Special Feature 1 Web開発の今
Special Feature 2 COBOLの本質

Software Design

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

2016 March 3

2016年3月18日発行
毎月1回18日発行
通巻371号
（発刊305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

Special Feature 1

チーム開発をまわす

現場のアイデア

なぜすぐリリースできるのか

Case 1 { SUUMO } Case 2 { Retty } Case 3 { Qiita }

Special Feature 2

好き嫌いで判断していませんか?

あなたの知らないCOBOLの実力

経済を支える基盤技術

特別企画

Webサイトオーナーが改ざん被害に気づいたときにとるべき行動

～ペンコンピュータの軌跡～
iPad Proのさきに見えてくるもの

新連載

RDB性能トラブルバスターズ奮闘記

宮原徹のオープンソース放浪記

2016
March
3
Software Design

第1特集

なぜすぐリリースできるのか

チーム開発をまわす 現場のアイデア

{ Case 1 SUUMO }{ Case 2 Retty }{ Case 3 Qiita }

contents

2016
March
3
Software Design

contents

第2特集

好き嫌いで判断していませんか?

あなたの知らないCOBOLの実力 061

一般記事

短期連載

アラカルト

Column

Development

OS/Network

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

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。

技術評論社の 確定申告本

平成28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28年3月締切分

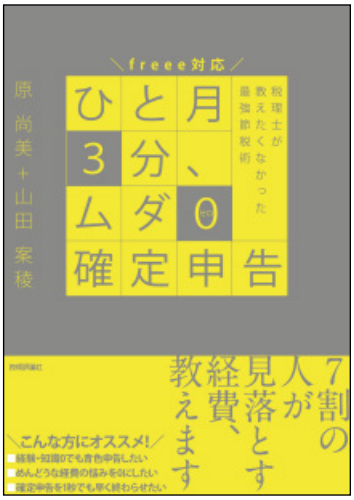
山本宏　監修／A4判／176ページ
定価（本体1,380円＋税）
ISBN978-4-7741-7684-0

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。ほかにもアルバイトやパート、フリーランス、不動産オーナー、年金生活者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。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オススメです！

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 第2版

大村大次郎　著
A5判／224ページ
定価（本体1,580円＋税）
ISBN978-4-7741-7664-2

税金には、シロなのかクロなのかはっきり線引きされていないグレーゾーンがいくつもあります。その境界線を賢く活用することで、確定申告は絶好の節税の機会になります。フリーランス・個人事業主の皆さんが確定申告するときにぜひ知っておいてもらいたいことを元国税調査官の著者が厳選しました。はっきりシロと認められていることでも、納税者に有利になる（＝税金が安くなる）確定申告のやり方を税務署がすすんで教えてくれることはありません。自ら知識を仕入れて、確定申告でトクする人になりましょう！

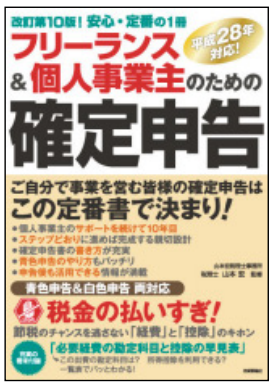


ひと月3分、ムダ0確定申告

税理士が教えたくなかった最強節税術

原尚美・山田案稜　著
A5判／272ページ
定価（本体1,580円＋税）
ISBN978-4-7741-7792-2

「勘定科目なんて知らなくていい」「仕訳なんてしなくてもいい」最高に簡単な確定申告の解説書です。経費で落とせる領収書と落とせない領収書といった悩みどころもしっかり解説。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」にも対応したかつてない確定申告本です！



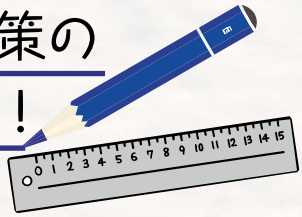
フリーランス&個人事業主のための確定申告 改訂第10版

山本宏　監修
A5判／240ページ
定価（本体1,480円＋税）
ISBN978-4-7741-7665-9

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

あなたを
合格へと導く
一冊があります！

効率よく学習できる
試験対策の
大定番！



金子則彦　著
A5判／536ページ
定価（本体3200円+税）
ISBN978-4-7741-7708-3

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

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

加藤昭・矢野龍王 他　著
B5判／456ページ
定価（本体1880円+税）
ISBN978-4-7741-7823-3

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

エディフィストラーニング株式会社　著
B5判／400ページ
定価（本体2980円+税）
ISBN978-4-7741-7807-3

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

山本三雄　著
B5判／576ページ
定価（本体1480円+税）
ISBN978-4-7741-7808-0

イエローテールコンピュータ　著
A5判／416ページ
定価（本体1880円+税）
ISBN978-4-7741-7852-3

角谷一成・イエローテールコンピュータ　著
A5判／432ページ
定価（本体1880円+税）
ISBN978-4-7741-7853-0

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

Software Design plus

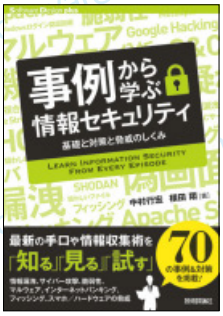
最新刊！

Software Design plusシリーズは、OSとネットワーク、IT環境を支えるエンジニアの総合誌『Software Design』編集部が自信を持ってお届けする書籍シリーズです。

斎藤祐一郎 著
A5判・160ページ
定価 2,280円（本体）+税
ISBN 978-4-7741-7865-3

養成読本編集部 編
B5判・192ページ
定価2,480円（本体）+税
ISBN 978-4-7741-7858-5

神原健一 著
B5変形判・192ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-7749-6



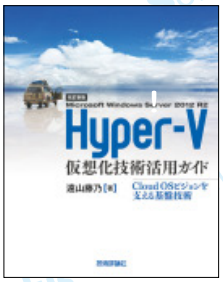
中村行宏、横田翔 著
A5判・320ページ
定価 2,480円（本体）+税
ISBN 978-4-7741-7114-2

川本安武 著
A5判・400ページ
定価 2,980円（本体）+税
ISBN 978-4-7741-6807-4

勝俣智成、佐伯昌樹、原田登志 著
A5判・288ページ
定価 3,300円（本体）+税
ISBN 978-4-7741-6709-1

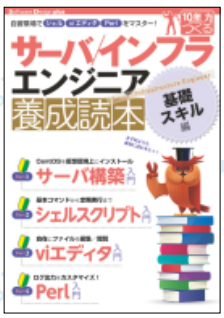
中島雅弘、富永浩之、國信真吾、花川直己 著
B5変形判・416ページ
定価 2,980円（本体）+税
ISBN 978-4-7741-7369-6

倉田晃次、澤井健、幸坂大輔 著
B5変形判・520ページ
定価 3,700円（本体）+税
ISBN 978-4-7741-6984-2



遠山勝乃 著
B5変形判・392ページ
定価 3,500円（本体）+税
ISBN 978-4-7741-6571-4

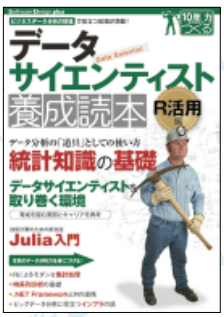
寺島広大 著
B5変形判・440ページ
定価 3,500円（本体）+税
ISBN 978-4-7741-6543-1



福田和宏、中村文則、竹本浩、木本裕紀 著
B5判・128ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7345-0

養成読本編集部 編
B5判・156ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7313-9

養成読本編集部 編
B5判・128ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7320-7



養成読本編集部 編
B5判・164ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7057-2

PHPライブラリ&サンプル実践活用[厳選100]
WINGSプロジェクト 著
定価 2,480円+税　ISBN 978-4-7741-6566-0

アドテクノロジー プロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6429-8

[改訂新版]サーバ/インフラエンジニア養成読本　管理/監視編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6424-3

[改訂新版]サーバ/インフラエンジニア養成読本　仮想化活用編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6425-0

[改訂新版]サーバ/インフラエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6422-9

iOSアプリエンジニア養成読本
髙橋俊光、諏訪悠紀、湯村翼、平屋真吾、平井祐樹 著
定価 1,980円+税　ISBN 978-4-7741-6385-7

[改訂新版]Linuxエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6377-2

Webアプリエンジニア養成読本
和田裕介、石田絢一（uzulla）、すがわらまさのり、斎藤祐一郎 著
定価 1,880円+税　ISBN 978-4-7741-6367-3

Androidライブラリ実践活用
菊田剛 著
定価 2,480円+税　ISBN 978-4-7741-6128-0

はじめての3Dプリンタ
水野操、平本知樹、神田沙織、野村毅 著
定価 2,480円+税　ISBN 978-4-7741-5973-7

PHPエンジニア養成読本
PHPエンジニア養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5971-3

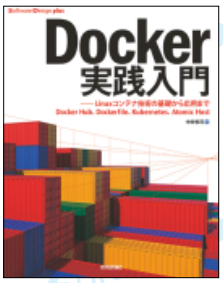
独習Linux専科
中井悦司 著
定価 2,980円+税　ISBN 978-4-7741-5937-9

データサイエンティスト養成読本
データサイエンティスト養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5896-9

Androidエンジニア養成読本Vol.2
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-5888-4

Raspberry Pi[実用]入門
Japanese Raspberry Pi Users Group 著
定価 2,380円+税　ISBN 978-4-7741-5855-6

データベースエンジニア養成読本
データベースエンジニア養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5806-8



中井悦司 著
B5変形判・200ページ
定価 2,680円(本体)+税
ISBN 978-4-7741-7654-3

養成読本編集部 編
B5判・192ページ
定価 2,280円(本体)+税
ISBN 978-4-7741-7631-4

岩永信之、山田祥寛、井上章、伊藤伸裕、熊家賢治、神原淳史 著
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7607-9

養成読本編集部 編
B5判・112ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7441-9

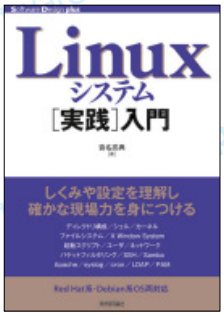
上田隆一 著
USP研究所 監修
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7344-3

森藤大地、あんちべ 著
A5判・296ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-6326-0

㈱パイプドビッツ 著
A5判・224ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-6205-8

久保田光則、アシアル㈱ 著
A5判・384ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-6211-9

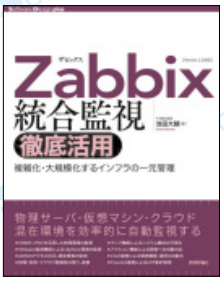
ニコラ・モドリック、安部重成 著
A5判・336ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-5991-1



沓名亮典 著
A5判・416ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-5813-6

松本直人、さくらインターネット研究所(日本Vyattaユーザー会) 著
B5変形判・320ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6553-0

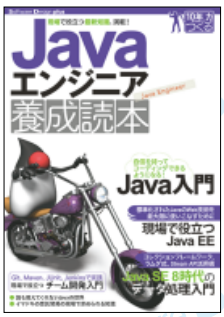
乾正知 著
B5変形判・352ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-6304-8



TIS㈱ 池田大輔 著
B5変形判・384ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6288-1

大谷純、阿部慎一朗、大須賀稔、北野太郎、鈴木教嗣、平賀一昭 著
㈱リクルートテクノロジーズ、㈱ロンウイット 監修
B5変形判・352ページ
定価 3,600円(本体)+税
ISBN 978-4-7741-6163-1

沼田哲史 著
B5変形判・360ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-6076-4



養成読本編集部 編
B5判・168ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6931-6

吾郷協、山田順久、竹馬光太郎、和智大二郎 著
B5判・136ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6797-8

養成読本編集部 編
B5判・168ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6787-9

養成読本編集部 編
B5判・164ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6983-5

養成読本編集部 編
B5判・212ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6578-3

〒162-0846 東京都新宿区市谷左内町21-13
販売促進部 TEL 03-3513-6150 FAX 03-3513-6151

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

毎月18日発売

PDF電子版
Gihyo Digital
Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読（12回）

14,880円（税込み、送料無料）1冊あたり1,240円（6%割引）

PDF電子版の購入については

Software Design ホームページ
http://gihyo.jp/magazine/SD

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

- インターネットから最新号、バックナンバーも1冊からお求めいただけます！
- 紙版のほかにデジタル版もご購入いただけます！
  デジタル版はPCのほかにiPad/iPhoneにも対応し、購入するとどちらでも追加料金を支払うことなく読むことができます。

※ご利用に際しては、／˜＼Fujisan.co.jp（http://www.fujisan.co.jp/）に記載の利用規約に準じます。

Fujisan.co.jpからのお申し込み方法

1 >> ／˜＼Fujisan.co.jpクイックアクセス
http://www.fujisan.co.jp/sd/

2 >> 定期購読受付専用ダイヤル
0120-223-223（年中無休、24時間対応）

第87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。

AGL Unified Code Base

車載機器向けLinux「AGL UCB」

自動車のIT化が進むにつれて、車載情報機器向けのソフトウェアにも大きな注目が集まるようになってきました。そんな中で、The Linux Foundationの協業プロジェクトである「Automotive Grade Linux（AGL）」は、自動車業界をターゲットとした新しいLinuxディストリビューション「AGL Unified Code Base」（以下、AGL UCB）を発表しました。AGLは、自動車関連企業と技術系企業の連携によってLinuxベースの車載情報機器向けソフトウェアスタックを開発するオープンソースプロジェクトです。自動車メーカーからは、トヨタ自動車や日産自動車、Jaguar Land Roverなど、計8社が参加しています（2016年1月現在）。

今回発表されたAGL UCBは、車載機器向けのソフトウェアの開発に使われることを前提としたディストリビューションです。AGLでは2015年に、車載情報機器特有の要件を満たすための共通的な要求仕様書を公開しており、AGL UCBはこれに準拠したものになっています。この要求仕様では、データ転送規格としてCAN（Controller Area Network）とMOST（Media Oriented Systems Transport）が、無線通信方式としてはWi-FiとBluetoothが採用されています。また、アプリケーション開発ではネイティブアプリとHTML5をサポートします。

AGL UCBの大きな特徴の1つは、同じコードベースから自動車に搭載されるさまざまなアプリケーション向けのプロファイルを作成できるように設計されているという点です。そのうえ、純粋な車載情報機器だけでなく、計器盤やヘッドアップディスプレイ、テレマティクスなどのための共通システム基盤としても利用可能な柔軟性を備えています。各種UIコンポーネントも完備されており、UIフレームワークはQtおよびQMLをサポートします。

AGLの公式サイトではAGL UCBを使ったデモアプリケーションが公開されており、ホームスクリーンやメディアブラウザ、カーナビゲーションシステムなどといった各種アプリケーションの実装例を確認することができます。

Tizen IVIやGENIVIによるノウハウも生かす

AGLでは、2014年まではTizenをベースとした車載情報機器向けOSの「Tizen IVI」をリファレンスとして採用していました。そして2014年以降はその方針を変更し、「Yocto Project」をベースに、車載向けに特化して機能強化した新しいLinuxディストリビューションの開発を目指しました。その成果物がAGL UCBです。

Yocto Projectとは、組込み機器のためのLinuxベースのカスタムシステムを構築するコラボレーションプロジェクトおよびその成果物の総称です。ビルドツールやビルド命令のメタデータ、ライブラリ、ユーティリティ、GUIツールセットなど、組込みLinuxシステムを開発するための環境が用意されており、独自のディストリビューションをハードウェアアーキテクチャに依存しない形で構築することができます。

さらに、AGL UCBの開発にあたっては車載情報機器向けのプラットフォームとしては競合関係にあたるドイツのGENIVIアライアンスとの連携も強化しており、実際にGENIVIによる成果物も取り込まれています。すなわち、AGL UCBはゼロから独自に設計されたディストリビューションではありますが、Yocto ProjectやTizen IVI、GENIVIアライアンスをはじめとする既存オープンソースプロジェクトで得たノウハウを積極的に取り入れたものになっているということです。

コネクテッドカーやスマートカーへの期待が高まる中で、車載情報機器の果たす役割はますます大きくなっています。自動車や自動車部品のメーカーは、この分野の主導権を特定のソフトウェアメーカーに委ねるつもりはないようです。AGLによる挑戦はその意思を明確にするものだと言えます。そういう意味でも、AGL UCBは自動車業界を取り巻く状況に対して大きな一石を投じる存在になりそうです。SD

Automotive Grade Linux
https://www.automotivelinux.org/

DIGITAL GADGET vol.207

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

Interaction Awardに見る、インタラクションの本質

インタラクションのコストと効果

Interaction Award（インタラクションアワード）は、インタラクション（相互の［inter］＋効力［reaction］）を活用したプロダクトを評価する、米国IxDA（Interaction Design Association）が主催するアワードです。

Interaction Award 2016 公式サイト
http://awards.ixda.org/2016-interaction-awards/

とくに最近の傾向は、製品として明確な存在や操作性をもったインタラクションだけでなく、インタラクションを重視したサービスやプロジェクト、企画そのものも評価の対象になっています。インタラクションそのものも、マイクロインタラクションと呼ばれる、操作をさらに細かい単位に分割して人の振る舞いを考えたり、ちょっとした操作や小さな動きで人の感情を喚起する手法も数多く用いられるようになってきました。

ユーザインターフェースでも、意味もなく動くアニメーションは邪魔と感じますが、動きに意味があり、必然性がある動作や操作であれば、より楽しく、親しみをもって扱うことができます。新しいデジタルデバイスが氾濫していく中で、既存のほとんどのインタラクションはいままでにあった何らかの物理的事象を模倣したものになっており、そのおかげで操作を覚えておくことが容易なものになっています。インタラクションには、適切な反応と、操作対象に対して直接的に影響を与えているような感覚が重要です。

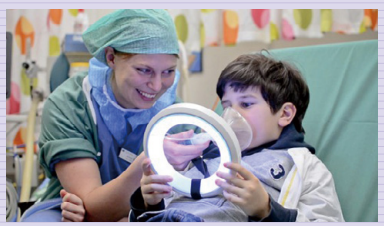
また、時間的な要素である「素早さ」「遅さ」「タイミング」「待機時間」などもポイントです。一見同じように見える操作も、タイミングの違いやタイミングの悪さによっては、操作感に雲泥の違いをもたらします。またどんなに綿密に計画し、設計したインタラクションも、実際に操作してみないとわからないことや、見当違いなどといった状況もあり、経験豊富な人にとってもなかなか安心できない要素でもあります。

一方「No UI」と呼ばれる、ユーザインターフェースなし、インタラクションなしで、目的を達成するようなインターフェースの考え方も広がってきました。操作そのものが楽しく、素早く扱えるものでないかぎり、操作しないで目的を達成できることは1つの理想形ではあります。けれど実際のところ、インタラクションの操作コストや学習コストがゼロになることはありません。考えただけで脳波でらくらく操作できるような未来になるまでは、実際の操作を始めるまでの面倒さをできるだけ少なくするのが良いと言われています。読む／予測する／理解する／探す／操作する／待つ／切り替える／確認するといった操作1つ1つの面倒さを極小化していくのです。

今年の傾向と作品の評価

Interaction Awardは、次に紹介する5つの部門に分かれ、それぞれの部門の主眼にもとづいて評価されます。複数の要素を持っており、複数の部門

*1）Kurbo



*2）ANNA breathing assistant

*3）The Color Visualizer

※本記事で紹介しているものは国内未発表・未発売のものを含んでいます。

に同時エントリーされている作品もあります。過去に本連載でも紹介した自転車方向指示器のHammerheadなど、すでに取り上げたものも数多くノミネートされています。

Connecting（つながり）部門

人と人や、人とコミュニティ間のコミュニケーションを手助けするしくみ

- **Edward M. Kennedy Institute**……教室内でのタブレットを活用した投票システム
- **SAP Tennis Analytics for Coaches**……テニスのコーチのための試合状況解析ツール
- **Trafficbridge**……紙ベースのワークフローの再構築
- **Who, Like Me, Is Threatened?**……人権運動のための装置。さまざまな種類のインタビューを聴くことができる

Disrupting（再構築）部門

当たり前となっている既存のサービスや事柄を壊して再構築し、新しい価値を生み出す

- **The New Eurosport Player**……スポーツ中継のサポートアプリ
- **reForm**……形状を記憶し、再構築する物体
- **Sensel Morph**……さまざまな用途に変化するタッチパネル（本連載204回に掲載）
- **Wayfindr**……目の不自由な人向けの旅行ガイド杖

Empowering（助力）部門

人々が限界を超え、今までできなかったような事柄をできるように仕向ける

- **Kurbo**（*1）……子供の肥満防止、カロリー管理のためのアプリ
- **Owlet**……乳幼児みまもり用デジタル靴下（本連載201回に掲載）
- **SAM**……デジタルプロトタイプ用キット
- **Wayfindr**［複数部門受賞］

Engaging（魅了）部門

日々の出来事に喜びや注目を集め、その事柄に意味をあたえる

- **ANNA**（*2）……呼吸のタイミングを補助してくれる子供向け医療機器
- **The Color Visualizer**（*3）……標本の色をビジュアライズする教育用展示機器
- **The Imagination Machine**（*4）……好きな場所に飛んでいけるデジタル地球儀と航路の表示装置
- **The Pursuit by Equinox**（*5）……ビジュアライズされたフィットネスバイクスシステム
- **SMART**（*6）……情報やデータをビジュアライズする絵画のような表現の表示装置

Expressing（表現）部門

人々の自己表現や、創造性を手助けするしくみ

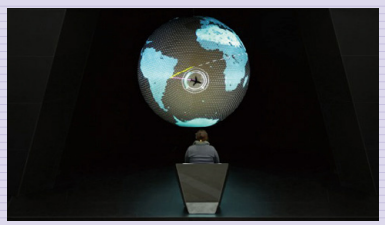
- **NailSnaps**（*7）……美しいカスタムネイルアートの作成・共有アプリ
- **Sound Blocks**（*8）……アナログシンセサイザーのような、音の成分を電子ブロックで構成するもの
- **SquareTalk**（*9）……コミュニケーションのためのデジタル灯籠
- **Step**（*10）……リアルな操作で扱えるデジタルドラムマシン
- **Trashtag**……社会的メッセージを発信するためのゴミ捨て用の文字タグ

Optimizing（最適化）部門

日々の行動をより効率的に行う方法

- **Adobe Fill & Sign**（*11）……デジタル署名用のソリューション
- **Arlanda Departure Sequencing Tool**……飛行機の発着とタクシーの待ち行列を連携させるシステム
- **Basen**……ノルウェーの軍用の宿泊管理システム
- **Kurbo**［複数部門受賞］

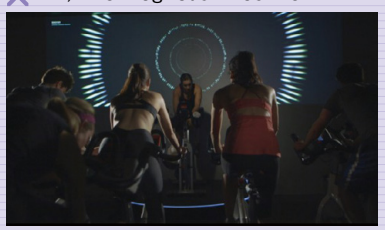
右ページのGadgetコーナーでは、ノミネートされた作品の中から、惜しくも受賞は逃したものの興味深い作品を紹介します。



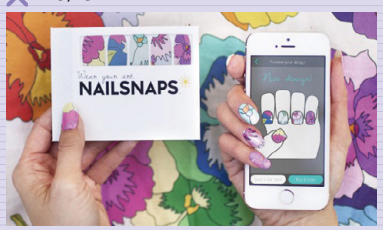
*4）The Imagination Machine

*6）SMART

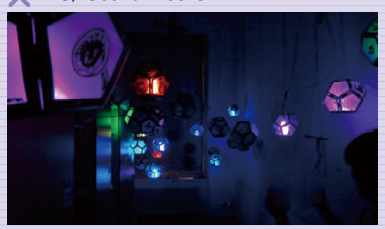
*8）Sound Blocks



*5）The Pursuit by Equinox



*7）NailSnaps



*9）SquareTalk

これからのインタラクション

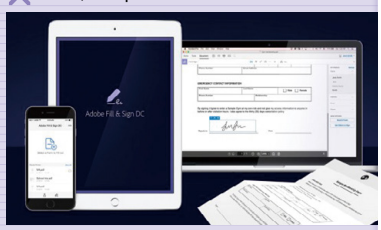
インタラクションの観点としては、次のような要素に配慮すると良いでしょう。

- 操作するときのペース配分
- 反応速度（瞬間、瞬時、そのとき、といったいくつかの異なるタイムスケール）
- 利用するときの状況（立っている、座っている、寝ている、走っている、急いでいるなど）
- コンテンツやデータが変化したときの見せ方、操作の仕方の変化
- 使い続ける場合への配慮。しばらく使わなかったときの配慮

人の動きや動作には個人差があり、インタラクションのための操作にも、慣れや身体的特徴も含めた個人差が存在します。そういった環境下において標準化やパーソナライゼーションが重要な要素になってくるとともに、個人差とは関係ない普遍的な要素も存在します。

さらに最近では、ユーザインターフェースや操作の微調整そのものを機械学習によって最適化しようという考えもあり、最適化のアプローチと一般的ジェスチャー操作の標準化が進むことによって、さらに便利で使いやすいものになっていくことでしょう。SD

*10）Step



*11）Adobe Fill & Sign

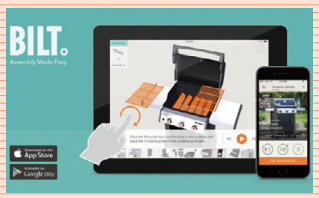
Gadget 1

BILT

https://www.biltapp.com/

デジタル組立図

BILTは複雑な組み立て作業が必要な製品の、組み立て方を指南するしくみです。たとえばある屋外用バーベキューコンロであれば、2人がかりで約1時間、40工程で組み立てる様子を、ステップを追って確認できます。また必要な道具も的確に明示されます。たいていは紙に書かれてわかりにくい手順も、BILTアプリでチェックすることで、平易に間違いなく組み立てることができます。単なる動画マニュアルとも異なり、個々人のペースで進められるところが便利な点です。組み立て用のコンテンツも、製品を提供する企業が作るだけでなく、一般の人による解説の可能性も考えられています。



Gadget 2

Camera Restricta

http://philippschmitt.com/

ソーシャル撮影カメラ

Camera Restricta（制限するカメラ）は、常にネットワークに接続されたカメラで、絶景が見える場所や写真撮影ポイントのような、多くの人が撮影済みのポイントを位置情報とともに教えてくれます。ただし、ネット上の写真共有サイトにある写真の位置情報と自動的に比較し、皆が撮影しているような場所で写真を撮ろうとすると、シャッターが降りなくなるのです。意図的に皆とは異なる写真を撮らなければならない、ネット時代の不思議なカメラです。カメラ的デバイスの中身は単なるスマートフォンですが、このカメラで撮影すると、必ずと言っていいほど誰もまだ撮影したことのない風景が撮影できるわけです。

Gadget 3

Mr. Piggy

http://awards.ixda.org/entry/2016/mr-piggy-your-pet-piggy-bank/

デジタル貯金箱

Mr. Piggyが提唱するのは、ネットワーク化されたデジタル貯金箱です。声をかけると耳を動かし、口の部分にはクレジットカードのリーダーが搭載されており、利用額や口座残額に応じて豚の体の大きさが変化するようになっています。残念ながらまだコンセプトモデルでしかありませんが、既存の技術で可能なサービスであり、お金や硬貨、貯蓄といった古くからある金銭感覚を、ネット時代に融合しているところがポイントです。電子マネーや仮想通貨が話題になるなかで、単純に新しいことに飛びつくのではなく、既存のしくみや既存のサービスを、うまくデジタルデバイスと連携させた好例です。

Gadget 4

Memo -tangible appointment

http://juliahunold.com/MEMO-tangible-appointments

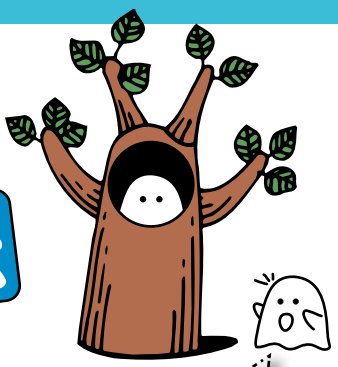
デジタルかつアナログなスケジュールボード

Memoはおもに認知障害の人向けに考えられた、アナログかつデジタルなスケジュールボードです。カレンダーに書き込んだり、冷蔵庫に貼り付けたメモ情報は、その場にはありますが、そのままの状態では平易に共有できません。また、メモや予定そのものを忘れてしまうこともあります。このプロジェクトでは、一目見ただけでわかるアナログ的なカレンダーと、そこに貼り付けた丸いオブジェクトの情報がデジタル化され、スマートフォンやネットと適切に連携して利用することができます。丸いオブジェクトは単体でも機能し、Arduinoを用いて試作品が作られています。

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

34

結城浩の再発見の発想法



Timestamp

Timestamp ——タイムスタンプ

タイムスタンプとは

データの**タイムスタンプ**(Timestamp)とは、**データが作成・修正・参照されたときの日時**のことです。たとえば、ファイルには自動的にタイムスタンプが付きます。普段はあまり意識しませんが、ファイルのタイムスタンプは3種類あり、ファイルの作成・修正・参照の日時がそれぞれ個別に管理されています。たとえば、この記事の原稿ファイルを`ls -lU`コマンドで調べると、`1 11 10:36:43 2016`と表示されました。これは、ファイルの作成日時が2016年1月11日10時36分43秒であることを表しています。`ls -lT`コマンドを使うと最後の修正日時が表示されます。`ls -lu`コマンドを使うと最後の参照日時が表示されます。デフォルトの`ls -l`で表示されるのは最後の修正日時ですが、秒は省略されます。

ファイルが作成されると、そのときの日時が作成日時として記録されます。そしてそれ以降、通常のファイル操作では作成日時は変わらず、修正日時と参照日時のみが更新されることになります。時間は、未来に向かって一方向に進む**単調性**を持っていますから、ファイルの作成日時もまた単調性を持つことになります。

順序とmake

タイムスタンプはファイルを管理するときに非常に重要な情報です。その理由の1つはファイルに**順序**が付けられるためです。ファイルを修正すると修正日時が更新されますから、結果として、最近作業したファイルはどれなのかわかることになります。自分のファイル一覧をいつも修正時刻順で表示している人は、野口悠紀雄氏の「超」整理法と呼ばれる書類管理法(書類を使った順番に並べておく方法)を自然にやっていることになりますね。

プログラムを開発するときに使うmakeなどのビルドツールでは、ファイルのタイムスタンプを利用します。makeは、ソースファイルをコンパイルしてオブジェクトファイルを作るとき、両者のタイムスタンプを比較します。そして、ソースファイルのほうが新しいとき(つまり前回のコンパイル後にソースファイルに修正が加わったとき)にのみコンパイルを実行します。これは、タイムスタンプを利用して不要なコンパイルを省略していることになります。

ユニークなURL

私は自分個人の小さなブログサイトを作るのが好きですが、そのときのURLには、よくタイムスタンプを使います。たとえば、`http://snap.textfile.org/20150906233506/`のようにします。この数字列`20150906233506`は、記事を作成した日時である2015年09月06日23時35分06秒そのものを表しています。

このようにURLにタイムスタンプを使うと、コンテンツが変化しても変化することのない、

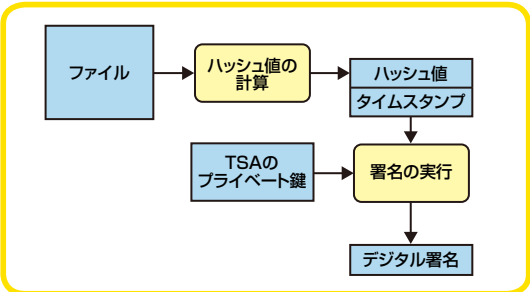
いわゆる「クールなURI」が簡単に実現できます。記事を最初に作成した日時は不変だからです。クールなURIについては、“Cool URIs don't change”[注1](邦訳「クールなURIは変わらない」[注2])を参照してください。

タイムスタンピング

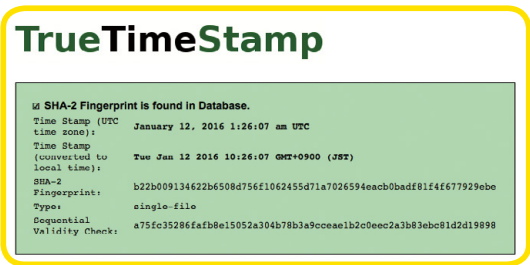
ファイルの作成日時を表すタイムスタンプは、通常のファイル操作では変更を受けませんが、絶対に変更できないわけではありません。ですから「このファイルは確かにこの日時に作成した」ということを証明することはできません。

ある日時にファイル(データ)の存在を保証するためのサービスをタイムスタンピングといい、そのサービスを実現するプロトコルとしてTime-Stamp Protocol(TSP)が定められています(RFC 3161)。TSPでは、タイムスタンピングを実行するTime Stamping Authority(TSA)という機関との通信手順を定めています。TSAは、ファイルのハッシュ値にタイムスタンプを連結し、そのデジタル署名を作成します(**図1**)。

▼図1　TSAのタイムスタンピング



▼図2　TrueTimeStamp.org



このデジタル署名によって「この日時にこのファイルが存在したこと」を検証するのです。

たとえば、“Hello!\n”という7文字からなるデータが2016年01月12日01時26分07秒に存在していたことはTrueTimeStamp[注3]というフリーのサービスで検証できます。実はこのデータは筆者がこの原稿を書いているときに登録したものです(**図2**)。

一般に、タイムスタンプはセキュリティで重要な役割を担います。SSL/TLSで通信を行うときに利用する証明書には、**有効期限**が付けられています。私たちが安心してセキュアな通信を行うためには、有効な証明書は欠かせません。ですから、コンピュータの時計が狂っている場合には、必ず警告が表示されるようになっています。たかが日時と侮ることはできません。

日常生活とタイムスタンプ

私たちはスーパーで買い物をするとき、新鮮な食品を購入するために**賞味期限**というタイムスタンプを読みます。賞味期限を食品に印刷しているのはその食品を加工している人です。つまり私たちは、そのタイムスタンプを信用することで、タイムスタンピングを行う人を信用していることになりますね。

また、大学受験の募集要項などに「消印有効」と書かれていることがあります。郵便物に押される**消印**をタイムスタンプにして、出願の書類を受け付けるかどうかの判断が行われます。大学側は、郵便局をタイムスタンピングを行うサービスとして信用していることになります。

◆　◆　◆

あなたの周りを見回すと、たくさんのタイムスタンプが見つかるでしょう。そのタイムスタンプは不変でしょうか。それとも、修正があるたびに更新されるでしょうか。そのタイムスタンプを生成しているサービスは誰でしょうか。ぜひ、考えてみてください。SD

注1) URL http://www.w3.org/Provider/Style/URI.html
注2) URL http://www.kanzaki.com/docs/Style/URI

注3) URL http://truetimestamp.org

増井ラボノート

コロンブス日和

第5回　Gyaki

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

Author　増井 俊之（ますい としゆき）　慶應義塾大学

お絵描きシステムを忘れていませんか？

今回は「Gyaki」というお絵描きシステムを紹介します。

計算機やWebがこれだけ普及しているのに、お絵描きシステムが世の中であまり使われていないのが不思議です。iPad Pro、Surface Pro、Surface Bookのような魅力的なペン計算機が最近たくさん登場しているのにもかかわらず、ペンを使ったお絵描きシステムは広く活用されていませんし、普通のユーザがお絵描きするシチュエーションも、あまり想定されていないように見えるのは気のせいでしょうか。スマホが流行る以前は、PalmなどのPDA（Personal Digital Assistant：携帯情報端末）でメモを描く人も多かった気がするのですが、スマホやタブレットでお絵描きする人は以前より減っているような気がしています。

文章を使うよりも絵で説明するほうがわかりやすいことは多いでしょうし、デザインをスケッチしたいことも多いでしょうし、考えをまとめるために図を描きたいこともあるでしょうし、ペンで絵を描きたい機会は多いはずです。実際、紙のノートを活用している人は多いと思われますが、現在のパソコンはマウスやキーボードを使って、テキストや図を編集するのに使うのが普通だと思われており、ペンを使って知的生産活動を行うことはとくに推奨されていません。

Palmのようなペン型PDAが広く使われていたときは、ペンによるお絵描きもそれなりに利用されていたと思うのですが、スマホが普及してペンが駆逐されたために、手書きでメモする機会が減ってしまったように思われます。

しかし、

- 性能が良いペンコンピュータが普及しつつある
- あらゆるブラウザでお絵描き機能が提供されている

という状況の現在、新しいWeb時代のお絵描きシステムが、もっと使われるべきだと思います。

Gyaki――ブラウザ上の手軽なお絵描きシステム

お絵描きアプリやWebサービスはたくさんありますが、達人のための機能を持つものが多く、とくにお絵描きが得意でない普通のユーザが手軽に使うためのものは多くありません。アイデアなどをメモしたくなったとき、最小の手間でお絵描きを開始して安全に保存できるようなお絵描きシステムを簡単に使えるようにしておきたいものです。

パソコンやタブレットで文章を書くときはエディタを起動するのが普通であるのと同様に、お絵描きするときはアプリを起動したりWebページを開いたりする必要があります。しかし絵を描く方法やセーブ方法を簡単化することによって、お絵描きのハードルを下げることはできるでしょう。私は、最小限の機能をもち手軽に利用できる「Gyaki」（楽ギャキ）というシステ

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

ムを作って使っています。

Gyakiでは、とにかく手軽にアイデアをスケッチすることを目標としているので次のような機能だけを用意しています。

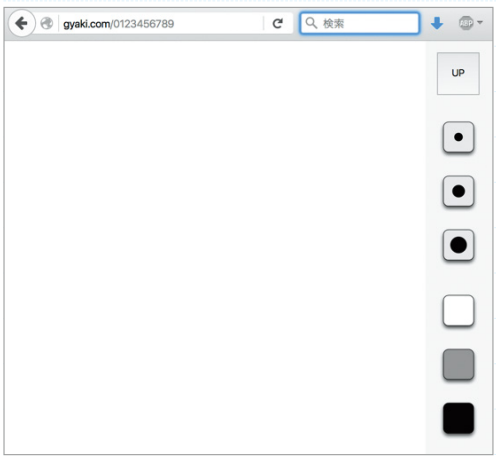
・手書きで線を描く
・3種類の太さを選択
・3種類の色(黒／灰色／白)を選択

一度描いたものを移動することはできませんし、undo機能すらありませんが、紙のスケッチでも同様ですし、たいていの場合においてそれでなんとかなっているわけですから、無理にたくさんの機能を用意する必要はないと割り切っています。

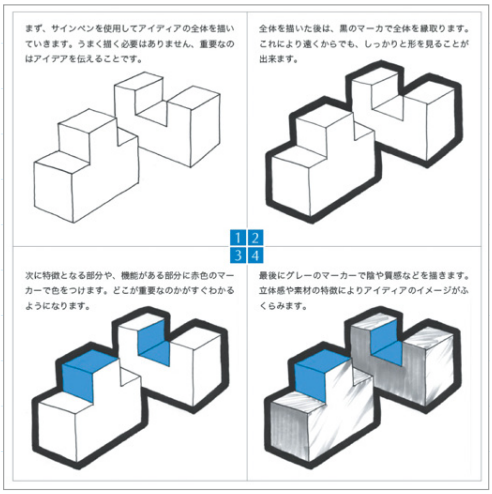
これらの機能だけを用意した理由は、情報科学芸術大学院大学(IAMAS)の小林 茂氏の著書『Prototyping Lab』(オライリー・ジャパン、2010年)で紹介されている「アイデアスケッチ」をスマホやタブレットで簡単に使いたいと考えたからです(**図1**)。アイデアスケッチは小林氏の同僚であるJames Gibson氏が考案したもので、学内外のワークショップなどを通じて小林氏が普及に努めているものです。

絵が下手な人間でもこの方法でスケッチを書けば、割とまともな感じに見えるという大きな利点があります。

▼**図2** Gyakiの初期画面



▼**図1** Prototyping Lab——「作りながら考える」ためのArduino実践レシピ(小林茂著、オライリー・ジャパン、2010年)より[注2] 作図：蛭田直



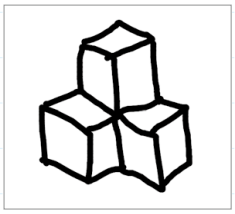
Gyakiの使用例

http://Gyaki.com/[適当な文字列]というURLにアクセスすることでGyakiのお絵描きを始められます(**図2**)。

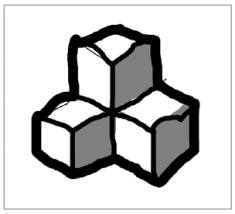
デフォルトのペンを使って、私が箱の絵を描くとこのようなサエない絵になってしまいますが(**図3**)、アイデアスケッチで書けばこのようになり(**図4**)、かなりカッコ良くなることがわかります。

普通の紙でアイデアスケッチを利用する場合は、灰色のコピックや薄墨の筆ペンなどが必要になりますが、Gyakiだともっと気軽にアイデアスケッチを使うことができます。

▼**図3** デフォルトではこんな具合



▼**図4** アイデアスケッチ導入!



注2) https://www.oreilly.co.jp/books/9784873114538/

前述のGyakiのURLの「適当な文字列」のところに、Gyazoのユーザ ID(Macの場合~/Library/Gyazo/idに書かれている文字列)を指定すると、アップロードボタンを押したとき、お絵描き結果が自分のGyazoアカウントに追加されます。URLが多少長くなってしまいますが、本誌12月号で紹介したGyumpを使って、短いURLを使用すれば便利です。描いた絵の一部分だけが必要な場合は、Gyazoで選択してアップロードすればよいでしょう。いずれの場合でも、何かを描いてからWeb上にセーブするための手間はかなり少なくなっていますし、Gyazo.com上で後から検索するのも簡単です。

Gyakiの実装

昔のブラウザ上でお絵書きするためには、FlashやJavaなどのプラグインを使う必要がありましたが、最近のブラウザのJavaScriptではいろいろな方法で何でも描けるようになりました。

- Canvas機能を使う
- SVG機能を使う
- WebGLを使う

3次元描画を行うにはWebGLが適しており、最近人気の情報視覚化システム「D3.js」ではSVGが利用されているなど、用途によって適した描画システムは異なりますが、単純なお絵書きにはCanvas機能を使うのが一番楽です。Gyaki.comではCanvas機能を使ったお絵描きをサポートしています[注3]。

上手なお絵描き方法の考察

白い紙の上にいきなり絵を描くのは難しいものです。Gyaki＋アイデアスケッチは、考え方をすぐに視覚化するのには適しているのですが、きれいな絵を描こうとする場合や奇抜な絵を描くのには向いていません。

トレース

お絵描きが得意でない人にいきなり自転車を描かせてみると、まともに書けないことが多いようです(**図5**)。

しかし「自転車」でWeb画像検索すれば、さまざまな自転車の写真が見つかりますから、それをなぞったりコピー&ペーストしたりすれば、苦労せずに正しい自転車の絵を書くことができます。最近は多くのものが画像検索で見つかるので、現実世界に存在するものを描く必要性は少なくなっているかもしれません。

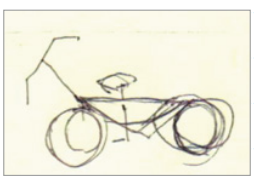
補助線の利用

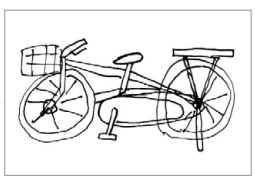
整った表や図を描きたいときは方眼紙を使うと便利ですが、背景として枡目や集中線を利用すると、絵を描きやすくなることがあります。たとえば集中線を背景にして先ほどと同じ絵を描くと(**図6**)、より正しいパースで絵を描くことができます。

枯尾花システム

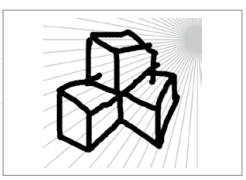
現実世界に存在しない物体を描きたい場合や、

▼図5　いろんな人に自転車を描いてもらった例





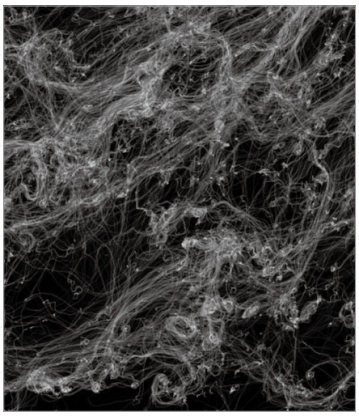
▼図6　集中線を使って描画してみる



注3) Gyakiのコードを公開しています(https://github.com/masui/Gyaki)。

▼写真1　パレイドリアの例[注4]

▼図7　踊っている人が見える？



▼図8　トレース結果

新しく形状をデザインしたいような場合は画像検索ができません。しかし、そのような場合でも想像力を喚起するお絵描き支援システムを作ることはできるかもしれません。

雲の中に動物が見えたり、自動車の前面が人の顔に見えることがよくありますが、このような現象はパレイドリア(Pareidolia)と呼ばれています(**写真1**)。「幽霊の正体見たり枯尾花(かれおばな)」という諺(ことわざ)がありますが、ススキのパターンが人間の想像力を喚起することがあり得るということなのでしょう。

ランダムな曲線群や直線群の中に関係ない絵が見える現象は、レオナルド・ダ・ヴィンチが指摘していたほどであり[注5]、『吾輩は猫である[注6]』の中では迷亭先生が次のように語っています。

> *レオナルド・ダ・ヴィンチは門下生に寺院の壁のしみを写せと教えた事があるそうだ。なるほど雪隠などに這入って雨の漏る壁を余念なく眺めていると、なかなかうまい模様画が自然に出来ているぜ。君注意して写生して見給えきっと面白いものが出来るから。*

枯尾花的パターンを自動生成すると、自分が描きたいものをその中に発見できるかもしれません。たとえば、先進的UIデザイナの深津貴之(fladdict)さんが開発した「Jackson Pollock的スケッチシステム[注7]」を使うと、絵心がない人でも躍動的な絵を描ける可能性があります(**図7**、**図8**)。

枯尾花お絵描きに向いたパターンというのは確かに存在するようですし、描きたいものの分野によって有効なパターンも異なると思われます。

Deep Learningで学習した画像認識システムを利用したDeep Dreamは人間の枯尾花認識機能を計算機でシミュレートしたものだと言えるかもしれません(**図9**)。こういう技法を組み合わせることによって、もっと気軽にお絵描きできるシステムを作りたいと思っています。SD

▼図9　Deep Dream(http://deepdreamgenerator.com/)

注4) https://ja.wikipedia.org/wiki/パレイドリア
注5) http://www.goodreads.com/quotes/978797-look-at-walls-splashed-with-a-number-of-stains-or
注6) 青空文庫(http://www.aozora.gr.jp/cards/000148/files/789_14547.html)
注7) fladdict(http://fladdict.net/blog/2016/01/jackson-pollock.html)

新連載

宮原徹のオープンソース放浪記

第1回 全国で開催されるOSCとOSunC

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

OSCは全国で開催中

皆さん、こんにちは。宮原徹です。今月号から毎月、私がオープンソースカンファレンス（OSC）で全国を飛び回っている様子を中心に、オープンソースに関連したトピックやIT業界の動向、エンジニア論などをお話していきたいと思います。

OSCとは、日本全国で開催されている「オープンソースの文化祭」という感じのイベントです。SD読者の中にも、参加したことがあるという方は実は多いのではないかと思っています。

2004年9月に第1回を東京で開催したあと、北は北海道から、南は沖縄まで、11年で120回以上開催されてきています。だいたい1ヵ月に1回開催されていることになりますが、ここ最近は、多いと月2回開催されています。これまで開催した地域を日本地図で見てみると、**図1**のようになります。

OSCの変形版、OSunCも開催

また、OSCの変形版である「オープンソースアンカファレンス（OSunC）」も以下の地域で開催されています。

・川越（埼玉県）
・鹿児島

OSunCは、事前に発表者を決めず、当日参加者から発表者を募る形式です。懇親会を兼ねて食べ物、飲み物を楽しみつつ発表を聞く、というスタイルで開催されるので、和やかな雰囲気で開催されているのも特徴でしょう。開催日時とプロジェクターの使える場所を決めるだけでよいので、今後はOSCが開催されていない地域でもOSunCを開催していきたいと考えています。

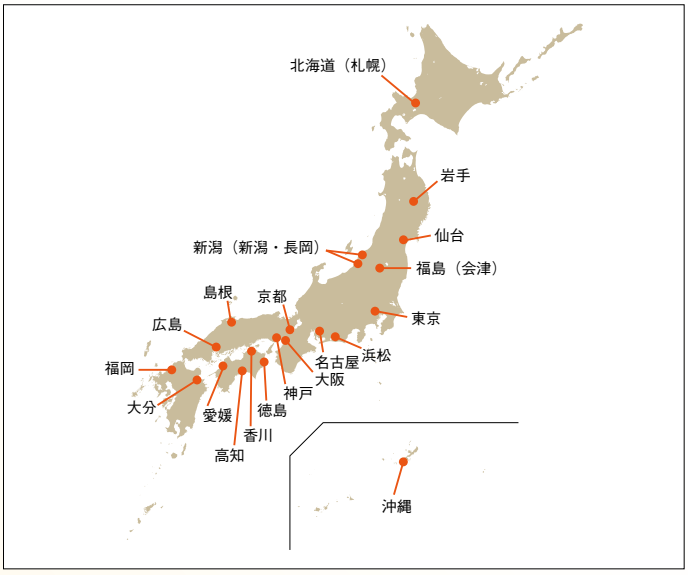
なぜOSCを開催するのか

OSCは学校行事の「文化祭」をイメージしてスタートしました。第1回の開催は日本電子専門学校の校舎をお借りしていましたし、それ以後もできるだけ大学、専門学校をお借りして開催するようにしています。

OSCではセミナーとブース展示の2つが柱になっていますが、とくに力を入れているのがブース展示です。OSSコミュニティ、協賛企業のみなさんが思い思いにブース（150〜180センチ幅の机）でデモやパネルなどを展示します。小規模な開催でも20、東京や京都などの大規模開催では100以上のブースが並びます。その様子を「コミケっぽい」と評する人もいます。ブース展示は、文化祭でいえばクラスや部活、サークルなどの発表に相当すると考えてもらうとよいでしょう。

ブース展示では、来場者と出展者

▼図1 OSCの開催地



の間で直接顔を見ながらの双方向コミュニケーションが発生しますから、双方にとって満足が得やすいというメリットがあります。また、効果的な展示を行うために出展者はパネルや看板、テーブルクロスを作ったり、ノベルティグッズを作って配ったりとさまざまな工夫をする必要が出てくるので、出展を重ねるごとにブースの展示内容が進化していくのもおもしろいところです。

このように、OSCはセミナーなどによる来場者に対しての情報発信だけでなく、どのようにしたら自分たちのOSSのよさが伝わるかを考えて、そして実践する、出展者にとっての「学びの場」でもあるというのが私の考えです。もともと、私が日本オラクルという会社で製品マーケティングにかかわってきた経験やノウハウをOSSコミュニティ全体に還元できないか？というのがOSC開催を始めたきっかけでもあります。10年以上経って、学びの場としてのOSCは徐々に機能しつつあるように感じています。

OSCはみんなで作るもの

OSCは、通常の商業イベントと異なり、私の会社であるびぎねっとが事務局となって準備などの作業を行っていますが、事前の企画や当日の運営などには各開催地域の人々、とくに学生スタッフのみなさんの協力に負うところが多いのも特徴です。

スタッフの役割には、受付などの人と接する作業もあれば、セミナーやブース展示などが円滑に進むよう準備や運営を行う作業などさまざまなものがあります。また、手が空いている時間にはセミナーやブース展示を見て廻ることもできますし、いろいろな経験ができるという意味でスタッフ参加は実はお得だったりします。各開催では常にスタッフを募集していますので、興味のある方はぜひ応募してください。

懇親会、そして家に帰るまでがOSCです

OSCは全国各地で開催されていることもあり、ちょっとした旅気分で各地のOSCに出展する人たちがたくさんいます。そのときの楽しみとして、地元の人たちとの交流、美味しいものを食べたり飲んだり、そしてちょっと観光をして帰る、なんていうこともあります。とくに前夜祭や終了後の懇親会への参加は知り合いを増やすチャンスでもありますので、OSCに参加するときは可能な限り懇親会まで参加してほしいと思っています。

OSCはとくに学生のみなさんを優遇するようにしているので、学生は懇親会参加費が安くなっています。社会人の参加者にアレコレと聞いてみてください(ときには就職活動に有利だったりもするようです)。

全国各地のOSCの楽しさをお伝えしていきます

次回からは、全国各地で開催されるOSCの様子をみなさんにお届けして、その楽しさを少しでもお伝えできればと思っています。ですが、OSCの本当の楽しさは参加してもらって初めて伝わるかと思っています。2016年は夏ごろまでの開催日程が決まっていますので、近くで開催の際にはぜひご参加ください。またWebサイト(http://ospn.jp)でも随時情報を発信していますので、確認してみてください。SD

2016年の開催スケジュール(確定分)
- 1月23日 浜名湖(浜松)
- 1月29日 大阪
- 2月26・27日 東京
- 4月24日 OSunC川越
- 5月28日 名古屋
- 6月17・18日 北海道
- 7月2日 沖縄

Report

全国各地の特色を活かした懇親会

「OSCは懇親会からが本番」というぐらい懇親会には力を入れています。過去の懇親会で印象深いものを紹介します。

北海道といえばジンギスカン。有名なサッポロビール園で圧巻の150名以上の参加者を集めて開催されました。ひたすらジンギスカンを食べ、ビールを飲む、北海道らしい懇親会です。沖縄では会場の目の前にあるビーチでビーチパーティーです。海に沈む夕陽を見ながらのBBQで沖縄の夏を満喫しました(写真)。

今後も、全国各地の特色のある懇親会を紹介していきます。

ツボイのなんでもネットにつなげちまえ道場

第九回 アナログ入力してみる

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

はじめに

これまで、デジタルな入出力をしてきましたが、今回はアナログ入力をしてみることにしましょう。「デジタル」と「アナログ」と突然書きましたが、マイコンボードの入出力には、それぞれアナログとデジタルの2種類があります。アナログは、**図1**の左の文字のように、特定のマスに収まりきらない情報を指します。デジタルは、**図1**の右の文字のように特定の大きさのマスに収められていて、1、2といった具合に数えられる量を指します。

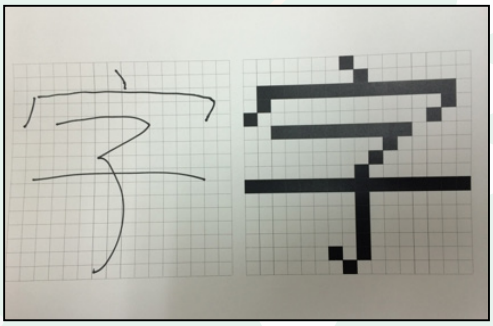
たとえば、私たちは気温を24℃といった具合に表現しますが、実際のところ気温は整数値とは限らず24.4857365836533……℃といった具合に厳密に測ればずっと細かく測ることができるでしょう。しかし、気温を小数点以下100桁まで測っても特段のメリットはないため、私たちは整数（24℃）とか、体温でしたら小数点以下一桁（36.5℃）といった実用的な区切りで、情報を簡潔に表して使っています。アナログ信号には段階というものがありませんが、デジタル信号には、段階があります。

音を表すにも、昔はレコード（円盤に掘った溝の深さ）ですとか、磁気テープ（テープの磁気の強さ）といった具合にアナログな方法で記録をしていました。今では、CDやMP3といった具合に、音を一定の細かさの数字で表してデジタル記録をしています。こういう変化する信号をデジタルにするには、アナログ信号を定期的にサンプリングします（**図2**）。たとえば、CDのサンプリング周波数は44.1kHzです。つまり、アナログ信号である音を、1秒間に44,100回サンプリングして、デジタル信号にしています。アナログ信号は連続していますが、デジタル信号は断続的な信号です。ちなみに、CDのサンプルビット数は16bitですので、音を2^{16} = 65,536段階でサンプルしています。

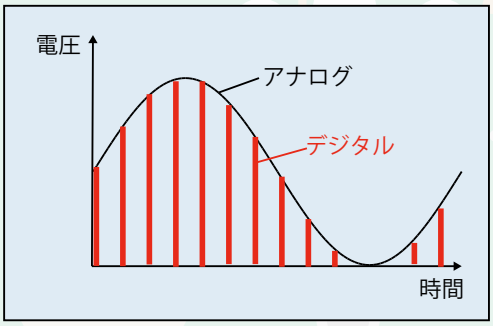
AnalogIn

この、アナログからデジタルに変換する回路が、ADコンバータ（アナログ→デジタル変換回路）です。先ほどのCDの例にも挙げたように、ADコンバータには、サンプリングを行える速

▼図1　アナログとデジタル



▼図2　アナログとデジタル



度と分解能があります。mbed LPC1768のマイコン、LPC1768に搭載されているADコンバータは、サンプリングを最大秒間20万回、分解能12bitで行うことができます。mbedの開発環境では、AnalogInを使って、このADコンバータで得た値を0.0～1.0のfloatか、unsigned short(16bit値)で得ることができます。先ほど述べたように、LPC1768で得られる値は12bitですので、mbedの開発環境でuint16_tで値を返すときには、4bitシフトが行われています。

実際に、AnalogInを使ってみましょう。例によって、mbedアプリケーションボードを使って実験してみます。このサンプルコード(**リスト1**)は、アプリケーションボードについている半固定抵抗によって分圧された電圧をAnalogInで読み取り、液晶に表示します。アプリケーションボードには半固定抵抗が2つ搭載されているので、それぞれの値を表示します。

このサンプルコードでは、AnalogInで読み取った値を表示するための液晶を制御するため、C12832というライブラリ[注1]を使用しています。ライブラリを組み込んだ状態でサンプルコードを公開[注2]していますので、こちらをインポートしていただくとすばやく実験できます。

半固定抵抗

半固定抵抗というのは、**写真1**のようなピンが3つある抵抗器です。回路記号は、**図3**の左のように抵抗器に接点が1つ追加されたようなものです。中身は**図3**の中央のように、①と③の間にカーボンなどの抵抗材があり、その上を移動する②の接点があります。抵抗材は電気が流れにくい材料で、抵抗材が長くなると両端の間の抵抗値が増えます。ですので、長さの変わらない①と③の間の抵抗値は一定ですが、①と②、②と③の間の抵抗値は②の接点の移動に応じて変わります。つまり、半固定抵抗は**図3**の右のように読み替えることができます。

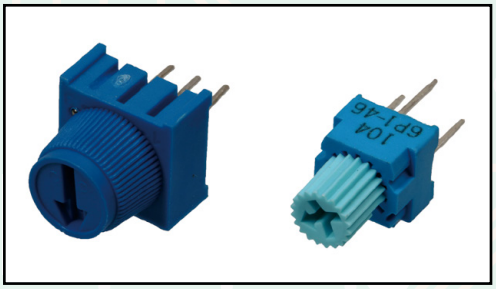
ここで、①を電源に、③をGNDに接続すると、②の電圧は②の接点の位置に応じて変化します。これは「分圧」と呼ばれます(**図4**)。電源はVinで、VinとVout、VoutとGNDの電位差(電圧)を足すと、VinとGNDの間の電位差と等しくなります。これは「キルヒホッフの第2法

▼リスト1　AnalogInのサンプルコード

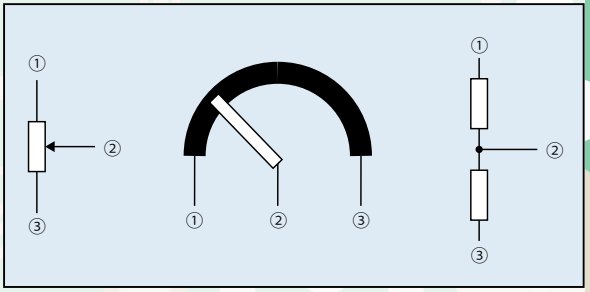
```
#include "mbed.h"
#include "C12832.h"

C12832 lcd(p5, p7, p6, p8, p11);

AnalogIn pot1 (p19);
AnalogIn pot2 (p20);

int main()
{
    while(1) {
        lcd.cls();
        lcd.locate(0,3);
        lcd.printf("Pot 1 = %.2f", (float)pot1);
        lcd.locate(0,14);
        lcd.printf("Pot 2 = %.2f", (float)pot2);
        wait(0.1);
    }
}
```

注1)　https://developer.mbed.org/users/chris/code/C12832/
注2)　https://developer.mbed.org/users/ytsuboi/code/app-board-pot/

▼写真1　半固定抵抗



▼図3　半固定抵抗のイメージ



則[注3]」と呼ばれるもので、高校の物理でも習います[注4]。

Voutの電位は、Vinと、R1とR2の抵抗値によって決まります。VinはR1とR2にかかる電圧、VoutはR2にかかる電圧です。電圧は抵抗値に比例するので(オームの法則)、Vin：Vout＝(R1+R2)：R2となります。つまり、Vin÷Vout＝(R1+R2)÷R2で、この式を変型すると、Vout＝Vin×R2÷(R1+R2)になります。たとえば、半固定抵抗のつまみを中間、つまり②の接点を①や③から等距離にすると、R1とR2は、1：1になります。この状態で先ほどの計算式に数値を入れると、Vout＝Vin×1/2です。こうして、半固定抵抗のつまみを回すことで②の電位は変化し、その値はAnalogInで読むことが可能になります。

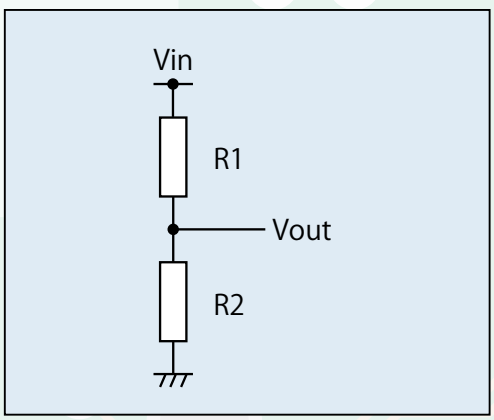
半固定抵抗と似た電子部品に、ボリュームがあります(**写真2**)。オーディオ機器に付いていて、つまみを回して音量を変えたりするために使われます。半固定抵抗は、「半固定」という名のとおり、ボリュームのように頻繁に操作されることを前提に設計されていません。このボリュームや半固定抵抗器を合わせて、可変抵抗と呼びます。英語では、可変抵抗器をポテンショメータ(Potentiometer)と呼びます。このため、先ほどのサンプルコードでは、半固定抵抗をpot1やpot2と名付けています。

なお、ボリュームには、Aカーブと Bカーブという2種類の製品があります(**図5**)。これは、つまみの回転角と抵抗値の関係を表していて、正比例するものがBカーブと呼ばれます。Aカーブのほうは、最初は回す量に対して抵抗値の増加が鈍く、後半は早く増えていきます。Aカーブは、人間の感覚に近い、対数変化[注5]をします。

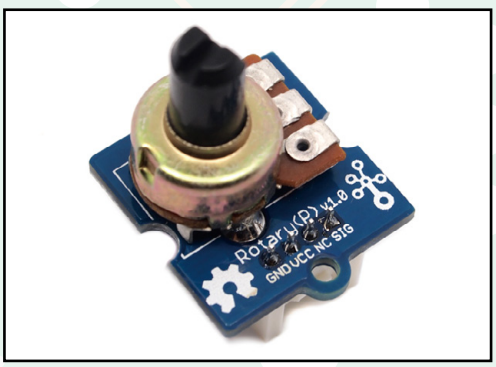
注3) **キルヒホッフの第1法則**：電気回路の任意の分岐点について、そこに流れ込む電流の和は、そこから流れ出る電流の和に等しい。**キルヒホッフの第2法則**：電気回路の任意の一回りの閉じた経路について、電位差の和は0である。

注4) 電位差や電圧については、この連載の第1回を読み返してみてください。

▼図4　抵抗分圧回路



▼写真2　ボリューム

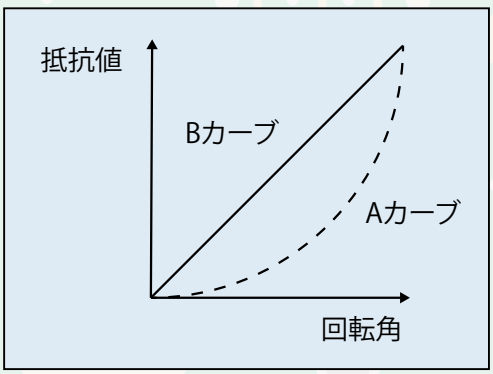


曲げセンサ

先ほど紹介した可変抵抗とは異なりますが、曲げ具合によって抵抗値が変わる「曲げセンサ」というデバイスがあります(**写真3**)。曲げセン

注5) たとえば、1,2,3,4……と変わる通常変化に対して、1,10,100,1000と急激に変化する(この場合桁数)ものを対数変化と考えてください。

▼図5　AカーブとBカーブ



サには、センサの長さが55mm程度のものと、112mm程度のものの2種類があります。今回は55mmのほう[注6]を使ってみます。データシートによると、この曲げセンサは平面時に抵抗値が25kΩ±30%で、180°曲げると抵抗値が平面時の2倍以上になるそうです。

先ほどの分圧回路を使えば、この曲げセンサの曲がり具合を、mbedのAnalogInで読み取ることができます。**図6**のように曲げセンサを接続してみましょう。

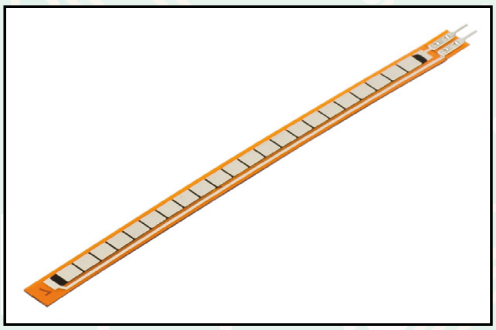
曲げセンサは、先ほど記したように25kΩですので、抵抗分圧回路のR1に10kΩの抵抗を接続します。抵抗分圧回路のVinには、mbedのVout(3.3V)を接続します。mbed LPC1768のアナログ入力端子に加えてよい電圧は最大3.3Vです。mbed LPC1768のアナログ入力端子はp15～p20ですが、p19とp20はmbedアプリケーションボードの半固定抵抗にすでにつながっています。ですので、ここではmbed LPC1768のp18を使って、分圧された電圧を読み取ってみます。

これで計算すると、mbed LPC1768のp18にかかる電圧は、Vinの25k/(10k + 25k) = 0.714倍になるはずです。mbedのAnalogInは、3.3Vのときに1.0を返しますので、AnalogInが返す値は、0.71くらいの値でしょう。実際にブレッドボードで回路を組んで(**写真4**)動かしてみたところ、0.74という値が表示されました。曲げセンサの抵抗値は、平面時に抵抗値が25kΩ±30%ということですし、このくらいの誤差はあるでしょう。

曲げセンサを曲げ、抵抗値が増えると、mbed LPC1768のp18にかかる電圧は、Vinの50k/(10k + 50k) = 0.833倍以上になり、1.0に近づいていくはずです。筆者の手元では、最終的に0.90になりました。

注6) http://ssci.to/508

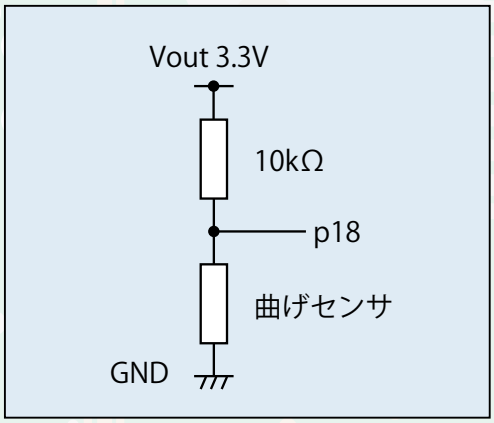
▼写真3　曲げセンサ



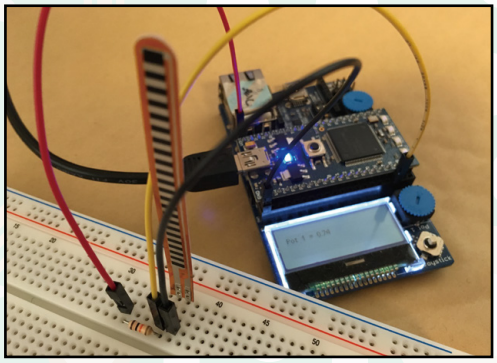
まとめ

AnalogInを使うと、0～3.3Vの間の電圧を測り、デジタル値にできます。抵抗分圧回路によって、抵抗値を電圧に変換できます。こうして、可変抵抗や曲げセンサの抵抗値の変化をマイコンで読み取ることができます。SD

▼図6　曲げセンサの接続



▼写真4　曲げセンサを使ってみた



読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2015年3月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

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

01 洗える防水キーボード「SKB-BS3W」

1名

表面にナノシルバー加工を施した抗菌仕様のキーボードです。保護等級「IP68」※をクリアした防塵防水設計で、キーボードの丸洗いができます。インターフェースはUSB（Aタイプコネクタ）、キーは、106キー＋ボリューム3キーで、109A日本語配列準拠です。対応OSは、Windows10/8.1/8/7/Vista/XP。

※「粉塵が中に入らない」「継続的に水没しても内部に浸水することがない」を満たした基準。

提供元 サンワサプライ　https://www.sanwa.co.jp

02 GitHub Tシャツ＆ステッカー

5名

エンジニアに人気のリポジトリサービス「GitHub」のマスコット「Octocat（海賊バージョン）」が描かれたTシャツ（Sサイズ）です。今回はOctocatのステッカーとセットでプレゼントします。

提供元 ギットハブ・ジャパン　http://github.co.jp

03 McAfee LiveSafe

PC、タブレット／スマートフォン向けのセキュリティソフトです。保護対象のデバイスは、Webの管理コンソールからその状況を確認できます。動作環境は、Windows 10/8.1/8/7、Mac OS X 10.8以降、Android 4.0以降、iOS 8以降（1年1ユーザ、台数無制限版）。

提供元 マカフィー
http://www.mcafee.com/jp

1名

04 Docker 実践ガイド

古賀 政純 著

Dockerの導入・運用ノウハウが凝縮された1冊。導入前のシステム設計、Dockerの基本的な利用方法、Dockerfileによる自動化の手法、管理・監視ツールなどについて、実際に操作をしながら解説します。

提供元 インプレス
http://www.impress.co.jp

2名

05 ネットワーク・デザインパターン

みやた ひろし 著

ネットワークの定番構成パターンを、豊富な図を使って解説。現代のネットワークを社内LAN、インターネット接続、サーバLAN、拠点間接続の4つに分け、それぞれの構成の最適解を提示しています。

提供元 SBクリエイティブ
http://www.sbcr.jp

2名

06 リモートチームでうまくいく

倉貫 義人 著

『「納品」をなくせばうまくいく』の著者が、自分が経営する企業でリモートワークを導入し、社員のワークライフバランスをうまくとりながら生産性を上げる「リモートチーム」を作り上げる過程を書いた1冊。

提供元 日本実業出版社
http://www.njg.co.jp

2名

07 ITインフラ監視［実践］入門

斎藤 祐一郎 著

クラウドの一般化で、インフラ管理を開発者が行うケースが増えた昨今。限られた時間とコストで、インフラ（とくにWebサービスにおけるインフラ監視）の設計・構築・運用のノウハウを解説しています。

提供元 技術評論社
http://gihyo.jp

2名

第1特集

なぜすぐリリースできるのか

チーム開発をまわす現場のアイデア

SUUMO　Retty　Qiita

Webサービスやi Phone／Androidアプリは次々に新しいものが登場してきます。とても活気があって挑戦しがいのある市場です。その激しい競争の中でユーザに高く評価され、継続して使われるためには、改善とリリースを“頻繁に”行うことが必要条件となりつつあります。しかし、どうやったらそんなに早いリリースができるのでしょうか？

本特集では、継続したリリースを実現するチーム開発のヒントとして、人気サービス／アプリを擁する気鋭の開発チームの方々に、自社で行っている手法や心得を教えていただきました。どこから手を付けたら良いのかわからない、というときのアイデアにつながることを願っています。

第1章

第1部

SUUMO流アジャイル開発［分析編］

現状分析からはじめた開発体制の改善

Author 吉田 拓真（よしだ たくま）
（株）リクルート住まいカンパニー
Mail t_yoshida@r.recruit.co.jp
Facebook https://www.facebook.com/takuma.yoshida.355

本章ではSUUMOのスマホサイトで取り入れているアジャイル開発について説明していきます。アジャイル開発を導入するにあたり、さまざまな課題に直面し、議論を重ねてきました。ここではそんな議論を通して得られた1つの答えをなるべく具体的にご紹介できればと思います。

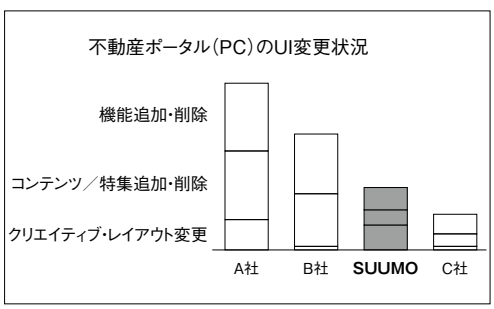
はじめに

手前味噌で恐縮ですが、現在のSUUMOスマホサイトは、比較的大規模な開発体制と言える中で、新しい技術の導入、たとえば日本で初めてのService Workerを用いたWeb-Pushや、Polymerを利用したローンシミュレーション機能など、Webの先端技術を取り込みながら開発ができている状態にあります[注1]。

今回はこの場をお借りして、先端系の技術も取り込めるある程度の余裕を持った開発体制の構築をするために我々が何を考え、何を実施してきたのか、具体的な細部まで紹介できればと思います。

注1） 実はChrome Developer Summit 2015においても日本で唯一SUUMOのWebサイトが紹介されています。https://tech.recruit-sumai.co.jp/中、「SUUMOスマホサイトへのService Worker導入① add to home screen編」に記事あり。

▼図1　SUUMOと競合他社のUI変更回数の比較



SUUMOの開発が遅いらしい……？

2013年10月、開発に関するアンケートを内部で実施したのですが、開発が速いと感じる意見がなく“遅い”といった意見が目立ちました。このままではよくない、ということで何がどのようにどれだけ遅いのか、定性意見だけでなく定量的にも調査してみました。

結果が**図1**になります。定量調査の具体内容ですが、紆余曲折を経てWebのフロントエンドに関するUI変更数を競合他社と比較するという形で実施しました[注2]。具体数は省きますが、残念ながら競合他社に比べて“速い”とは言えない結果となりました。

つまるところ、「SUUMOが**プロダクトの改善回数であるリリースの頻度が競合に劣っている状況**になりつつあるのでは」という話になりました。このままでは時代の流れに取り残されてしまう危機感を覚えることになり、改善に向けた取り組みがはじまりました。

注2） もし開発生産性を厳密に測る場合は「MM÷FP（総開発工数÷ファンクションポイント）」によって算出するのもいいのでは、との話も挙がりましたがこれは見送りました。そもそも開発生産性の高低を同業他社との比較が現実的に難しいということと、FPのモニタリングの基準設定や算出に手間がかかるためです。

開発のリードタイムが長い理由

実際に、なぜ開発が遅い（＝リードタイムが長くなる）のかをもう一段階調査してみました。**図2**の左が、実際に現場にヒアリングした課題をまとめた結果で、中央列がその課題がもたらす悪い影響を示しています。

中身を見てみると、開発に関わる人間が多いことで生まれやすいオーナーシップがまだ十分に持てていない、とか、失敗しないためについつい確実性を追求し過ぎてしまう、といった、どこかで聞いたような話が浮き彫りになりました。そしてこれらの課題を1段階抽象化したのが**図2**中の右になります。「システム（プロダクト）」「制度・ルール」「文化・風土」各々に課題が存在するのであろう、ということがわかりました。

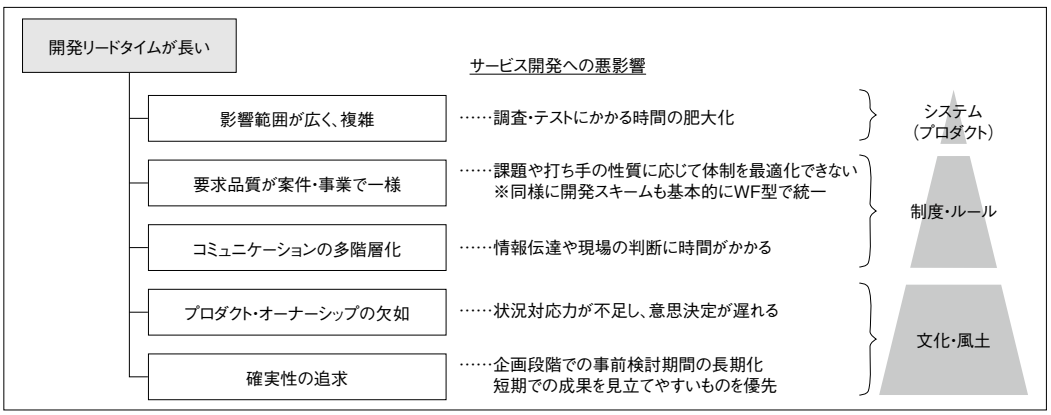
SUUMOで取り入れた新しい開発スキーム

競合他社に負けないためには、少なくとも現在と同じ工数でリリース回数を増やすということが必要です。これを実現するためには要件定義〜開発までの期間（開発リードタイム）を短縮すればよいことになります。

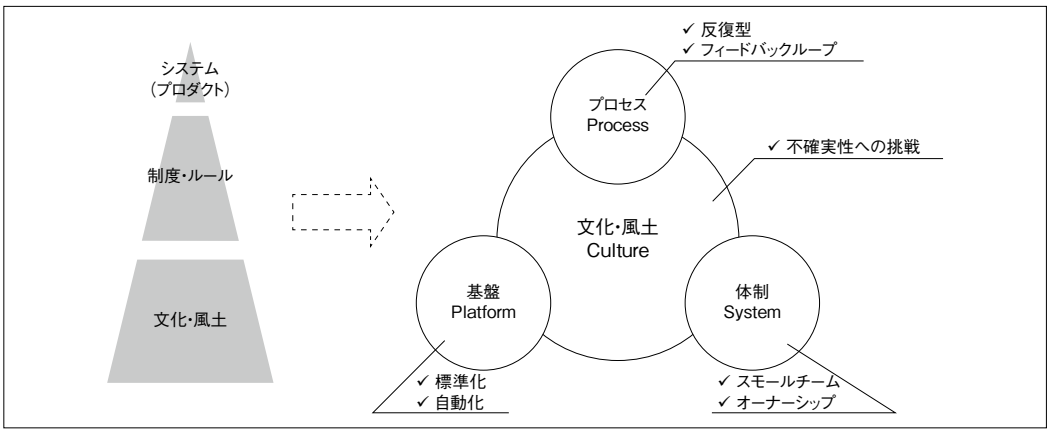
ということでまず前述の開発リードタイムを長期化させている要因に打ち手をマッピングする形で開発スキームを設置しました（**図3**）。

