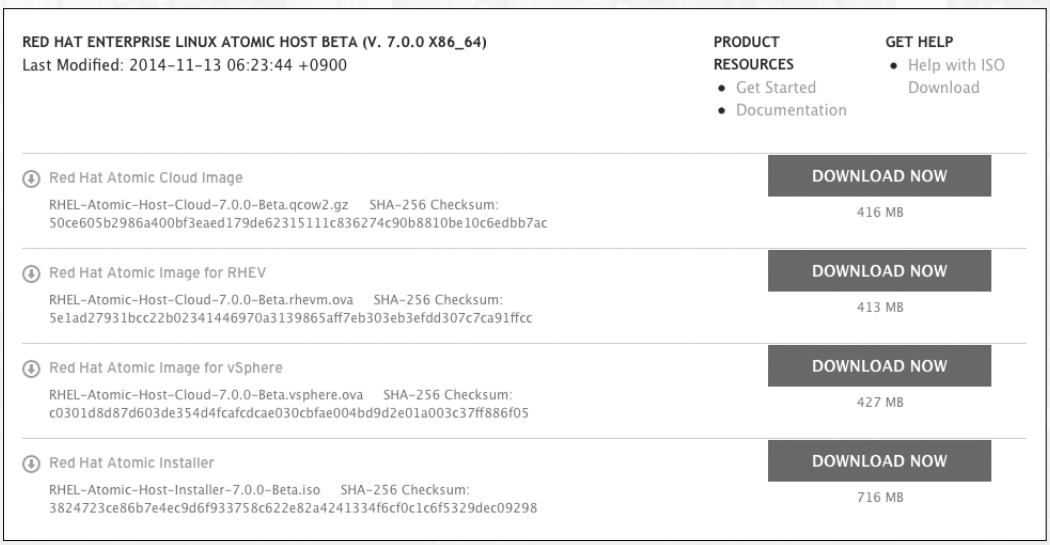
米国時間で2014年11月11日(日本時間:同12日)に、Red HatはRed Hat Enterprise Linux(以降、RHEL)Atomic Hostのパブリックベータの提供を開始しました。すでにRHEL 7にはDockerが同梱[注1]され、利用ができるものの、昨年来アナウンスされてきた「Dockerに最適化されたRHEL」が初めて公開されたことになります。

Atomic Hostは PROJECT ATOMIC[注2]をアップストリームとし、Fedora/CentOSでの統合・テストを経て企業向けのサポートを提供する製品です。2015年の早い時期には製品版(GA:General Availability)としてのサポートが開始されるものと思われます。

いますぐ始められるDocker

Atomic Hostは利便性を考慮して複数の配布

▼図1　Atomic Hostの配布形態



注1) RHEL 7のインストーラには含まれないものの、Extrasチャネルを追加することで利用可能。
注2) http://www.projectatomic.io

形態[注3]をとっています。

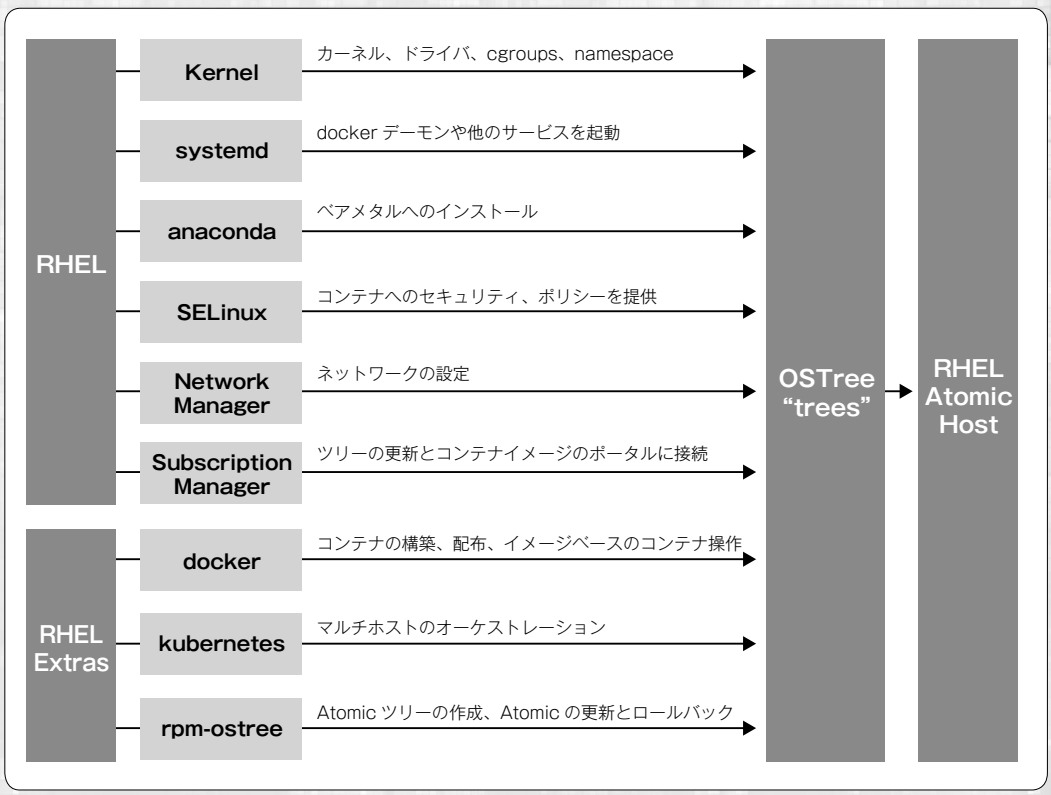
また、Google Compute Engine[注4]やAmazon Web Service[注5]上の評価も可能となっているため、Dockerを触ってみようという読者は、ぜひ試用してください。

RHELのサブセットを最適化

Atomic Hostを、「Dockerに最適化されたRHEL」と表現しました。より正確には「RHEL 7のサブセットをベースにパッケージングや初期設定をDocker用に最適化したもの」と言えます。パブリックベータではRHEL 7(x86_64)にも含まれる380程度[注6]のRPMパッケージで構成されており、RHELとAtomic Hostは**図2**のような関係になっています。

Atomic Hostを特徴づけている主なソフトウェアはDocker/Kubernetes/rpm-ostreeの3つです。前二者については、本誌2014年12月号の第1特集「Dockerを導入する理由」で紹介されていたのでご存じの読者が多いと思います。Kubernetesは米Google社が推進しているOSSプロジェクトであり、Red HatはKubernetesのAtomic Hostへの採用にあたって米Google社と協業しています。一方で、rpm-ostreeはあまり馴染みがないと思いますが、rpm-ostreeこそがAtomic Hostのキモです。このことはAtomic Hostで利用する主な管理ツールである

▼図2 RHELとAtomic Hostの関係



注3) Red Hatのアカウントを作成すればダウンロードが可能。
注4) https://access.redhat.com/articles/rhel-atomic-install-gce
注5) https://access.redhat.com/articles/rhel-atomic-install-aws
注6) RHEL 7を"Minimum"でインストールした場合と比較するとRPMパッケージが60程度多い。

▼図3　atomicコマンドの実体

```
# ls -l /usr/bin/atomic
lrwxrwxrwx. 1 root root 10 11月 13 00:36 /usr/bin/atomic -> rpm-ostree
```

▼図4　atomicコマンドの扱い方（バージョン確認・更新・ロールバック）

```
# sudo atomic status
  VERSION     ID             OSNAME               REFSPEC
*  7.0.0       dcf0c846ff     rhel-atomic-host     rhel-atomic-host-beta-ostree:rhel-atomic-host/7/x86_64/standard
 7.0.1.0     335ae35519     rhel-atomic-host     rhel-atomic-host-beta-ostree:rhel-atomic-host/7/x86_64/standard

# sudo atomic upgrade
Updating from: rhel-atomic-host-beta-ostree:rhel-atomic-host/7/x86_64/standard

496 metadata, 2235 content objects fetched; 109562 KiB transferred in 446 seconds
Copying /etc changes: 10 modified, 4 removed, 37 added
Transaction complete; bootconfig swap: yes deployment count change: 1
Changed:
  docker-1.2.0-1.8.el7.x86_64
  docker-storage-setup-0.0.3-1.el7.noarch
  glib-networking-2.40.1-1.atomic.el7.x86_64
  glib2-2.40.0-1.atomic.el7.x86_64
  kernel-3.10.0-123.9.2.el7.x86_64
  kubernetes-0.4-368.0.git8e1d416.el7.x86_64
  tzdata-2014i-1.el7.noarch
Updates prepared for next boot; run "systemctl reboot" to start a reboot

# sudo atomic status
  VERSION     ID             OSNAME               REFSPEC
*  7.0.0       dcf0c846ff     rhel-atomic-host     rhel-atomic-host-beta-ostree:rhel-atomic-host/7/x86_64/standard
 7.0.1.0     335ae35519     rhel-atomic-host     rhel-atomic-host-beta-ostree:rhel-atomic-host/7/x86_64/standard

# sudo atomic rollback
```

atomicコマンドが**図3**のとおり、rpm-ostreeコマンドへのシンボリックリンクであることからもわかります。

Atomic Hostは RHELではない

Atomic Hostの Public Beta を Red Hat の Customer Portal[注7]からダウンロードしてインストールすると、見た目はRHEL 7と大きな変化はありません。しかしDockerへの最適化によって、通常のRHEL 7とは随所で異なっています。

yumが使用できない

Atomic Hostではyumコマンドの利用は「禁止」されています。atomicコマンドを用いて“tree”のバージョンを確認・更新・ロールバック[注8]します(**図4**)。

ファイルシステム レイアウトが特殊

これはロングオプションを付けてlsコマンドを実行すればわかりますが、/(ルート)以下にあるディレクトリは/var/や/usr/へのシンボ

注7） http://red.ht/14BVxKE
注8） subscription-managerによるCustomer Portalへの登録が必要。

▼図5　Atomic Hostのファイルシステムレイアウト

```
# ls -l /
total 18
lrwxrwxrwx.   1 root root    7 Nov 13 00:36 bin -> usr/bin
drwxr-xr-x.   7 root root 1024 Nov 18 21:50 boot
drwxr-xr-x.  20 root root 3220 Nov 18 20:22 dev
drwxr-xr-x.  80 root root 4096 Nov 18 20:22 etc
lrwxrwxrwx.   1 root root    8 Nov 13 00:36 home -> var/home
lrwxrwxrwx.   1 root root    7 Nov 13 00:36 lib -> usr/lib
lrwxrwxrwx.   1 root root    9 Nov 13 00:36 lib64 -> usr/lib64
lrwxrwxrwx.   1 root root    9 Nov 13 00:36 media -> run/media
lrwxrwxrwx.   1 root root    7 Nov 13 00:36 mnt -> var/mnt
lrwxrwxrwx.   1 root root    7 Nov 13 00:36 opt -> var/opt
lrwxrwxrwx.   1 root root   14 Nov 13 00:36 ostree -> sysroot/ostree
dr-xr-xr-x. 286 root root    0 Nov 18 20:22 proc
lrwxrwxrwx.   1 root root   12 Nov 13 00:36 root -> var/roothome
drwxr-xr-x.  27 root root  880 Nov 18 21:38 run
lrwxrwxrwx.   1 root root    8 Nov 13 00:36 sbin -> usr/sbin
lrwxrwxrwx.   1 root root    7 Nov 13 00:36 srv -> var/srv
dr-xr-xr-x.  13 root root    0 Nov 18 20:22 sys
drwxr-xr-x.  11 root root  103 Nov 13 00:33 sysroot
lrwxrwxrwx.   1 root root   11 Nov 13 00:36 tmp -> sysroot/tmp
drwxr-xr-x.  12 root root 4096 Nov 13 00:36 usr
drwxr-xr-x.  24 root root 4096 Nov 13 00:37 var
```

▼図6　/sysroot/ostreeディレクトリの内容

```
# ls -l /sysroot/ostree/
total 0
lrwxrwxrwx. 1 root root  8 Nov 18 21:50 boot.0 ->
boot.0.1
drwxr-xr-x. 3 root root 29 Nov 18 21:50 boot.0.1
drwxr-xr-x. 3 root root 29 Nov 13 00:36 deploy
drwxr-xr-x. 7 root root 85 Nov 18 21:50 repo
```

リックリンクとなっています。Atomic Hostでは起動の初期段階[注9]でchrootが行われ、「物理的なroot」は/sysroot/にマウントされています。Dockerではそれぞれのコンテナがchrootされたファイルシステムを利用するように設定されますが、Atomic Hostは根幹からchrootされたRHELになっていると言えます(**図5**)。

また、/sysroot/ostreeへのシンボリックリンクとなっている/ostreeディレクトリには、boot、deploy、repoというディレクトリがあります(**図6**)。

repoディレクトリはGitによってバージョン管理されており、Atomic Host上に配置されるファイルの「ソース」となっています。

deployディレクトリは複数のバージョンのRHELを格納するディレクトリで、deploy/rhel-atomic-host/deploy/ 以下はrepo/objects/ディレクトリ以下にあるファイルへのハードリンクです。たとえば、**図7**のようにvmlinuzファイルがiノード番号794291であれば、repo/objects/以下に同じiノード番号となっているハードリンク先を見つけることができます。

bootディレクトリも同様で、シンボリックリンクとrepoディレクトリへのハードリンクを活用してデータ容量を最小限に抑えつつ、バージョン管理を行っています。

つまり、Atomic HostではRPMパッケージではなくファイルをバージョン管理の粒度とするために、特殊なファイルシステムレイアウトを採用しているのです。したがって、/ostreeディレクトリを直接手動で編集することは禁止されています。

tunedによるパフォーマンスの最適化

Atomic Hostはパフォーマンス面でもDockerに最適化されています。RHELではバージョン6以降、カーネルのパラメータなどを手動で設定しなくてもtunedのプロファイルによって自動的にパフォーマンスの最適化が可能です。Atomic Hostではデフォルトでは**図8**のように“atomic-guest”プロファイルが設定されています。

まとめ

これまで示してきたように、Atomic Hostは

注9）dracutのプラグインによってchrootされている。

▼図7　repo/objects/へのハードリンク

```
# ll -i /sysroot/ostree/deploy/rhel-atomic-host/deploy/
335ae3551939b8de8fd6663ec9db834b1d2150fe7a5b9b1017b74c1070006888.0/
boot/vmlinuz-3.10.0-123.8.1.el7.x86_64-
513f2a5e7f1ed9cc4e4fe655a92f187b4324f4f4f569b9d474d683581c34d5c3
794291 -rwxr-xr-x. 2 root root 4904912 Jan  1  1970 /sysroot/ostree/deploy/rhel-atomic-host/
deploy/335ae3551939b8de8fd6663ec9db834b1d2150fe7a5b9b1017b74c1070006888.0/boot/vmlinuz-3.10.0-
123.8.1.el7.x86_64-513f2a5e7f1ed9cc4e4fe655a92f187b4324f4f4f569b9d474d683581c34d5c3

# find /sysroot/ostree/repo/ -inum 794291
/sysroot/ostree/repo/objects/2d/8a6c1c05347867fc31fbd998ff88786911d82f2ec38cde0fe48928321772a9.file
```

▼図8　Atomic Hostのtunedのプロファイル

```
# tuned-adm list
Available profiles:
- atomic-guest
- atomic-host
- balanced
- desktop
- latency-performance
- network-latency
- network-throughput
- powersave
- sap
- throughput-performance
- virtual-guest
- virtual-host
Current active profile: atomic-guest
```

さまざまな最適化が施されており、Docker専用の「ホスト」として利用しやすくなっています。また、Kubernetesによって複数のホストを管理する機能はGA版で提供が開始される予定となっており、さらに利便性が向上するものと思われます。Red Hatが提供するPaaSであるOpenShiftも次のバージョン3.0からAtomic Hostベースとなる予定であり、今後しばらくは注目する必要がありそうです。

次回はセットアップが終わったAtomic Host上で、コンテナを実行する方法について紹介する予定です。SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2014年12月号