このスキームのポイントですが、①国内外のアジャイル開発事例研究を通じて我々に合致するように細かいところを最適化している

▼図2　開発リードタイムが長い原因と、サービスへの悪影響について



▼図3　開発リードタイムを縮小する開発スキーム



ということ、また実際に②F/S[注3]によるブラッシュアップを数回繰り返し実用レベルまで昇華させているところがミソです。とくに②に関してですが机上だけではなく実際に試す→ブラッシュアップする、ということが非常に大事だと考えています。事例研究を進めると、導入を失敗する企業の多くが、いきなりアジャイル開発のプロセスや体制だけを試してみてうまくいかなった話をよく見かけますが、試す→社内にあったやり方に変える、というブラッシュアップを行い続けることが必要なのではと考えています。

実際に試すとわかるのですが、プロセスだけでなく、その根底にある文化や風土の改善も同時に行わなければ、けっして新しい開発スキームの導入ができないことがわかります。

それでは次節よりこの新しい開発スキームの具体例を紹介していきます。

文化・風土

最も重要であり、また変えることも難しい文化／風土についてですが、これは根底にある考え方を合わせるところから始めました。

まず**図4**を見てください。今までは企画→開発を一気にやりきるという形でしたが、これだと検討が甘い場合に効果目標に達しない恐れがあります。この不確実性を可能な限り小さくしようとした結果、企画が長期化しやすいという課題がありました(**図4左**)。

対して新しい開発スキームでは、そもそもWebというマーケット自体が不確実なもので溢れていることを前提とした考え方にシフトしました。具体的に行ったこととしては案件の見立てをある程度で打ち切り、案件を小さく区切って高頻度にリリースするという形に変えました(**図4右**)。

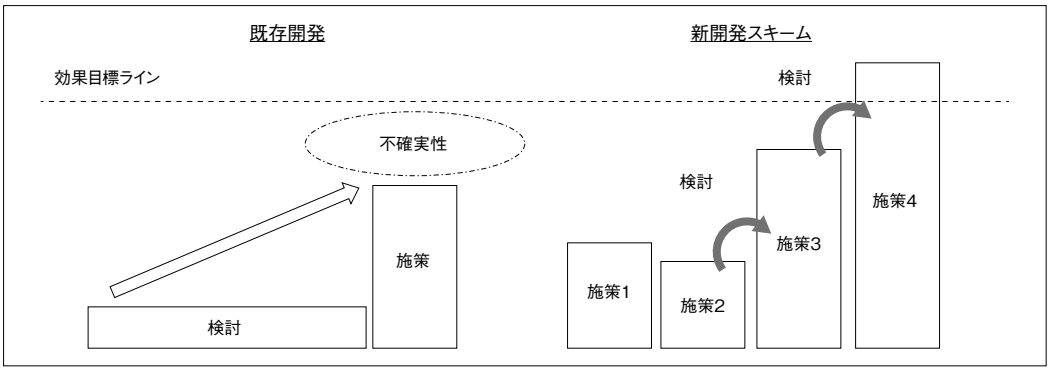
こうすることでリリースごとにカスタマーからのフィードバックが得られ、打ち手の方向性や質を、都度確認しながら進めることができ、設定した効果目標を追いやすくなるのではと考えました。

ただ、よく勘違いされるのですが、上記はあくまで基本となる考えであって、すべての案件に当てはまるわけではありません。下手をするとただの“行き当たりばったりな開発”になってしまいます。**事前に効果見立てや予測が立つような案件に関しては、今までと変わらずきっちり計画してやりきったほうがよく、解決策に唯一の正解がないような案件(≒不確実性が高い案件)に関しては、上記の考え方を当てはめるべき**と考えています。

ただし、不確実性と向き合うことで問題を小さく切って挑戦していくという考え方ができたとしても、これだけだとまだ課題があります。挑戦が評価されない環境では、実際に行動に移すことが難しいからです(挑戦して失敗したら周りから冷たくされる、という状況

注3) Feasibility Study。実現可能性調査。

▼図4　不確実性と向き合う考え方



を想像してみてください)。つまり**考え方が変わったとしても、同時に環境も変えなければ実際の行動につながらない**ということです。

図5左が実際に文化・風土を構成するヒエラルキーを示しています。独自の考えにはなりますが、文化・風土はそれぞれマインドと環境によって構成されていて、環境は責任・権限・資源・評価の4つから成ります。

冒頭の不確実性と向き合う話はマインド部分であり、これだけ変えても無理があるという話でした。では環境はどのように変えるかですが、こちらは重要なポイントが2つあります。それは**責任と権限の適切な分配**にあります。

まず責任の分配とは、明確なKPI[注4]を事業部門と合意のうえ、施策実現に対しての責任をチームが持つということです。具体的にはシンプルなKPI(例：コンバージョン数を120%にする)を策定し、チームのオーナーと事業部門の間できちんとすり合せることを行いました。

次に権限の分配ですが、KPIの実現に向けた各個別施策の意思決定権をチームに委譲するということで、たとえばKPIを120%にする個別の施策の立案・実施はすべて現場に任せることを指しています。こうすることでチームが自らの意思で動きやすくなるため、投資対効果(ROI)を見立てにくい案件などもチームの裁量で実施できるようになります。

注4) Key Performance Indicators。重要業績評価指標。

長々と説明しましたが、体制やプロセスだけではなく環境も同時に変えることも重要ということが伝わればと思います。

体制

新開発スキームでは、5〜8人で構成されるような小さい規模のワンチームで1つのプロダクトを見るようにしました(**図6**)。またメンバー同士が物理的にも近くなるように、席替えなども行っています。

小さい規模にした意図ですが、コミュニケーションを効率化させることと、コラボレーションの創出の2点を狙ったものです。

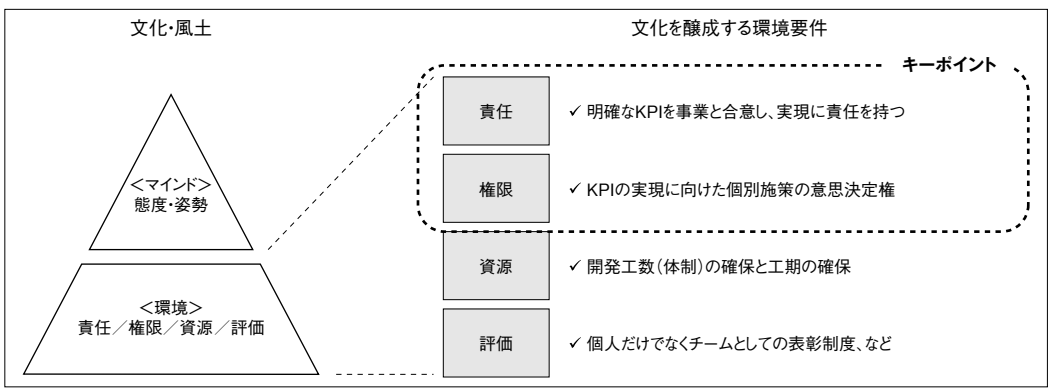
従来では実際の作業をする人が思ったことや考えたことが企画者までには伝わりづらく[注5]、単に依頼する側と受ける側という形になりがちでした。

たとえばある企画や施策があったときに、決められた工数や期間で実現できるのか／できないのか、ということ以上のコミュニケーションが生まれづらい状況になっていました。

新開発スキームでは企画者と作業者の距離が非常に近しいことと、企画者が課題背景の共有や仮説設定を個人ではなくチーム単位と

注5) 伝わりづらい原因はおもにコミュニケーションが多階層になることが問題であると考えています。企画者、デザイナ、ディレクター、開発者といったいろいろなステークホルダーを介すような形で案件が進むのですが、企画から遠くなればなるほど細かい意図が伝わりづらくまた企画者に聞きづらいという課題があります。

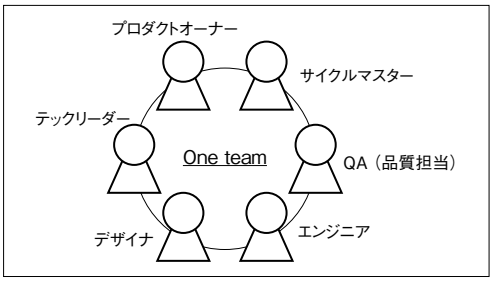
▼図5　文化・風土の構成要素と、醸成するための要件について



して実施するため、「できる／できない」ではなく「どうやったら実現できるか」へ議論が自然にシフトするような状態が作りやすくなります（コラボレーションの創出）。

またそれぞれ役割を明確にし、通常体制では存在しなかったプロダクトオーナー、スクラムマスタ、テックリーダー、QA（品質担当）という4つの役割を新規に導入しました（**図7**）。

▼図6　小さい規模のワンチームで1つのプロダクトを見る



ワンチームというのも大きなポイントで、これはプロダクトに対するオーナーシップを従来よりも持ちやすくすることを狙っています。今までは**図8左**にあるように、デザインやエンジニアを機能組織から個別にアサインをしていました。

ただ時期やそのときの状況によってアサインされるプロダクトが異なるため、「自分がプロダクトをよくしていくのだ」という責任感やそれを生み出す愛着感を持つのがなかなか難しいとの声が内部で挙がっていました[注6]。

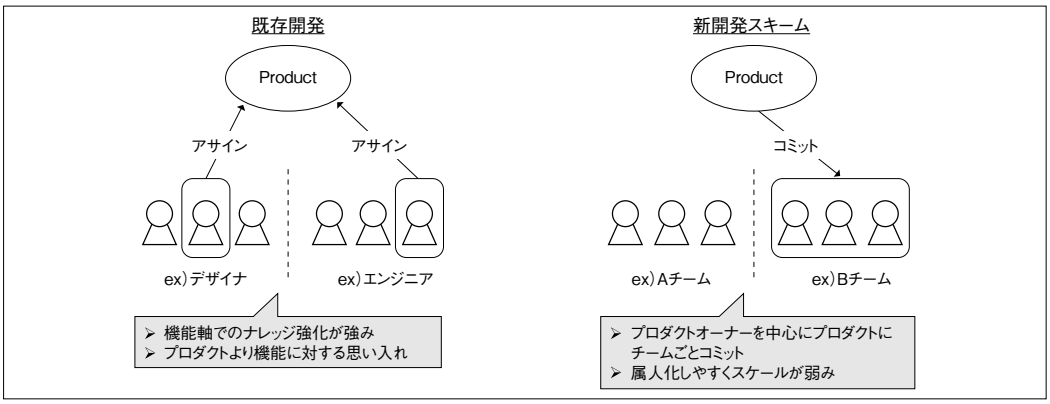
ですので、新しいスキームでは**図8右**にあるように**チームとして長期間同じプロダクトを担当する**ことで、従来よりも愛着感や責任感を持ちやすくするようにしています。ただしこうすることで逆にプロダクトがそのチーム

注6） 内部組織が半年単位でガラっと変わることも珍しくなく、逆に同じプロダクトを1年以上担当することのほうがまれ、みたいな感覚があります。

▼図7　役割についての説明

役割	通常体制での役割	新開発スキームでの役割
プロダクトオーナー	✓ 不在（近い役割はPL） ✓ 企画担当としてすべてを決定	✓ KPIの設定 ✓ 要件やプロダクトに対する意思決定
サイクルマスター（＝スクラムマスター）	✓ 不在（近い役割はディレクター） ✓ 進捗管理に従事	✓ KPI実現方法の決定、プロセス管理 ✓ チームファシリテーション
クリエーター（エンジニア・デザイナ）	✓ 要件に応じて実装	✓ 実装 ✓ 実装方法のボトムアップ、企画参加
テックリーダー	✓ 不在（近い役割はベテランのエンジニア） ✓ 非機能要件に関するアドバイスなど	✓ 非機能要件の担保 ✓ 技術支援（自動化、各種ツール作成）
QA・品質担当	✓ テスト担当 ✓ エンジニアが兼任 or テストのみ切出	✓ レビュー、テスト（自動・手動）

▼図8　よりオーナーシップを持ちやすいチームとする工夫



ありきになったり、同じようなチームを作りづらいため、横展開しづらい（スケールしづらい）といったデメリットもあります。しかしながら、現状ではデメリットの方が大きいからワンチームはやめよう、といった話は出たことはありません。

プロセス（サイクル）

開発のサイクルは**図9**にあるように反復型開発を基本思想としています。すでに文化・風土のところでも説明しましたが、不確実性と向きあうために仮説検証のPDCA[注7]を短縮化させ、それによる効果と学びの獲得機会を増やすことを目的としているためです。

ここではいくつか独自に工夫しているところと、共有したいポイントを絞って説明します。

そもそもなぜ反復のサイクルの長さを2週間にしたのかということですが、これは社内で実際に試した結果、チームが最もやりやすかったとの声が多かったことから上記の期間に設定しています。ほかに試した期間のアンケートを紹介すると、1週間だと精神的に疲れやすく継続するのがキツい、とか、3週間だと中だるみしやすい、などがあります。

また**図9**中のTech-Cycleについてですが、これはゴールデンウィークや年末など10営業日を満足に確保できないときに実施する特別なサイクルを指しています。

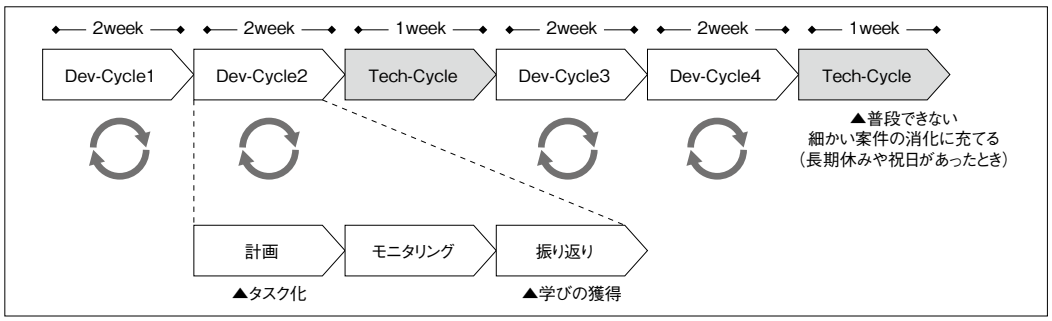
この期間は、いつも気になっているが解決できなかった優先度が低い案件[注8]を解消することに充てています。

役に立つかどうかはわかりませんが、実際に1サイクルの中でどのようなことを行っているのかをまとめてみました（**図10**）。1週目に開発＆テスト効率化に向けた開発計画の立案、2週目の水曜日にリリース、2週目の木／金は

注7） Plan（計画）、Do（実行）、Check（評価）、Act（改善）。

注8） 例：ドキュメント群の整理、1pixelのズレ、まれにしか起こらないが個人的に気になるバグ、普段のルーチンワークを解消するためのツール開発、などがあります。地味ですがやってみるとかなり良いです。精神衛生上、スッキリします。

▼図9　反復型開発のイメージ。仮説検証・学びの回数を多くすることを目指している



▼図10　2週間サイクルの内訳（スマホチーム）

		月	火	水	木	金
1週目		開発期間				
	午前	テスト骨子計画				
	午後			チーム会	管理定例	
2週目		テスト期間		リリース日	改善期間	
	午前			リリース作業		
	午後		リリース判断		振り返り会 管理定例	全体計画会

ドキュメント化や振り返り、次のサイクルに向けた計画を行っています。

基盤

新しい開発スキームを実践するうえで必要になってくるツール群(JIRA/HipChatなど)や開発環境のことを基盤としています。プロセスや体制を変えるのであれば、ベースとなるツール群もそれに合せて変えたほうがよりチームとしてのパフォーマンスを発揮できると考え、基盤を整備する専任チームを定義し、移行を実施しました。

基盤に求められる要件として重要になるのが、既存課題をなるだけ解消できていることです。こうありたい、こんなツールを使いたい、といった要望を持つのも重要ですが、きちんと既存課題が解決できていなければただ単にやりたいことをやってみました、になりかねません。

ですから基盤チームではまずは 1.既存課題、2.打ち手の方向性、3.具体的なツール／解決手法、をそれぞれマッピングし、現状と目指すべき姿を定義しました(**表1**)。

この表があることで、どのツールが何の課題に対する打ち手なのかが明確になり、またどこがゴールなのかをイメージしやすくなりました。

また基盤ツールの導入だけでなく、プロダクトのアーキテクチャ(ER[注9]／モジュール凝集度など)も非常に重要なポイントです。なぜならばいくら開発プロセスが優れていたとしても、複雑過ぎるシステムでは本来の力が発揮できないためです。

図11はシステムの複雑度合いをイメージしてみたものです。図中左にあるようデータの橋渡しが入り乱れると、開発がしにくいばかりか自動化もしづらくなるのは想像に難くないと思います。開発プロセスをうまく回そうとしたときは、やはりシンプルなシステムであることが求められるということが伝われればと思います。

新開発スキームの導入を行った結果……

新開発スキームの導入を行ってどうなったかについてですが、最終的に**図12**のようになりました。既存(ウォーターフォール型)・新開発スキーム(アジャイル型)のチームが混在する形になっていて、全体で40～50名ほどの規模です。

なぜ混在しているかですが、開発の規模が非常に大きく[注10]アジャイルのチームだけでは開発をまかなえないことが大きな理由です。したがって案件の種別に応じて、ウォーターフォール／アジャイルのチームを切り替えていて、おもにUI/UXにかかわる案件をアジャイルのチームが、そうでない案件を従来のウォーターフォールのチームが担当するようにして

column

カンバンについて

進捗管理にはカンバンを用いています。各担当の業務ボリュームの把握と進捗を見える化することで、進捗管理にかかる時間の短縮とコミュニケーションロスを防ぐことを目的にしていますが、私たちのチームではより効率的にするために壁に付箋を貼る形ではなく“デジタル”なカンバンをもちいています。

実際にアナログのカンバンでいくかデジタルでやるか実際にやってみて比較検討したのですが、やはりブランチを切ったり納品物の添付をしたり議論経緯を残したりストーリーポイントを記録・集計したり、といった開発の上で起こるアクティビティについてアナログのカンバンがついていけず、結局AtlassianのJIRA Agileを採用しました。具体的な使い方は以前の記事[注A]を見てください。

注A) 2015年5月号の『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』。

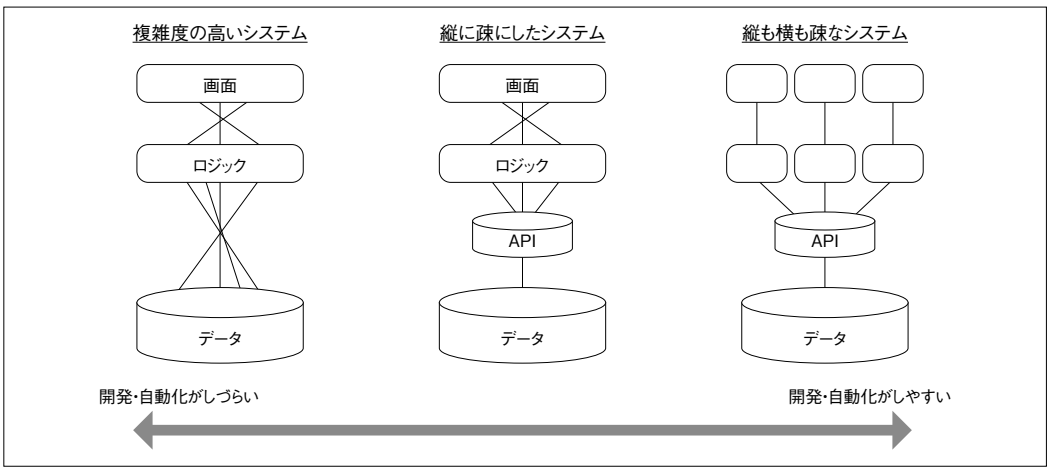
注9) Entity-relationship Model。実体関連モデル。

注10) SUUMOは賃貸、新築マンション、土地といった複数の領域があり、それぞれの領域でやりたいことを合算すると、どうしても開発規模が大きくなります。領域ごとにまったくデータ構造が違うので複数サイトを同時に開発しているようなイメージです。

▼表1　基盤チームが整理した課題・打ち手・ツール・目指すべき姿をまとめた表

開発フェーズ	重点課題	打ち手	キーワード	現状○／目指すレベル★ Lv1	Lv2	Lv3	Lv4	Lv5	優先度
要件定義 設計	コミュニケーションコスト大	ツール整備	JIRA、HipChat		○	★			1st
開発	品質低下	コード品質定量化	SonarQube	○		★			3rd
		テスト自動化	Selenium、E2E						
	開発効率低	リファクタリング	クラス整理		○		★		2nd
		テスト環境整備	検品環境拡充						
	開発不透明	開発コミュニケーション見える化	PR開発、Bitbucket			○	★		4th
テスト リリース	…	…	…	…					…
～以下省略～									

▼図11　システムの複雑度と新開発スキームの適用のしやすさのイメージ



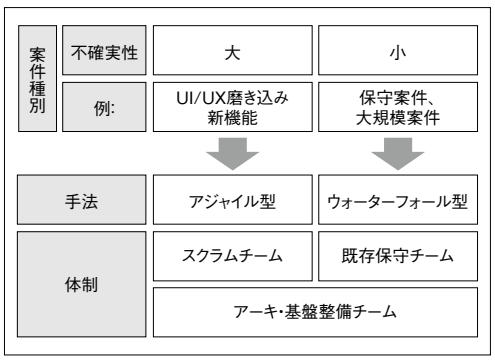
います。また開発基盤の保守や小物ツールの作成などは、上記の2チームとは別に専任のチームを置いてサポートする形になっています。

結局のところ、この混成チームにしてから良かった／悪かったのか、どちらかで言えば良かったことのほうが圧倒的に多いです。

導入してから約1年とちょっと経ちますが、事業とチームの間で定めたKPIに関しては今のところ目標数値を下回ったことはなく、むしろ超えることの方が多く、またシステム的にも経年劣化どころか以前より改修しやすい状態に改善し続けられており、むしろWebの最新技術であるService Worker/Polymerなどにいち早く追従できるような状態です。

結果論にすぎませんが、Webの最新技術を追える開発体制を目指して打ち手を考えて実践していくよりも、開発プロセスとそれを支える環境を整えて、あとはチームにお任せするといったほうが近道かもしれません。SD

▼図12　新開発スキームの導入と、既存開発スキームとの使い分け



第2章
第1部
SUUMO流アジャイル開発［データ編］

技術的負債、コンバージョン、パフォーマンスの“見える化”

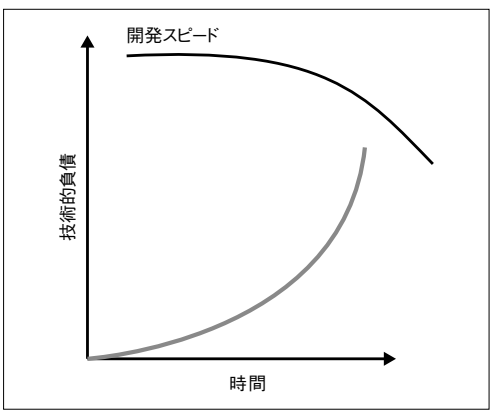
Author 吉田 拓真（よしだ たくま）
（株）リクルート住まいカンパニー
Mail t_yoshida@r.recruit.co.jp
Facebook https://www.facebook.com/takuma.yoshida.355

前章では、SUUMOでの新しいアジャイル開発スキームの概念や定義を紹介しました。この新開発スキームを運用していくにあたり、非常に重要になってくるのが各種指標（KPIなど）の見える化です。KPIが見えていないまま開発を進めてしまうと、行き当たりばったりな開発になりがちです。とはいえ、見える化といっても具体的にどうすればいいのかお悩みの方も多いと思われます。そこで本章では、SUUMOのスマートフォンWebサイトで実際に行っているデータの見える化をテーマに、KPIやログといった基本的な可視化から技術的負債やWebサイトのパフォーマンスといったより実践的な見える化までを紹介します。

新スキームの導入だけでは不十分?

新スキームは単純に開発のやり方（プロセス）のみを示していて、対象となるシステムが複雑な状態だと本来のパフォーマンスを十分に発揮できません。そこで新スキームの導入の傍ら、既存の複雑化しつつあるコードをよりシンプルに作り替える必要がありました。加えてウォーターフォール／アジャイル型のチームが同時に開発してもコンフリクトが起きにくいような状態にすることも必要でした。これらのリファクタリング活動を円滑に進めるために取り組んだのが技術的負債の見える化です。

▼図1　技術的負債を放置したまま時間が経てば、プロダクトの改修にかかる時間も長くなり、結果としてカスタマーへの価値の提供が遅くなる



技術的な“負債”が開発速度を下げている?

それでは技術的負債というのはいったい何を指しているのでしょうか。技術的負債とは、設計のまずさやコーディング規約に反するコードなど、開発を妨げるような課題の総称です。具体的には先ほど述べた“複雑過ぎて開発のスピードを遅くしている”ということだったり、“コンフリクトが思ったよりも多発する”というような事象を引き起こすものを指しています。

技術的負債は増えれば増えるほど、開発にかかるコストが増え→開発速度が下がり→カスタマーの期待に応えるまでの時間が延びる、というやっかいな性質があります（**図1**）。

また負債の名前が示すとおり借金と同じような性質を持つため、返済せずそのままにしてしまうと雪だるま式に大きくなってしまいます。

こう説明すると非常に恐ろしいものに見えますが、ポジティブにとらえるならば継続的に返済さえできれば開発速度を落とさずに開

発ができるということを意味しています。つまり高速なPDCAを回す開発活動を中長期的に維持するためには、技術的負債を見える化（債務整理）し、計画的な返済計画を立てて実施すればよいということになります。ただこの見える化についてはコツがあり、定性的かつ定量的に行う必要があります。それではこれらについて詳しく説明していきます。

負債の債務整理（定性）

まず定性的な見える化ですが、これは非常に単純な話で、開発を行ううえで気になった点を記録したものになります。この表を作る目的は、返済すべき負債をできる限り漏れのないよう管理することと、着手すべき順番を決めるときに利用するためとなります。

具体例として**図2**に、実際に作成した課題表の一部抜粋を示します。この表はConfluenceで作成していて、誰もが記述してもよいのですが、テックリーダー（TL）と呼ばれる役割が主担当としてこの表を管理・維持・解決の推進をしていきます。

この表は作るとわかるのですが、単純に作ると項目があり過ぎて何から着手していけばよいのかわかりづらくなってしまいます。そこで工夫としてそれぞれの課題について優先順位を記載するようにしています。

さらに優先順位の決め方にも工夫をしていて、**表1**のような判断マトリクスを用いて、ある程度機械的に判断するようにしています[注1]。

負債の債務整理（定量）

次に定量的な負債の見える化ですが、これは基本的にツールなどで機械的に検知・計算できるものを指しています。具体的にはソースコードを静的解析して得られるものが大半で、コーディングルールの違反数、複雑度、テストカバレッジ、重複度（コピー&ペースト割合）などを指しています。とくにコーディングルールの違反数は小さければ小さいほど読みやすいということになるため、常に最小に留めることが開発スピードの担保につながります。

注1）基本的な考えとしては**工数が小さくかつ想定効果が大きい案件を最優先**とするような考え方をベースとしていますが、工数が大きくて効果が小さい案件も実施するようなことはあります。これは時と場合に応じてテックリーダーが判断します。

▼表1　技術的負債の優先順位を決めるための判断マトリクス

優先順位		工数		
		Small	Medium	Large
効果	A	A	A	B
	B	A	B	C
	C	B	C	C

▼図2　定性的な技術的負債の見える化（通称TL課題管理表）

情報共有＞仕様書

大目標	対象	課題内容	起票者	性能効果	生産性効果	工数	優先順位	担当者
生産性	仕様書	ビーコンの仕様書 • 画面一覧にもあるし、別仕様書にもある		-	A	M	A	
生産性	仕様書	Cookieの仕様書 • confluenceからSVNに移行		-	C	S	B	
生産性	仕様書	ローカルストレージ • 仕様書を統合＆最新化 • isLogin について書く		-	C	S	B	
生産性	仕様書	DB設計書 • DB設計書にDDLを追加する		-	C	M	C	

この静的解析にはSonarQubeと呼ばれるツールを用いていて、実際の解析結果を**図3**に示します。具体的な活用方法ですが、このようなダッシュボード画面をリリース前後に確認したり、Jenkinsとリポジトリの機能を活用(**図4**)して**図5**にあるようにプルリクエスト単位でどれだけ増えたのか減ったのかを確認していたりします。

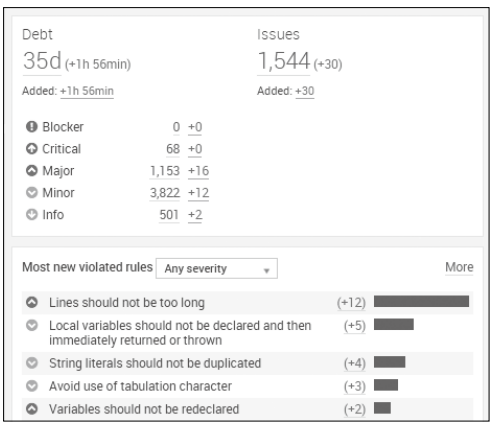
返済を継続的に推進するためには

それでは実際に見える化した負債を継続的に返済するために、どう工夫しているのかについて説明します。端的に言うと、アジャイルのチームの中ではプロダクトを直接的に改善するUI/UX改善系(通称プロダクト改善)だけでなく、プロダクトを技術的な観点から改善する系(通称開発改善)も1サイクルの中で必ず一定量こなすというルールを設けました(**図6**)。

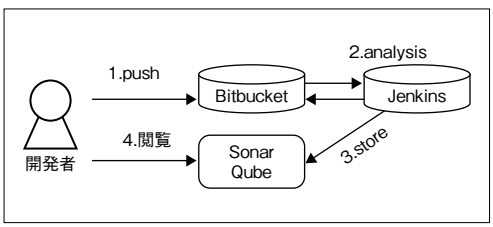
またチームの中でも開発者がそれぞれの専任のテーマを持つようにしていて、プロダクト改善をメインに担当するエンジニアと、開発改善を担当するエンジニアに分けています(**図7**)。

図6中のプチ改善という項目ですが、これはけっこう大きなポイントです。上記のように負債を一定量こなすという形にするだけだと、優先度が低い案件(例：句読点の打ち方が変、1pixelのずれを直す、など)はかなり長い期間

▼図3　SonarQubeで得られる指標の抜粋。ここでは負債の大きさと具体的な違反数、および内訳が表示されている(数値はダミーです)

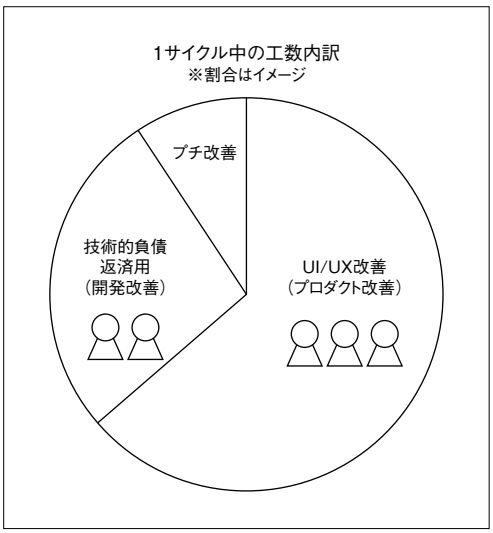


▼図4　開発者がリポジトリを更新するごとにJenkinsでSonarQubeを動かす

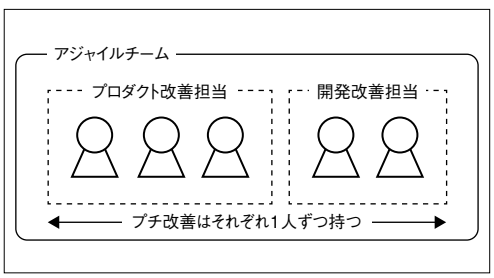


▼図5　プルリクエスト単位でSonarQubeの解析結果を通知するしくみの例

▼図6　1サイクル中の案件割合のイメージ。負債返済用の工数は3〜4割ほどある



▼図7　プロダクト改善、開発改善というエピック単位でエンジニアの担当が決まっている



後回しになってしまいます。ですから、優先度が低くて改修コストが小さい案件は“プチ改善”と名付け、1サイクルの中で開発者1人が1つの案件を必ず実施するようにしています。

我々は現在このようなアプローチで、技術的負債を特定→継続的に返済することに関して一定の成果を出すことができています。ですが本来は返済することに躍起になることではなく**負債が蓄積しづらいような設計・しくみ作り**を実践することが重要であるため、今後はしくみを作るというところに力を入れていく予定です。

KPIの見える化 ～コンバージョン～

SUUMOのスマホサイトではKPIとして、資料請求数（賃貸領域の場合）の拡大とパフォーマンスの改善という2つを主なKPIとしています。ここではまず資料請求数（コンバージョン）の見える化について説明します。

資料請求数、足りてる?足りてない?

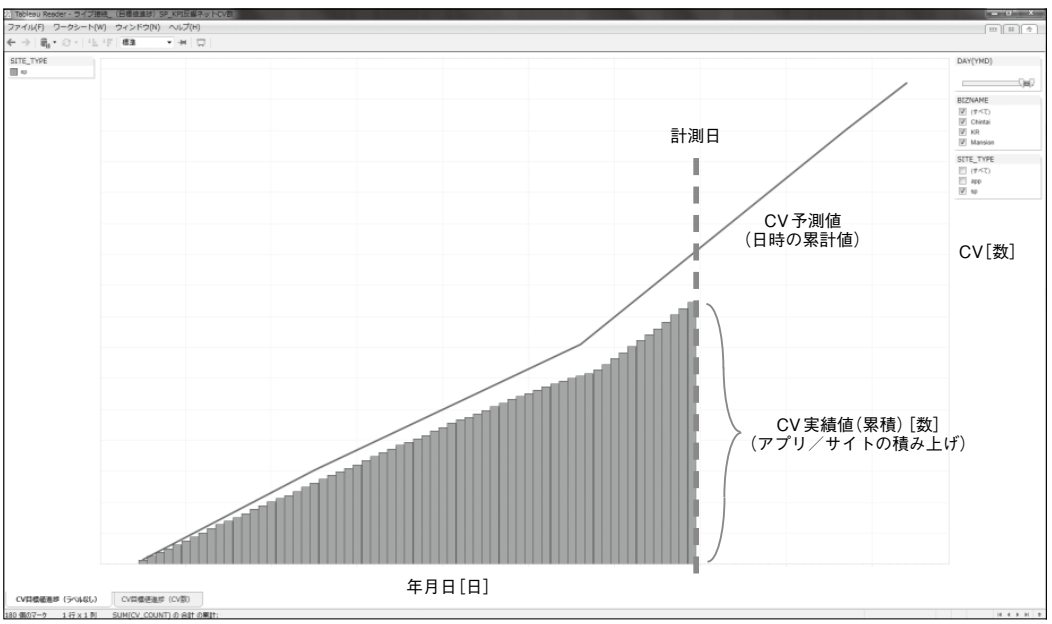
まず最も単純な見える化として、目標数と現状の進捗を可視化するという見える化があります。たとえばKPIである資料請求数を前期比+○○%とします、と言われても今日現在資料請求数が足りているのか足りていないのか聞かれてもすぐにわかりませんよね。これが瞬時にわかるようにするためには**図8**のような見える化が必要です。

この見える化にはTableauというBIツールを用いており、この画面は朝会で1回／週ぐらいの頻度で予測に対する実績がどうなっているのかを確認するようにしています。この見える化を実施したことで以前よりも早くに異常を察知できるようになっただけでなく、KPIの雲行きが怪しい場合はすぐさま先手を打てる状態になりました。

UI/UXの改善を見える化する

読者のみなさんも一度はご経験されたことがあるかと思いますが、ページごとのCTRや

▼図8　資料請求数の予測・実績を確認する（Tableau）



CVR[注2]の変動をいちいちモニタリングツールを開いたりExcelでまとめるのがつらいと思ったことはありませんか。高速なPDCAの開発ではこういったサブ指標は頻繁に参照するため、このモニタリングにかかるコストを最小にすることは非常に重要です。

ということで実際に社内のデータ専門チームと協業し、今までのモニタリングを見直し、画面に関するモニタリングの大部分を自動化&見える化しました。実際の画面が**図9**になります。数値を単純にまとめるだけでなく、よく使うグラフを10種類以上用意していて、このレポート上で数年前から～現在までの主要指標をインタラクティブに閲覧できます。

また詳細な数値は**図10**のようにまとめています(一部具体的な数字はぼかしてあります)。こちらは主要ページのCTR／CVR／離脱率なども各領域ごと・ページごとに日次・週次・月次などで確認ができます。このままExcelにも変換できるため、指標を見るだけであればもはやExcelは必要ない、というぐらいの状態に仕上がっています。

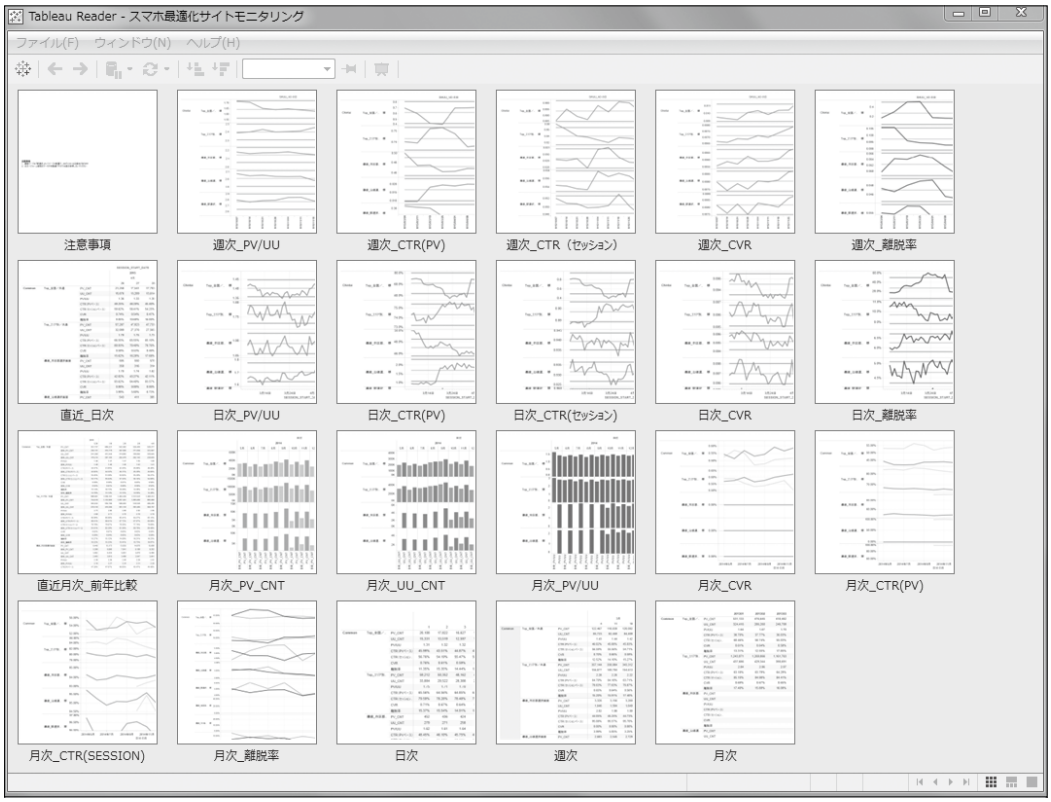
これらのTableauを用いた可視化についての具体的な実現方法は誌面の都合上割愛しますが、**図11**に示すようなフローで自動的に生成しています。

KPIの見える化 ～パフォーマンス～

SUUMOでは快適な家探しを実現するために、サイトのパフォーマンスをできる限り高速化

注2) CTR(Click Through Rate)。広告がクリックされた確率。
CVR(Conversion Rate)。アクセス数に対しての成果率。

▼図9　UI/UXの改善を確認するためのモニタリングシート。よく使う10種類以上のグラフが自動的に用意される(Tableau)



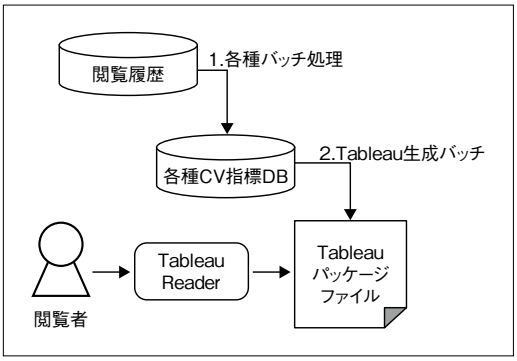
しようと試みていて、高速化というのがKPIの1つでもあります。

ではここでいうパフォーマンスの定義ですが、モバイルサイトとして備えるべき性質を持っているかどうかをベースとして、次の2つから構成されるものとして定義しました。

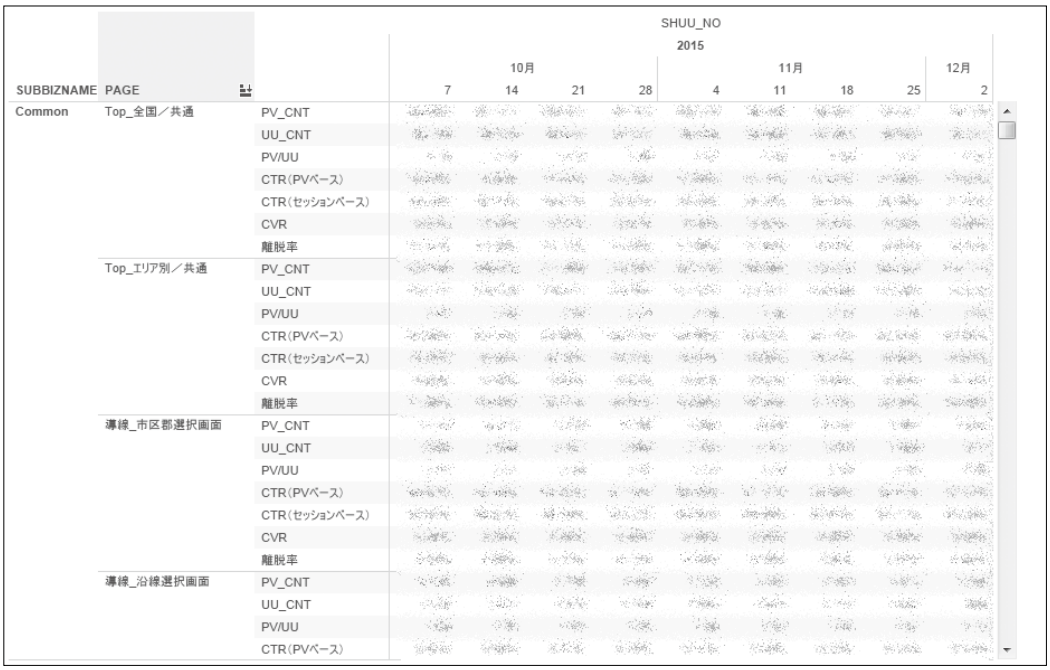
①ターンアラウンドタイムができる限り短いこと

ターンアラウンドタイム(Turn Around Time；TAT)とは、アプリケーションの応答速度を表す指標の1つで、Webアプリケーションにおいてはユーザがアクセスしたときにサイトが閲覧できるようになるまでにかかる時間を指します。デバイスの性能やネットワーク環境など多くの環境要因によって変動しやすいため厳密に定義することが難しいのですが、定量化しないことには何も始まりません。

▼図11　Tableauの生成フロー



▼図10　主要ページのCTR／CVR／離脱率などを、各領域ごとに日次・週次・月次などで確認ができる。全自動でまとめられるため、Excelでまとめる必要がない



▼図12　Webページが表示されるまで(TAT)にかかる処理の概念図。とくにサーバ処理／クライアント処理の部分は自助努力による改善がしやすい部分

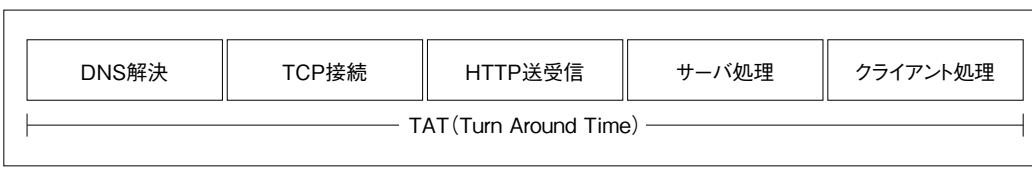


図12はWebページが表示されるまでにかかる各種処理を簡略化したものです。これらの処理のうち、それぞれを小さくすることでより速くユーザにコンテンツを提供できることになります。

②モバイルサイトとしてのお約束を守っていること

お約束とは一言で形容することが難しいのですが、下記のような一見するとアタリマエに近い暗黙のルールを指しています。また下記の一部は、暗黙的ではなくGoogleのSEOガイドラインに定義されているような項目もあります。

- HTML/JavaScript/CSSといったマークアップ言語やスクリプト言語に文法的な誤りがないこと
- 静的コンテンツ(画像／JavaScript/CSS)のサイズが可能な限り圧縮されていること
- 明らかにユーザがタップできない／表示できない位置に要素が存在する、といったユーザビリティを阻害するマークアップがされていないこと

いくらTATが速くても上記の項目が守れていないコンテンツは、ユーザにとって有益になるとは限らないため、これらのルールに則ったコンテンツを提供できているかどうかを継続的に確認することは重要です。

パフォーマンスを定量化してみる

それではTATと規約遵守度の2つをどのように定量化したのかですが、結論から言うとGoogleが用意しているPageSpeed Insights[注3]という計測ツールと、WebPageTest[注4]と呼ばれるOSSを複合して定量化しました。定量化を行うシステムの全体像を**図13**に示します。

ここでは詳しく説明しませんが、PageSpeed InsightsはWebページの速度とWebのお約束に関してのスコアを0～100点までの数値で定量化してくれるという非常に優秀なツールです[注5]。

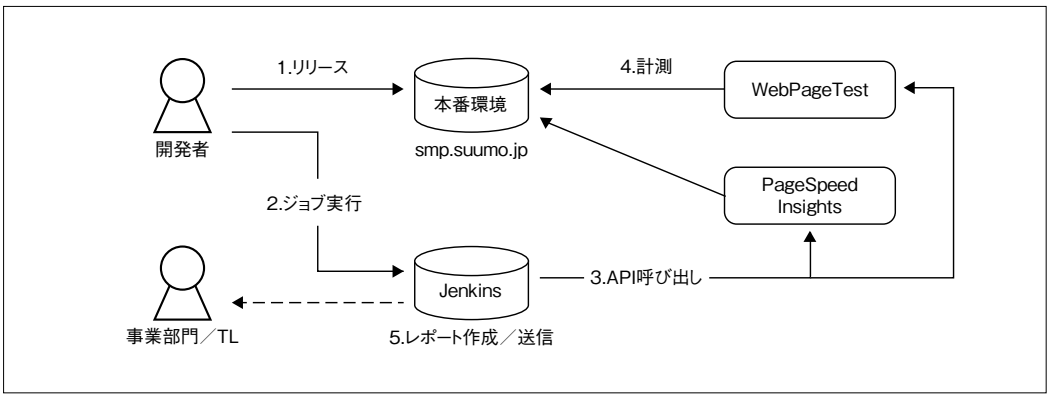
実際にどのように使っているかですが、公開されているHTTPのAPIとJenkins/Rubyを組み合わせて専用のモニタリングシートを日次で自動作成するようにしています。またリリース前後にも計測していて、もし大幅にスコアが下がった場合はすぐさま原因の分析と対策

注3) https://developers.google.com/speed/pagespeed/insights/

注4) http://www.webpagetest.org/

注5) ほかにもMobile-Friendlyテストというツールも公開していて、これと併用することでモバイルとしてのお約束を守っているかどうかのチェックも行うことができます。https://www.google.com/webmasters/tools/mobile-friendly/

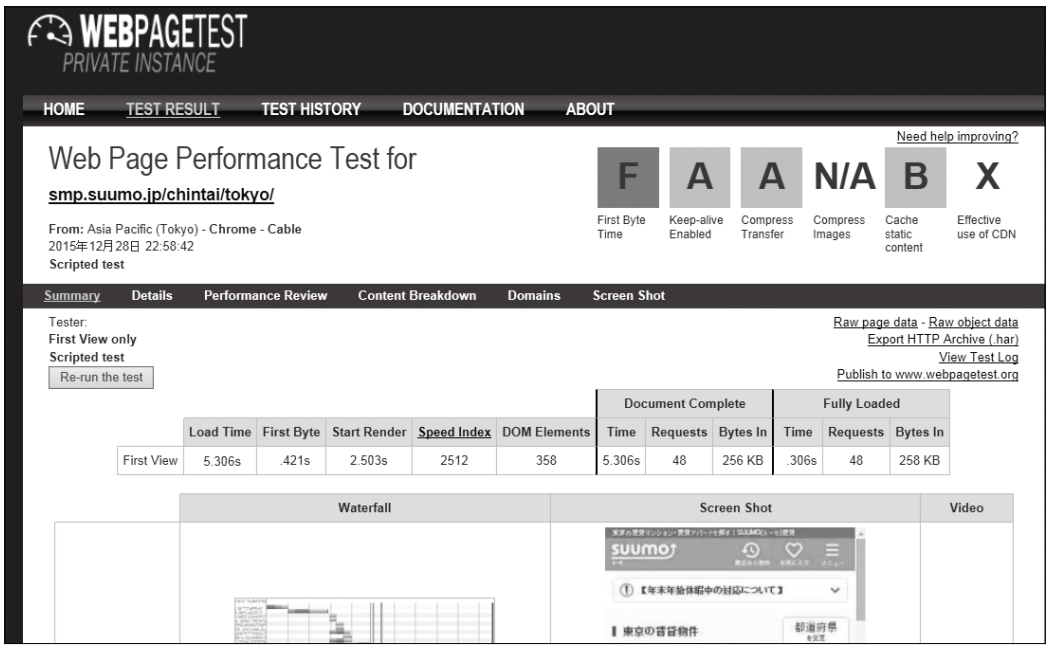
▼図13　パフォーマンスを定量化し継続的にモニタリングできるようにするシステムの全体像。手動で運用を続けることが難しいため一部を自動化している



を実施するようにしています。

ただこのPageSpeed Insightは万能ではなく、このツールだけではわからない指標もあります。それらをカバーする形でWebPageTestと呼ばれるパフォーマンステスターを併用しています。**図14**が実際にSUUMOのスマホサイトの

▼図14　WebPageTestの結果サンプル：サーバ処理時間やDocument CompleteといったTimingAPIから得られる各種タイミングまで詳細にわかる



column　Googleは“モバイルページは1秒以下で描画すべき”と定義している

サーバサイドの処理時間とクライアントサイドでの処理時間（描画など）も合わせて、何秒以内で描画したほうがいいのかみなさん考えたことはありますでしょうか。

実はGoogleの公式ドキュメントの中では“スクロールせずに見える（Above The Fold）コンテンツを1秒未満で配信しレンダリングする”と書いてあります[注A]。

これがどのぐらい速いのかぱっと感じ取ることは難しいと思いますが、スマホのWebページでチェックボックスを押して反応があるまでの極小時間（約300ms）内でサーバとクライアントサイドの処理を全部やれ、と言い換えると少しわかりやすくなるでしょうか。

そもそもすべてを1秒以内に収めようとすると、回線（3G）とDNSの正引き+HTTPのリクエスト／レスポンスの通信だけの時間で600ms前後はかかると言われているので、残りの300～400ms内でサーバ内処理とJavaScriptや描画処理を実行する必要があります。そしてスマホのブラウザでは一般的にダブルタップ判定のために、最初のタップから次のタップまで300ms待つような実装があるのですが、300～400msというのはそのぐらいの僅かな時間なのです。たしかに、もしその極小時間内にすべてを終わらせられるのであればユーザはストレスのないWebブラウジングが実現できそうです。

言うは易く行うは難しですが、SUUMOスマホサイトでも1秒以内に描画できるよう、日々改善を続けています。

注A）https://developers.google.com/speed/docs/insights/mobile?hl=ja

TOPページを解析した結果です。たとえば総読み込み時間(Document Complete)やサーバサイドの処理時間であるFirst Byteといった具体的な時間が確認できます。単体でも十分使えるのですが、画面同士の比較や日々に渡っての記録などの機能がないため、こちらも専用のツールを用意して計測を行っています(**図13**)。

サーバサイドのボトルネックを可視化する

スマホサイトではさらにサーバサイドの処理時間の改善に力を入れていて、Application Peformance Mangement(APM)と呼ばれる類の一種のプロファイリングツール(AppDynamics)を活用しています。**図15**にあるようにWebアプリケーションからのHTTPの呼び出しについて呼び出し回数／レスポンス平均を

▼図15　Webアプリケーションから呼び出すAPIのパフォーマンスを自動的に計測している(AppDynamics)

Application Flow - Grid View　　Explain this View

Type	Summary	Respo... Time (ms)	Calls	Calls / min	Errors	Errors / min
Tier Stats	smp.suumo.jp	200	258	351	-	3
Exit Call Stats	smp.suumo.jp →	71	1,251	83	-	-
Exit Call Stats	smp.suumo.jp →	25	2,252	150	-	1
Exit Call Stats	smp.suumo.jp →	97	31	2	-	-
Exit Call Stats	smp.suumo.jp →	186	56	4	-	-
Exit Call Stats	smp.suumo.jp →	20	62	4	-	-
Exit Call Stats	smp.suumo.jp →	115	18	1	-	-
Exit Call Stats	smp.suumo.jp →	163	65	4	-	-
Exit Call Stats	smp.suumo.jp →	108	708	47	14	1
Exit Call Stats	smp.suumo.jp →	77	3	0	-	-
Exit Call Stats	smp.suumo.jp →	90	24	35	-	-
Exit Call Stats	smp.suumo.jp →	26	3,128	209	-	-
Exit Call Stats	smp.suumo.jp →	2	332	42	0	0
Exit Call Stats	smp.suumo.jp →	125	2,488	166	3	0

▼図16　1,500msかかった処理のコールスタックと時間。外部APIの呼び出しが1,400msかかっていてアプリケーション自体の問題ではないことがわかる(AppDynamics)

自動的に記録したり、さらにコールスタックと実際に処理にかかった時間といったデバッギングツールでしか確認できないようなことをほぼ全自動的に計測してくれます(**図16**)。

APMのツールの大半が安くはない値段ではありますが、もし手動で調査するとなると、かなり大変なことを自動化できるため、我々にとっては費用以上に見合うメリットがありました。

組織目標と今やっていることを見える化する

突然ですが、プログラムをリファクタリングをすることで、どのように売り上げにつながっていくか論理的に説明することはできますか？　……おそらく、ぱっと答えられる方は少ないと思われますが、実はこのつながりを知ることは大切です。

たとえばWebサイトとしてカスタマーに本当に求められているのは使い勝手の向上だったのに、とりあえず設計が汚いから「リファクタリングをして保守性を改善した」結果、「売上が下がってしまった」みたいな本末転倒なことがあり得るということです。

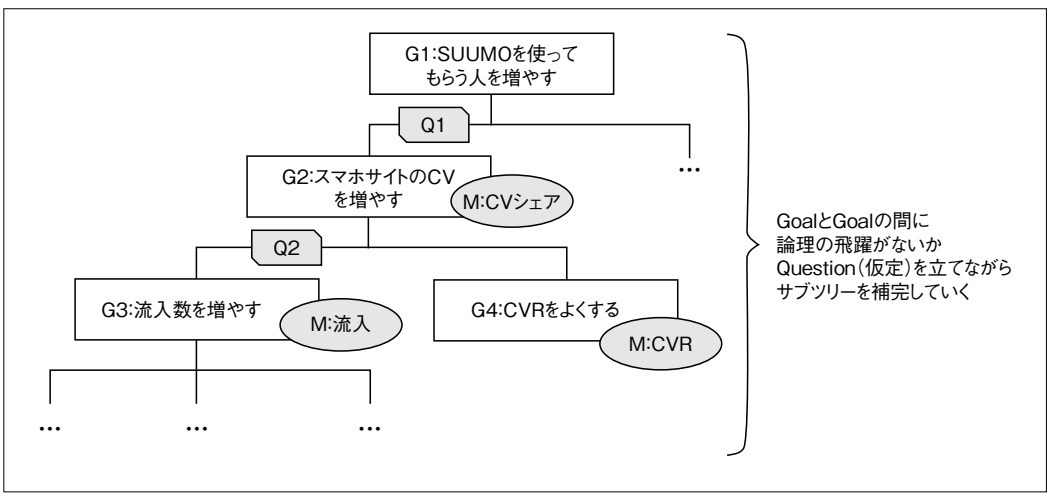
こうならないためにも、組織目標(Goal)と今やっていることが本当につながっているのかという整合性を保つために目標の整理が必要になってきます。

この整理を行う考え方の1つとしてGQM[注6]/GQM+Strategy[注7]というフレームワークがあるのですが、これを参考にチームのメンバ内で目標の整理を実施しました。**図17**が実際に整理したときのGQMツリーを簡略化したもので、最上位に住まいカンパニーの経営理念を置き、そこからどんどん仮定や事実をもとに掘り下げていくような形になっています。

このような目標の見える化をすることでよかったと思えることとして、チームの目標や役割を明確にできるため、取り組み前よりも目標達成までのストーリーを描きやすくなりました。本節の冒頭で述べた「リファクタリングをするとどう売り上げにつながるのか」という問いにドキッとした開発リーダーの方は、ぜひこのような目標整理をお勧めします。SD

注6) Goal, Question, Metric。
注7) GQM+StrategiesはIESEが保有する国際登録商標です。

▼図17　GQMの考え方をベースに独自に目標を可視化してみた結果(一部)。G1の大目標は住まいカンパニーの経営理念である「住まいを中心とした暮らしの進化を追求し、幸せな個人や家族をもっと増やす」から来ている



第3章
第1部
SUUMO流アジャイル開発［自動化編］

クリエイティブな作業時間を自動化で増やそう

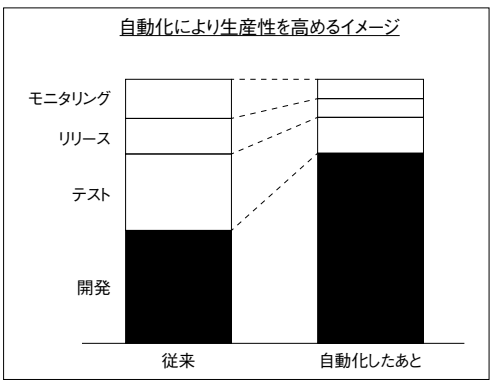
Author 吉田 拓真（よしだ たくま）
（株）リクルート住まいカンパニー
Mail t_yoshida@r.recruit.co.jp
Facebook https://www.facebook.com/takuma.yoshida.355

“自動化”とひとえに言っても、リリースやテストの自動化などいろいろな自動化があります。本章ではSUUMOのスマホサイトで取り組んでいる自動化の具体事例を紹介していきます。読者のみなさんに広く役立ちそうな、ちょっとした自動化からより実務的な自動化まで、広く紹介したいと思います。

自動化して実現したいこと

読者のみなさんは、自動化をして何を実現したいと考えていますでしょうか。我々が自動化を行う目的ですが、これは開発者の単位時間当たりの生産性を上げるために行っています。自動化して生産性を上げるイメージは**図1**になります。まず自動化することで、モニタリングやリリースにかかっている時間を削減できます。すると開発者は**削減した時間を開発に充てることで、結果的に同じ時間でより多くの成果を生み出しやすい**状況になります。「生産性＝成果物÷時間」ですから、自動化をするということは生産性を上げることにつながるというのが基本としている考えです。

▼図1　自動化を推進して開発というクリエイティブな作業を増やすイメージ



したがって自動化のポイントですが、“リリース”といった局所的な作業を集中して自動化するのではなく、モニタリングやテストといった作業についてもまんべんなく実施することが肝要です。以降はモニタリング、リリース、テストというそれぞれのトピックに分けて説明していきます。

モニタリング編（ChatOps/Kibana）

ChatOps＋リマインダー

まず、自動化できそうな定常業務とはどんなものがあるでしょうか。たとえばサーバリソースのモニタリングや、KPIのモニタリングなどがありますが、そもそもそういった定常業務を管理すること自体がけっこうたいへんではないでしょうか。せっかくなら、定常業務自体を管理してくれるようなしくみを実現したほうがよいだろうと考え、チャット上で使えるリマインダーを開発しました。

図2が実際にHubot（on Hipchat）にリマインダー機能を実装し、定常業務を通知している様子です。これは、朝にサーバリソースのモニタリングを促している様子で、メンション付きで通知しているのがわかります（毎日担当

は変わります）。

このリマインダーは**リスト1**のような仕様でHubot + cronモジュールを用いて独自に実装しています。ここには記述していませんが、cronの書式でリマインドを登録することもできます。

リマインダーはモニタリングや会議通知などといった定常業務の通知だけを実施しているわけではありません。ちょっとしたライフハックとして**図3**にあるように定時に早く帰るように促すような通知を行わせていたりします。

実際に私個人の話ですがこの通知を見て帰らなくてはと気づかされる場面があるので、よりよいワークライフバランスの実現に向けてChatOpsを活用してもいいかもしれません。ChatOpsをもっと知りたい方は弊社テックブログをご覧ください。

サーバリソースのモニタリング

能書きが長くなりましたが実際のモニタリング作業の自動化について説明していきます。上記のサーバリソースのモニタリング作業ですが、作業者がもう一段簡単になるように工夫しています。ボットに@bot graph 3dayというようなコマンドを打つことで、サーバリソースを確認するための専用のHTML画面（パーマリンク付き）を作成してくれます（**図4**）。

この画面ではDBサーバやWebサーバ、ロードバランサに付随するすべてのリソース状況を確認でき、また最大で過去1年（@bot graph 1year）まで遡（さかのぼ）ることもできます。このしくみは単純で、ボットがZabbixのHTTP APIを用いて各グラフデータ（画像）をダウンロードし、それを用いてHTMLを作成するという流れで実現しています。

それと、実はこのモニタリングは以前は一部のチームメンバーでしか行われていなかった作業だったのですが、ChatOpsに組み込んだことで、今ではチームメンバー全員でローテーションして実施できるようになりました。

ログモニタリングツール

ここでいうログモニタリングというのは、おもにサーバサイドのアクセスログに関するモニタリングを指しています。単純にログのモニタリングといってもいろいろな悩みがありました。大量のデータ（1日で100GB以上）で転送するだけでも手間がかかったり、物件が頻繁に入れ替わったりするため、404エラーが恒常的に発生し単純な集計だけでは不十分、といったような悩みです。

そこでログのモニタリングツールが必要になってくるのですが、結果としてKibana 4 +

▼リスト1　リマインダーの仕様～ChatOps～

```
リマインドをセットする（日付）
/remind set <曜日> <時刻> <メッセージ>
    <曜日>
        - 毎日              : daily
        - 平日のみ（月-金）: weekday
        - 曜日指定          : ア
mon|tue|wed|thu|fri|sat|sun
    <時刻>
        24時間表記（00:00 - 23:59）
    <メッセージ>
        出力するメッセージ
```

▼図3　ChatOps：残業をしないように気遣ってくれるスーモbot（スーモがしゃべっているわけではありません。あくまでボットです）

▼図2　ChatOps：スーモbotが定常業務をお知らせしてくれる

Elasticsearch＋fluentd＋AWS Kinesisというソフトウェアスタックで構築しました[注1]。実際に**図5**にKibanaでモニタリングしている様子を示します。秒単位レベルでの分解能を持ち、すべてのサーバのすべてのログにある、すべての項目について集計できます。

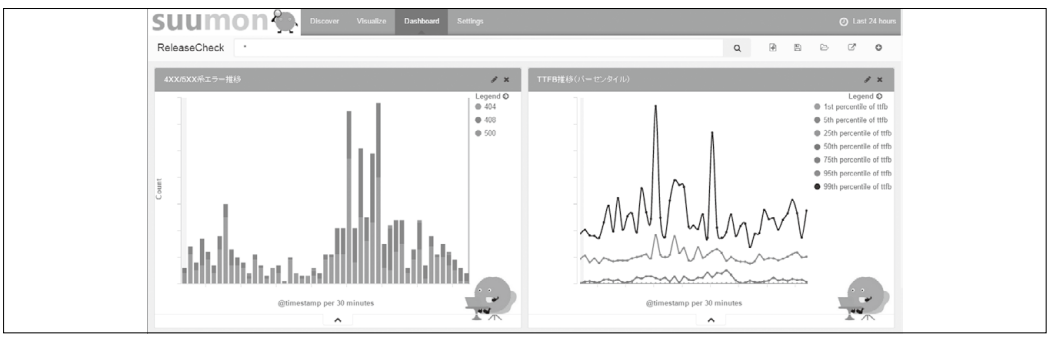
またElasticsearchではdynamic scriptingというしくみが用意されており、Groovyなどの言語を用いて検索結果をリアルタイムに加工・集計できます(**図6**)。

実際に詳細ページを領域ごとに集計するスクリプトを**リスト2**に示します。このスクリプトで処理された結果を、404や500というステー

注1) 余談ですが、このツールはすべて新人1人で作ってもらいました。お願いするときに「即応性があって、ちょっとした複雑な集計もできて、メンテナンスもしやすく、かつランニングコストを最小に抑えつつ作って！」という無茶振りでしたが、見事にこなしてくれました。

▼図4　ChatOps：サーバリソースの確認用ダッシュボードを作成&表示してくれる(上からELB/RDS/EC2を示している)

▼図5　suumon：Kibana＋fluentd＋ElasticSearchを用いてサーバのリアルタイムログモニタリングを実施している様子。ここではエラー系の時間別統計とサーバでの処理時間推移を示している



タスコードの条件フィルタを組み合わせることで領域ごとのエラー集計を実現できるという寸法です。

リリース編（ChatOps＋AWS）

サービスのリリースについて

まずリリース作業をどのぐらい自動化しているかを説明するために、実際のリリース手順をもとにChatOpsのコマンドと対比しつつ説明していきます。スマホサイトでのリリースは次のような流れで行っています。

①本番の代替機（コールドスタンバイ）を起動
②代替機にリリース
③ステージング環境で確認＆取り外し
④本番機と入れ替え
⑤後始末

これをChatOpsでのコマンドと合わせると**図7**のようになります。なお実際のリリースの雰囲気は**図8**に示します。

見ていただけるとわかるとおり、基本方針としてはフルマネージド（例：変更をコミットしたら自動的に本番反映する）ではなく、既存の手動での手順をできる限り自動化するという思想で自動化をしています。完全な自動化ではないとはいえ、AWSのEC2操作を含めて数行のコマンドでリリースできるため、リリース時間は従来と比べてかなり効率化されまし

▼図6　ダイナミックスクリプトを用いた集計の一例。賃貸や新築一戸建などといった領域単位で集計することも可能

領域別404エラー

Top 6 first_line	Count
chintai	49
ikkodate	1
chukomansion	0
chukoikkodate	0
mansion	0
tochi	0

▼リスト2　詳細ページを領域ごとに集計するスクリプト（Groovy＋Kibana）

```
import java.net.*

def m = (_value =~ /GET ¥/(¥w+)¥/¥w+¥/sc_¥d+¥/.+/)
if (m.size() > 0) {
  def ryoiki = m[0][1]
  if (ryoiki.find(/mansion|chintai|chukomansion|ikkodate|chukoikkodate|tochi/)) {
    return ryoiki
  }
}
```

▼図7　リリース作業のコマンド一例

```
//1.本番の代替機（コールドスタンバイ）を起動
@bot ec2 start server1 server2 ... serverXX

//2.代替機にリリース
//キャッシュのクリアや静的リソースのminifyなどリリース時の細かい操作が実施される
@bot release server1 server2 ... serverXX

//3.ステージング環境確認＆取り外し
//コールドスタンバイを本番相当のELBに組み込み最終チェックを行う
@bot elb replace staging server1 server2 ... serverXX
@bot elb remove staging

//4.本番機と入れ替え
//一種のBlue-Greenデプロイメント。ELB内のEC2を総入れ替えする
@bot elb replace production server1 server2 ... serverXX
```

た(とくにモジュールをリリースするだけならば1分以内です)。またそれぞれの操作がアトミックであるため、リリース手順が多少組み替わったとしても(例:前日にステージングまでリリースして当日は入れ替え作業だけ行いたい)柔軟に対応しやすいというメリットもあります。

ちなみに今の自動化のしくみはいきなりすべてを実現できたわけではなく、少しずつ継続的に改善を続けた結果です。たとえば最初は@bot releaseしかできず、EC2の起動やELBへの編入などはすべて手動でした。このように少しずつ自動化の範囲を広げていくことで、いつの間にか必要十分な自動化になっていた、というアプローチで自動化していく方法もありますので、ご自身が自動化を推進するときの選択肢の1つとして参考になればと思います。

検品・開発環境へのリリース

本番のリリースだけでなく、開発環境や検品環境といったテスト環境にもChatOpsでリリースを行っています。こちらはgitコマンドライクなインターフェースを用意していて、@bot checkout feature/hogeというようなコマンドを打つだけで適用することができます。

従来はQAやプロダクトオーナーが必要になったタイミングで開発者にリリースを依頼していたのですが、いまでは誰でもほぼコスト0でリリースすることができるようになっています。

ちなみにスマホサイトでは開発/検品環境が合わせて10環境以上(開発/検品の2種類×必要になる数)あります[注2]。どの環境に何が適用されているのか把握するのが困難なため、**図9**にあるようなChrome Extention(通称向き先さま～る)を作成し、すぐに把握できるようなしくみを実現していたりします。

品質保証編(自動テスト)

なぜ画面テスト(E2E)にしたのか

スマホサイトの自動テストの目的は、QAのテスト工数の削減・効率化だけでなく、中長

注2) なぜ10環境以上もあるかと言えば、スマホサイトでは同時並行で開発する案件が多いからです。開発規模が大きいため、どうしても並列開発を行う必要があります。中には数ヵ月に渡るような案件もあるため、複数の検品環境が必要になってきます。

▼図8 ChatOpsを用いたEC2の起動～サービスインまでの様子(@bot release)

期的なサイト品質の向上も狙っています。そして自動テストは単体テストではなく、画面のテスト(End to End；E2E)をメインとして実施しています。これには理由があり、本来ならば費用対効果が高いと言われる単体テストから実施したかったのですが、既存のコード自体がテスタビリティ(testability)が低く、単体テストを書くために膨大な改修コストがかかってしまうことが判明したため**図10**にあるようにまずは画面のテスト(GUIテスト)から着手することにしました。

資料請求の全自動テスト

まずE2Eテストをするにあたり、どのテストを自動化するのか決める必要があります。そこでサービスとしてのコアバリューを担保するためのテストから着手すべきと考え、各画面の**資料請求ボタンがきちんと動作するかどうか**の確認を自動化することとしました。

具体的なE2Eの実装方法ですが、Jenkins＋Gradle＋Geb＋Spock＋PhantomJSといったソフトウェア群を用いてGroovyで記述しています(**リスト3**)。実際に用いているテストコードを読みやすくしたものを**リスト4**に示します。

これらのテストコードですが保守性を確保するためにいくつか工夫が施されています。まず一番大きなポイントとして、テストコードの記述にPageObjectデザインパターン[注3]を適用している点です。

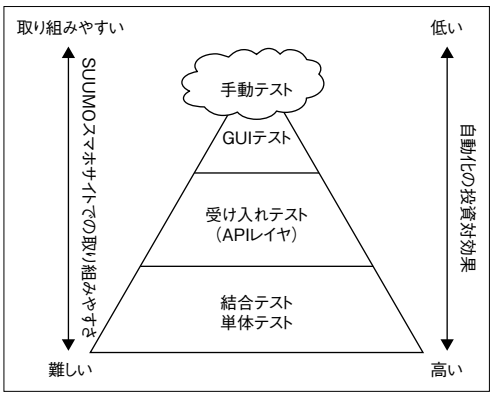
テストコードを見ていただくとわかるとおり、CSSのセレクタなどページに依存する要素がすべてPageObjectに隠蔽されており、テストコードはテストの内容のみが記述されています。こうすることで、ページの体裁が変わったとしてもPageObjectを修正するだけでテストコードに手を入れる必要がなくなるため、保守しやすいというメリットがあります。

またテストコード自体も非常にシンプルに見えるかと思いますが、こちらはGeb[注4]を用いてjQueryライクな記法と独自DSLを利用しているからこそ実現できています。自動テス

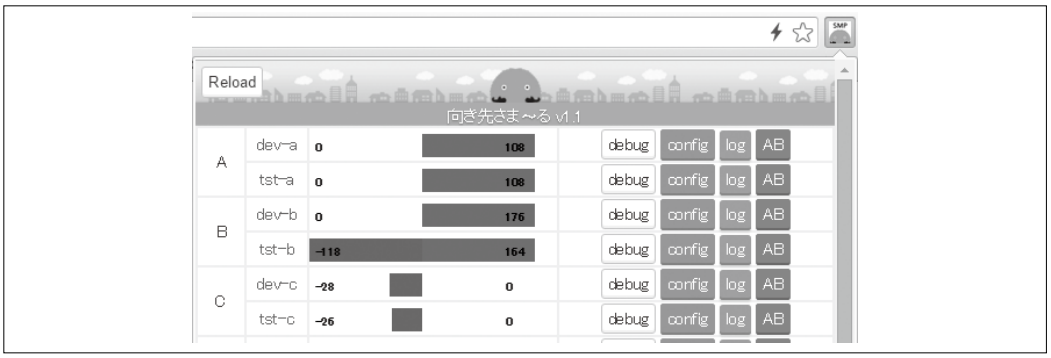
注3) ここでは詳しく解説しませんが、Seleniumのドキュメントがわかりやすいのでこちらを見てください。https://code.google.com/p/selenium/wiki/PageObjects

注4) http://www.gebish.org/

▼図10　自動テストの種類・費用対効果のヒエラルキー



▼図9　すべての開発／検品環境のモジュール適用状況を一覧化してくれるChrome Extention。図中のバーは本番(master)を基準として、左側がbehind、右側がaheadのコミット数を示している



トというとけっこう難しそうなイメージがある方も多いと思いますが、JavaScriptを書くような感覚でテストコードが書けます。

なお実際にJenkins上で自動テストを実行した結果が**図11**です。ここでは失敗したテストと、具体的になぜ落ちたのかを確認できます。

一般的にE2Eのレポーティングというと貧弱になりがちなのですが、Geb+spockを用いることで特別なことをせずとも**図12**のように期待値(expect)と実際の値(actual)が表示されます。

URLの自動チェック

またE2Eだけでなく、全画面が少なくともステータスコードが200で返却されることを確認するようなテストツールも運用しています。

▼リスト3　自動テストのテストコードサンプル(PageObject)

```
// ★ポイント1. PageObjectにページ依存の要素をまとめることで
// 画面体裁が変わってもPageObjectを修正するだけでよい
class 一覧ページ extends Page {
    static url = "/chintai/"
    static content = {
        // ★ポイント2. jQueryライクな記法ができ、可読性が高い
        物件カセット大枠 { $('#bukkenListAll') }
        一覧ページ数 { 物件カセット大枠.find('ul.toulist') }
        固定バー { $('#js-popupBtn') }

        物件のチェックボックスをタップ { n ->
            def 物件カセット = $('ul.toulist li', n-1)
            def チェックボックス = 物件カセット.find('input[type="checkbox"]')
            def checked = チェックボックス.attr('checked')
            チェックボックス.click();
            waitFor {
                checked != チェックボックス.attr('checked')
            }
        }
}
```

▼リスト4　自動テストのテストコードサンプル(実際にほぼこのままで動きます)

```
// ★ポイント1: このテストコードにはパーツのセレクタなどの要素がいっさい存在しない
// つまりページの体裁が変わってもこのテストコードは影響を受けにくい
def "賃貸エリア導線：一覧画面を直接開き、更読みをせずに資料請求を行う"() {
    setup:
    ブラウザの初期化()
    when:
    to 一覧ページ, 'tokyo', 'sc_101'
    then:
    物件カセット大枠 != null
    一覧ページ数.size() >  0

    when:
    interact { 物件のチェックボックスをタップ(1) }

    then:
    固定バーが表示されていること()

    when:
    資料請求ボタンを押す()

    then:
    at(資料請求画面)
}
```

実際のテスト結果を**図13**に示します。このツールはルーティングの設定ミスや意図しないデグレードを包括的にチェックすることを目的としています。

ツール自体はPHPで書かれていて、基本的にはURL一覧を逐次的にcurlし、結果をHTMLにまとめるというシンプルなしくみになっています。このURLチェッカーとE2Eテストを用いることで、最低限の安全ラインを確保できるため、テスト工数の削減だけでなくリファクタリングを含めた開発作業がしやすくなるという副次的な効果も得ることができました。

第1部のまとめ

第1部ではSUUMOでのアジャイル開発のイロハを紹介しました。ここで紹介した取り組みは1年とちょっとかかりましたが、前向きに捉えれば、1年あれば、今回説明したさまざまな取り組みが実現できるのではと思います。

なお疑問に思ったところや講演依頼などはできる限りお応えしますので、お気軽にご連絡ください。またここで紹介しきれない内容は住まいカンパニーのテックブログ[注5]でも紹介しているのでぜひご覧ください。SD

注5） https://tech.recruit-sumai.co.jp/

▼図11　E2Eテストのレポーティング結果（by Spock）。ここではSUUMOのサイトのCVである資料請求テストの全領域分のテスト結果を示している

▼図12　テストが失敗したときに詳細なアサーションが表示される（spock）

```
$('#pagetop header h1', 0).text() == 'お問い合わせ'
|                  |  |
|                  |  false
|                  地図検索で調布市の賃貸マンション・賃貸アパートを探す | SUUMO(スーモ)賃貸
[[[PhantomJSDriver: phantomjs on LINUX (2e425320-76cd-11e5-a2e6-057d523b1ab2)] -> css selector: #pagetop header h1]]
```

▼図13　リリース前にURLの全網羅チェッカーを運用している様子。ルーティングのミスや意図しないデグレードチェックに活用している

access report

	http_code	starttransfer_time	total_time	size_download
http://smp.suumo.jp/	401	0.126529 sec	0.15679 sec	381 bytes
http://smp.suumo.jp/edit/privacypolicy/	200	0.022252 sec	0.201927 sec	23,547 bytes
http://smp.suumo.jp/hokkaido/	401	0.02172 sec	0.050428 sec	381 bytes
http://smp.suumo.jp/aomori/	401	0.017186 sec	0.041409 sec	381 bytes
http://smp.suumo.jp/iwate/	401	0.018263 sec	0.050586 sec	381 bytes

第2部

Rettyがアプリ API の品質向上で考えた

開発言語／ツールの選定とテストを重視する工夫

Author 石田 憲幸（いしだ のりゆき）
Retty（株）
Mail noripi@retty.me

月間利用者数1000万人を突破し、成長を続けるグルメサイト『Retty』。急成長を支えるサービス開発がどのようにして行われているのか。その片鱗を、内部API開発のプロジェクトを通じて見てみましょう。

はじめに

筆者の勤務するRettyはWebとアプリで展開しているグルメサービスで、Webとアプリが利用する内部API(以下、アプリAPI)をPHPで記述しています。今まで、サービスの拡大に応じてPHPのコードに改修を加えてきましたが、ここ最近、アプリAPIにおけるメンテナンス面、パフォーマンス面などの問題が顕在化してきました。また、アプリAPIでは、データこそWebと同じものを使っていますが、ロジック面ではあまり関連性のないものも多くあり、Webから切り離したいという声もあって、まったく別のしくみとしてリニューアルすることになりました。

本稿では、筆者がかかわったアプリAPIのリニューアルに際して、どのような考えでツールや言語を選定し、また、どのように開発を進めていったか、その取り組みをご紹介したいと思います。

Rettyの開発体制

アプリAPIのリニューアルの話題に入る前に、Rettyにおける開発の体制について少しだけ触れたいと思います。

Rettyの開発チーム

Rettyの開発チームは、エンジニア、デザイナといった、業種ベースのチーム編成ではなく、アプリやSEOといった、プロジェクトベースのチーム編成になっています。各チームにはチーム内のミッションがあり、たとえば、アプリチームであればアプリ利用継続率○%、SEOチームであれば月間ユニークユーザ数○万、といった定量的なチーム目標をそれぞれ設定して、それを達成するために日々の開発案件をこなしています。

チームのメンバー構成はそのチームによって異なりますが、おおむね次のメンバーで構成されています。

- プランナー：開発案件の仕様を策定し、また案件の進捗全体を管理する
- デザイナ：画面の動きや要素の配置などのデザインを設計する
- アプリケーションエンジニア：コードを書いてサービスを改修する

各開発案件は多くの場合、プランナー、デザイナ、アプリケーションエンジニアが各1人割り当てられます。

また、インフラチームのように、ほかのプロジェクトとは独立した横串体制のチームもあります。インフラチームは日々の開発案件が滞りなく進むよう、また開発案件の不具合

に起因してサービスに致命的な影響が出ないよう、日々Rettyのサービスを守っています。

なお、開発チームがプロジェクトベースの編成になっているため、チームの編成は全社的な目標に応じて柔軟に変化します。アプリAPIリニューアルのチームも、全社的な目標の変化に応じて生まれたチームの1つです。

開発チームのメンバー構成

チーム内におけるプランナー、デザイナ、エンジニアの構成比は、そのチームが開発するプロダクトの性質に応じて設定されています。

たとえばアプリチームの場合、企画段階で時間を要する案件が比較的多いことや、大人数で開発するとコンフリクトが発生しやすい側面から、エンジニアに比べてプランナーが多めなメンバー構成となっています。一方SEOチームでは、素早く企画を立てて、その案件をなるべく早く進める必要があることから、プランナーとエンジニアが同数程度で、規模が大きめのチームになっています。ただし、こちらも全社的な目標に応じて柔軟に変化します。

どのチームにもプランナー、デザイナ、アプリケーションエンジニアのすべてがいるわけではなく、そのチームの目標に必要なメンバーが必要なだけ所属しています。実際、アプリAPIリニューアルのチームは、技術的な設計に終始するプロジェクトだったため、プランナーもデザイナもおらず、エンジニア2人が所属していました。

Rettyは、PDCA[注1]を素早く回すことを常に意識していて、チームの編成も、メンバーの構成も、より良くPDCAのサイクルを回すために、その段階で最適な形にしようとしています。

注1）Plan（計画）－ Do（実行）－ Check（評価）－ Act（改善）の過程を繰り返すことによって、業務を継続的に改善する手法。

アプリAPIのリニューアル

冒頭で述べたように、アプリAPIはPHPで記述されていましたが、規模の拡大に伴っていくつかの問題が顕在化し、アプリAPIのリニューアルに至りました。ここでは、リニューアルに至った経緯、目的について詳述します。

アプリAPIで発生していた問題

メンテナンス面の問題

リニューアル以前のアプリAPIにおいて発生していた問題の1つはメンテナンス面の問題です。リニューアル以前は、アプリのバージョンによらず同じエンドポイントを利用しつつも、しばしば違うレスポンス形式を期待するものになっており、APIのロジック内にバージョン分岐が大量に埋め込まれていました。そのため分岐が多く、ロジックが非常に複雑なものになっていました。テストの際にも、各バージョンごとにテストをする必要があったため、テストの工数が膨大になり、PDCAサイクルの速度にも影響していました。

また、APIのレスポンスコードが適切に設定されておらず、例外処理漏れなどによりAPI側でエラーが発生していても、レスポンスコードが正常のまま、不適切な値が返ってくるしくみになっており、エラーの発見が難しい状態になっていました。

パフォーマンス面の問題

もう1つの問題はパフォーマンス面の問題です。RettyではWebとアプリAPIを同じリソースの中で構築してきましたが、Webとアプリ APIでは、利用するユーザの性質も、負荷のかかる部分も異なっており、同じリソースで開発を続けるところに無理が生じ始めていました。

また、Rettyのユーザ数は近年Web、アプリ

共に急速に伸び始めており、Webとアプリ APIの両方のパフォーマンス改善のためにも、分離して個別に運用する必要性が出てきました。

リニューアルの要件

前節で述べた問題が顕在化してきたことを受けて、RettyではアプリAPIのリニューアルをすることになりました。このリニューアルでAPIが満たそうとしていた要件をここで整理したいと思います。

シンプルな設計と高品質なプロダクト

リニューアル以前に埋め込まれたバージョンの分岐を排除し、ロジックをよりシンプルにすることを目指しました。また、テストフレームワークを導入してテストを記述することにより、高品質なプロダクトを実現することを目指しました。

エラーの発見が容易

リニューアル以前は不完全だった例外処理を適切に行うとともに、エラーがあれば発見できるしくみを作ることを目指しました。

パフォーマンス上の問題がない

APIに最適化された設計で実装することによってパフォーマンスを改善するとともに、運用時、そのパフォーマンスに問題が発生していないかを監視できる体制を作ることを目指しました。

運用の手間がかからない

APIのリニューアルによって運用の手間が上がってしまうとPDCAサイクルの速度に影響してしまうため、テストや監視の体制を作りつつも、運用の手間が上がらないようなしくみを作ることを目指しました。

内部仕様の決定

前節で述べた目的を達成するためには、言語の選定やサーバの構成など、内部仕様も重要になってきます。ここではリニューアルを進めていく中で、言語やサーバの構成をどのように決定したかについて述べたいと思います。

言語の選定

言語の選定は、コードの品質を保持することや、エラーの発見を容易にする観点から非常に重要です。結論としてはJavaを選定することにしました。以下では、APIを実装するにあたって重視したJavaの言語的な性質をいくつか見ていきたいと思います。

コンパイル言語・静的型付け言語である

言うまでもなく、Javaは静的型付け言語であるため宣言なしに変数を使うことはできず、また、宣言した変数に異なる型の変数を代入することもできません。逆に言えば、それによって利用している変数の型が明確になり、想定外の値が代入されることによる不具合を軽減することができます。

また、コンパイル言語であり、例外処理も厳格なため、多くの例外はコンパイル時に発見できます。実行時例外はInternal Server Errorとして処理されるようにすることができます。

Webサービス、APIの実装に適した仕様がある

JavaにはサーバサイドでデータAPIを行うためのJava Servletという技術があり、この上でAPIなどのサーバサイド処理を行うことができます。また、JAX-RS(Java API for RESTful Web Services)という、RESTに基づくWebサービスを開発するためのAPIがJava EE 6以降で標準搭載されており、アノテー

ションを使って受け入れるHTTPメソッドや返り値の形式をシンプルに記述することができます(**図1**)。

言語レベルでテストフレームワークがサポートされている

テストコードを書くことにより、万能ではないながらも、想定外の処理や例外を防ぐ意味で一定の効果があります。JavaにはJUnitという実績のあるテストフレームワークがあり、多くのIDEやビルドツールではデフォルトでJUnitをサポートしているので、スムーズにテストフレームワークを導入できます。

サーバの構成

シンプルな設計をするにあたって、言語だけでなく、どのようなミドルウェアを導入するか、また、サーバの構成をどのようにするかを考慮する必要があります。

まず、Java Servletを動作させるために、サーブレットコンテナを導入する必要があります。サーブレットコンテナにはTomcatやJetty、WebLogicなどいくつかの種類がありますが、今回これらの中からJettyを採用しました。その理由として、TomcatやWebLogicに比べ起動時間が短く、また、構築手順がシンプルであった点があります。

さらに、サーバの構築を自動化し、構築手順をコード化するしくみとして、Dockerを導入しました。これによりサーバの構築手順をテストすることができます。副次的な効果として誰の開発環境でも同じ環境を作ることができ、それによって開発時の環境差分を小さくできます。

内部仕様と設計の指針

内部仕様や設計を決める場合、各要素には複数の選択肢があり、簡単には決められない場合がほとんどです。実装を始めてから、別の選択肢をとったほうが良かったと後悔することもあるかもしれません。こういった場合、満たしたかった要件まで立ち返って、設計を見直すことが最終的には有益になるケースもあります。

今回のリニューアルでも、実装がある程度進んだ時期に、データベース関連のライブラリを再度見直すかどうかの判断を迫られたことがありました。データベース関連のライブラリはプロダクトのかなり根幹であり、もし別のライブラリで再度作り直すとなれば、かなりの工数が見込まれましたが、最終的には別のライブラリで作り直す判断をしました。

この判断の軸となったのが、リニューアルにおいて満たしたかった要件の1つ、「運用の手間がかからない」でした。元のライブラリはリファレンスが少なく、しかも利用方法がかなり特殊なものでした。これではプロダクト完成後の運用のコストが上がってしまうと考えたのです。

今思えば、最初にそのライブラリを選んだ時点で気づくべき過ちですが、工数を割いてでも別のライブラリで作りなおした意味は大きいと思っています。

▼図1　JAX-RSの仕様のもとで記述されたメソッド

```
/**
 * “hello ${名前}” という文字列を返すAPI
 */
@Path(“/user/{user_name}”)        // /user/*** へのリクエストを受け付ける
@GET                              // GETリクエストを受け付ける
@Produces(MediaType.TEXT_PLAIN)   // text/plain形式でレスポンスを返す
public String hello(@PathParam("user_name") String userName) {
    return “hello ” + userName;
}
```

テスト手法

今回のリニューアルで重視した点の1つがコードの品質です。リニューアル以前はテストのしくみが導入されていなかったため、想定どおり動いているかを確かめるために、毎回すべての分岐を手動で確認する必要がありました。そのためテストの漏れも発生しやすい状況でした。これらの問題を改善するため、リニューアル後のAPI開発ではテストのしくみを導入することで、単体テストを自動化するとともにテストの漏れも削減し、プロダクトの品質の担保を目指しました。

テストのしくみは有用ですが、ただ導入するだけでは続かないことも経験上何度かあったため、どうやって運用していくか、どうやって品質を担保し続けるかを意識して導入を進めました。

ブラックボックステストとホワイトボックステスト

テストの書き方には、大きく分けてブラックボックステストとホワイトボックステストがあり、今回の開発ではその両方を用いています。

ブラックボックステスト

ブラックボックステストは、実装されているロジックがどうなっているかを考慮せずに、仕様のみから記述されるテストです。今回の開発では、コードの変更時に仕様を満たせているかどうかを確認するためのテストとして用いています。

ホワイトボックステスト

ホワイトボックステストは、実装されているロジックの各経路が意図どおり動作するかを検証するために記述されるテストです。APIのロジックには、Null Pointer Exceptionなど、コンパイル時にチェックされない実行時例外がしばしば含まれるので、ホワイトボックステストを用いて事前に検知しようとしています。

ビルドツールと自動テスト

テストのしくみを導入しても、エンジニア内である程度テストを書く文化が根付いていないと「テストを記述する分、工数が増大する」というイメージが先行して、結局書かなくなってしまいます。残念なことに、Rettyではまだまだテストを書く文化が根付いていなかったため、ある程度強制力を持ってテストを書いてもらう必要がありました。

ビルドツールの導入

リニューアル後のAPIではソースコードから最終的にサーブレットとなるwarファイルを生成しますが、そのビルドの手順を毎回手作業で実行するのは非常に面倒です。そこで、ビルドを実行してくれるツールとしてGradleを利用することにしています。うれしいことに、最近のビルドツールはデフォルトでJUnitと連携できるタスクが準備されていることが多く、また、テストが失敗するとビルドも失敗するものが多くなっています。

今回の開発では、ビルドの前提となるテストとして、コードカバレッジ[注2]を計測するタスクを追加しています。コードカバレッジ計測のためのライブラリは、Gradleと連携できるJacoco[注3]を利用し、この計測結果が一定以下であればテストが失敗するようにしました。これにより一定量以上のテストコードが記述されることを保証することができました。

自動テスト

ビルドツールの導入は、テストコード記述に対する強制力として一定の効果がありますが、

注2) コード全体のなかで、テストが行われた部分の占める割合のこと。

注3) http://eclemma.org/jacoco/

不慮の事故によってテストされないままレポジトリにPUSHしようとしたときに備え、自動テストのしくみを導入しました。

自動テストによく使われるツールとしてJenkinsが挙げられますが、これを利用する場合、自身でJenkinsが動作する環境を構築し、その環境をメンテナンスする必要が出てきます。リニューアルの要件の1つとして挙げた“運用の手間がかからないようにする”こととの両立が悩みどころでした。そこで、自身で運用する必要がなく、かつある程度カスタマイズができるSaaSとして、CircleCI[注4]を利用することにしました。CircleCIを使うと、GitHubと連携して、レポジトリへのPUSHをトリガにして自動的にテストすることができます。

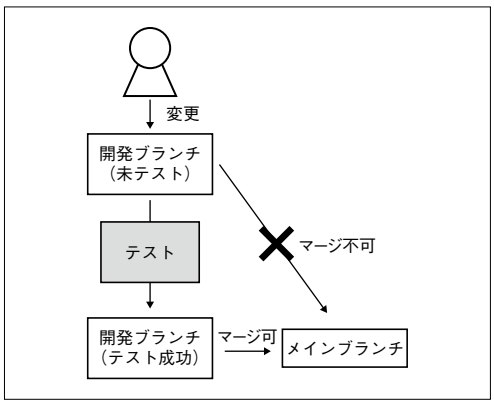
これに加えて、まだテストの通っていない開発ブランチ、テストが失敗したブランチはメインストリームのブランチへマージしてはならず、テストが通った後のブランチのみをマージして良いという運用フローを設定しました(**図2**)。これにより、メインストリームのブランチのコードが常にテストされていることを保証することができました。

自動デプロイ

今回のリニューアルで重視した別の項目として、運用の手間がかからないようにするというものがあります。一般的に、テストが終わってコードをメインストリームにマージした後は、そのコードをステージング環境にデプロイし、結合テストをする必要があります。

リニューアル以前のAPIは、コードに更新があった際に自動的にステージング環境にデプロイされるしくみを用意していて、そのデプロイにおいて運用コストがほぼかからないようになっていました。そこで、今回のリニューアルでも自動的にステージング環境にデプロイするしくみを導入することにしました。幸い、

▼図2　開発ブランチをマージするまで



CircleCIはRettyが利用しているAWSとの連携ができるので、❶ソースコードのビルド、テストを実行し、❷Dockerイメージのビルド、Docker Hubへの登録を行い、❸AWSへのステージング環境へのデプロイ要求を送るフローを作りました。また、AWSがデプロイ完了後にSlackに通知するようなしくみも、あわせて導入しました(**図3**)。

導入時の注意点

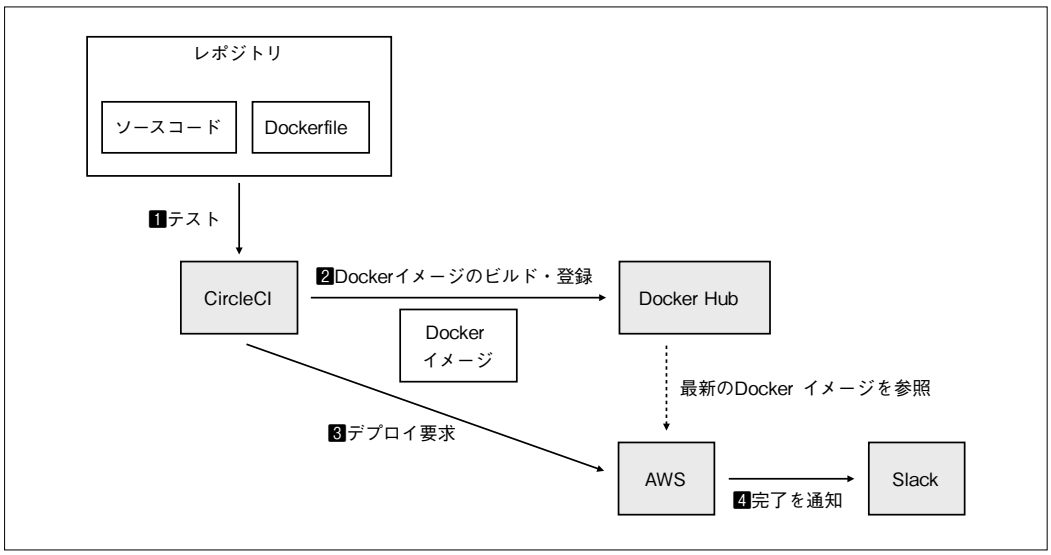
今回、テストを導入したうえで、そのテストが通らなければメインストリームにマージできないという開発フローを作りました。これにより、メインストリームのコードの品質を担保することに成功したと考えています。

しかしながら、新規開発するプロダクトにテストを導入する場合、注意して検討したいことがあります。それはホワイトボックステストのコードカバレッジです。先に述べたように、ホワイトボックステストは、コンパイル時にチェックされないNull Pointer Exceptionなどの例外チェックの意味で一定の効果があります。しかし、開発が始まって間もない頃は、そのしきい値を高く設定し過ぎないよう、注意を払う必要があります。

コードカバレッジの指標には、テストされた命令文の割合を示すステートメントカバレッジと、テストされた条件分岐の割合を示すブ

注4）https://circleci.com/

▼図3　自動デプロイのフロー



ランチカバレッジがよく使われますが、とくに初期の段階でブランチカバレッジを高く設定し過ぎると、しきい値を下回った場合にテストコードを記述することが困難になり、カバレッジをあげるために新規開発する事態になる場合もあります。実際、今回の開発ではステートメントカバレッジ90%、ブランチカバレッジ90%をしきい値としていましたが、とくにブランチカバレッジが初期の段階でネックとなるケースが頻発しました。

もちろん、常にカバレッジ100%で書ければ理想ですが、中にはファイル読み込み系の例外など、テストが非常に書きづらい、もしくは書けない条件分岐もあります。開発の初期段階では条件分岐数も少なく、テストを書きづらい条件分岐の割合が大きくなりがちで、このような事態が発生しやすくなります。

おわりに

本稿では、Rettyで実施したアプリAPIを題材として、開発フローの事例について述べました。今回の新規開発における要件はいくつかありますが、その中でもとくに重視した項目は「高品質なプロダクトを作る」と「運用の手間がかからないようにする」の2点です。

今回の開発では「高品質なプロダクトを作る」ために、静的型付け言語でかつテストフレームワークをサポートしているJavaを採用しました。また、テストフレームワークJUnitとコードカバレッジ計測ツールJacocoを導入したうえで、ブラックボックステスト、ホワイトボックステストを記述しないとメインストリームにマージできないフローを作り、コードの品質を担保することを試みました。

さらに「運用の手間がかからないようにする」ために、自動テストのためのツールとして運用が不要なCircleCIを導入し、自動デプロイのしくみを整えました。

最後になりますが、本稿で紹介した開発フローはまだまだ改善途上のもので、これからもより良いフローを作っていこうと日々取り組んでいます。今回ご紹介した事例が皆さまの開発の中で少しでもご参考になりましたら幸いです。SD

第3部

HRTと情報共有こそチーム開発の要

Qiita開発で知る、テスト、自動化、バグ／タスク管理術

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

プログラマに特化した情報共有サービス「Qiita」。エンジニアならば、一度は利用した経験があるのではないでしょうか。第3部では、その開発元「Increments」の開発事情について紹介します。どのような文化の下、どのようなチーム開発を行っているのか、人気サービスを支える技術をご覧ください。

はじめに

Incrementsは、プログラミングに関する知識を記録・共有するためのサービスとして、多くのエンジニアの方に愛用いただいているQiita[注1]を開発・運営しています。本章では、そのIncrementsが用いている開発プロセスやそれを支える文化について解説します。

QiitaとIncrementsの使命

Qiitaにアクセスしていただくと、「プログラミング知識を共有しよう。」というキャッチフレーズがトップに書かれていることに気づくでしょう。Incrementsは「ソフトウェア開発をよくすることで世界の進化を加速させる。」をミッションとして、QiitaやQiita:Teamという組織内情報共有サービスを提供しています。このキャッチフレーズやミッションステートメントからおわかりいただけるように、エンジニア、その中でもプログラマに特化したサービスを提供することで社会をより良くすることを、Incrementsは目指しています。そのために、情報共有サービスの提供に加え、自社の技術をオープンソースにすることや、開発プロセス、ノウハウなどを公開することも積極的に進めています。

筆者は昨年の11月にIncrementsに入社したばかりですが、Incrementsを支えるものは自社ツールのQiita:Teamや、チャットツールであるSlackを中心とした情報共有と、その源泉となるエンジニアリング文化にあると感じています。

一方、Qiitaはすでにアクティブユーザ数も200万人を超え、累計の投稿も10万件以上あります。このようなプログラマにとってのインフラになりつつあるサービスを支える開発プロセスは、もちろん一朝一夕で作られたものではありませんし、今も日々進化しています。まずは、この開発プロセスからお話します。

開発プロセスの全貌

Incrementsではどのようなツールを使って、どのような開発フローを実現しているのか。タスク管理やオフィス環境にも触れながら解説していきます。

ツール

Incrementsでは**表1**の開発ツールを用いています。このようにIncrementsでは誰でも利

注1）URL https://qiita.com

用できるツールを組み合わせて開発基盤を実現していますが、その弊害としてツールの吐き出す情報が分散してしまうことが挙げられます。この問題は、後述するように、コミュニケーションのハブとしてSlackとQiita:Teamを用いることで解決しています。

開発フロー

開発フローには、すでに一般的となったGitHubのPull Requestを用いた手法を用いています。ソースリポジトリとしては、オープンソース化されているものはGitHubのパブリックリポジトリを、それ以外はプライベートリポジトリを用いています。

Pull Reuestベースの開発としては極めて一般的な手法、すなわち、

①各リポジトリのmasterブランチから分岐して実装
②実装終了後にGitHubでPull Requestを投げ
③CircleCIによるCIを走らせるとともに
④ほかのエンジニアにコードレビューしてもらい
⑤コードレビューとCIが無事終了したことを確認し
⑤masterブランチにマージしてデプロイする

という、今や王道とも言える手法をそのまま用いています。

テスト

テストは、Circle CIが走る際に自動的に行われます(**リスト1**)。テストには2種類あり、1つがLinterによる文法チェック(静的コード解析)、もう1つがRSpecによる機能テストです。QiitaはRuby on Railsにより開発されており、フロントエンドではSCSSをCSSのプリプロセッサとして利用しています。文法チェックや機能テストはこれらのソースファイルに対して行われます。

Linterによる文法チェックはYAMLとSCSS、Rubyに対して行われるようになっています。SCSSにはscss-lint[注2]を、Rubyには

注2) URL https://github.com/brigade/scss-lint

▼表1 ツール一覧

ツール	概要	社内での用途
Amazon Web Services	インフラ(EC2、RDS、ElastiCache)	言わずもがな、QiitaやQiita:Teamのインフラ
GitHub	ソースリポジトリ	コードレビューやIssueを用いたタスク管理、ディスカッションを行う
CircleCI	継続的インテグレーション(CI)環境	CI基盤として用いる
Sentry	エラーの補足や通知	障害発生検知に用いる
New Relic	パフォーマンスの測定	パフォーマンス障害の検知や、改善対策用に用いる
Mackerel	サーバの管理・監視	同じく、パフォーマンスなどの監視を行う
Google Analytics	アクセスログ解析	Webアクセスログ解析としてお馴染みのツールで、基本的なアクセス解析を行う
Mixpanel	イベント計測	特定のイベントの成果を測定することなどに用いる
Google BigQuery	アクセスログ解析	fluentd経由で集約されたアクセスログの解析を行う
Optimizely	A/Bテスト用ツール	各種実験を行い、その成果を計測するのに用いる
MailChimp	メールマガジン配信	ユーザへのメール配信に用い、開封率など各種指標を管理する
Trello	一般的なタスクや仮説検証用タスクの管理	現在は個人やサブチームレベルで使うことはあるが、全社的には用いていない

Rubocop[注3]を使っています。Rubocop既定のルールが.rubocop.ymlに書かれていますが、Rubyの文法チェックはチームのコーディング規約に沿う必要があるため、Incrementsではこれに少し変更を加えて利用しています。

bundle updateの自動実行

IncrementsではGemパッケージの管理をBundlerで行っているのですが、Gemの更新を取り込むためのbundle updateをCIに組込み、定期的に実行するようにしています。

bundle updateすることで、依存しているGemのバグ修正やパフォーマンス改善、新機能などを取り入れることができます。しかし長い間更新していないと、大きな差分により、最悪の場合アプリケーションが動作しなくなってしまい、対応に大きなコストが発生します。そのため、IncrementsではこのbundleupdateをCIに組み込むことで常に最新版を使えるようにしています。

CIによるbundle updateの定期実行には、CircleCIのNightly Builds[注4]という機能を使っています。これは、外部から環境変数を設定したうえで、特定のブランチのビルドを行うという機能です。Incrementsでは、BUNDLE_UPDATEという環境変数を用意して、これがtrueのときにbundle updateが起動されるスクリプトを組み、cronで毎日実行するようにしています。**リスト2**がcircle.ymlの該当個所です。

この自動実行の流れは次の**図1**のようになります。こちらについては、弊社CTOの高橋がQiitaで記事[注5]にしていますので、参考にしてください。

バグを防ぐさまざまな工夫

QiitaもQiita:Teamも、社内でテストを行ったあとに外部リリースを行ってはいますが、それでも不具合は発生し得ます。そこで、大きなバグ修正や新機能の場合、社内の人間でしばらく使用してから外部リリースを行う、

注3）Rubocopの紹介記事を弊社CEOの海野が書いています。
URL http://qiita.com/yaotti/items/4f69a145a22f9c8f8333

▼リスト1　文法チェックと機能テスト（circle.ymlからの抜粋）

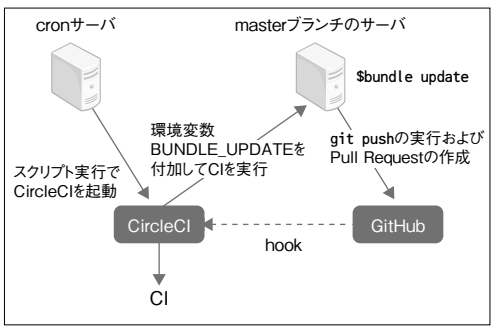
```
test:
  override:
    - bash script/circleci/lint_yaml.sh
    - bash script/circleci/lint_scss.sh:
        parallel: true
        files:
          - app/assets/stylesheets/**/*.scss
    - bash script/circleci/lint_ruby.sh:
        parallel: true
        files:
          - app/**/*.rb
          - lib/**/*.rb
          - spec/**/*.rb
    - bash script/circleci/test_rails.sh:
        parallel: true
        files:
          - spec/**/*_spec.rb
    - bash script/circleci/test_javascript.sh
```

▼リスト2　bundle update自動実行の設定（circle.ymlから抜粋）

```
test:
  post:
    - >
      if [ -n "${BUNDLE_UPDATE}" -a "${CIRCLE_BRANCH}" = 'master' ] ; then
        bundle update
      fi
```

注4）URL https://circleci.com/docs/nightly-builds
注5）URL http://qiita.com/yuku_t/items/0f8e48771822420d3ffa

▼図1　bundle updateの自動実行



いわゆるドッグフード環境を用意しています。「increments.qiita.com」はIncrementsが使っているQiita:Teamのアドレスですが、このアドレスのみ特定のサーバを示すようにDNSで切り替えたり、社内スタッフであることがわかるユーザのフラグが付与されている場合に、Gitの特定ブランチを参照するようにしたりすることで、ドッグフード環境を実現しています。今後は、これをリリースプロセスの中に組み込んで自動化することを検討しています。

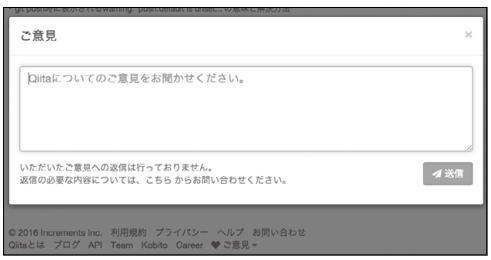
また、リリース後にもユーザからの反応をモニターし、不具合の報告などがあった場合にはすぐに対応するようにしています。TwitterでのQiitaを含むツイートはすべて社内で共有されており、不具合と思われる挙動があって追加情報が必要と思われる場合には、社員個人がQiita公式アカウント[注6]から問い合わせることもあります。また、ユーザからのフィードバックを受け付ける窓口をオンラインで開設しており(**図2**)、この窓口経由でのフィードバックも、オンラインカスタマーサポートツールであるZendesk[注7]経由で社内に共有されています。

タスク管理

開発に関係するタスク管理には、GitHubのIssueを使っています。GitHub Issueはラベルとマイルストーン、アサイン先しか項目を持たないシンプルなバグ・課題管理システムです。

注6) URL https://twitter.com/qiita
注7) URL https://www.zendesk.co.jp

▼図2 Qiitaについてのご意見



JIRAやRedmineなどが持つ優先度やカテゴリ・コンポーネント、課題の種類(バグや機能要求など)を持たないため、それらを必要とする場合はラベルやマイルストーンを活用して実現する必要があります。

Incrementsでは、ラベルで「Qiita」や「Qiita:Team」など製品を示すようにしています。一方、カテゴリやコンポーネントをラベルで管理することはしていません。バグも機能要求も複数にまたがることが多いうえ、粒度を定義することが難しく、たとえコンポーネント分けができたとしても、その依存度含めてうまく運用することに困難を感じたためです。

また、現在では優先度も用いていません。優先度(Priority)や深刻度(Severity)は定義が難しく、ふと気づくと「すべてのバグが中レベルの優先度になってしまっている」ということもありがちです。緊急性の高いバグは優先度で区別するまでもなく、すぐに社内で対策を検討できる環境になっているため、現在では優先度はとくに設けなくて良いだろうと判断しています。その結果、現在のIncrementsの運用では、ラベルにIssueの種類として「バグ(Bug)」と「機能要求(Feature Request)」を用意するだけにとどまっています。マイルストーンは、期間や締め切りを持つIssueを束ねるというごく一般的な使い方をしています。

Issueは、気づいたときに迅速に登録することを重視しています。ただし、そうは言っても必要な情報が書かれていないと、登録した本人もあとから見なおしたときに、どうして登録したか、重要度や緊急性がどの程度かもわからなくなってしまいます。そのため、必要な項目を定めたテンプレートを作成し、登録時にはそれを埋めるようにしています。GitHub IssuesはURIパラメータで項目を引き渡せるため、バグと機能要求用に2種類のURIを用意し、登録時にはそれらをクリックすることでテンプレートを埋めることができるようにしています(**図3**)。

また、GitHub Issue登録ガイドラインを社内で制定し、適宜見直しを図るようにしています(**図4**)。

製品リリースのためには、バグや機能要求以外のタスクも数多くあります。サポートの準備を行わなければいけないことや、製品のリリースに合わせて利用規約を変更するようなこともあり得ます。また、仮説検証を回すというタスクにも管理が必要です。以前はこのようなタスクのために、TrelloやPivotal Trackerを用いていました。しかし最終的に実装タスクに落ちるときに、結局GitHub Issuesに登録しなおす必要が発生していたため、現在全社的にはこれらのタスク管理ツールを用いていません。一方、GitHub Issuesはリポジトリをまたがるタスク管理が難しく、タスクのフロー管理も得意ではありません。そこで、現在はZenHubというGitHubを拡張し、カンバン方式のようなタスクボードを実現するサービスの導入を検討しているところです。

オフィス環境

最後に、少し話が脱線しますが、オフィス環境についても説明します。現在、Incrementsは小さなオフィスビルの1フロアを借り、全社員がそこで勤務しています。フリーアドレス制度を採用しているわけではないのですが、机の上にほとんど物がなく、筆者が入社した日にレイアウト変更を行ったときは、ものの30分もかからずに移動が完了しました。

モニターを所有するエンジニアは一部を除き居ないため、オフィス内でも好きな場所で開発を行います。サーバの類はいっさいありません。AWSやSaaSとして利用できるサービスを中心に採用しているため、オフィス内にサーバを置く必要がないためです。電話やファックスなどもなく、社外の方からの連絡もメールやチャットを用いて行われています。

エンジニアリングチームの文化

ソフトウェア開発は技術とツールを活用した“プロセス”に依るところが多いですが、一方でそれらを築き上げるのは“人”であり、“チーム”です。チームの意思が技術やツールを選び、プロセスを形作っていきます。Incrementsではそのような考えから、技術やツール、プロセス以上にチームの文化を大切にしています。

HRTという考え

HRTは、Googleのシカゴオフィスを立ち上げた2人のエンジニアによって書かれた「Team Geek」という書籍で紹介されている考えです。Hは Humility(謙遜)、Rは Respect(尊敬)、Tは Trust(信頼)をそれぞれ意味しています。以

▼図3　GitHub Issues登録用テンプレート

▼図4　Incrementsが使用するGitHub Issue登録ガイドライン

降、Team Geekから抜粋します。

謙虚（Humility）
世界の中心は君ではない。君は全知全能ではないし、絶対に正しいわけでもない。常に自分を改善していこう。

尊敬（Respect）
一緒に働く人のことを心から思いやろう。相手を1人の人間として扱い、その能力や功績を高く評価しよう。

信頼（Trust）
自分以外の人は有能であり、正しいことをすると信じよう。そうすれば、仕事を任せることができる。

いかがでしょう。「謙虚」「尊敬」「信頼」という単語だけで聞くよりも、この説明を見ると、もっと心に響くのではないでしょうか。コードを書くときや他人のコードをレビューするとき、方針を話し合うときなど、自分が自信のある分野ですと、ともすると自分が正しく、いかに自分の考えを他人に理解してもらうかに腐心してしまうことがあるのではないでしょうか。自分は常に間違っている可能性があると一歩引くことで、コミュニケーションが円滑になりますし、自分では気づかなかったことを発見できるかもしれません。

「ソフトウェア開発はチームスポーツである」。これもTeam Geekの中の言葉ですが、これを意識すると、個人技を極めるだけでなく、どのようにチームとして機能するかを考えないといけないことがわかります。Incrementsではこの精神を尊重し、次のような行動指針を立てています。

- オンラインでは直接口頭で話すとき以上に言葉遣いを丁寧にする
- コードレビューはあくまでコードを批判しているのであって、書き手を批判していると取られないように注意する
- 絵文字や画像を積極的に使う

3つめは、テキストではどうしても無味乾燥になりがちなコミュニケーションをより円滑に進めるためです。直接会って話す場合には、顔の表情や口調などから、伝えることは同じでも、より多くの情報が付加されます。テキストでも、可能な範囲で付加情報を加えることで誤解の生じないコミュニケーションが可能となります。

属人性の排除としての自動化

チームスポーツとしてソフトウェア開発を考えると、特定のエンジニア、すなわちプレーヤに依存するのがリスキーなことがわかります。

昔のプロ野球ならば絶対的なエースがシーズンで30勝40勝することもありましたが、今日では投手も分業制を採っており、1人への依存をできるだけ少なくしています。春や夏の限定した期間で1回でも負けたら終わりという高校野球と違い、プロスポーツは年間通して試合がありますし、1年だけでなく、翌年や翌々年も継続して勝てるチームを作らなければいけません。ソフトウェア開発チームもプレーヤである各エンジニアの成長とともに、特定個人に依存しない体制を作る必要があります。

この「属人性の排除」に対してIncrementsが行っていることとして、徹底した自動化というのが1つあります。開発の中でルーチンワークがあったならば、それは本当に人が行う必要があるのかと疑い、自動化の可能性を探ります。

自動化に関係する最近のエピソードとしては、勤怠管理にまつわるものがあります。Incrementsでは某ASPを用いた勤怠管理を開始しました。このシステムでは、出退勤時にブラウザからそのASPのWebページにアクセスしてログインしたあと、ボタンをクリックしなければなりません。エンジニアの1人がこの通信の中身を解析し、Slack上のbotに語りかけることで同じことを行えるように変更したのですが、それでもまだ「これって人間のやるこ

とじゃないですよね」と、今はオフィスに入っただけで出勤処理が行えるように実験中です[注8]。

情報共有

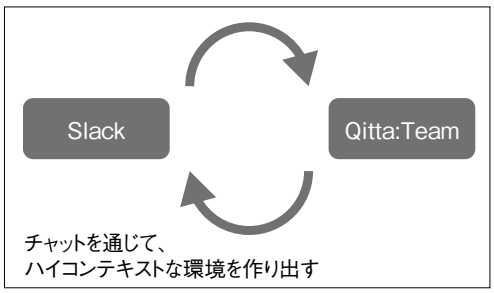
QiitaやQiita:Teamといったサービスを提供しているIncrementsですので、それらを用いた情報共有も徹底しています。テキスト情報として情報を共有することと、非同期コミュニケーションを重視することが特徴です。

共有する情報は、チャットのようなフロー情報とドキュメントやメモのようなストック情報に分かれます。Incrementsではチャットとして Slackを用いており、ストック情報にはQiita:Teamを用いています。

Slackは次節で紹介するように、各種CIツールやbotが連携されたChatOpsの基盤となっていますが、それ以外にも社内の他愛ない会話や開発に関する議論などもされています。オフィス内で口頭で済ますことも多いですが、それも過程や結果をSlack（長い場合はQiita:Team）に投稿するように心がけています。Slackでハイコンテキストな環境を作り、そこで投稿された内容をまとめたり、それ以外の暗黙知をテキスト化したりして、形式知としてQiita:Teamに投稿するという流れです。Qiita:Teamのストック情報を元に、またフロー情報としての会話が始まり、これがうまく循環されていくことでチームの情報共有は成立しています（**図5**）。

最近社内の一部で取り組み始めたこととしては、個人専用のチャンネル――たとえば「#status_takoratta」というのが筆者個人チャンネルですが――を用意して自分の作業ログを残していくというのがあります。プライベートチャンネルではなく、普通のチャンネルですので、ほかのチームメンバーも参加できます。

▼図5　フロー情報とストック情報の関係



もし筆者が自分の思考過程をこのチャンネルで共有したときに、ほかのメンバーがそれに対して知見を持っていたなら、アドバイスを得られます。また、そのような直接の利点がなかったとしても、誰が何をしているかがわかりやすくなります。Qiita:Teamでは日報を共有しているのですが、この個人専用チャンネルに書かれたものをベースに記載できるようになったので、手間も省けます。

Qiita:Teamでは、日報以外にも各種のメモやドキュメントが共有されます。現在のQiita:Teamには細かいアクセス制御の考え方はありません。一部のメンバーだけに共有すべき内容というのももちろんありますが、多くの場合は心のなかのハードルが全メンバーへの共有を妨げているだけではないでしょうか。実は筆者もIncrementsで勤務し始めたときに、特定メンバーにのみ情報共有をしたいと考えた1人です。実際には、特定メンバーにとくに見てもらいたいが、ほかのメンバーに見られても困るものではないことがほとんどでした。Qiita:Teamの今の機能が十分とは言えないですが、すべて公開するという精神はエンジニアリングチームの文化としてはとても大切なものではないでしょうか。

この“すべて公開する”という精神は社外に対しても当てはまります。Incrementsではオープンにできるものはオープンにするという精神で汎用性の高いコードのオープンソース化を進めており、積極的に技術情報を公開しています。

注8）Raspberry Piを用いて、社員の持つスマートフォンが社内Wi-Fiに接続されたら出勤、とみなすようなシステムです。

ChatOps基盤としてのSlack

Incrementsでも、最近多くの組織で導入が進みつつあるChatOpsという手法を用いて開発を行っています。ChatOpsとは、本誌2016年1月号でも特集が組まれていましたが、チャットをコミュニケーションツールとしてだけでなく、各種開発ツールから吐き出す情報を集約するのに利用したり、常駐する汎用botを利用することで開発ツールへの指示さえもチャット経由で行ったりするオペレーションです。

各種ツールと連携

Incrementsがチャットとして用いているSlackには、次のアプリケーションが連携されています(利用頻度の高い4ツールを抜粋)。

- CircleCI
- GitHub
- Googleカレンダー
- Zendesk

Googleカレンダーとの連携では、全社スケジュールが「#general」チャンネルに投稿されます。午前10時に当日のスケジュール一覧が投稿され、そして予定が開始される5分前にも投稿されます。また、個人のスケジュールを自分のチャンネルに投稿するように設定している人も多くいます。オンラインのカスタマーサポートにZendeskを使っていることはすでに書きましたが、ユーザからの問い合わせもその回答も、「#support」というチャンネルで共有されています。担当者以外でも、回答を率先して用意することが多くあります。

専用bot「Qiitan」によるオペレーション

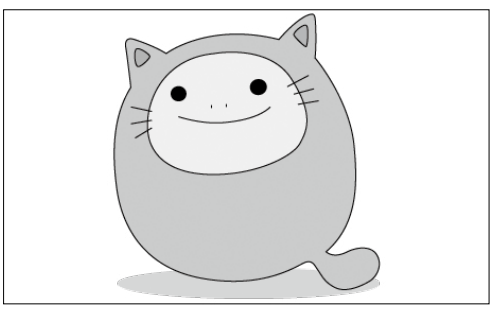
社員のコミュニケーションやツールが吐き出す情報の共有としてのチャットも重要ですが、なんと言っても開発のコアとなっているのは、Increments自社開発の専用bot「Qiitan」の存在です(**図6**)。

ChatOps用botフレームワークとしてはHubotが有名ですが、Qiitanはその互換Ruby実装のRubotyで作られています[注9]。このQiitanは、IncrementsのSlack上で次のようなさまざまな機能を提供しています(**図7**)。

- 天気予報
- 渋谷ランチ情報(社員が推薦するランチ情報が書かれたQiita:Teamの投稿の該当部分から抜粋)
- 任意のRubyコード実行(図7では、1から16の整数をランダムに並び替えていますが、これは社員のプレゼンテーションの順番を決めていたときのものです)

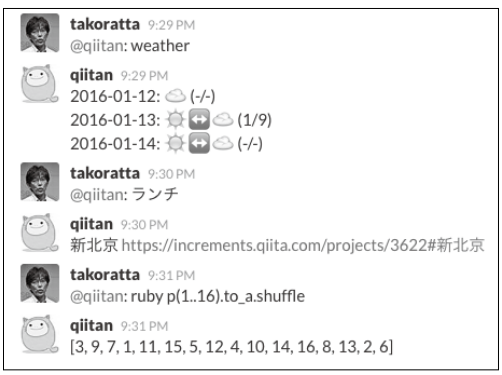
Qiitanはほかにも定期的なタスクの実行が

▼図6　自社製bot「Qiitan」



注9) Rubotyについては URL http://qiita.com/r7kamura/items/8d1b98e28154de6030b9にまとめられています。

▼図7　Qiitanによる天気予報の提供と渋谷ランチ情報、任意のRubyコード実行



可能です[注10]。たとえば、平日の朝10時に「おはよう！」と話しかけてもらうことも可能です(**図8**)。

Incrementsではこれをさらに応用し、オフィスの掃除の開始時間に掃除の分担をつぶやくようにさせています(**図9**)。この分担は社員の増加により変更の必要が出てきたのですが、それも**図10**のように簡単に変更できます。

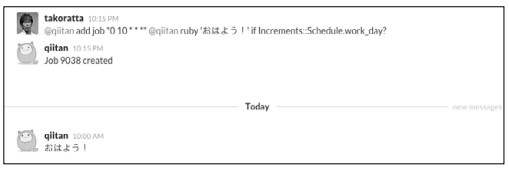
botによる開発オペレーションの実行

今まで紹介したQiitanの処理は、いろいろな定期オペレーションを自動化するものであったり、ともすれば無味乾燥に成りがちなチャットでのコミュニケーションを和ませてくれたりするようなものでしたが、Qiitanの真価は開発プロセスのハブとして機能するところにあります。

たとえば、GitHub Issueへの登録は**図11**のようにQiitanに話しかけることで行えます。

注10) Rubotyに定期的に何かしてもらう URL http://qiita.com/r7kamura/items/f7b5bf676494703b0758

▼図8　Qiitanによる定期的なタスクの実行



▼図9　掃除の分担の通知

▼図10　掃除の分担の更新

▼図11　GitHubへのIssue登録

この例では、タイトルだけ指定してIssue登録をしていますが、このような場合は時間ができしだい、GitHub上でIssueを更新してほかの必要な情報を入力します。

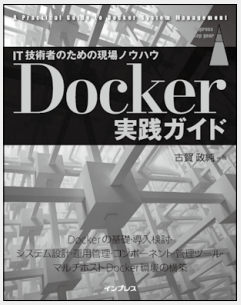
また、Qiitanを通じてデプロイすることもできます。masterブランチからproductionブランチへのPull Request作成と、レビュー後のそのPull Requestのマージを、Qiitan経由で行えます(**図12**)。

このほかにもQiitanは、Twitterでつぶやかれた「Qiita」が含まれたツイートの収集などもやっており、Incrementsにとってなくてはならない存在となっています。人間がするべきでない繰り返しの作業を代わりに行ってくれるだけでなく、開発プロセスの中心にいる愛すべきキャラクターとなっています。

おわりに

以上、Qiitaを支えている組織とチーム開発プロセスについて説明させていただきました。HRTを尊重し属人性を排除するための情報共有の工夫なども含めて、読者の方々のチーム開発に少しでもお役に立てたならば幸いです。SD

▼図12　Qiitan経由でのデプロイ



Docker 実践ガイド

古賀 政純 著
B5変形判／328ページ
3,000円＋税
インプレス
ISBN＝978-4-8443-3962-5

Dockerを現場でどのように導入・運用するのかに重きを置いた実践的な1冊。序盤の章は、Dockerという技術についての情報を整理しながら、向くシステム・向かないシステムといった導入に関する検討、導入前の準備・環境設計といった下準備を扱う。基本コマンド、Dockerfileの使い方を説明したあとは、各種ツールについて分野別で使い方を解説している。おもなものを挙げると、構築・スケールアウトのツールとして「Docker Compose」「Docker Swarm」、管理・監視ツールとして「DockerUI」、マルチホスト環境の構築として「Kubernetes」、コンテナ特化型OSとして「Atomic Host」「CoreOS」、クラスタ環境の構築として「Apache Mesos」などを紹介している。対象バージョンは、Docker 1.8/1.9。

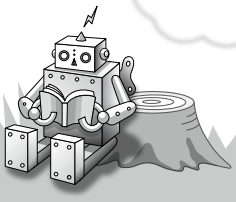
ネットワーク・デザインパターン

みやたひろし 著
B5判／440ページ
3,400円＋税
SBクリエイティブ
ISBN＝978-4-7973-8284-6

ネットワーク構築の現場における“鉄板構成”を紹介しながら、ネットワークの基礎知識、運用においてのTipsを解説している。本書は「Trust」「Untrust」「DMZ」「WAN」「総合構成」と、セキュリティゾーンによって章を分けており、各章はまたネットワークの規模によってSmall・Medium・Largeの3つのクラスに分けられ、さらに1つのクラス内で2つのデザインパターンが紹介されている。たとえばDMZの章では、Small：ミニマム冗長化構成／サーバ仮想化構成、Medium：負荷分散インライン構成／負荷分散ワンアーム構成、Large：ブレードサーバインライン構成／ブレードサーバワンアーム構成という順に紹介されている。

具体的なマシンの型番こそ出てこないが、限りなく実務に近いノウハウを学べるだろう。

SD BOOK REVIEW



リモートチームでうまくいく

倉貫 義人 著
四六判／224ページ
1,500円＋税
日本実業出版社
ISBN＝978-4-534-05342-8

著者は『「納品」をなくせばうまくいく』で話題になったソニックガーデンの倉貫氏。本書はその姉妹本とも言え、ソフトウェア開発会社の働き方として「リモートワーク」を提案する。タイトルにある「リモートチーム」とは、リモート先が個人でも、まるでオフィスでいっしょに働いているかのような「ほかの人の存在を感じられる」リモートワークのスタイル。チャットでの雑談の奨励や、いつでも相談できるように働く時間をそろえるなどの取り組みによって、「信頼関係」「セルフマネジメント」「チームワーク」が生まれてくる。リモートワークでより良いワークライフバランスを実現するために個人や組織で必要とされるものが何なのか、試行錯誤のうえに成り立っている同社の手法は、仕事環境を考えるときの貴重な参考となるはずだ。

ITインフラ監視[実践]入門

斎藤 祐一郎 著
A5判／160ページ
2,280円＋税
技術評論社
ISBN＝978-4-7741-7865-3

IaaS（Infrastructure as a Services）の普及によって、ITインフラの管理・運用を開発者が行う必要があるようなケースも増えてきた。本書ではそうした、インフラ管理・監視の経験があまりないが、自らがシステム監視を行う必要がある、管理・監視を専業としないエンジニアを想定して、システム監視の全体像とそのしくみの設計・運用のノウハウを解説する。監視の設計、現状分析、障害監視のための判断基準設計、監視サーバの選択、経路設計、監視業務設計、そして構築と運用後の問題への対処、というように、システム監視の全体像、監視システムの設計・構築と監視業務の勘どころを流れを追って説明する。いざというときに備えるために、筆者が経験してきたインフラ監視においての勘どころを解き明かす書籍だ。

第2特集

好き嫌いで判断していませんか？

あなたの知らないCOBOLの実力

―経済を支える基盤技術―

春先に決まって売れるIT書籍あり。それは「COBOL」の本。COBOLは、古い、レガシーな技術だ、今どきじゃないと思われていますが、金融システムを支え、社会の基盤であることは言うまでもありません。

初期のCOBOLは囲い込まれ部外者からは見えにくいものになっていましたが、現代にいたるまでにさまざまな情報科学の知見が反映され、いろいろな人や組織に門戸を開放した言語になっています。まずは第1章でCOBOLの基本的な話を紹介します。そして第2章ではopensource COBOLでCOBOLのパワーをほんの少し体験します。第3章ではCOBOLの本当の良さ・その本質とは何か考え、第4章では、COBOLプログラマが他のプログラミング言語と向き合うときの話を紹介します。最後に前述したCOBOL書籍の裏側を執筆者自らが解説します。COBOLを通して眺めれば日本が抱えるIT業界の問題も見えてきます。

多くのシステムで使われる理由はどこにある？

第1章 ちょっと深めのCOBOLの話

Author 高木 渉（たかぎ わたる）
COBOLコンソーシアム 会長（㈱日立製作所）

COBOLは事務処理用言語と言うだけあって、多くの金融／公共システムで利用されています。しかし、どんなところが事務処理に向いているのでしょう？「固定小数点の十進数による演算」「英語に近い書き方」「データ構造の定義の仕方」の3点に注目して、COBOLが事務処理に使われてきた背景を考えてみます。

はじめに

COBOLは、汎用のプログラミング言語です[注1]。その最初の仕様書は、1960年に米国で発行されました。

ここでは、初期の設計時に組み込まれたCOBOLの特徴を紹介して、現在の目で評価してみたいと思います。

一般的に、プログラミング言語は世に出たときに完成形であることはまずなく、次々に必要と考えられる仕様を追加していきます。仕様を追加する場合は、以前の仕様との互換性を保つようにするのが原則です。非互換になる変更をすると、新しい仕様に基づくコンパイラでは、既存のユーザ資産がコンパイルできなくなってしまうからです[注2]。

COBOLにも1960年当時の仕様が残っていて、それは長所でもあり、制約でもあります。

なお、本稿では、初期の言語設計の経緯をSammetの論文[2]に依拠します。

注1）事務処理用と思われがちだが、言語としては、国際規格書にもあるように「汎用言語として広まっている」[1]。
注2）コンパイルはできても、実行時の動作が以前と異なるような仕様変更もあり得る。このような仕様変更も影響が著しいので、同様に避けるようにしている。

COBOLは共通の事務処理用言語

1959年に米国で、政府関係者、コンピュータユーザ、コンピュータメーカーの人々が、共通の事務処理用言語を作ろうと集まりました。これがきっかけで、翌1960年4月にCOBOLの最初の仕様書が発行されました。

COBOLという名称は、COmmon Business Oriented Languageから取られていて、そのまま「共通の事務処理用言語」です。ここで、「共通の言語」と「事務処理用の言語」を分けて考えてみます。「共通の言語」の部分を理解するには、時代背景を考慮する必要があります。

1959年当時は、まだコンピュータの黎明期です。複数のメーカーが、異なるハードウェア・アーキテクチャのコンピュータを開発しては世に出していました。オペレーティングシステム（以下、OS）の概念はまだ確立されていません。FORTRANはすでにあったとはいえ、高級言語も黎明期です。いわゆる事務処理用言語は、実装が1つ知られていて、ほかにも実装しようとしている機関が現れている時期でした。1959年に集まった人たちは、1つのハードウェア用にプログラムを作っても、ほかのハードウェア上では再プログラミングすることになり、時間と費用が掛かってしまうことを問題として意識していました。

「共通の言語」とは、いわゆるOSがなくてむき出しの、ばらばらなアーキテクチャのハードウェアのどれにも依存しない言語、という意味合いです。プログラムに移植性(ポータビリティ)がないことは、コンピュータの黎明期から問題視されていたことになります。

「事務処理用言語」については、定義がありません。1959年5月の会議の記録で、簡単な英語を最大限使用する方針に、多数が賛成しています。筆者は、FORTRANのような数式を扱う言語とは異なる言語、という意識もあったのではないかと想像します。しかし、英語で書くことと「事務処理用言語」との関係は明らかではありません。

現在でも事務処理用言語の定義はないので、COBOLの当初からの仕様で、筆者が事務処理向けだと思う点を3点挙げます。

①固定小数点の十進数を扱う
②記号よりは英語での表現を好む
③入出力レコードを階層構造のある連接したフィールドで定義する

次節で、これら3点について評価してみます。

COBOLの事務処理向けの仕様の評価

❶十進数の演算に強いCOBOL

十進数の桁を明示する変数の定義

COBOLで数値を扱う変数[注3]を定義するとき、十進数の桁数と小数点位置を指定します[注4]。指定のしかたを例で見てみましょう。

```
01  hensuu PICTURE 9999V99.[注5]
```

「01」はレベル番号と言います(詳しくは後述)。「hensuu」[注6]は、定義する変数の名称です。

「9999V99」の部分をPICTURE文字列と言います。並べた9の個数が十進数の有効桁数を表し、Vの位置が小数点位置を示しています。ここでは「6桁の十進数で、小数点以下の桁数が2桁である、hensuuという名称の変数」を定義しています。

ほかの多くの言語では、数値型の変数が表現可能な値の範囲を考えるときに、その型が占めるバイト数を意識すると思います。たとえば、C言語なら、int型の変数が占めるバイト数を認識したあとで、表現できる値の範囲を確認するのではないでしょうか。

COBOLでは、数値変数の定義で、表現できる値の範囲を十進数の1桁単位で指定します。4バイトだ、8バイトだ、というバイト数が先ではありません。しかも、小数点の位置まで指定して、固定小数点の十進数として定義してしまうことが特徴です。整数の変数は、小数点を右端に置いた特別な場合と考えることができます。

変数の定義で丸めを制御

COBOLの実行文で、演算の結果を丸めるための処理は、通常書きません。演算結果は、変数の定義から、十進数の最小の桁位置を基準に丸めてくれます[注7]。

COBOLが十進数に強いと筆者が思う理由は、ここにあります。十進数での丸め位置の指定を変数の定義に片寄せしていて、実行文には丸め制御のためのコードを書きません。十進数での丸め制御のためのコードを書かなければいけないプログラミング言語の場合、本来の処理のためのコードが埋もれてしまいます。

整数でも十進の固定小数点数を扱える

十進数の0.1を二進数で表現すると循環小数になるように、十進数の小数点以下の値は、必ずしも二進数で表現できません。ですから、小数点を有効桁の左端のほうに寄せて正規化する

注3) COBOLの用語では「データ項目」という。本稿では、COBOLに独特な用語を使うことは、できるだけ避ける。
注4) 桁数を指定しない種類の変数もある。
注5) COBOLソースプログラムでは、英字の大文字は対応する小文字と等価である。ただし、文字列定数の中は除く。
注6) 変数名には日本語も使える。本稿では英字にした。
注7) 丸め方の指定がなければゼロ側に切り捨てる。四捨五入するには指定が必要である。

二進数の浮動小数点数型では、COBOLが扱う十進の固定小数点数を同じ精度では扱えません。

整数型の変数で工夫する方法はあります。小数点以下がn桁あれば、小数点をn桁移動して整数にし、このnを何らかの形でペアとして持っていれば整数だけで固定小数点数の演算はできます。演算は整数で行うので、二進数でも十進数と同じ精度で小数点以下の値を作ることができます。たとえば、2.3と3.7を掛け算するのに、次のように分解すれば途中に小数点以下の演算は不要です。

$$
\begin{aligned}
&2.3 \times 3.7 \\
&= (23 \div 10) \times (37 \div 10) \\
&= 851 \div 100 \\
&= 8.51
\end{aligned}
$$

この例で書いた2.3も3.7も8.51も、整数型では表現できないので、工夫して保持しなければならない値です。途中の演算も面倒です。丸め制御が絡むとさらに複雑になります。

COBOLでは、とくに指定しないで数値変数を定義すると、各桁を実際に十進数で値を持つ形になります。ASCII文字を使う環境ならば、ASCIIの数字列で値を持つイメージです。

ところで、筆者が耳にする言説で「COBOLは十進数で演算できるから誤差がない」というのがありますが、正しいとは思えません。丸めが絡む以上、演算誤差はあるのです。誤差の扱いが、十進数を基本にした丸め方の定義に沿っているだけです。

❷英語でプログラムを書くCOBOL

英語でプログラムすることで期待する効果

簡単な英語でプログラムを書くことは、当初のCOBOLの設計方針です。もし、経営層や、そこまでいかなくても、プログラミングと関係ない実務担当者が、英語で書かれているというだけでプログラムを読んで理解できるなら、夢のような事務処理用言語です。

しかし、英語だから誰でもわかるという期待は、ほぼ裏切られていると筆者は思います。一方で、英語で書くのでCOBOLの記述が長いという、通常批判的に語られる特徴が、プラスに働いている面はあると思っています。

英語によるプログラム表現

COBOLの実行部分[注8]は、英語の動詞で始まる文の連続です[注9]。たとえば、代入[注10]命令なら、次のように書きます。

```
MOVE a TO b.
```

これで、変数aの値を変数bに代入します。MOVEという動詞を使うので、ほかの言語に慣れた人には見慣れないと思いますが、代入です。英語らしく、ちゃんとTOという前置詞を入れています。また、英語の文になぞらえて、文がピリオドで終わっています。

もう1つの例として、割り算を実行するDIVIDE文の書き方の1つを見てみます[注11]。

```
DIVIDE a BY b GIVING q REMAINDER r.
```

それなりに英語として読める語順になっています。「変数aの値を変数bの値で割って、商を変数qに入れ、余りを変数rに入れよ」と読めます。このように、一定数の決まった英単語を規則どおりに並べることで、文を構成します。

次にCOBOLの実行部分の構成を簡略化したものを**リスト1**に示します。

構造として、複数の節があり、節の中に複数の段落が含まれ、段落の中に複数の文が含まれます。英語の文章の構造をなぞった構造になっています[注12]。

リスト1中の、「節の名前n」や「段落の名前n」は、いわゆるラベルですが、単に位置を示すものではありません。その節や段落に含まれる文

注8) COBOLの用語では「手続き部」という。
注9) IF文のように動詞でない語で始まる文もある。
注10) COBOLの用語では、「転記」という。
注11) 念のために補足すると、COBOLは1960年の仕様書のときから算術式を記述できる。このDIVIDE文の例のような、加減乗除それぞれ用の文も用意されている。
注12) 節や段落のない書き方も存在する。

全体に名前を付けたものです。この違いは、PERFORM文で節や段落に分岐するときに明らかになります。**リスト1**のプログラム構造の例の途中で、どこかに次のような文を書くとします。

```
PERFORM 節の名前2.
```

すると、「節の名前2」というラベルに分岐したあと、この節の中の段落に含まれる文をすべて実行したあとに、このPERFORM文の次の文に制御が戻ります(**リスト2**)。

英語で書けば経営層がプログラムを理解できるか[注13]

プログラムは英語の文章のようには読めません。仮に逐次処理しかないとしたら、頭から読んで理解できるかもしれません。しかし、分岐が出てくると、途端に文章らしく読むことができなくなります。また、処理の目的を知らずにプログラムを追うことが難しいのは、英語で書いても変わりません。

もしかしたら、1960年当時、コンピュータが現実的に実施できる業務処理の内容は、経営層が読んでもわかるくらいの複雑さだったのかもしれません。大きなコンピュータで時間をかけて計算しても、現代の表計算ソフトの基本的な機能で簡単にできることをやっと処理する感覚です。しかし今では、コンピュータに処理させる業務処理の内容が、膨大、かつ、複雑になり、専門にしている人でもすぐに理解できるものではありません。

そもそも経営層がプログラムを読む必要性のある場面は、そうそうないでしょう。多くの場合、いかに処理しているかは重要ではなく、結果のほうに興味があるはずです。

英語になりきれない構文

英語や日本語などの自然言語でプログラムを書こうという考え方は、好き嫌いはあっても、理解はできると思います。しかし、どうやら英語らしい構文だけでプログラムを記述することはできないように思います。

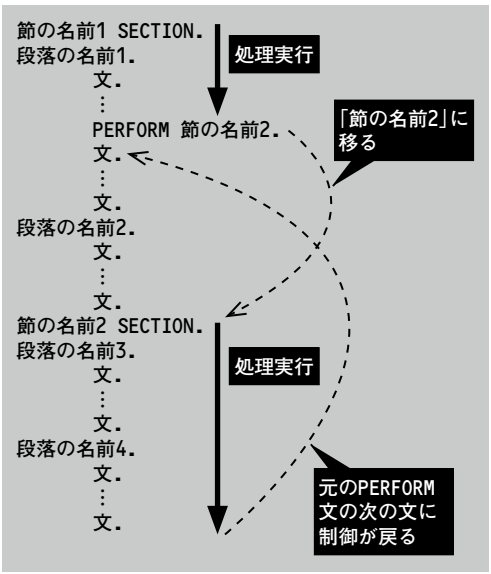
たとえば、英語では処理の枠構造が作れません。構造化プログラミングが広まり、COBOLに枠構造を作る構文が導入されたのは、1985年の規格でした。このとき導入したのは、英語とは言えない構文です。たとえばIF文なら、この

注13) 経営層でも、自在にプログラミングする方は除く。

▼リスト1　COBOLの実行部分の構成

```
節の名前1 SECTION.
段落の名前1.
        文.
        ⋮
        文.
段落の名前2.
        文.
        ⋮
        文.
節の名前2 SECTION.
段落の名前3.
        文.
        ⋮
        文.
段落の名前4.
        文.
        ⋮
        文.
```

▼リスト2　PERFORM文による分岐の例



文を終わらせるために導入されたのは、END-IFという語でした。しかし実際の英語には、END-IFなどという単語はありません。結局、英語らしく枠構造を作ることはできず、妥協したとも取れます。

枠構造を作る構文を導入しても、互換性の観点から、従来の構文は禁止していません。依然としてEND-IFで閉じないIF文の書き方を許しています。このとき、IF文を終わらせるのは、ピリオド(.)です。

次の例の「MOVE g TO y」はELSE部に含まれません。「MOVE f TO x」の直後のピリオドでIF文を終わらせているからです。このピリオドがなければ、「MOVE g TO y」もELSE部に含まれます。

```
IF a = b THEN
       MOVE e TO x
ELSE
       MOVE f TO x.
MOVE g TO y.
```

次のようにちゃんとEND-IFで閉じるほうがはるかに構造が明快です。

```
IF a = b THEN
       MOVE e TO x
ELSE
       MOVE f TO x
END-IF.
MOVE g TO y.
```

筆者は、枠構造を作らない書き方のプログラムを今から追加していくことには賛成しかねます。しかし、1985年のCOBOL規格以前のスタイルで書かれたプログラムは現役で存在します。ときに、そうしたプログラムを読んで理解し、変更する必要があります。

英語を採用して良かったこと

これまで見たように、英語(自然言語)で書くからという理由でプログラムの意図まで読みとれるとは思えません。また、英語らしい表現だけで、プログラムの構造を記述するのには無理があります。

COBOLが英語表現を採用したことで得た利点は、プログラム記述の粒度が適切なレベルに落ち着いたことだと、筆者は考えています。どうやっても、詰め込み過ぎない粒度でプログラムを記述するようになります。

理解しやすさの観点での粒度の適切さについては、筆者の感覚に過ぎないので、この主張に議論はあるかと思いますが、あえて述べておこうと思います。

プログラミング言語は、プログラマが楽しくプログラミングできるようなものであるべきだという趣旨の主張を聞いたことがあります。その点に異論はありませんが、同時に、読んで理解しやすいものでもあるべきだと筆者は考えます。楽しくプログラミングできても、他人が理解するのが困難なプログラムを書き散らすことになっては、のちに何度も変更しなければならない担当者に酷です。

❸入出力レコードを構造化するCOBOL

入出力レコードを分割して階層化

COBOLの最初の仕様書が書かれた1960年当時、OSの概念はまだ確立されていません。ですから、COBOLのプログラムの中で、入出力の対象には、あからさまにハードウェアデバイスの名前を指定しました[注14]。そして、入出力用に定義する領域は、ハードウェアと直接やりとりするときに使う入出力バッファのイメージです[注15]。

筆者は、COBOLのデータ構造の作り方は、この入出力バッファをフィールドに区切るところから始まっていると考えています。

入出力バッファにあたる領域をレコードと呼びます。入出力はレコード単位に行います。レコードの中をバイト数でフィールドに分割し、

注14) 今はデバイス名である必要はない。文字列定数で、UNIXやWindowsのようなファイルシステムのパス名を指定する構文がある。
注15) 現在のアーキテクチャではOSが間に入る。直接入出力バッファとして利用することはまずない。

それぞれに変数名を付けます。たとえば、**図1**のような形です。

レコード内のフィールド分割には、次の特徴があります。

1. フィールドが連接していて隙間がない
ほかのプラットフォームにデータを移しても、領域がずれない[注16]

2. 複数のフィールドをまとめたフィールドを定義できる
階層的にデータを定義すること自体はほかの言語でも珍しくないが、上位階層と下位階層の関係に特徴がある

図1のレコードをCOBOLで表現してみます(**リスト3**)。変数名は、**図1**に対応して筆者がそれなりに付けたものです。

各行の左端にある「01」「02」「03」は、レベル番号です。字下げに文法的な意味はありません。01レベルは、レコード全体です。番号が大きくなるほど階層が深いことを示します。PICTUREの後ろの「X(n)」は、領域に入る英数字文字の個数を示します[注17]。ここで「X(6)」は、「XXXXXX」と、Xを6回続けて書くのと等価です[注18]。

注16) 実際には、まったくずれることがないと言い切れるほど理想的ではない。
注17) 説明を簡単にするために、この例で日本語データは使っていない。
注18) 同様に、固定小数点数の例で示したPICTURE文字列の「9999V99」は、「9(4)V9(2)」とも書ける。

▼リスト3　COBOLのレコード表現

```
01  gakusei-rec.
  02  gakusei-no PICTURE X(6).
  02  kojin-zokusei.
    03  shimei PICTURE X(10).
    03  juusho PICTURE X(20).
    03  denwa  PICTURE X(11).
```

COBOLに特徴的なのは、下位の階層の変数のあり方を無視して、上位の階層の変数を使えることです。たとえばレベル番号02のkojin-zokuseiは、X(41)の変数として使えます。領域のサイズを41と言い当てられるのは、フィールドが連接しているからできる芸当です。kojin-zokuseiに、下位の階層の意味や切れ目を無視した値を代入してもかまいません。

入出力中心のデータ領域の扱い

レコードを分割する考え方は、プログラム内で定義する入出力に直接関係しない領域にも使います。入出力バッファと同じく、あるメモリ領域をレコードとし、階層的にフィールドに分割して、各フィールドに変数名を付けます。

入出力バッファ内のデータ領域の扱いが、そのままプログラム内部のデータ領域の扱いにつながる仕様は、入出力を中心にした言語設計だと筆者は考えています。入力したレコードをそのまま内部領域に移して処理でき、処理結果をすぐに出力レコードに移せるからです。

データの表現として、テキストで値と構造を表す記法がよく使われますが、その処理オーバーヘッドにも注目すべきです。単純なCSV(comma-separated values)形式にしても、入力した1行を分解して対応する各変数に入れる処理や、変数の並びからCSVの1行に組み立てて出力する処理に、計算リソースを使います。本来の処理でない傍流の処理でリソースを使い、性能ネックになりかねません。COBOLのレコード入出力は、そのような変換が不要な仕様が基本になっています。

▼図1　レコード内のフィールド分割のイメージ

領域のずれを生みやすい文字の扱い

文字コードに関してCOBOLに特異な点に触れます。

2002年のCOBOL規格で、英数字に加えて各国文字（National characterの訳）を扱う変数を定義できるようになりました[注19]。1文字に複数バイトを占めるコード（Shift_JISやUnicodeなど）の文字データを格納できます。

ここで、COBOLの仕様では、1文字を格納する領域の占めるバイト数が、英数字と各国文字でそれぞれ固定であることが原則です。英数字が1バイトで、各国文字が2バイトといった調子です[注20]。それぞれのバイト数は、COBOLの実装者が決めていいことになっています。しかし、たとえば各国文字は2バイトであると決めたら、可変にはできません。COBOLは、レコードやフィールドの領域サイズを意識する言語なのです。

ところで、プログラムをほかのプラットフォームに移行するとデータの移行も伴います。このとき文字コードが変更になることがあります。移行先の文字コード1文字の占めるバイト数が、移行元とは異なるとき、領域の再設計が必要になります。

ことを複雑にするのが、下位の階層の変数のあり方を無視して、上位の変数を扱える仕様です。たとえば、上位の変数を扱っているのに、下位の変数の切れ目を意識していると、フィールドの位置がずれて意図どおりにならないことがあるのです。1文字のデータを領域中の固定バイト幅の塊として扱い、抽象化していないことの副作用です。

なお、英数字データしか扱わないプログラムでは、データ移行に伴う領域サイズの問題は起こりません。

仕様を残し続けるプログラミング言語

COBOLは今もよく使われている言語で、国内ではJavaに次ぐ選択率（第1言語の場合）との調査結果もあります[3]。Javaが台頭する前は、最も使われた言語でしょう。これだけ使われていると、大量の資産プログラムが存在します。過去の言語仕様を、そう簡単に捨て去ることはできません。

COBOLは、1960年当時の人が事務処理には最善と考えていた技術から始まっています。本稿では紹介していませんが、COBOLはこれまで、規格の改正のたびに最新の技術を取り入れてきました。一度導入された仕様は今も残り[注21]、現在の知見で判断すれば、長所も短所もあります。

COBOLは初め、ハードウェア非依存という意味で「共通」言語を目指しました。それから半世紀を経て、COBOLは時代を超えて「共通」な言語としても使われています。SD

注19）日本市場のCOBOLベンダが仕様を提案して、国際規格に採用されたもの。
注20）UTF-8は1文字が可変バイト数を占めるので、各国文字に使うには不適切な符号化方式である。
注21）実際には規格から削除した仕様もある。次回の規格改正時に仕様を削除することを予告するしくみがある。

＜参考文献＞
［1］ISO/IEC 1989:2014 Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL
［2］Jean E. Sammet: The Early History of COBOL, History of Programming Languages, ACM (1981)
［3］独立行政法人情報処理推進機構（IPA）、『ソフトウェア開発データ白書2014-2015』

金額・利率計算で実践

第2章 opensource COBOLを試してみよう

Author 稲垣 毅（いながき つよし）㈱日立ソリューションズ 技術開発本部 研究開発部、OSSコンソーシアム
清水 真（しみず まこと）東京システムハウス㈱ マイグレーションソリューション部、OSSコンソーシアム

かつてのCOBOLはメインフレーム上で利用されるばかりでしたが、現代ではLinuxなどのオープン系サーバでの利用も増えています。本章では、そのようなCOBOLの1つ「opensource COBOL」を実際に使ってみましょう。COBOLが向いていると言われる金額・利率の計算を試して、その特徴を理解しましょう。

opensource COBOLとは

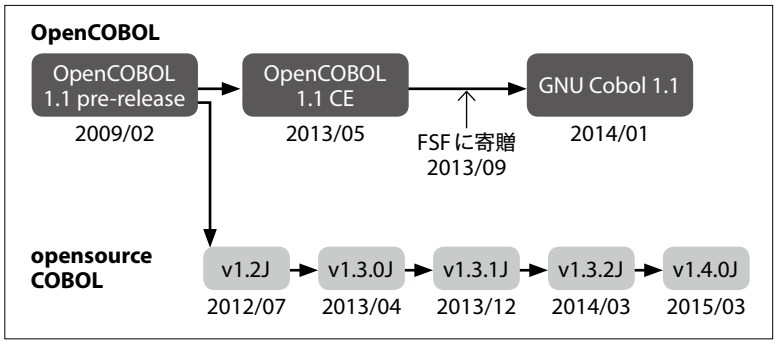
「opensource COBOL」は、COBOL85とCOBOL2002のいくつかの重要な仕様に準拠したオープンソースのCOBOLコンパイラです。メンテナンスは、任意団体であるOSSコンソーシアムのオープンCOBOLソリューション部会[注1]で行っています。

opensource COBOLは、オープンソースのCOBOLコンパイラ「OpenCOBOL」から派生したものです。OpenCOBOLは、日本医師会総合政策研究機構ORCAプロジェクト[注2]で日医標準レセプトソフトのために開発されました。原作者は西田圭介氏（当時、㈱ネットワーク応用通信研究所）です。先述の日医標準レセプトソフトは、2002年の本運用開始以来、現在も多くの医療機関で実運用されています。

それでは、OpenCOBOLとopensource COBOLの関係をそれぞれのバージョンの推移とともに振り返ってみましょう（**図1**）。

OpenCOBOLは、ORCAプロジェクト完了後、ヨーロッパを中心としたOpenCOBOLコミュニティに移管され、2009年2月にOpenCOBOL 1.1 pre-releaseがリリース、コミュニティ有志によるバグフィックス反映版のOpenCOBOL 1.1 CE（Community Edition）が2013年にリリースされています。その後、フリーソフトウェア財団（FSF）に寄贈されGNUパッケージとなり、2014年1月にGNU Cobol 1.1がリリースされています。GNUパッケージとなった際、COBOLコンパイラ（ランタイム除く）のライセンスは“GPLv3 or later”に、ランタイムのライセンスは“LGPLv3 or later”に移行しています。

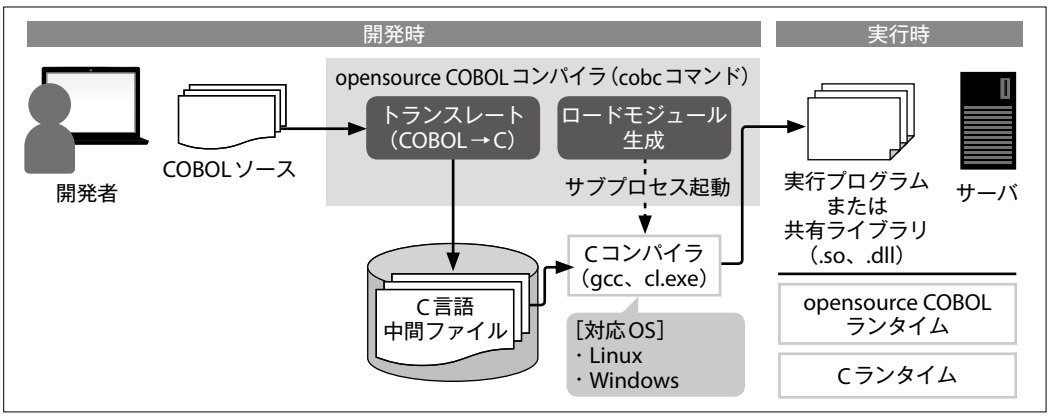
▼図1　OpenCOBOLとopensource COBOLの関係



注1）オープンCOBOLソリューション部会のページ
URL http://www.osscons.jp/osscobol/
開発ページ（GitHub）
URL https://github.com/opensourcecobol/opensource-cobol

注2）URL https://www.orca.med.or.jp/

▼図2　開発から実行までの流れ



一方、日本のビジネス利用を指向した拡張[注3]をするため、先述の団体で参加企業を募りコミュニティを立ち上げました。コミュニティ立ち上げ後の2012年2月にopensource COBOLとしてフォークし、定期的にメンテナンスを行っています。現在は、バージョン1.4.0Jがリリースされており、フォーク後は機能拡張やバグフィックスで100以上のパッチを作成・適用しています。また、導入実績も着実に進んでおり、コミュニティ内だけでも、企業の基幹系システムへの導入やクラウド環境への適用が進んでいます。ちなみに、開発を行っている企業が、opensource COBOLを使ったマイグレーションサービスやサポートを提供しています[注4]。ライセンスに関しては、OpenCOBOL 1.1のときと同じ（COBOLコンパイラ（ランタイム除く）は“GPLv2 or later”、ランタイムは“LGPLv2 or later”）で変更はありません。

opensource COBOLのしくみ

opensource COBOLは、gccなどのコンパイラと同様、コンパイラとランタイムが存在します。コンパイラの一部の実装（ISAMや数値演算、動的CALL文、SCREEN SECTION）を外部のライブラリに任せており、たとえば、COBOL言語の特徴でもある数値演算にはGNU MP（Windows環境の場合はMPIR）を使っていて、一部は高速化を狙ってシンプルにした独自のコードを生成するよう工夫もされています。

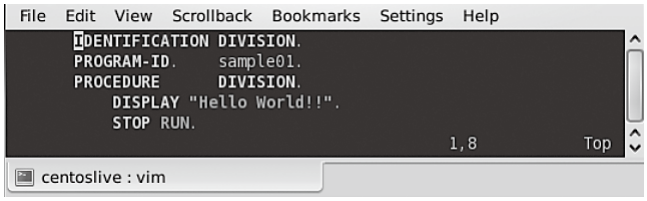
次に、COBOL開発から実行までの一連の流れを**図2**に示します。opensource COBOLコンパイラは、COBOLソースをC言語に一度トランスレートしたあと、実行プログラムを生成します。実行プログラムを生成する際にCコンパイラを使用するため、gcc（Windows環境の場合はVisual Studio）が必要になります。この実行プログラムは、通常の実行可能ファイルのほかにsoやdllも生成できるので、他言語や他システムから呼び出すことができるといったメリットがあります。なお、生成された実行プログラムは、opensource COBOLランタイムとCランタイム上で実行するため、それぞれのランタイムが必要になります。