第1特集
急速に普及するコンテナ型仮想環境
Dockerを導入する理由

第2特集
基礎の基礎から押さえる必須技術
やさしくわかるVPNの教科書

一般記事
・bashの脆弱性“Shellshock”その影響と対策
・SoftLayerを使ってみませんか？［最終回］
・Jamesのセキュリティレッスン［2］

定価（本体1,220円＋税）

2014年11月号

第1特集
Docker・Ansible・シェルスクリプト
無理なくはじめるInfrastructure as Code

第2特集
オンプレミスもクラウドも縦横無尽
サーバの目利きになる方法［後編］

一般記事
・8086時代から今を俯瞰する　CPU温故知新
・はてな謹製、サーバ管理ツール　Mackerel入門

定価（本体1,220円＋税）

2014年10月号

第1特集
今ふたたびのJava
言語仕様・開発環境・デバッグ機能

第2特集
オンプレミスを制するものはクラウドを制する
サーバの目利きになる方法［前編］

一般記事
・オーケストレーションツールSerf・Consul入門［Consul編］
・SoftLayerを使ってみませんか？［2］　ほか

定価（本体1,220円＋税）

2014年9月号

第1特集
この夏に克服したい2つの壁
C言語のポインタとオブジェクト指向

第2特集
止まらないサービスを支えるシステム構築の基礎
クラスタリングの教科書

一般記事
・SoftLayerを使ってみませんか？
・NICをまとめて高速通信！（前編）
・Serf・Consul入門　特別定価（本体1,300円＋税）

2014年8月号

第1特集
システムログからWebやDB、ビッグデータの基礎まで
ログを読む技術

第2特集
forkを通して考える・試す・コードを読む
Linuxカーネルのしくみを探る

一般記事
・OpenSSLの脆弱性"Heartbleed"の教訓（後編）
・使ってみよう！tcpdump

定価（本体1,220円＋税）

2014年7月号

第1特集
［多機能］［高速処理］［高負荷対策］
そろそろNginx移行を考えているあなたへ

第2特集
知っているようで知らない
DHCPサーバの教科書

一般記事
・OpenSSLの脆弱性"Heartbleed"の教訓（前編）
・Webアプリのパフォーマンス改善（最終回）ほか

定価（本体1,220円＋税）

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。

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第15回 ❖FreeBSD 10.1-RELEASEで何が変わったの?【パート2】

◉著者プロフィール
後藤 大地(ごとう だいち)
BSDコンサルティング(株)　取締役／(有)オングス　代表取締役／FreeBSD committer
エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

お勧め!　10.1-RELEASE

安定版FreeBSDの最新版にして10系第2回目のリリースとなる「FreeBSD 10.1-RELEASE」が登場しました。当初の予定から2週間ほど遅れましたが、2014年11月14日(協定世界時)にプロジェクトから公開されました。amd64版、arm版、i386版、ia64版、powerpc版、powerpc64版、sparc64版のインストールイメージが提供されています。

10.1-RELEASEからは従来のインストールイメージに加えて、UEFI[注1]対応版のインストールイメージの提供が開始された点も注目されます。UEFI版が提供されているのはamd64版のみです。現在市場に流通しているラックマウントサーバやPCは従来のBIOSとUEFIの双方に対応していますが、向こう数年でUEFIへの移行が進むものとみられます。今後はUEFIを使ったシステムの起動がデフォルトになっていくでしょう。

amd64版とi386版に関しては仮想ディスク形式も提供されています。提供されている仮想ディスクのフォーマットは次の4種類です。

- QCOW2(QEMU Copy-on-Write形式)
- RAW(通常のディスク形式)
- VHD(Microsoft Hyper-V形式)
- VMDK(VMware形式)

クラウドプラットフォーム向けにはAmazon EC2向けのインスタンスとMicrosoft Azure向けのインスタンスが提供されています。FreeBSDプロジェクトは以前からAmazon EC2向けのインスタンスを提供してきましたが、これからはMicrosoft Azureも重要なプラットフォームになる見通しです。MicrosoftはFreeBSDをゲストオペレーティングシステムとして正式サポートすることを検討しており、企業サポートが必要なケースなどで採用しやすいプラットフォームになる可能性がでてきています。

10.1-RELEASEはこれまでのどのバージョンよりもARMのサポートが充実しています。これはFreeBSDを組込み機器などで利用しているベンダからのサポートが積極的なためです。ARMは消費電力が少ないといった特徴があることから、組込み機器での活用が進んでいます。そうしたシーンでFreeBSDが利用されているため、ARMの機能はリリースを重ねるごとに充実しています。この傾向は今後も継続する見通しです。

すでに前回紹介済みのものもありますが、10.1-RELEASEは10.0-RELEASEにいくつかの新機能が取り込まれているほか、SMP(Symmetric Multi processing)におけるパフォーマンスやスケーラビリティ向上が図られています。とくにストレージ関連の機能が強化されており、ストレージ系で採用している場合にはアップグレード価値の高いバージョンとなっています。

注1) Unified Extensible Firmware Interfaceの略。OSとプラットフォームファームウェア間のソフトウェアインターフェースを定義する仕様。従来のSystem BIOSを強化している。

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。

前回に引き続いて、注目の新機能を紹介していきましょう。

IPv4/IPv6、UDP-Liteに対応

FreeBSD 10.1-RELEASEのIPv4およびIPv6スタックには、新しくUDP-Lite(Lightweight User Datagram Protocol;RFC 3828)サポートが追加されました。UDP-Liteはインターネットでの動画や音声の配信を主な目的としたプロトコルです。UDPと同じようにパケットが到達したかどうかの確認は実施しません。加えて、データが破損していてもそのまま転送を続けるといった特徴があります。動画や音声の場合にはデータの破損よりも再送で発生する遅延のほうが問題になりますので、動画や音声の配信に適したプロトコルとされています。

iSCSI/NFSパフォーマンス向上と新機能

FreeBSDはストレージシステムのオペレーティングシステムとしての普及が進んでいます。細かい点になりますけれども、sysctl(8)の値として**kern.iscsi.fail_on_disconnection**が追加された点も押さえておきたいところです。これはターゲットとのコネクションが切れた場合にiSCSIクライアントがディスクデバイスをデタッチするのを有効にするためのフラグです。従来の実装ではコネクションが回復するまで待機していましたが、この部分の挙動を制御できるようになりました。この変更はFreeBSD Foundationのスポンサードのもとで実施されました。

10.1-RELEASEにはiXsystemsのスポンサードのもとで実施されたカーネルRPCコードが取り込まれています。これはNFSサーバの基盤技術として活用される技術で、SMPにおけるパフォーマンスの向上とスケーラビリティの向上が実現しています。iXsystemsはほかにもiSCSI関連のパフォーマンスの改善や新機能の追加にスポンサードしていまして、その成果物が10.1-RELEASEには取り込まれています。

NFSという点でいきますと、NFSデーモンにあたりnfsd(8)サーバのバージョンが4.1へアップグレードされました。この結果RFC 5661のサポートが実現しています。細かい変更点になりますけれども、mount_nfs(8)ユーティリティで「-o vers=4」といったように、使用するバージョンも指定できるように拡張が実施されています。

GEOMの性能向上

iXsystemsスポンサードのもとで実施された改良はGEOMサブシステムにも及んでいます。GEOMサブシステムはI/Oダイレクトディスパッチがサポートされましたし、スレッド回りの処理が改善しています。GEOM RAIDおよびvirtio_blk(4)ドライバ、virtio_scsi(4)ドライバ、xen(4) blkfrontドライバにはUnmapped I/Oのサポートも追加されました。

同様の改善はCAMサブシステムにも取り込まれています。10.1-RELEASEのCAMサブシステムはロック回りがより細かく実施されるようになったほか、ダイレクトディスパッチのサポート、マルチキューのサポートなどが実現しています。GEOMサブシステムと組み合わせることで、SMPにおいて従来よりも性能の向上が期待できます。

GEOMのほかの特徴としては、いくつかのGEOMプロバイダでリサイズ系の機能が強化された点をあげることができます。従来よりもリサイズ関連の機能の強化が図られていて、動的にサイズを変更するといった作業がやりやすくなっています。

LSI MegaRAID SASオフィシャルドライバ導入

LSIのスポンサードのもとでmpr(4)ドライバとmrsas(4)ドライバが追加された点も注目ポイントです。GENERICカーネルに取り込まれています。これらデバイスドライバはラックマウントなどエンタープライズシーンで採用されるサーバでよく採用されているもので、LSI Fusion-MPT 3 12Gb SCSI/SATAコントローラへの対応、LSI MegaRAID SAS

への対応などが実現しています。これまでFreeBSDではmfi(4)デバイスドライバで対応していましたが、10.1-RELEASEからはmrsas(4)で代替可能です。mrsas(4)ドライバを有効にする場合には/boot/loader.confに「hw.mfi.mrsas_enable=1」の設定を追加してください。

なお、今のところmfiutil(8)ユーティリティはmrsas(4)ドライバでは動作しないので、mrsas(4)でユーティリティを使いたい場合にはLSIがリリースしているユーティリティを導入するか、または対応するまでmfi(4)ドライバとmfiutil(8)ユーティリティを使うといった方法が考えられます。10.1-RELEASEのmfi(4)ドライバはUnmapped I/Oに対応していますし、MegaRAID Furyカードのサポートも追加されています。実際にどちらのドライバを採用するかは、mfi(4)とmrsas(4)の双方で負荷試験を実施して、より安定しているほうを採用するのがよいでしょう。

ZFSのデフォルト性能向上

ZFS回りではsysctl(8)値として「vfs.zfs.zio.use_uma」がふたたび有効になった点が注目ポイントです。この値はマルチコアのマシンでメモリを豊富に搭載している場合にプロセッサの負荷割合を下げ、さらにZFSのパフォーマンスを向上させるためのフラグです。カーネル内メモリを使い切ってしまう問題が報告されたことで一時無効化されましたが、問題が解決したのでふたたび有効になりました。ZFSを使っているのであれば、この機能を利用するためだけに10.1-RELEASEにアップデートする理由にもなるでしょう。

10.1-RELEASEには、OpenSolarisから移植されたlibzfsスレッドプールAPIが取り込まれています。このAPIが取り込まれたことで並列ディスクスキャンが可能になりましたので、特定の負荷状況で発生するzpool(8)にインポートタイムを改善することができるようになりました。

また、ZFS ARCハッシュテーブルのデフォルトサイズが引き上げられた点も、今回のリリースのポイントとなっています。この値はloader(8)でも「vfs.zfs.arc_average_blocksize」の値として調整できるようになりました。これまでのデフォルト値は性能を引き出すには値が小さく、とくにキャッシュリードでの性能に制限が出ていました。10.1-RELEASEに引き上げることで性能の改善が期待できます。

マルチスレッドに対応したSoft updatesと新規導入のautofs(5)

今回のリリースではFFS(Fast File System)にもアップデートが盛り込まれました。これまでシングルスレッド対応だったSoft updatesが、今回からマルチスレッドに対応しています。このため今後はFFSのマウントポイントごとにSoft updates機能が対応することになり、SMPにおける性能向上やスケーラビリティの向上などが期待できます。

また、新しい機能としてautofs(5)が追加された点も注目ポイントです。これはMac OS XやSolarisなどで使われているautofs(5)とよく似た機能で、自動マウント機能をサポートします。設定ファイルにはSun互換のauto_master(5)ファイルが利用できます。ユーザランドからはautomount(8)ユーティリティを使って制御します。状況に応じた柔軟なマウント処理を実現できる機能です。

Jail関連の機能強化

10.1-RELEASEのユーザランドコマンドや設定ファイルには新しい機能が細かく追加されていますが、とくにJail関連ということでps(1)コマンドとtop(1)コマンドを紹介しておきます。

ps(1)コマンドには新しく「-J」というオプションが追加されました。このオプションはJail IDを指定するもので、特定のJail内部のプロセスだけを表示するといったことができます。

同様の機能がtop(1)ユーティリティにも追加されました。こちらはJail IDでもJail名でもフィルタリングが可能です。また、jail(8)ユーティリティも拡張されまして、ip4.addrやip6.addrパラ

メータにおいてcarp(4)インターフェースが指定できるようになりました。

ベンダとの連携が進むFreeBSDプロジェクト

最近のFreeBSDにはベンダのスポンサードのもと開発された機能の取り込みが続いています。これはFreeBSD FoundationがFreeBSDを活用しているベンダとの連携を進めている成果が現れている結果だといえます。FreeBSD Foundationではベンダを活用しているベンダとの連携を強めてドネーションを求めるとともに、要望を聞き出して必要になる機能の開発をデベロッパにアサインしたり、コントリビュートされるコードをプロジェクト側へマージするように勧めるといった活動を進めています。

とくにこの数年は、FreeBSDとZFSをベースとしたNASソリューションの開発が世界中の企業(大手や中小、ベンチャーに限らず)で活発化していることもあり、iSCSIやNFSなど、こうした関連機能の強化が続いています。10.1-RELEASEにはそうした機能が取り込まれており、プロダクトのベースオペレーティングシステムとして活用しがいのある状態になっています。

10.1-RELEASEへのアップグレード方法

デフォルトカーネルのままFreeBSDを使っている場合には、FreeBSD Updateの機能を使ってFreeBSD 10.1-RELEASEへアップデートするのが簡単でしょう。ダウンロードやアップデート時間を含めても10分前後といったところです。**図1**のように作業します。

「freebsd-update upgrade -r 10.1」で10.1-RELEASEに必要になるコンポーネントのダウンロードが実施されます。「freebsd-update install」でダウンロードしてきたコンポーネントのインストールを実施します。インストールが完了すると、いったんシステムを再起動してから再度「freebsd-update install」を実行するように求められますので、指示通りに作業を行います。

再起動して「freebsd-update install」を実施すると、Ports Collectionからインストールしたサードパーティ製のソフトウェアをすべて再構築するように指示がでます。pkgを使っているのであれば、ここですべて再インストールするか、アップグレードの作業を実施しておくとよいでしょう(**図2**)。

サードパーティ製ソフトウェアの入れ替えなどを実施したら、もう一度「freebsd-update install」コマンドを実行して、アップグレード作業は完了です(**図3**)。

freebsd-version(1)というのは1つ前のバージョンとなる10.0-RELEASEで導入された新しいコマンドです。FreeBSDはセキュリティパッチの適用などでカーネルだけパッチレベルがあがったり、逆にユーザランドだけパッチレベルがあがったりすることがあります。freebsd-version(1)はユーザランドとカーネルのバージョンを個別に表示するためのコマンドで、-kでカーネルのバージョンとパッチレベル、-uでユーザランドのバージョンとパッチレベルを表示させることができます。SD

▼**図1　FreeBSD 10.1-RELEASEへのアップデート作業**

```
% freebsd-update upgrade -r 10.1
% freebsd-update install
% shutdown -r now
% freebsd-update install
```

▼**図2　サードパーティ製ソフトウェアのアップデートなどの作業も実施しておく**

```
% pkg upgrade
% freebsd-update install
```

▼**図3　FreeBSD 10.1-RELEASEへアップグレード完了**

```
% uname -sr
FreeBSD 10.1-RELEASE
% freebsd-version -k
10.1-RELEASE
% freebsd-version -u
10.1-RELEASE
%
```

22

Debian Developer　やまねひでき　henrich@debian.org

FLOSSコミュニティへのContributionとは?