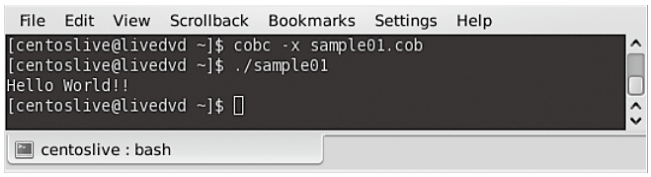
開発にあたっては、Eclipseなどの統合開発環境が準備されていませんので（連携することは可能で実績あり）、COBOLソースの開発はvi（Vim）やEmacsなどのエディタで行い、コンパイルと実行はシェルで行います（**図3**）。

COBOLプログラムをデバッグする場合は、基本的にgdbを用います。opensource COBOLのコンパイラオプションでC言語のコードにト

注3）日本語機能のサポートやWindowsネイティブ（MSVC：Microsoft Visual C++）のサポート、データベース連携など。
注4）提供サービス一覧
URL http://www.osscons.jp/osscobol/service/

▼図3　Vimでのコーディング（上）とシェルでのコンパイルと実行（下）





▼図4　トランスレートのみして、Cのコードを出力

```
$ cobc -C sample01.cob
$ vi sample01.c
（略）
/* MAIN SECTION */

/* MAIN PARAGRAPH */

/* sample01.cob:4: DISPLAY */
{
  cob_display (0, 1, 1, &c_1);
}
/* sample01.cob:5: STOP */
{
  cob_stop_run ((*(int *) (b_1)));
}
（略）
```

ランスレートした結果を出力するようにして、COBOLソースに対応したC関数を確認します（**図4**）。そのうえでgdbを使ってその関数にブレークポイントを設定し、ステップ実行してデバッグをするといった流れです（**図5**）。

なお、開発目的でopensource COBOLをインストールする際には、gccオプションの「-g -O0」を付与して、コンパイルするようにしてください。

opensource COBOLの環境構築

opensource COBOLは、Windowsもしくは Linux環境で実行することができます。今回は最もシンプルなLinux環境で使用する方法を紹介します。

Linux環境にopensource COBOLを構築する場合、ソースパッケージからインストールする方法とrpmパッケージからインストールする方法があります。またソースパッケージには、文字コードがShift_JISとUTF-8の2種類があります。

文字コードの違いについては、コラム「COBOLと文字コード」をご覧ください。

パッケージは、OSSコンソーシアムのオープンCOBOLソリューション部会のダウンロードサイト[注5]よりダウンロードできます。2016年1月時点の最新バージョンは、1.4.0Jとなります。

ダウンロードサイトより、**図6**のパッケージを入手します。

注5）ダウンロードサイト
URL http://www.osscons.jp/osscobol/download/

▼図5　gdbでのデバッグ

```
$ gdb sample01
(gdb) b cob_display   ←gdbを起動して、ブレークポイントを設定
Breakpoint 1 at 0x4008d0
(gdb) r               ←実行
Starting program: /home/inagaki/sample01

Breakpoint 1, cob_display (outorerr=0, newline=1, varcnt=1) at termio.c:214
214             if (!outorerr && !cob_screen_initialized) {

(gdb) s
（ステップ実行して処理を進める）
(gdb) s
222                     display (f, fp);
(gdb) p f->data       ←ある時点の値を取得
$1 = (unsigned char *) 0x400c40 "Hello World!!"
```

また、Linux環境でopensource COBOLを実行するには次のモジュールが必要です。

・gcc
・gmp
・ncurses
・db4

今回、Amazon EC2に用意したCentOS 6.5(x64)環境で進めたところ、これらのモジュールが標準ではインストールされていませんでした。そこで、それぞれyumコマンドでインストールを行っています。

```
$ yum install gcc
$ yum install gmp-devel
$ yum install ncurses-devel
$ yum install db4-devel
```

続いてrpmコマンドでopensource COBOLをインストールします(**図7**)。

これでopensource COBOLのインストールは完了です。なおopensource COBOLのライブラリは、/usr/local/libにインストールされますので、環境変数「LD_LIBRARY_PATH」にこのディレクトリのパスを通してください。

```
$ export LD_LIBRARY_PATH= $LD_LIBRARY_
PATH: /usr/local/lib
```

これでopensource COBOLの環境構築は完了です。

opensource COBOLでローン返済額の計算をする

COBOLの特徴の1つに、四則演算に優れて

▼図6　rpmパッケージのダウンロード

opensource COBOLのダウンロード

opensource COBOL v1.4J系のパッケージ、ソースコードが入手できます。

●v1.4.0J
☆Linux[x86_64, x86]
-opensource COBOL v1.4.0J[tarball]

opensource COBOL v1.4.0J [SJIS, Source Package, include vbisam (extend OSSCons patch)] (opensource-cobol-1.4.0J.tar.gz)	ダウンロード
opensource COBOL v1.4.0J [UTF-8, Source Package, include vbisam (extend OSSCons patch)] (opensource-cobol-1.4.0J-utf8.tar.gz)	ダウンロード
-opensource COBOL v1.4.0J[rpm]	
opensource COBOL v1.4.0J [SJIS, RHEL/CentOS 6 x86_64, RPM Package, Berkeley DB] (opensource-cobol-1.4.0J-1.el6.x86_64.rpm)	ダウンロード
opensource COBOL v1.4.0J [SJIS, RHEL/CentOS 6 x86, RPM Package, Berkeley DB] (opensource-cobol-1.4.0J-1.el6.i386.rpm)	ダウンロード

▼図8　元利均等返済方式の計算式

$$\text{毎月の返済額} = \frac{\text{借入金額} \times \text{月利} \times (1 + \text{月利})^{\text{返済回数}}}{(1 + \text{月利})^{\text{返済回数}} - 1}$$

いるという点があります。プログラマが、COBOLで扱うデータ項目の整数・小数の10進桁数を明示的に定義することができます。また、四則演算やべき乗の計算も自由に記述できます。このことが、COBOLが事務計算処理に向いた言語であると言われる所以(ゆえん)の1つになっています。

この特徴を活かして、今回はローンの返済額を計算するサンプルプログラムをopensource COBOL環境で実行しましょう。

今回は、元利均等返済方式で計算を行います。元利均等返済方式とは、返済額(元金＋利息)が一定となるため、返済計画が立てやすい返済方式です。

元利均等返済方式の計算方法は**図8**のとおりです。

この計算式をCOBOLで記述すると、次のと

▼図7　opensource COBOLのインストール

```
$ rpm -ivh opensource-cobol-1.4.0J-1.el6.x86_64.rpm
Preparing...                ########################################### [100%]
   1:opensource-cobol       ########################################### [100%]
```

おりとなります。

```
COMPUTE PAYMENT =
     (LOAN * INTEREST-MONTH *
((1 + INTEREST-MONTH) ** (PERIODS))) /
((1 + INTEREST-MONTH) ** (PERIODS) - 1).
```

またCOBOLでは、プログラムで扱うすべてのデータ項目をデータ定義部に記述する必要があります。計算式で必要なデータ項目は次のように定義します。

```
01  COMPUTE-AREA.
  03  INTEREST-MONTH PIC 9(02)V9(09).
  03  PERIODS        PIC 9(03).
  03  PAYMENT        PIC 9(09)V9(09).
  03  LOAN           PIC 9(09)V9(09).
```

データ定義名の前に記載されている「01」はレベル番号と呼びます。レベル番号を変更することでデータ項目の階層や従属関係を定義できます。

このサンプルでは、01レベルの「COMPUTE-AREA」のデータ項目の従属データとして、03レベルで「INTEREST-MONTH」「PERIODS」「PAYMENT」「LOAN」の4つのデータ項目が定義されています。

COMPUTE-AREAを初期化することで、すべての従属データを初期化することができます。

```
INITIALIZE COMPUTE-AREA.
```

また変数名の後ろにある「PIC …… 」は変数の型を表しており、9は正の整数、括弧内の数字は桁数、Vは小数点の区切り位置を意味しています。

このほかにも、COBOLにはさまざまなプログラミングのお作法があります。もし興味があればCOBOLの解説本やインターネットをご覧ください。

それでは「借入金額」「利率」「返済回数」から「毎回の返済額」を求めるプログラムを紹介します(**リスト1**)。このサンプルプログラムは、画面にそれぞれの値を入力すると答えが求められ

▼リスト1　サンプルプログラム「loan.cbl」

```
      ******************************************************************
      *                                                                *
      *         ローン返済額計算サンプル(元利均等返済)                *
      *                                                                *
      ******************************************************************
       IDENTIFICATION          DIVISION.
      ******************************************************************
       PROGRAM-ID.             loan.
       AUTHOR.                 TOKYO-SYSTEM-HOUSE.
       DATE-WRITTEN.           2016/01/14.
       ENVIRONMENT             DIVISION.
       CONFIGURATION           SECTION.
       SOURCE-COMPUTER.        OPEN-COBOL.
       OBJECT-COMPUTER.        OPEN-COBOL.
      ******************************************************************
       DATA                    DIVISION.
      ******************************************************************
       WORKING-STORAGE         SECTION.
       01  WK-AREA.
         03  INTEREST-RATE     PIC 9(02)V9(03).
         03  INTEREST-YEAR     PIC 9(02)V9(09).
         03  KAKUNIN           PIC X.
       01  COMPUTE-AREA.
         03  PERIODS           PIC 999.
         03  PAYMENT           PIC 9(09)V9(09).
         03  LOAN              PIC 9(09)V9(09).
         03  INTEREST-MONTH    PIC 9(02)V9(09).
```

見出し部。ここから下にプログラム名や作成者、作成日などの情報を記述する

環境部。ここから下に実行環境情報や環境変数、プログラムが取り扱うファイル情報などを記述する

データ部。ここから下にファイル定義、ワーク変数、外部引数、画面定義などプログラムで使用するすべてのデータを定義する

次ページへ続く↘

↗前ページの続き

```
      ****************************************************************
       SCREEN                  SECTION.
      ****************************************************************
       01  SCR-AREA.
         03  LINE 01 COL 01    VALUE "LOAN CALCULATOR".
         03  LINE 02 COL 01    VALUE "LOAN AMOUNT: ".
         03  LINE 02 COL 23    USING LOAN
                               PIC   ZZZ,ZZZ,ZZ9  BLANK WHEN ZERO.
         03  LINE 03 COL 01    VALUE "ANNUAL INTEREST RATE: ".
         03  LINE 03 COL 28    USING INTEREST-RATE
                               PIC   Z9.999 BLANK WHEN ZERO.
         03  LINE 04 COL 01    VALUE "MONTHS: ".
         03  LINE 04 COL 30    USING PERIODS
                               PIC   ZZZ9   BLANK WHEN ZERO.
         03  LINE 05 COL 01    VALUE "MONTHLY PAYMENTS: ".
         03  LINE 05 COL 23    FROM  PAYMENT
                               PIC   ZZZ,ZZZ,ZZ9.
       01  SCR-KAKUNIN.
         03  LINE 06 COL 01    TO    KAKUNIN.
      ****************************************************************
       PROCEDURE               DIVISION.
      ****************************************************************
       HAJIME.
           INITIALIZE WK-AREA
                      COMPUTE-AREA.
       MAIN-000.
           DISPLAY SCR-AREA.
           ACCEPT  SCR-AREA.
       MAIN-100.
           COMPUTE INTEREST-YEAR   = INTEREST-RATE / 100.
           COMPUTE INTEREST-MONTH  = INTEREST-YEAR / 12.
      *  毎回の返済額 = (借入金額 * 月利 * ((1 + 月利) ** (返済回数)))
      *               / ((1 + 月利) ** (返済回数) - 1)
           COMPUTE PAYMENT =
                 (LOAN * INTEREST-MONTH *
             ((1 + INTEREST-MONTH) ** (PERIODS))) /
             ((1 + INTEREST-MONTH) ** (PERIODS) - 1).
       MAIN-900.
           DISPLAY SCR-AREA.
           ACCEPT  SCR-KAKUNIN.
       OWARI.
           STOP RUN.
      *--------------------<< END OF PROGRAM >>----------------------*
```

画面定義節。ここから下に画面への入出力情報を記述する

画面の1行1桁目から“LOAN CALCULATOR”という文字を表示させる

DATA DIVISIONで定義したLOANを入出力項目として使用する

LOAN項目を画面に表示するとき、ゼロサプレスとカンマ編集を行う。また、値がゼロのときは空白を表示する

DATA DIVISIONで定義したPAYMENTを出力項目として使用する

手続き部。ここから下にプログラムで実行する処理を記述する

WK-AREA、COMPUTE-AREAを初期化する

SCR-AREAを画面に表示する

SCR-AREAの画面から入力値を受け取る

計算された結果を反映し、再度SCR-AREAを画面に表示する

SCR-KAKUNIN項目の入力値を受け取る。通常は、ここで受け取る値（“Y”や“N”など）を判定し、繰り返すか終了するか分岐するのだが、このプログラムでは無条件に終了へ流れる

るようにしています。

リスト1をLinux環境でコンパイルします。

```
$ cobc -x loan.cbl
```

コンパイルが成功すると、カレントディレクトリに実行形式のオブジェクトが作成されます（**図9**）。このオブジェクトを実行します。

```
$ ./loan
```

すると、次のような画面が表示されます。

```
LOAN CALCULATOR
LOAN AMOUNT:          ----------
ANNUAL INTEREST RATE:      ------
MONTHS:                      ----
MONTHLY PAYMENTS:               0
```

ここで、借入額は「15000000」を、利率（年）には2016年1月現在の主要都市銀行最低変動金利である「0.625」を、返済回数は「420」（35年×

12ヵ月)を入力し Enter キーを押します(なお、項目の移動は TAB キーもしくは ↑ キー、↓ キーで行います)。

すると、1回あたりの返済額が表示されます。この場合は「39,772」となっています。

```
LOAN CALCULATOR
LOAN AMOUNT:          _15,000,000
ANNUAL INTEREST RATE:       _0.625
MONTHS:                       _420
MONTHLY PAYMENTS:          39,772
```

まとめ

opensource COBOLは、Linux環境でCOBOLの開発・実行を手軽に行うことができるOSSです。その昔、COBOLで開発をしていた方でも、これからCOBOLを使う方でも馴染みやすいものです。この機会にぜひ一度お試しください。

最後に告知です。opensource COBOLの次期バージョンのリリースが2016年春ごろに計画されています。それに伴い、新バージョンのリリース内容や事例紹介のセミナーも同じく春ごろに計画されています。日程などが決まり次第、OSSコンソーシアムのWebサイト[注6]で発表しますので、興味/関心のある方は参加してみてはいかがでしょうか。SD

注6) URL http://www.osscons.jp/

▼図9　実行形式のオブジェクトが作成される

```
$ ls -l
-rwxr-xr-x. 1 user user  18492 Jan 21 12:00 loan   ←実行形式のオブジェクト
-rw-r--r--. 1 user user   3387 Jan 21 12:00 loan.cbl
```

COBOLと文字コード

COBOLと最も相性の良い文字コードは何でしょうか?

COBOLはもともと、メインフレームなどの大型コンピュータで使うための汎用プログラミング言語として開発されました。そのメインフレームが標準としていた文字コードはEBCDICコードです。これは、半角文字と全角文字のバイト数が、それぞれ明確に半角は1バイト、全角は2バイトと決まっており、COBOLプログラマもそれを前提としたプログラムを作成してきました。

たとえば、COBOLのデータファイルは固定長を基本としていますが、並び替え(ソート)処理においては、バイトオフセット(xバイト目からyバイト目)で並び替えのキー値を指定します。また、データ項目の内部を参照する場合でもバイトオフセットを使用します(**リストA**)。

このようにCOBOLは、データのバイト位置を意識してプログラミングされているケースが多いため、バイトが可変となるUnicode(UTF-8など)とは相性が悪いと言えます。またEUCについても、半角カナ文字が2バイトとなるため、ここがEBCDICとは異なります。

その結果、半角、全角文字のバイト数がEBCDICとまったく同じであるShift_JISが、COBOLと最も相性が良い文字コードと言えます。

▼リストA　データ項目の内部参照例

```
01  IN-ADDRESS      PIC  X(20).
01  OUT-ADDRESS.
 03  OUT-ADDRESS-1  PIC  X(10).
 03  OUT-ADDRESS-2  PIC  X(10).

MOVE  IN-ADDRESS(1:10)   TO  OUT-ADDRESS-1.
MOVE  IN-ADDRESS(11:10)  TO  OUT-ADDRESS-2.
 ※COBOLのバイトオフセットは1から始まる
```

本質を見極める力

第3章 良いCOBOL、悪いCOBOL

Author 谷口 有近（たにぐち ありちか）
team Sirocco, LLC/チームシロッコ合同会社　代表執行役員社長　Twitter @arichika

COBOLがDisられるのはなぜか？　それは誤解が多いからではないでしょうか。本稿ではその原因を明らかにし、IT業界を生き抜くための知恵をそこから導出します。あえて本音で書き散らしますが、それゆえ辛口・口語調はご容赦ください。ガツンと行こうぜ!

真面目な話、COBOLの何が悪いのか。

COBOLは、COmmon Business Oriented Language=共通事務処理用言語というその名が表すように、始まりからして事務処理や帳票業務のプログラミングを宿命として生まれてきた言語です。最近のシステム開発は、COBOLが立ち上がった時代には想定していなかった技術や業務領域での開発が多数を占める状態になってきていますが、金融においてもFintech然（しか）り、そのトレンドは避けられるものでもなく、結果として、言語仕様上そんなことは考えてなかった的なCOBOL界隈の苦労がコストにも跳ね返っている状態が、筆者のまわりで観測されています。

当然、COBOLへの風当たりはいよいよ厳しく、業界向けネットメディアではもっぱらDisられるばかり。莫大な人月ベースでの改修予算を捻り出すための責任を押し付けられる、悲しい言語に成り下がっている感があります。でも、COBOLってそんなに悪い子なんだっけか？

COBOLの言語仕様の特徴として、自然言語に近い構文を持つとWikipediaには書かれています[注1]が、実際そのとおり、人語の様相そのままです。自然言語と同じように、処理系依存（地域）の方言があり、時代に合わせての拡張（若者言葉）もあれば、文脈依存で処理が振れる（ここではきものをぬげ）など、よく言えば実に多国籍で華やか、悪く言えばお隣さんとでさえ言葉は通じない的なノリ。

共通言語として世に放たれた仕様ですが、プラットフォーム依存するJavaやJavaScript、CSSの初期と同じように、実行環境の多様化と利便性追求という囲い込みのなか、共通仕様が分断されていく世界線がここにもあります。

至高のDSLとしてのCOBOL

もしDSL（ドメイン特化言語：Domain Specific Language）の究極 VS. 至高の対決を企画するならば、究極のDSLとも言えるUNIXシェルスクリプトに真っ向から勝負を挑む、至高のDSLとしてCOBOLを推薦したいわけですよ。COBOLこそ、ドキュメントDBなど差し置いて、真の意味で（紙の）伝票と台帳を管理するDSLであることに異論はないはず。

一例を挙げれば、Cの構造体での順序保証のもとになっているであろう[注2]、COBOL[注3]の数

注1）URL https://ja.wikipedia.org/wiki/COBOL

注2）興味深い議論は URL http://nanyanen.jp/comp/struct.html などに。

注3）Professor Charles Nicholas, The University of Maryland (2002) UMBC CMSC631 -- Fall 2002 Principles of Programming Languages Section 0101 URL http://www.csee.umbc.edu/courses/graduate/631/Fall2002/ より URL http://www.csee.umbc.edu/courses/graduate/631/Fall2002/COBOL.pdf が地味に面白い。RubyにはCOBOLの思想が受け継がれているという評価。

値型や集団項目[注4]の実装でしょう。ハードウェア側から見れば、当時の潤沢ではないリソース環境でのテクニックの駆使で、ただひたすらに感心し、モダン言語の富豪っぷりに感謝の気持ちを抱いてしまいます。

しかし、これを業務ロジックの実装側から見れば、IEEE754制定以前の世界で小数点以下の存在を前提とする10進数で上等な通貨を、誤差なく運用しなければならないという要求をふまえたうえで、日付情報の部分管理や階層化グループ化された各入力内容の操作に対するプログラミング上の便宜、さらには固定長電文としての管理上の利便性と性能の担保という機能があります。これはつまり、伝票の管理事務作業というプログラミング、要は契約台帳＝紙の束の管理というドメインに最適化された要素になっています。

当時の事務作業の機械化という行為は、事務作業のハードウェア化に直結していた時代です。伝票台帳の管理業務は、オープンリールなどの磁気テープメディアの管理とほぼイコールだったので、台帳管理に対する指示命令は、磁気テープや磁気ディスクに対する指示と同等です。伝票の印刷は連帳プリンタの操作と同等です。ゆえに、変数の値をコピーして計算するといった抽象化された演算命令と、台帳管理としてのハードウェア特性に依存するファイル操作やデバイスの指示が、それぞれに同じ抽象化水準として言語空間で同等に扱われているのが、事務作業DSLとしてのCOBOLの特徴です[注5]。

COBOLの歴史は、契約伝票と台帳としての管理の歴史とイコールです。手作業だった伝票管理事務作業が、公衆網の発達とともに電子化され、経済発展とともに地方拠点の増加とセンター業務との分業化に適したシステムデザインを生み、技術の進化がさらに高速で大容量のデバイスを登場させ、その環境変化に都度、伝票管理業務を適応させてきた英知の蓄積が、現在のCOBOL、至高のDSLなのです。

COBOL言語の成功

DSLであることは、事務処理数≒仕様書数≒実装（画面）数、という関係性を高い信頼度で維持して開発プロセスが進行することを担保しますから、仕様を単純化したり使い回すなりして業務仕様の整合性の観点でその品質を一定水準で維持できれば、機械が理解する命令への転記という作業のコストは、機能数や画面数、そして並行稼働可能な転記作業の人数をもって近似ができるわけです。

このことは、規模や工数をFP（ファンクションポイント）やステップでとらえるアプローチの信頼度を高めるので、見積もりのブレを抑えることにつながり、結果として安定した利益率を実現します。エンタープライズなプロジェクトマネジメントと大手SIerの爆誕ですね。利益率の向上には、仕様工程と転記工程での品質担保が必須です。製品の歩留まりを見るように、業務区分やコード行数、担当者とバグとの相関を観測し始めるわけです。コード転記作業の自由度は品質を悪化させる悪ですので、仕様書通りに書くことこそが正義です。

台帳管理業務の一部として、業務仕様書に書かれることが多い管理対象データの検索更新操作の手順[注6]ですが、これも、仕様書から命令セットへの転記として処理ができたほうが、DSLの役割としては適切です。RDBMSが登場した際にも、その操作を埋め込みSQLという形で、仕様書からコードの一部として容易に転記できるようになっていきます。

COBOLが伝票管理事務作業のDSLであるこ

注4） 無理矢理に今っぽく言えば、任意の場所を指定可能な名前付き添え字を持つスコープのないバイト配列、または、メモリ上の格納順序が保証されていて境界の踏み抜きが可能な名前付きでグローバルなタプル。どう考えてもバグの温床ですが、その固定長バイト配列のような発想は、GC（ガーベジコレクタ）を極力回避しなければならないような高負荷環境では今でも大切な設計視点。

注5） 抽象化のレイヤが少ないことそのものが、古い言語では時代背景的に一般的。MSX-BAISCでのVPOKE/VPEEKしかり。

注6） どの帳票を検索してどう更新するか、というCRUD。

とは強力かつ有益な個性です。その強烈な個性は、社会の電子化というトレンドのなか、方言バリバリの言語仕様を提供する汎用機メーカーと、方言だらけのSQL文法を提供することになるRDBMSメーカーにとって、自分達のしばしの成功を約束するものとなりました。

現場の実態と転換点

「仕様策定の工程に対して、プログラミングは製造工程である」という考え方[注7]がありますが、筆者は、前述したCOBOL言語（＝言い換えれば、その時代のプログラミングへの要請）がもたらしたプロセスとその標準化の成功がこの観点を決定的なものにした、という見方をしています。

製造工程である認識であれば、ミスのない製造をするためには仕様書から自動製造すべし、というアプローチが生まれます。業務仕様書に書かれた内容からCOBOLコードを自動生成するExcel VBAが現場で開発され、モジュール管理はAccessフォームで実現されていくわけです。秘伝のタレと化したExcelやAccessは、時としてOfficeのアップデートに抗い、そして誰かが破損させたであろうファイルを巡って軋轢（あつれき）を生むことになるのですが、一方で営業はこの便利道具が売り物になると気がついてしまい、高速開発ツールと銘打って売り出すわけです。業務仕様書のさまざまな観点での整合性を無視しながら。

「プログラミングは製造である」。この観点は、品質マネジメント規格ISO9001で要求される製造工程の観測や、開発と運用の分離という命題とも影響しあいながら[注8]、まさに今のSIの現場を形作っていきます。結果として、利益率の向上を実現し、経済成長を支えていくことになります。皆さんがお察しのとおり、インターネットが登場するまでは。

足枷となった標準化

製造である以上、業務仕様書をもとに具現化されたロジックは観測可能な指標を用いて数値化されます。たとえばステップ数とバグとの関係です。転記においてはステップ数≒製造時間であり、機械の稼働時間のように費用が計算され、人月が決定されていきます。プログラマの創意工夫は否定されるべきで、設計時点ですべてを決めることになります。高コストであっても利益の源泉になるからこそ再利用される前提の業務設計は、環境の変化を追従せずいつまでも蔓延（はびこ）る結果、おいてきぼりです。この世界では、業務設計は実装とイコールで、基盤の予算が実装規模と関係なく先に決定してしまいます。また、先に決まっているわりに業務仕様がそれを前提としないアホな設計だったりするなんてことも。そんなこんな分断されたプロジェクトでは出来上がってもまともに動くはずもなく、開発と設計と運用はいよいよ対立し、設計は業務設計しかできず、ゴミのようなシステムが生まれるわけです。

過去の開発現場では、相応のコンパイル時間を必要とするのが当時の当然でした。しかし、モダンな開発言語やプロセスで実現されていく世界の改造は、コンパイルを帰宅前に仕込んで翌朝結果を見ていた時代の速度とは、まるで違います。CPU速度は、開発言語や開発体制そのものの変化を誘発し、時代を変える速度さえも加速させているという現実に、我々は自覚的でしょうか。

誤解を恐れず、強い意志を込めて意見を表明したいと思いますが、COBOL時代に実現した成功プロセスが生み出した標準化により隠蔽された“何か”を意識せず、先人が当時に苦労して発明した手順上の理想をなぞった経験だけで自らを正当化した程度の人材が、いわゆるネットな今と、これからの時代に期待される技術主導

注7）ソフトウェア工学のうち生産工程として力点を置く分野など。
注8）機会があればここも掘り下げたいのですが、今回は省略。

によるビジネスアイデアや仕様の検討、そのような価値を売るサービス営業、そしてそれらの開発を管理職として工程を管理するようなことが果たしてできるものなのでしょうか。筆者は無理だと考えています。

変えられなかった価値観

PMやSEは技術を知らずして成り立つか、という命題もちょくちょくネットで取り上げられますが、COBOL時代のような製造工程管理としてのプログラミングで良い程度の案件なら、そりゃ技術なんぞ知らなくても心理戦で現場を押し切ってしまえばオンスケ達成なんぞできるぜよ程度の話ですわな、と。COBOL時代のノリで今も見積もりやら設計やらマネジメントやらを極めようとしてるから、そりゃぁクッソダサい画面の業務システムしか創れないし、そんなんでクラウドのシステムが組めないってさ、何言ってやがんの？　FP法でFacebookやらTwitterやら見積もってみやがれ、と本音では言いたくなるわけです[注9]。

非効率な現場をさらに混乱に陥れるような人材採用で毎月ウン十万円の出費は許されるのに、開発環境を改善する月額2万円程度クラウドサービスやアプリケーションの稟議は通らず、今や設計品質にさえ影響を与える開発者用の端末はなぜか非力なノートPCのままです。しかもPCの性能はコードを書かない管理職と同じかそれ以下です。さらに、担当範囲の進捗率をパーセント単位で日々のExcel日報で報告することをいまだに求められているという現状さえ、事実として存在しています。挙げ句の果てに業界団体は、「総工数（人月）＝係数A×画面数＋係数B×バッチ数」で開発工数との相関ガー、などというショーモナイ議論を飽きずにやっているのです。

ゴールありきで走り出したエンタープライズな案件のアンケート結果から推定したら、そりゃトートロジーでしょ。

COBOL“的”からの即時脱却を

COBOLは、時代背景を考えれば素晴らしい役割を果たした言語です。とはいえ、命令セット抽象化レベルが、今どきのインフラ構成や機能の分離とは一致しておらず一体化しているため、実行環境の制限の中で、開発後半での調整範囲は限定的でドラスティックに変えにくく[注10]、新規開発で選ぶことは絶対にない言語の1つだと思います。

とくに、継続的にメンテナンスし続けていくことが強く求められる業務システムでは、書く人で省略記法の癖が全然違う！　このNOTはどこまで係ってるんだっけか？　などという有様は当然ながら困るわけです。もちろん人手に任せればモダン言語でも容易に起こるこれらの問題、今ならコードライティングの時点で随時指摘してくれるアプリケーションも存在します。ではCOBOLでは？——実行時エラーでしか判断できないコードも容易に書けてしまう言語では、やはり厳しいでしょう。

それならば「Javaで開発だ！」としたところで、COBOL時代の成功パターンとしてのエコシステム[注11]を変えなければ、staticだけの無駄に長大なメソッド実装とクラス作成申請書による詳細設計管理のコンボで、その言語特性の恩恵を無視しているだけでなく、ドブに捨てているのと同じです。だからといって、大人数での並行開発なのに何でもアリにしてしまえば、あっという間にオブジェクト迷宮なOOPの出来上がり。1行変えたら全サービス死亡みたいなノリも容易に起こる無法地帯になります。

ここに、我々が注目しなければならない事実があると考えています。COBOL時代の開発の

注9）言っちゃってますけどね。

注10）書けばわかる……直そうとすればわかる……変えようとすればわかる……。

注11）開発プロセスだけでなく、仕様決定プロセスから基盤の発注管理まで含めたシステムマネジメントのライフサイクルとしての生態系。

常識は、今やその大部分が非常識なアプローチである、そう言い切っても過言ではないと思います。しかしながら、シリコンバレーで流行ってる、有名なあの人が言ってた、バズってるのでこれ言っておけば先端っぽいだろ的なノリで突入してしまうと、そりゃもうモダン言語でモダンテクノロジーだろうが、システムが容易に死ねるわけです。

いやね、メモリ足りてないドキュメント指向データベースにログぶち込むとか、ぐにゃっとした業務システムをオサレだからと関数型で開発してみたりとか、コスト構造が高止まりの背景を顧みず、改竄不可な低価格ストレージだ！と言ってみたり、安くなると言われてIaaS上にシステム載せ変えて性能が出ないとか、もう屍の山ですよ。

COBOLこそDevOpsへ！

COBOL言語を前提に、よりよく書こうとする行為そのものは、クラウド時代になった今でも、プログラミングにおける要点の観点で、それほど変わらないのも事実です。最適なインデックスでデータを当てて、少ないリソースでデータを処理し、効率的なデータの変換操作を目指す——ハードウェアリソースが限られていた時代だからこそ、そのコードの書き方は、クラウド的な分散処理の発想と似ています。埋め込みSQLは、その抽象化を進化させていけば、O/Rマッパー、そして統合言語クエリ（LINQ[注12]）のようなアプローチにもたどり着きます。

「要求仕様に合わせ、確実に設計し、あとあと手戻りしないようプロセスを管理する」ことと、「変わり続ける仕様を前提に手戻りしても変えやすいように確実に構造を設計しプロセスを回す」ことは、どちらもプロジェクトの成功を目指しているものの、決定的に異なる知識と環境、判断の上に成り立ちます。

継続的インテグレーションの概念が育っていない時代に成功した開発体制や道具の使い方のままで改修を続けていれば、そりゃテスト費用も改修費用も人力総当たりですし、囲い込みのど真ん中ですから、コストに跳ねかえって当然です。それが嫌なら、違う成功パターンを目指していかなきゃなりません。そのためには、発注元の覚悟も要求される時代になっているわけです。それがDevOpsであり、アジャイルスクラムの思想でもあるわけです。

成功体験を捨てろ！

COBOL言語としてイケてないから脱却しなきゃならないなんてことはありません[注13]。古い時代のCOBOL言語を前提とした仕様検討と開発の体験、そして囲い込みに見事にはめ込まれてしまった経営的課題を解決しようとせず、結果としての高コスト構造のCOBOL界隈から逃げ出すためだけに改修するなど、いかほどの費用をかけようとも、それはすでに形骸です。あえて言いましょう！　カスであると！　脱却すべきは、COBOL時代に習った成功体験です。

COBOLが抱える課題は業界の歴史そのものです。一時代を確実に築き上げた言語からは学ぶことが多いはず。同じ失敗を他言語でも踏み抜かないよう、精進していこうじゃありませんか。SD

注12）URL https://ja.wikipedia.org/wiki/統合言語クエリ

注13）古い古いってね、モダンなCOBOLならオブジェクト指向＋マルチスレッドで書けるんですわ。いや、そりゃ、だからといって書きたいとは思わないんですがね、エンジニアのライフプランとして……。

壁を越える力を身に付けよう

第4章 COBOLから別のプログラミング言語を習得するときのヒント

Author 吉谷 愛（よしたに あい）フロイデ㈱ 代表取締役

皆さん、こんにちは。フロイデ㈱の吉谷 愛と申します。筆者はちょうど10年以上前に、エンジニア人生をCOBOLプログラマからスタートさせ、その後Visual Basic/Java/C＃.Netにキャリアチェンジしたのち、PHP/Rubyを習得しました。現在は経営者兼技術講師という肩書です。本稿では講師経験をもとにCOBOLプログラマが別の言語を習得するノウハウを紹介します。

COBOLから別の言語に移った理由

筆者がCOBOLからその当時Visual Basic/Java/C＃.Netといった今どきな言語を使うプログラマにキャリアチェンジしようと思い立った理由は、「これからCOBOL案件は減少する。他のプログラミング言語を習得しないと生き残れない」というまわりの話を聞き、それを真に受けたからでした。加えて当時のメインフレーム開発で一般的に行われている仕事のやり方や慣習に漠然とした疑問を抱いていたことも挙げられます。

そのあと、幸か不幸かあまり学習しないままCOBOL以外の言語を使う現場に移ることができましたが、「漠然と疑問を抱いていた仕事の慣習」は以前のままでした。具体的には、次のようなものです。

1. 大規模なシステム開発のメンバ（プログラマ）という位置付けのため、自分のコードが誰の役に立っているのか把握できない
2. 1行で済むコードの仕様を、半日かけて設計書に書き起こさなければいけない
3. 一度テストして納品したあとに、仕様に欠陥があることがわかり、修正が必要なのに、上長からの指示なしでは対応できない
4. 実際にプログラムを動かしてみればすぐにわかるのに、テストエビデンスとして画面のハードコピーを毎回とり、Excelのシートに貼り付けねばならない

これらが筆者にはどうしても理不尽に感じられ、SNSやブログなどで目に入る「Web系」と言われる人たちがとてもキラキラして見えました。そして彼らの「ソフトウェアエンジニアを大切にする」という価値観が非常に魅力的に感じ、独学でWebプログラミング言語であるPHPやRubyの習得に取り組むようになりましたが、それらを習得するのはたいへんでした。その一番大きな理由は、Webプログラミング関連の書籍が、読者がWebの基本的な知識をすでに持っている前提で執筆されていたからです。

きっとCOBOLからWebプログラミング言語の習得で苦労されている方も、「COBOLだから」だけではなく、暗黙的に「Webの知識」を求められるところで苦労されているのではないでしょうか。それならWebプログラミング言語の勉強をする前にWeb技術の勉強をすればいいのか——と思われるかもしれません。が、ご存じのようにWeb技術は、非常に幅広く深いため、業務の片手間に独学をするのはなかなかたいへんです。そこで、仕事と並行して無理なくWeb系言語（ここでは、PHPを例として取り上げます）を習得するためのノウハウを本稿では紹介します。

COBOL技術者がPHPを習得する際に躓きやすいポイント

COBOLとPHPを紐づけながら学習しようとする

これは筆者が最初にやった失敗です。たとえば、COBOLの「DISPLAY」とPHPの「print」など、ごく簡単な命令語であれば問題ありません。やはりPHPなどと比べるとCOBOLは非常に冗長なプログラミング言語です。PHPにはIDENTIFICATION DIVISION.に該当する構文はありません。逆にCOBOLにはジェネレータ機能がないので、無理にyieldキーワードに該当する機能を探そうとしても結び付けられません。そこで無理に紐づけようとすると、COBOLにないプログラミング言語のメリットを理解しづらくなったり、変な勘違いを起こしたりする恐れがあります。

プログラミング言語だけにフォーカスして学習しようとする

PHPのようなWebプログラミング言語を習得する場合に使用する言語は、(そういうただし書きがないまま)Webの基本的な知識が前提となっていることがあります。その場合、プログラミング言語の文法自体は何となく理解できても、その全体像はもちろん、その書籍が本当に言わんとしていることが理解できないままとなってしまう可能性があります。COBOLエンジニアにCOBOLの文法だけではなくメインフレームやJCLなどの知識も必要なのと同様です。

一言一句完璧に理解してから先に進もうとする

たとえば「一言一句完璧に理解しながらPHPを勉強しよう」と思いたったあなたが、同僚のPHPエンジニアに「PHPの開発環境を作るにはどうすればいいか」と質問したとします。おそらくその方は「XAMPP」を使った環境構築を勧めるでしょう。そこで「XAMPPとは何なのか？」と問えば、「PHPインタプリタはもちろん、ApacheやMySQLや、そうそうphpMyAdminなど、いろいろなフリーのソフトウェアやライブラリをパッケージとしてまとめたものだよ」と答えるでしょう。そこで「インタプリタ？　Apache？　MySQL？　phpMyAdmin？」と聞くと、きっと「後はググってくれ」と言われておしまいです。そこであなたはPCを立ち上げて検索します。すると「HTTP」や「Webサーバ」についての"膨大な量の"記述を見つけます。もしかしたら、その過程であなたはWebテクノロジの深淵に少しだけ触れられるかもしれません。しかし、そんなふうにXAMPPについて完璧に理解するまで、XAMPPのインストーラをダウンロードしないのであれば、あなたの最大の目的であるはずの「PHPの開発環境を作るにはどうすればいいか」という疑問の解決はかなり遠くなることでしょう。

Web技術は非常に深く広く豊穣であり、情報量もまた膨大です。そしてそれがすさまじい速さでアップデートされています。それを理解し、さらに支配しようとしても、なかなかうまくいかないでしょう。

なぜCOBOLプログラマは壁にぶつかるのか？

実は筆者たち技術講師にとって、「COBOLからPHP/Rails/Java講座」は難易度の高い案件になります。なぜかと言えばアンケート結果が、他の講座に比べ今一つ辛い評価になりがちだからです。思い当たる理由はいろいろありますが、講師たちと話をしているとよく「技術の説明をしているのに、なぜか設計など業務の話に結び付けてくる方が多い」という話になります。理由としては次のものがあるのではないでしょうか。

1. 自分の実業務と紐づけることで理解を深めようとしている
2. 実装(プログラミング)より業務知識が最優先だと考えている

1. に関しては、アジャイル開発のように、「PHPを使用したWebアプリケーション開発では普通に行われているけれど、メインフレームを使用する開発ではめったに行われない開発手法」があります。その開発手法を理解しないまま実業務に無理やり結び付けようとすると、前述したように混乱する一因になります。

2. については一概には言えません。ただ、一般的にWebアプリ開発では、いわゆる上流工程を担当する設計者にもWebの技術知識や開発・テスト手法、インフラ／ネットワークやプログラミング言語などの、より「Webエンジニアリング」な知識を幅広く求められることが多いようです。またメインフレーム開発に比べ、非常に短い期間で要件定義・設計・製造・テストを行うため、役割をあまりはっきり区別しないこともあります。要件定義をしたエンジニアがそのままプログラミングを行うことは、往々にしてあるようです。

COBOLエンジニアが効率よくWeb系言語を習得するコツ

COBOL言語はいったん忘れましょう！

前述のように、COBOLとWebプログラミング言語を無理に紐づけて理解しようとすると、かえって混乱することが多くなります。いったんCOBOL言語は忘れましょう。新人が最新の技術をあっという間に習得できるのは脳が柔らかいだけでなく、そういう「混乱をきたす紐づけ」自体ができないのでそのまま素直に吸収せざるを得ないというのもあります。

「勉強する」ではなくWebアプリを作りましょう！

とにかく最初は「勉強」というスタンスではなく「豊穣なWebアプリ技術の恩恵にあずかる」という気持ちで、素直に工作を楽しむようにWebアプリを作ってみてください。「理解する」ことを目的とせず「Webアプリを作って動かす」ことを目的にしましょう。とにかくよくわからないままでも作ることで、0だった知識は1になります。COBOLエンジニアの方はエンジニアだからこそなのでしょうが、真から理解することで安心し、かつ深い満足感を得る傾向にあると思います。その気持ちは元COBOLエンジニアとしてとても共感できます(**図1**)。

▼図1　習うより慣れろ！　まずは実践あるのみで作ってみましょう!

しかし、Webアプリ開発に関しては、最初は「習うよりも慣れろ」だと割り切って、まず1つWebアプリを作ってデプロイし、自分のスマートフォンやPCで自分の作ったWebアプリが思ったように動くかどうかを、試してみるところから始めてください。そして「では今度はこんな機能を追加できるだろうか？」と実際にそれを試してみましょう。要は「勉強」を目的とせずに手段とするのです。英語を習得したいなら日本語の通じない海外に行けば嫌でも覚えるとよく言います。その場合、英語は「目的」ではなく、その土地で生きていくための「手段」です。同様に「Webアプリを作る」ことを目的とすれば、自ずと「技術習得」は、そのための手段となります。

Webアプリ構築時は「デプロイ」までやってみる

デプロイとは、レンタルサーバやクラウドに、ローカル環境で作成したWebアプリを配置して

インターネット経由などで使用可能な状態にすることです。デプロイすることでインターネットに接続されたPCやスマートフォンから構築したWebアプリを利用できるようになります。自分で考えて作ったWebアプリを、自分や友人のPCやスマートフォンなどで確認できると、それだけで確実にモチベーションが上がります。それと同時に「よくわからないままチュートリアルに沿ってWebアプリを作ってみたけれど、なぜこれで動くのだろう？」という疑問もわくはずです。その時は疑問に思った部分をしっかり調べてみてください。きっと自分でも驚くほど効率よく知識を吸収することができるようになっているでしょう。

Webアプリ開発技術習得のコツ

繰り返しになりますが、とにかく効率よく技術を習得するコツは、手を動かしてWebアプリを作ることに尽きます。それも「Hello, World」のようなものではなく、シンプルでもデータベースを使用したCRUDアプリケーション[注1]を、クラウドにデプロイするところまでやってみましょう。

Web開発の現場で活用できるCOBOLエンジニアの知識や経験

個人的な感想ですが、ソフトウェアテストを含む品質保証に関するノウハウは（実際は現場によって違うのでしょうが）メインフレーム開発のほうがWeb開発よりもしっかりしていると思います。最初、筆者がメインフレームからWindows系のアプリ開発に移った際、メインフレーム開発で行っていたテストをすると――「時間がかかり過ぎ」といわれたものの――品質で高い評価を受けました。それはWindows系からWebアプリ開発に移った際も同様でした。そこで筆者は基本的なWebの技術と言語を習得した後は、自分のテストの知識を捨てはせず、Web開発の中でも比較的テスト領域内もしくはそれに近い「テスト自動化」や「TDD（テスト駆動開発[注2]）」などを学ぶことで、少しずつWebアプリ開発の世界に入っていくことができました。

スティーブ・ジョブズ氏の『connecting the dots』をご存じでしょうか？

「先をあらかじめ見通して点を繋ぐことはできない。振り返って、繋ぐことしかできない。だから将来何らかの形で、その点が繋がると信じることだ。何かを信じ続けることだ。直感、運命、人生、カルマ、その他何でも。この手法が私を裏切ったことは一度もなく、そして私の人生に大きな違いをもたらした」

あなたがCOBOLや汎用機で習得した知識や経験は、まさにここでいう「dot」です。Webプログラミング言語学習で得た知識も「dot」です。無理にCOBOLとWeb系のプログラミング言語という「dot」をつなぐのではなく、いずれそれらがつながると信じて学習していってください。

私事で恐縮ですが、筆者自身もCOBOL技術を習得してWeb系に転じたときは、まったくそれらがつながるなどとは考えてもいませんでした。ところがテスト技法だけはなく、メインフレーム開発から離れて十年以上たって「COBOL技術者向けRails講座」を考案し、それが2年前に福岡県主催の「フクオカRuby大賞」の奨励賞を受賞しました。その結果COBOLエンジニアの方向けのWebアプリケーション技術習得セミナーの企画をさせていただく機会が増えるようになるとともに、筆者の会社の知名度も向上しました。

筆者はこのような結果を想定して、COBOL

注1） CRUDアプリケーション：一般的なアプリケーションに必要とされるデータのCreate（登録）、Read（読込）、Update（更新）、Delete（削除）の基本的な機能を持つアプリケーションのこと。

注2） TDD（テスト駆動開発）：アプリケーション開発の際、まず最初にテストコードを書き（これをテストファーストと言う）、そのテストコードを実行するために必要な実装を必要最低限な範囲で行い、そのたびにテストを実行しながら少しずつコードを洗練させる工程を繰り返すスタイル。近年はとくにアジャイル開発におけるXP（エクストリームプログラミング）で取り上げられることが多い。

エンジニアになったわけでもWebアプリ言語を習得したわけでも決してありません。技術習得活動が自分の中の「dot」を作り続けていくことであるなら、学んだことを活かすために必要なのは、“dotをつなげることをあきらめないこと、自分のdotがつながっていくと信じる心を持つこと”だと筆者は思います。

おわりに

実際にメインフレーム開発からWebアプリケーション開発にキャリアチェンジされた方とお話しすると、現場の「スピード感」に付いていけないと感じる方が多いようです。毎週毎週サービスをデプロイし続ける「アジャイル開発」に違和感をいだいていた方もいました。今までデプロイする際には厳重なクロスチェックを経る必要があったり、顧客からの仕様変更の必要が発生しても上長の承認を得るまではソースに触れることができなかったりするような現場に携わっていた方からすれば、それは当然でしょう。ただ、COBOLエンジニアの方がWeb系の技術を習得すること自体は、今までお伝えした事項に気を付けていただければ、そんなに難しいことではありません。実際筆者が受け持った受講生のほとんどの方が、そこまで無理せず習得できています。

メインフレーム上で稼働しているCOBOLで構築されたシステムをWebアプリに置き換えるマイグレーション案件は、おそらく今後も出てきます。また、すでに完成され安定した稼働を保証するCOBOL資産を活かし続けるために、COBOLでのシステム開発も行われ続けるでしょう。

「Webアプリケーションに置き換えるべきか、それともこのままCOBOL資産を使用し続けるか」

そこで適切な判断を行うことができる「メインフレーム／COBOL」と「Webアプリケーション／Webプログラミング言語」、その両方の知見を持つエンジニアの需要は、今後もますます高まるでしょう。SD

お勧め書籍

短期集中講座
土日でわかる
PHPプログラミング教室
――環境づくりからWebアプリが動くまでの2日間コース
（http://www.amazon.co.jp/dp/47973827）

Webアプリケーション言語を習得するには、まず小さいWebアプリを何か作ってみるのが一番です。ですが、実際に未経験者が1つWebアプリを作成するのはなかなか難しいでしょう。そこで未経験者向けに、

- 徹底的に実践型の「このとおり作ればちゃんとうまくいく」という、成功体験が得られる
- 実際のWeb開発の現場で使用する本格的なツールを使っている
- 最低限必要な機能を最低限のコードに絞り込んで最低限の時間で完了できる

という本があれば良いのにと思いました。

そして筆者は、未経験者でも実際に業務で使ううえで必要な最低限の情報に触れながら、土日の2日間（約16時間）でデータベースを使用したWebアプリケーションを完成できるようにこの書籍を執筆しました。

もともとはシステム開発未経験者を意識して作成しましたが、現在、メインフレームや制御からのキャリアチェンジ希望者の方やWebデザイナ、HTML/CSSコーダーや元プログラマのマネージャ・経営者の方からも高い評価を受けています。

Appendix

著者自らが自著を解説

COBOL書籍が必要とされる背景と読者のニーズ

Author 細島 一司（ほそしま ひとし） トータルセブン 代表

毎年、春になると売れるCOBOLの書籍。入社や異動で、COBOLで開発している部署に配属された方々が購入しているようです。そんな顕著な傾向が見られるのも、COBOLを使う開発現場がまだ多いからでしょうか。そんなCOBOL書籍を多数執筆している著者に、COBOLへの思いや、執筆時の考えを述べてもらいました。

執筆のきっかけ・経緯

――たくさんのCOBOL書籍を執筆なされていますが、きっかけはどういう経緯からでしたか？

COBOL書籍を初めて手掛けた1999年は、メーカーを退職して独立した年でした。

退職前に筆者も所属していた情報処理学会傘下のJIS規格COBOL作業委員会の委員長から「COBOLの本を書いてみないか？」とお話をいただいたのがきっかけでしたね。出版社も発売時期も決定していたので、ほかの仕事をそっちのけで書いていた思い出があります。執筆に必要なソフトウェアや資料などは委員長の所属会社である㈱日立製作所のCOBOLチームのみなさんから提供を受けて、全面協力って感じでした。

――COBOL書籍に需要があると思われましたか？

当初は思いませんでしたね。というのも、プログラミングの世界ではオブジェクト指向技術が定着しつつありましたし、世の中では「COBOLはもう時代遅れでしょ」とささやかれていましたから。しかし、COBOL全盛だったころの膨大なソフトウェア資産がまだまだ枯れることなく金融業界、官公庁や文教市場などで稼働していましたし、当時の技術者が退職で年々不足していく中、いかにしてレガシーシステムを維持していくかが企業内の検討課題であったので、そこに注力すれば「きっと需要がある」と自分に言い聞かせて執筆していました。

おかげさまで、最初の書籍が増刷を繰り返して順調に売れていたため、複数の出版社から執筆依頼があり、スキマ産業的なCOBOL本ですが、結果的に需要があったと思っています。

――著書は語り口調的な文章ですが、なぜこのような手法を取り入れたのですか？

手法というほどの高尚なものではないのですが、COBOLの本というと、お堅いイメージがあるじゃないですか。古くからある言語というのはだいたい同じ感じになってしまいがちです。文法書的な。そこで、入社以来、COBOL一筋で培った技術と経験をそのまま、やわらかくわかりやすく本にしてみようというコンセプトと、学生時代は家庭教師や塾講師、企業時代には臨時講師をしていたので、目の前に読者がいるイメージで書いていたら、あのようになってしまいました。

――すべての著書はANS85を中核とした第3次規格で記述されていますが、その意図はどのようなものですか？

第4次規格からオブジェクト指向に突入していくわけですが、新規で、COBOLで、オブジェ

クト指向で、というよりはレガシーCOBOLをなんとかしたい(しないといけない)という声が読者から多数ありました。大学の講義テキストや企業の新入社員研修テキストにも採用されているのですが、いずれも今後というよりは、今までのCOBOLのメンテ人材養成という観点で使用されていることから、ニーズに応えた結果です。

それぞれの本の特徴

――それぞれの書籍についての特徴と思い入れなどがありましたら教えてください。

標準COBOLプログラミング[注1]、標準COBOLプログラミング 第2版(図1)

最初に執筆したCOBOL書籍で、読者層のターゲットを絞り込まずに文法基礎から開発現場で必要になる実務内容まで網羅したオールマイティ的なものにしました。時期的に2000年問題対策中であったのと、開発現場について執筆された本が珍しかったのか、その需要は予想以上でしたね。

第2版は、読者の声を反映して、使用頻度の高い文法の例題を増やしたり、演習問題を精査したり、COBOL規格環境が変わったりで、その対応をした書籍で、基本路線は初版と変わっていません。

▼図1　標準COBOLプログラミング 第2版(細島一司(著)、カットシステム、2014年発行)

"演習問題で基礎から学ぶ"やさしいCOBOL入門(図2)

前書籍での反応で、情報処理試験対策用、学校教材、独学用として読まれている方から、「ページ数が多くて読み終わらない」「実務レベルはいらないから基礎中心で」といった声が結構あったので、基礎を中心とした学習用ワークブック形式の書籍を書きましょうとなりました。学習した内容を演習問題の穴埋め式で補完するというストーリーですね。発行から相当年数が経過していますが、息の長い書籍です。

▼図2　"演習問題で基礎から学ぶ"やさしいCOBOL入門(細島一司(著)、カットシステム、2000年発行)

開発現場で役立つCOBOLプログラミング入門[注2]、開発現場で役立つCOBOLプログラミング入門 第2版(図3)

COBOL教本としてこれまでの2冊で「完結かな」と考えていましたが、世の中では現場スキルを持ったCOBOL技術者が退職などでどんどん少なくなっていて、その代替とする社員を教育するにもお金も時間も余裕がない、文法も重要だけれども開発現場に特化した書籍がほしいと

注1) 細島一司(著)、今城哲二(編)、カットシステム、1999年発行、10刷、2014年絶版

注2) 細島一司(著)、秀和システム、2008年発行、3刷、2013年絶版

▼図3　開発現場で役立つCOBOLプログラミング入門 第2版
（細島一司（著）、秀和システム、2013年発行、2刷）

▼図4　COBOLポケットリファレンス
（細島一司（著）、技術評論社、2012年発行）

いう要望を経て実現した書籍です。

この本を読んで「手軽に」「すぐに」確認ができるようオープンソースのOpenCOBOLコンパイラの適用やデータベース接続にOracleを選択するなど「動作」を意識した構成と、現場の必須技術となる指南書項目を章立てしたのが喜ばれていますね。

COBOLポケットリファレンス（図4）

周知されているポケットリファレンスシリーズのCOBOL版です。通常の書籍は結構大きいので、システムエンジニアやプログラマが毎日、カバンで持ち歩くのはたいへんですよね（かさばるし重いですし）。このシリーズは小さくて携帯性が優れているのと、出先で手元にマニュアルがないときにサッと出せるところが特徴です。1ページ、見開きで調べたいことが完結できるように心がけました。

電子ブック対応したのもこの本からですね。もともとの本が小さいのでページ字数が少ないためか、スマートフォンに入れているという読者も多いです。

読者の反応

――何か裏話的なものはありますか？

読者からのアクションとしては、文法質問は学生の方からが圧倒的に多いですね。また、現場からの相談事などコンサルティングに近い質問もあれば、仕事上の愚痴やよもやま話などもあります。「景気の高低で質問って変わるんだなぁ」と実感したり、「COBOLをやっている人たちって変わっているのかな？」なんて思ったり、いろんなことがありますが、執筆っておもしろいなぁと。ただ、書いているときはお金が一銭も入らないのがつらいですが（笑）。

――学校でプログラミングを教えているとお聞きしましたが、執筆と講義では何が違いますか？

執筆は本の向こう側の見えない人たちを意識して書きますが、講義は目の前ですからね。その辺の難しさはありますが、執筆しているときと同じように、自分の持っているものをすべて出して、現場の話を入り交ぜて講義していますね。

今年度はJava、Android、JavaScript、PHPのプログラミングを講義しましたが、言語が変わると如実に拒否反応を示すのですよ。言語の特性はあるにしろ、プログラミングは「手段」であって、その根本は「何も変わらないんだよ」と、年間をかけてわかってもらえば良いかなと。執筆とは違う、講義でしか得られないものもあるので、それを含めて次の執筆に展開しようと思っています。SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2016年2月号

第1特集
［最新］MySQLとPostgreSQL徹底比較

第2特集
1Gbps超ネットワーク高速化時代の適切なLANケーブリングの教科書

一般記事
・Android Studioのスタイルで効率アップ!

定価（本体1,220円+税）

2016年1月号

第1特集
はじまっています。ChatOps
導入を決めた7社の成功パターン

第2特集
手軽さとコード化しやすさが人気!
Ansibleでサーバ管理構成を省力化

新連載
・Androidで広がるエンジニアの愉しみ

定価（本体1,220円+税）

2015年12月号

第1特集
［決定版］Docker自由自在
実用期に入ったLinuxコンテナ技術

第2特集
ネットワーク・システム管理の定石
SNMPの教科書

短期連載
・クラウド時代のWebサービス負荷試験再入門

定価（本体1,220円+税）

2015年11月号

第1特集
すいすいわかるHTTP/2
HTTP/1.1から変わること・変わらないこと

第2特集
攻撃を最前線で防ぐ
ファイアウォールの教科書

特別企画
・SMB実装をめぐる冒険
File System for Windowsの作り方

定価（本体1,220円+税）

2015年10月号

第1特集
多層防御や感染後対策を汎用サーバに実装
攻撃に強いネットワークの作り方

第2特集
Webメールの教科書
クラウドサービス利用か？　自社で構築か？

特別付録
・創刊300号記念　Vim&Emacsチートシート

定価（本体1,220円+税）

2015年9月号

第1特集
特講
正規表現・SQL・オブジェクト指向
苦手克服のベストプラクティス

第2特集
メールシステムの教科書
日本語もバイナリもちゃんと届くのはなぜか

特別企画
・なぜ俺の提案は通らないのか?

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。

iPad Proのさきに見えてくるもの

Dynabook, Newton, Palm, Zaurus, Pocket PC, Tablet PC and ...

Author 清水 亮(しみず りょう)
(株)ユビキタスエンターテインメント

手書きコンピュータの聖杯伝説

今年はApple Pencilが主役になる年かもしれない。現状はiPad Proでしか使えないが、今年のアップデートでiPad AirやiPad mini、iPhone 7やiPhone 7 Plusといった端末で使えるようになると、Apple Pencilの活用範囲はぐんと広がることになる。

実際にAppleがそれをやるかどうかは、いつものようにまったくわからない。やるかもしれないし、やらないかもしれない。しかし確実にひとつ言えることがある。

Appleはひとつチャンスを失ったということだ。

そしてこのチャンスは、もしかするとAppleだけでなく、我々人類が永久に失ってしまったかもしれないチャンスなのだ。

何のチャンスか？　それはコンピュータが紙に取って代わる、その先頭を切り開くチャンスをAppleが失ってしまったことを意味するのだ。

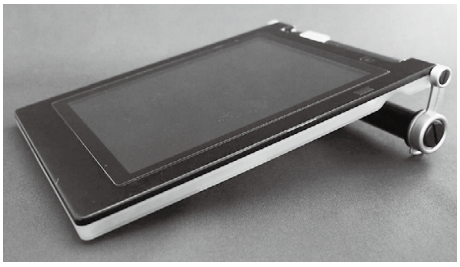
スティーブ・ジョブズが見落としたケイのコンセプト

筆者らが開発した手書きプログラミング端末enchantMOONを最初にアラン・ケイに見せに行ったとき(**写真1、2**)、アラン・ケイはビジュアル・プログラミングのコンセプトとして参考にすべき事例としてGRaILのデモを見せてくれた(一部は、https://www.youtube.com/watch?v=QQhVQ1UG6aMで見られる。**写真3**)。

GRaILは1968年ころ、米軍の研究機関のひとつ、RANDコーポレーションが開発したペンをベースとしたビジュアル・プログラミング環境である。ペンでフローチャートを記述していくだけでプログラムの全体構造が記述でき、さらに深く潜っていくと最終的にはアセンブリ言語までペンで操作できるという画期的なものだった。

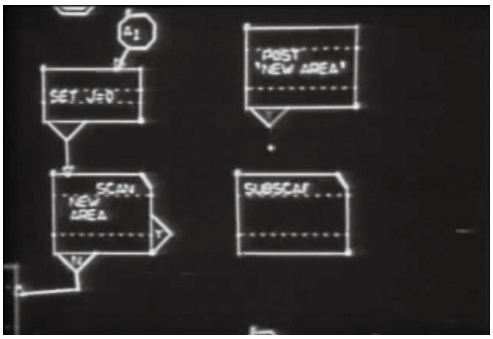
GRaILはGRaphical Input Languageの略と

▼写真1　筆者らの開発したenchantMOON



▼写真2　アラン・ケイとenchantMOON

▼写真3　GRaILの画面



引用元：YouTube（「Alan Kay Demos GRaIL」作成者：Vincent Gable）

▼写真4　初期のMacintosh

されているが、「聖杯（Holy Grail）」を意識したネーミングであることは疑いようもない。当時からすでにペンによるコンピューティング（とりわけプログラミング）は、コンピュータの究極の利用法と位置づけられ、いつかはたどり着かなければならない境地であると考えられていたのだろう。

もっとも古いペンコンピュータは、アイヴァン・サザーランドのSketchpadである。このSketchpadに刺激を受けたアラン・ケイによって、「パーソナル・コンピュータ」というコンセプトが産まれた。ケイが1970年代に描いたビジョン、「Dynabook（ダイナブック）」コンセプトでは、今見るとiPadそっくりの端末に、子供が自由に絵を描くことが前提となっている。

このコンセプトを暫定的に実現した環境のひとつがSmalltalkとAltoであり、これを見学に来たスティーブ・ジョブズとビル・ゲイツがそれぞれLisa、Macintoshと、Windowsを作ったことはあまりにも有名だ。

ちなみに暫定ダイナブックコンセプトの実装環境はAlto（およびSmalltalk）だけではない。ごく初期のプロトタイプとして、可搬型のものもあった。当時の技術水準を反映してディスプレイはあまりに小さく、お世辞にも操作しやすいものではなかったようで、ポインティングデバイスとしてはマウスが用いられた。これは専（もっぱ）ら画面サイズの制約だった。これはXerox NoteTakerとして知られている。

スティーブ・ジョブズは、Lisaでアラン・ケイの背中を追いかけ、こだわりすぎてLisaのチームを追放され、さらにジェフ・ラスキンの率いたMacintoshチームを事実上乗っ取って、のちにMac OSと呼ばれるOSの最初のバージョンを完成させた。最初のMacintoshは商業的には失敗したが、ジョブズが追放されたあと、Macintoshは成功への道筋を歩み始めた（**写真4**）。

1995年、Appleに復帰する前に行われたスティーブ・ジョブズへのインタビューで、ジョブズは興味深い証言をしている。

「アラン・ケイのAltoを見たとき、最初はそのグラフィカル・ユーザ・インターフェースに目を奪われてもっと重要なことを見落としていた。それはオブジェクト指向と、ネットワークだ」

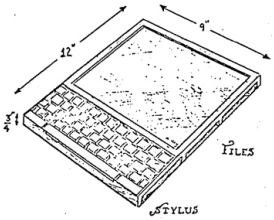
そしてネットワーク機能とオブジェクト指向を大胆に取り入れたNeXTを作り（**写真5**）、これが今のMac OS XやiOSの直系の先祖になったことはよく知られている。実際、よくできたOSである。いまだにOS Xのシステムオブジェクトが NSで始まるのは、OSがNeXTSTEPと呼ばれていたことの名残である。

しかしジョブズはもうひとつ見落としていたことがあった。

それはジョブズが見て、直接影響を受けたAltoおよびSmalltalkは、あくまでもより大き

▼写真5　筆者とNeXT Cube

▶写真6
ケイの描いた初期の
ダイナブックのスケッチ。
右下にスタイラス
（STYLUS）の記述がある



なビジョンDynabookの暫定的な実装に過ぎないということだ。そして完全なDynabookには、ペンが必要とされていたのだ。なぜならケイはDynabookを子供でも“プログラミングで”自分の考えを表現したり確かめたりできるツールとして定義しており、初期のコンセプトスケッチにも、右下にスタイラスを格納するためのスリットが描かれている（**写真6**）。

そしてデジタルペンは文字どおり聖杯となった。数々の人間がデジタルペンのコンセプトに魅了され、挑み、そして散っていったのだ。

PalmTop ComputerとNewton

ジョブズがAppleを追放され、NeXTを発表した直後、日本の雄、SONYは、それまでにないまったく新しいコンセプトのコンピュータを打ち出した。それが、PalmTop Computerである。

PalmTop Computerは1990年に発売された。きわめて野心的な製品で、当時としては先進的な手書き文字認識はもちろん、通信のための音響カプラまで内蔵されており、FAXの送受信もできた。ただし、残念ながらこのPalmTop Computerは成功したとは言い難かった。コンセプトは優れているものの、製品の中に閉じた世界になっていた。

当時、すでにコンピュータとは、豊富なアプリケーションがサードパーティから多数提供されるものであり、成功するコンピュータとは、多数のサードパーティを味方に引き入れたものに限られていた。この分野で当時最も成功していたのはNECのPC-9801シリーズであり、ゲーム機であればファミリーコンピュータだった。いずれもサードパーティの活気が肝だ。

そうした世界観に比べると、PalmTop Computerの出現はあまりに異質すぎた。当時はバブル絶頂期ではあったものの、20万円という高価格もあり、サードパーティを巻き込みひとつのエコシステムを作るまでには至らなかった。

それから時間をおかずして、ジョブズを追放した当の本人である、ジョン・スカリーは自分が単なるコーラ屋ではなく、コンピュータの歴史に英雄として永遠に名を刻むために、ひとつのコンセプトに取りつかれていた。

それが「パーソナル・デジタル・アシスタント」という思想である。パーソナル・アシスタントといえば個人秘書を意味するが、間にデジタルが入ることによって、これが電子化される。

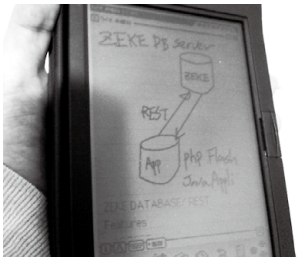
スカリーはまず豪華なコンセプトビデオを作らせた。とある地質学の教授が、パーソナル・デジタル・アシスタントを内蔵したノート型のコンピュータと会話しながら仕事を進めるというものだ。

このコンセプトビデオは極めて異様だった。非常によくできているが、現実感がどこにもない。英国訛（なま）りでしゃべる執事（バトラー）のようないでたちで表現されたパーソナル・デジタル・アシスタントを、スカリーは「ナレッジナビゲーター」と名付けた。

ナレッジナビゲーターは、ペンが使える。というよりも、一見すると紙のノートのようにい

▼写真7
筆者のNewton MessagePad 2100

▼写真8
Newtonで書いたメモ



▼写真9
Newton向けの路線図アプリ

ろいろなことが書き込めるようになっており、必要に応じてデジタル・アシスタントと会話することで仕事を効率的に進められるようになっていた。

今このビデオを見ると、このビデオの完成から30年を経た現在であっても、これを実現するのはあまりに難しいということがわかるだろう。これは短編SF映画と言っても過言ではなく、現実のテクノロジーから激しく乖離（かいり）している。そのうえ、どう考えても、現在の通常のOS、つまりややこしいデジタル・アシスタントの介在しない普通のOSのほうが、はるかに効率的に同じような仕事をできるように思えてならないのである。

しかしスカリーはこのコンセプトビデオを単なるコンセプトで終わらせるつもりはなかった。彼が満を持して投入したのが、パーソナル・デジタル・アシスタント製品のNewtonである（**写真7**）。

Newton OSとはいかなるものであったか。まず、Newtonはすべての PDAの元祖である。なにしろ「パーソナル・デジタル・アシスタント」を縮めたものがPDAなのだから、こういってしまっても乱暴ではないだろう。

筆者は実際にNewtonを入手し、使ってみたことがある。モノクロの画面に感圧式のペンがついていて、これで文字を書くことができる（**写真8**）。たいていのことはペンのジェスチャーで指示する。書き損じを消したいときはグチャグチャっとペンで書けば、紙をくちゃくちゃに丸めたようなエフェクトが出て、データが消える。新しい項目を書きたいときは、画面を横切るようにまっすぐに線を引くと、新しいノートが引いた線の下にできるという感じだ。これは使っていてなかなか楽しく、小気味よい。

さらに特筆すべき機能は、デジタル・アシスタント機能だ。

「I've a dinner with Mary next friday
（次の金曜日にメアリーと夕飯をとる）」

こんな感じで手書きメモをして、Newtonの「デジタル・アシスタント（Assist）」ボタンをタップすると、住所録からMaryで始まる人物のリストを表示して「どのメアリーですか?」と聞いてくる。特定のメアリーを指定すると、それがそのまま次週の金曜日の夕方7時くらいの予定に設定される。つまりノートでありながらスケジューラーやアドレス帳とも連動しているのだ。

Newtonは開発者たちを興奮させた。このほかにも開発者たちを興奮させるしかけはいくつもあった。たとえばIMEそのものを加える機能。NewtonのIMEは単なる手書き文字認識だけではない。ときには図形を認識させたり、筆ペンのようになる機能もあった。

Newtonがデフォルトでは日本語に対応してないことに業を煮やした在日オーストラリア人が、自ら日本語対応FEPを開発し、それが逆にアップルジャパンのオフィシャルになるなどの快挙も果たした。そのほかにも無数のアプリが開発された（**写真9**）。つまりNewtonは非常にハッカブルであり、開発者たちを夢中にさせるのに十分なプラットフォームに思えた。

第一弾のNewton MessagePadは非力すぎるCPUに貧弱な文字認識エンジンのせいであまり実用にならなかったが、そうした問題もプロセッサの進歩が少しずつ解決していった。1997年ごろに発売されたNewton MessagePad 2000になると問題の大部分は解決され、実用になるかと思えた。しかし思い出してほしい。このころのAppleは控えめに言っても瀕死の状態だった。開発者コミュニティに愛されたNewtonはAppleの膨大な赤字を支えるほどの利益を生み出すことができなかった。

そしてスティーブ・ジョブズが復帰するやいなや、Newton関連のプロジェクトはすべて中止された。もちろん当時のAppleには利益に貢献しない余計なことをやっている暇はなかった。Appleが産み、育てようとしたペンコンピューティングの世界を自ら閉ざしたのは、ほかならぬスティーブ・ジョブズだったのである。

Palm

スカリーの後を追い、自ら聖杯を探し求める者も少なくなかった。Newtonプロジェクトにもかかわった日本のシャープは手書き対応電子手帳として液晶ペンコム「ザウルス」シリーズを展開し、一定の成功を収めた。

ジェフ・ホーキンスが立ち上げたPalm社も成功した企業のひとつである。Palm社の提供するPalm OSは複雑で巨大な計算リソースを必要とするNewtonとは異なり、コンパクトで必要最低限の機能だけを提供するシンプルな設計を採用した。

特筆すべきは手書き認識をGraffiti(グラフィティ)という簡易な一筆書きに制限してしまったことだ。こうすることで慣れたユーザはストレスなく文字入力ができるようになり、Palmは本家のNewtonを差し置いて大成功を収めた。

Palmもまたハッカブルであり、標準アプリと同等の機能を持つ「置き換えアプリ」と呼ばれるサードパーティ製アプリが大いに盛り上がった。フリーウェアはもちろんのこと、シェアウェアなどの商流も盛んであり、コンパクトな手書きコンピュータを求める人々にとってほとんど唯一の選択肢となった。

Palmは2008年にiPhoneが登場するまで、モバイルインターネットの主役であり続け、しかし残念ながら、2008年にiPhoneが登場して以来、すっかり影を薄くしてしまっている。2010年にヒューレット・パッカードが買収し、新OSとして「webOS」を発表し、短い期間脚光を浴びたが、2011年にヒューレット・パッカードはwebOS関連機器からの撤退を表明した。以後、2013年にLGにライセンスしたというニュース以来、歴史の表舞台からは姿を消してしまった。

ビル・ゲイツ、三度の挑戦

Microsoftが「ペンコンピューティング」の聖杯を求めて挑戦したのは、少なくとも3回以上はある。

1回目の挑戦は、Newtonと同じく1992年、「Windows for pen computing」である。これは、Windows 3.1をベースとしたペンコンピューティング用のOSで、鳴り物入りで登場した割には人々の話題にもあまり上らなかった。その後、このpen computing関連のコンポーネントはWindows 95以降ではOS本体に統合された。

2回目の挑戦は、2000年、先行するPalmに強く影響を受けたWindows CE 3.0ベースのPocket PCである(**写真10**)。これはほぼPalmを完コピしつつWindows(のようなOS)が携帯機で走るということで、一定数のファンを獲得したが、メインストリームには至らなかった。

Microsoftの主力製品群であるOfficeもMicrosoft Pocket Officeとして、機能制限版ながらWord、Excel、PowerPointなどが移植された。しかしこうした挑戦の数々は「こんなに小さいモバイルマシンでもWindowsやOfficeが使えることの感動」よりも、逆に「こんなに小さいモバ

イルマシンなのになぜWordやExcelを使わなければならないのか」という疑問のほうをむしろユーザに強く問いかける形になった。実際、実用に耐えることはほとんどなく、筆者もかなりの種類のPocket PCをかなりの期間愛用していたが、Pocket Officeが実用的に使えたと感じたことは残念ながら一度もなかった。

おそらくビル・ゲイツとしては三度目にして最後の挑戦が、Windows XP for Tablet EditionとTablet PCコンセプト、そしてMicrosoft OneNoteの三点セットである。

そもそもMicrosoft OneNoteのタイトルは、通常のOffice製品から逸脱している。通常、Office製品は一語で終わるものが圧倒的に多いのに対し、OneNoteだけが二語になっている。なぜMicrosoft NoteではなくOneNoteなのか。噂によればOneNoteはビル・ゲイツがアーキテクトとして最後に作らせた製品であるといわれる。それは彼が、それまでの人生の集大成としてペンコンピューティングを自らのOSに取り込みたいという強い意志の表れだったと考えることもできる。

ゆえにペンコンピューティングは聖杯なのである。

OneNoteは、それまでのOfficeアプリケーションとは異なり、キーボードとペン入力を区別しない。どちらの情報も対等に扱えるよう工夫されている。階層的なフォルダ構造を持つことが可能で、整理しやすいため、根強いファンも多い。そしてビル・ゲイツはついにペンをNewtonやPalmのような、OSのネイティブな特徴としてではなく、OneNoteという閉じられた空間の中でだけ使えるように封じ込めることに成功した。

ビル・ゲイツの執拗(しつよう)なまでのペンへのこだわりは、いまだにMicrosoft Surfaceの最新機種にペンが用意されていることからもうかがえる(**写真11**)。もちろん、ビル・ゲイツはもはや引退して久しい。しかしレドモンドには、いまだにビルの亡霊が聖杯を求めて彷徨(さまよ)っているのである。

Courier ――消されたプロジェクト

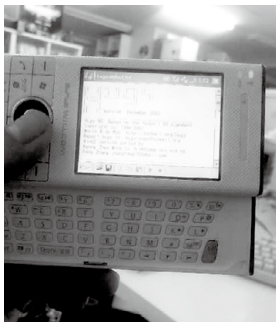
ビル・ゲイツの亡霊ともいうべき、ペンコンピューティングへの聖杯を求める者は、レドモンドの内部にもいた。

2008年、Microsoft内部の研究所、Microsoft Research(以下MSR)でひそかに開発されていた画期的な製品コンセプトが発表された。それが「Courier(クーリエ)」である。

Courierは2つ折りの本のような形をしており、開くことによって起動する。マルチタッチスクリーンとデジタイザを備え、ユーザは紙のノートにメモするのと同じような感覚でそれを操作することができる。試作機はWindows CEをベースとしたカスタムOSを搭載し、NVIDIAの高性能なモバイルGPUであるTegraを内蔵していた。

この試作機のデモビデオは世界に衝撃を与えたが、Microsoftは2010年にこのチームを解散させ、開発終了を宣言してしまう。なぜわざわざ実験プロジェクトの終了を宣言したのかという意図は不明だが、Microsoftという会社は内部に無数の優秀なエンジニアやアイデアを抱えな

▼写真10
Windows CEを搭載したW-ZERO3[es]



▼写真11
Microsoft Surface Book。やはりペンが同梱されている

がら、結局はWindows中心主義という原理主義に回帰せざるを得ないジレンマを抱えている。このほかにも数多くの有望なコンセプトやプロジェクトが歴史の影で葬られてきた。

Courierのチームは MSRを解雇された後、Fifty-three社を立ち上げ、iPad向けの画期的なお絵かきアプリ「Paper」を開発する。そして皮肉にも、このPaperが、iPad ProとApple Pencilのキラーアプリの1つでもあるのだ。

Paperが示す理想とアプリの限界

Paperは、Courierチームの残党が立ち上げたFifty-three社の代表的なプロダクトである。そしてApple Pencilの登場で最も光り輝いたアプリでもある。

Paperはその名のとおり、iPad Proを紙そのもののように扱いやすくする(**写真12**)。機能は徹底的にシンプル化され、洗練され、余計なものを足さず、かといって必要なものまで取り除かない。Courierコンセプトにあって Paperにないものは少なくないが、それはそれで正しい選択だったと思えるような哲学と美学をユーザはPaperというアプリを通じて感じることができるだろう。

Paperはある意味で完璧なアプリに思える。そしてまた同時に、iPad Pro、そしてApple Pencilの限界を示すものでもある。Paperは、元がCourierという独自コンセプトのOSだっただけに、ひとつの世界観の中で完結した美しいアプリになっている。

しかし同時に、これはAppleが許容する「アプリ開発者の世界」の境界線を示すものでもある。Paperはどこまでいってもアプリであり、Newtonのように開発者がワクワクするようなハックを施すものでもなければ、ザウルスのようにコミュニティによって進化するものでもない。ただひたすら、Fifty-three社が自社のアプリをアップデートするという手法によってのみ、発展し続けることを許されたクローズドな環境に過ぎないのだ。

いまどき、それが何らかのアイデアであれ、そうでないものであれ、クローズドな世界の発展はたかが知れている。Paperで創りだされるコンテンツは、Adobe Photoshopで創りだされるコンテンツと同程度に多いかもしれないが、サードパーティ製プラグインで拡張可能なソフトウェアプラットフォームでもあるPhotoshopに比較すると、Paperの示す世界観は窮屈に感じる。

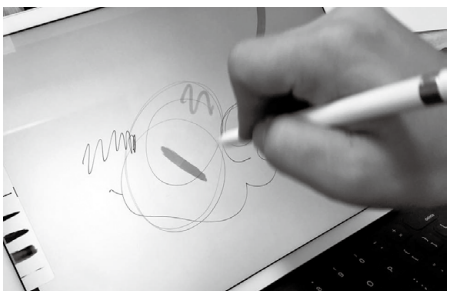
しかしこれこそがAppStoreで頒布されるソフトウェアの限界であり、Appleの無垢(むく)にして邪悪な欲望——市場すべてを意のままにコントロールしたいという欲望——を暗示している。

OS Xを含む従来のPC向けOSがどちらかというと“悪意のあるプログラムは作られない”という性善説に基づいて作られていたとすれば、iOSを含むスマートフォン／タブレット用OSは性悪説、すなわち“人は生来的に悪徳なプログラムを作り、ユーザの個人情報を盗み出すものである”という立場で設計されている。

携帯電話のようにリテラシーの低いユーザが圧倒的多数を占める世界では、もちろんそれがトラブルを最小化する方法として有効なのは間違いない。しかし筆者からすると、そうした発想そのものが少々傲慢(ごうまん)にも思える。アプリケーションを檻に囲い込むということは、アプリケーションの未来の可能性を閉ざすことをも意味するのではないだろうか。

Apple Pencilの登場は、Appleにとって再び彼らが聖杯を手にする最短のチャンスだった。

▼写真12　iPad ProとApple Pencil



しかし彼らはビル・ゲイツと同じようにペンを矮小化し、単なるOSのオプション機能としてしまった。筆者も含めてこの決断に少々失望したファンは少なくないのではないだろうか。もう二度と、Newtonは戻ってこないのだ。

聖杯を求める旅の終着点

ダグラス・エンゲルバートは初期の光ペン(GRaILで利用されているような)には根本的な問題があることを指摘していた。まず、垂直に近い画面に対して光ペンを保持する姿勢は無理があるし、画面が操作するペンで隠れてしまう。ダブルクリックなどのマウスでは当たり前のテクニックがペンでは極端に難しくなる、などである。では果たしてペンはいらないのか、間違っているのか。

初期のタッチスクリーンが登場したとき、エンゲルバートはそれが光ペンと同様の問題を持っていることを指摘した。しかし現在、タッチスクリーンではなくマウスまたはトラックボールで操作するスマートフォンが欲しいなどと本気で言うユーザがいるだろうか。ペンが不便なら、これほど多くの人が未だに紙のノートを完全に捨てられない理由はなにか。単純にそれはコストか、重さか、それとも別の何かか。

マシンの処理速度やユーザとの距離感、使用環境、そういったものが刻々と改善されていった結果、ようやく我々は聖杯を再び追いかけることのできる場所にやって来たと言っても良い。

最新のハードウェア、たとえばMicrosoft Surface BookやTOSHIBAのREGZA TabletやdynaPad、そしてもちろんiPad ProとApple Pencil、ここまで環境がそろってくると足りないものは自ずと見えてくる。ソフトウェアプラットフォームだ。

Windowsにしろ、iOSにしろ、Androidにしろ、ペンを前提として組み立てられたプラットフォームはひとつもない。頼みの綱のPalmでさえ、webOSではペンを前提とすることをやめてしまった。

ようやく一般向けに販売されたSONY Digital Paperがなぜいまいち盛り上がらないのか。SONY Digital Paperの軽さは十分である。大きさも、A4サイズで十分過ぎると言って良い。ところがSONY Digital Paperを実際に活用するには想像力が必要だ。想像力なしでこれを使いこなすのは相当な忍耐が必要だし、そのために10万円というプライスタグを適正と考えるユーザがどれだけいるかは疑問が残る。いっそシャープの電子ノートのように電子ペーパーではなく白黒液晶のほうが追従性がいいのにという気もする。

ユーザに想像力を要求する製品は、一般には「難しい製品」とうけとられがちである。そしてユーザの想像力を掻き立てるような小回りのきいた機能を実現するには、本来はそうした製品はソフトウェアプラットフォームでなければならない。Newtonやザウルスがそうであったように。そこに小さくとも開発コミュニティがあり、実用性が高いものも低いものも、大小さまざまなソフトウェアが開発可能であり、流通可能であり、場合によってはそれ自体を生業にすることが可能でなければならない。

その意味では、手書きコンピュータのソフトウェアプラットフォームとして現在までに生き残っている環境はひとつもない。かつて惜しいものはたくさんあったが、今はどこにいったかもわからない。

しかし聖杯は今、ようやく手の届きそうなところに来た。

これまで、手書きはあまりにも黒魔術的であり、それをコンピュータが理解するためには手書き文字認識をはじめとする人工知能関連技術を使いこなすことが必須だった。しかし現在、人工知能関連技術、とりわけディープラーニングが急速に進歩を遂げている。

こうした知見を取り入れれば、あるいは聖杯はついに人々の手に取り戻されるのだろうか。

SD

Webサイトが改ざん！
サイトオーナがとるべき行動と注意点

Author
宮本 尚志（みやもと ひさし）

Webサイトオーナを悩ます外部からの悪意ある攻撃。幾重にもセキュリティ対策を行っていることかと思います。しかし、万一改ざん被害に遭ってしまったら……。そのときになって慌てない、そして間違った対応をしてしまわないために、本稿で手順を再確認し、自社の対策作りに役立ててください。

ウチのWebサイトがマルウェアを配布!?