Debian Hot Topics

コミュニティにContributionするには

以前、読者からいただいたメール[注1]に「一人のエンジニア視点から、OSSコミュニティに対してどういった貢献ができるか？ということについて取り上げてください」という提案がありました。2014年11月号の本連載で「積極的にDebianにcontributionをして来年のDebConfの旅費を勝ち取りましょう！」と書いた手前もありますので、今回はこのあたりを説明します。

筆者は、趣味的な立場でコミュニティにかかわってきたので、業務でOSSとかかわっている人とは視点が違うかもしれませんが、1つの意見として参考にしていただければと思います。

積極的な参加

よくContribution(Contribute)を「貢献」と訳しますが、実際のところ字面の印象から受けるような奉仕的なものではなく、「積極的な参加」程度のものだと筆者は考えています。そして、FLOSS(Free/Libre and Open Source Software)界隈へのContributionというのは、大きくいくつかの種類があるように思います。

1つは「ソースコード」で貢献するパターンです。たとえば「このプロダクトを使っていて困ったことがあったが、うまい具合に手元で修正できたからパッチを送る」というパターンです。最近だと「おすそ分け」という形で、GitHubによりカジュアルに行われるようになってきていますね。別の理由としては「いちいち手元で修正版を都度メンテしていくのはコストがかかるのでフィードバックしておこう」というのもあります。

もう1つは「参加」して貢献するパターンです。コミュニティやプロジェクトに入り込んで「改善できるところはないかな？」とプロジェクトの発展に喜びを求めていくタイプ(個人に多く見られます)や、「なるべくコミュニティへの関与を多くして発言力や影響力を大きくしたい」と参加していくタイプ(多くの企業の場合はこれですね)です。筆者の場合は「喜びを求めていくタイプ」にあたります[注2]。

どのパターンでも、コミュニティというエコシステムにおいて「すべてをコントロールしよう」という姿勢(過剰なガバナンス重視)は、傲慢に見られて関係者の賛同や注視している人間からの納得を得られにくいので、注意が必要です。参加するメンバーの納得という「空気感」が重要な点であり、力ずく(何らかのルールの強制や著作権／特許)で従わせようとするのは、横暴なので嫌われます。

注1) 本連載の感想や要望を編集部へのメール(sd@gihyo.co.jp)、あるいは、Twitterの@gihyosdや@debianjp宛にお寄せください。次回以降の内容に反映させていただきます。

注2) ただ、コミュニティに参加を始めた最初の動機は実はちょっと違いました。まだ初心者だった筆者はDebianを使っていくうえでいろいろと疑問や困ったことがありました。そこでメーリングリストで質問を……と思ったときに、「有名どころの人に良い意味で名前を覚えてもらって、自分が質問をする際に的確に答えてもらえるようにしよう！」と考えたのが動機だったのです。翻訳などから始めたのですが、Debianに関するバックグラウンドの知識や現状の把握もできたので、一石二鳥でした。

まずは小さなタスクから

どのパターンでも参加者にとって重要なのは「小さなタスクを実行して『信頼貯金』を獲得すること」です。小さなタスクは失敗しても被害は小さいですし、既存のコミュニティメンバーに相談もしやすい(相手も気軽に返答しやすい)ものです。さまざまな小さなタスクをこなすことで「この人のやろうとすることには意味があるから聞いてみよう」という状況を作り出していきましょう(RPGでクエストをこなしてレベルアップするみたいな感じですね)。

初めから大きなタスクに取り掛かろうとすると、「私がやるから手を出さないで!」とほかの人の参加をブロックしてしまいContributionとは真逆な負の影響を与えてしまったり、コミュニティに対する姿勢の論争などに発展して生産性を落としてしまったりします。また、がんばったけれども失敗したというときには、それまでの作業が無駄になり、その結果、自身のモチベーションがガクッと下がってそれ以上の作業ができなくなることも大いにあり得ます。

パッチを送付する際にも、巨大なパッチをいきなり送ると、送りつけられる側の人はかなりの負担を受けることになります。レビューを行うにも相当の苦労を要するのは、容易に想像がつくでしょう。あるいは、「こんなにデカイの見てられないよ」と単にスルーされてしまうことも多くなります。せっかくの作業の成果が取り込まれないのは悲しいですよね。これを回避するには小さな1つのタスクを改善するシンプルなパッチを送付するのが良いお作法です。今風に言えば「すばやく小さなサイクルを回すアジャイルな姿勢」が重要だと言えましょうか。

信頼貯金の獲得

Contributionというやりとりの中で信頼貯金を獲得するのに重要なのは、押しつけではダメだということです。相手が喜ぶことをするのが重要であって、一方的な押しつけは相手を困らせるだけという、このあたりは人と人との「お付き合い」に似ているのではないでしょうか。

COLUMN コードを書くのが苦手な人でもプロジェクトに協力できる?

「ドキュメントの翻訳ならコードを書けなくても大丈夫」と耳にすることがありますが、実際にコミュニティにはそういう人もいるのでしょうか? 答えは「Yes」です。

英語から日本語への翻訳では、英語を「適切な日本語に変換する」ことが重要です(ポイントは「適切な」です)。これには、英文の意味を理解する能力も重要ですが、「日本語の文章能力」と「ドメイン(分野/領域)の知識」も重要になります。

英文が流暢に読み書きできなくてもかまいませんが、日本語文章の書き方に慣れていないと、いわゆる「翻訳口調」の読みづらい直訳にしかなりません。そんな文章でもないよりはマシですが、こなれた文章のほうがずっと良いことはおわかりでしょう。

ドメインの知識については、たとえばIT系のドキュメントでは「default」を「デフォルト」あるいは「既定」と訳すのが通常ですが、金融系の文脈では「債務不履行」を意味します。「develop」も分野が違うとまったく違った言葉に訳す必要があり、写真の文脈だと「現像」となります。「develop images」だと「画像を開発する」ではなく「画像を現像する」としなければならないわけです。このようなものは前提知識がないと誤訳することが必至でしょう。

コミュニティでは、業務のようにプロの翻訳者に頼むわけにもいかず、必ずしもその分野に精通している人が訳すことにはなりません。かと言ってすべてをカタカナで済ませるのは日本語訳ではなく「ルー語訳」になってしまいますね(「グッドなトランスレイションをイナフするには?」のように)。

では、そのドメインの知識がない場合にはどうしたら良いのでしょうか? 前後の文章/コンテキスト/ユーザインターフェースから違和感を推測できる「嗅覚」が必要になります。これがないと先の例のような致命的なミスを犯すことになります。また、ミスを防ぐにはほかの人の目を借りる(ピアレビューしてもらう)ことが有用ですが、これは普段からコミュニティの参加者と十分なコミュニケーションを行って「信頼貯金を獲得」しておくのが重要です。

では、「相手が喜ぶこと」ってなんなの？という点ですが、そのコミュニティで「手が回っていないことを改善してあげること」が多いように思います。単なる指摘もありがたいのですが、汗をかいてくれるほうがもっとうれしい、ということは理解できるかと思います（汚れている場所を見つけたとして、「なんで掃除してないの？」と責め立てる姿勢ではなく、「汚れていたところを見つけたから、きれいにしておいたよ」というほうが良いよね、という話です。いや、我が家の話でもあるのですが……）。

しかし、たとえ相手が喜ぶことであったとしても、自分がうれしくない／楽しくないのであれば単なる「無償奉仕」となり、Contributionのモチベーションは消耗していく一方です。これはまったくフェアではないですし、活動の継続性も見込めませんので最終的にはお互いにとってデメリットとなります。「相手と自分がともに喜びや楽しみを分かち合えるポイントがどこにあるのか」を見つけるのが重要です。

Contribution——Debianの場合

ここからは、Debianを例にしてもう少し具体的に説明しましょう。

まず、Debian開発コミュニティへの参加について理解をしておくべきポイントは「Debian公式開発者（DD：Debian Developer）にならなければ貢献できないわけではない。1ユーザでもContributionはできる」という点です[注3]。

公式開発者になるには、ほかの公式開発者からの推薦を受けて申請し、メンターの補助のもとでさまざまな質問や課題が要求され、資格取得までにそれなりの気力を要するのは確かではあるものの[注4]、Contributionをするのに特別な資格などは必要ありません。

たとえば、DebianはBTS[注5]を介してパッケージその他の問題を収集しており、ここで進捗を管理／確認し、修正を実施／適用していきます。こう言うと大げさに聞こえるのですが、実はこのBTSへの投稿は「誰でも」行えます。利用可能なメールアドレスさえあれば良く、別のシステムのアカウントを取得する必要もありません。気になるバグがあれば、追加の情報を投稿したり、あるいは手元で修正できているのであれば修正のパッチを投稿したり、とさまざまなことが行えます。

ということで、DebianでBTSを介して自分で馴染みの深いソフトウェアのバグ報告作業に参加してみることを検討してみましょう。とはいえ、「でも、Debianでのバグ報告なんてやり方がわからないよ」という方が大多数かと思います。知らないことについては敷居が高く思えるものですので、この場を借りて「Debianでの

注3）DDは「自由にすべてのパッケージをアップロードできる」権限とプロジェクト運営での投票権を持っているだけで、パッケージのメンテナンス自体はDD以外の人（DM：Debian MaintainerやDDにスポンサーされたパッケージメンテナ）も行っています。

注4）「Debian公式開発者になるには？」というテーマについてはご要望があれば、筆者の実体験から紹介したいと思います。ぜひ感想をお寄せください。

注5）バグトラッキングシステム。詳細は URL http://bugs.debian.orgを参照。

COLUMN で、何から始めればいいのですか？

「私はこのプロジェクト／コミュニティで何をすればいいのか？」というのはプロジェクトに参加しようかな、と思った人に共通する疑問のようです。しかし、このような部分について丁寧なガイダンスをするプロジェクトやコミュニティは稀（まれ）ですので、「誰かがいつか自分の望む説明を丁寧にしてくれるという幻想」は早めに捨ててしまいましょう。ネットワークの遥か彼方の画面の向こうにいる人に対し、毎回手取り足取り説明することを要求するのは酷なものです。Face to Faceで状況を把握できるような仕事場とは違い、インターネットが主たるコミュニケーションの場である多くのプロジェクトは「来る者拒まず、去る者追わず」というスタンスであり、参加者には「何をするのが自分とプロジェクトにとってメリットになるのか？」を自問自答することがスキルとして最低限要求される、と理解しましょう。

バグ報告のお作法」をざっと説明いたしましょう……というところで誌面が尽きてしまいました。次回は具体例としてこのあたりを説明したいと思います。お楽しみに。

最近のトピック

最後にDebian界隈のトピックをいくつか紹介します。

kFreeBSDはDebian 8では非公式に

kFreeBSDはどうも進捗具合がよくないようで、Debian 8ではテクノロジープレビューでもなく、非公式リリースという位置づけになるようです。

Debian 9とDebian 10のコードネーム

ちょっと気が早いですがDebian 9、そしてDebian 10のコードネームが発表されました。Debian 9は「Stretch」(映画『Toy Story 3』に出てくるゴム製の紫色のおもちゃのタコ。)、Debian 10は「Buster」(同じく『Toy Story 3』に登場するペットのダックスフント)です。興味のある方は「ToyStory Strech/Buster」で画像検索などをしてみてください。

プロジェクトメンバーの多数の消耗、そして……

すでに結論が出たはずのsystemdへの移行……なのですが、いまだに不満の火種がくすぶっています。その結果としてsystemdパッケージメンテナチームから消耗したTollef Fog Heen氏が、技術委員会からはRuss Allbery氏、Ian Jackson氏がそれぞれ脱退することを述べています。

筆者からは「systemd採用のメリット／デメリットではなく感情的にsysvinitからsystemdへの移行を拒否する勢力と、争いの火に油を注ぎたいだけの人が多く、先の3名は自分の言いたいことだけを言っている人たちへの対処とノイズの多さに消耗していった」ように見えます。メディアなどもこういう諍(いさか)いごとは取り上げやすく記事として掲載するのですが、それを見てまた自分の意見だけを騒ぎ立てる人がさらに参加することで、この言い争いが続く要因となっているようです。

そして非常に残念なことに、前回の本連載で取り上げたdebhelper作者のJoey Hess氏がDebianプロジェクト自体からの脱退を表明しました。「自身が参加したころと空気が変わってしまった」というのが理由だそうですが、ただただ残念なことです。しかし、無理に引き留めても、彼とプロジェクトのどちらにも良いことはありません。せめて、彼の行く手に幸があらんことを祈りましょう。So Long and Thanks for all the Fish! SD

COLUMN パッチを送るために必要な技術力ってどれくらい?

最初に回答を要約すると「ゼロではだめだけど、高くなくても別に平気」です。

正直言って筆者は一からスクラッチで何かを書くというのは非常に苦手なのですが[注6]、パッチを作成することはよくあります。何かのソフトウェアを使っていてバグらしきものを発見した際、挙動から推測した正しいと思う動作と、誤っている現在の動作、そして(自分がまったく書くことのない言語で書かれたものであったとしても)ソースコードを見比べて突き合わせていけば、コードに書いてある意図がぼんやりと推測できて、たいていミスを見つけられるものです[注7]。そしてミスが見つかればあとはそこを1、2行程度だけ変えて直してやるだけで十分ですので、パッチを書くには「その言語で書ける」というほどの技術力は必要ない、というのが実感です。筆者はCもC++もPerlもPythonもRubyもPHPも書けませんが、パッチを作ったことはあります。そんなものですよ:-)

注6) なんと、基本である「hello, world」レベルもFizzBuzzも書けません。

注7) 元がひどい出来のソースコードではないことが前提ですが。でも、ソースコードが直に確認できるFLOSSというのは強みだな、と思うのはこんなときですね。

Ubuntuの10年を振り返る

Ubuntu Japanese Team
あわしろいくや　ikuya@fruitsbasket.info

やや旧聞に属してしまう話ではありますが、Ubuntuは14.10でリリースから10周年を迎えました。今回はその歩みを振り返ってみます。

Ubuntu 10周年

本誌2014年11月号の第55回でも少し触れましたが、2014年10月にUbuntuが10周年を迎えました。Ubuntuは1年に2回、4月と10月にリリースされます。これは当初から変更はありません。駆け足ではありますが、1月号ということで、今回はそれらを振り返っていきたいと思います。読者の方が最初にUbuntuを知ったバージョンはどのあたりなのでしょうか？

なお、バージョンの下の日付はリリース日とサポート終了日です。タイムゾーンはCDT（アメリカ中部夏時間）です。

Ubuntu 4.10 Warty Warthog

（2004/10/20～2006/04/30）

最初のリリースアナウンスは、Ubuntu創始者でCanonical社の創業者Mark Shuttleworthが投稿しています。Ubuntuの特徴として、

- CDに含まれるソフトウェアはすべてフリーソフトウェア
- 100%無料で、今後も有料化はしない
- セキュリティアップデートを18ヵ月間提供
- 半年ごとに最新版をリリース
- サポートするアーキテクチャはx86、AMD64、PowerPC

が挙げられています。また1週間後、すなわち2004年10月27日にライブCDもリリースされています。このころはまだライブCDからインストールできませんでした。インストールCDから最小限のサーバインストールは可能でした。

また、4.10はXFree86を採用する、最初で最後のUbuntuでした。5.04からはX.Orgに移行しています。今はもうすっかりXFree86なんて聞かなくなったので、10年という年月の長さを感じます。

Ubuntu 5.04 Hoary Hedgehog

（2005/04/08～2006/10/31）

5.04では初の公式派生版であるKubuntuが同時にリリースされました。デフォルトのデスクトップ環境がGNOMEではなく、KDEのエディションです。また、ライブCDも同時にリリースされています。

2005年7月8日にはUbuntu Foundationの設立が発表されています。同時に、6.04（当時）がLTS（Long Term Support）としてリリースされる旨の発表もありました。

Ubuntu 5.10 Breezy Badger

（2005/10/12～2007/04/13）

5.10ではインストールCD、ライブCDのほかにDVDイメージも用意されました。また、派生版とし

て教育機関向けのEdubuntuも追加されています。サーバ版が別イメージになったのもこのバージョンからです。

Ubuntu 6.06 LTS Dapper Drake

（2006/06/01～2009/07/14（Desktop）、～2011/06/01（Server））

2005年3月20日に、当初リリース予定だった4月20日から3週間延期して6月1日にリリースすることが発表されました。それに伴い、バージョンも変更されています。理由としては7点が挙げられていますが、そのうちの1つは日本語を含むアジア諸言語のサポートです。

予定どおり6月1日に6.06がリリースされましたが、たくさんの変更点がありました。まずはLTSの名のとおりサポート期間ですが、サーバ版で5年間、デスクトップ版で3年間に延長されました。インストールイメージも、今日見られるライブCDからインストールできるものがデフォルトになりました。以前のテキストモードのインストーラもAlternate CDとして残されています。

LTSは、これまでになかったメンテナンスリリースもあり、8月には6.06.1が、2008年1月21日には6.06.2がリリースされています。

新しい派生版として、Xfceデスクトップ環境をフィーチャーしたXubuntuも同時にリリースされています。採用したXfceのバージョンは4.4 Beta1とのことです。なお、Xubuntuの正式名称は“Xubuntu 6.06”ではあるのですが、どうも3年間のLTSだったようです[注1]。

Ubuntu 6.10 Edgy Eft

（2006/10/26～2008/04/26）

6.06のリリースは2ヵ月延びたにもかかわらず、6.10のリリースは延期なしでした。ということは開発期間が短かったということであり、大幅な変更はありませんでした。

Ubuntu 7.04 Feisty Fawn

（2007/04/19～2008/10/19）

このバージョンも割と落ち着いており、大きな変更点はありません。動画や音楽を再生する際に適切なコーデックがない場合、自動的に検出してインストールするしくみが備えられていますが、それはこの7.04から導入されました。デュアルブートにしてあるWindowsから個人データを取得して移行するツールも導入されましたが、後に廃止されています。

派生版として、Ubuntu Studioが追加されました。

Ubuntu 7.10 Gutsy Gibbon

（2007/10/18～2009/04/18）

やはり特段の大きな変化は見られません。Gobuntuという、完全にオープンなパッケージしか含まない派生版がリリースされましたが、8.04を最後に消えました。リリースアナウンスには“It is recommended only for experienced Linux enthusiasts.”と書かれていますが、そういう人たちはUbuntuではないものを使用するでしょうし、そのあたりが短命に終わった理由ではないかと思います。

日本ではほぼ使われない派生版のMythbuntuも、このバージョンから登場しています。

Ubuntu 8.04 LTS Hardy Heron

（2008/04/21～2011/05/12（Desktop）、～2013/05/09（Server））

2回目のLTSは順調にリリースされました。しかし、KubuntuはLTSではありませんでした。このころはKDE 3.xから4への移行期で、3年後の見通しが立っていなかったからだと思われます[注2]。これまでのKDE 3.5を採用したKubuntuと、4.0を搭載したKubuntu KDE 4 Remixの両方がリリースされたりも

注1）8.04のリリースノートに、6.06と7.10からのアップグレード方法が書かれていました。

注2）結果論ではあるものの、TDEという派生版がリリースされたこともあって3年サポートは可能であったように思います。

しています[注3]。

WindowsのパーティションにUbuntuをインストールできるWubiも8.04で登場しています。残念ながら現在はメンテナンスされていませんが。

やはり8.04も8.04.1から8.04.4までポイントリリースが提供されています。

おそらくこのころから日本でも知名度が上がってきたのではないでしょうか。壁紙が印象的だったのも一役買っていたように思います(**図1**)。

注3) ちなみに筆者は当時KDEユーザであり、ずっとKubuntuを使用していました。しかしKDE 4.xに移行する気はなかったので、Ubuntuユーザになりました。

図1　Ubuntu 8.04 LTS

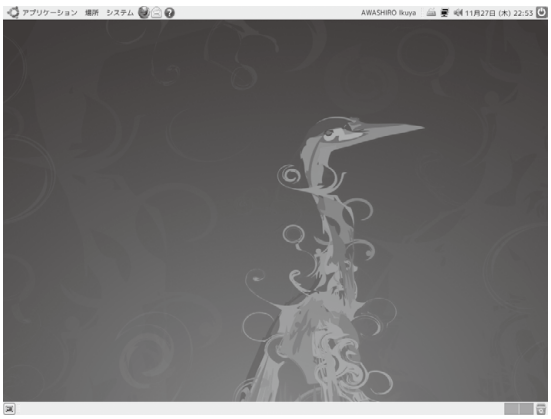


写真1　シャープのUbuntu搭載マシン NetWalker

Ubuntu 8.10 Intrepid Ibex

(2008/10/27～2010/04/30)

あまり大きな変化がないといえばそうなのですが、このころからインストールイメージをUSBメモリに書き出すしくみが導入されています。ということは、CDドライブがないPCが増えたということで、ネットブックブームのまっただ中であったことが伺えます。

Kubuntuは完全にKDE 4.xベースだけのものになりました。

Ubuntu 9.04 Jaunty Jackalope

(2009/04/23～2010/10/23)

このころからUbuntuは大きく変わっていきます。まず、Ubuntu Netbook Remix(UNR)というネットブック向けの派生版がリリースされました。今から思えば、このUNRこそがいろんなことのきっかけだったように思います。まずはランチャー。ネットブックの狭い解像度でも快適に使用できるようにと、netbook-launcherという専用のランチャーが誕生しました。これが後のUnityにつながります。当時から見ても非力なネットブックのために、起動や動作全般の高速化も図られました。また、UNR用にLPIAという専用のアーキテクチャを用意したものの、結局継続しないことになり、後にアップグレードするためには再インストールが必須になるという負の面もありました[注4]。

9.04といって忘れてはいけないのが、シャープからNetWalkerという9.04をプレインストールしたガジェットが発売されていたことです(**写真1**)。結局2モデルが発売されただけでラインナップとしては消滅しましたが、継続していたらきっといろいろとおもしろいことになっていたでしょうし、いまだに残念に思います。

注4) 専用のアーキテクチャを用意して最適化などを図る予定だったと聞いたことがありますが、結局そのようなことは行われず放棄されることになり、迷惑に思ったことをよく覚えています。

Ubuntu 9.10 Karmic Koara

（2009/10/26～2011/04/30）

日本のユーザにとって影響が大きかったのはインプットメソッドがSCIMからIBusに代わったところだとは思うのですが、GRUBがバージョン2になったり、ext4がデフォルトになったり、ARMに力を入れ始めるなど、かなり大幅な変更点があります。ほかにもUbuntu Oneのサービス開始もこのころでした。リソースを無尽蔵に投入しているように見えるので、いつかのタイミングでの整理は避けられなかったのだなと思います。

Ubuntu 10.04 LTS Lucid Lynx

（2010/04/29～2011/10/29（Netbook and ARM）、2013/05/09（Desktop）、継続中（Server））

今から考えても10.04は記憶に残るリリースでした。9.04や9.10での変更点が理想的な形で盛り込まれていました。とくに起動の速さは、当時触ったことがある人であればよく覚えているのではないでしょうか。

10.04は3度目のLTSであり、どちらかといえばサーバに大きな変更が見られます。後に発展的に解消するUbuntu Enterprise Cloud（UEC）のほか、Amazon EC2での動作も謳（うた）われていました。

LTSではないものの、Ubuntu Netbook Edition[注5]もリリースされています。このバージョンからLPIAサポートがなくなりました。

同時リリースではなかったものの、派生版としてLxdeをフィーチャーしたLubuntuもリリースされています。

10.04.1から10.04.4までポイントリリースが行われ、とくにサーバは現在でもサポート中ですので、まだまだいろいろなところで使われているものと思われます。

Ubuntu 10.10 Maverick Meerkat

（2010/10/10～2012/04/10）

2010年10月10日にリリースされたのは偶然ではなく、意図的なものです。これにより通常よりも2週間ほど開発期間が短くなりました。

特筆すべきは、なんと言ってもNetbook EditionにUnityが搭載されたことです（**図2**）。このUnityは次のバージョンでデフォルトになるので、結果的に素のGNOMEを採用した最後のバージョンがこの10.10で、同じルック＆フィールを持つ派生版が登場するのにはこの後4年ほどの歳月が必要となります。

Ubuntu 11.04 Natty Narwhal

（2011/04/28～2012/10/28）

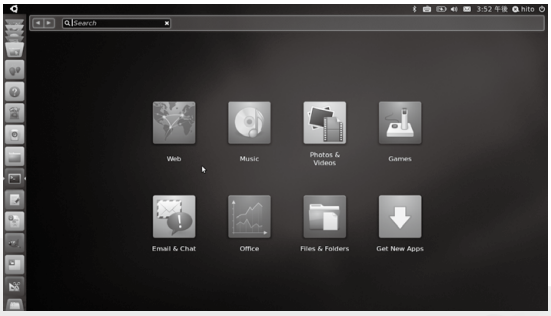
11.04ではデスクトップシェルにUnityを採用し、Netbook EditionとDesktop Editionが統合されました。厳密にはこのネーミングにも変更があり、Netbook EditionとDesktop Editionを統合したものを“Ubuntu”と呼ぶようになりました。Server Editionも同じくServerとなりました。ServerでOpenStackをサポートし始めたのがこのころです。

Ubuntu 11.10 Oneiric Ocelot

（2011/10/13～2013/05/09）

11.10での大きな違いは、ベースとなるGNOME

図2　Ubuntu 10.10 Netbook Edition



注5）10.04からはRemixではなくEditionと呼ばれるようになりました。

のバージョンが2から3に上がったことです。同時にカーネルのバージョンも3.0になっています。

Ubuntu 12.04 LTS Precise Pangolin

（2012/04/26～継続中）

4度目のLTSには大きな変更がありました。デスクトップ版のサポート期間が5年に延長されました。ただしCanonicalがサポートしない派生版に関しては3年のままです。ポイントリリースはカーネルやX.Orgスタックの更新を伴うものになりました。LTSまわりのことに関しては、本誌2014年6月号の第2特集に掲載されているので、気になる方はそちらをご覧ください。

CanonicalがKubuntuのサポートを行うのは、このバージョンまでということになりました。以後も別の会社に移管して継続中です。

兎にも角にも、現在もサポートは継続中です。

Ubuntu 12.10 Quantal Quetzal

（2012/10/18～2014/05/26）

このタイミングで、いろいろと整理が始まりました。どうにかして頑張ってCDサイズに収めていたインストールイメージはDVDサイズになりました。Alternate CDも提供されなくなりました。ブートにはPAE対応CPUが必須となりました。あとはセキュアブート対応もこのバージョンからです。

Ubuntu 13.04 Raring Ringtail

（2013/04/25～2014/01/27）

通常版のサポート期間が9ヵ月に半減しました。よって、12.10よりも13.04のほうがサポート期間が短いということになりました。通常版でのリリース飛ばしはできないものの、特例で12.10から13.10へのアップグレードができるように配慮されました[注6]。

新規の派生版として、中国向けのUbuntu KylinとGNOME ShellをフィーチャーしたUbuntu GNOMEがリリースされています。

Ubuntu 13.10 Saucy Salamander

（2013/10/17～2014/07/17）

日本のユーザにとっては、IBusのバージョンが1.5になって操作感が大幅に変更されたのが印象深いのではないでしょうか。現在でも若干混乱が続いています。

Ubuntu for Phoneの最初のリリースもありました。

Ubuntu 14.04 LTS Trusty Tahr

（2014/04/17～継続中）

本誌2014年6月号の第2特集をお読みください。特筆すべきこととしては、Ubuntu Oneのサポート終了が発表されたことでしょう。Ubuntu Oneの同期サービスで使用されていたサーバのソースコードは公開されるということでしたが、11月末現在でも公開されていません。

Ubuntu 14.10 Utopic Unicorn

（2014/10/24～継続中）

本誌2014年11月号のUbuntu Monthly Report第55回をお読みください。ただし一点訂正があり、Ubuntu MATEは結局公式フレーバーにはなりませんでした。

同人誌を出しました

最後に微妙な宣伝ですが、Ubuntu 10周年を記念した同人誌を発行しました[注7]。UbuntuのインストーラやCUPSやUbuntu Studioの歴史などが記載されています。同人誌ですので10周年に関係ない記事もあります。SD

注6） その後まもなく14.04へのアップグレードも必須なのですが。

注7） http://zapppaaan.freepub.jp/article/105631657.html

第34回 Linux 3.17の新機能 ～DRM機能とDRM render node～

Text：青田 直大　AOTA Naohiro

今月もLinux 3.17の新機能について紹介していきます。今月はLinux 3.17でデフォルトで有効にされるようになったDRM render nodeについてDRMの機能もまじえながら紹介します。

DRM render node

Linuxでは、DRM(Direct Rendering Manager)というフレームワークを用いて、GPUを操作し画面への描画を行っています。Linux 3.12では、このDRMの新しいインターフェースとなる“render node”というものが追加されました。しかし、この機能はデフォルトでは有効にならず、kernelの起動パラメータに“drm.rnodes=1”とあるときにだけ、ユーザランドから見えるようになっていました。Linux 3.17では、この“render node”機能が起動パラメータなしのデフォルトでも使用できるようになりました。これからDRMの機能を、昔からのインターフェース、そしてrender nodeを含めた新しいインターフェースという順番に紹介していきます。

DRMの3つの役割

DRMには大きく分けて3つの役割があります。1つ目はMode-Setting機能です。これは描画先行のモニタや、その解像度といったグローバルな設定の変更を行うものです。2つ目はメモリ管理機能です。アプリケーションはこの機能を用いて描画用のバッファを確保します。この領域のデータを操作することで、画面への描画が行われます。3つ目は前の2つの機能のための認証機能です。これについては前の2つを詳しくみたあとに解説します。

DRMを使ったサンプルコード

Mode-Settingとメモリ管理について詳しく知るために、実際にDRMを用いて画面に描画するコードを見てみましょう(**リスト1**)。

DRMは「/dev/dri/card<カード番号>」ファイルへのioctlによって実現されています。そのため、まずは「/dev/dri/card0」を開きます(**リスト1-①**)。このファイルは通常、ownerがrootで、groupがvideoでパーミッションが660となっていますが、udevのルールファイル(/lib/udev/rules.d/70-uaccess.rules)によって“uaccess”というタグが付けられています。その結果rootやvideoグループ以外であっても、画面を見ているアクティブなユーザに対して、ACL(Access Control List)による読み書き権限が与えられるようにできています(**図1**)。

▼リスト1　DRMを用いて画面に描画するコード

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <xf86drm.h>
#include <xf86drmMode.h>
#include <string.h>

struct fbinfo {
  struct drm_mode_create_dumb creq;
  uint32_t fb;
};

uint8_t *createfb(int fd, drmModeConnector *conn, struct fbinfo *fbinfo)  ……③
{
  struct drm_mode_create_dumb creq;
  struct drm_mode_destroy_dumb dreq;
  struct drm_mode_map_dumb mreq;

  uint32_t width = conn->modes[0].hdisplay,
           height = conn->modes[0].vdisplay;

  memset(&creq, 0, sizeof(creq));
  creq.width = width;
  creq.height = height;
  creq.bpp = 32;
  if (drmIoctl(fd, DRM_IOCTL_MODE_CREATE_DUMB, &creq))
    return NULL;

  uint32_t pitch, size, handle;
  pitch  = creq.pitch;
  size   = creq.size;
  handle = creq.handle;
  memcpy(&fbinfo->creq, &creq, sizeof(creq));

  uint32_t fb;
  if (drmModeAddFB(fd, width, height, 24, 32, pitch, handle, &fb))
    goto destroy_dumb;
  fbinfo->fb = fb;

  memset(&mreq, 0, sizeof(mreq));
  mreq.handle = handle;
  if (drmIoctl(fd, DRM_IOCTL_MODE_MAP_DUMB, &mreq))
    goto rmfb;

  uint8_t *buf;
  buf = mmap(0, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, mreq.offset);
  memset(buf, 0, size);
  return buf;

rmfb:
  drmModeRmFB(fd, fb);
destroy_dumb:
  memset(&dreq, 0, sizeof(dreq));
  dreq.handle = handle;
  drmIoctl(fd, DRM_IOCTL_MODE_DESTROY_DUMB, &dreq);
  return NULL;
}
```

（次ページに続く）

（前ページの続き）

```
void draw(int fd, drmModeConnector *conn)
{
  if (!conn->encoder_id) {
    printf("Encoder not connected¥n");
    return;
  }

  drmModeEncoder *enc = drmModeGetEncoder(fd, conn->encoder_id);
  if (!enc->crtc_id) {
    printf("CRTC not connected¥n");
    drmModeFreeEncoder(enc);
    return;
  }

  uint32_t crtc = enc->crtc_id;
  drmModeFreeEncoder(enc);

  /* フレームバッファの作成 */
  struct fbinfo info;
  uint8_t *buf = createfb(fd, conn, &info);
  if (buf == NULL) {
    printf("createfb failed¥n");
    return;
  }

  /* 元の状態を保存し、フレームバッファの割り当て */
  drmModeCrtc *saved_crtc;
  uint32_t connector = conn->connector_id;
  saved_crtc = drmModeGetCrtc(fd, crtc);  ……④
  if (drmModeSetCrtc(fd, crtc, info.fb, 0, 0, &connector, 1, &conn->modes[0]))
    return;

  /* 赤いラインを上から下へ */
  unsigned int x, y;
  uint8_t r = 255, g = 255, b = 255;
  for (y=0; y < info.creq.height; ++y) {
    for (x=0; x < info.creq.width; ++x)
      *(uint32_t*)&buf[info.creq.pitch * y + x * 4] = (r << 16) | (g << 8) | b;
    usleep(1000);
 }