ここ数年、改ざん被害を間接的に見ることが多くなりました。

「間接的に見る」というのは、昔のように「どこかの国旗がぱたぱたとはためく画面」のように「これみよがしに目立つ」改ざんではなく、Webサイトにアクセスしたときに「ウイルススキャナが反応する」ような形で改ざんに気付くことを指しています。

ウイルススキャナが反応する理由は、Webアクセスの延長でマルウェアに代表される「悪意あるコンテンツ」がダウンロードされているためなのですが、このようなことに至る理由はいくつかあります。

①Webサイトが直接提供するコンテンツもしくはWebサーバの設定が改ざんされ、Drive-by Download攻撃の拠点にされた

具体的には、Webブラウザでサイトにアクセスしてきたとき、当該Webブラウザが稼働しているコンピュータに、マルウェアに代表される「悪意あるコンテンツ」をダウンロード・実行させるようなしくみを入れ込まれた状態です（**図1**）。

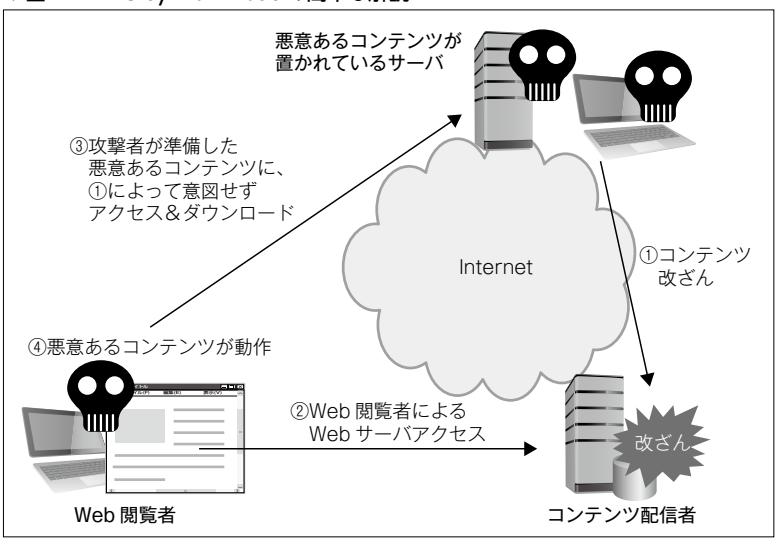
②マルバタイジング[注1]に代表されるような手法で、広告などWebサーバが本来提供する「以外」のコンテンツにマルウェアをダウンロードするようなしくみを入れ込まれた

①とほとんど同じですが、インターネット広告で配信される内容に悪意あるコンテンツをダウンロードさせるような記述が行われれば、Webサイト改ざんがなされなくてもDrive-by Downloadが成立します。

②については、Webサイトのオーナ側では対処が

注1　マルウェアをはじめとする不正なソフトウェアの拡散や、不正なサイトへのリダイレクトなどといったことを行う悪質なオンライン広告を用いた攻撃のこと。

▼図1　Drive-by Downloadの簡単な解説



難しい部分もあるのですが、①は比較的対応がしやすいものになります。それでは、Webサイトオーナが改ざん被害に気付いたときにどのような対応を取るべきか？について、説明を始めます。

なお、本稿では分量の兼ね合いもあり、具体的な方法やケーススタディには触れません。その代わり、読者のみなさまの環境でどのようなことを行えるかの検討ができるように、どのようなツールがあるか？についても触れておきますので、参考にしていただければ幸いです。

◎注意とお願い◎

攻撃の検証などは、自分が責任を持てる環境のみで！

以降、実際にどうやって対処していくか？という話を展開するわけですが、話の流れから「どのような形で攻撃を実現可能か」という例を示すことがあります。しかしこれは、あくまで例示であり、読者のみなさまに対して攻撃行為を推奨するものではありません。試す場合には、壊れてもいい環境で試し、自分以外のところに迷惑がかからないようにしてください。そうでないと罪に問われることもありうるので、攻撃検証などはあくまで自分が責任を持てる環境で実施するようにしてください。

Webサーバの改ざん被害に遭ったときはどうするの？～基本的な考え方と手順

Webサーバの改ざん被害は前述のとおり比較的目にする状態になっていますが、改ざん被害に遭遇した際に取るべき手順は大まかに言って次のようになります（**図2**）。

- Webサイトの一時停止／閉鎖を行い、被害の拡大を防ぐ
- Webサイトをホストしているコンピュータを保全し、改ざんされた状態のデータを確保する
- 改ざんされている／不正に置かれているコンテンツを特定する
- コンテンツが改ざんされた／不正に置かれた時刻を特定する
- 不正に置かれた時刻周辺でシステムに起こったことを確認する
- 復旧する
- 管理下にある他のWebサーバでの確認を行い、改ざんされていたら上記の手順を繰り返す

Webサイト改ざん対応～改ざん対応は、基本的に気付いてから始まる

改ざんされてしまったWebサイトを指して「なぜ改ざんされたのか」とか「こうすれば改ざんされなかったのに」という方は多いですが、じゃあ「どう対応するか」という話まで踏み込む人はそれほど多くありません。

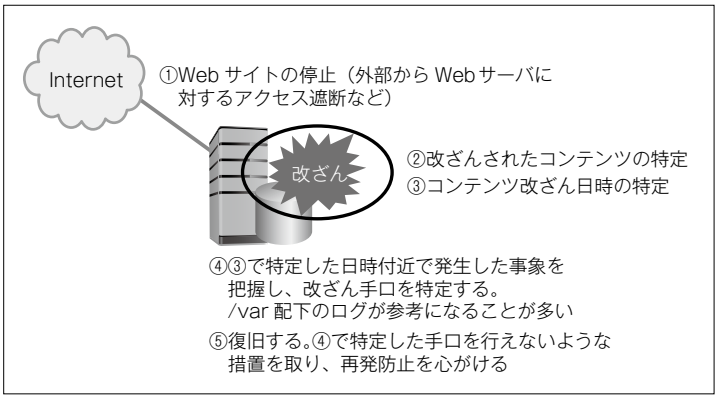
情報セキュリティマネジメント上、「抑止」－「防止」－「検知」－「回復」というフェーズがありますが、改ざんを認識するところは、抑止も防止もすっとばして、検知というところが走っています。このため、Webサイト改ざんからの回復は「検知に始まるサイクル」が走ると考えるのがよいでしょう。

なぜ改ざんされるのか？～モチベーションはさまざまだが、手法は限定される

攻撃者がWebサイトの改ざんを行う理由はさまざまですが、最近よく見られるのは「マルウェア感染させてから何かをする」というようなものであり、「何か」はたいていの場合、情報収集やお金儲けというところにつながることのように見えます。そんなWebサイトの改ざんですが、大きく次の2通りの手法で実現されます。

- Webサイト上のコンテンツを改ざんし、よろしくない内容を入れ込む

▼**図2　復旧の手順**



- Webサイトの設定を改ざんし、よろしくない内容を入れ込む

Webサイト上のコンテンツ改ざんの場合、静的ファイルに保持されたコンテンツを改ざんされるケースと、WordPressなどCMSで管理されたコンテンツ(静的ファイルもあれば、DBMS上に保持されるコンテンツもある)を改ざんされるケースがあります。

レアなケースとして、Webサイトの名前解決を行うDNSを書き換えたり、ドメインを乗っ取ってDNSそのものを攻撃者が管理するものに変更したりというようなものもありますが、これはどちらかというとWebサイトの書き換えではなく、すでに攻撃者が用意したWebサーバに被害者を追い込むような手法なので、本稿では述べません。

改ざんに気付くトリガ〜自分で気付くか、他者に指摘されるか

いずれの場合も、Webサイトにアクセスした場合に提供されるコンテンツが本来意図したものと異なったものになっています。このような場合、自分で気付くか他者から指摘されるかのいずれかで被害が発覚しますが、改ざんを検知するためのしくみを導入していない場合、他者から指摘されるケースが多くなります。

自分で気付く例

- 改ざん検知システムを導入していて、通知がWebサイト管理者に送られてきた
- メンテナンス時に見慣れないファイルが置かれているのを確認した
- メンテナンス時に見慣れない内容がコンテンツに書かれているのを確認した

他者から指摘される例

- 外部からの連絡先を設置している場合、その窓口に対して改ざん時に連絡される
- CSIRTを設置している企業の場合、当該CSIRTの連絡窓口に対して連絡がある

ビッグウェーブに乗り遅れるな〜改ざんの入口や兆候のつかみ方

ここでは改ざんの入口や、どのような兆候が見られるのかについて解説します。

改ざんの入り口は脆弱性?〜実は結構あるWebサイト改ざん

改ざんされたWebサイトのオーナの中には、「なぜウチが改ざんされるのか?」という疑問を抱く方もいらっしゃることでしょう。でも、オーナの事情など一切勘案してくれないのが改ざんを行う者であり、攻撃者です[注2]。さらに、改ざん結果によっては、改ざんされたWebサイトが攻撃に加担させられることにつながり、被害を受けるだけではなく他のユーザに対する加害者にもなりえます。

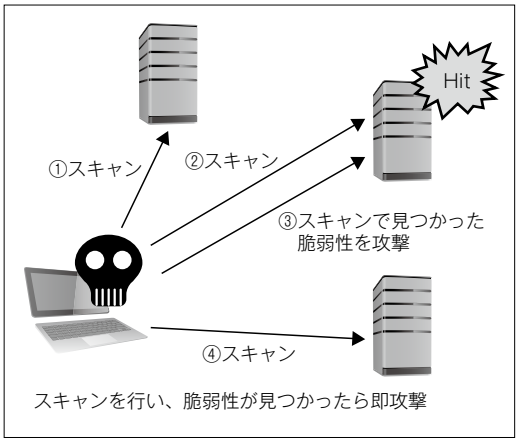
- 特定の脆弱性を保有するか否かを無差別にスキャン
- 特定の脆弱性を保有するWebサイトをピンポイントで攻撃

もちろん「スキャン→ヒット→攻撃」というのを繰り返す方法もあるでしょうが(**図3**)、まず高速にスキャンを実施し、脆弱性を保有することがわかったところをピンポイントで攻撃するというほうが効率はよいでしょう[注3](**図4**のように、とりあえずスキャン時のレスポンスは保存しておいて、後でまとめて結果を処理してリスト化し、脆弱性のあるWebサイトをまとめて攻撃するほうが、出回っているツールの組み合わせで普通にやりやすいです)。もしくは脆弱性の有無にかかわらず、いきなり脆弱性をトリガするようなリクエストを投げてくることも想定されます。

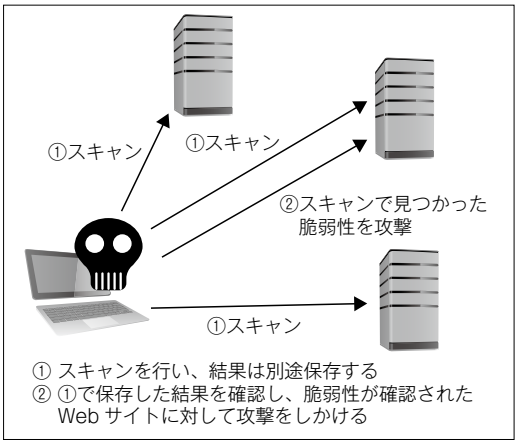
そのほかに、コンテンツ書き換えに必要なIDとパスワードを別のマルウェアによって窃取されたり、脆弱なIDとパスワードを使っていて不正アクセスを受けたりするパターンも意外とあります。数

注2 勘案してくれるような攻撃者がいたら、それはそれでコワイです。
注3 ZMap(https://zmap.io/)などの高速スキャンツールを用いて広範囲に情報を収集し、あとでゆっくり攻撃対象を見定めて攻撃するなど。

▼図3　攻撃者の取るアプローチ1



▼図4　攻撃者の取るアプローチ2



年前に流行した「Gumblar」などは、Webサーバのコンテンツを更新するPCにマルウェアを感染させ、コンテンツ更新に必要な情報を当該PCから抜き取って、攻撃者がコンテンツを改ざんするというものでした。

ログに残る攻撃の痕跡～Webサーバログやの認証ログを例に

改ざんに限りませんが、誰かが何らかの攻撃をWebサーバにしかけた際に、特徴的なログが残ることが多いです。OSの脆弱性、CMSの脆弱性、パスワードリスト攻撃、ブルートフォース攻撃のそれぞれで特徴が異なります。

◆OSの脆弱性

OSの脆弱性を突かれた場合、当然ですがどこを突かれたかによって残るログが異なります。OSカーネルの脆弱性を突かれた場合とOS上で動作するサービスの脆弱性を突かれた場合でも、ログの残り方が異なります。

◆CMSの脆弱性

CMSの脆弱性を突かれた場合は、当該脆弱性を突いた際の特徴的なリクエスト文字列が残ることがあります。この場合、CMSのログもさることながら、Webサーバのログに特徴的な文字列が残ることが多いです。

また、CMSの脆弱性を探索するようなツールもあります。たとえばWordPressの脆弱性をスキャンするWPScanというツールがありますが、動作させた場合にはWebサーバのアクセスログには**図5**のように現れます。このとき、WPScanで脆弱性が見つかると、WPScan側にはその旨出力されます(**図6**)。**図6**の出力結果を見ると、「バージョンの特定」「当該バージョンで存在する脆弱性」「当該脆弱性が修正されているバージョン」が★印部分に表示されていることがわかります。

◆パスワードリスト攻撃

パスワードリスト攻撃が行われた場合は、攻撃を受けたサービスの認証ログに攻撃の痕跡が残ることが多いです。またCMSの仕様により、1回のリクエストで多くの認証試行が行われていることもあります。

◆ブルートフォース攻撃

ブルートフォースには、「ID固定でパスワードを複数チャレンジする」ものと、「パスワード固定でIDを複数チャレンジする」ものがありますが、ログイン画面でブルートフォース攻撃を行う場合には、多くの場合は認証失敗の記録が多数残されます。これもまたパスワードリスト攻撃と同じく、CMSの仕様によっては1回のリクエストで多くの認証試行が行われることもあります。

図7は、筆者が管理するコンピュータの1つに残された、SSHの認証失敗ログの一部です[注4]。ありが

注4　Debian GNU/Linuxの場合は、/var/log/auth.logに残されます。

▼図5　WPScanが残すWebサーバログ(一部抜粋)

```
192.168.44.131 - - [16/Jan/2016:18:50:01 +0900] "GET /wordpress/ HTTP/1.1" 200 8000 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:02 +0900] "GET /wordpress/ HTTP/1.1" 200 8000 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:02 +0900] "GET /wordpress/wp-content/plugins HTTP/1.1" 301 571 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:02 +0900] "GET /wordpress/robots.txt HTTP/1.1" 404 459 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:02 +0900] "GET /wordpress/readme.html HTTP/1.1" 200 7448 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:02 +0900] "GET /wordpress/wp-includes/rss-functions.php HTTP/1.1" ➡
500 185 "http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:02 +0900] "GET /wordpress/wp-content/debug.log HTTP/1.1" 404 469 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:03 +0900] "GET /wordpress/.wp-config.php.swp HTTP/1.1" 404 467 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:03 +0900] "GET /wordpress/wp-config.php.bak HTTP/1.1" 404 466 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:03 +0900] "GET /wordpress/wp-config.orig HTTP/1.1" 404 463 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:03 +0900] "GET /wordpress/wp-config.php~ HTTP/1.1" 404 463 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:03 +0900] "GET /wordpress/%23wp-config.php%23 HTTP/1.1" 404 464 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:04 +0900] "GET /wordpress/wp-config.php.save HTTP/1.1" 404 467 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:05 +0900] "GET /wordpress/wp-config.php.swp HTTP/1.1" 404 466 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:07 +0900] "GET /wordpress/wp-config.php.swo HTTP/1.1" 404 466 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
192.168.44.131 - - [16/Jan/2016:18:50:08 +0900] "GET /wordpress/wp-config.php_bak HTTP/1.1" 404 466 ➡
"http://192.168.44.207/wordpress/" "WPScan v2.8 (http://wpscan.org)"
```

▼図6　WPScan出力結果(一部抜粋)

```
root@kipple:~# wpscan http://192.168.44.207/wordpress/
_______________________________________________________________
        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|

        WordPress Security Scanner by the WPScan Team
                       Version 2.8
          Sponsored by Sucuri - https://sucuri.net
   @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
_______________________________________________________________

[i] It seems like you have not updated the database for some time.
[?] Do you want to update now? [Y]es [N]o [A]bort, default: [N]y

(中略)

[+] WordPress version 4.0.8 identified from meta generator
[!] 1 vulnerability identified from the version number

[!] Title: WordPress  3.7-4.4 - Authenticated Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8358
    Reference: https://wordpress.org/news/2016/01/wordpress-4-4-1-security-and-maintenance-release/
    Reference: https://github.com/WordPress/WordPress/commit/7ab65139c68389104265678 49c7abed723932b87
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1564
[i] Fixed in: 4.0.9

(中略)

[+] Finished: Sat Jan 16 18:50:13 2016
[+] Requests Done: 78
[+] Memory used: 1.766 MB
[+] Elapsed time: 00:00:10
```

★

ちと思われるユーザ名に対するスキャンを短時間に複数行っていることがわかります。

◆スキャンツールの例〜当然だが、よくある攻撃はツール化されている

図8はMedusaと呼ばれるスキャンツールを用いたスキャンの例です。スキャン対象ホスト、ユーザ名、パスワード文字列をまとめたファイルを指定して、SSHでログイン可能かどうかをスキャンさせている様子になります。もちろん、コマンドでユーザ名とパスワードを直接指定することも可能なので、ユーザIDとパスワードの組を指定してのパスワードリスト攻撃もMedusaを用いて実現可能です。

改ざんされたときに踏むべき手順〜まずは慌てず騒がず落ち着いて

改ざんされてしまうということは、「運用環境も含めたWebサーバの環境に何らかの不備がある」と

▼図7　SSHの認証失敗ログの一部

```
Jan 18 00:15:05 mx1 sshd[30569]: Invalid user usuario from xx.yy.253.132
Jan 18 00:15:07 mx1 sshd[30571]: Invalid user unison from xx.yy.253.132
Jan 18 00:15:08 mx1 sshd[30573]: Invalid user oracle from xx.yy.253.132
Jan 18 00:15:10 mx1 sshd[30575]: Invalid user postgres from xx.yy.253.132
Jan 18 00:15:11 mx1 sshd[30578]: Invalid user postgres from xx.yy.253.132
Jan 18 00:15:12 mx1 sshd[30580]: Invalid user support from xx.yy.253.132
Jan 18 00:15:14 mx1 sshd[30583]: Invalid user student from xx.yy.253.132
Jan 18 00:15:15 mx1 sshd[30585]: Invalid user plcmspip from xx.yy.253.132
Jan 18 00:15:16 mx1 sshd[30587]: Invalid user PlcmSpIp from xx.yy.253.132
Jan 18 00:15:18 mx1 sshd[30589]: Invalid user webuser from xx.yy.253.132
Jan 18 00:15:19 mx1 sshd[30591]: Invalid user mysql from xx.yy.253.132
Jan 18 00:15:21 mx1 sshd[30593]: Invalid user testuser from xx.yy.253.132
Jan 18 00:15:22 mx1 sshd[30595]: Invalid user ftpuser from xx.yy.253.132
Jan 18 00:15:23 mx1 sshd[30597]: Invalid user zabbix from xx.yy.253.132
Jan 18 00:15:25 mx1 sshd[30599]: Invalid user pi from xx.yy.253.132
```

▼図8　Medusaを用いたスキャンの例

```
root@kipple:~# cat hostfile
192.168.44.207
root@kipple:~# cat userfile
root
admin
root@kipple:~# cat passfile
12345678
qwerty
abc123
root
admin
root@kipple:~# medusa -H hostfile -U userfile -P passfile  -M ssh
Medusa v2.1.1 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: root (1 of 2, 0 complete) ↵
Password: 12345678 (1 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: root (1 of 2, 0 complete) ↵
Password: qwerty (2 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: root (1 of 2, 0 complete) ↵
Password: abc123 (3 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: root (1 of 2, 0 complete) ↵
Password: root (4 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: root (1 of 2, 0 complete) ↵
Password: admin (5 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: admin (2 of 2, 1 complete) ↵
Password: 12345678 (1 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: admin (2 of 2, 1 complete) ↵
Password: qwerty (2 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: admin (2 of 2, 1 complete) ↵
Password: abc123 (3 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: admin (2 of 2, 1 complete) ↵
Password: root (4 of 5 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.44.207 (1 of 1, 0 complete) User: admin (2 of 2, 1 complete) ↵
Password: admin (5 of 5 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.44.207 User: admin Password: admin [SUCCESS]
```

いうことを疑うべきです。単純に改ざんされたコンテンツを書き換えるだけでは、また同じことが発生する可能性もあります。なので、踏むべき手順をきっちり踏んで、同じ手口で改ざんされないようにという必要があります。

手順は0番から述べており、基本的な流れは押さえていますが、場合によっては手順が前後したり並行に走ったりすることもありえる点に留意してください。状況によっては手順を飛ばすなどもやむを得ませんが、基本は押さえたうえで、“なぜ飛ばしたのか”という理由も残しておきましょう。

やるとまずいこと～改ざん発生時あるある

実際の対応手順に入る前に、「やっちゃまずいこと」を述べておきます。まず、昔は「攻撃されたらOSから再インストール」というような対処もされていましたが、相当古い考え方であると言わざるを得ません。そして筆者が見る限り、ありがちな「間違った」対処は大きく次の3つになります。

①改ざんされたコンテンツを修正して元に戻す
②バックアップされているコンテンツを上書きして元に戻す
③OSごとやられたことを考慮して、すべて再インストールを行い、改ざんされる前の環境に戻す

程度によりますが、いずれもダメです。まず、①②については「改ざんされた理由／トリガを特定せずに」コンテンツを復旧しても、改ざんを行った攻撃者はどうやればコンテンツを改ざんできるか知っており、また同じやり口でコンテンツを改ざんされる危険性が高いためです。③についても、同じソフトウェアを用いてパッチなどの状況もすべて改ざんされたものと同じにしてしまうと、やはり再度改ざんされる危険性が高いです。

一方、改ざんされたコンテンツを含むシステムに対し、適切な証跡を残すようにしてあれば、それらの証跡を手がかりにして改ざんされた原因や手口をすみやかに特定することも可能です[注5]。

このために必要なことは「やられたWebサーバの保全」になりますが、後述するとおり、Webサーバの所在／Webサーバの所有者が誰かによっても保全の仕方が異なりますので、各人の環境によってどうすべきかを事前に確認しておいてください。

手順0 まずはWebサイトの一時閉鎖

改ざんが確認された時点でまずやるべきは、Webサイトの一時閉鎖です。この際に気をつけるべきは、後述する「手順1」の内容と競合しないように、Webサイトに触れずに閉鎖する、ということです。たとえばですが、次の方法が考えられます。

- 改ざんされたWebサイトを置いているコンピュータとは別のコンピュータ上に、「一時閉鎖してます」というメッセージを置き、DNSのエントリを変更する
- とりあえずマシンを停止させる

いずれにしても、改ざんされたWebサーバを継続的に用いるということは避けましょう。

手順1 改ざんされた状態を「保全」しよう

後でも述べますが、Webサーバを改ざんされた際によく言われるのが「保全しよう」ということです。保全とは、おおまかに言うと「改ざんが発覚した時点」のWebサーバの状態をそのまま保存し、あとで行う調査に必要な情報を確保する、ということです。

ここで言う「Webサーバの状態」とは、HDDや、可能であればメモリの中身のことを指しますが、メモリの中身についてはできればでよいでしょう。「メモリの内容についてはいいや」と割り切れるのであれば、Webサーバを保全する際には「電源をいきなり切断する」のが経験上はベストです[注6]。

◆仮想環境の場合は？～スナップショットをまるごと保存

仮想環境上で動作するVPSなどの場合、スナップショットを保存可能な場合があります。このよう

注5　もちろん、それらの証跡を読み解く力は必要です。

注6　どこかしら壊れる危険性はありますが、保全対象となるデータがなくなるよりはマシです。

な場合は発覚した段階のスナップショットを保存して、調査に備えるのがよいでしょう。

◆保全のあとは？～とりあえず調査用の複製を

保全が終わったらさぁ調査！と行きたいところですが、まだ待ってください。保全した内容が調査手順の不備で改変されてしまったら、場合によっては改ざんされた際の手がかりを消し飛ばしてしまうこともありえます。保全した内容は、調査のための複製を取得するようにして、極力手を触れないように保存しておきましょう。たとえば、次のような手順が考えられます。

- コンピュータからHDDなど記憶装置を取り外す
- 専用の装置などでHDDを複製する

専用の装置を準備できない場合には他のコンピュータに接続することになりますが、この際には保全対象のHDDに書き込みをしないように細心の注意をはらって、ddなどのコマンドでHDD全体の複製を取ってください。

仮に記憶装置の取り外しが困難な場合には、リスクはありますが改ざんされたWebサイトが稼働するコンピュータ上に、データを保全するための必要最小限のコマンドを送り込み、HDDのイメージなどを取得する方法が考えられます。

◆さらに興味がある人は？～「証拠保全ガイドライン」を読もう！

保全ひとつを取っても、留意しなければならない点が多々あることはご理解いただけたかと思います。ここで示したのは原則かつ最小限の話であり、本格的に知りたいという方には、デジタル・フォレンジック研究会という団体がリリースしている「証拠保全ガイドライン」をお勧めします。

■「証拠保全ガイドライン第4版」公開のお知らせ
https://digitalforensic.jp/2015/03/06/guidelines-4/

手順2 改ざんコンテンツ特定と改ざん時刻とログ～Webサーバのログ、CMSのログ、その他いろいろ～を調査

手順1が終わって「さて調査するぞ」となったときに、まず何を調査するか？を確定しましょう。経験的に最もわかりやすいのが「見知らぬファイルが置かれている」とか「変な内容が追加されている」というものなので、そのようなケースでどのように調査するのかを示していきます。

◆確認する範囲を絞り込み～置かれたファイル／改ざんされたファイルのタイムスタンプを取得する

よく「何かあったときにログを調査する」という話を聞きますが、取得されたログが膨大な量にわたる際には、多分やる気がなくなると思います。また、仮に相談を受けて大量のログを送りつけられても「何の手がかりもない状態でどこから見るのか」というやるせなさが後に残ります。

そんなときに役立つのが、タイムスタンプです。周到かつ高度な技術を持つ攻撃者はタイムスタンプすらも改ざんすることがありますが、多くの場合はタイムスタンプまでは更新できません。これは、タイムスタンプの改ざんには、適切な権限をもって、OSに密接に関連したしくみを利用して実現されるため、アプリケーションが動作する権限ではほぼ改ざんできないことに起因します。このため、まずは改ざんされたファイルのタイムスタンプを取得しましょう。

見知らぬファイルが置かれた場合にはファイルの作成時刻、既存のファイルが改ざんされた場合にはファイルの更新時刻を取得するのが定石となります。

◆絞り込んだ範囲のログを確認する

ログには多くの情報が残ります。たとえば次のような情報がログには残されます。

- 脆弱性を突いたものであるならば、特徴的なログが残存するケースがある
- 想定していないIPアドレスからのアクセスがある
- ツールによるスキャンがされていれば、スキャン

をした跡がログに残るケースがある
- 「?」なリファラをともなったリクエストがある
- やたらとPOSTばかり行われた形跡がある

でも、やみくもに手がかりがない状態でログを見ても、正直なところ役に立ちません。そこで役立つのが直前に取得したタイムスタンプです。当該タイムスタンプの前後で何が行われたのかを集中的に確認することで、改ざんが行われた経路を特定する速度が早まります。

なお、肝心のログが残っていないような事態に陥らないために、ログの保存期間を長めにとっておく必要があるのは言うまでもありません。

◆その他のケース～絞り込み手段と確認すべきログの種類はさまざまだが、基本はいっしょ

上記は最もシンプルな「ファイル改ざん」を想定した手順になりましたが、コンテンツがDBMSに置かれている場合などは、こんなにシンプルにはいかない可能性ももちろんありえます。そのような場合でも「どうやって範囲を絞り込むか」という考え方および、「絞り込んだ範囲で残存する痕跡を見つけ出す」という考え方は役立ちます。

あくまで上記は「ごく基本的なやり方」を述べているに過ぎません。このやり方をもとに、みなさまのところでできそうなやり方を模索してみてください。

また、保全から調査に至るまでの手順は、「コンピュータ・フォレンジック」と呼ばれる手法をベースに組み立てています。コンピュータ・フォレンジックそのものについて興味がある人は、先にも述べたデジタル・フォレンジック研究会という団体のWebサイトを見てみるなどして調べてみることをお勧めします。

■特定非営利活動法人 デジタル・フォレンジック研究会
https://digitalforensic.jp/

手順3 復旧～手順2で得られた情報をもとに穴を塞ぎ、元に戻す

手順2が無事完了すると、改ざんされた原因をつかめることと思います。復旧する際には、そのような原因を根絶し、少なくとも同じ手法で改ざんを行えないようにすることが必要です。

多くの場合は「CMSやOSなど、環境に含まれる脆弱性」「脆弱なパスワード」というのが挙げられるので、環境の最新化や強固なパスワードへの変更などが必要になってきます。運用やコンテンツの都合上、環境の最新化が難しいなどの場合には、改ざんを検知／ブロックできるようなソリューションの導入などを検討してください。特定のパラメータを伴ったリクエストを防げればよいという場合には、Web Application Firewall（WAF）の導入も効果的です[注7]。

手順4 影響範囲の確定～入口によってはほかの個所の確認も必要／場合によっては全部作り直し

手順2で確定された入口がきわめて限定的なものであれば、手順3までで改ざんそのものの対応は完了します。しかしシステムアカウントが乗っ取られた結果の改ざんなど、そうでない場合には、確認範囲が多岐にわたります。

最悪なのは、システム基盤がLinuxなどで構成されていて、root権限まで乗っ取られている場合です。このようなケースは、単に既存の穴を塞いでコンテンツを正常なものに書き戻すだけでは対処が足りないことがあります。

手順5 改ざんされたWebサイトがほかにもないか確認する（オプション）

Webサイトを1つ管理しているだけであれば、手順3もしくは手順4で対応は終わりますが、複数のWebサーバを管理している場合には、ほかのWebサーバが改ざんされていないか、そして同じ穴を持っていないか確認する必要があります。不幸にしてやられてしまったWebサーバが存在する場合に

注7 Apache2のモジュールとして提供されるmod_securityもWAFの一種といえます

は、上記の手順1～3を繰り返し、手口の特定から復旧までを行うことになります。

手順6 改ざんされた原因が発生しないような管理手順を確立する（オプション）

もし手順を確立してWebサイトを管理している場合には、改ざんされた原因への対応を（可能であれば）手順に反映し、少なくとも同じ攻撃方法で改ざんされないようなものにしましょう。

そのほかやるべきこと～Webサイトの性質によって異なるが、最低限告知はしておこう

自分が管理するWebサイトが改ざんされていた場合、Webサイト改ざん発覚から復旧までの流れと、改ざんされたり不正に置かれたコンテンツにどの程度のアクセスがあったかという情報くらいは可能な限り早く告知しましょう。とくにWebサイトが「マルウェア感染を引き起こす」ような改ざんをされていた場合や、何らかのプロモーションサイトであった場合には、（個人的には）必須の対応であると考えます。

改ざんされてしまったという事実は消せないので、発生してしまった事実に対して取りうる対応を実施するのが正しい姿といえます。

事前にできる「事後対応や被害低減の準備」

基本、改ざん対応が行われるのは、改ざんされてしまった後のことです。ちょっとしたことに留意しておくことで、改ざんされた後の対応を効率化できたり、場合によっては攻撃そのものを撃退することも可能になります。ここからはそういった対応をいくつか例示してみます。

「こんなのムリ！」という方は～対応依頼を行う先の確保を

すでに述べた手順はどちらかというと「わかる人が自前でどうやっていくか」という方法であり、正直こんな手順をすべての人にやってくれ、というのは相当ムリがあります[注8]。となると、相談できる先を確保するのが必要になってきます。

自前でいろいろ管理している人で、自分でなんとかしなきゃいけないという場合には、もうその人自身がなんとかするしかないのですが、専門家に対応依頼する費用をなんとかできる場合には、たとえば次のようなサービスを利用するのも手です。

■サイバー119サービス（株式会社LAC）
http://www.lac.co.jp/service/incident/cyber119.html
■セキュリティ・インシデント救急サービス（NTTデータ先端技術株式会社）
http://www.intellilink.co.jp/security/services/consulting/10.html

ただ、いったん改ざん事故が起きると、対応の要所要所で費用はかかります。このような対応を依頼する際の費用もかなりかさむということは覚悟してください。

Webサーバは誰のもの？～できる対応と難しい対応

Webサーバが自分の家やオフィスにあるなどして、気がついたらすぐに対処可能な場合は停止もラクですし、その後の対処も比較的容易に行えます。しかし、Webサーバがそのような場所以外のところ――たとえばデータセンターなど――に置かれていたり、Webサーバを所有しているのがほかの人だったりすると、できる対応とできない（難しい）対応があります。

- Webサーバを外部のデータセンタなどに置いてある場合
- Webサーバの所有者がWebサイトのオーナであっても、Webサーバを他者と共用している場合
- Webサーバの所有者がWebサイトオーナでない場合

上記のいずれの場合も、Webサーバを置いてある場所に入る手続きがあったり、Webサーバに他者の情報も入ったりするため、証跡をすぐに確保するのが難しいことがあります。このような場合の対応

注8　わかってて書いてます。

は、まだ攻撃による被害がなく余裕があるうちに確認し、できる対応を明確にするのが吉といえます。

基本は脆弱性対応とデータの自衛～パッチとバックアップは確実に(!)

本稿は「改ざん被害に遭ったら」という前提で書いてますが、それでもなるべく改ざん被害に遭わないようにするにこしたことはありませんし、改ざん被害に遭った際の復旧を手早く行えるにこしたことはありません。そのために必要なのは、次のことになります。

- Webサイトで使っているソフトウェアで発見された脆弱性への対処を迅速にする(=公開されたパッチを迅速に適用する)
- プログラム以外のデータのバックアップをとっておく(=改ざんされる前の状態に戻せるようにする)

いずれも通常運用に含める形で実践していただくのがよいでしょう。ちなみにバックアップを取ったつもりでもレストアできないというトラブルはよく聞くので、取得したバックアップデータをレストアできることは確認してください。

あと留意するべき点は、脆弱性が含まれているプログラムもいっしょにレストアしないようにするということです。もしそのようなプログラムもいっしょにレストアしてしまうと、せっかく対処した脆弱性がまた復活することになります。稼働する状態のシステム全体を復旧目的でバックアップする際には、バックアップデータよりレストアを完了した後からシステムを再開(公開)する前までの間に、パッチを適用するようにしてください。

転ばぬ先の杖～保全の訓練と使うツールの確保を

やられてしまってからツールをそろえるのでは、なかなか調査も進みません。となると、「やられてしまうこと」をある程度念頭に置いた道具立てが必要になってきます。

対応手順の中では「保全」という話を展開しており、解説中でも「コンピュータ・フォレンジック」という言葉を用いていますが、保全をはじめとするフォレンジック活動をまともに行うためには事前の準備が必要です。保全の手順や道具を準備していないと、その後の原因究明まで時間がかかることも多々あります。

たとえばですが、事前準備には次のようなアプローチが考えられます。

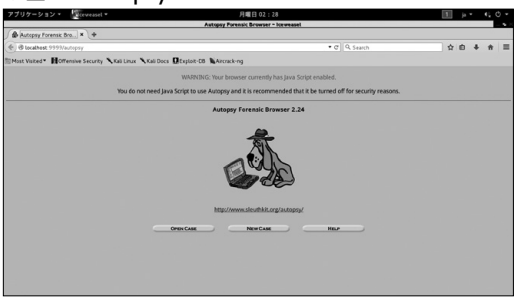
- 保全のために必要な環境をあらかじめそろえておき、改ざんが発生したときの保全や調査にその環境を用いる
- 保全を含めた調査活動の訓練を事前に行い、手順を確認する

必要な環境については、上を見たらきりがないのですが、調査訓練のために必要な道具立ては、たとえば次のようなものを用いることで対処可能です。

- SANS investigative Forensic Toolkit(SIFT) Workstation Version 3
  `http://digital-forensics.sans.org/community/downloads`
- Kali Linux
  `https://www.kali.org/`
- Autopsy(図9)
  `http://www.sleuthkit.org/autopsy/`

それぞれのツールの使い方1つを紹介するにもかなりの分量になってしまうのですが、機会があればご紹介させていただければと思います。SD

▼図9　Autopsy



短期集中連載

開発に効く数字の測り方

クラウド時代のWebサービス負荷試験再入門

Author 仲川 樽八（なかがわ たるはち） ㈱ゆめみ Twitter@tarupachi

第4回

段取りに従った負荷試験の進め方（後編）・最終回

はじめに

前回の特集にて、段取りに従った負荷試験の進め方の前編として、次のステップを紹介しました。

step1. 静的ファイルを叩くことで利用する負荷試験ツールや設定の試験を行う
step2. HelloWorldを叩く
step3. 参照系のページ、APIを叩く
step4. 更新の発生するページ、APIを叩く
step5. 外部サービスとの結合を含むページ、APIを叩く

今回は最終回として、段取りに従った負荷試験の進め方の後編を紹介します。

step6. シナリオを組んで試験を行う
step7. 離れたネットワークから試験を行う
step8. 各リソースのスケールアップ・スケールアウトをして試験を行う
step9. より強い負荷を与えてみる

負荷試験対象システム

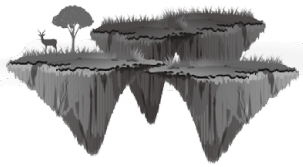
負荷試験の対象システムは、前回同様にAWS上に、スケール可能かつ、単一障害点を作らないWebサービスを構成するものとします（**図1**）[注1]。

・MySQLはRDSというAWSの提供するサービスを利用
・上記MySQLにはMultiAZオプションを付けて、ホットスタンバイ方式による冗長化を行っておく
・DNSとしてAWSの提供するRoute53サービスを利用
・ロードバランサとしてAWSの提供するElastic Load Balancer（以降ELB）を利用
・Webサーバはデータセンターに相当するAvailability Zone（以降AZ）をまたがる形で複数台利用することで冗長化とスケールアップ性能を担保
・キャッシュとしてAWSがサービスとして提供するElasti CacheをMemcacheプロトコルでAZをまたぐ形で利用
・外部のサービスとhttpまたはhttps経由で連携している

[Step6]シナリオを組んで試験を行う

前回までの負荷試験は単体のページやAPIを対象としましたが、今回からはいよいよ負荷試験シナリオを利用した負荷試験を行います。負荷試験シナリオとは、複数のページやAPIに対して実際のユーザが行うようにページやAPIをまたいで連続してリクエストを発生させる方法などを記載したファイルで、負荷試験ツールはこのファイルに記載された内容に基いて負荷試験を行います。

注1）今回は、説明を単純にするために、MySQLのスケール対応はRDSのスケールアップのみで対応することにしていますが基本は同じです。



一般的にWebシステムの負荷試験といえばこのシナリオを利用した試験のことを指すことも多いですので、いきなり負荷試験シナリオを組んで試験を始めることも多いと思われますが、過去に負荷試験がうまくいかなかった方に関しては、ぜひ前号の記事で紹介した[Step1]～[Step5]に関して見なおしていただければ問題解決のヒントが埋まっているかもしれません。

シナリオを記載して負荷試験を行うツールとしては以前の記事で紹介したJMeter、Locust、Tsungのいずれを利用してもかまいませんが、利用する負荷試験ツールによって作成方法は異なりますのでツールに合わせて個別で作成する必要があります。

シナリオの組み方

JMeterやTsungではシナリオを組むためにproxyを立てて、実際のリクエストをキャプチャする方法もありますが、多くの場合必要ありません。デバッグ用に適切なアクセスログを出力するようにしておけば、そちらからリクエスト内容を抽出するなどしてもいいでしょう。ブラックボックス的なページに対する試験などでシナリオの組み方がよくわからない場合は、ブラウザのデバッグ機能を利用して確認できるパラメータを埋め込む方法もあります(例：Chromeの要素を検証→Networkから確認できるパラメータなど)。

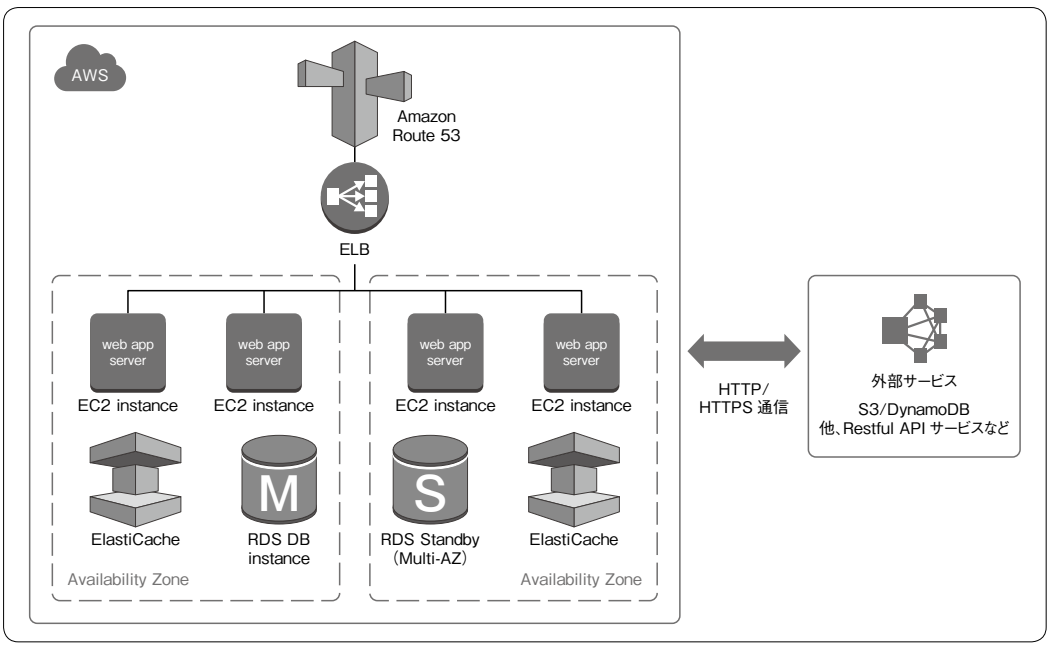
あまり深く考えずに典型的なユーザの導線を“適当に”考えて“適当に”組んでいってください。いずれにせよ、ユーザ導線の完全なモデルケースを作成するのは非常に困難ですので、よりアクセス頻度が高く、負荷が高く、問題が発生しそうなシナリオが含まれていればOKです。クラウド時代における負荷試験の目的は「システムのスケール性能を担保すること」であり、特定の条件下での完全な性能担保ではないと考えてください。

シナリオ作成時の注意事項

作成中のテスト実行と、実際に負荷をかけるときには、次の部分が異なります。

・シナリオ作成中は同時アクセス数を1にしたほうがログや結果を追跡しやすい

▼図1　負荷試験の対象となるシステム概要



・シナリオ作成中は詳細な結果レポートを出力したほうが作成しやすいですが、本番の負荷をかけるときには外す必要がある
・シナリオ作成はネットワーク的に離れた場所から作成して良い

スループットの評価について

シナリオを組んだ負荷試験を行う場合において取得される個別のリクエストに関するスループットと、単体ページに対する負荷試験で得られるスループットは考え方が変わりますので注意が必要です。

例として10件の懸賞が登録された懸賞応募サイトのユーザ導線をシミュレートするシナリオを次に挙げます。

①ログインする（ログインAPI）
②応募可能な懸賞の一覧を取得する（一覧取得API）
③応募可能な懸賞が残っている場合は懸賞に応募する（懸賞応募API）→②に戻る

このときに次の試験結果が得られたとします。

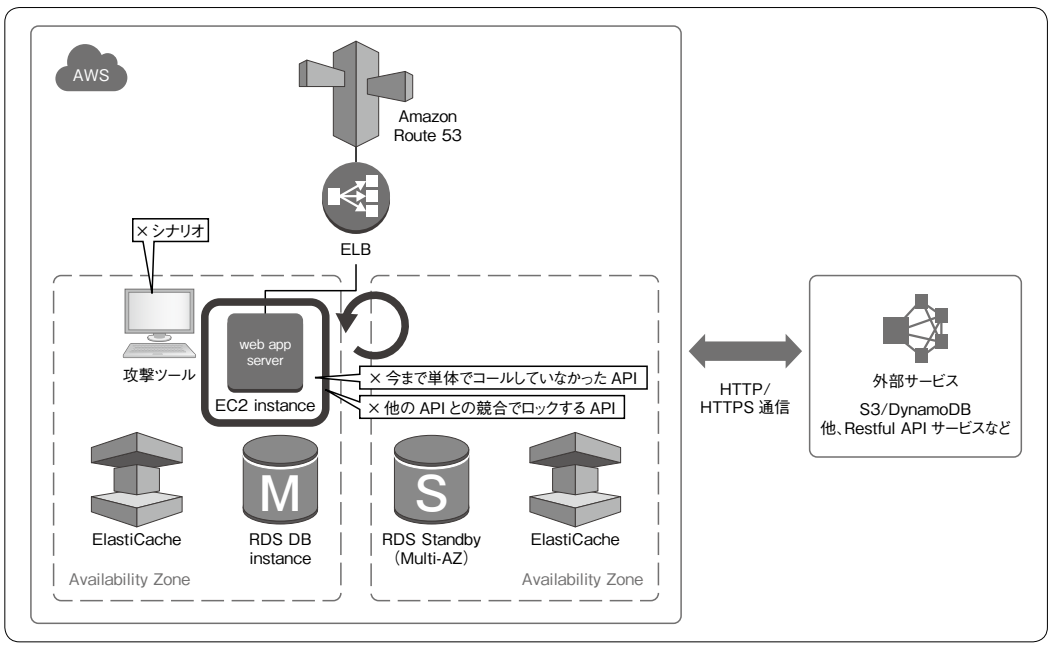
ログインAPIのスループット：100 req/sec
一覧取得APIのスループット：1000 req/sec
懸賞応募APIのスループット：1000 req/sec
Total: 2100 req/sec

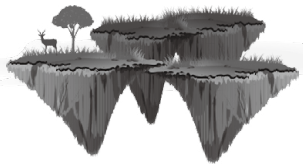
このシナリオにおけるシステムの性能を知りたい場合には、この数字のどこに注目するかによって結果が変わってきます。

・参加可能なユーザのシナリオに注目する場合 → 100 req/sec
・懸賞の応募数に注目する場合 → 1000 req/sec

また、とくに補足のない状態で上記の結果を見たときに、一覧取得APIや懸賞応募APIは1秒間にログイン処理の10倍の回数をこなしているため、処理性能としてもそれぞれ10倍高速に応答をこなすことができるように受け取られることがありますが、もちろんこちらは間違いで単体のAPIの応答性能はスループットではなく、API個別のレイテンシを見ないと判定することはできません。同一シナリオ中の個

▼図2　Private IP経由での攻撃をかける。前回記事でシナリオ不要のケースと同様の構成





別のページやAPIのスループットは単純にそのシナリオ内で設定された実行回数に比例しますので、とくに負荷試験レポートを記載する際にレポートを読む人間がこの数字を読み間違えないようなレポートの記載をする必要があります。

対象のシステム

負荷をかける対象のサーバが1台、ローカルホストまたは同一セグメントのサーバからPrivate IP経由での攻撃をすることは前回の特集で記載したシナリオ不要のケースと同じです(**図2**)。

目的

セッション的な手続きのあるシナリオにおけるシステムのスループットを計測します。実際のユーザのアクセスをシミュレートした環境での負荷試験を行います。

次へ進む条件

シナリオが正常に流れ、結果として、Webサーバもしくは DBサーバのリソースのいずれかが逼迫することが確認できれば、次に進んでください。結果の数値が今までの個別の試験結果から予想できる範囲に収まっていれば大丈夫です。また、負荷試験実施の結果、DBなどに正常にデータが登録されていることの確認は必要です。

負荷がうまくかからない原因の例

負荷試験ツールの設定が悪い

次の対策などが考えられます。

- これまでの試験と別のツールを利用した場合、静的ファイルへのアクセスからいったんやり直して設定を見なおしてみる
- JMeterの場合、シナリオ中でJavaScript関数を多用するとJMeter側でリソースを大量に使用してしまうことがあるので注意する
- JMeterの場合、長時間の試験を行っている間に攻撃側の能力が下がってくることがあるため、システムのスループットが落ちてきたように見えるときがある

アプリケーションに問題がある

次のような問題が考えられます。

- 今まで試験していなかったページやAPIのロジック不備、リソース競合など
- コールされるページやAPIの組み合わせによるリソース競合など

[Step7]離れたネットワークから試験を行う

対象のシステム

今までの試験は、すべてローカルホストまたは同一セグメントのサーバからPrivate IP経由での攻撃を行っていました。これをロードバランサを経由した攻撃に変更します。この試験では今まで行ってきた試験を攻撃サーバのみ変更した状態ですべて再実行します。

ロードバランサの配下に設置するWebサーバはまだ1台だけとしてください(**図3**)。

目的

グローバルセグメントからの攻撃が適切に行われることを確認します。独立した負荷試験攻撃サーバからかけることができる攻撃の限界値を計測します。

次へ進む条件

この試験の結果として、今までの試験と同じく適切に各リソースが利用されて、同等のスループットが出ていれば次に進んでください。ELBは急激な負荷上昇を検知したときに自動でスケールするため、ウォームアップが済んでいない状態だと本来のスループットを出せなくなることがありますが、実際には数分間負荷をかけ続けた時点で十分な能力までスケールします。しばらく負荷をかけてもスループットが改善しない場合はELBではなく、ほかのインフ

ラの能力を疑ったほうがいいかもしれません。

負荷がうまくかからない原因の例

インフラの問題

ELBのウォームアップが完了していない場合が考えられます。

ネットワークの問題

ネットワーク側の問題として、次の項目が考えられます。

・攻撃ツールにGlobal IPが割り当てられておらず、NAT経由での攻撃をしている(=NATサーバがボトルネックになっている)
・SSLを利用した負荷試験を行っている[注2]
・攻撃サーバとELB間でKeep-Aliveされていない

[Step8]各リソースのスケールアップ・スケールアウトをして試験を行う

いよいよ本番です。クラウド時代における負荷試験の目的はまさにこれで、これまではこの試験を円滑に進めるための前準備に過ぎません。今までの試験においてボトルネックとなっていたリソースに関して、スケールアップまたはスケールアウトを行い、再試験を行うことを繰り返します(**図4**)。

目的

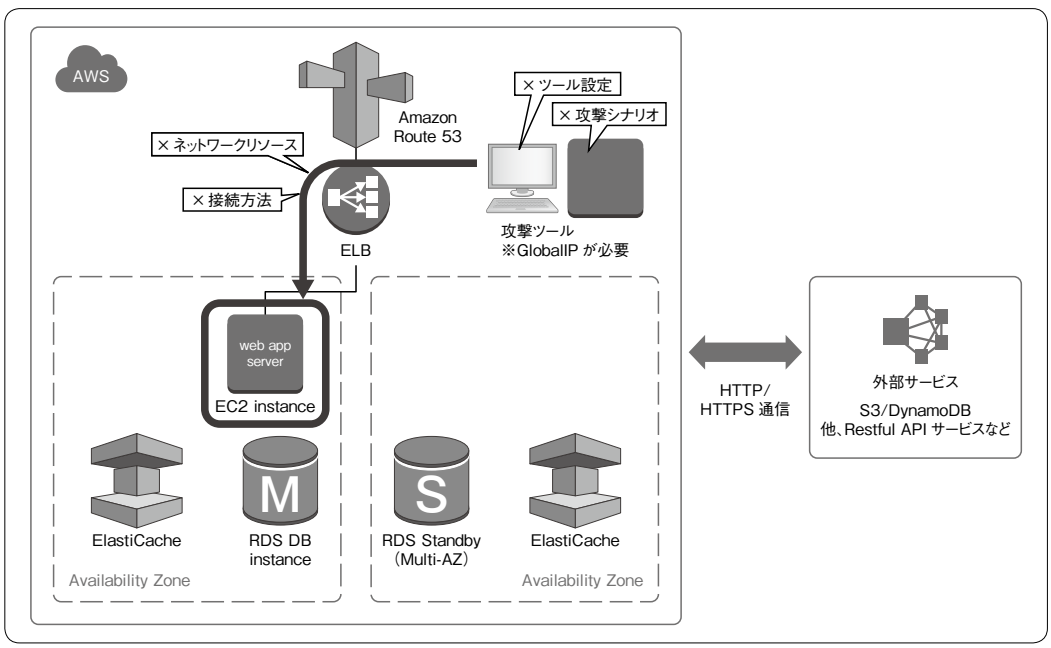
システムのスループットがスケールアップやスケールアウトに追従して改善することを確認します。

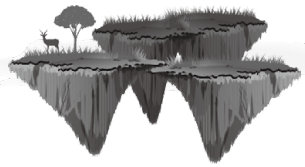
スケールアップ、スケールアウトの例

ここまでの試験が順調に進んでいた場合、システムのリソースが逼迫しているのはWebサーバのCPUリソースであることがほとんどです

注2) AWSでシステムを組む場合において実際にはELBにてSSLの終端を担うことが多いと思います。その場合は各Webサーバへの SSLの負荷はかかりません。であるにもかかわらず、SSLを利用したアクセスをしている場合、負荷試験にて観測可能なスループットは1/10以下になるなど、著しく劣化するといった状態が発生します。このとき、Webサーバには実際にはほとんど負荷がかかっておらず、Webサーバ上で見たレイテンシはほとんどない状態で高速に応答しているにもかかわらず、負荷試験ツールから見たレイテンシが高いため、次のリクエストを投げることができないという状況となります。これは負荷試験サーバ側でSSLのデコードのためのコストがかかるために適切な負荷試験を行えていないパターンとなりますので、負荷試験においてはSSLを利用しない試験を行う必要があります。

▼図3 対象のシステム





ので[注3]、この場合はまずWebサーバに対してスケールアップおよびスケールアウトを行います。理想的なCPUボトルネックの場合であれば、システムのスループットはWebサーバのスケールアップやスケールアウトに対応して正比例して向上しますので、システムリソースの増強に合わせて、攻撃ツールのクライアントの同時接続数も増やしながら試験を繰り返します。

Webサーバの追加に関しては2台→4台→8台→16台と、リソースの逼迫がWebサーバ以外の部分に移行するまで、思い切って倍々で増やしていきます。これはWebサーバリソースの余裕が中途半端だった場合にはたとえ他の部分のリソースにボトルネックがあり、そのボトルネックを解消することで全体のスループットが改善するという状況であったとしてもその効果が目に見えにくいためです。

Webサーバの追加により、Webサーバのリソースに十分な余裕が出たうえで、Webサーバの追加によってはそれ以上システムスループットが改善しなくなった時点でWebサーバ以外に発生しているリソース逼迫個所(DBのCPUリソースなど[注4])を探します。次のボトルネックが判明した場合はそのボトルネックに対してシステムリソースのスケールアップ、スケールアウトを行っての負荷試験を行ってください。

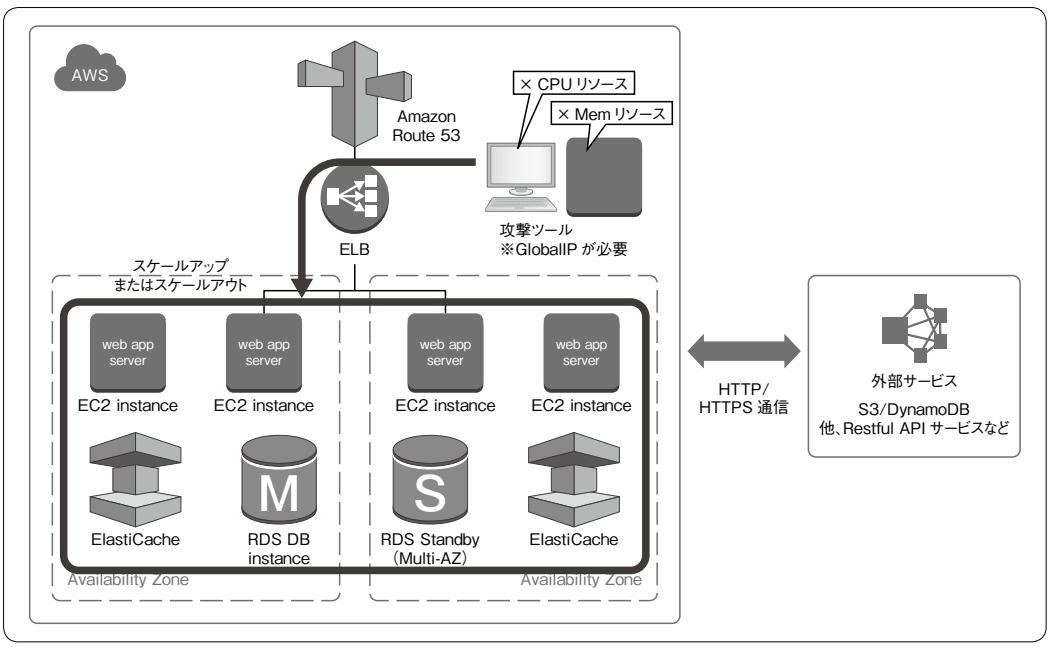
実際の商用環境ではボトルネックでない個所に対して必要以上のリソースを確保することにはまったく意味がなくリソースの無駄遣いとなってしまうのですが、今回は試験ですので、それぞれのリソースに対してWebサーバの増強時と同じく、十分な余裕のあるリソースサイズになるまで余分に増強したうえで次のボトルネックを探してそちらを対策するという作業を繰り返します。

この作業の途中でそれぞれのリソースに対し

注3) Webサーバの CPUボトルネックの場合は非常にわかりやすく、負荷試験中のCPUリソースがほぼ100%の状態で張りつきます。

注4) DBサーバの場合、WebサーバのCPUボトルネックとは異なり、CPU使用率が60%〜80%など、100%になる前に単体スループットが頭打ちとなることがありますので注意してください。また、RDS for Auroraの場合はCPUを積極的に利用するという設計のため、CPU利用率は100%近くとなっていても、まだ単体スループットが上がるということがあるようですので、こちらも注意が必要です。

▼図4　スケールアップ・スケールアウトして試験を行う構成



ては今まではかからなかった負荷がかかるようになってきます。とくにスケールアウトではなく、スケールアップにてスケール性能を担保しようとしてきたリソース(今回の例ではRDS)には負荷が集中しますので、そちらがボトルネックとなり今まで出なかったロックなどが発生することがありますので、システムのスケールアップやスケールアウトと平行してアプリケーションのチューニングも進めます。

次へ進む条件

リソースが逼迫していた部分のスケールアップ、スケールアウトをすることで、ボトルネックの解消とともにシステムのスループットが改善することを確認してください。システム全体のスループットが当初の目標値を上回った時点、または先の試験で確認した攻撃ツールの限界まで負荷がかかったため、それ以上の負荷をかけることができなくなった時点でこのステップは終了します。

スケールアップで負荷がうまくかからない原因の例

ミドルウェア設定、カーネルパラメータ設定に問題がある

サーバ1台あたりで処理させなければならないリクエスト数が増えていきますので、Apacheなど、各種ミドルウェアの設定値をそれに応じた設定に見なおさなければなりません。

DBに問題がある

とくにMultiAZオプションを利用していた場合、DBのスケールアップによるスループットはインスタンスタイプごとのCPU処理能力とは比例しない(ことも多い)ですので、ある程度以上のインスタンスタイプを最初から利用している場合にはリソースのインスタンスタイプを変更してもあまりスループットが改善しないことがあります。

スケールアウトで負荷がうまくかからない原因の例

こちらは、サーバの台数に左右されない外部リソースへのアクセス部分を中心に疑ってください。

インフラに問題がある

次の3例が考えられます。

・ロードバランサに問題がある
アプリケーションサーバをMulti AZ構成としたときに、負荷試験ツールによっては適切に負荷を分散できないという症状が発生する(JMeterなど)。この場合は、どちらかのAvailability Zoneにサーバをすべて集めて構築することで対応

・ネットワーク設定に問題がある
外部APIを利用する場合にNATによるルーティング部分がボトルネックとなり、途中までしかスケールしないという症状になることがある

・ネットワーク帯域に問題がある

アプリケーションに問題がある

次の問題が考えられます。

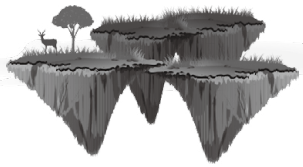
・キャッシュ設計に問題がある
・ログ転送先が詰まっている

過去事例では、rsyslogによる外部転送がボトルネックになったことがあります。

このときは、Webサーバを一定数以上追加すると、負荷試験中のスループットが乱高下するようになってしまいました。

[Step9]より強い負荷を与えてみる

多くの場合はこのステップは必要ありませんが、システムに要求されるスループットが高いときには、攻撃サーバ1台では適切な負荷をかけることができないことがあります。その場合



には複数の攻撃サーバを起動して、同時に負荷をかけてその結果を観測します(**図5**)。JMeterやLocust、Tsungにはそれぞれ複数の攻撃サーバ間の連動機能が提供されていますが、詳細な結果でなくても良いのであればCloudWatchによるモニタリングだけでもつかむことができますので、複数の攻撃サーバを連動させることができないツールや、連動機能を利用しない場合であってもとくに問題はありません。また、適宜複数のツールを組み合わせて同時に負荷試験を行っても大丈夫です。この試験においては攻撃サーバの増強に応じて、前のステップでスケールアップ、スケールアウトを行い解消させたボトルネックにふたたび負荷が集中し、新たなボトルネックとして発現するようになりますので、再度[Step8]に戻ってシステムの増強をすることを繰り返し行います。

JMeterで攻撃Slaveサーバを連動させる

JMeterにおける連動機能の利用方法は誌面で紹介するには量が多くなりますので、ここではWeb上に公開されている記事の紹介だけとさせていただきます。

・SpotInstanceとJMeterを使って400万req/minの負荷試験を行う[注5]

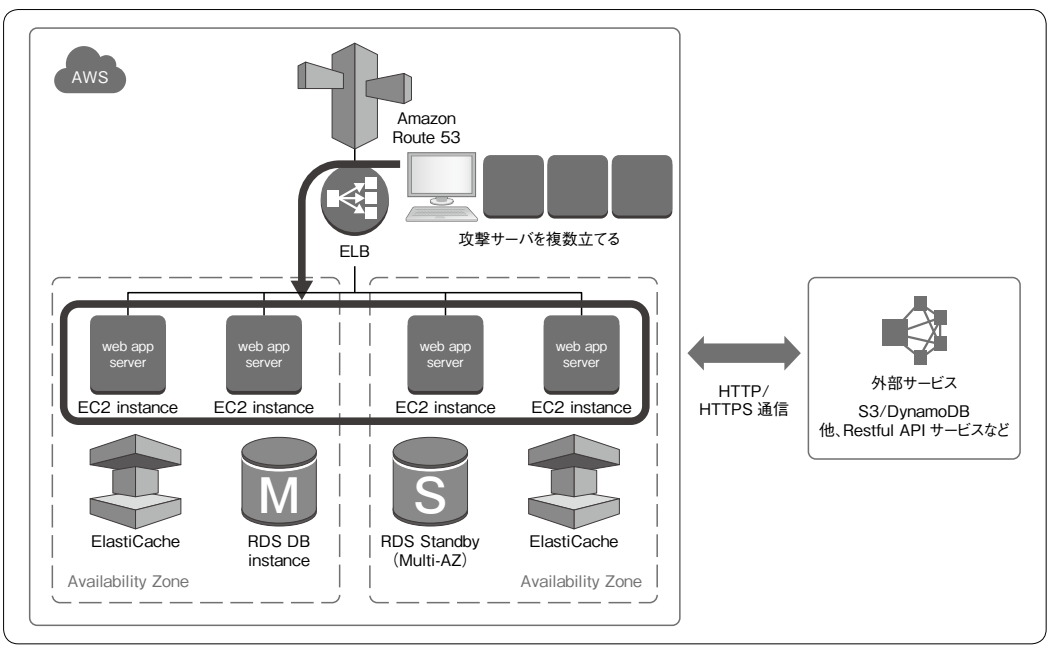
以前の記事で紹介したクラスメソッド社が公開している記事です。JMeterで攻撃をかける場合は攻撃サーバ1台あたりの攻撃性能が低くなりがちですので、ほかのツールと比較して大量の攻撃サーバを準備する必要がありますが、こちらの記事ではクラウドのメリットを最大限に利用して安価に大量の攻撃サーバを準備しています。

Locustで攻撃Slaveサーバを連動させる

以前の記事で紹介した複数のSlaveサーバを同一のインスタンス上で起動するパターンと同様にしてSlaveサーバを物理的に他のサーバ上で起動させることができます(**リスト1**参照)。このときも、Slaveサーバ上のCPUの空きリソースに応じて複数のSlaveサーバを起動させ

注5) http://dev.classmethod.jp/cloud/apache-jmeter-master-slave-100mil-req-min/

▼図5　より強い負荷を与えてみる



▼リスト1　起動ファイルの例(run_remote_slaves.sh)

```
#!/bin/sh

LOCUST_FILE=${1:-src/locustScinario.py}
SCINARIO_CLASSNAME=WebsiteUser
MASTER_HOST=127.0.0.1

sudo /bin/sh -c "
ulimit -n 65535

 for slaves in {1..10}
do
    /usr/local/bin/locust -f ${LOCUST_ﾌ
FILE} ${SCINARIO_CLASSNAME} --slave  ﾌ
--master-host=${MASTER_HOST}&
done
```

ることができます。

Tsungで攻撃Slaveサーバを連動させる

XML設定ファイル中の設定部分において攻撃用Slaveサーバの定義を記載します。

こちらは設定サンプルがWeb記事[注6]に記載されています(Xexaさんの公開記事)。3台のTsungサーバを連動させて攻撃させています。Tsungでは比較的少ない台数のサーバでも大きな負荷をかけることができます。

目的

単体の攻撃サーバからでは負荷をかけきれない、より大規模なシステムにおける負荷試験を実施します。

完了条件

攻撃サーバのリソース追加および、システムのリソース追加に対応してシステムがスケールすることを確認し、最終的にシステムの応答が目標のスループットを出していること、高負荷時にシステムが予期せぬ挙動となっていないことなどを確認できた時点で負荷試験の終了です。

リリースに必要なリソースを見積もる

負荷試験としてはここまでで挙げた内容で一通り終わりになります。しかしながら、負荷試験としては各リソースがボトルネックとならないようにリソース増強の際には必要十分以上のリソースを追加していく作業を繰り返してきましたが、実際にはより多くのリソースを使用するということはよりコストがかかるということですので、リリース時点においては必要十分な中で適正なリソース量に絞り込むことが必要となります。実際に想定されるユーザアクセス数に一定の安全係数をかけたうえで、各リソースが余り過ぎない範囲でリソースの選定を行い、最後にもう一度負荷試験をかけて性能を測定します。

最後に

全4回にわたって負荷試験に関する全体的な考え方や効率的な実施手順を紹介してきましたが、いかがでしたでしょうか。「クラウド時代」のと銘打ってはいますが、当然オンプレミスにおける負荷試験に関しても負荷試験ツールの考え方などは同じです。これらの考え方は負荷試験実施を効率的に行うことだけでなく、負荷試験レポートをまとめる際にどの数字に対してフォーカスしなければならないかなど、見せる数字、見せるべきではない数字などを選定するためにも非常に重要な考え方となります。負荷試験に関しては筆者もいまだに試行錯誤を繰り返している状態ではありますが、この記事がシステムリリース後の事故を減らすことに少しでもつながれば幸いです。SD

注6) Qiita:Tsungで負荷試験(http://qiita.com/Hexa/items/323d69ee19a68f217191)

RDB性能トラブルバスターズ奮闘記

原案 生島 勘富(いくしま さだよし) info@g1sys.co.jp ㈱ジーワンシステム
構成・文 開米 瑞浩(かいまい みずひろ) イラスト フクモトミホ

第1回 SQLは集合指向の言語だということを知っていますか?

遅いシステムができてしまう原因として、DBを適切に使えていないケースがよくあります。その背景には、多くのエンジニアがSQLを正しく理解していないということがあるようです。生島氏と大道君の事例を通じて、SQLの正しい使い方を楽しく学びましょう。

登場人物紹介

生島氏
DBコンサルタント。性能トラブルのヘルプのため大道君の会社にやってきた。

大道君
大阪のSI企業に勤める新米エンジニア。素直さとヤル気が取り柄。

本番システムでは性能が出ない!

それは今からXX年前のこと、ある取引先からの1本の電話がきっかけでした。「来週カットオーバー予定のうちのシステムなんですが、どうにもレスポンスが遅くて困ってるんです。生島さん、ちょっと見てやってもらえませんか?」(以下、私(生島)は根が大阪人なので、会話では大阪弁が出ることをご容赦ください)。

2つ返事で引き受けて担当者に状況を聞いてみたところ、システムというのはあるECサイトで、お客様がネットを通じてアクセスし、商品カテゴリを選ぶと、該当するカテゴリの商品一覧をその瞬間の在庫数量とともに表示するもので、その商品一覧画面の表示が遅い、ということでした。

「応答時間の目標はなんぼやの?」と聞くと、「3秒切ればなんとか格好がつくんですが……」と答えてくれたのは担当の大道君。ずいぶん若いなと思ったら、まだ24歳だそうでした。画面自体は見たところ、どこのECサイトにでもあるような何の変哲もない商品一覧です。「(普通に作れば1秒かからんやろうけど)……で、今はどれぐらい?」「3分です」「そら〜、しんどいな」「はい……」と、いかにも意気消沈の様子。これが解決しないとサービスインできないため、上司も含めて2週間ほどあれこれやってみたがダメだった、もはやリリース延期やむなしか、というところで私が呼ばれたようでした。

「開発中も遅かった?」

「いえ、本番のデータを食わせたら遅くなったんです」

「開発用のデータは本番と同じ件数ある?」

「ありません。10分の1もないと思います」

「まあよくあるパターンやな。もっと早めに本番と同じデータでテストしとくんやったな」

「やっぱりそうですよね……次からはそうします」

お、素直なリアクション。若いときにこの姿勢はとくに大事。見どころあるじゃないか、と密かに思いつつ、次はプログラムを見せてもらいました。

意外によくある怒濤の二重ループ

「ふむ……原因はこれやな」

「えっ、わかったんですか？」

と驚く大道君。まあ、この2週間ほど上司と2人で四苦八苦してダメだったのに、DBコンサルタントの生島とかいう知らないオッサンに30分もしないうちに「わかったでえ」とか言われても、にわかに信じられないことでしょう。それは無理もないことなので、その場で簡単なプログラムを作ってみせたところ、同等の結果を得るための応答時間は3分から1秒に縮まりました。その差約200倍。大道君、口あんぐり状態。

「な……なんでこうなるんですか？」

「ほな、説明しよか」

原因はこうでした。在庫数量は入荷数量の合計から受注数量の合計を引くことで得られますが、要するにその集計処理をアプリケーション（以下、アプリ）側で行っていたのです（**図1**）。

本来この計算は、適切なSQL（Structured Query Language）を書いてデータベース（以下、DB）側で行うべきものです。それをアプリ側でやると、①DBの実行計画作成回数、②DBのディスクアクセス回数、③DBサーバとAPサーバ間のデータ転送量、④AP側の実行命令数、という4つの点で不利になり、負荷が重くなりやすいのです。

「え？　ダメなんですか、これ？」と大道君。

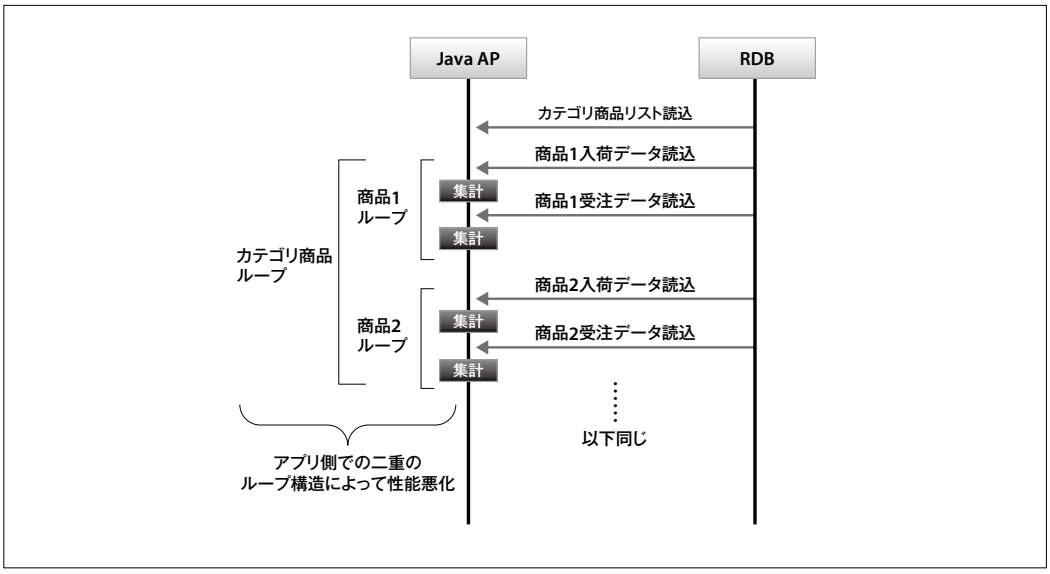
「実はな、あかんのや」

「知りませんでした……」

メモを取りながらの、やはり素直なリアクションに好感が持てます。こういう若者は環境しだいで大きく成長します。

さてこのパターン、AP側で二重にループを回すことになるので怒濤の二重ループ問題とでも呼んでおきましょう。単純ですが性能トラブルの原因としてわりとよく見かけるものです。原因はすぐにわかりましたが、問題はなぜこれを自力で解決できず、私のところに相談が回ってきたのかということです。率直に言ってこの二重ループ問題、RDB（Relational Database）とSQLの基本を知っていれば起こすはずがありま

▼図1　怒濤の二重ループ構造による性能悪化



せんし、たとえ起きてもすぐわかるはずなのです。それがわからなかった、ということは……。

RDBとSQLの基本を知らない?

つまり、RDBとSQLの基本を知らずに設計／プログラミングをしているのでしょうか。

そう考えざるを得ない事例を、私は長年見てきました。現代のオープン系システム開発、とくに業務系の開発ではRDBは必ず使われると言って良いでしょう。にもかかわらず前述の事例のようなケースに限らず、設計者の知識を疑わざるを得ないような設計をよく目にします。そしてこれは根本的には会社の組織体制に問題がある、と私は考えるようになりました(**図2**)。

「システムの性能が悪い」問題を受けて原因を調べると「SQLの使い方が悪い」部分が見つかります。コードは直せば動きますが、実際には「設計者がRDBとSQLのしくみをわかっていない」場合は同じ失敗を繰り返しますので、教育が必要です。しかし、現実には「DB技術者が育たない組織体制になっている」会社が多いのです。

「こういう二重ループはアカン、て、誰か教えてくれへんかった？」

「いえ、誰も……」

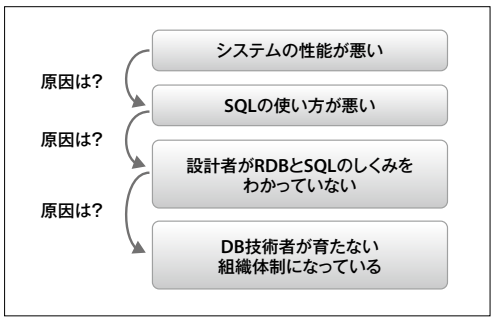
「SQLは集合指向の言語やって、聞いたことないか？」

「集合指向？　なんですかそれ？」

というこの答えが象徴しているように、きちんと教えてくれる先輩が身近にいなかったわけですね。技術というのは日々の実践で向上していくものなので、社外に1日2日の研修に出すだけでは限界があります。教育をするにしてもそれが可能な組織体制でない限り成果は上がりません。とはいえ、組織体制を変えるには会社を動かす必要があります。勉強だったら自分がやれば済むことなので、エンジニア自身で1人でもできます。

というわけで本連載では、DB技術を学びたいエンジニアに役立つ情報提供をしていきます。まずは集合指向言語と手続き型言語の違いを押さえておきましょう。

▼図2　DB技術を重視しない組織体制が根本原因



SQLは「集合指向」言語です

現在のIT技術者がおもに使うプログラミング言語というと、Java、C#、C/C++、Python、PHP、Visual Basicなどが挙げられますが、これらはいずれも「手続き型言語、またはそれを発展させたオブジェクト指向言語」であり、コードの細部は手続き型の考え方を基本としています。一方、RDBへの問い合わせに使うSQLは「集合指向」という、手続き型とは根本的に異なる設計思想を持った言語です。これが、一般のIT技術者にとってSQLの理解を難しくしている原因なのです。では、手続き型と集合指向ではどのように違うのでしょうか？

手続き型言語では実行順序を記述する

図3に食材からカレーやシチューのような料理を作る作業をイメージしたフローを示しました。手続き型言語が想定しているのは、このように「多数の異なる手続きの実行順序を記述する」ことです。そしてその際よく出てくる特定のパターンを簡潔に書くために、「判断」「繰り返し」のような制御構造、あるいはサブルーチン化、例外処理、オブジェクト指向といった仕様が導入されてきました。その具体的な仕様は個々の言語によって違いますが、「実行順序を記述するものである」という基本は手続き型言語に共通しています。

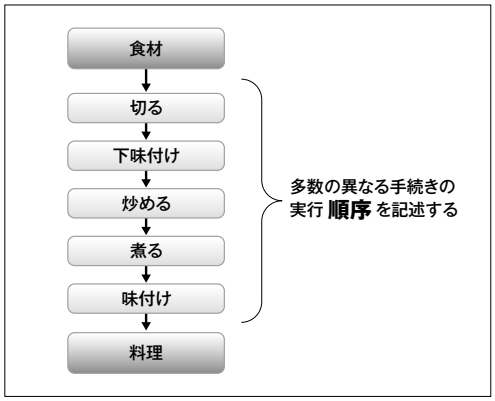
SQLでは範囲指定で同じ操作を一括適用する

一方、SQLが想定しているのは**図4**のようなモデルです。こちらは、ホテルの宴会場のような場所で、何種類ものメニューを一気に大量に作る場面をイメージしてください。最終的にできる料理がカレー、シチュー、肉じゃがと違っていても、中には共通の部分があります。たとえば「食材を一口大に切る」ところはすべての料理に共通だから一括してできるし、カレーとシチューについては味付け以外は共通なのでまとめられるでしょう。SQLはこのような場面で「範囲指定して同じ操作を一括適用する」ための言語なのです。手続き型言語は**図4**の縦方向の処理、SQLは横方向の処理に向いているわけです。

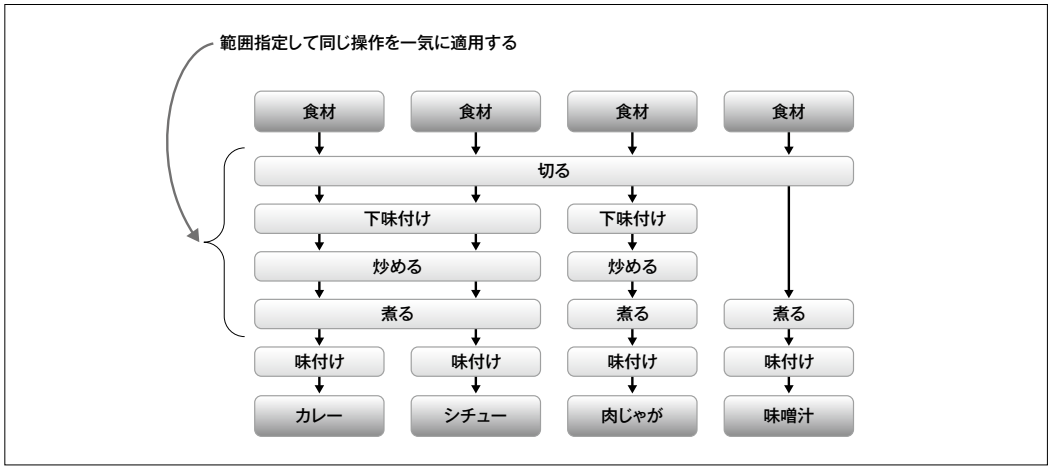
こうした違いが典型的に表れるのが、「ループ」です。

リスト1のコード例はいずれも「注文データを集計して金額の合計、最大、平均を出す」という想定のものですが、手続き型言語(Java)の例ではfor文を使ったループ処理があり、SQLではそれがないことに注目してください。SQLでは「同じ性質を持ったデータの集合に対して同じ操

▼図3　手続き型言語の処理モデル



▼図4　集合指向言語(SQL)の処理モデル



▼リスト1　集計処理

手続き型言語(Java)での集計処理(orders配列のBillingの合計、最大、平均値を算出)

```
int sum = 0, max = 0, avg = 0;
for(int i=0; i < orders.length ; i++){
   sum += orders[i].Billing;
   max = (max > orders [i] .Billing) ? max : orders [i] .Billing;
}
if( orders.length > 0 ) avg = sum/orders.length;
```

集合指向言語(SQL)での集計(orderテーブルのBillingの合計、最大、平均値をcustomer_idごとに算出)

```
SELECT customer_id, sum(Billing) , max(Billing), avg(Billing)
       FROM order
       GROUP BY customer_id;
```

作を一括適用」するのが基本です。ループ制御構造を記述する必要がない分、SQLの例のほうが簡潔に書けていることがわかりますね。しかも、SQLのコード例が「customer_idごとに分類して集計」しているのに対して、手続き型言語では全体を集計しています。もし手続き型言語でもcustomer_idごとに分類しようとして、そのために連想配列を使わずにもう一段ループをかますと……はい、こうして「怒濤の二重ループ構造」が発生するわけです。

設計思想が違うものは理解しにくい

「同じ性質を持った要素群」つまり「集合」に対して「同じ操作を一括適用する」という場面には、それに向いた言語があります。それがつまり「集合指向言語」であり、SQLなのです。

もう一度書きますが、手続き型言語は**図4**の縦方向の処理を書くのに向いているため、横方向の「繰り返し処理」を書こうとするとループ制御構造の作業変数や処理の開始／終了ロジックが必要になり、コードが複雑化します。言語の設計思想が根本的に違うのです。

この根本的な設計思想の違いが、SQLの「理解しにくさ」の原因です。「プログラム言語は、1つ覚えればほかの言語にも応用が利く」と聞いたことはありませんか？

「聞いたことあります。そう思ってました」と大道君。

「手続き型同士ならそのとおりなんやけどな。集合指向は感覚が違うんよ」

手続き型言語の感覚でSQLを学ぼうとしてもうまくいかないので、ゼロからやりなおすつもりで取り組むべきなのですが、それを教えてくれる先輩はなかなかいません。その結果「よくわからん。なるべく使わないでおこう……」とSQLから逃げると、本来SQLで処理すべきことまで手続き型言語で書くようになります。その行き着くところが「怒濤の二重ループ問題」のような事例です。そんなケースを私はこの20年間何度も見てきました。

「二重ループは絶対にやっちゃいけないんですか？」

「まずは全力でなくす方向で考えること。使ってもいい場合がないこともないけれどね。でないと、複雑な帳票では7重、8重のネスト構造になってしまう。そうなってから1つのSQLに直そうとすると、仕様書からソースコードまでまったく別物やから、無駄になる工数／納期は、今回の数倍になるよ」

「二重ループを使わないってことは……発行するSQL文は今より複雑になりますよね」

「そのとおり」

「SQLが複雑になっても、こういう繰り返し処理はできるだけDB側でやらせて、最後の結果だけを一発でアプリに返すほうがいいんですか？」

「そのとおり！」

そう、そこが本質なんです。大道君、なかなか筋がよさそう。私は少しずつ、彼に期待したくなっていました。きちんと勉強すれば伸びるタイプでしょう。

役割分担が適切に引かれていない

そもそもアプリケーション開発にかかわるテクニカル・スキルは大きく2種類あり、UI層（ビジネス・ロジックを含む）は手続き型言語、DB層は集合指向言語（SQL）の領域なのです。ところが手続き型言語を使い慣れた技術者に比べて

熟練のDB技術者は少ないため、適切な役割分担ができていないケースが非常に多いのが現実です。その結果SQLを使うべきところできちんと使えないと、性能悪化／開発工数の増加などの問題を引き起こしてしまいます。この問題を解決するためには、第1に経営者がDB技術の重要性を理解して、技術者間で適切な役割分担ができる組織体制を組むこと、第2にSQLをきちんと理解している技術者を増やすことが必要です。

「ちゃんと勉強すれば、僕にもできますか……？」

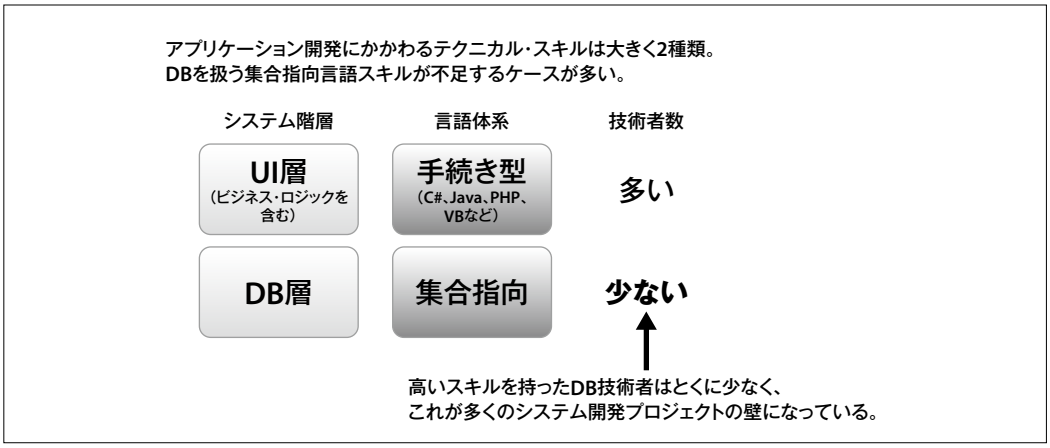
「あったり前やないか。やったらできるもんやで。やるか？」

「やります！」

「今からがんばったら、大阪で3本の指に入る技術者になれるわ」

若いエンジニアがその気になったのなら、オジサンも頑張らなければいけません。というわけで、本連載では第2のポイントである「技術者育成」に役立つよう、現場の技術者の悩みの種である性能トラブル事例をもとにして、集合指向言語としてのSQLの特性を理解しやすい技術解説を提供します。また第1のポイントである組織体制についての提言も随時行いますので、よろしくご期待ください。SD

▼図5　集合指向言語と手続き型言語の役割分担



COLUMN 実行計画とは

今回は詳しく触れませんでしたが、二重ループを使うと負荷が重くなりやすい理由の1番目の指摘に出てきた「実行計画」は、Javaのような一般のプログラミング言語とSQLの違いを理解するうえでの重要なキーワードの1つです。一般の言語の場合は、最終的にほしい「結果」を得るために、途中で行う処理の「アルゴリズム」をプログラマが考えてソースコードにそれを書きます。それに対して、SQLの場合は「アルゴリズム」を考えるのはDBエンジン（のオプティマイザ）の役割です。SQL文は「最終的にこんなデータがほしい」という結果のイメージをプログラマからDBエンジンに伝えるために書くもので、実際の処理アルゴリズムはデータの状況に応じてDBエンジンが動的に生成します。そうして動的に生成された「アルゴリズム」を「実行計画」と呼ぶわけです。この意味でSQLはインタプリタ言語の一種と言えます。DBの動きを知るためには実行計画の理解は欠かせません。この件は、今後の本連載で別途詳しく扱います。

コミュニティメンバーが伝えるAndroidで広がるエンジニアの愉しみ

presented by
Japan Android Group
http://www.android-group.jp/

第3回 Nearbyとコミュニティ運営

三宅 理（みやけ おさむ）
日本Androidの会運営委員・
日本Androidの会埼玉支部

Androidは世界で出荷される8割のスマートフォンに搭載※される、事実上スマートフォンの標準OSです。このOS上で動くアプリケーションを開発することは、自分のアイデアを世界に発信し、最も多くのスマートフォン上で動かしてもらえる可能性がある行為です。このAndroidの最新情報と、それを取り巻くコミュニティを知って、魅力的な開発をはじめませんか？

※IDC Worldwide Mobile Phone Tracker, August 7, 2013

小さなデータを端末間でサクっとやりとり

新しいコミュニティに入ったり、勉強会で隣になった人とSNSのIDを交換するのは度胸がいりますよね。IDを交換しなくても手軽にメッセージやファイルを交換できたら便利だと思いませんか？ iOSやOS Xでは、AirDrop機能を使ってこのようなデバイス同士の連携ができます。iPhoneとiPhone、iPhoneとMacBookなどのデバイスで写真や音楽などのファイルを共有することができるわけです。

Androidには標準でそのような機能はありません。そこで今回は、Androidでもデバイス同士の連携をすることができるAPI「Nearby」を紹介します[注1]。

Nearby

Nearbyは、Wi-FiやBluetooth、あるいはスピーカーとマイクを使って近くのAndroidと通信をするための、Googleが開発したサービスです。Android 2.3以上に対応しています。またiOS版も用意されていますので、iOS端末と通信するアプリを作ることもできます。

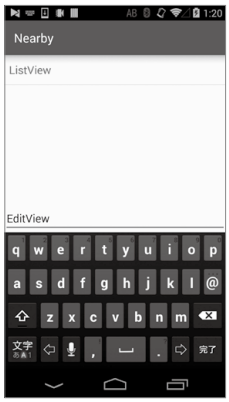
NearbyではNearby MessagesとNearby Connectionsの2つのサービスを提供しています。実際の利用ではGoogle Play Servicesの機能として動作します。そのため「Google Play開発者サービス」のアプリケーションがインストールされているAndroid端末で動作します[注2]。

Nearby Messagesは端末から、近接する複数のAndroid端末に向けてメッセージの送信ができます。送信は一方通行のため、リクエストしたデータに対してレスポンスを受け取るといった双方向の通信はできません。また通信経路はWi-FiやBluetoohを使うため、同じネットワークに接続していなくても通信が可能です。データはバイナリで送信します。文字列以外にもデータを送れるのでアプリのデータや制御にも使えます。

Nearby Connectionsでは同じネットワークに接続している1台のAndroid端末がホストとなり、それ以外のAndroid端末はクライアントとして動作します。これはマルチスクリーンを用いたマルチプレイのゲームや、複数の端末で同期をとって動作するようなアプリケーションで使う機能になります。ルータの機能でマルチキャストが有効になっている必要があります。実際に動かしたときに動作しない場合はルータの設定を確認してみてください。

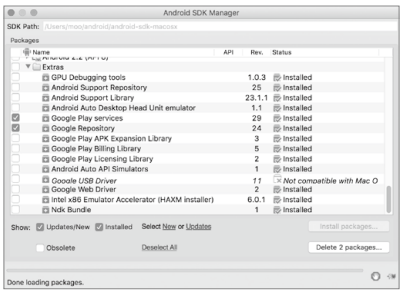
注1）本稿で解説するサンプルコードが、本誌サポートサイトからダウンロードできます。http://gihyo.jp/magazine/SD/archive/2016/201603/support

注2）インストールされていない端末の例として、Amazon Kindle Fireが挙げられます。



◀図1　本稿で解説するNearbyアプリのイメージ

▼図2　Android SDK Managerからのインストール



Nearby Messagesを使うための準備

本稿では**図1**のような、EditViewで文字を入力すると、Nearby Messagesを使ってメッセージを相手に送り、受け取った側はListViewでメッセージを表示するアプリを作ってみましょう。開発の環境として、Android StudioとAndroidの実機を2台用意する必要があります。スマートフォンとタブレットの組み合わせでも大丈夫です。

Android SDK Manager

Nearbyは、Google Play Servicesの中にAPIが含まれています。Android SDK Managerを起動して「Google Play Services」「Google Repository」がインストールされているかチェックしてください。インストールされていない場合は、**図2**のように2つの項目をチェックして「Install Packages」ボタンを押します。

APIキーの取得

Nearby Messagesを利用するためには、Google Developer ConsoleでAPIキーの取得が必要になります。Googleアカウント(Gmailなどを利用する際に使うアカウント)を持っていない場合は、アカウントを作成してください。Google Developer Console[注3]にアクセスします(**図3**)。

「続行」ボタンを押して新しいプロジェクトを作成します。認証情報画面でパッケージ名とSHA-1証明書のフィンガープリントを入力します(**図4**)。ここでは例としてパッケージ名「com.example.nearby」とします。SHA-1証明書のフィンガープリントは、APKファイルを署名する際に利用する証明書のハッシュ値です。Android Studioで開発をしている場合、デバッグ用のキーが次に示すパスにあります。

➡Macの場合

```
keytool -list -keystore ~/.android/debug.keystore
```

➡Windowsの場合

```
keytool -list -keystore C:¥Users¥ユーザ名¥.android¥debug.keystore
```

▼図3　Nearby Messages APIキーの取得

▼図4　Google Developer Console(認証情報)

注3) https://console.developers.google.com/flows/enableapi?apiid=copresence&keyType=CLIENT_SIDE_ANDROID&reusekey=true

▼図5　証明書のハッシュ値を取得

```
DEV-no-Mac:~ DEV$ keytool -list -keystore ~/.android/debug.keystore
キーストアのパスワードを入力してください:

キーストアのタイプ: JKS
キーストア・プロバイダ: SUN

キーストアには1エントリが含まれます

androiddebugkey,2016/01/08, PrivateKeyEntry,
証明書のフィンガプリント(SHA1): 6E:40:61:98:F9:00:93:D3:EF:2B:02:F2:3A:AD:58:94:7F:F1:52:DC
DEV-no-Mac:~ DEV$
```

▼リスト1　build.gradleに追加するコード

```
dependencies {
    compile 'com.google.android.gms:play-services-nearby:8.4.0'
}
```

▼リスト2　Androidmanifest.xmlに追加するコード

```
〈manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.nearby" 〉
    〈application ...〉
        〈meta-data
            android:name="com.google.android.nearby.messages.API_KEY"
            android:value="APIキー" /〉
        〈activity〉
        ...
        〈/activity〉
    〈/application〉
〈/manifest〉
```

今回はこのデバッグキーを使ってフィンガープリントを登録します(**図5**)。keytoolを実行[注4]した際に入力するパスワードは「android」です。フィンガープリントの登録を行うと、APIが発行されます。このキーを後ほど使うのでメモしておいてください。

プロジェクト作成

Android Studioを起動して新しくプロジェクトを作成します。新規プロジェクト作成時のウィザードの中で「Add an activity to mobile」は、「Empty Activity」を選択して作成します。作成したプロジェクトのappフォルダの中にあるbuild.gradleのdependenciesの中に、**リスト1**のコードを追加します。

GradleのSyncをするとNearbyを使う準備ができました。次に、Androidmanifest.xmlの中に**リスト2**の内容を追加します。その際「APIキー」と書かれた部分は、発行したAPIキーを入力します。

Nearby Messagesを使う

Android Studioで新しくプロジェクトを作成します。その際パッケージ名はAPIキーを取得したときに使用した「com.example.nearby」を使います。

Google開発者サービスの利用方法

NearbyはGoogle Play Servicesに含まれるAPIを利用するため、**リスト3**のようにActivityのonCreateメソッド内でGoogleApiClientのインスタンスを生成して、onStartメソッド内で接続しています。その際APIを指定するため、addApiメソッドでNearby.MESSAGES_APIを指定しています(**リスト3**①)。

初回起動時のパーミッション確認

初回の呼び出しでは、パーミッションの確認ダイアログが表示されます(**図6**)。許可ボタンを押すことでNearbyの機能を使うことが可能です。権限が許可されない場合、アプリ内でNearbyの機能を使うことはできません。

パーミッションの確認は、後述するPublishとSubscribeメソッドを呼び出した際のエラー処理で判定して、ダイアログを表示するようにしています(**リスト4**)。

Publish(メッセージの送信)

メッセージを送信するときは、Nearby.Messages.publishメソッドを呼び出します(**リスト5**)。メッセージはバイトコードで送信するため、文字列以外のデータも送信可能です。Messageクラスのインスタンスを生成するときに送信するデータを定義しています。Nearby.Messages.

注4) keytoolが起動しない場合は、JDKのbinフォルダのPATHを設定してください。

publishメソッドでメッセージの送信を行っています。公式サイトにはデータのサイズを3KBまでにすることが望ましいと書かれています。大きなデータサイズになる写真や動画などのデータを送信する用途には適しませんのでご注意ください。

▼リスト3　Nearby Messages APIの指定

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

        mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addApi(Nearby.MESSAGES_API)  ←①
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .build();
}

@Override
protected void onStart() {
    super.onStart();
    mGoogleApiClient.connect();
}

@Override
public void onStop() {
    super.onStop();
    if (mGoogleApiClient != null && mGoogleApiClient.
isConnected()) {
        mGoogleApiClient.disconnect();
    }
}
```

▼リスト4　パーミッションの確認判定

```
private void handleUnsuccessfulNearbyResult(Status status) {

    if (mResolvingError) {
        return;
    } else if (status.hasResolution()) {
        try {
            mResolvingError = true;
            status.startResolutionForResult(this,
                    REQUEST_RESOLVE_ERROR);
        } catch (IntentSender.SendIntentException e) {
            mResolvingError = false;
        }
    } else {
        if (status.getStatusCode() == CommonStatusCodes.NETWORK_ERROR) {
            Toast.makeText(getApplicationContext(),
                    "ネットワークに接続できませんでした。設定アプリを開いて確認してください",
                    Toast.LENGTH_LONG).show();
        } else {
            // エラー発生
        }
    }
}
```

Subscribe(メッセージの受信)

Publishで送信したメッセージを受け取るには、Nearby.Messages.subscribeメソッドを実行します(**リスト6**)。Strategyクラスのインスタンスに使うパラメータ(DiscoveryMode、DistanceType、TTLSeconds)は公式ドキュメントを参照ください。

画面デザイン

今回はListViewとEditTextの2つのViewを使ったシンプルな構成です。デザインXMLは誌面の関係上掲載できなかったので、Webからダウンロードしたコードを参照ください。

▼図6
Nearbyパーミッション確認ダイアログ

Nearbyのまとめ

実際にアプリを動かしてみます。メッセージを入力して完了ボタンを押すと、数秒後にもう一方の端末に入力したメッセージが表示されると思います。今回は簡単にするために文字列を送信していますが、バイナリデータであればなんでも送信可能です。Nearbyの機能を使って皆さんもAndroid端末同士で通信するプログラムを書いてみましょう。

開発者のコミュニティ運営

コミュニティへ参加

IT技術を学ぶなら今では書籍やインターネットがあります。日本の場合、日本語の書籍がたくさんあるので勉強するだけなら1人でも可能です。知識の幅を広げたい、技術以外も学びたいと思ったら一度はオフラインコミュニティへ参加してみることをお勧めします。自分が興味あるテーマを扱っているコミュニティの勉強会を一度のぞいてみましょう。同じ技術、テーマで切磋琢磨している人がたくさんいます。

コミュニティでは技術そのものよりも、ロジックや考え方、つまづいたときのフォローを学ぶことがあります。それがコミュニティに参加する楽しみだと筆者は思います。

コミュニティを運営

自分が参加したいコミュニティがないときはどうしたらいいのでしょうか。その場合はコミュニティを自分で作るという方法があります。自身が発起人になり、コミュニティを作って紹介用のサイトを用意します。コミュニティに関する情報があると参加する人が情報収集する際に活用できます。また、検索サイトから見つけてもらえる可能性が上がります。自身のTwitterなど、SNSで告知するのも効果的です。これでコミュニティのできあがりです。後は定例会や勉強会を開催していきます。

コミュニティを作るのは比較的簡単です。一方、運営は大変です。定期的に活動してないと活動しているコミュニティと認知されません。定例会などのイベントに備えてネタを作り、会場を設定してイベント告知を行い、人を集めないといけません。

運営の中でも一番大変なことは人を集めることです。集客はもちろんですが、一緒にコミュ

▼リスト5　メッセージ送信部

```
private void publish(String strMessage) {
    mMessage = new Message(strMessage.getBytes());

    if (!mGoogleApiClient.isConnected()) {
        if (!mGoogleApiClient.isConnecting()) {
            mGoogleApiClient.connect();
        }
    } else {
        PublishOptions options = new PublishOptions.➥
Builder()
            .setCallback(new PublishCallback() {
                @Override
                public void onExpired() {
                    // 配信されなかった時の処理
                }
            }).build();

        Nearby.Messages.publish(mGoogleApiClient, ➥
mMessage, options)
            .setResultCallback(new ResultCallback<Status>() {
                @Override
                public void onResult(Status status) {
                    if (status.isSuccess()) {
                        // 配信した時
                    } else {
                        // 配信されなかった時
                        handleUnsuccessfulNearbyResult(➥
status);
                    }
                }
            });
    }
}
```

ニティを運営してくれる人たちがいると助かります。1人で開催していくのには限界があります。

まとめ

日本Androidの会でも毎月定例会を開催しています。スピーカーを招待して会場を押さえる必要がありますが、これらを運営委員で行っています。これらを毎回設定するのがとても大変です。

大変なことをあえてする理由はなんなのでしょうか？ 筆者は新しいことを知りたい欲求が高く、そのためならなんでもするということで動いているからだと思います。そして、そういった人達の集まりが運営を動かしているのだと思います。

皆さんも興味があるコミュニティがあればぜひ参加してみてください。また、新しいコミュニティを作ってみてください。そうすると世界が広がると思います。SD

COLUMN

Android Topics

Android Bazaar and Conference 2016 Spring

開催日：2016年3月12日
場 所 ：東京都 青山学院大学
テーマ：IoTの発展に向けたAndroidの新たな役割

日本Androidの会が主催するAndroid Bazaar and Conference（ABC）の2016年春開催のイベント。開発者を中心にしたAndroidの総合イベントでセミナー形式のカンファレンスと展示会形式のバザールから構成される。無料で誰でも参加できる。今回はIoTの発展に向け、「神経回路たるAndroidスマートフォンとかかわるセンサ機器・組込み機器」「BLEや将来の5Gも含めた無線ネットワーク」「クラウド側のディープラーニングをはじめとする機械学習機能」に焦点を当て、Androidを中心とするIoTの体系的なカンファレンスを予定。加えて、センサ、スマートフォン、クラウドの連携を進め、人類の機能拡張の可能性を感じられるバザール出展を集めた祭典を目指している。参加者と同時にバザールの出展希望者、運営スタッフも募集中。
http://abc.android-group.jp/2016s/

▼リスト6 メッセージ受信部

```
private void subscribe() {
    Strategy strategy = new Strategy.Builder()
        .setDiscoveryMode(Strategy.DISCOVERY_MODE_DEFAULT)
        .setDistanceType(Strategy.DISTANCE_TYPE_DEFAULT)
        .setTtlSeconds(Strategy.TTL_SECONDS_INFINITE)
        .build();

    SubscribeOptions options = new SubscribeOptions.Builder()
        .setStrategy(strategy)
        .setCallback(new SubscribeCallback() {
            @Override
            public void onExpired() {
                // タイムアウト
            }
        }).build();

    Nearby.Messages.subscribe(mGoogleApiClient, mMessageListener, options)
        .setResultCallback(new ResultCallback<Status>() {
            @Override
            public void onResult(Status status) {
                if (status.isSuccess()) {
                    // 購読成功
                } else {
                    // 購読失敗
                    handleUnsuccessfulNearbyResult(status);
                }
            }
        });
}
```

一歩進んだ使い方のためのイロハ

Vimの細道

第6回

mattn
twitter:@mattn_jp

mapを極める者がVimを制す

今回はエディタとしての基本に立ち返り、キーに対する機能の割り当て「map」について解説していきます。一口にmapといっても、適用するモードによって、また「再マップ」の許可／不許可によってそれぞれコマンドが分岐し、さらに修飾子を加えることで特殊な動作が可能になるなど、複雑で奥が深い機能です。

一般的なキーのカスタマイズとは

一般的に、カスタマイズが可能なテキストエディタには、キーのカスタマイズ機能が存在します。Vimはもちろん、Emacs、秀丸、Sakuraエディタ、gedit、EmEditor、あのnanoでさえキーのカスタマイズができます。

多くのテキストエディタが提供するキーのカスタマイズ機能はキーボードショートカットと呼ばれ、Ctrlや Shiftといった修飾キーと別のキーを同時に押すことでテキストエディタの機能を呼び出す、といった連携機能を提供しています。

かたやVimやEmacsでは、キーストロークという複数のキーの組み合わせにより、1回だけのキー押下よりも数多くのアクションをアサインできるようになっています。とくにVimにおいては、文字を入力するインサートモードに限ったものではなく、ノーマルモードやコマンドモードにおいても、モード別のキーをアサインできるため、無限のカスタマイズを行えるようになっています。Vimでは、このキーに対する機能の割り当てを「map」と呼び、ユーザに細かいカスタマイズができるようにコマンドやオプションを提供しています。

しかしながら、皆さんはこのmapを十分に活かし切れているでしょうか？　筆者が知る限り、Vimのカスタマイズを極めている人の多くは体系立ったキーマッピングを施しており、多少キーアサインを忘れてしまっても簡単に思い起こせるような設定を行っています。キーのカスタマイズを極めることで編集操作のバリエーションに幅が生まれ、Vimの機能を何倍にも引き出せるのです。

今回は、mapのことは知っているけどあやふやだったという人にもわかりやすく、mapとは何か、どんな機能なのか、どんな応用方法があるのかを説明したいと思います。

そもそもmapってどんな機能？

前述したとおり、Vimのmapはモード別に用意された機能とキーの割り当てを意味します。

imap

インサートモードでは多くの場合、タイプした文字がそのまま入力されることが期待されるため、あまりストロークは使用しません。できれば1キーで機能が呼び出せるようファンクションキーを使ったり、Ctrlを使用したりします。

簡単な例から説明します。たとえば、インサートモードでCtrl-F（以降<C-f>と表記）を押したとき、「World」という文字を入力するmapを

定義してみます。

```
imap <C-f> World
```

コマンドモードでこれを実行し、インサートモードで<C-f>をタイプすると「World」が入力されるはずです。ここではimapというコマンドを実行しましたが、これはインサートモード(i)でのキーのマッピング(map)を意味します。imapのヘルプ(:help imap)では、

```
imap {lhs} {rhs}‘
```

のように表記されており、{lhs}(Left Hand Side)がタイプされると{rhs}(Right Hand Side)として再生される、という意味になります。

次に、以下のコマンドを実行します。

```
imap <C-b><C-f> Hello
```

これは、<C-b>のあとに続けて<C-f>をタイプすると「Hello」を入力するmapを登録するコマンドです。インサートモードで<C-b><C-f>をタイプすると、期待どおりに「Hello」が入力され、続けて<C-f>をタイプすると「World」が入力されます。ここで注意してほしいのが、先にimapで登録した<C-f>が、あとに登録した<C-f>と被っていることです。

このように、Vimは複数のキーからなるストロークをmapコマンドにより登録できます。さらにはmapから、異なるmapの{lhs}を呼び出すこともできるのです。

再マップとinoremap

先に説明した{lhs}に対して、別の{lhs}を割り当てられます。

```
imap <C-z> <C-b><C-f><C-f>
```

インサートモードで<C-z>をタイプすると「HelloWorld」が入力されます。複数のmapを1つのmapに置き換えられました。これを「再マップ(remap)」と言います。もし逆に、remapを行いたくない場合は、

```
inoremap <C-z> <C-b><C-f><C-f>
```

と実行してください。<C-z>をタイプすると、画面に「^B^F^F」という本来のキーコードがそのまま入力されるはずです。inoremapコマンドはインサートモードでremapを行わないという意味で「i-no-remap」と覚えてください。

nmap/nnoremap

imap/inoremapと同様に、ノーマルモードの機能を呼び出します。

```
nmap <C-b> iHelloWorld<ESC>
```

<C-b>をタイプすると、iでインサートモードに入り、カーソル位置に「HelloWorld」を入力します。inoremapと同様、remapしたくないのであればnnoremapを使います。

コマンドモードでもimap/nmap同様に、cmapとcnoremapでキーアサインを行い、ビジュアルモードではvmapとvnoremapを使います。

◆ ◆ ◆

これまでの説明では基礎を学ぶため意図的にimap/nmapを使用しましたが、remapを行いたい意図がとくにない場合には、inoremap/nnoremap/cnoremapを使うべきです。

また、これらのmapはVimを終了すると消えてしまいます。起動時に使いたいのであれば、mapコマンドをvimrcに記述してください。

cmap/cnoremap

コマンドモードに作用します。たとえば、コマンドモードでEmacsのように先頭と末尾へのカーソル移動を<C-a>、<C-e>にしたい場合は、次のように実行します。

```
cnoremap <C-a> <Home>
cnoremap <C-e> <End>
```

これは、もともと<Home>や<End>に割り当てられている機能をmapしています。

vmap/vnoremapとxmap/xnoremapの違い

vimrcにvmapやvnoremapを設定している人はよくいますが、xmap/xnoremapを設定している人は意外と見かけません。違いを確認するために、次のmapを実行してみます。

```
nnoremap j gj
nnoremap k gk
vnoremap j gj
vnoremap k gk
```

このmapは、ノーマルモードおよびビジュアルモードでのj/k移動を、wrapによる行の折り返し時にも1行ずつ移動させるようにするための設定で、ほとんどの場合これは正しく動作します。 ただし、このmapには1点だけ間違いがあります。

Vimのビジュアルモードは、選択中に<C-g>をタイプするとセレクトモードに移行します。このセレクトモードは、一般的なテキストエディタと同様に印字可能な文字がタイプされると選択部分が削除され、インサートモードになるモードのことです。試しに、このmapを実行したあとでビジュアルモードから<C-g>でセレクトモードに移り、jをタイプしてみてください。本来であれば、選択部分が削除されjが入力されるはずですが、期待どおりに動作しません。実はvmap/vnoremapはビジュアルモードとセレクトモードの両方に作用するmapなのです。よってこのようなケースでは、セレクトモードの動作を壊さないようにxmap/xnoremapを使わなければなりません。

smap/snoremap

前述のxmap/xnoremapに対して、逆にセレクトモードだけに適用したい場合にはsmap/snoremapを使用します。

omap/onoremap

omap/onoremapは入力待ち(Operator Pending)状態の場合にのみ作用します。次から、テキストオブジェクトとomap/onoremapを使った簡単な例を示していきます。

テキストオブジェクトって何だ

ここでひとまず、そのテキストオブジェクトについて説明します。

```
int x = do_something(dog, cat, cow);
int y = do_everything();
int z = do_nothing();
```

このソースコードの「cat」の部分でdi(をタイプしてみてください。括弧の中身が消えるはずです。次に1行下の「do_everything()」の「(」の部分に移動し、pをタイプしてください。

```
int x = do_something();
int y = do_everything(dog, cat, cow);
int z = do_nothing();
```

「do_something」の引数が、すべて「do_everything」の引数へ移動しました。Vimを知らない人にとっては、このテキストオブジェクトという機能は魔法のように見えると言われます。

たとえば、元のソースコードにて「do_something」の引数を「food」だけに変更する場合、引数部分でciwfoodとタイプします。続けて、「do_everything」「do_nothing」の引数部分で「.」をタイプすると、すべての関数の引数が「food」だけになります。

Vimの操作は、yやd、cといったオペレータと、それに続くwや$といったモーションで構成されます。テキストオブジェクトはwや$といった一方向のモーションではなく、カーソル位置のコンテキストに依存した、もう少し複雑なモーションとして作用します。ほかにも便利なテキストオブジェクトがありますが詳細は:help text-objectsを見てください[注1]。

注1) 本誌2015年10月号に付属する、創刊300号記念チートシートにも、いくらか便利なテキストオブジェクトがまとめられています。

◇本題へ

さて、このテキストオブジェクトのi(はdをタイプしたあとの入力待ち状態でタイプするのですが、omap/onoremapはこの入力待ちに作用するmapです。

```
:onoremap p i(
```

のように設定すると、ノーマルモードでのpには反応しませんが、dpというコンビネーションでdi(を実行できます。また、/もモーションとして扱えるので、

```
:onoremap ; /;¥zs<CR>
```

とすることで、カーソル位置から「;」までをd;で削除できるようになります。

mapコマンドは、いろいろな機能を持ち合わせた修飾子を付与することで、特殊なmapを実行できます。

◇修飾子<expr>

imapコマンドを<expr>修飾を付けて実行すると、{rhs}を式として評価した結果が再生されます。たとえば、インサートモードで<C-b>をタイプしたときに現在の時刻を入力させたいのであれば、次のように実行します。

```
inoremap <expr> <C-b> strftime('%c')
```

実は先ほど紹介したj/kで行を1行分移動するmapは、3jや2kといったカウントを指定した場合に正しく動作しません。そこで<expr>を使い、カウントv:countが指定されていない場合のみgj/gkを再生するmapを実行します。

```
nnoremap <expr> j (v:count == 0 ? 'gj' : 'j')
nnoremap <expr> k (v:count == 0 ? 'gk' : 'k')
xnoremap <expr> j (v:count == 0 ? 'gj' : 'j')
xnoremap <expr> k (v:count == 0 ? 'gk' : 'k')
```

◇修飾子<buffer>

<buffer>修飾を付けると、そのmapが現在のバッファにのみ適用されます。たとえば、バッファがJavaのソースコードであった場合だけ、<C-b>で現在の時刻を入力させるmapを行いたいのであれば、**リスト1**をvimrcに追加します。

みなさんがよく使う、特定のファイルタイプにだけ作用するVim pluginのキーマッピングは、このように実装されています。

◇修飾子<silent>

<silent>修飾を付けると、mapが再生されているあいだ、あらゆるメッセージの表示(エラーメッセージを除く)が抑制されます。たとえば、バッファから「HelloWorld」を検索するmapは次のように書けます。

```
nnoremap <silent> ,h /HelloWorld<CR>
```

<silent>を付けることで検索コマンドラインに「/HelloWorld」が表示されなくなります。

◇修飾子<nowait>

次のmapを実行してみてください。

```
imap <C-b><C-f> Hello
imap <buffer> <C-b> World
```

1つめのmapは、すべてのバッファに適用されます。これをグローバルマッピングと言います。また2つめのmapは、現在のバッファにのみ適用されます。

この場合、後者を設定したバッファでは<C-b>をタイプしても、即座に「World」が入力されません。これはグローバルマッピングとバッファ

▼リスト1　バッファがJavaのコードの場合適用されるmap

```
augroup JavaKeyMapping
  autocmd!
  autocmd FileType java inoremap <buffer>ア
<expr> <C-b> strftime('%c')
augroup END
```

ローカルマッピングで<C-b>がバッティングしているためです。Vimは<C-b>がタイプされたあと、後続の<C-f>が入力されるのを一定時間待ち、入力されなければ「Hello」を、<C-f>が入力されれば「World」を入力します。この入力待ち時間の長さはtimeoutlenオプションで変更できます。

「このコマンドを実行したバッファでは、グローバルマッピングよりもバッファローカルマッピングを優先したい」といった場合、<nowait>修飾を利用します。

```
imap <C-b><C-f> Hello
imap <buffer> <nowait> <C-b> World
```

こうすることで、<buffer>を指定して実行したバッファでは<C-b>をタイプしても入力待ちが発生しなくなります。

◇修飾子<unique>

mapは基本的に上書き登録されます。プラグインを複数登録している場合や、気づかず同じキーに別の機能をmapしてしまった場合、期待しない設定により事故が起きる可能性があります。キーがバッティングする可能性がある場合は、次のように<unique>を付けて、意図的にエラーを発生させることが推奨されます。

```
nnoremap <unique> ,, :e .<CR>
```

◇修飾子<script>

この修飾子は、mapに直接作用しません。たとえば、

・<Space>では何もしない
・<Space>wでwrapのトグル
・<Space>lでlistのトグル

というキーマッピングを行いたいとします。簡単に、それぞれのmapを<Space>を使って書いても良いのですが、ときにはこのプレフィックスを簡単に変更したいこともあります。しかし、目的は「トグル動作」であることから、できれば抽象化して定義したいものです。こういった場合に、<script>修飾子を使用します[注2]。

```
nnoremap <script> <Space> <SID>[Toggle]
```

これにより<Space>に[Toggle]というマッピング名がアサインされます。<Space>をタイプした際に画面右下に表示されるキーにも[Toggle]と表示されます。

あとは、このプレフィックスを使ってトグル動作を行うマッピングを行います(**リスト2**)。プレフィックスを別のキーに変更する場合でも1行で済むようになります。

もう1つ例を示します。通常、Vimで画面内のウィンドウをリサイズするには<C-w>のあとに+ - > <のどれかをタイプします。つまり1回のリサイズに2個のキーをタイプする必要があります。しかも、1つめはCtrlと同時に押す必要があり、2つめはShiftを同時に押さなければなりません。リサイズで微調整を行う場合には、非常に面倒な入力方法です。かと言って、1キーだけで動作するmapを登録してしまうとウィンドウリサイズのために計4つものキーが消費されてしまいます。

そこで、先ほど説明したプレフィックスキーを使ったmapが必要になります(**リスト3**)。はじめの4つは<C-w>と+ - > <で構成されるウィンドウリサイズのキーと、<SID>wsという<script>修飾子を指定した、残りの4つのmapへのプレフィックスを付加して再生しています。これにより、どれかをタイプしてリサイズが行われたあとに意図的な入力待ちが発生するようになります。ここでユーザが、+ - > <をタイプすると、さらにウィンドウリサイズが行われ、再度入力待ち状態となります。

つまりユーザはリサイズをするのに、<C-w>+++++---->>>><<などと入力すれば良いことになります。<C-w>と+や<などを交互にタイプする必要がなくなるので、簡単にウィンドウ

注2) <script>修飾子や<SID>修飾子のあるコマンドは.vimrcに記述して試してください。

のリサイズを行えるようになります。また、この<script>が付いた<SID>wsは、このmapが実行されるスクリプトの外からは参照できないようになっているため、一連のしくみを隠蔽できるようになっています。

この手法はサブモードと呼ばれており、tinymode.vimやsubmode.vimといったプラグインで簡単に登録できるようになっています。

<leader>プレフィックスキー

前述のプレフィックスをもう少し簡単に設定する方法があります。Vimにはmapleaderという変数が用意されており、この変数に登録されたキーが<leader>という修飾子に置き換えられます。

```
nnoremap <leader>, :cwindow <Bar> cc<CR>
nnoremap <leader>h :help<CR>
```

デフォルトではmapleaderは<C-\>になっているので、このmapは、

```
nnoremap <C-\>, :cwindow <Bar> cc<CR>
nnoremap <C-\>h :help<CR>
```

に置き換えられます。たとえば、この<C-\>を,で置き換えたい場合には、一連のmapコマンドを実行する前に、

```
let mapleader = ','
```

を設定しておけば良いことになります。

<plug>マッピング

たとえば、みなさんがVim pluginを作ったとしましょう。そして、便利な関数を作ってユーザからmapで呼ばせたいとします。その際、

```
nnoremap ,G :call sugoi_plugin#Sugoi(1)
```

と書いてしまうと、開発者は以後sugoi_plugin#Sugoi(1)を名称変更したり、引数の構成を変更したりできなくなります。そのためにVimでは、<plug>マッピングという機能を提供しています。

▼リスト2　トグル動作を実現するmap

```
nnoremap <script> <Space> <SID>[Toggle]
nnoremap <SID>[Toggle] <Nop>
nnoremap <SID>[Toggle]w :setlocal wrap!<CR>
nnoremap <SID>[Toggle]l :setlocal list!<CR>
```

▼リスト3　ウィンドウのリサイズを簡単にするmap

```
nmap <C-w>+ <C-w>+<SID>ws
nmap <C-w>- <C-w>-<SID>ws
nmap <C-w>> <C-w>><SID>ws
nmap <C-w>< <C-w><<SID>ws
nnoremap <script> <SID>ws+ <C-w>+<SID>ws
nnoremap <script> <SID>ws- <C-w>-<SID>ws
nnoremap <script> <SID>ws> <C-w>><SID>ws
nnoremap <script> <SID>ws< <C-w><<SID>ws
nmap <SID>ws <Nop>
```

まずプラグイン開発者は、sugoi_plugin#Sugoi(1)に対するインターフェースを外部に公開するために、次のmapを行います。

```
nnoremap <plug>(sugoi-command) :call sugoi
_plugin#Sugoi(1)
```

そして、このプラグインを使用するユーザに対しては<plug>(sugoi-command)へmapするように指示します。

```
nmap ,G <plug>(sugoi-command)
```

これによりプラグイン開発者は、気にせずsugoi_plugin#Sugoi(1)のインターフェース変更を行えるようになります。

◆　◆　◆

mapの基本的な機能説明から、常套テクニックに至るまでを解説しました。応用例はまだまだあります。次のURLで毎週土曜の夜11時から「vimrc読書会」が開催されており、筆者も時おり参加しますが、まれに「へぇこんなmap見たことなかった」といったものが登場します。興味のある方はぜひ参加してみてください。SD

・vimrc読書会
http://vim-jp.org/reading-vimrc

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23回 EmacsでGitを使う!

Emacsには、「vc」というバージョン管理システムとの連携を行う標準パッケージがあります。RCS、CVS、Subversion、Gitに対応しているvcですが、今回はGitを選んで、リポジトリの作成、ファイルの登録・コミット、履歴管理の方法について解説していきます。

バージョン管理システムとEmacs

ども、るびきちです。今回はEmacsでバージョン管理システムを扱うお話となります。Emacsは昔から、バージョン管理システムとの連携がとてもうまくできるようになっています。それは「vc」という標準パッケージのおかげです。

vcを使えば、バージョン管理システムの基本的な操作――コミット・diff・履歴参照など――が簡単に、しかも、どのツールを使っているかに関係なく共通した操作で行えます。そのため、使っているバージョン管理システムに詳しくなくても、Emacsの中で簡単に操作できるというメリットがあります。

バージョン管理システムには、原始的なものから先進的なものまで、本当にたくさんのソフトウェアが存在します。フリーのもので言えば、1ファイルしか管理できないRCSから始まり、複数ファイルを管理できるように進化したCVS、そしてCVSの欠点を克服したSubversionが登場しました。今はGitが主流ですね。vcは、これらに挙げたバージョン管理システムすべてに対応しています。

おすすめはGit

Gitは先進的な分散バージョン管理システムで、完璧にマスターするのはたいへんです。Gitはもともと Linux カーネル開発のために開発され、実運用されているので信頼性は高く、大規模開発にも耐えられます。一方、1ファイルからの小規模な用途でも手軽に使えます。

vcを使えば過去にほかのバージョン管理システムを使っていても、Gitに簡単に移行できます。実際に、筆者も長期に渡ってRCSとSubversionを使っていましたが、vcを介して使っている限りはGitでも違和感なく使えました。

それだけではありません。日頃からGitを使っていれば、そのリポジトリを簡単にGitHubで公開できます。GitHubで公開してしまえば、思いがけないpull requestを受けられます。また、MELPAによるパッケージの公開も簡単になります。非公開リポジトリならば、Bitbucketで無料で何個でも持てます。

そういう理由で、デフォルトで使うバージョン管理システムはGitがいいのです。

vcとGitでファイルの変更記録を付ける

vcは簡単!

vcはEmacs標準添付ですので、何の設定もせずにバージョン管理を始められます。

また、vcは汎用的に作られているため、使っているバージョン管理システムに依存せずに共

通の操作を行えます。その代わり、バージョン管理システム固有の先進的な機能はサポートされていません。vcのすばらしいところは、C-x v系のコマンドを実行すればすぐに使うことができることです。そのうえ、C-x v v(vc-next-action)が空気を読んでくれるので、これさえ知っていればほかに何も知らなくても、変更履歴を記録できます。

リポジトリ作成・ファイル登録・コミット

それでは、Gitとvcを使って1つのファイルをバージョン管理下に置いてみましょう。

```
C-x C-f /tmp/vc-first/greeting.txt
```

を実行し、新規ディレクトリにファイルを作成します。/tmp/vc-firstは存在しないディレクトリですので、エコーエリアの指示どおりM-x make-directory RET RETでディレクトリを作成してください。そこでファイルの内容に、

```
Hello!
```

と書いておきます。

greeting.txtをバージョン管理するために、最初に使うコマンドは万能なC-x v vです。すると、初めてバージョン管理をするので、どのバージョン管理システムを使うか聞いてきます。

```
/tmp/vc-first/greeting.txt is not in a ➥
version controlled directory.
Use VC backend:
```

これには「Git」と答えます。次に、どのディレクトリをGit管理下に置くかを聞いてきます。

```
create Git repository in: /tmp/vc-first/
```

リポジトリというのはGitが管理する変更履歴データベースのことで、範囲はリポジトリが置いているディレクトリおよびサブディレクトリです。この場合はそのまま Enter を押します。すると、新しくGitリポジトリが作成され、greeting.txtがリポジトリに登録されます。エコーエリアには次のように表示されます。

```
Registering (/tmp/vc-first/greeting.txt)➥
... done
```

ただ、この状態は「ファイルを管理下に置きます」と宣言しただけで、ファイルの内容は登録されていません。最初のC-x v vで、内部的に、

```
—git init
—git add greeting.txt
```

が実行された状態です。ファイルの内容を登録する(コミット)には、再度C-x v vを実行します。すると、*VC-log*バッファが出てきます。

```
Enter a change comment.  Type C-c C-c ➥
when done
```

と表示されますが、これは「*VC-log*バッファには変更した内容のメッセージを書いてからC-c C-cを押せ」という指示です。この場合は新規登録ですので、

```
greeting.txt: New
```

と書いてC-c C-cを押しましょう。内部では、

```
—git commit -m 'greeting.txt: New'
```

が実行されます。エコーエリアには、

```
Checking in /tmp/vc-first/greeting.txt...➥
done
```

と表示されます。Check inとは、変更内容をリポジトリに登録したという意味です。そのあとは、引き続きgreeting.txtを編集できます。以後、C-x v vはコミットの働きをします。

キリがいいところまで編集してからC-x v vを押せば、編集内容がリポジトリに登録されます。今度は、greeting.txtの内容を、

```
Hello world!
```

に書き換えてから、C-x v vを押します。コミットメッセージには、

```
greeting.txt: Add "world"
```

とでも書いてから、C-c C-cです。このとき内部では、

ーgit add greeting.txt
ーgit commit -m 'greeting.txt: Add "World"'

が実行されます。

このように、C-x v vは空気を読んで次の働きをしてくれます。

・リポジトリがないときはリポジトリを作成してからファイルの登録
・ファイルが未登録のときは登録
・ファイルが登録されているときはコミット

新規ファイルではC-x v vを2回実行することを覚えておいてください。vc経由でGitを使い始めるには、何も考えずにC-x v vを叩けばいいのです。決して難しいことはありません。

コミットメッセージには、あとで何を変更したのかを思い出せるように書いておくべきです。コミットメッセージは、子供やペットの成長記録みたいなものです。Gitはまさにタイムマシンと言えるもので、過去のバージョンに変更を加えて、歴史を変えることすら可能です。

ステージングエリアは隠蔽される

Gitでのコミットは、従来のバージョン管理システムとは異なります。従来のバージョン管理システムでは、最後のコミットから変更された部分がすべてコミット対象となりますが、Gitではステージングエリアに上がった変更個所がコミット対象となります。そのため「変更したからコミットしろ」と言われても、Git側からすると「ステージングエリアに何もないからコミットできません」ということになります。

ステージングエリアとはコミット対象となる変更個所を登録する場所のことです。この仲介者はGit初学者を悩ましますが、コミットに柔軟性を与えてくれます。たとえば、ファイルA、Bを変更した場合、従来だとAとBの変更が一緒にコミットされますが、GitではAとBを別々のコミットにできるのです。それどころか、同じファイル内に複数の変更がある場合でも、コミットを分割できます。

vcは従来のバージョン管理システムに合わせて作られているため、ステージングエリアは隠蔽されます。これではGitの魅力を押し殺しているようですが、vcはシンプル性を選択しました。C-x v vでコミットする場合、カレントファイルのすべての変更をステージングエリアに上げてからコミットします。C-x v vは変更が1ファイルに閉じている場合にとても便利です。

過去の変更を参照

ファイル変更履歴を見る

バージョン管理システムを導入すれば、いつでも過去にアクセスできます。

現時点で、greeting.txtは2回コミットされています。「Hello!」と「Hello world!」でしたね。それでは、変更履歴を見てみましょう。C-x v lを押してください。すると、*vc-change-log*バッファが出てきて、**リスト1**のように過去に書いたコミットメッセージが表示されます。内部では、

ーgit log -- greeting.txt

が実行されます。また、*vc-change-log*内で

▼リスト1　*vc-change-log*に変更履歴が表示される

```
commit fd06fa1e7933933ac3fb8648a69023816f4aba79
Author: rubikitch <rubikitch@ruby-lang.org>
Date:   Wed Jan 6 10:41:24 2016 +0900

    greeting.txt: Add "world"

commit f35c3fad9f106635b9e7209daca634e808b4d7dd
Author: rubikitch <rubikitch@ruby-lang.org>
Date:   Wed Jan 6 10:41:11 2016 +0900

    greeting.txt: New

Process git finished
[Show 2X entries]    [Show unlimited entries]
```

dを押せば、カーソル位置のバージョンと直前のバージョンの差分が*vc-diff*に表示されます(**リスト2**)。diffはunified diff形式で表示され、「+」の行が加えられた行、「-」の行が取り除かれた行を表します。*vc-diff*内で Enter を押せば、その部分に移動できます。

直前のコミットからの差分を見る

直前のコミットとの差分を見るにはC-x v =を使います。ファイルの修正個所のみを見る場合に使います。greeting.txtに「Hi!」を付け加えて保存してください。

```
Hello world!
Hi!
```

その後、C-x v =を押すと、**リスト3**のような出力になります。筆者は、コミットする直前によく使っています。もちろん Enter で該当行に移動できます。次に移る前に、C-x v vでコミットしてください。

ファイルの各行がいつ更新されたのかを見る

vcのおもしろい機能として、各行がいつ変更されたのかを色分けして見ることができます。C-x v gを押してください。すると、**図1**のように表示されます。

左側にその行が更新されたコミットと変更者と日時が表示され、右側に色付けされた行が表示されます。白黒の誌面ではわかりづらいですが、1行目は青、2行目は赤になっています。暗い色は変更時期が古い行、明るい色が新しい行です。そこで、pやnを押すと、前後のバージョンも手軽に見ることができます。

おわりに

gittyパッケージを導入すればgit stashという便利な機能がvcから使えるようになります。magitパッケージはGitのEmacsインターフェースとして大人気です。git-gutter/git-gutter+パッケージは未コミットの行を教えてくれます。ほかにもたくさんのGit関係のパッケージが存在するので、気になる方はM-x list-packagesから探してインストールしてみましょう。

筆者のサイト「日刊Emacs」は日本語版Emacs辞典を目指すべく日々更新しています。手元でgrep検索できるよう全文をGitHubに置いています。また、Emacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでもお答えします。「こんなパッケージ知らない?」「挙動がおかしいからなんとかしてよ!」はもちろんのこと、自作elispプログラムの添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

▼リスト2　*vc-change-log*内のカーソル位置のバージョンと、直前のバージョンの差分

```
diff --git a/greeting.txt b/greeting.txt
index 10ddd6d..cd08755 100644
--- a/greeting.txt
+++ b/greeting.txt
@@ -1 +1 @@
-Hello!
+Hello world!
```

▼リスト3　直前のコミットとの差分

```
diff --git a/greeting.txt b/greeting.txt
index cd08755..f7e4ed5 100644
--- a/greeting.txt
+++ b/greeting.txt
@@ -1 +1,2 @@
 Hello world!
+Hi!
```

▼図1　各行がいつ変更されたのかを見る

書いて覚える Swift 入門

第12回 Protocol Oriented Programming

Writer 小飼 弾（こがい だん）　twitter @dankogai

すでに始まったPOP（Protocol Oriented Programming）の時代

今回はいよいよ懸案のProtocol Oriented Programmingについて解説します。

Swift Standard Library[注1]を眺めていると、際立った特徴があります。

Classが少ないのです。たったの5つ。しかも1つは継承でつながっているので実質3つ（**図1**）。

それに対してStructとEnumとProtocolはどっさりあります。これは何を意味するのか？

SwiftにおいてClassというのはあくまでもObjective-Cの遺産を活用するためのものであって、Swift的なProgramとはStructやEnumをProtocolで「つなげて」活用することであるという「中の人の心の叫び」であると弾言しておきます。

ClassとStructやEnumの違い

それを理解するためには、ClassとStructやEnumの違いを理解しておく必要があります。細かい違いは数あれど、「若者のClass離れ」の理由を理解するために必要なのは、ただ1つです。

Classは継承できるが、StructやEnumは継承できない。つまり、**リスト1**の事例のようなコードは書けないのです。

なぜ継承できないのか？　参照型であるClassと異なり、StructやEnumには実体があるからです。上記の例ではインスタンス変数は実際3つあり、それが参照でつながっています。xにアクセスするには親クラスの親クラスまで参照を

▼図1　Swift Standard Library（https://developer.apple.com/library/ios/documentation/General/Reference/SwiftStandardLibraryReference/）

Classes

AnyGenerator	An abstract GeneratorType base class over Element elements.
NonObjectiveCBase	A common base class for classes that need to be non-@objc, recognizably in the type system.
ManagedProtoBuffer	A base class of ManagedBuffer<Value,Element>, used during instance creation.
ManagedBuffer	A class whose instances contain a property of type Value and raw storage for an array of Element, whose size is determined at instance creation.
VaListBuilder	An object that can manage the lifetime of storage backing a CVaListPointer.

Protocols

AbsoluteValuable	A type that supports an "absolute value" function.
AnyCollectionType	A protocol for AnyForwardCollection<Element>, AnyBidirectionalCollection<Element>, and AnyRandomAccessCollection<Element>.
AnyObject	The protocol to which all classes implicitly conform.
ArrayLiteralConvertible	Conforming types can be initialized with array literals.
BidirectionalIndexType	An index that can step backwards via application of its predecessor() method.
BitwiseOperationsType	A set type with O(1) standard bitwise operators.
BooleanLiteralConvertible	Conforming types can be initialized with the boolean literals true and false.
BooleanType	A type that represents a boolean value.
CVarArgType	Instances of conforming types can be encoded, and appropriately passed, as elements of a C va_list.
CollectionType	A multi-pass sequence with addressable positions.
Comparable	Instances of conforming types can be compared using relational operators, which define a strict total order.
CustomDebugStringConvertible	A type with a customized textual representation suitable for debugging purposes.
CustomLeafReflectable	A type that explicitly supplies its own Mirror but whose descendant classes are not represented in the Mirror unless they also override customMirror().
CustomPlaygroundQuickLookable	A type that explicitly supplies its own PlaygroundQuickLook.
CustomReflectable	A type that explicitly supplies its own Mirror.
CustomStringConvertible	A type with a customized textual representation.
DictionaryLiteralConvertible	Conforming types can be initialized with dictionary literals.
Equatable	Instances of conforming types can be compared for value equality using operators == and !=.
ErrorType	
ExtendedGraphemeClusterLiteralConvertible	Conforming types can be initialized with string literals containing a single Unicode extended grapheme cluster.
FloatLiteralConvertible	Conforming types can be initialized with floating point literals.
FloatingPointType	A set of common requirements for Swift's floating point types.
ForwardIndexType	Represents a discrete value in a series, where a value's successor, if any, is reachable by applying the value's successor() method.
GeneratorType	Encapsulates iteration state and interface for iteration over a sequence.
Hashable	Instances of conforming types provide an integer hashValue and can be used as Dictionar

Feedback

注1） Protocol-Oriented Programming in Swift（https://developer.apple.com/videos/play/wwdc2015-408/）

たぐらなければなりません。sizeofValue(cv3)はポインターのサイズである8。Classのインスタンスであれば、必ずそうなります。

これに対し、StructやEnumは、必要なインスタンスはすべて自前で持っています。

```
struct StructV3 {
    var x = 0.0, y = 0.0, z = 0.0
    init (x:Double, y:Double, z:Double) {
        self.x = x
        self.y = y
        self.z = z
    }
}
let sv3 = StructV3(x:1, y:2, z:3)
```

ここでsizeofValue(sv3)は、Doubleのきっかり3倍である24。確かに実体を持っています。StructやEnumは、何も共有していないのです。Shared Nothingというは、並列プログラミングを格段に容易にします。共有は競合を産み、その競合をどう調停するかで我々はかなり苦労してきました。なら共有しなければいい。簡単ですね？

「えー、ちょっと待って！　それってDRY(Don't repeat yourself)に反しない？」と反応した読者は鋭い。そうなのです。共有をやめるということは、型の数だけ実装がいるということでもあるのです。共通項を取り出してまとめるというのは、プログラミングの作法で最重要なものの1つなのに。

でも、それはソースのレベルの話であって、必要なマシンコードはコンパイラーのほうで生成してくればいいじゃん？——我々がしたいのは、ソースの共有であって、実体ではないのですから。

それを可能にするのが、Protocolなのです。

実践例 [swift-complex]

論より証拠。実例を見てみましょう。[swift-complex]というgithub projectがあります。Swiftの演習用に筆者がずっと書いてきたものですが、今回の記事のためにごっそり書き直しました(**図2**)。

▼リスト1　Class継承のサンプル

```
class ClassV1 {
    var x = 0.0
    init (x:Double) {
        self.x = x
    }
}
class ClassV2 : ClassV1 {
    var y = 0.0
    init (x:Double, y:Double) {
        super.init(x:x)
        self.y = y
    }
}
class ClassV3 : ClassV2 {
    var z = 0.0
    init (x:Double, y:Double, z:Double) {
        super.init(x:x, y:y)
        self.z = z
    }
}
var cv3 = ClassV3(x:1, y:2, z:3)
```

要は複素数を使うためのライブラリです。使い心地はRubyであれば、

```
require 'cmath'
include cmath
```

Pythonであれば

```
from cmath import *
```

したときにとてもよく似ています。余談ではありますが、去年のクリスマスにリリースされたPerl 6では、複素数サポートは組み込みです。

そのまま遊べるように、Playgroundも用意してあります。みんな大好きマルデンブロー集合もこのとおり(**図3**)。

で、自分で言うのもなんですが、割によく書けていると思います。よく書けているというのは、

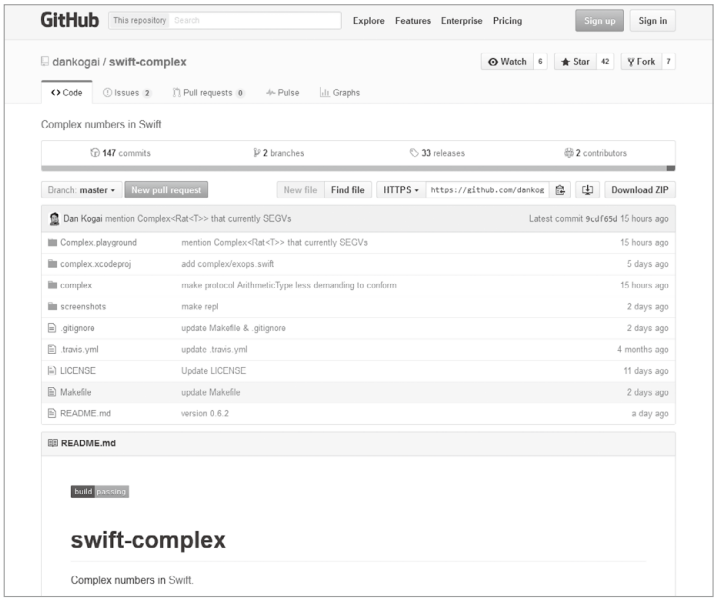
Swiftの特徴を活かしていること

・演算子関数による直感的な操作
→Playground
・クロスプラットフォーム
→OS XやiOSだけでなく、Linuxでも動く

「実数」の実装が入れ替え可能であること

・現時点でDoubleだけでなく、FloatやIntもサポート
・任意制度の数値ライブラリを別途用意すれば、それを使うことも可能

▼図2　swift-complex（https://github.com/dankogai/swift-complex）



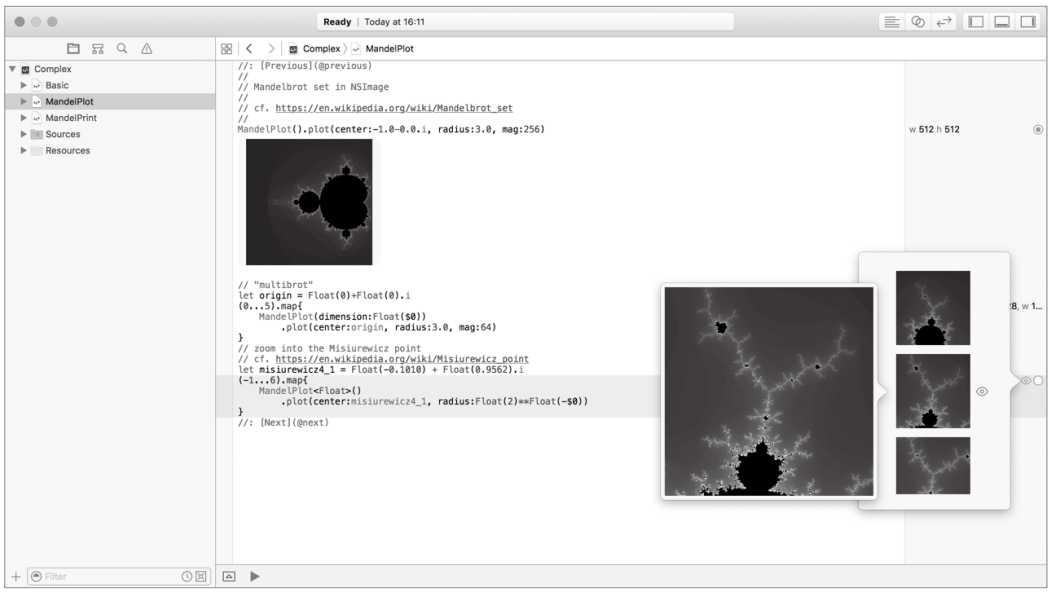
では実際に見てみましょう。500行ちょっとしかないので全部掲載したいところですが、紙幅が足りないので要点だけ（**リスト2**）。

まず、Protocolで複素数の要素として最低限満たしておくべき要件を列挙しておきます。

・Swiftの組込み型からの初期化をサポートしていること
・基本的な四則演算をサポートしていること

というのをSwift語で書き

▼図3　みんな大好きマンデルブロー集合



下しただけです。

で、Intはすでにこれらを満たしているので、

```
extension Int : ArithmeticType {}
```

とすでにArithmeticTypeに準拠(conform)していますよ、と一言で済みます。

ここまではSwift 1の時代からあったのですが、Swift 2の時代ですごいのは、ここです(**リスト3**)。

ご覧のとおり、42.iとかと書くと(0+42.i)になるのは、ここでやっています。わざわざ他の型で実装する必要はないんです。

Protocolといういうのは、あくまで規約であって、その規約をどう満たすかはそのProtocolを準拠する型(types)に任されてきたのですが、Swift 2になって、Protocol Extensionで実装まで一緒にできるようになったのです。つまり、準拠する型が10あれば10、100あれば100、一挙にメソッドやプロパティを追加することが可能になったのです。これはすごい。

実際Swift 2では、Arrayだけではなく Sequenceに準拠する型であればすべて.mapや.reduceが使えるようになったのですが、まさにProtocol Extensionの賜物と言えるでしょう。

ではいよいよComplexを見てみましょう(**リスト4**)。

リスト4は見てのとおり、ArithmeticTypeに準拠したTによる総称型です。1+1.iは

▼リスト2　複素数実装のサンプル

```
public protocol ArithmeticType:
AbsoluteValuable, Equatable, ↵
Comparable, Hashable {
    // Initializers (predefined)
    init(_: Int)
  //// [中略]
  init(_: Double)
    init(_: Float)
    init(_: Self)
    // CGFloat if !os(Linux)
    #if !os(Linux)
    init(_: CGFloat)
    #endif
    // Operators (predefined)
    prefix func + (_: Self)->Self
    prefix func - (_: Self)->Self
    func + (_: Self, _: Self)->Self
    func - (_: Self, _: Self)->Self
    func * (_: Self, _: Self)->Self
    func / (_: Self, _: Self)->Self
    func += (inout _: Self, _: Self)
    func -= (inout _: Self, _: Self)
    func *= (inout _: Self, _: Self)
    func /= (inout _: Self, _: Self)
}
```

▼リスト3　ArithmeticTypeの実装サンプル

```
public extension ArithmeticType {
    /// self * 1.0i
    public var i:Complex<Self> ↵
{ return Complex(Self(0), self) }
    /// abs(z)
    public static func abs(x:Self)->Self { ↵
return Swift.abs(x) }
    /// failable initializer to conver the type
    /// - parameter x: `U:ArithmeticType` ↵
where U might not be T
    /// - returns: Self(x)
    public init?<U:ArithmeticType>(_ x:U) {
        switch x {
        case let s as Self:     self.init(s)
        case let d as Double:   self.init(d)
        case let f as Float:    self.init(f)
        case let i as Int:      self.init(i)
        default:
            return nil
        }
    }
}
```

▼リスト4　Complexの実装

```
public struct Complex<T:ArithmeticType> : Equatable, CustomStringConvertible, Hashable {
    public typealias Element = T
    public var (re, im): (T, T)
  //// [中略]
}
```

Complex<Int>、1.0+1.0.i はComplex<Double>になるわけです。

ところで賢明な読者は、この時点で絶対値.absや偏角.argがないことに気づかれるかもしれません。これらは複素数自体がComplex<Int>、つまりガウス整数であっても整数におさまるとは限らないからです。

うまいこと、整数の場合は整数を返さないメソッドを持たせず、しかし「実数」の場合にはこれを追加するということができるのでしょうか？できます。そう。Swiftならね。

まず、ArithmeticTypeの要件をすべて満たす上位互換Protocolを1つ追加します(**リスト5**)。

そしてこれをProtocol Extensionで拡張します(**リスト6**)。

要するに、三角関数や指数関数などをLinuxであればGlibc、そうでなければFoundationからごっそり持ってくるわけです。ちなみにProtocol Extensionと型のExtensionで同名の識別子がある場合、型のほうが優先して使われます。実際[swift-complex]でも、Floatに関してはいったんDoubleに変換してFloatに戻すのではなくFloatのままで計算するためにcosfなど末尾にfがついた関数を使いたかったので、extension Floatで上書きしています。

そうしたうえで、**リスト7**です。

つまり複素数の要素がRealTypeに準拠してある場合にのみ、.absや.argを追加するということがSwift 2で可能になったのです。

あとは、関数や演算子を粛々と定義していけばいいだけです。たとえば除算/はこんな感じ。

```
public func / <T>(lhs:Complex<T>,
rhs:Complex<T>) -> Complex<T> {
    return (lhs * rhs.conj) / rhs.norm
}
```

共役.conjとノルム.normは、Complex<T>であれば必ず持っているので、複素数の掛け算と複素数と実数の掛け算でこのように定義できるわけです。実際のソースをGitHubでご覧いただくと、ほとんどすべての演算子がこのような1行定義になっています。

次に"cmath"な関数を見てみましょう。

▼リスト5　上位互換Protocolの追加

```
public protocol RealType : ArithmeticType, FloatingPointType {
    static var EPSILON:Self { get } // for =~
}
```

▼リスト6　Protocol Extensionによる拡張

```
extension RealType {
    /// Default type to store RealType
    public typealias Real = Double
    //typealias PKG = Foundation
    // math functions - needs extension for each struct
    #if os(Linux)
    public static func cos(x:Self)->    Self { return Self(Glibc.cos(Real(x)!))! }
  //// [中略]
    #else
    public static func cos(x:Self)->    Self { return Self(Foundation.cos(Real(x)!))! }
  //// [中略]
    #endif
}
```

```
public func exp<T:RealType>(z:Complex<T>)
-> Complex<T> {
    let r = T.exp(z.re)
    let a = z.im
    return Complex(r * T.cos(a), r *
T.sin(a))
}
```

「博士の愛した数式」(小川洋子)でもお馴染みの、e ** (x+y.i) = e**r * (cos(y) + i*sin(y))そのままですね。ただしcosでなくてT.cosと書いています。RealTypeのProtocolでpublic static func cos(x:Self)->Selfとなっているものを指定しています。なぜメソッドではなく型関数(static method)かというと、既存の型をなるべく上書きしたくなかったから。かつてはメソッドとして追加していたのですが、その方法だとXcodeなどで.cosまで補完されてしまってちょっと驚きなのです。Rubyistsなどからするとちょっと残念かもしれませんが。

▼リスト7　Swift 2ならではの実装例

```
extension Complex where T:RealType {
    public init(abs:T, arg:T) {
        self.re = abs * T.cos(arg)
        self.im = abs * T.sin(arg)
    }
    /// absolute value of self in T:RealType
    public var abs:T {
        get { return T.hypot(re, im) }
        set(r){ let f = r / abs; re *= f; im *= f }
    }
    /// argument of self in T:RealType
    public var arg:T  {
        get { return T.atan2(im, re) }
        set(t){ let m = abs; re = m * T.cos(t); im = m * T.sin(t) }
    }
    /// projection of self in Complex
    public var proj:Complex {
        if re.isFinite && im.isFinite {
            return self
        } else {
            return Complex(
                T(1)/T(0), im.isSignMinus ? -T(0) : T(0)
            )
        }
    }
}
```

Todo

というわけで弾言します。総称関数とプロトコルを制するものが、Swiftを制するのだ、と。Swift 2のprotocol extensionで、その可能性はさらに高まりました。

とはいえ、Swift 2でもまだ至らないことも多々あります。たとえばプロトコルに準拠するためのメソッドや関数を書いている最中には、“type Foo does not conform to protocol Bar”というエラーメッセージでXcodeが真っ赤になったりするのですが、具体的にどんなメソッドやプロパティが足りないかを一挙に調べてくれるとうれしいのですが。

あと、Linuxでもimport Foundationできるのに、これがOSX/iOSのそれと全然違うってのも悩ましい。なるべく#ifを書かずに済ませたいのに……。

それにしても、これほど書いていて楽しい言語というのはそうありません。[IBM Swift Sandbox[注2]]のおかげでブラウザからも試せるようになったSwift、皆さんもぜひ遊んでみてください。SD

注2) IBM Swift Sandbox(http://swiftlang.ng.bluemix.net/)

Erlangで学ぶ並行プログラミング

Author 力武健次技術士事務所 所長 力武 健次(りきたけ けんじ) http://rikitake.jp/

最終回・第12回 ErlangのWebサーバとライブラリ

この連載ではプログラミング言語Erlangとその並行プログラミングについて紹介していきます。最終回の今回はErlangによるWebサーバとライブラリの使い方について紹介します。

OTP最新版の状況

Erlang/OTPは昨年12月16日に18.2[1]がリリースされ、SSL/TLSやSSH、そして高速化のためのHigh Performance Erlang(HiPE)の更新などが行われました。その後昨年12月18日には18.2.1[2]でWindowsのパス名に関する問題と、FreeBSDでのHiPE実行の問題の解決注1が行われました。本稿入稿時のErlang/OTPの最新版は1月11日のパッチリリースである18.2.2です[3]注2。

OTP最新版の更新状況は、erlang-questionsメーリングリスト[4]、そしてFreeBSDのPortsにあるlang/erlang[5]の更新などで知ることができます。

ErlangとWebサービス

Webの基本プロトコルであるHTTPは、並行して大量のコネクションやストリームを扱いオブジェクトを迅速に転送することを想定しています注3。これはErlangで想定している「多数の軽量プロセスを大量に作り、並行して仕事をさせる」という並行処理のやり方に適しています。Webサーバの場合、Erlang/OTPのプロセスにHTTPによるデータ転送の仕事を対応づければ、効率よく大量のコネクションやストリームをさばくことが期待できます注4。

Erlang/OTPではgen_tcpモジュール[6]でTCPを直接扱えるしくみがあり、これの上にいろいろなネットワークサービスを構築できます。OTPではinetsアプリケーション[7]HTTP/1.1のクライアントとしてhttpcモジュール[8]、サーバとしてhttpdモジュール[9]が提供されています。`httpc:request/1`を使った簡単なクライアントの例を**図1**に示します[10]。

もっとも、実際の運用ではinetsアプリケーションが提供する機能だけでは機能が不十分なことが多いため、現在多くのWebアプリケーションやフレームワークがErlang/OTP上で開発され使われています。今回はこれらの中から筆者が普段使っているWebサーバ／アプリケー

注1) 本稿筆者によるFreeBSDのOSシグナルの取り扱いの改善提案が取り込まれました(https://github.com/erlang/otp/pull/926)。

注2) 最新版はGitHubリポジトリを使いタグを指定することでビルドできます。詳細は「kerlでGitHub版のErlangをインストールする」(http://qiita.com/jj1bdx/items/4f7d7b5a53fcec32ab8d)を参照してください。

注3) HTTP/1.1では複数のTCPコネクションを張って並行してデータ転送を行っていたのが、HTTP/2では単独のTCPコネクション上に複数のストリームを使って並行してデータ転送を行うという違いはありますが、高い並行度が要求されることは変わりありません。詳細は本誌2015年11月号の特集「すいすいわかるHTTP/2」を参照してください。

注4) このようなアイデアは2001年10月ごろにすでにあったようです(http://erlang.org/pipermail/erlang-questions/2001-October/003944.html)。

▼図1　httpcモジュールによるWebページ内容取得の例