  /* 元の状態へ */
  drmModeSetCrtc(fd, saved_crtc->crtc_id,
      saved_crtc->buffer_id,
      saved_crtc->x, saved_crtc->y,
      &connector, 1,
      &saved_crtc->mode);
  drmModeFreeCrtc(saved_crtc);

  munmap(buf, info.creq.size);
  drmModeRmFB(fd, info.fb);

  struct drm_mode_destroy_dumb dreq;
  memset(&dreq, 0, sizeof(dreq));
  dreq.handle = info.creq.handle;
  drmIoctl(fd, DRM_IOCTL_MODE_DESTROY_DUMB, &dreq);
}

int main()
```

（次ページに続く）

（前ページの続き）

```
{
  int i;
  /* カードを開く */
  int fd = open("/dev/dri/card0", O_RDWR);  ……①

  drmModeConnector *conn;
  /* コネクタ数、コネクタIDの取得 */
  drmModeRes *res = drmModeGetResources(fd);  ……②
  for (i = 0; i < res->count_connectors; ++i) {
    conn = drmModeGetConnector(fd, res->connectors[i]);
    printf("connector: %u¥n", conn->connector_id);
    /* モニタが接続されているか? */
    if (conn->connection != DRM_MODE_CONNECTED) {
      drmModeFreeConnector(conn);
      continue;
    }
    printf("¥tCONNECTED¥n");
    int j;
    /* 解像度の列挙 */
    for (j = 0; j < conn->count_modes; ++j)
      printf("¥t%ux%u %u¥n",
          conn->modes[j].hdisplay, conn->modes[j].vdisplay,
          conn->modes[j].vrefresh);
    draw(fd, conn);
    drmModeFreeConnector(conn);
    break;
  }
  drmModeFreeResources(res);