```
Eshell V7.2.1  (abort with ^G)  inetsアプリケーションを起動する
1> inets:start().
ok
httpc:request/1 を使って筆者のホームページにアクセスする。この関数では同期受信のためオブジェクトの受信完了まで待つ。ヘッダーはHeadersという変数に入る。変数Bodyは本文のUTF-8の文字列を示すリストになっている

2> {ok, {{Ver, Code, Reason}, Headers, Body}} = 
httpc:request("http://www.k2r.org/kenji/").
{ok,{{"HTTP/1.1",200,"OK"},
     [{"cache-control","public, max-age=5"},
      {"date","Tue, 19 Jan 2016 12:59:48 GMT"},
      {"accept-ranges","none"},
      {"server","GSE"},
      {"vary","Accept-Encoding"},
      {"content-length","19910"},
      {"content-type","text/html; charset=utf-8"},
      {"expires","Tue, 19 Jan 2016 12:59:53 GMT"},
      {"last-modified","Sun, 03 Jan 2016 08:00:45 GMT"},
      {"x-frame-options","SAMEORIGIN"},
      {"x-robots-tag","noarchive"},
      {"x-content-type-options","nosniff"},
      {"x-xss-protection","1; mode=block"}],
     [60,33,68,79,67,84,89,80,69,32,104,116,109,108,32,80,85,66,
      76,73,67,32,34,45|...]}}
io:format/2でBodyの内容を表示する。実際にはかなり長いので途中を省略している
3> io:format("~s~n",[Body]).
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" itemscope=""
itemtype="http://schema.org/WebPage">
<head>
<meta http-equiv="X-UA-Compatible" content="chrome=1" />
(中略)
</body>
</html>
inetsアプリケーションを止める。INFO REPORTの部分はアプリケーションsaslによるログ
4> inets:stop().
ok
5>
=INFO REPORT==== 19-Jan-2016::22:01:08 ===
    application: inets
    exited: stopped
    type: temporary
```

ションのYaws[11]と、軽量Webアプリケーションを構築するためのライブラリであるCowboy[12]を紹介します。

Erlangで書かれたHTTPサーバYaws

Yaws[注5]は、Apache Web ServerやnginX同様、静的動的双方のコンテンツを配信でき、単独でも使え、かつ他のErlangアプリケーションとも組み合わせて使えるHTTPサーバです。

2002年[注6]からErlangの初期からの開発者ClaesWikströmが開発を始め、現在ではSteve Vinoskiほか多くのErlangエンジニアの支持を受けて広く使われています。筆者も自宅のサーバをYawsに置き換えて長期にわたり動かしています。

Yawsの導入にはOS Xであればhomebrewのyaws、FreeBSDならPortsのwww/yaws、Ubuntuであればapt-get install yawsとすることでインストール[14]ができます。

最も簡単な設定ファイルを**リスト1**に示します。この設定ファイルをカレントディレクトリに置いて次のコマンドを起動すると、コンソールからErlangの仮想マシンBEAMを操作できる状態でYawsが立ち上がります[注7]。

```
$ yaws --conf ./yaws.conf
```

この状態でドキュメントルート(**リスト1**では/var/tmp/yaws/localhostの下)にindex.htmlを置けば、http://localhost:8000/の中身としてアクセスできます。これだけの設定で通常の静的コンテンツサーバとして、ドキュメントルートにファイルを展開して使えます[注8]。

注5) Yet Another Web Serverの略で、筆者は「ヨーズ」と発音しています。

注6) Gitに変換されたSubversionのログによる情報です[13]。

注7) デーモンモード(--daemonオプション)を付けることで、デーモンとしても起動できます。詳細はYawsのマニュアルを参照してください(http://yaws.hyber.org/doc.yaws)。

注8) YawsはBEAM上で動作するため、BEAMに対するOSプロセスのアクセス制限が適用されます。不正アクセスなどのセキュリティ事故を防ぐには、BEAMを実行する環境に対して、一般的なOSの仮想化、あるいはFreeBSDのJailやLinuxのコンテナ、最近ならDockerなどの実行環境隔離のしくみを使うことを強く推奨します。本記事では実験のためアクセス制限はとくにOSレベルでは行わないものとします。

▼リスト1　localhostにサーバを設定するためのYawsの設定ファイルの例

```
# http://localhost:8000/ にてIPv4/v6両方でサーバを立ち上げます

<server localhost>
        port = 8000
        listen = 127.0.0.1
        listen = ::1
        # 以下はドキュメントルートのディレクトリです
        docroot = /var/tmp/yaws/localhost
</server>
```

Yawsによる動的コンテンツ生成

Yawsでは.yawsという拡張子のファイルにHTMLとErlangコードが混ざったものを書くことで、Erlangコードを使ったコンテンツの動的生成ができます。具体的な方法としては、静的コンテンツをHTMLで書き、「<erl>　</erl>」というブロックの中にYawsが理解するErlangコードを書きます。各コードブロックにはout/1という関数が必要です。out/1からはHTMLを文字列として出力したり、ErlangのデータからHTMLを生成するためのEHTMLという形式での出力ができます。

リスト2に動的コンテンツの生成のための設定例を示します。最初にHTMLに必要なタグを列挙した上で、1番目のErlangコードブロックでは、Yawsから与えられたクライアントのIPアドレスとポートの情報を出力しています。2番目のコードブロックでは、Yawsの動いているBEAMノード中のプロセスID(Pid)を列挙し、各Pidに対応した登録名、メモリ消費量、リダクション(reduction)[注9]の回数をテーブルにして返します。

リスト2の中身をドキュメントルートにindex.yawsというファイルとして置いてアクセスした結果を**図2**に示します。**図2**の元となったHTMLファイルを読むと、YawsでのEHTML記法がどのようにHTMLに変換されているかがわかります。

Erlang自身をYawsの拡張言語として利用できることで、Yawsを柔軟性の高いWebサーバとして使うことができます。Yawsにはこの他にもCGIやWebsocketなどへの対応や、Yaws自身を他のErlangアプリケーションに組み込む多くの機能が満載されています。Yawsをおもに取り上げたErlangによるWebアプリケーションの制作については、専門の参考書[15]が出ており、YawsのWebサイト[11]と併せて一読の価値はあるかと思います。

Cowboyによる Webアプリケーション制作

Cowboy[16]は、Yawsよりもさらに細かい操作を集めたWebアプリケーション制作用のライブラリ集です。Yawsとは違い機能を厳選したアプリケーションをリリース(本連載第5回を参照)としてまとめることができます。作者のLoïc Hoguinは、GNU Makeによる汎用ビルドツールerlang.mk[17]、そしてTCPのソケット受信ライブラリRanch[18]、Cowboy用のWebプロトコルライブラリCowlib[19]を公開していて、Webアプリケーション制作用ツールセットの定番の1つとなっています。

Cowboyでのアプリケーション制作手順の概要は次のとおりです。詳細はユーザガイド[20]に書かれています。

①プロジェクト名と同じ名前の開発用ディレクトリを作り、そこにerlang.mkの最新版[17]をコピーして、次のコマンドを実行する

```
$ make -f erlang.mk bootstrap bootstrap-rel
```

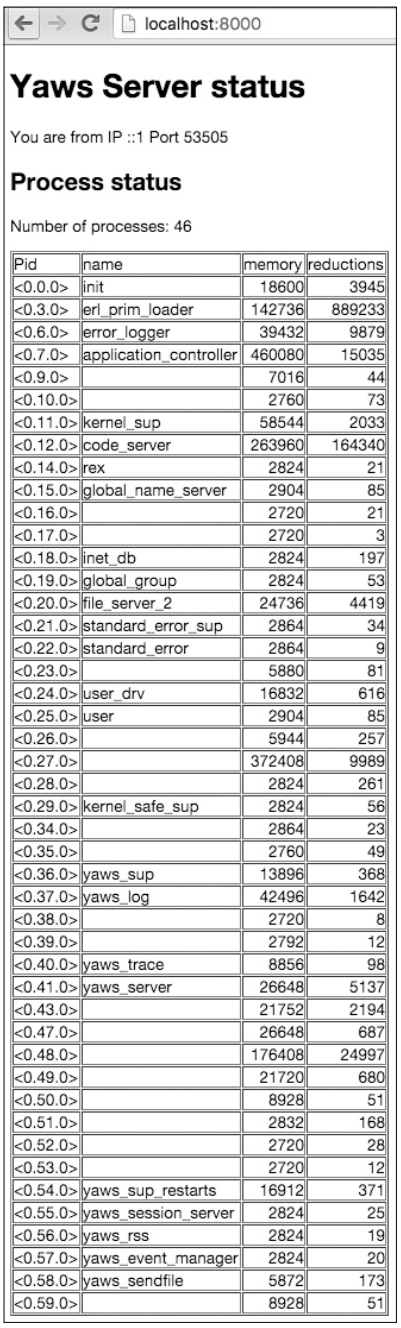
②プロジェクトのMakefileができるので、“include erlang.mk”の前に“DEPS = cowboy”を追加する

注9) BEAMの各プロセスはリダクションのカウンタを持ち、関数実行1回ごとに1つずつ増えていきます。このカウンタの値がスケジューリングに影響します(http://erlang.org/doc/man/erlang.html#bump_reductions-1 を参照)。

▼リスト2　動的コンテンツ生成のための.yawsファイルの例

```
<!-- yawsファイルではHTMLの中にErlangのコードを入れることで動的コンテンツ生成ができます -->
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Yaws Server status</title>
<style> .num { text-align: right; } </style>
</head>
<body>
<h1>Yaws Server status</h1>
<erl><!-- erlタグで囲まれた中はErlangのコードです -->
 out/1という関数の返り値を解釈してHTMLを出力する。ここではクライアントのIPアドレスとポート番号を出力
out(A) ->
    {Addr, Port} = A#arg.client_ip_port,
 out/1の返り値が出力になる。 {html, String}ではStringをそのままHTML出力として埋め込む。関数f/2はio_lib:format/2と等価で、Yawsの設定ファイル中で簡便に書けるようにしたもの
    {html, f("<p>You are from IP ~s Port ~.10B</p>",
           [string:to_lower(inet:ntoa(Addr)), Port])}.
</erl>
<!-- 一度erlタグのブロックを出るとHTMLに戻ります -->
<h2>Process status</h2>
<erl><!-- 複数のErlangのコードを単独のyawsファイルの中に置けます -->
 複数の関数を定義できる
pinfo(P, Item) ->
    element(2, process_info(P, Item)).
 ここではYawsを実行しているBEAMノードのプロセスの状況をテーブルにして出力する
out(_) ->
    PL = processes(),
 {ehtml, List}では、リストListの中身をHTMLのタグやブロックに変換する。{タグ名、タグの属性リスト、タグで囲われた中身の文字列}という3要素のタプルを列挙していく
    {ehtml, [
      {p, [], f("Number of processes: ~.10B",
[length(PL)])},
      {table, [{border, "1"}], [
        {thead, [],
          [{tr, [], [
            {td, [], "Pid"}, {td, [], "name"},
            {td, [], "memory"}, {td, [], "reductions"}
            ]}]},
        {tbody, [],
          [{tr, [],
 この行からはプロセスごとの情報表示をリスト内包表記で書いている
            [{td, [], f("~w", [P])},
             {td, [],
               case process_info(P, registered_name) of
                 {registered_name, Name} -> f("~s",
[Name]);
                 [] -> ""
               end},
             {td, [{class, "num"}], f("~.10B", [pinfo
(P, memory)])},
             {td, [{class, "num"}], f("~.10B", [pinfo
(P, reductions)])}]}
            || P <- PL ]
 ここまでがリスト内包表記
         }]
       }
     ]}.
</erl>
</body>
</html>
```

▼図2　index.yaws の実行結果の表示例



localhost:8000

Yaws Server status

You are from IP ::1 Port 53505

Process status

Number of processes: 46

Pid	name	memory	reductions
<0.0.0>	init	18600	3945
<0.3.0>	erl_prim_loader	142736	889233
<0.6.0>	error_logger	39432	9879
<0.7.0>	application_controller	460080	15035
<0.9.0>		7016	44
<0.10.0>		2760	73
<0.11.0>	kernel_sup	58544	2033
<0.12.0>	code_server	263960	164340
<0.14.0>	rex	2824	21
<0.15.0>	global_name_server	2904	85
<0.16.0>		2720	21
<0.17.0>		2720	3
<0.18.0>	inet_db	2824	197
<0.19.0>	global_group	2824	53
<0.20.0>	file_server_2	24736	4419
<0.21.0>	standard_error_sup	2864	34
<0.22.0>	standard_error	2864	9
<0.23.0>		5880	81
<0.24.0>	user_drv	16832	616
<0.25.0>	user	2904	85
<0.26.0>		5944	257
<0.27.0>		372408	9989
<0.28.0>		2824	261
<0.29.0>	kernel_safe_sup	2824	56
<0.34.0>		2864	23
<0.35.0>		2760	49
<0.36.0>	yaws_sup	13896	368
<0.37.0>	yaws_log	42496	1642
<0.38.0>		2720	8
<0.39.0>		2792	12
<0.40.0>	yaws_trace	8856	98
<0.41.0>	yaws_server	26648	5137
<0.43.0>		21752	2194
<0.47.0>		26648	687
<0.48.0>		176408	24997
<0.49.0>		21720	680
<0.50.0>		8928	51
<0.51.0>		2832	168
<0.52.0>		2720	28
<0.53.0>		2720	12
<0.54.0>	yaws_sup_restarts	16912	371
<0.55.0>	yaws_session_server	2824	25
<0.56.0>	yaws_rss	2824	19
<0.57.0>	yaws_event_manager	2824	20
<0.58.0>	yaws_sendfile	5872	173
<0.59.0>		8928	51

③あとは次のコマンドを実行すればリクエストハンドラのひな形ができる

```
$ make new t=cowboy_http n=リクエストハンドラ名
```

④コーディングを終えたら“make”でビルドして“make run”でできあがったリリースを走らせることができる

Cowboyを使ってWebサーバを書く際は、クライアントからの処理要求を扱うリクエストハンドラを書く必要があります。このリクエストハンドラは専用のビヘイビア(本連載第5回を参照)となっており、外部からの要求に対してどのような処理をするかだけを書けばあとはCowboy側で処理するようになっています。

リスト3では、前述のYawsとほぼ同様の処理を行っています。cowboy_req:peer/1で接続クライアントのIPアドレスとTCPポート番号を取得した後、cowboy_req:reply/3で返答を返しています。サンプルソースの全文はGithubのリポジトリ[21]に公開しています。

Erlang/OTPのI/O出力とiolist/Deeplist

YawsでもCowboyでも使われているErlang/OTPのI/O出力に関するデータ構造として、iolist(またはDeeplist)と呼ばれるものがあります。iolistは複数の階層を持つバイナリや文字列のリストで、リストで表記されている文字列を複数個並べるときや、高速化のためバイナリとして文字列を扱う際に頻繁に発生します。

Erlang/OTPでは、iolistを出力の際平準化(flatten)することで、単一階層のリストと同じように扱うよう工夫しています。平準化とは簡単に言えば、複数階層のための大括弧を取り払い、全要素を左から右に結合する演算です(**図3**、**図4**)。この演算は木構造の探索とリスト要素のコピーが必要で、コストの高いものです。

Erlang/OTPのポート(本連載第11回を参照)出力では、このiolistを直接扱えるようになっており、平準化演算の関数lists:flatten/1を使う必要がありません。YawsのHTML出力やCowboyのcowboy_req:reply/3も同様にiolistを直接扱えます。また、最終的にiolistから出力をまとめるための組込み関数list_to_binary/1やiolist_to_binary/1も用意されています[注10]。

本連載を終えるにあたって

今回はYawsとCowboyを例としたErlang/OTPによるWebサーバのプログラミングについて紹介しました。2015年4月号より12回にわたりErlang/OTPとその関連技術について紹介してきましたが、本連載では他の参考書等ではあまり紹介していない内容を努めて取り上げるようにしました。至らぬところも多かろうと思いますが、みなさんの今後の学習の参考になれば幸いです。

手元の記録を見てみると、筆者がErlang/OTPを勉強し始めたのは2008年の4月とあります。もうすでに8年近く時が経ったことになりますが、その間にErlang/OTPを取り巻く環境は大きく変わりました。ネットでは多くのErlang/OTPの運用実績を読むことができますし、ElixirなどBEAM仮想マシンで動く新たな言語も登場し、より多くの人達がErlang/OTPの関連技術を使うようになっていることを日々感じています。

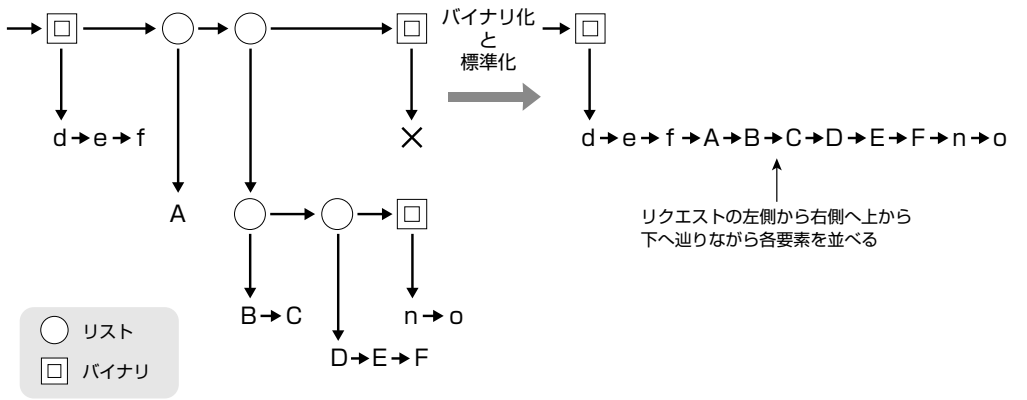
Erlang/OTPの魅力は処理速度、スケジューリング、メモリ管理といった基本機能のギリギリのバランスを取りながら、できる限り並行処理を書きやすくし、分散システムを組みやすくするための工夫を各所に盛り込んでいるところだと、筆者は考えています。他の言語とコミュニティのような華やかさには欠けますが、地道な縁の下の力持ちとして、世界の情報基盤を支

注10) iolistの要素にはアトムを混ぜることはできません。また、iolistは定義上0から255までの整数とそれらを要素にしたリスト、そしてバイナリのみが要素となり得るため、Unicode文字列をそのまま扱うことはできないことに注意が必要です。

▼リスト3　Cowboyのリクエストハンドラの一例 beam_status_handler.erl（Cowboyによるハンドラモジュールからの抜粋）

```
-module(beam_status_handler). gen_server同様の使い方
-behaviour(cowboy_http_handler).
-export([init/3]).
-export([handle/2]).
-export([terminate/3]).
-record(state, {}).
init(_, Req, _Opts) ->  ハンドラの初期化
  {ok, Req, #state{}}.
 ここからは内部の補助関数。Yawsのf/2と同じ定義
f(F, X) -> io_lib:format(F, X).
 バイナリによる静的コンテンツ部（関数3つ）
static1() ->
    << "<!DOCTYPE html>" "<html>" "<head>"
    "<meta charset=\"utf-8\">"
    "<title>Cowboy server status</title>"
    "<style> .num { text-align: right; } </style>"
    "</head>" "<body>" "<h1>Cowboy server status</h1>"
    >>.
static2() ->
    <<"<h2>Process status</h2>">>.
static3() ->
    <<"</body>" "</html>" >>.
 index.yaws同様の記述
pinfo(P, Item) ->
       element(2, process_info(P, Item)).
 プロセスごとのテーブルエントリを作る
pinfo_table() ->
    PL = processes(),
 以下のコードはネストしたリストを返すが、これらは出力の際ネストが平滑化されて、あたかも単一階層のように扱われる（iolistまたはDeeplist）
    [
      f("<p>Number of processes: ~.10B</p>", [length(PL)]),
      "<table border=\"1\">",
      "<thead><tr><td>Pid<td>name<td>memory<td>reductions</tr></thead>",
      "<tbody>",
      [["<tr>",
        f("<td>~w", [P]),
        "<td>",
        case process_info(P, registered_name) of
            {registered_name, Name} -> f("~s", [Name]);
            [] -> ""
        end,
        "<td class=\"num\">",
        f("~.10B", [pinfo(P, memory)]),
        "<td class=\"num\">",
        f("~.10B", [pinfo(P, reductions)])]
        || P <- PL ],
      "</table>"
    ].
 ハンドラ本体。それぞれの処理ごとにReqの中身を変えていく
handle(Req, State=#state{}) ->
 接続してきたクライアントのアドレスを取得する
    {{Addr, Port}, Req2} = cowboy_req:peer(Req),
 リクエストに返答を返す
    {ok, Req3} = cowboy_req:reply(200, [
 このタプルでヘッダのコンテンツを決める
           {<<"content-type">>, <<"text/html">>},
           {<<"cache-control">>, <<"private, max-age=0, no-cache">>}
       ],
 ここからはHTML文書のbody部の内容。これもiolistですが出力では平滑化される
       [static1(),
        f("<p>You are from IP ~s Port ~.10B</p>",
           [string:to_lower(inet:ntoa(Addr)), Port]),
        static2(),
        pinfo_table(),
        static3()
       ],
       Req2),
 gen_server同様に変わったReqの中身を返す
    {ok, Req3, State}.
 サーバ終了時の処理（何もしない）
terminate(_Reason, _Req, _State) ->
    ok.
```

▼図3　iolistの標準化



▼図4　iolist平準化の例

```
Eshell V7.2.1  (abort with ^G)
 バイナリを3つ定義する
1> B1 = <<100, 101, 102>>.
<<"def">>
2> B2 = <<110, 111>>.
<<"no">>
3> B3 = <<120>>.
<<"x">>
 文字列等が混ざった多階層のリスト(iolist)を作る
4> IL = [B1, "A", ["BC", "DEF", B2], B3].
[<<"def">>,"A",["BC","DEF",<<"no">>],<<
"x">>]
 ただ平準化するだけではリストとバイナリは別に扱われる
5> lists:flatten(IL).
[<<"def">>,65,66,67,68,69,70,<<"no">>,<<
"x">>]
 iolist_to_binary/1を使うことで平準化がでる
6> iolist_to_binary(IL).
<<"defABCDEFnox">>
 それぞれをあらためて出力してみると次のようになる
7> io:format("~w~n~w~n~w~n",
     [IL, lists:flatten(IL), iolist_to_
binary(IL)]).
[<<100,101,102>>,[65],[[66,67],[68,69,70],
<<110,111>>],<<120>>]
[<<100,101,102>>,65,66,67,68,69,70,<<110,
111>>,<<120>>]
<<100,101,102,65,66,67,68,69,70,110,111,
120>>
```

えるだけの力のある言語システムをErlang/OTPは提供しています。今後もそうあってほしいと、筆者は願っています。

ソースコードとサポートページ

今回の例に使用したCowboyのコードはGitHubリポジトリにて公開しています[21]。また、本連載の各記事で紹介したソースコードなどは、GitHubのリポジトリに置いています(https://github.com/jj1bdx/sd-erlang-public/)、どうぞご活用ください。SD

参考文献

[1] http://www.erlang.org/news/97
[2] http://www.erlang.org/news/98
[3] http://erlang.org/pipermail/erlang-questions/2016-January/087311.html
[4] http://erlang.org/mailman/listinfo/erlang-questions
[5] http://www.freshports.org/lang/erlang/
[6] http://erlang.org/doc/man/gen_tcp.html
[7] http://erlang.org/doc/apps/inets/users_guide.html
[8] http://erlang.org/doc/man/httpc.html
[9] http://erlang.org/doc/man/httpd.html
[10] http://erlang.org/doc/apps/inets/http_client.html
[11] http://yaws.hyber.org/
[12] https://github.com/ninenines/cowboy/
[13] https://github.com/klacke/yaws
[14] http://yaws.hyber.org/configuration.yaws
[15] Zachary Kessin, "Building Web Applications with Erlang", O'Reilly Media, 2012, ISBN-13: 978-1-449-30996-1.
[16] http://ninenines.eu/
[17] https://github.com/ninenines/erlang.mk
[18] https://github.com/ninenines/ranch
[19] https://github.com/ninenines/cowlib
[20] HEAD版：http://ninenines.eu/docs/en/cowboy/HEAD/guide/
[21] https://github.com/jj1bdx/cowboy_beam_status/

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第25回 ホカン

バレンタインデー前だけでなく、いつでも女性に優しいくつな先生はフェミニストだからですよね?! チョコの数と涙の数は反比例するらしいよ!

to be Continued

シェルを初めて使う人に、最初に教えるのはパスワード変更時にパスワードは表示されないことと、Tabでコマンドを補完できること。補完を利用するとタイプミスも防げるし、長いコマンドやオプションを失念した時に「ああ、タブがあってよかった!」って誰にでもなく感謝します。シェルだけではなくvimやemacsなどのエディタやIDEでも補完があります。長い変数名や関数名とかは補完使えないと編集作業に集中できません。メールソフトによってはメールアドレスを補完することもありますね。同じ「井上」でも別のアドレスが補完されることがあるので、補完に頼りきるのも注意が必要ですね。このページは今年も補完されないようにがんばります。

Sphinxで始めるドキュメント作成術

若山 史郎 Wakayama Shirou ツキノワ㈱

第12回 Sphinxで本を書こう——EPUBで出力する

今回のテーマ

今回のテーマはSphinxで「本を書こう」です。SphinxはHTMLだけでなく、多様なフォーマットを出力でき、EPUB[注1]やPDFなど、電子書籍として読めるフォーマットにも出力できます。

今回は、筆者自身の経験をもとに、実際にSphinxを使って執筆・出力した本[注2]をAmazon Kindleダイレクト・パブリッシング(以下、KDP)[注3]で販売した経験を紹介します。

本という、それなりに分量が多い文章を執筆する場合、流れを概観できたり、構成を組み替えたりということが容易にできることが大事です。Sphinxは、本連載第3回(本誌2015年6月号)で述べられているtoctreeのしくみにより、文章を細かなファイルに分けておいて、簡単に組み替えることができる点で優れています。

また、執筆時はHTMLで確認し、校正を依頼するときはPDFで出力したりと、気軽にフォーマットを使い分けられる点も大事です。なお、筆者が執筆時にPDFではなくHTMLで閲覧しているのは、執筆時と校正時でフォーマットを変えることで新鮮味が生まれ、最初は気がつかなかった間違いに後から気がつく可能性が高くなる、と考えているからです。

さらに、Sphinxで執筆するとすべてがテキストファイルになるため、ソースコード管理ツールと相性が非常に良くなります。また、ソースコード管理ツールには多くの場合、課題管理ツールも付いているため、筆者は誤植の指摘や追加の要望などをソースコード管理ツールの課題管理で受け付けるようにしました。これにより、誰からでも気軽に指摘を受け付けられ、それを修正してすぐにKDPで公開する、という流れを構築できました。このように、すぐに修正を反映できる点は電子書籍の良い点だと思います。

Amazon Kindleダイレクト・パブリッシング(KDP)

Amazon KDPは、Amazonが展開している電子書籍販売プラットフォームであるKindleストアに出版するツールです。実際に出版するまでには、次の情報を入力する必要があります。

①アカウントの作成・銀行口座の設定・税処理
②本の詳細情報の入力
③本のコンテンツのアップロードとプレビュー
④出版権の確認
⑤価格とロイヤリティ情報の入力

今回は誌面も限られていますし、すべてを取り上げるとSphinxという本題から外れますので、これらのうち③に相当する本のコンテンツ作成に関してのみ取り上げます。その他の項目は、KDPのページからスタートガイドなどをご覧ください。

注1) http://idpf.org/epub
注2) 『入門Ansible』 http://www.amazon.co.jp/dp/B00MALTGDY
注3) https://kdp.amazon.co.jp/

EPUB出力

KDPはWord、HTML、PDF、EPUB、Textなどさまざまなフォーマットをサポートしています。しかし、KDPのPDFは日本語の本をサポートしていないので、Sphinxで出力する際には実質的にはEPUBを使います。

SphinxでEPUBを出力するには、sphinx-quickstartで、「epubビルダーを使用するか」という**図1**の質問に「y」を入力しておく必要があります。これにより、conf.pyとMakefileにEPUBの設定が書き加えられます。

EPUBを出力するにはHTMLなどと同じくmake epubを使います。実行すると、「_build/epub/<タイトル>.epub」が出力されます。出力ファイル名を変更するにはconf.pyのepub_basenameを変更します。

EPUBはiPhoneやAndroid、PCやブラウザなどいろいろなツールで閲覧できます。後からKDP上でのプレビューもできますが、この段階で確認すると、執筆と確認を早く繰り返せます。

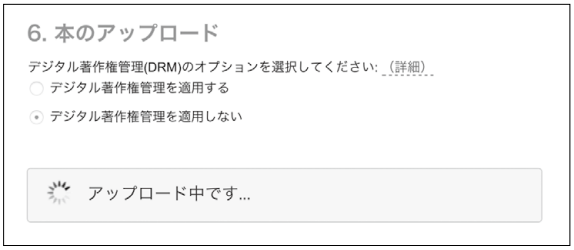
できあがったEPUBファイルを、KDPの画面にてアップロードします(**図2**)。

変換とプレビュー

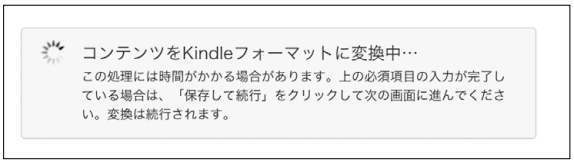
アップロードが完了したら、「Kindleフォーマットに変換中」の表示が出ます(**図3**)。

変換は数十秒で完了しますので、完了したらプレビューしましょう(**図4**)。プレビュー画面

▼図2　アップロード画面



▼図3　変換表示画面



▼図1　sphinx-quickstartにおけるepubビルダーの質問

```
Sphinx can also add configuration for epub output:
> Do you want to use the epub builder (y/n) [n]:  ←ここでyを入力
```

▼図4　プレビュー画面

では、Kindle Fire HDXやKindle Voyageなどの端末種類ごとの見た目、横向き縦向きの見た目、フォントサイズごとの見た目などを確認できます。

ここまでできたら、あとは出版権の確認や価格を決めていくだけでKindleストアから出版できます。実際にストアに並ぶには24時間程度待つ必要がありますが、それさえ過ぎれば、あなたもKindle出版デビューです。

より綺麗に

ここまでで簡単にEPUBファイルを作成し、KDPから出版できることを説明しました。ここからはもう少し手を加えて見栄えが良い本にする方法を紹介します。

CSSを変更

EPUBファイルの中身は、HTMLファイルとEPUB専用の情報を持つファイルを、zipで1つにまとめたものです。SphinxのEPUB出力は、標準で持っているEPUBテーマを適用して、HTMLと同じように書き出したあとzipで1ファイルにまとめる、というしくみで作られています。

標準のEPUBテーマの見た目を変えたい場合は、HTMLと同じくCSSを変更します。CSSをカスタマイズする方法は、本連載の第4回(本誌2015年7月号)で少し触れていますが、ここでもう一度解説します。

❶conf.pyの末尾に次の2行を追加

conf.py
```
def setup(app):
    app.add_stylesheet('custom.css')
```

❷sphinx-quickstartで作成された「_static」ディレクトリに「custom.css」を作成

custom.css
```
table.field-list th, table.field-list td {
    border: 1px solid black !important;
    padding: 0.4em;
}
th { background-color: #eee; }
```

これでEPUBにもCSSが適用されます。

CSSを適用するときには、classの指定が自由にできると便利です。Sphinxで使える基本的なディレクティブやロールには、classというオプションが指定できるようになっています。このオプションを指定すると、たとえば**リスト1**のreStructuredText(以下、reST)が**リスト2**のようなHTMLに展開されます。

また、ロール(本連載第5回、本誌2015年8月号を参照)に対してclassを設定するには、class指定を持つ新しいロールをroleディレクティブを使って定義します(**リスト3**)。ここの括弧の中は継承するロール名で、ここではcodeというロールを継承しています。

こう書くと、出力結果は**リスト4**となり、code要素のclass定義にsomenewが入っているのが

▼リスト1　ディレクティブのclass定義(reST)
```
.. image:: picture.png
   :class: newpict
```

▼リスト2　ディレクティブのclass定義展開後(HTML)
```
<img alt="picture.png" class="newpict" src="picture.png" />
```

▼リスト3　新しいロールを定義し、使用する(reST)
```
ロールディレクティブで新しい somenewroleを定義しておきます。

.. role:: somenewrole(code)  ←somenewroleロールを定義
   :class: somenew           ←somenewroleにclassを設定

その後、:somenewrole:`これは新しい` classです、というように使います。
          ↑somenewroleを適用
```

わかります。

このように、指定したロールやディレクティブに個別にclassを定義できますので、CSSの追加と合わせて自由に装飾を設定できます。注意点として、「:」と「`」の間は空けないことと、拡張したrole定義は定義したファイル内だけの使用に限られる、という点です。すべてのファイルでこの定義を使いたい場合はconf.pyのrst_prologに設定しておく必要があります。

表紙

表紙に画像があると見栄えが良くなります。EPUBで表紙を設定するには次のようにします[注4]。

まず、表紙画像をたとえば「cover.jpg」という名前で「_static」ディレクトリ以下に置き、conf.pyに**リスト5**の設定を加えます。

「_templates」ディレクトリ以下に「epub_cover.html」を置きます。これが表紙のHTMLとなります。この中に**リスト6**のように記述すると、conf.pyで指定した画像ファイル名がHTMLの{{ image }}に挿入され、EPUBの表紙として認識されることになります。

KindleGenでmobiファイルを作成

KDPのプレビューでEPUBからの変換結果を見ることができますが、Amazonから提供されているKindleGenを使うことで、手元のKindleで表示できるmobi形式に変換できます。変換したmobiファイルをUSBやメール経由でKindleに直接送ることで、実機での表示確認ができます。

KindleGenは、KDPのページ[注5]からダウンロードできます。ダウンロードして展開すると、kindlegenというバイナリファイルがありますので、それを**図5**のように実行します。

-oオプションで指定したファイル名(通常「.mobi」という拡張子を付けます)に変換結果が書き出されているはずです。あとはこのファイルをUSBで繋いだKindleに送り込むことで実機での確認ができます。

縦書き

EPUBはCSSを縦書き用に設定することで縦書き表示ができます。先ほど述べたCSSのカスタマイズの方法を使い、**リスト7**のようなCSSを設定します。

注4) KDPではEPUBの表紙画像ではなく、別途、表紙画像をアップロードする方式ですので、ここで説明する方法とは異なります。

注5) https://kdp.amazon.co.jp/help?topicId=A3IWA2TQYMZ5J6

▼リスト4　ロールでのclass定義、展開後(HTML)

```
<p>その後、<code class="code somenew docutils literal"><span class="pre">これは新しい</span></code> class
です、というように使います。</p>
```

▼リスト5　表紙をconf.pyで設定

```
epub_cover = ('_static/cover.jpg', 'epub_cover.html')
```

▼リスト6　EPUB表紙のHTML例(epub_cover.html)

```
<div class="epub-cover">
  <img src="{{ image }}" alt="Cover image" width="612" height="792" />
</div>
```

▼図5　kindlegenコマンドの実行

```
$ kindlegen <できあがったepubファイル名>.epub -o <任意のファイル名>.mobi -locale ja
```

これをEPUBからkindlegenで変換して、Kindle Paperwhiteの実機で表示させると**図6**のようになります。

ルビ表示

縦書きができると、ルビ表示もしたくなります。Sphinxでルビ表示をするためには、拙作「sphinxcontrib-textstyle」を使います。次のようにpipでインストールします。

```
$ pip install sphinxcontrib-textstyle
```

あとはconf.pyで有効化します。

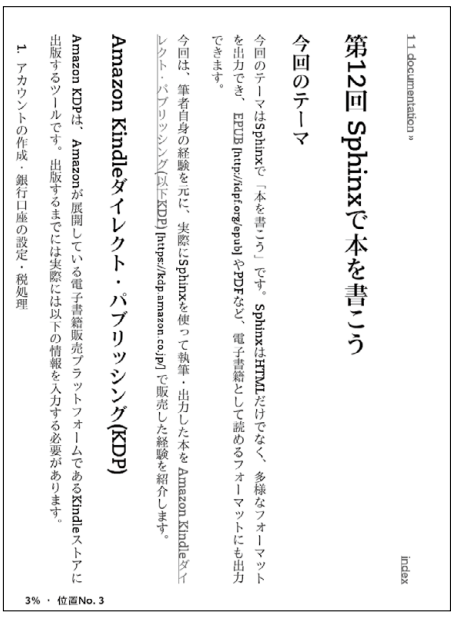
conf.py
```
extensions = ['sphinxcontrib.textstyle']
```

使い方は、:ruby:ロールを指定し、<>内にルビを指定します。

```
これから :ruby:`強敵<とも>` に会いにいく
```

Kindle Paperwhiteの実機上では**図7**のように表示されます。

▼図6　Kindleでの縦書き表示



注意点

SphinxでEPUBを作成するときの注意点としては、フロー型となり、固定レイアウト型の本は作れない、という点です。フロー型とは、端末やフォントの大きさによって1画面に表示される文字数が変わり、ページ数やページ区切りも変わっていく形式のことです。その反対が固定レイアウト型で、1画面に表示される内容が固定されています。固定レイアウト型のほうが本としての体裁は整います。その分、目が悪いためフォントを大きくしたい、というような要望には応えられなくなります。

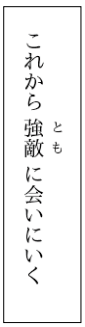
SphinxでのEPUB作成は、HTMLをベースとしているため、フロー型にしかできません。レイアウトやページ区切りにこだわる場合は、Sphinxではできないかもしれません。

次回予告

今回は、Sphinxを利用して電子書籍を出版した話を説明しました。次回はMarkdown形式のドキュメントをSphinxで扱う話を紹介します。

SD

▶図7 Kindle上でのルビ表記



▼リスト7　縦書きにするためのCSS設定例

```
@charset "utf-8";
html {
    writing-mode: vertical-rl;
    -webkit-writing-mode: vertical-rl;
    -epub-writing-mode: vertical-rl;
    line-height: 1.75;
    text-align: justify;
}
```

COLUMN

Sphinx-1.3.5 リリース／公式サイトに日本語訳ドキュメント掲載

Author 清水川 貴之

本連載執筆陣の1人、清水川です。

■Sphinx-1.3.5リリース

2016年1月12日にSphinx-1.3.4、1月24日にSphinx-1.3.5をリリースしました[注A]。これらは1.3系のマイナーバージョンアップで、1.3.3以降合わせて54個の不具合を修正しています[注B]。

Sphinx開発チームは、1.4系開発と並行してIssueの整理を進めています。不具合の再現手順確認や、ドキュメントの更新など、実装コードの修正以外にも多くのタスクがあります。ご協力いただける方がおりましたら、ぜひ、メーリングリストまでご連絡ください。よろしくお願いします。

■公式サイトに日本語訳ドキュメントが掲載されました

SphinxドキュメントをRead The Docs[注C]でのホスティングに移行しました。これによってRead The Docsが提供する機能である、ドキュメントの過去のバージョンの閲覧、翻訳された別言語のドキュメントを提供できるようになりました。

・日本語訳ドキュメントの新しいURL
http://www.sphinx-doc.org/ja/stable/

これまで日本語訳ドキュメントは「http://docs.sphinx-users.jp/」で公開していましたが、今回の移行によって、Sphinx公式ドキュメントと同じドメインで提供されるようになりました。既存のサイトへのアクセスは新しいURLへリダイレクトされます。

Sphinxにはドキュメントを国際化するためのしくみがあり、SphinxやPythonの日本語ドキュメントもこの国際化のしくみを使って提供されています。Sphinxの国際化は、gettext形式の翻訳カタログファイル(*.po)を用いて実現しています。原文に対する翻訳文を*.poファイルとして用意することで、原文を書き換えずにドキュメントをほかの言語向けに生成できます。これによって、原文の更新への追従がとても簡単に行えます。このしくみとオンラインの翻訳支援サービスTransifex[注D]を組み合わせることで、翻訳者は手軽にドキュメントの翻訳に参加できます。Sphinxの国際化機能についての詳細はPyCon JP 2015で紹介した資料[注E]をご参照ください。

■Sphinx入門 in PyLadies Tokyo Meetup #8

2016年1月16日に、PyLadies Tokyo[注F]主催のイベントでSphinxのハンズオンを実施しました(**写真A**)。PyLadies Tokyoは、PyLadiesの東京支部として女性Pythonista(＝Python利用者)をつなぐために活動している団体です。今回のミートアップには11名が参加しました。Sphinxの紹介後、ハンズオン形式で各参加者のドキュメント作成を講師4名でサポートしました。

Sphinx-users.jpでは、要望に応じてSphinxのワークショップやハンズオンの講師を派遣していますので、お気軽にご相談ください。

注A) https://pypi.python.org/pypi/Sphinx/1.3.5

注B) 不具合報告はメーリングリスト、またはSphinxのGitHubへお願いします。
メーリングリスト http://sphinx-users.jp/howtojoin.html
GitHub https://github.com/sphinx-doc/sphinx

注C) Sphinxのドキュメントビルドとサイト公開を自動的に行ってくれるサービス(本連載第7回(本誌2015年10月号)を参照)。https://readthedocs.org/

注D) https://www.transifex.com/sphinx-doc/sphinx-doc-1_3/dashboard/

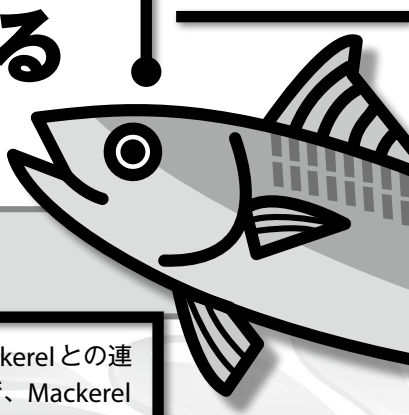
注E) http://www.slideshare.net/shimizukawa/sphinx-53764167

注F) http://tokyo.pyladies.com/

▼写真A　ハンズオンの様子

Mackerelではじめるサーバ管理

Writer 坪内 佑樹(つぼうち ゆうき) ㈱はてな
Twitter @y_uuk1



第13回 MackerelとServerspecを組み合わせたインフラテスト

今回は、サーバの状態をテストするフレームワーク「Serverspec」とMackerelとの連携がテーマです。Mackerel APIを利用してホスト情報を取得することで、Mackerelにおける「ロール」単位でテストを実行するといったことができ、テスト作業の省力化につながります。

インフラはコードで管理する時代

ここ数年、アプリケーションのテストと同じように、テストコードを記述してインフラをテストするという動きが盛んです。「テスト駆動インフラ」「インフラCI」というようなワードも登場しています。

Serverspec[注1]はインフラのテストを実践するための代表的なツールです。Serverspecはシンプルなツールがゆえに、さまざまなツールと組み合わせた運用ができます。今回は、Mackerel[注2]のAPIを用いて、Serverspecを効率よく運用するノウハウを紹介します。

Serverspecとは

Serverspecは、宮下剛輔氏により開発された、サーバの状態のテストをコードにより自動化するためのツールです。Ruby製のテストフレームワークであるRSpec[注3]がベースになっています。サーバの状態をテストするためのツールはほかにもありますが、ChefやPuppetのようなサーバ構成管理ツールに依存しないというのが、Serverspecの特徴です。

Serverspecは、さまざまな使い方を許容する懐の深いツールです。筆者は、Serverspecの使い方のうち、とくにサーバの本番投入前のチェックの自動化に注目しています。

たとえChefやPuppetなどでサーバの構築を自動化していたとしても、投入前には人の手でサーバにSSH接続して動作確認をしている、ということはよくあることです。しかし人の手で行われるチェックでは、一部の項目をチェックし忘れたり、チェック内容が属人化したりすることがあります。そこで、Serverspecでチェック内容をコード化しておけば、サーバ構築と同時にテストを走らせたり、コードの形でノウハウを共有できたりします。

Serverspecで何をテストするのか

Serverspecでテストできる項目は多岐に渡ります。具体的に、何をテストすればいいのでしょうか。たとえば、「nginxのプロセスが起動しているかどうか」などをテストできます。さらに例を挙げると、次のようなテストが記述できます。

・ポート80番がListenされているか
・/etc/nginx/nginx.confファイルが存在して

注1） URL http://serverspec.org
注2） URL https://mackerel.io
注3） URL http://rspec.info

いるか
・nginxユーザが存在しているか
・/etc/nginx/nginx.confに文字列gzip_vary onが含まれているか
・curl -I -X GET http://localhostを実行した結果に、文字列200 OKが含まれているか

実際のコードも紹介しましょう。ポート80番がListenされているかどうかを確かめるテストは次のように書けます。

```
describe port(80) do
  it { should be_listening }
end
```

そのほか、Serverspecが直接サポートするテスト内容やコードの書き方については、公式ドキュメント[注4]を参照してください。

基本的に、“普段人の手で確認していたような内容をテストすればよい”と筆者は考えます。逆に言えば、普段確認していないような内容を無理してテストする必要はないでしょう。もし、テストが書かれていれば防げた障害が発生すれば、その都度テストを書き加えていくことで再発防止につながります。

MackerelとServerspec

Serverspecがどのようなものかを簡単に紹介しました。次に、本題であるServerspecとMackerelの連携について紹介します。実際にServerspecを導入するとき、Mackerelを併用するとスムーズにServerspecを導入できます。

ロール単位でspecを書く

ServerspecはRSpecベースであるため、基本的にRSpecの流儀に則ってテストコードを書きます。specディレクトリ以下に、hoge_spec.rbというようなファイル名でspecファイル(テストコードを記述したファイル)を配置します。Serverspecのデフォルトでは次のように、ホストごとにディレクトリを作るようなレイアウトになります。

```
spec
├── myblogproxy001
│   └── nginx_spec.rb
├── myblogdb001
│   └── mysql_spec.rb
└── spec_helper.rb
```

しかし、システムの成長に伴ってホストが増加すると、同じ役割のホストのテストコードを共通化したくなります。

Serverspec Advanced Tips[注5]には「How to share Serverspec tests among hosts」という項目があり、たとえばMackerelにおけるロールのような、あるグループごとにspecを書く構成が紹介されています。ロールに所属するホストはあらかじめ設定に書いておき、テスト実行時にホスト名を指定すると、設定内容をみて、ホストに紐づくロールに対応するspecが実行されます

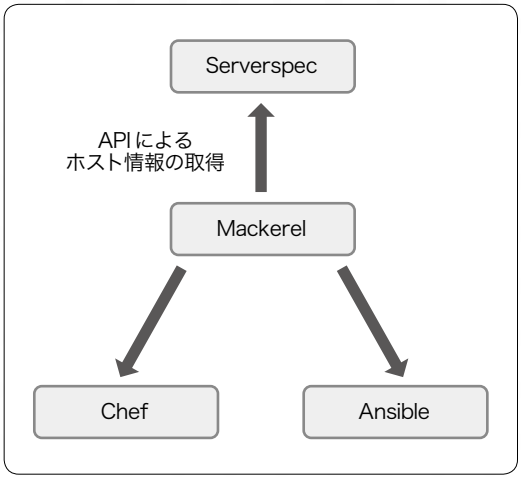
しかし、ホスト単位またはロール単位であっても、ホストの作成や退役にあわせて設定を書いていくのは面倒です。Serverspecに限らず、ツールを導入するたびにホスト情報をあちこちに書くはめになることは往々にしてあります。このような運用は、とくにホストをどんどん捨てて新しいホストを作成するようなImmutable Infrastructure[注6]的な運用にはあまり沿いません。

そこでMackerelでは、APIを用いて「ホスト情報を一元管理する」という思想を推奨しています。ツールを導入するたびにホスト情報を設定として書くのではなく、Makcerel APIを用いて動的にホスト情報を取得するという考え方です(**図1**)。

はてなでは、Mackerelをさまざなツールと

注4) URL http://serverspec.org/resource_types.html
注5) URL http://serverspec.org/advanced_tips.html
注6) URL http://chadfowler.com/blog/2013/06/23/immutable-deployments

▼図1　Mackerelと各種ツールの連携イメージ



も連携させています。たとえば、本連載の第4回(2015年6月号)で紹介した、tmux-cssh[注7]やCapistrano[注8]があります。さらに、AnsibleやChefのような構成管理ツールとも連携させています。AnsibleとMackerelの連携については、筆者ブログ記事[注9]も併せて参照してください。

Serverspecの場合は、ホスト名が与えられるとMackerel APIを叩いてホスト情報を取得し、サービスとロールから対応するspecファイルを決定できます。もちろん、逆にロールを与えて、ロール配下のホスト群にテストを回すということもできます。Serverspecは良い意味でホスト管理機能のようなものをサポートしていないため、APIで動的にホスト情報と実行すべきspecファイルを対応付けるのが簡単なのです。

では、具体的にMackerelと併用した場合のディレクトリレイアウトとテスト実行方法を説明します。

▼図2　ロール単位でspecを書くための構成

```
spec
├── common
│   ├── nginx
│   │   ├── default.rb
│   │   └── proxy.rb
│   └── postgresql
│       ├── default.rb
│       ├── master.rb
│       └── slave.rb
├── base_spec.rb
├── service
│   ├── myblog
│   │   ├── proxy_spec.rb
│   │   ├── app_spec.rb
│   │   └── db-master_spec.rb
│   └── mybookmark
│       ├── proxy_spec.rb
│       ├── app_spec.rb
│       ├── db-master_spec.rb
│       └── db-slave_spec.rb
└── spec_helper.rb
```

●ディレクトリレイアウト

ロール単位でspecを書くためのServerspecのディレクトリレイアウトの一例を**図2**に示します。base_spec.rbに各ロール共通のspecを書き、serviceディレクトリ以下にMackerelのサービス・ロールに対応するspecを書きます。

ミドルウェア単位でspecをまとめたいこともあります。そのようなときは、commonというディレクトリ以下に複数のロール間で使い回せそうなspecを書きます。

●Serverspecの実行

Serverspecのデフォルトでは、spec実行用にRakefileが生成されます。筆者はRakefileの書き方が難しくてあまり好きではないので、ここではThor[注10]を使います。Thorは、簡単にRubyのコマンドラインインターフェースを作成できるツールです。もちろん、Serverspec

注7）URL https://github.com/dennishafemann/tmux-cssh
注8）URL http://capistranorb.com
注9）「Ansible + Mackerel APIによる1000台規模のサーバオペレーション」
　　URL http://yuuki.hatenablog.com/entry/ansible-mackerel-1000
注10）URL http://whatisthor.com

▼リスト1　spec実行のためのThorfile

```
require 'mackerel/client'

RSPEC_OPT = ENV['SABASPEC_RSPEC_OPT'] || '--format doc -c'

class Spec < Thor
  include Thor::Actions
  default_task :host

  desc 'host', 'run spec for a host'
  def host(hostname)
    mackerel = ::Mackerel::Client.new(
      mackerel_api_key: MACLEREL_API_KEY
    )

    host = mackerel.get_hosts(name: hostname).first
    raise "Not found host #{hostname}" if host.nil?

    spec_files = ['spec/base_spec.rb']
    spec_files += host.roles.flat_map {|service, roles|
      roles.flat_map {|role| "spec/service/#{service}/#{role}_spec.rb" }
    }.select {|f| FileTest.exist?(f) }

    ENV['ASK_SUDO_PASSWORD'] = '1'
    ENV['RSPEC_SSH_HOST'] = hostname
    run("bundle exec rspec #{RSPEC_OPT} -r spec_helper #{spec_files.join(' ')}")
  end
end
```

がデフォルトで使用するRakefileを使っても問題ありません。

たとえば、**リスト1**のようなThorfileを用意します。Mackerel APIのRubyクライアントであるmackerel-client-ruby[注11]を用いて、与えられたホスト名からAPIを引いてホスト情報を取得し、ホストが所属するサービス・ロール名から対応するspecファイルの一覧を出し、rspecコマンドに渡してやります。

myblogproxy001.domainというホストに対してテストを実行したければ、次のようなコマンドでテストを実行できます。

```
thor spec:default myblogproxy001.domain
```

このとき、myblogproxy001.domainのMackerel上のサービスがmyblog、ロールがproxyならば、spec/service/myblog/proxy_spec.rbに書かれたspecが実行されます。myblogproxy001.domainと、サービス・ロールの対応を新たに書かなくてもよいところがポイントです。

◆ ◆ ◆

Serverspecを実際のサーバ運用にうまく組み込むために、Mackerel APIを活用する方法を紹介しました。

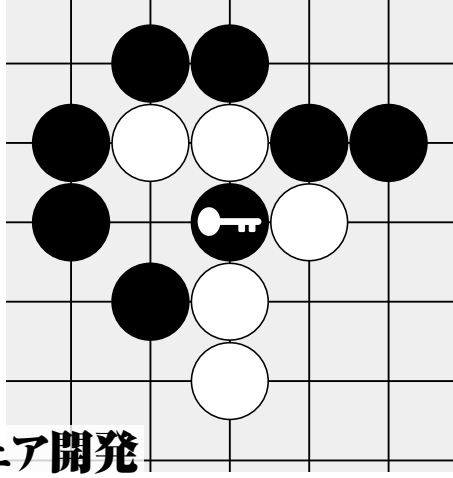
Serverspec以外にも、ホスト情報を扱うようなツールの運用効率を改善できると思います。たとえば、公式ドキュメント[注12]ではアプリケーションデプロイツールであるCapistranoとの連携を紹介しています。

APIを用いたホスト管理の一元化は、ほかのモニタリングツールにはなかなかない考え方だと思います。ぜひお試しください。SD

注11) URL https://github.com/mackerelio/mackerel-client-ruby
注12) URL http://help-ja.mackerel.io/entry/advanced/capistrano-2.x

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第三十回】セキュリティを意識したソフトウェア開発

セキュリティに限らず何事も予防が大切です。しかし、それを突き詰めていくと、「まずセキュリティを考慮した安全なソフトウェアの開発から始めなければならない」ということになります。今回は、セキュリティを意識したソフトウェア開発について考えてみます。

ソフトウェア開発と情報セキュリティを考えてみる

これまで脆弱性の枠組みの話題を取り上げたり、動的なメモリアロケーションを考える際にセキュアコーディングについて言及したりしてきましたが、ソフトウェア開発そのものは取り上げていませんでした。なぜならソフトウェア開発のプロセスにセキュリティを組み込むといっても、基本定石と呼べるようなものは、まだできあがっていないからです。

しかし将来的に、セキュリティを意識した開発プロセスがみなさんの目に入ってくる、あるいはソフトウェア開発やサービスの現場に入ってくることは間違いない、と筆者は考えています。そこで今回は、ソフトウェア開発と情報セキュリティを考えてみることにします。これまでとは違い、概念的な話が多くなりますし、紹介している方法論もまだ日本では普及しているとは言いがたい部分はあります。しかし、取り上げる価値は十分あると思います。

安全なソフトウェアを作るということ

本連載でのセキュリティの話題でも、「脆弱性が存在していること／設定に誤りがあることを前提に対策をする」または、「それらの事象が見つかった際に事後対応する」といったものが中心です。それゆえに(あるいは一般的に)、セキュリティ対策とはそのようなものと受け取られているかもしれません。

しかしながら、もしソフトウェアがより安全に作られているならば、セキュリティの問題も少なくなるのは当然です。ならば、「ソフトウェアを安全に作ることが先決ではないか」という考えに至るのは自然なことかと思います。

ところが、さまざまなニーズを仕様に取り入れていくための方法、たとえば1つの方法として要求工学からのアプローチを考えてみても、その中でユーザのニーズを汲み取るための議論は聞いたことはあっても、明示的にセキュリティそのものを仕様に入れていくための議論はあまり聞いたことがないと思います。ソフトウェア品質の向上のためにプロセス改善をするという議論はあっても、セキュリティ品質のためにプロセス改善をするという議論はあまりないと思います。

現状では、ソフトウェア工学として要求段階や設計段階から信頼性の向上を目指す体系的なアプローチと、情報通信分野におけるセキュリティの向上とは、必ずしもリンクしているとは言えない状況にあります。とくに脆弱性対応という視点から見たとき、リリース後の運用やメンテナンスの段階から入ってくることが多く、ソフトウェアのライフサイクルを考えた場合、上流工程でどう具体的に位置づけるべきかも明確とは言えません。次項で、FREAK攻撃を具体例として、もう少し具体的に考えてみたいと思います。

ソフトウェアのライフサイクルと脆弱性

ここでは、ソフトウェア・ライフサイクル・モデ

ルを前提として考えていきます。ソフトウェアの「要求」から「設計」、「開発」、「テスト」、「導入」、「保守／維持」に至るまでのライフサイクルとします。なお、ISO/IEC 12207（JIS X 0160）ではAcquisition（取得）、Supply（供給）、Development（開発）、Operation（運用）、Maintenance（保守）、Destruction（廃棄）と定義されています。

その前提で、約1年前の2015年1月に話題となったSSL/TLSの後方互換性に関する脆弱性（CVE-2015-0204）、いわゆるFREAK攻撃を考えてみましょう。この脆弱性の具体的な問題は、SSL/TLSには512bitの弱いRSA暗号を使うEXPORT_RSAという仕様が過去に存在し、その実装を持っているSSL/TLS実装では中間者攻撃が可能になるというものです。これはオープンソースの実装のOpenSSLだけではなく、Microsoft社の実装であるMicrosoft Schannel（CVE-2015-1637）や、Apple社の実装（CVE-2015-1067、APPLE-SA-2015-03-09-3）も同じFREAK攻撃が可能になっていました。

まず「要求」の段階で、「セキュリティのレベルを意図的に低くする」という誤った要求をしていたと言えるでしょう。そして、それが仕様化されていました。

この仕様が取り入れられ実装されてリリースされるわけですが、その際に、実装する側で「この仕様が必要かどうか」という検討がなされた様子はありません。別の言い方をすると仕様のフルスペックを実装することが要求になっていて、どの実装にも取り入れられてしまっていたという言い方も可能です。まず導入時リスクの評価をしておらず、また経年リスクの評価もしていません。

その状態で2015年になり、フランスの研究組織INRIA（フランス国立情報学自動制御研究所）の研究チームがFREAK攻撃を発見します。それ以降、脆弱性に対応する一連のプロセスに入りました。

「プロトコル仕様の要求と、実装に組み入れられる要求は別のことである」と言うことも可能かもしれません。しかし、RSA512はもうずいぶん前から、その安全性は無効です。今ではRSA1024も仕様から外されています。

SSL/TLS仕様の無効化、RSAの鍵長という形での暗号の危殆化（きたいか）がソフトウェアには反映されていません。つまり、このような状況の変化を取り入れるタイミングがありません。

仕様側に反映されるべきなのか、実装に反映されるべきなのかという議論はありますが、少なくともセキュリティの問題はライフサイクルに入っていないということになります。これはオープンソースの実装、Microsoftの実装、Appleの実装のどれにも入っている脆弱性ですから、たまたまどれかのソフトウェア・ライフサイクルのプロセスが悪かった、ということではないはずです。構造的な問題だと筆者は考えます。

ソフトウェア開発プロセスの成熟度

1980年代のことです。米国防総省（DoD：United States Department of Defense）が調達するシステムのソフトウェアは、予算の超過、スケジュールの超過、品質のばらつきなどさまざまな問題に悩まされていました。安定したソフトウェアの開発が難しいのは、今も昔も変わりません。DoDの必要とするソフトウェアですから、当然ながら軍事的な分野で使われます。そのようなソフトウェアが不安定なのは誰が考えても大きな問題です。

そこでDoDは、カーネギーメロン大学（CMU：Carnegie Mellon University）のソフトウェア工学研究所（SEI：Software Engineering Institute）に、ソフトウェア開発契約を行うベンダを峻別するための基準の研究を依頼しました。そして、1988年（出版は1989年）に出てきたのが、ソフトウェア開発プロセスの成熟度を計測する能力成熟度モデルCMM（Capability Maturity Model）です。成熟度は5段階あり**表1**のようになっています。

DoDはベンダにレベル3の基準を要求します。

後に、CMMは分野別に、システム・エンジニアリング（SE-CMM）、ソフトウェア開発（SW-CMM）、ソフトウェア調達（SA-CMM）、統合製品開発（IPD-CMM）などが作られます。それらを統合したモデルがCMMI（Capability Maturity Model Integration）

◆表1　CMMの開発プロセスの成熟度

成熟度	説明
レベル1：初期（Initial）	プロセスの導入も不十分で非常にレベルの低い管理である
レベル2：管理されている（Managed）	プロジェクトのレベルでプロセスの導入が行われ一部管理が可能となる
レベル3：定義されている（Defined）	組織のレベルでプロセスが導入され管理されている
レベル4：定量的に管理されている（Quantitative Managed）	プロセスは定量化され管理されている
レベル5：最適化されている（Optimizing）	プロセスの改善を行うことができる

です。現在では、CMMIのガイドラインが作られており、DoDの調達以外に政府調達にも使われています。日本でも、ソフトウェア開発プロセスの改善ということでCMMIに取り組む会社も増えています。

セキュリティとCMMI

CMMIにセキュリティを取り込む試みは、2010年に“Considering the Case for Security Content in CMMI for Services”として議論が始まっており、9ページの短いドキュメント[注1]にまとめられています。

このドキュメントではCMMI-SVC V1.2（2010）を対象としています。CMMI-SVCとは、サービスのためのCMMIとして作られたものです。今日の情報産業分野の企業の活動は、昔のように情報システムを顧客に提供するという単純な役割ではなく、「サービス」という付加価値を付けた形で提供しています。そのサービスのプロセスを改善し、あるいは評価するのがCMMI-SVC（CMMI Service）です。

CMMIアーキテクチャ・チームとCMMIプロダクト・チームは、サービスのプロセスの中にセキュリティを組み込むというアプローチを採用するようです。CMMIが非常に広く評価されて利用されているので、このアプローチがCMMI-SVCの中の要件として広く使われるのではないかと思います。

CERT-RMM

SEI CERTチームが作ったセキュリティ版CMMIとも言えるのが、CERT Resilience Management Model（CERT-RMM）[注2]です。とはいえ、CMMIとは違い、能力成熟度モデルをそのままの形では取り入れていません。

このモデルは大きく3つの要素からなります[注3]。

①プロセス（Process Areas）
②一般的な終了条件／経験（Generic Goals/Practices）
③特定生産物と副次的経験（Typical Work Products and Subpractices）

CERT-RMMとほかの規格との対応を見てみましょう（**図1**）。CMMIのCMMI-SVC（サービス分野のためのCMMI）とCMMI-DEV（ソフトウェア開発分野のためのCMMI）が、CERT-RMMの「プロセス」に対応しています。

ISO 27000シリーズは、CERT-RMMの「特定生産物と副次的経験」に対応しています。ISO 27000シリーズは情報セキュリティマネジメントシステム（ISMS：Information Security Management System）におけるベストプラクティスを提供するものです。セキュリティにおけるISO 27000シリーズは、よく品質管理のISO 9000シリーズや環境保護のISO 14000に例えられます。

反論は多々あるかと思いますが、CERT-RMMはいろいろな要素が集められてキメラ的になっており、これを一般に理解するには、あるいはプラク

注1）Eileen Forrester and Kieran Doyle, "Considering the Case for Security Content in CMMI for Services", Carnegie Mellon University, Oct., 2010. **http://cmmiinstitute.com/resources/considering-case-security-content-cmmi-services**
注2）Richard A. Caralli and et al., "CERT Resilience Management Model,Version 1.0 Improving Operational Resilience Processes", Software Engineering Institute, Carnegie Mellon University, May, 2010. **http://www.sei.cmu.edu/reports/10tr012.pdf**
注3）ここに挙げた訳語は、正式な訳語が見つからないため、筆者が仮に翻訳したものです。今後、適切な翻訳が出てきたなら、そちらを参照してください。

ティカルに現場に適応するには、たいへん難しい内容になっている、と筆者は感じました。

ソフトウエアセキュリティ保証成熟度モデル

「ソフトウエアセキュリティ保証成熟度モデル(Software Assurance Maturity Model：SAMM)」は、Open Web Application Security Project(OWASP)によって開発されました。『ソフトウエアセキュリティ保証成熟度モデル ソフトウエア開発にセキュリティを組み込むための手引き第1.0版』として邦訳され公開されています[注4]。

SAMMは、能力成熟度モデル(CMM/CMMI)と同様に、成熟度モデルを採用しています。CMM/CMMIのように開発プロセスのレベルを評価するために使えます。また、今の成熟度からさらに次の成熟度のレベルに上げるための指針としても利用できます。

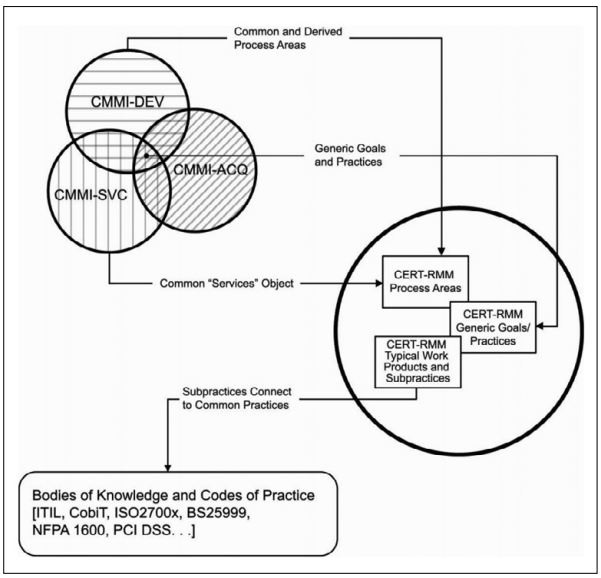
SAMMの全体像は**図2**のようになります。まずビジネス機能として「ガバナンス」「構築」「検証」「配備」という分類を作っています。各々の分類の中に3つのセキュリティ対策を作っています。

手引きからビジネス機能、セキュリティ対策のレベルを書き出してみたのが**表2**です。

ソフトウェア開発プロセスとしてのSAMM

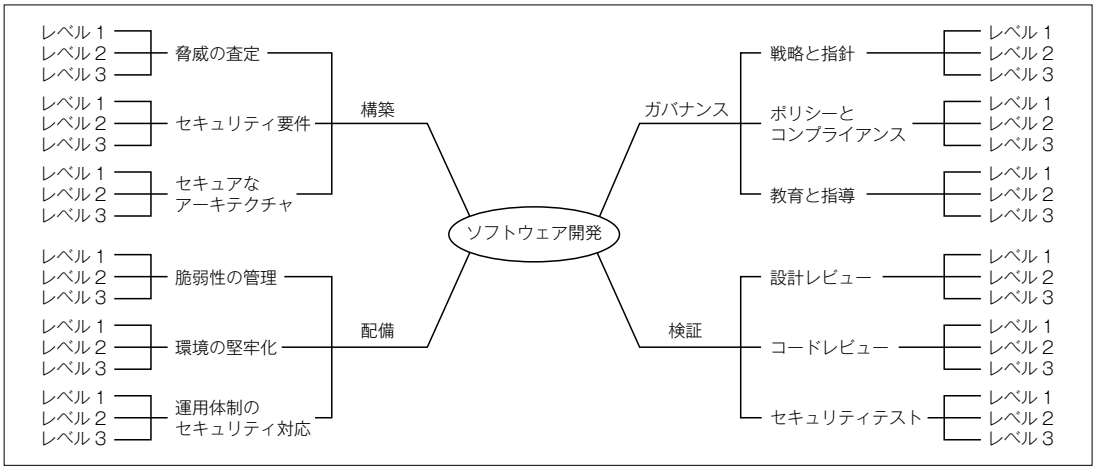
SAMMは、さらに各々のセキュリティ対策に対してレベル0～3の成熟度を示す基準を用意します。レベル3が最高で、レベル0は何もしていないという意

◆**図1　CERT-RMMとCMMI、ISO 2700xとの関係性**



(出典：CERT Resilience Management Model,Version 1.0 Improving Operational Resilience Processes)

◆**図2　SAMMの全体像(手引きを参考に作成)**



注4)　https://www.jpcert.or.jp/securecoding_materials.html#owaspsamm
ドキュメントのライセンスはCreative Commons Attribution-Share Alike 3.0 License(表示-継承3.0非移植(CC BY-SA 3.0))ですので、自由にダウンロードできますし、ほかの人にコピーを渡すことも許されます。

◆表2　SAMMのビジネス機能、セキュリティ対策の要約

分類		説明
ガバナンス		組織におけるソフトウェア開発活動全体の管理に関するプロセス。ビジネスプロセスも含む
	戦略と指針	ソフトウェアセキュリティ保証プログラム全体の戦略的な方針をたてる
	ポリシーとコンプライアンス	セキュリティおよびコンプライアンスの管理や監査フレームワークを作る
	教育と指導	ソフトウェア開発に携わる人にセキュリティ知識を広める
構築		組織における目標設定の方法と開発プロジェクト内でのソフトウェア作成方法に関するプロセス。製品管理、要件収集、概略レベルのアーキテクチャ仕様、詳細設計、実装などを含む
	脅威の査定	ソフトウェアに対する潜在的な攻撃を把握し脅威となるリスク管理する
	セキュリティ要件	ソフトウェアに必要なセキュリティの要件をまとめ仕様化する。ソフトウェア開発工程にセキュリティ関連の要件も含む
	セキュアなアーキテクチャ	デフォルト設定で安全な設計をする。ソフトウェア構築の基盤となる技術やフレームワークの管理も含む
検証		ソフトウェア開発を通して得た成果物をチェック、テストする方法に関するプロセス。テストやレビュー、あるいは評価などを含む
	設計レビュー	セキュリティの水準が確実に満たされるように設計のレビューなどを行う
	コードレビュー	脆弱性の発見やセキュアなコーディングがされているかレビューを行う
	セキュリティテスト	脆弱性の発見のためのテスト、リリース可能なレベルに達しているかを確認するためのテストを行う
配備		リリース、およびリリース後の対応に関するプロセス。エンドユーザへの製品出荷、内部／外部ホストへの製品配備、ソフトウェアの運用も含む
	脆弱性の管理	ソフトウェアに使われている外部のライブラリや環境なども含め脆弱性レポートや情報を管理し影響度を把握する
	環境の堅牢化	ソフトウェアが使われる環境の管理体制を整える。また、リリースされ運用されているソフトウェアの利用環境の改善を行う
	運用体制のセキュリティ対応	オペレータへセキュリティ対応として必要な情報を提供する。また、ソフトウェアの構成／配備／稼働などが安全に運用できるための対応を行う

味です。各々の成熟度レベルを評価するための基準として、さらに具体的な内容が定義されています。

ソフトウェア開発プロセスとしてのSAMMを見た場合、ソフトウェアのセキュリティを保証するための成熟度モデルとしては、これまでの中でもっとも明確で利用しやすいのではないかと思います。

ソフトウェア開発プロセスとセキュリティ

ここではCMMI-SVCにセキュリティを取り込むアプローチと、セキュリティのプロセスを別に用意するCERT-RMMとSAMMのアプローチの2つを取り上げました。CERT-RMMやSAMMは仕様も進んでいるので、そちらを取り入れる組織も多いかと思います。

しかし、筆者は疑問に思うことがあります。それは、たとえば「malloc()のときにバッファの最大長を確実に定義することによりバッファオーバーフローを発生させないようにする」という基本ができていないのは、セキュリティの問題というよりも、malloc()の呼び出しインターフェースの理解ができていない問題なのではないだろうか、そして、それは純粋にソフトウェア品質が低いだけではないだろうか、という疑問です。

どんなソフトウェアも完全なものは存在しません。ですから、セキュリティであろうと、ほかの理由であろうと、影響度に応じて対応する体制は必要だと筆者は考えます。想定外の誤った入力を与えるとプログラムがハングアップするのは昔から珍しいことではありません。しかし、これを意図的に発生できるとなると、Denial of Service attack（DoS攻撃）ということになります。両者の原因は同じ部分に存在しています。違いは、攻撃者が存在しているか否かです。

ソフトウェア開発プロセスとセキュリティというのは、実はほとんどが一体化できるもので、特別にセキュリティと区別したプロセスとして定義しなければならないものは、本来はそれほど多くないのかもしれません。

基本定石にはほど遠い

今回は、セキュリティを意識したソフトウェア開発プロセスというテーマで議論してきました。いくつかのアプローチを紹介するにとどまりましたが、必要性は誰もが認めていることでしょう。しかし、ソフトウェア工学的に見れば、現在はまだCMMが出現してきた1990年あたりの状況にいるのではないかと筆者は思います。セキュアコーディングのようなベストプラクティスを取り入れるレベルは可能でも、ソフトウェア開発のプロセスを改善していくのはまだまだ始まったばかりです。

このような議論を通して、ソフトウェアの脆弱性の問題やソフトウェアの安全な運用の問題は今後も難しい問題であるということを、感じてもらえれば幸いです。SD

◉CMMIとスプートニク・ショック

カーネギーメロン大学のSEIはピッツバーグにある研究所です。CMMIの研究資金はDoDから得ています。

ソフトウェアの信頼性を担保するための枠組みを研究するというのは、ソフトウェア開発すべてに関係する極めて基本的かつ汎用的な研究です。米国では「そのような技術に資金を投入し、その技術を広く公開するのが、米国の軍事技術を支えるために必要だ」という考え方をします。

技術の積み重ねはピラミッドに似ており、広い底辺から積み上げることによって、高い頂きを作ることができます。近視眼的に特定の兵器としての特定の機能に資金をつぎ込んでも、基礎的な技術がなければ高度な兵器を作ることはできません。とくに高度なソフトウェアは典型的です。

1957年にソ連は、米国より先に人類初の人工衛星を打ち上げました。それがスプートニク1号です。米国はソ連に負けないようにロケットを作り打ち上げますが、片っ端から打ち上げに失敗します。ソ連に技術的に先を越されてしまったうえに追いつけない状況でした。これがスプートニク・ショックです。

米国はなぜこのような状況になったのか、問題を調査しました。そして、「米国がソ連に追いつこうとしてもできなかったのは、米国内の基礎的技術や研究インフラが十分ではなかったためである」という結論に達しました。

米国はそれ以降、基礎的な技術にも軍事資金をつぎ込むことになります。そこで開発された技術を研究開発インフラとして広く公開することにより、米国全体の技術力および研究環境を向上させるというアプローチを採ります。その優れた研究開発環境から次の新しい技術が生まれ、あるいは高度な技術が安定的に確保でき、そこから優れた軍事技術が得られると考えたのです。

「インターネットの前身は核戦争に生き残るための技術として開発された」という誤った俗説をよく耳にします。なぜインターネットの基本技術であるパケット通信を研究開発したかというと、全米に散らばる高度な計算機を使うためのネットワーク・コストを下げるためです。研究コストの削減と、研究インフラの向上が目標です。技術というピラミッドの底辺を広げるためだったのです。

CMMIも最初は、「DoDがソフトウェアを調達する際に、ベンダ評価を行うためのメトリクスを作る」という目的で資金を得ました。現在もSEIのCMMI研究開発にはDoDの資金が入っています。基礎研究色の強いDARPA（Defense Advanced Research Projects Agency）ではなく、国防長官直属の組織である米国防長官府（OSD：the Office of the Secretary of Defense）とDoDと国防産業企業からなる組織NDIA（the National Defense Industrial Association）からの資金です。

資金面からみると軍事色がたいへん強いわけですが、CMMIは広く公開されています。なぜならばアメリカのソフトウェア産業が安定していなければ、高度な兵器のソフトウェアなど作れないからです。そう考えてみるとスプートニク・ショックがなければインターネットもCMMIも違うものになっていたかもしれません。

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19回 最終回 Identity Managementを使おう(その2)

今回は2015年12月号で紹介したIdentity Manager（IdM）によるクライアント認証について紹介します。

Author 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)
Twitter @rioriost

IdMクライアント設定の内容

あるシステムをIdMのクライアントとして設定することで、IdMが提供するドメイン管理機能を利用することが可能になります。設定方法として、ipa-clientパッケージによる半自動設定と手動設定が利用できますが、RHEL/CentOSであれば半自動設定のほうがもちろん簡単です。また、RHEL/CentOSの場合、kickstartによってIdMクライアントの設定を全自動で行うこともできます。

半自動・全自動でIdMクライアントの設定をすると、内部的には次の作業が行われます。この作業内容を大まかにでも理解しているとIdMのトラブルシューティングで役立ちます。

- IdMの認証局の証明書の設定
- IdMのXML-RPCサーバへのケルベロス接続の設定および有効化
- ipa-joinコマンドによるドメインへの参加
- ホストサービスのプリンシパルの取得と、/etc/krb5.keytabへのインストール
- certmonger[注1]サービスの有効化、SSLサーバ認証の設定、証明書の/etc/pki/nssdbへのインストール
- nscd(Name Service Cache Daemon)の無効化
- SSSDあるいはLDAP/KRB5の設定
- OpenSSHサーバとクライアントの設定、およびDNSのSSHFP[注2]レコードの作成
- NTPの設定

作業項目を見るとわかるとおり、IdMが提供する各種サービスをIdMクライアントが利用できる必要があるため、IdMのファイアウォールを**図1**のように設定[注3]しておきます。

IdMクライアントの準備

IdMクライアントとして設定するシステムがRHELの場合、有効なサブスクリプション契

▼図1 IdMのファイアウォール設定

```
# firewall-cmd --permanent --zone=public --add-port={80/tcp,443/tcp,389/tcp,636/tcp,88/tcp,464/tcp,53/tcp,88/udp,464/udp,53/udp,123/udp}
```

注1) 証明書の失効を監視し、認証局との連携により失効間近の証明書の更新を行うサービス
注2) SSH Fingerprintレコード
注3) 2015年12月号を参照

▼図2　ipa-clientパッケージのインストール

```
# yum -y install ipa-client
読み込んだプラグイン:product-id, search-disabled-repos, subscription-manager
依存性の解決をしています
--> トランザクションの確認を実行しています。
---> パッケージ ipa-client.x86_64 0:4.2.0-15.el7_2.3 を インストール

 ……<中略>……

 sssd                x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms  91 k
 sssd-ad             x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms 215 k
 sssd-common         x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms 1.1 M
 sssd-common-pac     x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms 134 k
 sssd-ipa            x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms 252 k
 sssd-krb5           x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms 129 k
 sssd-krb5-common    x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms 154 k
 sssd-ldap           x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms 194 k
 sssd-proxy          x86_64 1.13.0-40.el7_2.1           rhel-7-server-rpms 123 k