  close(fd);
  return 0;
}
```

▼図1　DRIカードデバイスのACLを確認

```
$ getfacl /dev/dri/card0
getfacl: Removing leading '/' from absolute path names
# file: dev/dri/card0
# owner: root
# group: video
user::rw-
user:naota:rw-
group::rw-
mask::rw-
other::---
```

次にioctl(DRM_IOCTL_MODE_GETRESOURCES)を用いて、DRMの環境情報を取得します。とはいえ、直接ioctlを叩くよりも、libdrmというラッパとなるライブラリがあり、これを使うのが簡単ですのでこちらを使います。

libdrmの場合、drmModeGetResources()が対応する関数となります(**リスト1-②**)。これによって、「コネクタ」の数といった情報を取得できます。コネクタというのは、ビデオカードやマシンのVGAケーブルやHDMIケーブルといったケーブルをつなぐ接続口のことです。drmModeGetResources()によって、コネクタのIDも取得できます。このIDをdrmModeGetConnector()に指定することで、モニタが接続されているかどうかといったコネクタの状態や、サポートされている解像度といった各コネクタの情報を取ることができます。

次に描画先であるコネクタ用のエンコーダと

CRTC(CRTコントローラ)を取得／設定します。先の説明に進む前にエンコーダとCRTCについて解説します。CRTコントローラは、解像度やリフレッシュレート、フレームバッファを設定する構造体です。ここで設定されたフレームバッファに描画されたデータは、エンコーダを通して変換されコネクタへとつながっています。一般にシステムには複数のCRTC、複数のエンコーダ、複数のコネクタがあります。これらは自由に組み合わせることができるわけではありません。**図2**のように特定のコネクタにしかつながっていないエンコーダ、特定のCRTCにしかつながっていないエンコーダが存在しています。すなわち、あるモニタ(コネクタ)に描画したいと思ったら、そのコネクタにつながるエンコーダと、そのエンコーダにつながるCRTCを検出する必要があります。

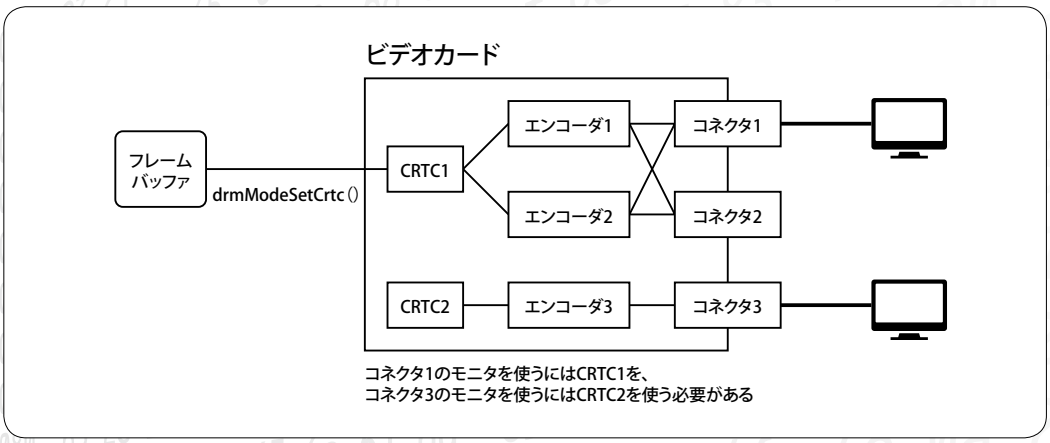
コードでは接続されている最初のコネクタに対して、draw()関数で、エンコーダとCRTCがすでに接続されているかどうかを確認しています。接続されていない場合は、システムのエンコーダ、CRTCをひとつひとつ調べて接続できるかどうかを見ていくことになりますが、今回のコードではその部分は割愛しています。

話を元に戻します。次に、createfb()関数のフレームバッファの確保へと移ります(**リスト1-③**)。この関数の目的は、メモリ領域として読み書きできるフレームバッファ領域を作ることです。まず、DRM_IOCTL_MODE_CREATE_DUMBを用いて、幅と高さがモニタと同じサイズのdumbバッファを作ります。dumbバッファは、ほとんどのDRMドライバで用いることができるシンプルなバッファです。mmapしてCPUからのレンダリング用途に用いることはできますが、GPUからはアクセスできないことが多く、シンプルな操作向けのバッファです。このioctlによって、バッファの“pitch”、“size”、“handle”の3つがDRMから返されます。pitch(またはstride)はあるピクセルのデータのアドレスと、その1つ下のピクセルのデータのアドレスとのバイト差になります。sizeはバッファ全体のサイズ、handleは、このdumbバッファのIDということになります。

このhandleを用いて、drmModeAddFB()を呼び出します。これによってフレームバッファオブジェクトが作成されます。このオブジェクトが後にCRTCに登録されます。さらにmmap用にdrmIoctl(DRM_IOCTL_MODE_MAP_DUMB)を使います。これによってoffsetが取得でき、それを使ってmmapされたバッファが取得されます。

ここまででフレームバッファオブジェクト、そして 描画用のメモリ領域が作成されました。まず、drmModeGetCrtc()を使って現在の状態

▼図2　エンコーダとCRTC



を保存し(**リスト1-④**)、drmModeSetCrtc()を用いてフレームバッファを設定します。これでメモリ領域を編集することで描画できるようになりました。コード例では左上から右下へと赤いラインを描画しています。

描画が終われば、crtcの状態を戻しさまざまなデータの解放を行っています。

Mode-Settingとメモリ管理

Mode-Setting APIはいわゆるXサーバやWaylandのcompositorであるWestonが用いるAPIとなります。前項のサンプルコードで言えば、drmModeSetCrtc()を用いてフレームバッファを設定した部分、すなわちモニタに描画されるフレームバッファが設定される部分だけがMode-Setting APIの役割となります。このほかにも、マウスカーソルやスプライトといったハードウェアによって合成されるデータの管理も行っています。

逆にメモリ管理に相当するのはそのほかのDRMの呼び出しになります。サンプルで紹介したシンプルなdumbバッファ以外にも、後述するようにアプリケーション間でバッファを共有するGEM(Graphics Execution Manager)などのインターフェースもあります。

DRM Masterと認証

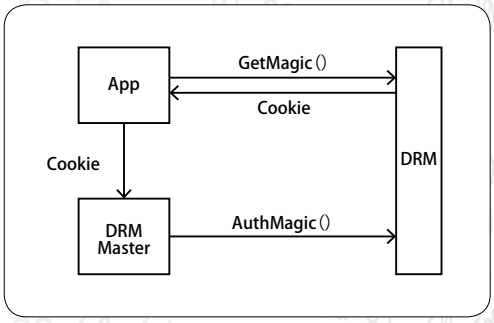
Mode-Settingには1つ重要な制限があります。もしも、すべてのアプリケーションが好き勝手にモニタに描画しようとする、すなわち好き勝手にそれぞれのフレームバッファを設定しようとすると、描画が混乱するといった問題だけでなく、場合によっては不正なブラウザ画面が描画されるといった問題が起こることも考えられます。そこでDRM Masterというファイルにモニタを扱う権利が占有されるようになっています。/dev/dri/card0を最初に開いたプロセス(多くの場合Xサーバ)がDRM Masterとなり、モニタ設定を管理します。そのほかのアプリケーションはDRMのもう1つの機能である認証機能を用いてDRM Masterプロセスに認証されることで始めて描画やメモリ管理といったGPUの機能にアクセスできます。

DRMでの認証は**図3**のような流れで行われます。まずアプリケーションはdrmGetMagic()を用いてDRMから一意のマジックナンバーを取得します。次に、取得したマジックナンバーをDRM Masterであるプロセスに送ります。DRM Masterは受け取ったマジックナンバーを用いてdrmAuthMagic()を用いて認証を行います。こうすることで、やっとアプリケーションはバッファの取得や共有ができるようになります。

バッファの共有には、GEM-flinkによる古くからの共有方法と、DMABUFという共有方法があります(**図4**)。GEM-flinkでは、まずバッファを共有しようという側が、最初にioctl(DRM_IOCTL_GEM_FLINK)を実行します。するとカーネルがグローバルなハンドラ(name)を返します。このハンドラのIDを別のプロセスに伝え、別のプロセスがioctl(DRM_IOCTL_GEM_OPEN)を使うと共有されたバッファを取得できます。

DMABUFによる共有は、基本的にはGEM-flinkと同じように進むのですが、カーネルから返されるのがハンドルではなくファイルデスクリプタであるという点が異なります。ハンドルはただの32bitの数字ですのでさまざまな方法でほかのプロセスに知らせることができますが、

▼図3　DRMでの認証



ファイルデスクリプタの場合はUNIXドメインソケットによる転送を行う必要があります。

render nodeの必要性

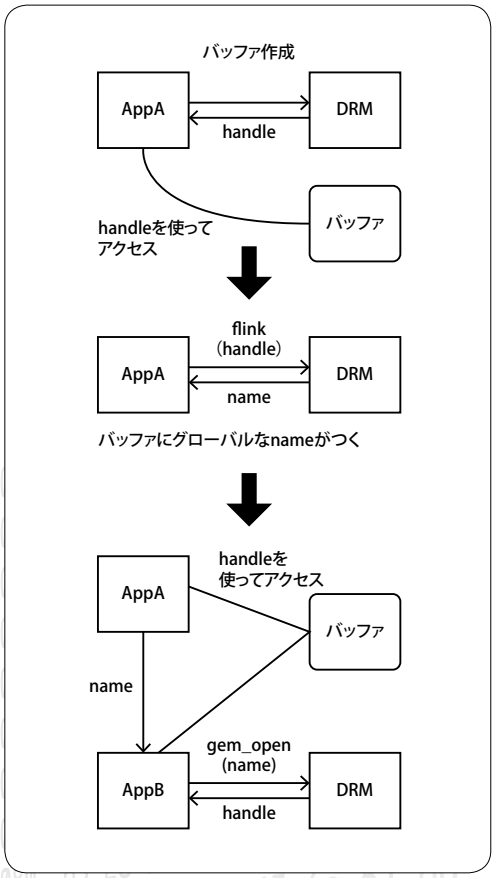
さて、ここまでで以前のDRMのシステムについての解説が終わりました。簡単にまとめると、1つのDRM Masterプロセスがモニタに対する描画の管理を行い、ほかのプロセスはDRM Master認証されることでGPUの機能にアクセスし、描画・バッファ共有を行うというモデルになります。

しかしながら、現在ではGPUはモニタへの描画のためだけに使われていません。GPUは、GPGPUといった計算だけの用途や、モニタへの表示を意図しないレンダリングにも使われるようになっています。こうしたアプリケーションにとっては、DRM Masterは邪魔な存在となってしまいます。たとえば、GPGPUを行うアプリケーションを2つ同時に走らせたいという場合、今までのモデルではどちらかがDRM Masterとなり、認証を行ってやる必要がありました。認証を行うためにはなんらかの方法で2つのプロセス間でCookieを渡す必要があり、そのようにプログラムを書かなければいけません。こうした場合、結局のところ、実際にはGUI環境を使わなくても認証のために(無駄に)Xを動かしておくということが行われていました。

この問題を解決するために導入されたのが“render node”です。render nodeでは、/dev/dri/card0から提供される機能のうち、メモリ管理・描画に関するものだけを「/dev/dri/renderD128」からアクセスできるようにしています。render nodeでは認証の必要がなく、そのためDRM MasterのためのXを動かすことなくGPGPUのプログラムを動かすといったことが可能になっています。

また、render nodeではインターフェースが新しくされたついでにセキュリティ的に問題のあったGEM-flinkが使えなくなっています。GEM-flinkによる共有では、一度認証されてしまえば、32bitの番号だけで自由にほかのプロセスのバッファへのアクセス権を手にできます。そのため、まったく関係のないプロセスが総当たりによってバッファへ不正アクセスをすることも考えられるというわけです。

▼図4 バッファの共有



まとめ

今月はDRMの機能について、これまでのシステムと、GPGPUアプリケーションをより簡単に動作させるための機能分割として生まれたrender nodeについて紹介しました。SD

January 2015

NO.39

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
榎 真治　ENOKI Shinji　shinji.enoki@gmail.com
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

中国地方転戦記

今回は、8月に島根、9月に広島で行った研究会の模様をお伝えします。

島根大会

■ディスカッション：オープンソースと企業とのかかわりについて
【講師】野田 哲夫（島根大学）、榎 真治（日本UNIXユーザ会）
【日時】2014年8月23日（土）10:15～10:30
【場所】松江テルサ 4階 中会議室

島根大会はオープンソースカンファレンス2014 Shimaneの1コマとして行い、参加者は約40名でした。

はじめに野田さんから、研究者の立場からのオープンソースソフトウェア（以下、OSS）研究の紹介がありました。エリック・レイモンド氏のコードを贈与する話[注1]から始まり、最近のアメリカでのキャリア追跡調査からは、OSSに携わってきた人は起業したり、Googleなどの大きな企業に雇われたりするケースが多いことがわかってきているそうです。贈与することによって、有名になったりキャリアや金銭につながったりしているのでは、という話でした。

また、OSSはフォークによって衰退していくのか、継続していくのか、という研究も行われているそうです。オープンイノベーションなど、OSSの成果を使うだけでなく企業が関与していくという動きがありますが、これに対してコミュニティの中には企業が関与することを嫌がる人がいます。それがフォーキングの一要因になっているという説があるそうです。

ここで、筆者（榎）からLibreOfficeがOpenOffice.orgからフォークした経緯を簡単に説明しました。OpenOffice.orgではボランティア貢献者のパッチはなかなか取り入れられない一方で（3年間もたなざらしにされたというケースも聞いたことがあります）、特定企業のパッチは簡単に取り入れられるなど、貢献者がフェアに扱われていませんでした。プロジェクト発足以来、パッチだけでなくあらゆることでそのような状態だったことが、フォークの原因として考えられる、と紹介しました。

野田さんからは、「とはいっても企業の貢献も必要ではないか」という問いかけがあり、これに対して筆者は、「LibreOfficeのような大規模なプロジェクトでは企業の貢献がなければ活動は難しいが、小さなプロジェクトではボランティアだけでも成り立つ可能性もある」という意見を述べました。また、「企業の貢献は開発などを加速させるので、コミュニティのルールを外れない限りはどのコミュニティでも歓迎されるだろう」という話をしました。また、野田さんからは、「OSSで活躍して有名になりたい、職を得たいというモチベーションがあるのでは？」という質問もありましたが、個人的にはそういう方にはあまり会ったことがないこと、企業に雇われるケースは以前に比べると出てきているが、まだまだ少なそうという印象を述べました。

最後に野田さんから、「日本でもアメリカの企業のようにOSSに貢献することで、開発者の獲得につなげるなどのモチベーションが高まってくるのではな

注1）URL http://cruel.org/freeware/noosphere.html

いか」という話がありました。

15分という短い時間の中で、野田さんからOSS研究をコンパクトにわかりやすく紹介いただき、コミュニティ側の実例と少しつなげる試みができてよかったです。一方で、ディスカッションとしては、これからというところで終わってしまってたいへん残念でした。このディスカッションの続編をjusのイベントとして実現したいと考えています。

広島大会

■これからのアプリ開発はIPv6対応で行こう！
【講師】渡辺 露文（IPv6普及・高度化推進協議会）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2014年9月20日（土）11:00～11:45
【会場】サテライトキャンパスひろしま 502会議室

3年ぶりの開催となった広島大会は、IPv6普及・高度化推進協議会で調査が進められているアプリケーションのIPv6対応をテーマに設定し、同協議会のメンバーである渡辺さんを講師にお迎えしました（**写真1**）。参加者は17人でした。

はじめに、最近はIPv6を利用できる環境が増えてきていることやIPv4アドレスの枯渇について触れたあと、もう少しIPv6を知ろうということで仕様の説明がありました。IPv4とIPv6は互換性がないので、サーバ側でIPv6に対応しないと、IPv6のみの環境とは通信できません。また、ネットワークとサーバだけがIPv6に対応すれば良いわけではなく、アプリケーションも対応が必要です。たとえばプログラムの中にIPv4アドレスを直接書いているとIPv6に対応できないので、ホスト名で指定するように変更する必要があります。

写真1　広島大会の様子

続いて本題であるアプリケーションのIPv6対応について話がありました。対応の基本方針は「IPv4とIPv6の両方で動作する」「1つのソースコードで対応する」の2点です。そして、具体的な対応ポイントを次の3つに分けて説明しました。

①IPv4/IPv6両対応のプログラミング言語やOSを使う
サーバのホスト名をDNSで名前解決したときにIPv4アドレスとIPv6アドレスが得られるが、そのどちらでも通信できるようにしておく。また、プログラミングにおいても、IPv4/IPv6の双方に対応するライブラリやデータ型を使う

②通信処理をIPv4/IPv6の両方に対応させる
サーバ側ではIPv4/IPv6双方の接続を処理できるように作る。クライアント側も同様だが、いずれかが接続できないときはもう一方に切り替えて接続するように作る。このフォールバック機能の作りが悪いと切り替えに失敗したり時間がかかったりしてしまい、ユーザの利便性を損なってしまうので注意が必要

③データとしてIPアドレスを扱う個所をIPv4/IPv6の両方に対応させる
具体的にはデータの入出力／検索／整列／格納などの処理が該当する。IPv4とIPv6ではアドレス体系や表記が異なるので、両方に対応できるようなプログラムにする。PostgreSQLなどIPアドレス型が定義されているソフトウェアではそれを使い、定義されていない場合は文字列型を使用し、IPv6アドレスは完全表記に変換して扱う

最後に渡辺さんから、「IPv6対応は要点を理解すればけっして難しいものではないので、今日から開発するアプリケーションはぜひともIPv6に対応してほしい」というメッセージがありました。全体的に要点がわかりやすくまとめられていて、とても良い講演でした。SD

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
及川 卓也　Takuya Oikawa
Twitter @takoratta
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha

第37回 第2回 ITx災害会議レポート

“東日本大震災に対し、自分たちの開発スキルを役立てたい”というエンジニアの声をもとに発足された「Hack For Japan」。今回は2014年10月に開催された「第2回 ITx災害会議レポート」の模様をお届けします。

災害へのIT活用を考える会議

ITを活用した復興支援や防災／減災を行っている人たちをつなぐためにスタートしたのが「ITx災害」コミュニティです。2013年の10月に第1回の会議を行った後、Facebookグループなどで情報交換などを行っていましたが、2014年の10月に第2回目の会議を行いました。

第1回目の会議以降、いくつかのプロジェクトが立ち上がりました。また、東日本大震災以降も雪害や水害など多くの自然災害が日本を襲っており、それらに対してのITからのかかわりを考える機会も多くありました。そこで第2回目となる今回は、実際に動き出そうという意味を込めて、テーマを「つながりx動く」としました。

今回はこの第2回 ITx災害会議の模様についてお伝えします。

午前中のショートスピーチ

会議の午前中は10名の方から「ITx災害コミュニティに期待すること」、「現在進めているプロジェクトの紹介」、「支援活動を通じて見えてきたこと」など、多岐にわたるテーマでショートスピーチを行っていただきました。Hack For Japanからもスタッフの及川がこれまでの経緯と翌週のCode for Japan Summitの中で行う防災・減災ハッカソンについて案内をしました。

登壇された方（敬称略）とその概要は、次のとおりです。

●東京大学CSISとしての東日本大震災以降の取り組み～地理空間情報と復興・防災・減災

▪古橋 大地・瀬戸 寿一　東京大学空間情報科学研究センター

「東日本大震災以降、地理空間情報がどうやって世の中の役に立てるかという観点で、復興支援アーカイブ、NHK震災ビッグデータ、アーバンデータチャレンジなどに取り組んできました。世の中を今よりももっとよくするために、地理空間情報を必要とするすべてのコミュニティを応援します！」

●情報支援レスキュー隊 IT DARTの活動

▪佐藤 大　IT DART

「現地に入って被災地の状況を把握し、後方支援チームと連携して被災地の外にいる支援を考える団体が動きやすくなるよう情報発信をしていくのがIT DART[注1]の活動の目的です。災害後100時間の緊急活動の中で被害や避難の状況、支援ニーズ、各支援団体の現状などを収集します。」

●支援者のための情報発信～平時、災害時の痛み悲しみを減らすために情報ができること

▪小和田 香　IT×災害情報発信チーム

「誰のための情報を届けるかということが大切。自分たちは支援者のための情報を届けるために活動を始めました。ITx災害情報発信については災害時に信頼性あるメディアから情報を収集、発信します。災害時自治体Twitter調査が2014年9月のNHK NEWSWEBで紹介されました。各地

注1　http://itdart.itxsaigai.org/

域のキーマンをつなぐこともやっていきたい。」

●災害時におけるIT支援活動の成果と課題
～調布、大島、前橋、広島での事例

▪柴田 哲史　災害IT支援ネットワーク

「被災地の災害ボランティアセンターに入って、Webサイト作成支援などを行ってきました。直近では広島の災害でも活動しました。公式サイトを公開すると問い合わせの電話の数が激減（3～4割）します。電話の問い合わせ内容はだいたい決まってるので、FAQの効果が大きい。」

●大槌における支援活動を通じて感じたこと

▪臼澤 良一　遠野まごころネット

「被災地では雇用の喪失、高齢化、コミュニティ不全、農業漁業の衰退、インフラの不備などの問題があります。きめ細やかな支援には、他の地域の事例ではなく現地に入り被災地の声に耳を傾け、その地域に息づいているものを掘り起こすことが重要です。」

●震災対策アプリ「ホイッスル on Android」
～小さなコード、大きな成果

▪安川 要平　ヤスラボ代表

「震災直後に“ホイッスル on Android”という震災対策アプリを開発しました。生存確率をあげるためにあなたに代わってSOSを発信するアプリです。笛はあると良いけどみんな持ってない、持っているものに笛が付けばというところから着想し、30行という短いコードで実現しているのですが、30万という数のダウンロードがありました。要望はいろいろといただくのですが、シンプルさを心がけてきました。」

●災害時に生き残るための知識を共有できるサービス

▪中塩 成海　一般社団法人イトナブ石巻 理事

「石巻で被災した実体験から災害時に生き残るための知識を共有できるサービスを開発しました。災害発生時は少なくとも1週間は自分で生きぬかなくてはいけない。そのようなサバイバル状態の災害時に必要な情報を被災前から考えられる場を提供し、被災時に速やかに共有します。Race for Resilience[注2]からスタートしたプロジェクトで、ネットが止まっても印刷物を提供・拡散するようなものを考えていきたい。」

●すごい災害訓練DECOの紹介

▪田口 空一郎　すごい災害訓練DECO

「災害対策には人材育成が最も重要で、自助のためのコーチングのしくみで目標に近づけていくしくみです。311の災害経験をどのように継承していくか、防災教育プログラムの開発を考えていたことがきっかけでした。iPadを使った災害対応訓練などを実践しています。」

●災害の経験から得た、災害発生時に備えた虎の巻

▪津田 恭平　一般社団法人イトナブ石巻 理事

「『まさかここまで津波がくるとは思わなかった』という実体験から万人が知識を持つことが防波堤を作ることよりも重要と考え、危険度を認識してもらうためにRace for ResilienceでFloodARというアプリを開発しました。ARアプリ上のアバターと同じ速度で歩くことで実際の避難にかかる時間を実感できるようになっています。」

昼飯とアンカンファレンス

午前中のショートスピーチの後、昼食には「模擬の炊き出しの体験」としてパックの弁当となめこ汁が提供されました（**写真1**）。昼食を担当いただいたいのは、小林幸生さん（NPO連携福島復興支援センター）です。なめことネギはいわき市産のものを使っています。

昼食に入る前に、午後のアンカンファレンスの進め方が説明されました。参加者は事前に配布された付箋紙にトピックを記入し、スタッフに手渡します。それをスタッフが昼食の間に整理します（**写真2**）。

注2　2014年2月に行われた世界銀行主催の防災、減災ハッカソン。本連載の9月号でもレポートしています。

スタッフによる整理の結果、アンカンファレンスの時間割は**表1**のようになりました。

アンカンファレンスまとめ

午後のアンカンファレンスの成果を紹介します。

「オープンデータ・官ができること」をテーマにした議論では、データを使って災害の経験をどのように伝承していくかを議論しました。著作権やその他のしがらみを乗り越えていくためにクラウドソーシングを活用し、権利関係に配慮したデータ作成の提案がされ、また、何をどのようにオープンにしていくかといった観点でのデータ形式や運用の必要性を訴えました。

「災害時情報発信」の発表は情報発信の観点を誰に(Who)いつ(When)どんな情報(What)をどうやって(How)届けるかを軸に議論しました。被災地域とそれ以外の地域や受け手の情報感度などの分類を行って、被災地からの生の情報発信が重要になるのではないかという仮説から、情報発信にITの力を使って付加情報を加えていくアイデアが出されました。

「マイノリティ・ハンディキャップ」をテーマにした発表は、日本で被災した外国人や障害を持つ人のような要援護者に対して、どのような支援ができるかを議論しました。そのような要援護者に対する情報の受発信や、情報を受け取った後の行動に結びつけるためのリテラシー格差を埋めるため、予防的な情報の整理や事前の情報公開を平常時に実施していくという施策を発表してくれました。

「災害ボランティア」の議論の発表では、ボランティアを運営する側としても、被災地外からボランティアにかかわる人々に求められる意識をあらかじめ明示しておく必要性を訴えました。受け入れ側の体制構築や継続的にボランティアを集めるために、わかりやすさと受け入れやすさを事前に用意する視点が大切と訴えました。

◆写真1　昼食のお弁当

◆写真2　アンカンファレンスの時間割を検討中

「連携」をテーマに議論したチームでは、組織間の連携時に起きる問題について発表してくれました。災害時に連携すべき組織が、日常における組織間での派閥争いの影響を受けてしまうなど、災害発生時の活動として本質的ではないところで問題が発生する可能性があります。これを回避するために、災害時の協定を事前に結んでおくことなどの事前策を中心とした議論でした。

「お金・持続可能性」のテーマでは、災害時の資金運営のワーキンググループを立ち上げたことを発表してくれました。教育や予防に対する予算が付きづらい、年度決済という構造のため継続的な企業の支援が得られづらい、人手が不足しがちなどの課題があります。これらを解決するため、下支えとなるITの技術をきちんと理解してもらい、予算が付いた活動となるための土台作りを支援するというコミットメントをしてくれました。

「防災教育」をテーマにした成果発表では、地域ご

◆表1　アンカンファレンス時間割

13:15～	IT DART	オープンデータ・官ができること	災害時情報発信	マイノリティ・ハンディキャップ	災害ボランティア
14:05～	IT DART 2	連携	お金・持続可能性	防災教育	コミュニティ運営
14:55～		ツール	産業復興	人材育成	

との特性に配慮した教育の必要性を説きました。どうすれば主体的に防災について市民が考えられるようになるかという視点から、浦安市で防災教育コンテンツを作った「すごい防災訓練DECO[注3]」の事例を紹介。地域の道という道を歩き、その地域特性を深く理解したうえで災害をシミュレーションすることが大切で、現場の暗黙知をいかに言語化していくか、地域の人々とのつながりをどう作るかを、活動の理解と共に各地に拡げていく必要性を訴えました。

「コミュニティ運営」の成果発表では、誰かがいつも必ず居て「集まる・共有する・共感する」ことができる場の用意や、行政任せではなく自分達自身で災害に備えるためにコミュニティを作り上げていく必要を訴えました。そうしたコミュニティの質を高めるためには構成員の理解、また緩いつながりの構築をしていくことの重要性を挙げています。今後の取り組みとして、コミュニティをサポートしていく中で、数%でも芽が出ればその後の活動につながるというポジティブな意識付けと、活動の失敗を記録することで持続性のある活動とし、最後は長いお付き合いを目指すということで締めくくられました。

「産業復興」をテーマにした議論では、被災地における産業復興に主眼を置きましたが、その根底には地方の産業復興という大きな課題を含み、2つに分けて発表されました。1つが既存産業の復興をプロボノ[注4]の協力を得ながら実現していく案。もう1つがシリコンバレーのような新しい企業が生まれやすい文化を作っていくために、大学と連携して新規産業の育成をしていくスキーム作りの案を提案してくれました。

「人材育成」ではITに関する教育で、我々Hack For Japanの活動やこの連載でも何度か紹介しているイトナブ石巻の活動なども事例として議論されました。その中から、小さくても良いからアウトプットを出すことや、達成するべき目標を可視化することの大切さ、メンターを用意することの重要性を訴えました。

「ツール」のセッションでは情報の集積、その分析、そして可視化を軸に話し合われ、他の発表でも挙げられたデータの標準化や緊急時のデータの取り扱い、システム間連携や可視化を行うためのツールを事前に準備していくための議論が行われました。出てきたアイデアとしては、被災者の状況にあわせ、状況判断のための材料をレコメンデーションしていくしくみ、平時の情報蓄積と災害時の差分を見るしくみ、などが発表されました。

以前から活動されている「IT DART」の発表は、これまでの取り組みの振り返りと今後のアクションプランの策定をテーマにした議論でした。情報を軸にした議論から情報の整理や構造化をするためのテンプレート作りのみならず、そのUIとUXまで踏み込んだものにしなければ本来の目的を実現できないという課題意識から、最終的なアクションプランとしては、情報の整理に向けての活動や支援を行っていくという包括的なものが発表されました。

第2回会議を終えて

今年で2回目となる「ITx災害」会議でしたが、東日本大震災をきっかけに始まった各団体の活動も3年以上経った今、そのほかの災害も視野に入れた幅広いものとなりつつあります。参加者の多くが共通した課題を感じていることを知ることができ、良い機会にもなりました（**写真3**）。読者の皆さんにとって、この記事が減災、防災という観点から事前にできることを平時に考えてみるきっかけとなれば幸いです。SD

◆写真3　会議を終えての記念撮影

注3　http://sugoisaigaikunren.org/
注4　各分野の専門家が職業上持っている知識・スキルや経験を活かして社会貢献するボランティア活動全般。また、それに参加する専門家自身。（Wikipediaより）

温故知新
IT むかしばなし

第40回

PSG音源

LINE㈱　佐野 裕（さの ゆたか）　Twitter：@sanonosa

はじめに

今回は1980年代のパソコンや家庭用ゲーム機などに用いられたPSG（Programmable Sound Generator）音源についてお話します。

PSG音源とは

PSG音源は、複数の基本波形を合成した音波とノイズ発生装置を組み合わせたサウンドチップです。PSG音源にはさまざまなバリエーションがありますが、一般的にはゼネラルインスツルメンツ（GI）のAY-3-8910および互換品を指すことが多いようです。AY-3-8910では矩形波発声3音とホワイトノイズ1音の計4音を同時に発声できました。GIのライセンスのもとに製造された互換品としては、ヤマハのYM2149やFM音源に含まれていたYM2203やYM2608、そして東芝からもMSXなどのパソコンに搭載されたチップなどがあります[注1]。

注1）FM音源登場以降の互換品チップにはSSG（Software-Controlled Sound Generator）音源と呼ばれるものがありました。

矩形波（くけいは）

PSG音源では矩形波が用いられました。矩形波というのは**図1**のようにカクカクした波形を発生するものです。矩形波では音が鳴る／音が止まるというのがデジタルに表現されます。ファミコンサウンドと評されるピコピコ音は矩形波で作られたきらびやかな音のことを指すことが多いです。PSG音源の独特のポップな音楽感、アーティストでいうとPurfumeやきゃりーぱみゅぱみゅさんたちの音楽に引き継がれていたりもして、PSG音源は1つのカルチャーだったんだなあと思ったりもします。

ホワイトノイズ

ホワイトノイズとはノイズという名のとおり雑音のことです。当時はホワイトノイズをうまく扱ってさまざまな効果音が表現されてきました。代表的な使われ方としてはドラムやパーカッションなどの音の代替です。生音には到底似つかない音色でしたが、ツツチャツ ツツチャツのようにドラムでたたくリズムでノイズを鳴らしてみると、なんとなくその音がドラムっぽく聞こえたものです。

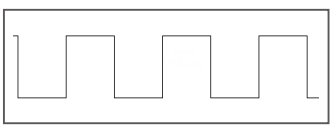
同時発音数

すでに記したとおり、PSG音源は3音程度しか同時発音ができませんでした。バンド演奏などでは、メロディー、ベース、和音（3〜5音程度）など、3音ではとても表現できません。ゲーム音楽などでは、メロディーとベースかサブメロディか和音のアルペジオを鳴らし、残り1音をミサイル発射などの効果音に用いる組み合わせが多かったように思います。

PSG音源を搭載していたパソコン

当時のパソコンは、ビジネスユースとパーソナルユースが明確に区分されておらず、各社から発売されるパソコンもいずれかに決めかねる中途半端なス

▼図1　矩形波



ペックのものが多かったように思います。当時パソコンを売るためにはゲームの充実が必須条件でしたが、そのゲームに欠かせないのが音楽でした。そのため、多少なりともパーソナルユースを意識したパソコンには漏れなくPSG音源が搭載されていました。

たとえばPSG音源を搭載していたパソコンとしては、NECのPC-6000/8000/8800/9800シリーズ、富士通のFM7シリーズ、そして各社から発売されていたパソコンの共通規格 MSXなどがありました。また家庭用テレビゲームとしても、PSG音源はセガマスタシステム、メガドライブなど、さまざまな機種に搭載されていました。

一方、任天堂 ファミリーコンピュータはPSG音源のように見えますが、実際はPSG音源を参考にして作られたと思われるCPU内蔵型音源が用いられたためPSG音源が用いられていたわけではありません。

MML

当時よく使われていたBASIC言語ではMML(Music Macro Language)と呼ばれる音楽用記述言語がよく用いられていました。MML言語を使うとBASIC言語からPSG音源を簡単に鳴らすことができました。

Lは音符の長さ(L1は全音符、L4は4分音符、L8は8分音符)、CDEFGABはドレミファソラシ、Rは休符の要領で記載していきます。また和音はカンマ(,)で区切って記載していきます。音楽家が楽譜を見るだけで頭の中で音楽を鳴らすことができたのと同様に、当時MML言語を使いこなしていた人たちは、MML言語で書かれたプログラムを見るだけで頭の中で音楽を奏でることができたような気がします。

BGMとしての利用

ゲームにはBGMが欠かせませんが、音を鳴らしながら画像出力やキーボードやジョイスティックなどの入出力処理を行うためには割り込み処理を駆使する必要がありました。BASIC言語でMMLを使って音楽を鳴らす場合、CPU処理がMML処理に奪われてしまいそのほかの処理ができず、ゲームのBGMを鳴らせませんでした。そのため、機械語などを使って適時割り込み処理を入れながら音楽を鳴らす必要がありました。

PSG音源は、音の出だしや音の停止をリアルタイムに指示する必要があります。現代の開発環境であればこういった低レベル処理はすべてプログラミング言語や各種APIに標準機能が用意されていますので難しくないですが、当時はそのレベルまで自身で開発する必要がありました。開発者はハードウェアに近いところまで知らないといけない時代であったため、自分の書くプログラムによって直接ハードウェアをコントロールしている実感があり、なかなか楽しい時代でありました。

FM音源

その後音色を自由自在に変えられるFM音源が普及していきました。FM音源にはPSG音源チップも搭載されていたため、実際はFM音源とPSG音源を同時に鳴らして音楽を鳴らすことが行われていました。

余談ですが、FM音源はさまざまな波長を組み合わせることによっていろいろな音色を作ることができ、伝説的なシンセサイザーであるヤマハのDX7で用いられたことが有名です。

終わりに

筆者が幼いころ、PSG音源+MML言語の組み合わせでずいぶん遊んだ想い出があります。ただいろいろ遊んでいく内に、MML言語だと楽器奏法的に言うビブラートやポルタメントなどといった表現が非常にしづらく、できたとしてもどうしても機械的で不自然な感じになるのが不満でした。この不満がその後筆者を生楽器演奏の世界に導き、高校時代に吹奏楽の世界に飛び込むきっかけとなりました。しかし実際に生楽器を経験すると、楽器はいくら練習してもなかなか思ったとおりに鳴ってくれないのに対してコンピュータはプログラミングしたとおりに鳴ってくれるので、音楽演奏はむしろコンピュータにやらせる方が楽だなあと思うようになったりもしました。SD

開発のボトルネックはどこだ?

迷えるマネージャのための プロジェクト管理ツール再入門

第3回 今どきのバージョン管理システムとは? Stashで実現する快適な開発環境

Software Design編集部

多くの開発現場でGitが使われている理由

ソースコードのバージョン管理を効率化するためのツールとして、これまで多くの現場で使われていたのがApache Subversionです。それ以前に使われていたCVS(Concurrent Version System)と同様の操作性を実現しつつ、CVSが抱えていたさまざまな課題を解決したことで、Subversionは人気を博しました。

ただ、Subversionにもいくつか難点があります。その中でもとくに大きいのは、複数の拠点で開発する際のレスポンスの問題でしょう。Subversionは中央のサーバでソースコードを集中的に管理するクライアント/サーバ型のモデルであるため、サーバから物理的に離れた拠点でアクセスすると必然的にレスポンスが低下し、開発効率にも影響が生じてしまいます。また、機密情報であるソースコードに遠隔地からアクセスするときにはセキュリティのためにVPNサービスなどの閉域網を使うケースが一般的ですが、利用者数の増加などによって広帯域化が必要となれば、コストの問題にも直結するでしょう。

そこでSubversionの代わりに広まりつつあるのが、分散バージョン管理システムである「Git」です。クライアント/サーバ型のSubversionとは異なり、Gitは複数の拠点や端末でリポジトリ(管理対象となるデータのまとまり)を管理できる分散型バージョン管理システムの1つであり、そのメリットから多くの開発現場ですでに使われています。

Gitの使い勝手を大幅に高めるStash

Gitでバージョン管理を行うには、まず全体のマスタとなる中央リポジトリを構築し、各開発者はマスタのリポジトリをクローンして自分のローカルリポジトリを作成します。

端末で実施した修正のコミットやブランチの作成、タグ付けといった作業は、ローカルリポジトリに対して行います。作業が完了したあとには「プッシュ」と呼ばれる操作でローカルリポジトリの内容を中央のGitリポジトリに反映します。

このように、普段はローカルリポジトリで作業を行い、作業が完了したときだけGitリポジトリにプッシュすればよいため、ネットワークのレスポンスを気にせず快適に作業を進められるのがGitの特長です。

このGitリポジトリを管理するためのツールとしてアトラシアンから提供されているのが「Stash」です(**図1**)。Gitをそのまま利用した場合、リポジトリやユーザの管理操作をすべてコマンドラインから行うことになりますが、Stashを使えばわざわざコマンドを覚えることなくWebインターフェース上で各機能を利用できます。

JIRAやJenkinsとの連携もサポート

StashのWebインターフェースは多機能で、バージョンやブランチの比較、ブランチのマージなどの操作を手軽に行えるほか、リポジトリ内のソースコードを参照するためのソースコー

▼図1　Webブラウザで Gitリポジトリを管理できる、アトラシアンの「Stash」。左はブランチのパーミッション画面、右はコードレビューの画面

ドビューアも用意されています。また、実際の開発現場でのワークフローに合わせて権限設定ができます。たとえばソースコードに何らかのバグがあり、ブランチを作成して修正を行った際に、レビュアーの承認がなければ作成したブランチをマスタにマージできないようにするなど、さまざまな設定ができます。

業務で利用することを想定した機能が用意されていることも特長でしょう。具体的には、ユーザごとにきめ細かくパーミッションを設定できるほか、リポジトリやプロジェクトの設定について誰がどんな変更をしたのかを監査ログとして記録するといったしくみが用意されています。これらの機能により、たとえばオフショア開発で外注先が作成したソースコードも集約して管理したいといったニーズにも対応できます。

課題管理ツールであるアトラシアンの「JIRA」との連携も、Stashならではの魅力でしょう(**図2**)。両者を連携させれば、JIRAで課題を作成した際に、その画面からStash側のブランチを作成してブランチと課題を紐(ひも)づけることができます。ブランチで実施したコードレビューやマージの履歴もJIRAの課題にリンクします。また、Stash側のコミットとJIRAの課題を結び付けられるため、どの課題を解決するためのコミットなのかが素早くわかるのも便利です。

CIツールであるJenkinsとの連携も可能で、Stashでのコミットをフックに Jenkins側で自動的にビルドを行うという環境も構築できます。すでにJenkinsで自動ビルド環境を整えている企業にとっては、うれしいポイントでしょう。

▼図2　課題管理ツール「JIRA」と連携できることも「Stash」の大きな魅力。Stashを操作するとJIRA側の課題のステータスを自動的に変更

このほかにも、Stashには生産性向上につながる魅力的な機能が多数盛り込まれています。現状のバージョン管理システムに課題を感じているのであれば、まずはStashの体験版で実際の機能を試してみてはいかがでしょうか。SD

Stashの無料体験版を提供中：
https://www.ricksoft.jp/product/atlassian/stash

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。

Software

ノベル、「SUSE Linux Enterprise 12」を提供開始

ノベル㈱は11月13日、「SUSE Linux Enterprise 12」の提供を開始した。同製品はエンタープライズ向 けOSの「SUSE Linux Enterprise Server 12 (SLES12)」を中心とした複数の製品から構成される。

SLES12では、Linuxカーネルは3.12に、サービス管理はSystemdに、デフォルトファイルシステムはBtrfsに、デフォルトDBはMariaDBに更新されている。また、保守作業などで問題が発生した場合にワンクリックでシステムを丸ごと復元できるフルシステムロールバック機能や、サービスを中断することなくカーネルを更新できるライブパッチ機能が搭載された。

さらなる高可用性を実現するための機能として、仮想／物理が混在したローカル環境でクラスタリングを実現する「SUSE Linux Enterprise High Availability Extension」と、遠距離間でのクラスタリングを実現する「Geo Clustering for SUSE Linux Enterprise High Availability Extension」もリリースされた。

同製品のWebサイト（**https://www.suse.com/products/server/download/**）からは60日間無料の評価版をダウンロードできる。

CONTACT ノベル㈱
URL http://www.novell.com/ja-jp

Hardware

Skeed、IBM Power Systemsベースの「SilverBullet Powered by LoP」を発表

㈱Skeedは11月27日、大容量・高速データ伝送ソフトウェア「SilverBullet」をIBMサーバ「Power Systems」に組み込んだアプライアンス製品の性能・機能を拡張し、「SilverBullet Powered by LoP」として発売開始した。

IBMの開発したプロセッサPOWER8 を搭載したPower Systemsは、スケールアウトモデルの高性能マシンであり、同時実行スレッド数が96と、x86サーバの82倍の処理性能を有している。Skeedが今回発売開始した「SilverBullet Powered by LoP」は、このハードウェアスペックを持つLinuxサーバ「Linux on Power」に、同社が独自開発した大容量・高速データ伝送ソフト「SilverBullet」を組み込んだもので、2014年1月17日に発表したPOWER7搭載モデルのエンハンス（性能・機能の拡張）製品として位置付けられる。

POWER8における、SilverBulletとFTPの転送速度比較実験では、4GBのファイル転送を50ミリ秒のRTT（遅延）／パケット損失率1%という近隣アジア諸国との間でよくみられる回線状況下で行った結果、FTPに比べ、SilverBulletは約80倍もの高速性を発揮した。

CONTACT ㈱Skeed
URL http://skeed.jp

Event

2015年2月3日・4日、「OpenStack Days Tokyo 2015」開催

2015年2月3・4日、日本OpenStackユーザ会主催の第3回OpenStack専門カンファレンス「OpenStack Days Tokyo 2015」がグランドプリンスホテル高輪（東京都港区）で開催される。

OpenStackは、クラウド基盤を構築するためのオープンソースソフトウェア。標準的なハードウェア上で、KVMなどの仮想化ソフトと組み合わせ、IaaSやストレージサービスを構築するための仮想マシンやストレージ、ネットワークの管理機能などを提供する。

「OpenStack Days」は、OpenStackの開発コミュニティのメンバー・開発ベンダ・ユーザ企業が一堂に会するイベント。今回は、「創る、活かす、つなぐ」をテーマとして、既存のオンプレミス環境とクラウド環境のハイブリッド運用に向けた導入・移行のポイント、コミュニティやエコシステムの活用法、OpenStackの今後など、さまざまプログラムが提供される。イベント初日には、OpenStackプロジェクトの創始者Mark Collier氏が来日し、基調講演を行う予定。

参加費は無料で、下記Webサイトから事前の登録が必要となる（締め切りは2015年1月27日17時まで）。

CONTACT OpenStack Days
URL http://openstackdays.com

Service

NTTスマートコネクト、「ハイブリッドクラウド接続サービス」「スマートスケーリングサービス」をβサービスとして提供開始

㈱NTTスマートコネクトは、クラウドサービスの利便性向上に向けて「ハイブリッドクラウド接続サービス（仮称）」と「スマートスケーリングサービス（仮称）」をβサービスとして提供することを発表した。

「ハイブリッドクラウド接続サービス」は、ユーザの環境にVMware NSXによる仮想ルータを提供し、オンプレミス環境とクラウド環境をシームレスに接続するサービス。オンプレミスとパブリッククラウドが混在することで複雑になりがちなシステムを、コントロールパネルにより効率的に統合管理できる。

「スマートスケーリングサービス」はユーザの利用形態に合わせ、仮想サーバ・仮想ロードバランサを生成しながらシステムリソースの自動拡張や縮退を行えるサービス。同社独自のオートスケールロジックに加え、ユーザカスタマイズでの設定が容易にできる。さらに、オートスケールで生成される仮想サーバのそれぞれにNSXの分散ファイアウォールを適用することによってマイクロセグメンテーション化し、負荷を高める原因となる仮想サーバへの不要な通信を遮断できる。

これらβサービスの提供期間は、2015年2月1日～3月31日。SI事業者をおもな対象として、機能性や利便性についての評価の収集、サービス化における最終確認を目的としている。

▲特設サイトのイメージ

また同社は、11月25日よりアニメーション作品「宇宙戦艦ヤマト2199」を各サービスプロモーションに起用している。

今後、特設サイトでは、人類滅亡の危機を救うため地球を再生するコスモリバースシステムを求めてイスカンダル星へ向かう「ヤマト計画」と、企業などがICT環境を「クラウド化」し、地球環境保護に貢献するイメージを重ね合わせ、クラウド化による環境負荷の軽減（低消費電力）やBCP対策などについて紹介する予定。

CONTACT NTTスマートコネクト㈱
URL http://www.nttsmc.com

Service

リンク、「ベアメタル型アプリプラットフォーム」においてテンプレート機能の提供、仮想化機能の強化開始

テンプレート機能を提供開始

㈱リンクは11月5日より、「ベアメタル型アプリプラットフォーム」において、テンプレート機能の提供を開始した。

「ベアメタル型アプリプラットフォーム」はGUIで物理サーバの追加・削除・コピーが行えるベアメタルクラウドサービス。今回追加された機能により、保存したイメージファイルをテンプレート化し、そこから新規サーバの作成などができるようになる。

同サービスではオプションとして「バックアップストア」という機能を提供しており、この機能を利用することで、現在の物理サーバのデータをイメージファイルとしてバックアップできる。「テンプレート」機能は、このバックアップストアを申し込むことで利用できる新機能。テンプレート化したデータをもとに、サーバの複製が可能となるため、ロードバランシング対象としてのサーバ複製などがより容易に行える。テンプレートから作成元となったサーバをリストアすることはもちろんのこと、別サーバへのリストアも行える。

さらに、このテンプレートは、作成元のサーバを解約した場合でもバックアップストア上に保存されるため、無駄なコストをかけずにテンプレートを保存し続けることができる。

仮想サーバの機能を強化

また同サービスは、11月26日より仮想化機能の強化を行っている。その第一弾として、リソース（CPU、メモリ）の変更が、サーバ作成後も行えるようになった。これによりCPUは1コア単位から、メモリは0.5GB単位から変更することが可能となり、運用するサービスの拡大に応じて仮想サーバのスケールアップが手軽にできるようになる。新たに追加料金などが発生することなく、運用開始後のサーバのリソース変更ができるため、より柔軟な運用が実現できる。

今後も、HA（High Availability）やVM（Virtual Machine）のバックアップ、V2V（Virtual to Virtual）／V2P（Virtual to Physical）／P2V（Physical to Virtual）のサーバ移行など、機能を拡充していく予定。

CONTACT ㈱リンク
URL http://www.link.co.jp

Hardware

アールエスコンポーネンツ、名刺サイズのスパコン「Parallella board」を発売

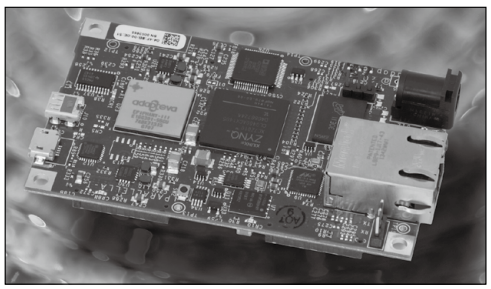
アールエスコンポーネンツ㈱は、米国Adapteva社と販売代理店契約を締結し、同社が開発／生産する名刺サイズのスパコンボード「Parallella board（パラレラ・ボード」（以下「Parallella」）を11月7日より、販売を開始した。

Parallellaは16個の演算コアで並列処理プログラミングが行える小型スーパーコンピュータボード。制御用SoCのXilinx社製「Zynq Z7000」（デュアルコアARMとFPGAを内蔵）と演算用アクセラレータ「Epiphany III 16 core」の2チップを名刺サイズの基板に搭載している。

安価でコンパクトながら、UbuntuなどのLinux OSを通じて、高速な並列処理プログラミングが行える。物理シミュレーション、ビッグデータの解析、教育用プラットフォーム、組込み機器の高速演算エンジンなど、幅広い用途で活用できるほか、複数のボードを連結拡張してより高速な並列処理アプリケーションを実現することも可能。

同製品は3機種を展開しており、Micro HDMIポート／Micro USB 2.0ポート／GPIOを搭載しない「サーバー版」（13,500円）と、サーバー版にMicro HDMIポート／Micro USB 2.0ポート／24 GPIOポートを搭載した「デスクトップコンピューター版」（16,600円）と、さらに48 GPIOポートとより上位のSoCを搭載した「組込み版」（26,700円）が存在する（価格はすべて税別）。

いずれも同社オンラインサイト「RSオンライン」（**http://jp.rs-online.com**）にて購入できる。



▲Parallella board

CONTACT アールエスコンポーネンツ㈱
URL http://rs-components.jp

Report

ロボットは東大に入れるか2014 ～東ロボくん、代ゼミ模試に挑戦～成果報告会

11月2日、代々木ゼミナール（以下、代ゼミ）本部校の代ゼミタワー（東京都渋谷区）にて、「ロボットは東大に入れるか2014～東ロボくん、代ゼミ模試に挑戦～成果報告会」が行われた。

「ロボットは東大に入れるか」は、国立情報学研究所が中心となって2011年に発足されたプロジェクト。2016年度までに大学入試センター試験で高得点をマークすること、また2021年度に東京大学入試を突破することを目標に研究活動が進められている。プロジェクトの主体となる人口知能「東ロボくん」は、受験科目ごとの「解答器」で構成されており、科目・問題種によって異なる手法を使い分ける。それぞれの解答器は専門家で編成された別々のチームが開発を行っている。

今回東ロボくんが受験したのは、代ゼミセンター模試および東大プレ（記述式：数学）。発表会では、それぞれの科目の解答器を開発したメンバによるシステムの説明・成績報告とともに、代ゼミ講師による成績の講評も行われた。成績結果は**表**のとおり、科目によっては受験生平均を上回る／引けを取らない点数を出している。2013年に同科目で受けた際よりも全体的に成績が向上しているとのこと。

開発メンバからは「単語の定義から理解させたい」（英語）、「誤答の傾向も人間に似せていきたい」（世・日・政）といった意見が出た。一方、代ゼミの講師陣からは「暗記が得意」「一般常識に疎い」「図をよく見よう」などの講評がされた。

▼成績表

代ゼミセンター模試		
教科	東ロボくん	受験生平均
数学ⅠA（100）	40	47.1
数学ⅡB（100）	55	50.4
物理（100）	31	31.7
英語（200）	95	93.1
国語（150）	69	60.2
世界史B（100）	52	40.8
日本史B（100）	44	47.2
政治経済（100）	17	38.1
合計（950）	403	408.6
代ゼミ東大プレ（数学）		
教科	東ロボくん	受験生平均
理系選択（120）	36	26.8
文系選択（80）	32	25.9

※括弧内の数字は満点

現時点では、東ロボくんが問題を問題として認識するための形式表現（XML、MathML、注釈付イラスト）に直すというプロセスを間に入れているが、今後は自然言語のまま入力に与えるよう実装する動きもあるという。

CONTACT ロボットは東大に入れるか
URL http://21robot.org

Report

グレープシティ、「Forguncy使い方セミナー」を開催

グレープシティ㈱は11月27日、エッサム本社ビル（東京都千代田区）にて「Forguncy使い方セミナー」を開催した。

「Forguncy」は、グレープシティ開発のソフトウェア製品。Microsoft Excelと同様の操作でWebアプリケーション（以下、アプリ）を簡単に開発できる。

セミナー前半の＜基本編＞では、同社の八巻雄哉氏がForguncyの基本的な操作をデモを交えて紹介した。

Forguncyのインストーラは、開発ツールとWebサーバ（運用環境）の2種類がある。開発ツールでアプリを開発し、Forguncy専用Webサーバに発行する。そのForguncy専用Webサーバ上でユーザがアプリを使用するという流れだ。

開発するアプリひとつひとつにDBが内蔵されており、ユーザ管理機能も持っている。専用Webサーバには複数のアプリケーションを発行でき、同様にアプリごとにDBを持つことができるとのこと。

「Forguncy開発ツール」を使えば、Excelの画面と同じような見た目や操作でアプリを開発できる。開発ツールにはテスト用のForguncy専用Webサーバが内蔵されているので、デバッグ実行が可能となる。

既存のExcelファイルをForguncyにインポートすると、レイアウトだけでなく、数式やセルの書式設定もインポートされる。Forguncyの開発環境で発行してForguncy専用Webサーバで開くと、Webアプリになっても書式や数式が反映され、Webページ上で値を変えると、数式が適用され、値も変わる。

後半の＜実践編＞では、簡単なサンプルアプリ「見積書発行システム」をゼロから作る過程が解説された。これは、請求書一覧から会社名を選択すると、その会社の請求書一覧が表示され、新たに請求書を作成することもできるというもの。参加者は、その場で実際に手を動かして、アプリ作成を体験した。

また、セミナーではForguncyの開発チームメンバに直接質問する時間が設けられた。評価版をダウンロードしたものの使い方がよくわからないといった質問や、自分でアプリを作ってみたがデータの連結がうまくいかないといった質問が出た。

CONTACT グレープシティ㈱
URL http://www.grapecity.com

Service

IDCフロンティア、「ioMemory PX600」を搭載した、ベアメタルサーバおよびクラウドサービスを提供開始

㈱IDCフロンティアは、ベアメタルサーバのラインナップに国内で初めてフュージョンアイオー社の高速Flashストレージ「ioMemory PX600」を搭載したサーバを追加し、11月10日よりサービスの提供を開始した。

また、併せてクラウドコンピューティングサービス「IDCFクラウド」でも、同型のサーバを用いたハードウェア専有のハイパフォーマンス仮想マシンタイプを11月19日から提供開始している。

これにより、ユーザはデータの読み書き両方に高いI/O性能を持つ2つのサービスを、クラウドやハウジングなどと自由に組み合わせて使うことができ、大量のデータ処理をより高速に行うことが可能となる。クラウドサービスの特長は次のとおり。

- 既存Flashストレージ搭載タイプと比較しI/O性能が約5割向上
- 月額固定または上限付き従量課金を利用形態に応じ選択可能
- IDCFクラウドハードウェア専有タイプはポータルでサーバをワンタッチで作成可能
- 高いI/O性能を持ちながら月額10万円を下回るハードウェア専有タイプも登場予定

▼ベアメタルサーバの利用プラン（税別）

プラン	高速IO1000			
CPU	Xeon 8コア ×2			
ディスク	292GB ／ハードウェア RAID10			
Flash ストレージ	ioMemory PX600 1000GB MLC			
メモリ（GB）	32	64	128	192
初期費用	0円			
月額料金	158,000 円	163,000 円	173,000 円	183,000 円

▼IDCFクラウドの利用プラン（税別）

プラン	HighIO 5XL128 ハードウェア専有タイプ
CPU	40 コア
Flash ストレージ	ioMemory PX600 1000GB MLC
メモリ（GB）	128
従量料金（時間）	370 円
月額上限料金	179,300 円

CONTACT ㈱IDCフロンティア
URL http://www.idcf.jp

ひみつのLinux通信

作)くつなりょうすけ　@ryosuke927

年末年始スペシャル
ITエンジニア出世双六

ITエンジニアの出世は恋愛フラグにあり！ どんなデスマーチもラブコメ要素があれば乗り切れる！

Letters from Readers

ハードウェアスタートアップの魅力

ネットのニュースサイトで、ハードウェア製品のスタートアップ記事をよく見かけます。折り畳みバイクや骨のない傘、飼い猫用の回し車などなんでもありで、そのアイデアにはいつも驚かされます。プロジェクトの多くは「kickstarter」などのクラウドファンディングを支援する企業を通して資金調達を行っており、アイデアの良さ、実現の可能性をいかにプレゼンできるかが成功への鍵となります。

2014年11月号について、たくさんのお便りをありがとうございました！

第1特集　無理なくはじめるInfrastructure as Code

インフラストラクチャの構成管理をコードで自動化する「Infrastructure as Code」。その手法を、Webアプリケーション開発の定番構成「LAMP」を題材に、仮想化ソフト・シェルスクリプト・クラウドサービスを通して学ぶ特集でした。

今一番興味のある内容でとても参考になりました。プログラマブルなインフラ。パラダイムシフトが起こっているのを実感します。

神奈川県／jacoさん

LAMP環境の歴史も含めてとてもわかりやすい記事だと思います。

東京都／tekitoizmさん

設定変更を繰り返すうちに何を直したのかわからなくなることはよくある。コード管理することで履歴管理したり冪等性（べきとうせい）が担保されたりするのはたいへん便利だと思った。

岩手県／隼さん


設定や変更履歴が確認しやすくなる、同じ設定ファイルを使えば誰でも同じ環境が組めるなど、インフラをコードで自動化する利点は多いです。特集では「Infrastructure as Code」を実現する複数の方法を紹介したので、気になったものがあればぜひ試してみてください。

第2特集　サーバの目利きになる方法【後編】

物理的なサーバマシンに立ち返り、学習する前後編の特集。後編となる今回はネットワーク機器の選定基準、主要なストレージの紹介、サーバの管理機能の3章立てで、サーバまわりの重要な知識を取り上げました。

ハードウェアの記事は少ないので、たいへんありがたいです。

東京都／山下さん

サーバをどのように選べばいいか、判断の1つになる感じだった。

静岡県／ももんがさん

実務経験をたくさん積まないと得られないような選定ポイントが簡潔にまとめられており、とても参考になった。

千葉県／若山さん


実際にサーバマシンを調達する立場にある人にとっては、非常に実用的な記事になったかと思います。そうでない人には、自分たちが利用している／開発しているシステムがどのような物理的資源の上で動いているかといった、貴重な知識を得られたのではないでしょうか。

一般記事　CPU温故知新

Intel 8008 CPUから最新のCPUに至るまで、命令セットやレジスタ構成など、ソフトウェアからみたCPUの発展の歴史を振り返りました。

より深い技術的理解の助けになると思う。

東京都／HIさん

ハードウェア寄りの歴史を振り返る記事は1つの節目として興味をそそります。CPUはIntelだけではありませんし、なかなかお目にかかれるものではありませんが、IBMのハードウェアやOSなどには感心するところが多く、記事として出てくることを期待しています。

熊本県／鈴木さん


CPUの進化にはめざましいものがあり、専門外の人間には何が起こっているかわかりづらいものがあります。しかし、その発展の歴史を1からたどることで、今後どのような方向へと進んでいくのかが見えてくるのではないでしょうか。

一般記事 Mackerel入門

「はてなブログ」で有名な㈱はてなが、自社ツールを基に開発したサーバ管理ツール「Mackerel」を紹介しました。さまざまな外部ツールと連携してシステムを監視できるのが特徴です。

この分野には明るくなかったので、わかりやすく書かれていて助かりました。

宮城県／オミオさん

サーバ管理もソフトウェアの1ジャンルになったと思います。

奈良県／捨てられないが役に立たないさん

何十、何百台ものサーバの状況を、外部のツールと連携しながらわかりやすいグラフで管理できるツール。非常に注目度が高いようです。ちなみに、Mackerel＝鯖（サバ）というのはご存じでしたでしょうか?

一般記事 Jamesのセキュリティレッスン【1】

ネットワークを流れるデータのかたまり「パケット」を解析するためのソフト「Wireshark」。その最新バージョン1.8.0で導入された新しいファイル形式「pcap-ng」を紹介する短期連載です。第1回では既存のファイル形式「pcap」の構造を細かく見ていきました。

キャプチャを解析しなくちゃ。

愛知県／kmさん

pcapしか使ったことがなかったため、おもしろかった。

東京都／匿名希望さん

ネットワークでは毎日すさまじい数のパケットがやりとりされています。そのひとつひとつはけっして単純な作りではなく、層状の複雑な構造をしています。その構造を意識しながら「Wireshark」を使って、実際にパケットの中身をみてみましょう。

一般記事 SoftLayerを使ってみませんか?【3】

SoftLayerの使い方講座、連載3回目となる今回は実際にサーバを構築するうえでのTipsや、SSL、ファイアウォール、ロードバランサなどのネットワークの構成方法を紹介しました。

クラウドサービスの技術が学習できるのでいい。

長崎県／Splictさん

この記事のサンプルコード例に付箋紙が貼ってありました。あとで打ち込んでみるつもりだったようです。

千葉県／Tayuさん

SoftLayerについて、実践的な内容を扱いました。設定を通じて、クラウドサービスでどんなことが実現できるかを知ることができ、サーバやネットワークの基礎的な勉強にもなるでしょう。本誌を片手にぜひチャレンジしてみてください。

フリートーク

テレビやレコーダーをはじめさまざまな家電製品がネットにつながるようになってきました。これらの機器のセキュリティ対策について知りたいです。

千葉県／澤下さん

今月号「セキュリティ実践の基本定石（P.126～）」では、IoTのセキュリティについて詳しく解説しています。ぜひご覧ください。

海外で電子版を定期購読しています。技術的な雑誌を毎月、日本語で読めるというのは、海外にいる技術者／研究者としてはとてもありがたいです。スマホで移動中にも読める、そんな時代の恩恵も受けている感じですけれども。

神奈川県／quazmaさん

場所を問わず発売後すぐに読めるのは、電子書籍の利点ですね。本誌は2014年5月から毎月、紙媒体と同時にPDF版を販売しており、そちらで読まれている方も多いようです。

表紙について

犬は良いですね。むささびもかわいいです。亀もかわいいです。

愛知県／かっとびめんど～さん

2015年度の表紙は何がいいでしょうか? 読者アンケートにてご意見お待ちしております!

11月号のプレゼント当選者は、次の皆さまです

①ジュエリーボックス
東京都　海野秋男様

②USB 3.0 Flashメモリ「Extream」
宮城県　赤畑慶幸様
北海道　杉木恵様

③揉まれる肩・首スッキリピロー
千葉県　卯木輝彦様

④無敵の天才たち
神奈川県　丸山貴之様
東京都　中澤直也様

⑤Amazon Web services入門
広島県　藪兼智英様
神奈川県　塚越厚英様

⑤アカマイ～知られざるインターネットの巨人
神奈川県　赤羽永寿様
千葉県　平井幸三様

⑥内部構造から学ぶPostgreSQL設計・運用計画の鉄則
千葉県　今井英敏様
愛知県　権田裕昭様

次号予告

Software Design
February 2015

2015年2月号
定価（本体1,220円+税）
176ページ

1月17日発売

［第1特集］使ってよかった!

systemd入門
あなたの知らない実践技

RHELをはじめとしたLinuxディストリビューションで導入されて久しいsystemd。エンジニアにとって頭の切り替えが必要なこの機能体系を、基礎から学び現場で活かす方法を紹介します。

［第2特集］ユーザ側も開発側もWin-Win

そろそろやめませんか？　運用でカバー
押しつけ型開発の終焉

■UNIX用語読み方講座

sudo、lib、awk、ext3、Btrfs……ちゃんと声に出して読めますか?

※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

「シェルスクリプトではじめるAWS入門」（第10回）は都合によりお休みさせていただきます。
「ハイパーバイザの作り方」はしばらくの間お休みさせていただきます。

SD Staff Room

●今年の売れ行きベスト3は、5月号TCP/IPネットワーク図解、8月号ログを読む技術、9月号のポインタとオブジェクト指向（付録付き）でした。これらはおかげさまで完売です。そして現在発売中の「インフラエンジニア教本」は再編集した厚さ2cmのMegaMixムックですが、続編も予定しています。（本）

●本誌ではVimを特集したが、自身ではWZ8とCotEditor2.0を使っている。指が慣れるという意味では、さんざんTurbo-Cで使っていたダイヤモンドカーソルが懐かしい。Ctrl+Cがコピーになってしまい、しばらく慣れなかったのは遠い思い出。今はCotEditorでRubyスクリプトを書くのが楽しみ。（幕）

●この編集後記を書いている今、あと1時間後に「はやぶさ2」が打ち上げられようとしています。戻ってくるのは6年後の2020年。そのときに自分の成長に胸を張れるか。壮大な夢のあるミッションにあやかって、6年後の自分のイメージを考えてみようと思います。無事の帰還を祈ります!（キ）

●事後報告ですが、Letters from Readersで掲載してきた「エンジニアの能率を高める一品」は前号でいったん終了となりました。これまでの全33製品におけるマイベスト3は、ゆびトング付き菓子ボウル（2014年9月号）、ブックストッパー（2013年8月号）、ルルドマッサージクッション（2012年8月号）です。（よし）

●人生で初めて、電動歯ブラシを使って歯を磨きました。あまり手を動かす必要がないので腕が疲れにくく、普通の歯ブラシを使ったときよりも歯がツルツルになります（気のせい?）。ただ、思った以上に振動が強いんですよね。振動に慣れていなかった最初の数回は、なぜか頭痛に悩まされました。（な）

●先日無事結婚式を終えることができました。諸事情により準備期間が長かったので、慌ただしくならないようにと思っていましたが、直前でないとできないこともあり、結局お休みをいただきました。おかげさまで当日は幸せな気分で過ごせました。ただ現実に戻ると家の中は大混乱。今度は大掃除に励みます。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2015年1月号

発行日
2015年1月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
松本涼子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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