トランザクションの要約
================================================================================
インストール  1 パッケージ (+37 個の依存関係のパッケージ)

総ダウンロード容量: 9.8 M
インストール容量: 38 M
Downloading packages:

 ……<中略>……

完了しました!
```

約を保持していることが前提となるので、subscription-managerで有効化しておきます。identityサブコマンドによって登録状況を確認してから、作業を進めてください。なお以下に設定する手順では、最小構成でインストールしたRHEL 7.2を前提としています。

```
# subscription identity
```

サブスクリプションが確認できたら、ipa-clientパッケージをインストールします(**図2**)。

図2を見るとわかりますが、依存性を解決した結果としてsssd(System Security Services Daemon)がインストールされます。sssdはRHEL 6[注4]から導入された機能で、ホストやユーザの識別および認証を集中管理するためのサービスです。認証サーバへのアクセスができない場合にも、設定された時間内であればキャッシングによって認証サービスを利用することができます。

ipa-client-installによるクライアント設定

次にipa-client-installコマンドを実行します。この際、DNSサーバにKerberosやLDAPのSRVレコードとしてIdMが設定されていればIdMを自動検出しますが、そうでない場合には--serverオプションで指定します。あるいはIdMがDNSサーバを兼ねる場合には/etc/resolv.confでIdMをDNSサーバとして指定してからipa-client-installコマンドを実行します。詳細はmanページを参照してください(**図3**)。

これでIdMクライアント設定が完了しました。もし何らかの問題が発生した場合は、まずIdMのファイアウォール設定を確認してみてください。

注4) RHEL 7はもちろん、RHEL 5.6以降でも利用可能。

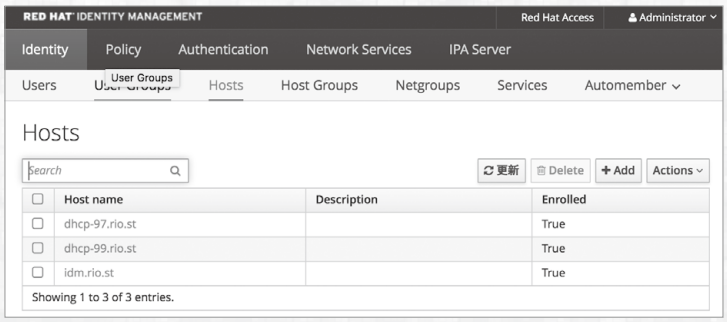
▼図3　ipa-client-installコマンドの実行

```
# ipa-client-install
Discovery was successful!
Client hostname: dhcp-97.rio.st
Realm: RIO.ST
DNS Domain: rio.st
IPA Server: idm.rio.st
BaseDN: dc=rio,dc=st

Continue to configure the system with these values? [no]: y  ←[y]を入力
Synchronizing time with KDC...
Attempting to sync time using ntpd.  Will timeout after 15 seconds
User authorized to enroll computers: admin  ←IdMの管理アカウントを入力
Password for admin@RIO.ST:   ←IdMの管理アカウントのパスワードを入力
Successfully retrieved CA cert
    Subject:     CN=Certificate Authority,O=RIO.ST
    Issuer:      CN=Certificate Authority,O=RIO.ST
    Valid From:  Fri Oct 09 12:36:47 2015 UTC
    Valid Until: Tue Oct 09 12:36:47 2035 UTC

Enrolled in IPA realm RIO.ST
Created /etc/ipa/default.conf
New SSSD config will be created
Configured sudoers in /etc/nsswitch.conf
Configured /etc/sssd/sssd.conf
Configured /etc/krb5.conf for IPA realm RIO.ST

 ……<中略>……

Client configuration complete.
```

▼図4　ユーザIDの取得確認

```
# id
uid=0(root) gid=0(root) groups=0(root) ↵
context=unconfined_u:unconfined_r:unconfined_ ↵
t:s0-s0:c0.c1023
# getent passwd admin
admin:*:662200000:662200000:Administrator:/ ↵
home/admin:/bin/bash
# getent group admins
admins:*:662200000:admin
```

▼図5　IdMのGUIでのホストの確認



設定完了後にIdMクライアント上でコマンドを発行すると、ユーザIDが取得できていることがわかります(**図4**)。

また、IdMにWebブラウザでログインするとIdMクライアントのホスト名、ここでは“dhcp-97.rio.st”が確認できます(**図5**)。

IdMのユーザでログイン

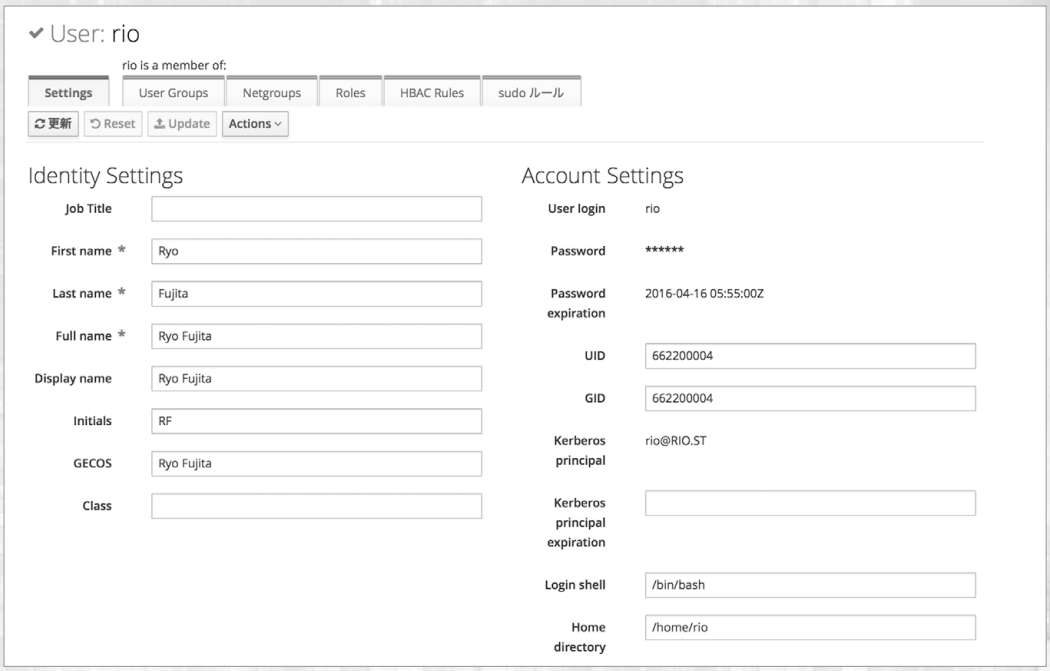
さらにIdMで管理されているユーザアカウントも、IdMクライアント上で利用できます。先にWebブラウザでユーザアカウントを確認しましょう(**図6**)。

次にIdMクライアント上でidコマンドを実行してみます(**図7**)。

これで**図6**で示されたユーザIDを取得できていることがわかります。では、**図8**でsuコマンドを実行してみましょう。

何が起きたのかは一目瞭然です。**図6**の右下に“Home directory”として“/home/rio”が指定されていますが、IdMクライアント上に存在しないためエラーが発生しました。IdMではPAMの

▼図6 IdMのGUIでのユーザアカウントの確認



モジュールであるpam_oddjob_mkhomedirあるいはpam_mkhomedirを利用して、ユーザのログイン時に自動的にユーザホームディレクトリを作成することができます。IdMクライアント上でauthconfigコマンドを実行し、再度、suコマンドを実行してみましょう(**図9**)。

実際の運用を考慮すると、ユーザ認証を一元化した一方でユーザのデータが各ホストに分散してしまう上記の方法はあまり望ましくありません。NFSや共有ストレージを用いて集中管理することになりますが、IdMではそれらのデータ領域をautomountと連携してユーザホームディレクトリに自動的にマウントすることもできます。

まとめ

IdMによるクライアント認証の一端を紹介しました。このほかにもホストごとのsudoの管理やパスワードの一貫したポリシーによる強力な管理など、セキュリティに役立つ機能が豊富に含まれており、ぜひ利用を検討してください。

次回から本連載は同僚の小島啓史氏にバトンタッチしますが、また近々本誌でお会いできることを楽しみに筆を置かせていただきます。

長期にわたる連載の機会をいただいた本誌・池本編集長に感謝いたします。SD

▼図7 idコマンドの実行

```
# id rio
uid=662200004(rio) gid=662200004(rio) ↵
groups=662200004(rio)
```

▼図8 suコマンドで実行

```
# su - rio
su: warning: cannot change directory to /home/ ↵
rio: そのようなファイルやディレクトリはありません
```

▼図9 IdMクライアント上でホームディレクトリを作成する

```
$ exit
# authconfig --enablemkhomedir --update
# su - rio
Creating home directory for rio.
最終ログイン: 2016/01/18 (月) 22:19:50 JST日時 pts/1
```

33

Debian Developer　やまねひでき　henrich@debian.org

Debian創設者の死

Debian Hot Topics

創設者Ian Murdockの死

2015年12月28日にDebianの創設者であるIan Murdockさんが42歳の若さで亡くなったことを、各種報道によって知った人は多いかと思います。

筆者が目にしてきたメディアでは、彼がDebianの創設者であることは述べられているものの、Debian創設時はどんな状況だったのか、彼は何をしてきたのか、今のDebianに対してどのような影響があるのか、などについてまとまった説明を目にすることはありませんでしたので、ここで振り返ってみようかと思います。彼の略歴は**表1**のとおりです。

Debianの創設～離脱

Debianは、1993年8月16日に当時大学生であったIan Murdockさんにより、ニュースグループcomp.os.linux.developmentで開始が宣言されました[注1]。この当時、ディストリビューションと呼べるものはSLS(Softlanding Linux System)[注2]と初の商用ディストリビューションであるYggdrasilぐらいしかなく、SLSを使っていてそのクオリティにフラストレーションを覚えていたIanが若さに任せて「ぶちあげた」という感じでしょうか。その投稿には、DebianはSLSをちょっと変えたものではなくスクラッチで作っ

▼表1　Ian Murdock氏の略歴

年	略歴
1993	Debianの開発を開始
1994	Debian宣言(Debian Manifesto)を発表
1995	Free Software Foundationからの支援を受け、システムプログラマとして勤務 業務としてDebianの開発を行う
1996	プロジェクトリーダーから降り、Bruce Perens氏を2代目リーダーとして指名
1997	アリゾナ大学でプログラマとして勤務
1999	Progeny Linux Systems社を創設
2005	Debian Common Core Alliance(DCCA)構想を発表 Progenyを離れ、Linux Foundationへ
2007	Sun Microsystems社に入社。Project Indianaを率いる Progeny操業停止
2011	ExactTarget社(後のSalesforce Marketing Cloud)に入社
2015	Docker社に入社

注1）URL https://groups.google.com/forum/#!msg/comp.os.linux.development/Md3Modzg5TU/xty88y5OLaMJ
注2）現在も残っている最古のディストリビューションSlackwareは、SLSをベースに1993年7月から開始されました。

ていることが述べられており、また、「SLSに比べてこんなにも良い点がある」と9つの点についての説明があります。

そして、「Debian宣言」を発表し、Debianで何を目指すのか、どのような形で開発を進めていくのかなどを明らかにしました。

「Debian Linuxはまったく新しいLinuxディストリビューションです。今までに開発された他のLinuxディストリビューションのように限定的な個人やグループが開発しているものではなく、LinuxとGNUの精神に則り、オープンに開発されています。Debianは、最終的にLinuxの名に恥じないディストリビューションを作り出すことを第一の目的としています。Debianは注意深く、また良心的にディストリビューションをまとめており、同様の配慮で保守・サポートしていく予定です。」
(Debian宣言[注3]より一部抜粋)

ただ単に現状に不満を覚えて理想を述べるだけなら誰にでもできますが、彼が違ったのは、実際に彼には優れた才能と馬力と魅力があったことです。彼は、今とは違い依存関係の処理などない荒削りな状態ながらもパッケージングツール「dpkg」の初期開発を行い[注4]、数十名のハッカーをまとめ上げて、Debian 0.01から0.90までのリリースを成し遂げます。

彼は1993年から1996年までの3年間リーダーを務めましたが、このころのDebianは若々しく少人数の集まりであり、今よりも素朴でテクニカルな側面が強かったようです。このあたりについては「Debian小史」[注5]にまとまっているのでご覧ください。

Progeny～DCCAの失敗

次にIanさんはアリゾナ大学でプログラマとして勤務する傍(かたわ)ら、1999年に新興企業Progeny Linux Systemsの創設に携わりました。

そして、Progeny社は、Debianベースの独自ディストリビューションを作りました。その内容として、Red Hat Linux(当時)に使われていたAnacondaインストーラをDebianへ移植しインストールを容易にしたり、さらにハードウェア認識が弱かった当時のDebianの弱点を「Dicover」と呼ばれるプログラムを開発して改善したりしました。ただ、利用者を多くは獲得できずに業績もさほどではなかったようです。

そのような状況を払拭するためか、Progenyは2005年に、Knoppix、LinEx、Linspire、MEPIS、Sun Wah Linux、Xandrosなどの派生ディストリビューションと共同して「Debian Common

COLUMN Debian GNU/LinuxのGNUとは

ちなみにですが、Debianは、GNUプロジェクト(FSF、フリーソフトウェア財団)によって1994年11月から1995年11月までの1年間支援を受けています。その絡みもあり、初期は単にDebian Linuxと呼んでいたのを、Richard Stallmanの主張を受け入れてDebian GNU/Linuxと呼ぶようになりました。

勘違いしている方も多いのでひとこと述べておきますと、Ian Murdockさんがリーダーを務めていた初期のころは確かにGNU/FSFに支援を受けており、その影響が大きいのですが、後年に独自の組織と哲学(Debian社会契約とDFSG[注6])を持つに至っている現在、「Debian GNU/Linuxというふうに、GNUの文字が入っているから、GNUと関連があって云々」などというのは見当違いな物言いです。

注3) URL https://www.debian.org/doc/manuals/project-history/ap-manifesto.ja.html
注4) 実際にchangelogファイル(/usr/share/doc/dpkg/changelog.Debian.gz)を覗いてみると最初期の2ヵ月ほどですが精力的に活動している様子がうかがえます。
注5) URL https://www.debian.org/doc/manuals/project-history/
注6) Debianフリーソフトウェアガイドライン。「オープンソースの定義」はDFSGをもとにできています。

Core Alliance(DCCA)」という組織を立ち上げることを、LinuxWorld San Franciscoで発表しました。

しかし、最も大きな派生ディストリビューションであるUbuntuは含まれなかったり、そもそものDebian自体の支持も得られていなかったりという状態で、弱小連合の体であり、あえなく失敗に終わります(このあたりはOSDNの佐渡秀治さんによる「kazekiriの日記：Ian Murdockが亡くなった件」[注7]に詳しくあります)。

Project Indiana

そして、Progenyを離れ、Linux Foundationを経てSun Microsystems社に入社したIanさんは、「Project Indiana」を率いることになります。これはLinuxの便利なところを積極的にSolarisに取り入れようとしていたプロジェクトで、とくにパッケージングシステムの面では遅れをとっていたため、当時のOpenSolaris上にDebianのようなパッケージシステム(Image Packaging System：IPS)とパッケージ・リポジトリを備えるなど「Linux風のモダンで便利なものに」と開発を進めていました。まさに彼が適任だったと言えましょう。

残念ながらその後、Oracle社のSun Microsystems買収に伴う方向転換により、2010年にOpenSolarisはオープンソースプロジェクトとしては停止し、Project Indianaの成果も一段落した形になります。

そして、IanさんはExactTarget社へ転職し、フリーソフトウェア／オープンソースソフトウェアの界隈ではその活動が表立って見えなくなり、今日に至ります。

彼の功績

2代目のプロジェクトリーダーであり、オープンソースの定義の創設に携わったBruce Perensさんは自身のブログ[注8]で、Ianさんについての評価として、彼に対する大学宛の推薦文で「Ianは、ゼロから何かを創りだすことができる稀(まれ)な人物の1人」と書かれていたことに触れ、「そのとおりの人物だった」と振り返っています。

今のDebianの特徴と言われるdpkg/APTや「フリー」の精神などについては、後年のプロジェクトメンバーの貢献によるものが大きいのですが、それでもIan Murdockという人物がDebianという「場」をゼロから作り、志に燃えた癖ぞろいのハッカーをまとめ上げてプロジェクトを形作ったことは偉大な事実です。

そして、Debianという場がなければ、Debianフリーソフトウェアガイドラインが生み出されることはなく、それを引き継いだ「オープンソース」という概念自体も作られることはなく、そして今みなさんが目にしているように、さまざまなソフトウェアがオープンソースとして自由に利用／改変／配布できている現在はなかったでしょう。そのこと考えると、その価値は途方もないものです。

Debianへの影響は?

Progeny Linux Systemsのビジネスが頓挫(とんざ)したあと、IanさんがDebianにかかわることはなかったため、彼の死によってプロジェクトの運営や開発が影響を受けることはありません(20年前に引退した創設者の死によって会社の業績が影響を受けるか、という話ですね)。しかしながら、1つの時代が終わってしまった、という寂しさは禁じえません。

Debianの開発状況

さて、現状のDebianの開発現場では、どんな進捗／議論が起こっているのでしょうか。

注7) URL http://srad.jp/~kazekiri/journal/599205/
注8) URL https://perens.com/blog/2015/12/31/ian-murdock-dead/

APTの高速化

パッケージメンテナの1人、Julian Andres Klodeさんのブログによるとバージョン1.1.7においてAPTが大幅に高速化されたそうです[注9]。

とくにpdiff[注10]の扱いについては40倍もの高速化を遂げ、さらにその後のバージョン1.2ではキャッシュ生成を15%ほど高速化したとのこと。testing/unstable利用者の方はapt-getを使用して、そのスピードを楽しんでみてください[注11]。

debian-installer Stretch alpha 5のリリース

debian-installerも新しいバージョンがリリースされました。今回のリリースで、i386アーキテクチャではi686未満のプロセッササポートを打ち切ることになりました[注12]。

Twitterなどのタイムラインでは、「32bitアーキテクチャを打ち切る!?」と誤解をしている人が散見されましたが、i686は64bit CPUのことではありません……。i686とはPentium Pro以降のCPUのことです。これを考えると、20年ほど前からのCPUもこの変更があったとしても続けて利用できることがわかります。さらにそれより古いi586マシン、つまり無印Pentium相当のサポートを打ち切っても、機器の経年劣化を考えると実質的に影響はないでしょう[注13]。

逆に、i386アーキテクチャについて、これまでi686未満のサポートのために実施できなかった最適化が行えるようになるため、多少なりともパフォーマンスがアップすることになります。

また、ストレージについても、高速なストレージに利用されているNVMe[注14]のサポートが追加されましたので、Intel SSD 750シリーズなどをお持ちの方はお試しください。

独立したnon-free-firmwareセクションを導入?

現状のDebianでは、non-freeリポジトリはデフォルトでは利用しないようになっています。

しかし、ネットワーク、とくにWi-Fiの利用においては、non-freeリポジトリを有効にしてファームウェアを導入しないと使えません。現実的には、そのような問題があるのは、みなさんもご存じのとおりです。

この夏のDebConfにてファームウェアの取り扱いに関する議論があり、「Going ahead with non-free-firmware」というメーリングリストのスレッドで、実際にこの作業を進めようという呼びかけがされました。これは、non-freeリポジトリを全面的に利用する設定を避けつつも、ファームウェアの利用を楽にすることでユーザの利便性を上げようという試みです。

「ほかのディストリビューションに比べて利用しづらい」という評判が、今までのDebianにはありました。その大部分は、このファームウェアの分離によるネットワーク設定の問題でしたので、これが実現できれば大きな前進となるでしょう[注15]。

どのように実装するのかについては、インフラストラクチャの制約などとの兼ね合いになりますし、そもそもこの実装が行われるかどうかは未定でもありますので、今後も注目して良いトピックだと思われます。SD

注9) URL https://juliank.wordpress.com/2015/12/26/much-faster-incremental-apt-updates/
注10) パッケージ一覧の差分データ。毎日4回新規パッケージが導入されることから、毎回のupdateですべてのデータを取得すると効率が悪くサーバにも負荷が高いため、パッケージの差分だけをpdiffとして切り出して取得するようにしている。
注11) ちなみに、「Debianはaptitudeを推していたのでは?」という方、aptitudeが積極的に推されていたのはDebian 5「lenny」リリースのころ(2009年ごろ)までで、現状はAPTのほうが活発に開発されています。
注12) 「i386アーキテクチャと呼んでいるのにi386(やi486、i586)はサポートしてないの? i686にリネームしたら?」という提案もありましたが、「i386という文字列は、さまざまなコード内に深く埋め込まれているのでとても変更できない……」という回答がありました。
注13) 例外的なものとしては、IntelのGalileoがi586らしいです。このような場合には困りますね。
注14) ストレージを接続するための規格で、HDD前提のSATAよりもSSDの性能が引き出せるようになっています。
注15) もう1つの問題は、旧態依然のdebian-installerのインターフェースですが、こちらも、GUIがデフォルトになったので多少マシになっているのではないかと思います。

Ubuntuのエディタといえばgedit

Ubuntu Japanese Team
あわしろいくや

Vimもいい、Emacsもいい。でも1つ忘れていませんか？　Ubuntuデフォルトのエディタはgeditです！

第3くらい? のエディタ、gedit

エディタといえばVimとEmacsで、しばしば論争（というかネタ）になります。本誌でも両者の連載があり、人気があるということを端的に示しています。またAtomだVisual Studio CodeだBracketsだと、最近リリースされたものもあり、Webブラウザの次くらいによく使うツールであるにもかかわらず、不思議と似たような時期に集中し、盛り上がりを見せています。

ここで忘れてはいけないのは、我らがgeditであります[注1]。Ubuntuを始め、多くのLinuxディストリビューションで、デフォルトでインストールされるエディタであるということもあってか、あまり顧みられることはありません。では低機能なのかと言われるとけっしてそのようなことはなく、むしろプラグインによる機能の拡張にも対応しているので、かなり「できるやつ」と断言できます。

というわけで、今回はgeditの解説をお届けします。

geditの歴史

geditの歴史をたどることは後にも先にもなさそうで、ちょうどいい機会ですのでスクリーンショットを中心にお届けします（**図1～4**）。ロゴや著作権者の変遷をご確認ください。

geditはGNOME Appsとして、すなわちGNOMEの一部分として開発されています。また歴史も古く、GNOMEの開発が始まったのは1997年8月15日とのことであり、geditの開発がいつから始まったのかは残念ながら調査できませんでしたが、バージョン管理システム（おそらくCVS）に登録されたのは1998年3月5日の0.2.1でした。GNOMEからあまり間をおかずに開発が始まったものと思われます。

GNOMEのバージョンアップに伴いネーミングルールも変更になり、たとえばNautilusが「ファイル」、Epiphanyが「ウェブ」といったように一般名称化されていきましたが、geditはgeditのまま残っています。

筆者の手元にあるgeditで一番古いのは、2002年7月19日にリリースされたDebian GNU/Linux 3.0（Sarge）のgedit 0.9.6です。**図1**を見ていただければ、GNOMEを含めて歴史を感じていただけると思います。

図2は、次に古いRed Hat Linux 8です。geditのバージョンは2.0.2で、GTK2+リリース直後のバージョンであることがわかります。

図3は、今回のメインプラットフォームであるUbuntu 15.10のgeditです。バージョンが3.10と古めです。

注1）nanoのことは忘れておきます……。

そして**図4**が、Ubuntu 16.04 LTS開発版のgeditです。3.18とバージョンが大幅に上がり、見た目もかなり違ったものになっています。いったいどうなっているのでしょうか。

UbuntuとgEdit

初期のUbuntu、正確にはUnityになるまでUbuntuはGNOMEとともにありました。ですので最初のバージョンである4.10からgeditがあり、Unityになっても継続して採用していることになります。

geditは3.12から、ほかのGNOME Appsに合わせてルック＆フィールを大幅に変更しました。しかしUbuntuはそれには追随せず、14.04から15.10まで3.10のままにしておきました。それがいよいよ16.04から3.18という新しいバージョンになり、変更を受け入れたということです。

今回の紹介もベースは3.10ですが、3.18でもおおむね同様です。

基本性能

まずはgeditにもともとある機能の解説から行います。もちろん全部は無理ですので、特徴的なもののみです。

シンタックスハイライト

シンタックスハイライトはエディタの極めて基本的な機能ですが、gedit[注2]も3.10で100強、3.18で120弱のハイライトに対応しているので、おおむね困ることはないでしょう。原則としてはファイルの読み込み時に自動的にハイライトが選択されますが、［表

注2）正確には使用しているライブラリであるGtkSourceView。

図1　gedit 0.9.6

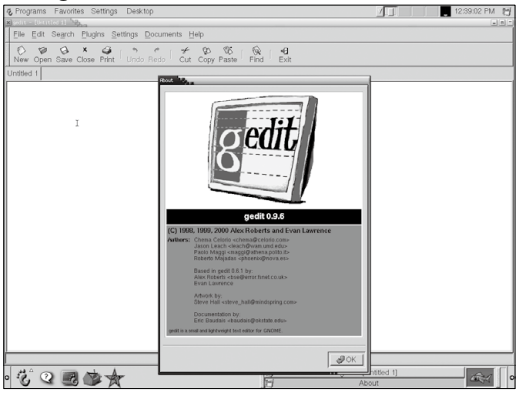


図2　gedit 2.0.2

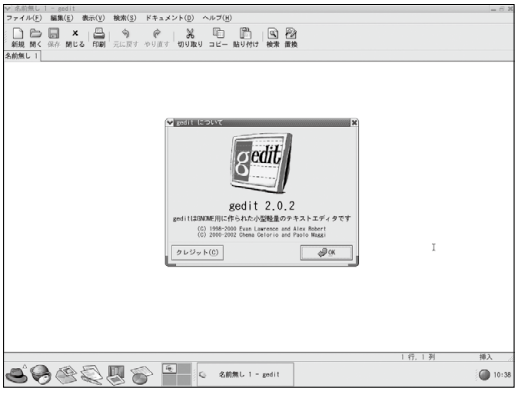


図3　gedit 3.10.4

図4　gedit 3.18.2

示]-[ハイライトモード]で任意のハイライトモードを選択できます。拡張子が違う場合などに便利でしょう。

ドキュメントの統計情報

プログラムを書く場合はさほど必要ではありませんが、文章を書く場合は大まかな文字数は把握しておきたいところです。そのような場合には[ツール]-[ドキュメントの統計情報]で知ることができます。選択した部分のみの情報を表示することもできますが、その場合は一度閉じなくてもそのまま範囲選択し、[更新]をクリックすると反映されるので便利です。

スペルチェック

[ツール]-[スペルチェック]から英語のスペルチェックもできます。[ツール]-[スペルミスの強調表示]も便利です。ついついtypoしてしまうものですので、必須の機能といえます。

カラースキーム

[設定]-[フォントと色]-[カラースキーム]で文字色や背景色を変更できます。白背景に黒文字は見にくいという場合には変更してみるといいでしょう。外部の定義ファイルの取り込みもできます。

プラグイン

[設定]-[プラグイン]を見ると、実は前に紹介した機能は1つを除いてプラグインで実装してあります。もちろんすべてのプラグインがあらかじめ有効になっているわけではありませんし、あとから追加することもできます。

コードスニペット

プラグインの[コードスニペット]を有効にすると、[ツール]-[Manage Snippets]という項目が追加されます。これを見るとだいたい使い方がわかります。

たとえばMarkdownで画像を挿入する場合は、

```
![説明](画像のパス)
```

と入力しますが、コードスニペットを有効にしてあれば、"img"と入力してTabキーを押すだけでテンプレートが即座に入力され、手間が省けます。

外部βツール

外部ツールに関しては、後ほど詳細に解説します。

再変換機能

IBusまたはFcitxでMozcを使用している場合、変換ミスした個所を選択し、変換キー[注3]で再変換ができます。これは別にgeditの機能というわけではないのですが、Ubuntu上だとできないアプリケーションがとても多いので[注4]、これだけでもgeditを使用したくなる理由になります。

geditのプラグインはUbuntuのリポジトリにいくつかあります。その中で必須なのはgedit-pluginsパッケージでしょう。このパッケージをインストールし、geditを再起動すると[設定]-[プラグイン]にたくさんのプラグインが追加されます。そのうちのいくつかを紹介します。

Bracket Completion

その名のとおり、ブラケットを補完してくれます。具体的には、"(" を入力すると "()" となります。

Text Size

このプラグインを有効にすると[表示]タブに[Larger Text]と[Smaller Text]と[Normal Size]が追加され、表示しているフォントサイズの変更が簡単にできるようになります。

組み込み端末

組み込み端末プラグインにチェックを入れ、[表示]-[ボトムパネル]を表示すると、ボトムパネルに端末が表示されるようになります。これでさっと書

注3) ATOKキーバインドだとShift+変換キー。
注4) LibreOffice WriterやAtomなどもそうです。

いたシェルスクリプトを実行してみたり、gitの操作などを行えば、いちいち端末とgeditを行き来しなくてもよくなり、利便性が上がります。画面を広く使いたい場合は[表示]-[ボトムパネル]のチェックを外してください。

外部ツール

説明には「外部コマンドとシェルスクリプトを実行します。」と書かれていますが、実のところこの外部ツールはシェルスクリプト以外にもPythonなどさまざまなLL(Lightweight Language)に対応します[注5]。

外部ツールを使いこなすために必要な情報はWiki[注6]にまとめられています。英語ですが、環境変数と外部ツールでできる設定についてという重要なことが書かれているので、事前に目を通しておいてください。

外部ツールプラグインを有効にすると、[ツール]-[External Tools]と[Manage External Tools]が増えています。まずは後者をクリックし、設定を行います。

MarkdownをHTMLに変換し、プレビューするツール

geditにはgedit-markdown[注7]というMarkdownプレビュープラグインがありますが、今回はそれを使用せず、Pandocで簡単なHTMLに変換してそれをWebブラウザに渡す、ということをやってみることにします。Pandoc(パッケージ名は"pandoc"と小文字です)とプレビュー用のWebブラウザであるウェブ(パッケージ名は"epiphany-browser")をインストールしてください。もちろんWebブラウザは既存のものでもいいのですが、プレビューの場合は単独にしておいたほうが便利でしょう。

[外部ツールの管理]を起動したら、左下の"+"をクリックして項目を増やします。タイトルを「Markdownプレビュー」などとし、[編集]の下には**リスト1**のように入力します。そして[ショートカットキー]と[保存]は[なし]にし(ショートカットキーはあってもいいです)、[入力]は[編集中のドキュメント]、[適用範囲]は[すべてのドキュメント]-[Markdown]にします(**図5**)。

これを[ツール]-[External Tools]-[Markdownプレビュー]で実行するとWebブラウザが起動し、そこにはMarkdownのプレビューが表示されています。

見てのとおり、このサンプルはpandocもepiphany-browserも存在するかどうかのチェックはしていません。そこは注意ですが、あとは何をやっているのかは見ればだいたいわかるかと思います。

最初の行で環境変数GEDIT_CURRENT_DOCUMENT_NAMEから取得したファイル名の拡張子を削除し、BASENAME環境変数に入れています。あとはpandocで変換してfindで見つけたそれと思わしきHTMLファイルをEpiphanyに渡しているだけです。Epiphanyは文字化けするかもしれませんが、その場合は文字コードをUTF-8に変更してください。SD

図5　Markdownプレビューの設定方法

注5) そもそもサンプルコードはPythonです。
注6) https://wiki.gnome.org/Apps/Gedit/Plugins/ExternalTools
注7) https://github.com/jpfleury/gedit-markdown

リスト1　外部ツールの管理に追加する

```
#!/bin/sh
BASENAME=$(basename -s .md $GEDIT_CURRENT_DOCUMENT_NAME)
pandoc -f markdown -t html5 $GEDIT_CURRENT_DOCUMENT_NAME -o $BASENAME.html
epiphany-browser --private-instance $(find $GEDIT_CURRENT_DOCUMENT_DIR -name $BASENAME.html)
```

第48回

LinuxのNVDIMM対応機能 ～libnvdimm～

Text：青田 直大　AOTA Naohiro

2015年はやはりLinux 4.3までで、新年最初となるLinux 4.4は1月10日にリリースされました。今回は、Linux 4.1と4.2で実装された、LinuxのNVDIMM対応機能libnvdimmについて見ていきます。

NVDIMMとは

NVDIMMとはNon-Volatile DIMMのことで、メモリ並の速さで読み書きが可能でありながら、一般的なストレージのようにシステムが再起動したり電源が切れたりしてもデータが残るデバイスです。

現状のストレージはCPUに対して速度が遅くセクタ単位でデータが読み書きされます。既存のI/Oスタックはストレージのそうした性質に対応して開発されてきました。一方でNVDIMMは十分に速度が速くbyte単位でデータの読み書きが行われます。この性質の違いから、既存のI/Oスタックをそのまま NVDIMM上で使うと不都合が起きることがあります。そこで、どのようにNVDIMMをユーザに見せるかが問題となります。Linux 4.2ではlibnvdimmという名前でNVDIMMに対応するサブシステムが導入されています。

NVDIMMはどのように見えるか

まず、NVDIMMがどのようにシステムから見えるのかを見ていきましょう。NVDIMMの存在をシステムに通知するには3つの方法があります。1つ目は、ACPI6.0に策定されているNVDIMM Firmware Interface Table（NFIT）を使う方法です。仕様が策定されていることからわかるとおり、この方法がこれからの標準となることでしょう。残る2つはlegacyな方法で、一方はe820を使って取得できるメモリリージョンとして指定する方法で、もう一方はカーネルのコマンドラインを使った方法となります。

e820とはx86におけるBIOSのメモリマップを取得する機能のことです。Linuxでも起動後にまずメモリマップを取得し、各メモリ領域のアドレスおよびその領域の種類を表示しています（**図1**）。領域の種類には、usable（OSから利用可能）、reserved（予約領域であり利用不可）、あるいはACPI data（ACPI Table領域）などがあります。この領域の種類の1つとして“type-12”があります。“type-12”は“OEM独自領域”を意味しますが、これが“Persistent Memory（legacy）”としてNVDIMMであると認識されます。

カーネルコマンドラインを使う方法では“memmap=nn[KMG]!ss[KMG]”として、アドレスssからサイズnnのメモリ領域を、type-12の領域となるようにe820で取得したメモリマップを修正します。こうして、先ほどと同様に指定した領域がNVDIMMとして認識されます。この方法ではメモリマップを手動で修正しているので、NVDIMMがシステムになくてもNVDIMMがあるかのように見せかけることもできます。

NVDIMMへの2つのアクセス方法

e820を使ったlegacyな方法では、ある領域がbyte単位でアクセス可能なPersistent Memoryであることだけが通知できました。ACPI NFITをサポートしたNVDIMMにおいては、これまでの方法(PMEMと呼ばれます)に加えてBLKというアクセス方法をサポートできます(**図2**)。

PMEMではアクセス可能な全領域がシステムのメモリ空間にマップされ、その領域を通して直接データが読み書きされます。このアクセス方法はシンプルではありながら、いくつかの弱点もあります。1つの問題は、byte単位でのアクセスが既存のI/Oスタックやファイルシステムとの相性が悪いことです。既存のI/Oスタックはセクタ単位でアトミックにデータが書き換えられることを前提として設計されています。こうしたシステムをそのままNVDIMM上で動かすと、電源やOSが落ちた場合に1つのセクタの一部のみが書き換えられ、残りは書き換え前のままであるという状態が起こりえます。既存のI/Oスタックはこうした状態は予測していないので、チェックサムなどが記録されていない限りシステムがめちゃくちゃになってしまうことでしょう。

もう1つの問題点としてメモリとストレージシステムでのエラー処理の違いが挙げられます。PMEMのようなメモリコピーでのアクセスの場合、デバイスのアクセスした部分にハードウェア的な障害があると例外が発生します。一方でストレージシステムでは、データ転送コマンドを発行し、その完了後にエラーが起きたかどうかを確認するようなしくみになっています。

こうしたメモリとストレージとの違いを吸収するために使われるのが、BLKとなります。BLKではアクセス可能な領域に対していくつかの「読み書き用の領域」(sliding window)が設定され、その領域に読み書きしたいアドレスやサイズを書く領域、および読み書き用バッファが置かれます。この“sliding window”を通すことで、

▼図1　e820によるメモリマップ 取得例

```
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000e0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000bffdffff] usable
[    0.000000] BIOS-e820: [mem 0x00000000bffe0000-0x00000000bfffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000feffc000-0x00000000feffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fffc0000-0x00000000ffffffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000100000000-0x000000023fffffff] usable
```

既存のストレージシステムとの齟齬がなく、データを読み書きできます。

また、デバイスによってはFlush Addressというものが設定されていることもあります。これはNVDIMMに書いたデータが永続化されたことを保障するために使われるアドレスを指定するものです。このアドレスにデータが書かれるとfsyncのようにデータが永続化されたことが保障されます。

ACPI NFIT

次にNFITの構造について見ていきましょう。前項で紹介したようにNFITに対応したデバイスの場合、NVDIMMに対して2つのアクセス方法があったり、あるいは“sliding window”があったりと、legacyなPersistent Memory領域より複雑な構造となっています。

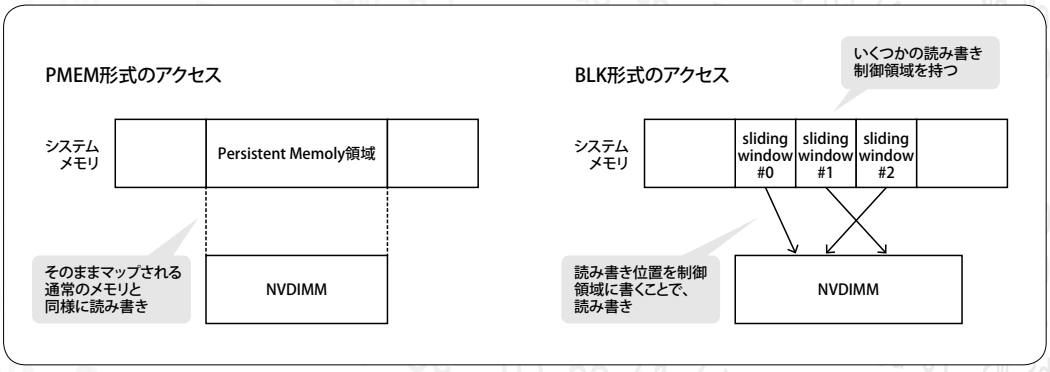
ここでは**図3**に示すようなマシンを例として、NFITがどのようになるかを見ていきます。この**図3**で示している構成はtools/testing/nvdimm/下のNFITおよびlibnvdimmのテスト用モジュール(nfit_test)を読み込むことで疑似的に作られる環境です。

この**図3**の環境にはnmem0からnmem3の4つの32MBのNVDIMMがあります。そして、そのNVDIMMの上にregion0からregion5まで6つのregionが定義されています。このうちregion0とregion1はPMEMでアクセス可能な領域で、region2からregion5の4つがBLKでアクセス可能な領域です。region0はnmem0とnmem1の前半分にまたがった領域で、region1はすべてのNVDIMMの後ろ半分にまたがる領域になっており、それぞれ32MBと64MBのサイズとなっています。region2からregion5はそれぞれのNVDIMM全体を占める、それぞれ32MBの領域になっています。このようにPMEMの領域は、複数のNVDIMMにまたがり1つの領域を作ることができます。一方で、BLKの領域はNVDIMM間を越えることができません。

NFITは、こうした領域を記述するための複数の構造体が並んだものです。各構造体は、構造体の種類とデータ長を示す共通のヘッダとそのあとに続く種類ごとのデータからなっています。テスト用のモジュールの場合、❶システムのメモリ領域を記述するテーブル(System Address Table)、❷NVDIMMとメモリ領域のマッピングを記述するテーブル(Memory Map Table)、❸BLK用のコントロール領域を記述するテーブル(Control Region Table)、❹BLK用のデータ用の領域を記述するテーブル(Data Region Table)、❺データのflush用のアドレスを記述するテーブル(Flush Address Table)が設定されています。

まずSystem Address Tableのデータから見ていきましょう(**図4**)。これはシステムの物理メモリアドレスのうちどの部分がどのような種類のメモリ領域であるかを記述するものです。

▼図2　PMEM形式とBLK形式のアクセス比較



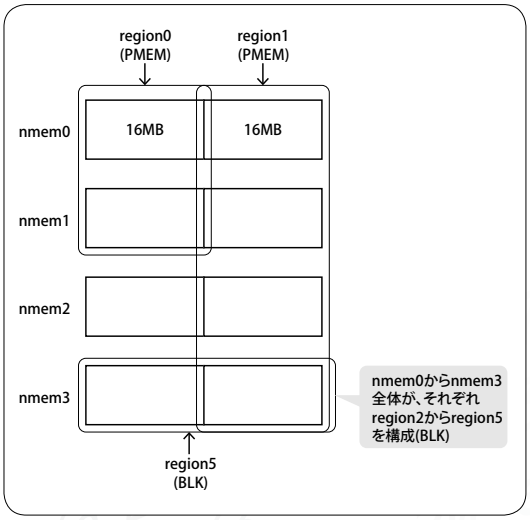
たとえば、region0に対応する部分を見るとアドレスXから32MBの領域がNFIT_SPA_PM(=byte単位でアクセス可能なPersistent Memory領域)として記述しています。また、BLKアクセス用の読み書きに使われる領域であるNFIT_SPA_DCRなども設定されています。

次にMemory Map Tableを見てみましょう。これはNVDIMMと先ほどのメモリ領域のマッピングを記述します。たとえば1つ目のエントリが、nmem0の0byte目から16MBが、region0の0byte目からの16MBの領域に対応していることを示し、2つ目のエントリはnmem0の0byte目から16MBが、region0の16MB目から32MBの領域に対応していることを示します。これによってregion0のどの部分が、どのNVDIMMのどこに位置しているのかのマッピングが完成しています。

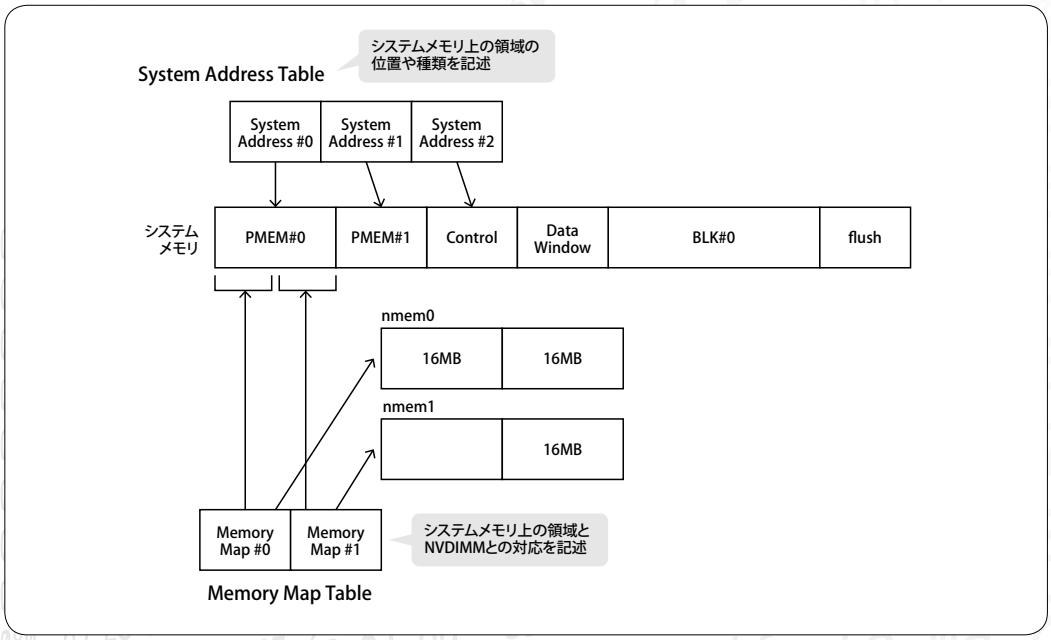
次のControl Region Tableでは、BLKアクセス用の"sliding window"の個数、およびwindowのコントロール用のレジスタの先頭アドレス、ステータスレジスタの先頭アドレスといったデータを保持しています。同様にDataRegion Tableは、BLKアクセス時のデータ転送バッファの位置などの情報を保持しています。最後のFlush Address Tableは前述のflushに用いられるアドレスを記述しています。

これらの情報が読み込まれているのを、/sys下のファイル群からも確認してみましょう。先ほどのテストモジュールが読み込まれると/sys/devices/platformの下に"nfit_test.0"、"nfit

▼図3　NFITの例



▼図4　System Address TableとControl Region Table



_test.1"とplatformdriverのディレクトリができています。"nfit_test.0"の方が、前述の**図3**に示したものですのでそちらを見ていきます。"nfit_test.0"ディレクトリの中にはさらに"ndbus0"ディレクトリがあり、その下からおもなコンテンツが始まります(**図5**)。

ディレクトリnmem0からnmem3が**図3**中の4枚のNVDIMMに対応し、ディレクトリregion0からregion5がそれぞれのregionに対応するデータを保持しています。**図3**中にないnmem4、region6、region7は、もう1枚別のNVDIMMによるものです(図からは省略しています)。

この中で、たとえばnmem0/nfitの中のファイルを見てみると、Memory Map Tableに書かれているNVDIMMのデバイス情報がとれていることがわかります(**図6**)。

また、region0とregion2の中を見てみましょう(**図7**)。まず、"devtype"ファイルにそれぞれ"nd_pmem"と"nd_blk"と、どちらのアクセス方法が使えるregionであるのかが書かれています。また、"size"ファイルからはたしかにregionのサイズである32MB(= 33,554,432byte)が取得できています。さらに、mapping0やmapping1といったファイルの中の各regionが、どのNVDIMMのどの領域にマップされているのかの情報が書かれていることがわかります。

NVDIMMのnamespace分割

さて、**図3**中のregionを見てみると、PMEM

▼図5　nfit_test.0/ndbus0の全体像

```
$ ls -l /sys/devices/platform/nfit_test.0/ndbus0
total 0
-r--r--r-- 1 root root 4096 Jan 25 09:54 commands
drwxr-xr-x 3 root root    0 Jan 25 09:54 nd
drwxr-xr-x 2 root root    0 Jan 25 09:54 nfit
drwxr-xr-x 4 root root    0 Jan 25 09:54 nmem0
drwxr-xr-x 4 root root    0 Jan 25 09:54 nmem1
drwxr-xr-x 4 root root    0 Jan 25 09:54 nmem2
drwxr-xr-x 4 root root    0 Jan 25 09:54 nmem3
drwxr-xr-x 4 root root    0 Jan 25 09:54 nmem4
drwxr-xr-x 2 root root    0 Jan 25 09:54 power
-r--r--r-- 1 root root 4096 Jan 25 09:54 provider
drwxr-xr-x 6 root root    0 Jan 25 09:54 region0
drwxr-xr-x 6 root root    0 Jan 25 09:54 region1
drwxr-xr-x 4 root root    0 Jan 25 09:54 region2
drwxr-xr-x 4 root root    0 Jan 25 09:54 region3
drwxr-xr-x 4 root root    0 Jan 25 09:54 region4
drwxr-xr-x 4 root root    0 Jan 25 09:54 region5
drwxr-xr-x 4 root root    0 Jan 25 09:54 region6
drwxr-xr-x 6 root root    0 Jan 25 09:54 region7
-rw-r--r-- 1 root root 4096 Jan 25 09:54 uevent
-r--r--r-- 1 root root 4096 Jan 25 09:54 wait_probe
```

▼図6　nmem0のデバイス情報表示

```
$ grep . nmem0/nfit/*
nmem0/nfit/device:0x0
nmem0/nfit/format:0x0
nmem0/nfit/handle:0x0
nmem0/nfit/phys_id:0x0
nmem0/nfit/rev_id:0x1
nmem0/nfit/serial:0xffffffff
nmem0/nfit/vendor:0xabcd
```

▼図7　region0とregion2のデータ表示

```
$ grep . region0/{devtype,size,mapping*}
region0/devtype:nd_pmem
region0/size:33554432
region0/mapping0:nmem0,0,16777216
region0/mapping1:nmem1,0,16777216
region0/mappings:2
$ grep . region2/{devtype,size,mapping*}
region2/devtype:nd_blk
region2/size:33554432
region2/mapping0:nmem0,0,33554432
region2/mappings:1
```

の領域とBLKの領域が重複している部分があるのに気がつきます。当然、PMEMでの読み書きとBLKでの読み書きとを同時に行った場合の動作は未定義となっています。そこでストレージのパーティションのように、namespaceと呼ばれる単位で領域を分割できます。

namespaceについての情報は、label storageというNVDIMM上の特別な領域に“label”と呼ばれる形式で記録されます。label storageは、2つのlabel index blockと複数のlabelから構成されています。label index blockというのは、ファイルシステムのsuper blockのようにlabel storage全体のメタデータを保管しているblockで、保管可能なlabelの数や、どのlabelが空いているのかといったデータが保存されます。

ここでlabel index blockが2つあるのはatomicに更新を行うためで、有効なindexはこの2つのうちのどちらか1つとなります。label indexの書き換えを行うときには、まず現在有効なindexをもう一方のindexにコピーし、書き換えを行ったうえでchecksumおよびシーケンス番号を更新します。checksumとシーケンス番号が書き変わるまでは、書き換え中のindexは古い、またはchecksum不整合とみなされます。

labelにはPMEMアクセス領域用とBLKアクセス領域用との2種類があり、regionのときと同様にPMEMの方はNVDIMM間にまたがることができ、逆にBLKの方はまたがることができません。

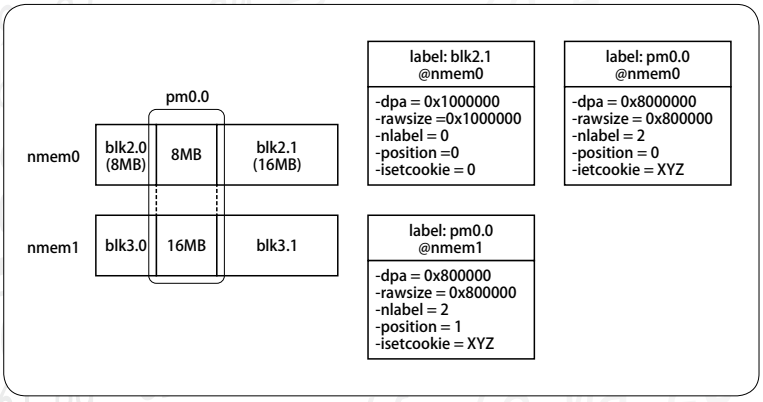
図8に示すように、namespaceを分割するときに各NVDIMM上のlabelがどうなるかを見ていきましょう。まず、blk2.1のlabelを見てみましょう。こちらはNVDIMM間をまたがらないため簡単で、dpaとrawsizeフィールドに記録されているNVDIMM上のアドレスとサイズだけが重要な情報です。

次にpm0.0を見てみましょう。このnamespaceはnmem0とnmem1にまたがるため、それぞれに1つのlabelが作られます。こちらのlabelの場合、dpa、rawsizeのほかにnlabel、position、isetcookieも重要なフィールドとなります。nlabelは、そのlabelが記述するnamespaceがいくつのlabelで成立するかを表します。ここではpm0.0がnmem0とnmem1から成立するため、nlabel = 2となります。positionはnamespaceを構成するlabelの中で、このlabelが何番目であるかを示しています。つまり、nmem0上ではposition = 0、nmem1上ではposition = 1となるわけです。最後のisetcookieはnamespaceを識別するためのIDとなっています。

Block Translation Table(BTT)

最後にBlock Translation Table(BTT)について簡単に紹介します。前述したとおり、既存のI/Oスタックはセクタ単位でatomicに書き換えられることに依存した設計を行っています。しかし、NVDIMMでは書き換えがbyte単位で行われるため、この前提が崩れてしまいます。BTTは変換テーブルを設置することでソフトウェアでNVDIMM上にセクタ単位の書き換え機能をつけ加えるものです。こちらを使うことでBLKによるセクタ単位の書き換えができないデバイス上でも既存のI/Oスタックをそのまま動かすことが可能となります。SD

▼図8　namespace分割時のNVDIMM上label



March 2016

NO.53

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
内山 千晶　UCHIYAMA Chiaki　chiaki@jus.or.jp
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

オープンソースの知見を深める秋の大阪２連戦

今回は、2015年10月末に大阪で行ったSphinxの勉強会と、2015年11月にやはり大阪で行った研究会の模様をお伝えします。

jus勉強会（大阪）

■Sphinxワークショップ@関西
【講師】安宅 洋輔（Sphinx-Users.jp）、清水川 貴之（Sphinx-Users.jp）
【司会】榎 真治（日本UNIXユーザ会）
【日時】2015年10月31日（土）13:00～17:30
【会場】大阪・ECCコンピュータ専門学校 2号館 2階 2303教室

10月31日の大阪開催のjus勉強会は、オープンソースのドキュメンテーションツール「Sphinx」[注1]を用いたドキュメント作成についての勉強会を開催しました。講師には関東からSphinx-Users.jpの安宅さん、清水川さんのご両名をお迎えしました。

SphinxはPython製のドキュメンテーションビルダーで、文章を構造化しやすく、美しいドキュメントを簡単に生成することができます。勉強会の前半は安宅さんによる自社にSphinxを導入したときの事例紹介、後半は清水川さん主導によるハンズオンが行われました。

安宅さんの事例紹介は、一般的にドキュメンテーションツールとして利用されているワープロソフトやスプレッドシート、Wiki、MarkdownとSphinxとの比較から始まり、どのような経緯でSphinx導入に至ったか、そして利用に際し周囲への理解と環境整備の働きかけをどのように行ったかについて、丁寧に話をしていただきました。後半は、まず清水川さんからSphinxの歴史やAPIなどの説明をしていただいた（**写真1**）あと、実際に手を動かして文書作成に取り組みました。最後の質疑応答や、ライトニングトークでは、参加者の方々の取り組みや抱えている問題の共有が活発に行われ、好評のうちに終了しました。

写真1　清水川貴之 氏

遠方よりお越しいただいた講師のご両名、そして参加者のみなさん、誠にありがとうございました。勉強会の資料が講師により公開されていますので、こちらもぜひご覧ください。

- **安宅さんの資料「Sphinx事例紹介 ～SIerの場合～」**
  http://www.slideshare.net/kk_Ataka/jus-sphinx-sphinx-54608065
- **清水川さんの資料「Sphinx紹介」**
  http://www.slideshare.net/shimizukawa/jus-sphinx-sphinx

注1）URL http://www.sphinx-doc.org/

jus研究会　大阪大会

■Symfony2で港湾管理者の業務システムを作った話
【講師】佐々木 伸幸（㈲サンビットシステム）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2015年11月6日（金）17:00～17:50
【会場】大阪南港ATC ITM棟 10階 サロン

2015年も関西オープンフォーラム（KOF）の中で研究会を実施しました。講師は札幌からサンビットシステムの佐々木さん（**写真2**）をお招きし、オープンソースソフトウェア（OSS）で構築した港湾業務システムの話をしていただきました（**写真3**）。参加者は15人でした。

「今回のシステムを理解するには、まず港湾業務を知る必要がある」ということで、研究会の前半は港湾管理者が行っている業務の説明がなされました。おもな内容としては、港湾施設の定義、利用料の徴収、国土交通省に報告する港湾統計、入出港や荷さばき地使用などの申請許可、申請の業務フローなどです。

これに続いて、開発した港湾業務システムの話がありました。旧来のシステムはWindowsとAccessとOracleで動いていましたが、実務の大部分は紙の書類や管理者による手作業で行われていました。これをLAPP（Linux＋Apache＋PostgreSQL＋PHP）とSymfony2で構築したシステムに置き換えることにより、Webによる申請と許可、書類のPDF化、SVG（Scalable Vector Graphics）や地図を用いた区画の可視化、予約業務の自律化などを実現しました。たとえば、荷さばき地の確保を行う場合、従来は利用者側が申請すると港湾管理者が手動で割り当てていましたが、本システムでは地図を表示して利用者側が自分で用地を確保するので、管理者の手間が軽減されます。

さらに、システムの要素技術の解説がありました。ベースとしてLAPPを使用し、その上でフレームワークとしてSymfony2を用いてプログラムを開発しています。Symfony2は大きなフレームワークなので手軽とは言えないが、大規模開発には向いているとのことです。また、地図や区画の表示にはHTML5のCanvasとSVGを使用しています。さらに、地図の制作にWMS（Web Map Service）やWFS（Web Feature Service）などのGIS（Geographic Information System：地理情報システム）関連技術を使用しています。


写真2　佐々木伸幸 氏


写真3　jus研究会 大阪大会の様子

構築したシステムは港湾局に納品され稼働していますが、大きなトラブルもなく、港湾管理者の調整業務は確実に減少したそうです。今回のシステムを一般化したものを「あまつみ」[注2]という名前で自治体向けに提供しており、またシステムの一部をOSSとして公開すべく準備中だそうです。

業務システムの話ということで、取っつきにくい印象があったかもしれませんが、講演ではSymfony2やSVGのコード実例に基づく解説もあり、技術的にも内容の濃い研究会でした。SD

注2）URL http://amatsumi.net/

Hack For Japan
エンジニアだからこそできる復興への一歩

Hack For Japan

第51回

島ソン2015！ 離島でのアイデアソン！

●Hack For Japanスタッフ
小泉 勝志郎　KOIZUMI Katsushiro
Twitter @koi_zoom1

この連載の第32回（2014年8月号）でも、Code for Shiogama/Hack For Japanが2014年5月24〜25日に行った「島ソン〜浦戸諸島ハッカソン〜」を紹介しました。今回はその島ソンの第2弾となる「島ソン2015」（2015年10月31日開催）を紹介します。

島ソンとは？

第1回 島ソン

島ソンの舞台は宮城県塩竈（しおがま）市の離島である浦戸諸島。塩竈市の中ではもっとも東日本大震災の爪痕が色濃く残る地域です。昨年の島ソンは宮城県塩竈市出身の筆者が、同じ塩竈で震災被害の大きかった浦戸諸島にITの力でより貢献したい、と以前から思っていたところからスタートした企画でした。

テーマは「島の課題を解決する」。島の住民の方たちや島で仕事をする方たちに集まってもらい、課題を集めてハックをするという内容です。島を歩いて離島での生活や仕事の様子を感じて課題を洗い出し、それに向かっていくハッカソンです。

離島だけあっていろいろ大変なこともありました。まず電波の入りに難がありネットワークにつながらないので、電波ハックとしてステンレスボウルでパラボラアンテナを作りました。そして食事を提供してくれる場所もないので、食材を持ち込んでの食材ハック（調理）も自分たちで行っていたのです。

また、島ソンはハッカソンが抱える問題点について自分なりに対処してみたハッカソンでした。その問題点とは「このプロダクトは本当に役に立てるのだろうか？」「プログラムを組めない人が暇になってしまう」ということ、そして「どんな良いプロダクトでもハッカソンが終わるとそれで終わりになってしまう」ということです。

そこで前回の島ソンではここを解決するために、石巻専修大学の舛井研究室とタッグを組み「学生のみなさんがこのイベントを通じて、自分の名刺代わりとなるコンテンツを作る」という方法を取りました。舛井研究室のみなさんは経営学部なのでプログラム経験のある方はわずかです。しかし、「自分の名刺代わり」にするためにはハッカソンが終わった後も活動を続けなければならない。この流れでハッカソンの後も活動を続けようという試みでした。この試みのおかげで、昨年の島ソンは今も動くプロジェクトを生み出せました。

昨年の成果

島ソンはハッカソンが終わったあともプロジェクトが続き、成果を上げています。

①「アカモク」

「浦戸諸島で採れる海藻アカモクをPRしたい」という要望に対して行動し、「渚の妖精ぎばさちゃん（ギバサはアカモクの別名）」というキャラクターが生まれました。現在もTwitterを中心に活躍中で（アカウントは@gibasachan）、人気もぐんぐん上昇中です。

本連載第41回（2015年5月号）では、まるまるぎばさちゃんの紹介記事も書きました。新しくパッケージ商品も作られます（イラスト1）。

◆イラスト1　ぎばさちゃん商品のパッケージイラスト

②「謎解きゲーム」

「観光資源はあるのに訪れる人が少ない」という課題を受け、リアル脱出ゲームの人気にあやかって謎解きゲームを開催して、若い人に浦戸諸島を知ってもらうきっかけイベントを開催するというプロジェクト。2015年12月に開催され、学生たちが多く参加しました。

③「Island Girls」

浦戸諸島について知らない人が多く、まずは興味を持ってもらうことから始めるために聖地化するというアイデアを提案。浦戸諸島をモチーフにしたIsland Girlsというアドベンチャーゲーム、つまりギャルゲーを作ろうというもの。ほかのハッカソンに舞台を移しながら開発が継続中です。

そして島ソン2015

島ソン2015は2015年10月31日に開催、場所は前回同様、野々島にある合宿施設であるブルーセンターです。前回はCode for Shiogama[注1]とHack for Japanの共同での開催でしたが、今回は復興庁の「新しい東北」先導モデル事業である「塩竈アイランズネットワーク協議会」の「浦戸 サスティナビリティ プロジェクト」の一環として行われました。

昨年の島ソンは泊りがけのハッカソンでしたが、今回は日帰りのアイデアソンです。泊りがけになると先述のように食事の用意も全部自前で行わなければならず、ネットワークの問題は相変わらず解決していないのでプログラミングもハードルが高い。そこで今回はアイデアソンで行うことにし、イベント自体も日帰りとしました。今回も石巻専修大学の舛井研究室とタッグを組んでいます。

また、テーマも「島の課題を解決する」から「島をより楽しむためのしくみを考える」にしました。第1回の島ソンに限らず、ほかのハッカソンでもよく見られる現象に「一発受けネタのほうが盛り上がってしまう」というものがあります。チームを組むときも受けが狙えるほうに人が流れてしまう。受け狙いを避けるように進行するという手段もありますが、参加者に学生も多いため、よりイベントを楽しめるように受け狙いになっても構わないテーマ設定にしました。

島ソン2015の流れ

昨年と違いアイデアソンのみなので流れは変わっています。しかし、一番の肝となる「島あるき」は健在です。そのあとにアイデアワークを行い、チーム分け、そして発表の流れです。今回の参加者は19歳から65歳までと非常に幅広い参加者なので、そこで起きる化学反応にも期待しました。

島あるきへ！

浦戸諸島には桂島（かつらしま）、寒風沢島（さぶさわじま）、会場であるブルーセンターのある野々島（ののしま）、そして朴島（ほおじま）という4つの島で構成されています。今回は前回よりも拡大し、ブルーセンターのある野々島だけではなく、渡船を用いて浦戸諸島の中心地区である桂島、そして浦戸諸島最大の面積を誇る寒風沢島も「島あるき」をして島の様子を見て回りました。

●野々島

会場のある野々島です。野々島はなんと島内に100以上もの洞穴があるという特徴があります。そのほとんどが人為的に掘られたもので、島では「ボラ」と呼ばれています。**写真1**を見てもわかるように、きれいな長方形に掘られていて、天然のものでないことがわかります。ここに漁具や農作業具を入

◆写真1　野々島のボラ

注1　Code for Shiogamaは東日本大震災で被災した宮城県塩竈市の地域活性をITの力で行っていこうという団体です。震災復興団体とIT技術者がお互いに意見を出し合うミーティングの開催をしています。Facebookページ：**https://www.facebook.com/CodeForShiogama**

れたりと物置的な使われ方をしています。

●寒風沢島

寒風沢島は塩竈市で唯一水田のある土地です。天水を利用した伝統の方法で稲作していて、塩竈が全国に誇る酒蔵「浦霞」から「寒風沢」という日本酒も出ています。

●桂島

桂島は浦戸諸島の中で最も人口が多く、設備も整っている島です。塩竈本土との窓口にもなっています。松島の象徴的な島に、島の上に岩がアンバランスに乗っている仁王島（におうじま）があります。実はこの島は桂島の海水浴場のすぐそばにあるんです。震災で閉鎖されていた海水浴場も、現在は夏に開かれるようになりました。仁王島も間近に見られます。しかし一部は震災当時のまま残されていて、いまだに震災の爪痕が深いことがわかります（**写真2**）。

アイデアソン！

島あるきから戻ったあとは島の幸たっぷりのお弁当を食べてアイデアソンへ。

アイデアソンではまず島あるきをしてきて印象に残ったところを箇条書きしてもらい、次にそこをより楽しむにはどうするかを一言で良いので書き込んでもらいます。

このあとがアイデアソンの肝となるスピードストーミングです。スピードストーミングは**写真3**のように対面し、持ち時間3分で対面しあった2人でのブレインストーミングを行います。持ち時間が終わったら片方の列が1人分ずれて、また別の人と2人でブレインストーミング。この手法の良いところは通常のブレストと異なり、しゃべらないで終わってしまう人が出ないことです。人によって見ているところが異なるので、このスピードストーミングでお互いの情報を交換しながらアイデアを膨らませます。

スピードストーミングが数セット終わったら、次は自分なりのアイデアをまとめてもらう「アイデアスケッチ」と呼ばれる工程に進みます。ここで各人最低1つは自分のアイデアをまとめます。このアイデアスケッチをグルーピングし、ここでチーム分けを行います。

チームに別れてからはチーム内でディスカッションし、その内容を模造紙にまとめて発表となります。

◆**写真2　桂島海水浴場の震災跡**

◆**写真3　スピードストーミング**

そして成果発表へ

今回のチームのテーマと発表内容を紹介します。

①「島の幸の体験ツアー」

コンセプトは「育てる×食べる」（**写真4**）。海苔を作る、牡蠣を育てる、といった島での農作業を通じて島を体験し、最後は自分で育てたものを食べるというツアー。一度の来訪ではなく複数回にわたって来てもらうことを狙ったアイデア。

②「3D Four Islands」

浦戸諸島4島の3Dモデルを作り、Unityを用いた一人称探索ゲームにするというもの（**写真5**）。ゲームを遊ぶことで島を知ってもらおうというアイデア。そこから島のPRへつなげていく。

③「MOVIX浦戸」

ものすごいネーミングでドキドキしますが、やる内容はまっとうです。島のお母さんたちの料理教室や島の見どころ紹介を動画にし、気になった人向けに情報を提供して実際に足を運んでもらおうというもの。そして、島に来た人たちを対象としたイベントを行い、島に活気を呼ぼうというアイデア（**写真6**）。

④「ボラフェス」

野々島に多数存在している「ほら穴」に着眼。なんと洞穴でライブを行おうというアイデア（**写真7**）。洞穴という環境がライブに適しているのではないかというところから膨らませたとのことです。

以上、4つのアイデアを発表して島ソンは終了となりました。

イベントを終えて

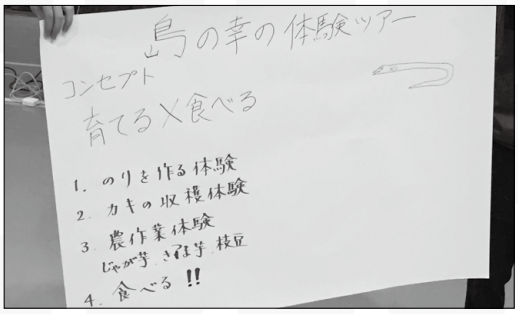
前回は次の言葉で締めていました。「こうやって地方でハッカソンなどイベントを起こしても、イベントが終わったら、それで終わりというケースが多いです。ここでハッカソンをしたことにもっと意味を持たせるためにも、今回ここで出たアイデアや、作ったものを、これからも継続していきましょう」。実際にイベント終了後も継続していくプロジェクトが複数生まれました。

対して今回の島ソンですが、アイデアソンに絞ったのもあるのかもしれませんが、実現可能性が昨年の島ソンに比べてちょっと難しいものが多いかなという印象です。これからの学生の皆さんの継続に期待したいところではありますが、人を巻き込まなければ動かないものが多いので、そう簡単には進まなさそうです。これからの奮起に期待します。

最後に、今回のアイデアソンでも多くの方々に協力をいただきました。島の方や参加いただいた方はもちろんのこと、島の案内をしてくださったe-frontの國吉さんに太田さん、そして塩竈アイランズネットワーク協議会のみなさん。本当にありがとうございました。

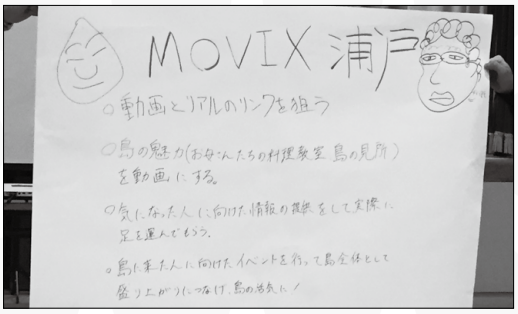
Code for Shiogamaでは復興に携わっている方とIT技術者が議論することで生まれる化学反応を起こしたいと思っています。今後も定期的に開催していきますので、よろしくお願いいたします。SD

◆写真4　島の幸の体験ツアー

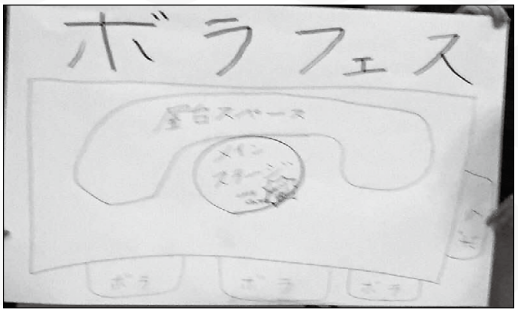


◆写真5　3D Four Islands

◆写真6　MOVIX浦戸



◆写真7　ボラフェス



温故知新
IT むかしばなし

第52回

BASIC〜ベーシックなプログラミング言語〜

速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami

はじめに

マイコンボードIchigoJam[注1]や、任天堂3DSで動作するプチコン[注2]など、ふたたびベーシックなプログラミング言語であるBASICが注目を集めています。今回は、その「BASIC」についてお話をしましょう。

プログラミング言語BASICとは

東京でオリンピックが開かれた1964年、ダートマス大学でBASICが誕生しました。当時の限られたコンピュータの環境における、プログラミング教育は長時間で非効率に行われていました。1台のコンピュータを複数のユーザで共用(タイムシェアリング)した端末で、命令を入力したら、その端末画面で即時実行結果が表示されるBASICのプログラミング環境は、効率の良いプログラミング教育の環境を提供したのです。

10年後の1974年インテルから8bitCPU i8080が発売され、次の年にこのCPUを搭載したマイコンであるAltair 8800が登場しました。ビル・ゲイツとポール・アレンは、このマイコンの魅力に虜(とりこ)となり、Micro-Soft BASIC[注3]を開発しました。

非力なマイコンにおいて、BASICは対話型言語として適しており、1977年に登場したPET 2001、APPLE II(ともにCPU 6502)、TRS-80(CPU Z80)はすべてBASICが搭載されていました。これらマイコンはすべてMicro-Soft BASICも動作が可能であり、Micro-Soft BASICがマイコンの標準のプログラミング言語になっていきました。

日本においては、1976年登場のCPU μPD8080を搭載したマイコンボードNEC TK-80とベーシックステーションボードTK-80BSを組み合わせてBASICを動かすことから始まり、1978年にはシャープMZ-80K、日立ベーシックマスターで独自BASIC搭載のマイコンが発売されました。

1979年に登場したNEC PC-8001にMicro-Soft BASICを元とするN-BASICが搭載され、この後登場するマイコンは、Micro-Soft BASICを基本とするものが主流になっていきました。

1981年には、PC-8001を高機能化したPC-8801が登場し、そこに搭載されたのがN88-BASICです。N88-BASICは、N-BASICとの互換性を保ちながら、グラフィックや漢字処理などの高度な命令を持ち、16bitパソコンPC-9801にも16bit版のN88-BASIC(86)が搭載され、利用度が高い標準的なBASICとしての地位を築きました。

N88-BASICの特徴と問題点

以下に標準BASICとして長く利用されているN88-BASICの特徴と問題点を挙げてみます。

①プログラムは行の先頭に行番号が必要である

先頭に数字を入力するとこれが行番号になり、BASICプログラムソースの入力になります。RUNコマンドでプログラムが行番号順に実行されます。プログラムの実行の移動制御は、基本的に行番号を指すGOTO/GOSUB文になり、見通しの悪いプログラムになってしまいます。

注1) こどもパソコンのコンセプトのもとに開発された、BASIC言語でI/Oポート制御ができるワンボードパソコン。

注2) ニンテンドーDSi/3DS用ソフトウェアで、BASIC言語でゲームなどのプログラミングを楽しむことができる。

注3) 当時はMicrosoftは、Micro-Softと"ー"が入っていました。https://en.wikipedia.org/wiki/Microsoft

②変数の宣言が必要ない

配列変数以外の変数の宣言は必要がありません。使用頻度が高い変数は早めに実行しておくことや配列変数の宣言の前に変数を一度使っておくことなどの実行スピードアップのテクニックが存在します。

③＝が代入と比較と両方で使用される

C言語では、代入は「=」、比較は「==」ですが、BASICでは双方とも「=」であるため、BASICでプログラムを組んでいる時点でのバグは少ないのですが、C言語でプログラムを組む段になると、その癖を払拭できずバグが多発します。また、初心者に対して「K=K+1」などの代入構文の説明が難しく、代入の「=」と数学の「=」との違いを理解するための壁があります。

④IF THEN ELSEは、マルチステートメントを使い1行に記述しなくてはならない

THENの後ろが、複数の命令を並べるマルチステートメントになることが多くあり、ELSE文まであると1行が非常に長くなり、可読性の低いプログラムソースになってしまいます。

⑤引数を伴うサブルーチン呼び出しができず、ローカル変数も使えない

サブルーチンは、GOSUB行番号で呼び出し、戻りはRETURN文となり、引数も戻り値も存在しません。ローカル変数の概念がなく、すべての変数がグローバル変数であるため、再帰呼び出しのプログラミングを行うためには、特別なプログラミングテクニックが必要になります。

⑥インタプリタのため実行スピードが遅い

対話型の即時処理用言語を簡易に実現するために、インタプリタであり、当時のマイコンの処理スピードの低さと相まって、実行速度は非常に遅く、さまざまな実行速度を高めるプログラミングテクニックが駆使されました(**リスト1**)。

Quick BASICとTurbo Basic

1985年Microsoft社からQuick BASIC、同じ年にBorland社からTurbo Basicが発売されました。2つのBASICは、行番号が不要で構造化プログラミングが可能なネイティブなコンパイラ型のBASICでした。前述のBASIC問題点をほぼ解決しており、テキストベースの総合開発環境(IDE)上で優れた開発環境が用意されています。コンパイラが基本ですが、デバッグ時はインタプリタとして動作し、対話型の即時処理用言語としてのBASICの機能も実現しています。

Quick BASICは、1988年のVersion 4.5を最後に、Windows環境で動作して、優れたGUIプログラミングを実現したVisual BASICに成長していきます。Turbo Basicは、IBM-PCのみで、Borland社からの販売は終わってしまいましたが、その後Power BASIC[注4]としてよみがえり、現在も販売が行われています。

おわりに

BASICは、1980年代前半の性能の低いマイコン上で動作する言語として、そのエディタと一体化した開発環境が適していたこともありました。しかし、現在の高速／高機能なPC環境では、ほかに多くのプログラミング教育に適した言語が存在します。BASICが復活して利用されることは懐かしくうれしく思いますが、子どもたちに最初に使ってもらう言語として、そのあとの学習発展を考えると、問題があるかもしれません。SD

注4) GUIを使用してアプリケーションを作成できるWindows用のネイティブコードコンパイラ。http://www.powerbasic.com/products/

▼リスト1　BASICで再帰呼び出しを実現したハノイの塔のプログラム

```
10 'ハノイの塔
20 DIM N(4),A(4),B(4)
30 SP=0
40 N(1)=4:A(1)=1:B(1)=3
50 GOSUB 100
60 END
100 SP=SP+1
110 IF N(SP)>1 THEN N(SP+1)=N(SP)-1:A(SP+1)=A(SP):B(SP+1)=
6-A(SP)-B(SP):GOSUB 100
120 PRINT "円盤";N(SP);"を";A(SP);"から";B(SP);"へ移動する"
130 IF N(SP)>1 THEN N(SP+1)=N(SP)-1:A(SP+1)=6-A(SP)-B(SP):
B(SP+1)=B(SP):GOSUB 100
140 SP=SP-1
150 RETURN
```

Service

さくらインターネット、「さくらのVPSベアメタルプラン」提供開始

さくらインターネット㈱は2月3日、「さくらのVPS」の新プランとして、安定性に優れた物理サーバをまるごと1台専有できる「さくらのVPSベアメタルプラン」の提供を開始した。

さくらのVPSは月額685円から使える手軽な仮想専用サーバサービス。今回追加されるベアメタルプランでは、シンプルかつ気軽に使えるVPSライクな物理サーバサービスをコンセプトとしており、さくらのVPSのコントロールパネルでのサーバの一元管理や、サーバ間のローカルネットワーク接続に対応しており、仮想と物理のそれぞれのメリットを活かしたハイブリッド構成を容易に実現できる。なお本プランには、2週間の無料お試し期間が用意されている。

●プラン仕様（さくらのVPSベアメタル8Gプランの場合）

項目	詳細
月額料	7,776円（税込み）
初期費用	486,00円（税込み）
メモリ	8GB
ストレージ	SSD（RAID1）111GB
CPU	物理2Core
ネームサーバ	10ゾーン
ゾーン	東京

CONTACT
さくらインターネット㈱ URL http://www.sakura.ad.jp

Service

ソラコム、4つの新サービスを発表

IoTプラットフォーム「SORACOM」を提供する㈱ソラコムは1月27日、「Air」「Beam」に続く新サービス、「Canal」「Direct」「Endorse」「Funnel」を発表した。

- **SORACOM Canal**
  Amazon Web Services上に構築したユーザの仮想プライベートクラウド環境（Amazon VPC）とSORACOMを直接接続するプライベート接続サービス。
- **SORACOM Direct**
  SORACOMとユーザのシステム（IoTバックエンド）を専用線で接続する専用線接続サービス。
- **SORACOM Endorse**
  SORACOM AirのSIMの情報を、外部のサービスにおいてユーザ認証やデバイス認証などのために利用できるサービス。
- **SORACOM Funnel**
  デバイスからのデータを特定のクラウドサービス（現状は、AWS Kinesis Streams、AWS Kinesis Firehose、Microsoft Azure Event Hubsの3つ）に直接転送するデータ転送サービス。

CONTACT
㈱ソラコム URL https://soracom.jp

Software

ビーブレイクシステムズ、ERPパッケージ「MA-EYES」が支払調書へのマイナンバー対応を実施

㈱ビーブレイクシステムズは2015年12月25日、統合型基幹業務パッケージ製品「MA-EYES」において、マイナンバー制度対応を実施したことを発表した。

同製品は、IT企業のようなプロジェクト型企業の業務全般をトータルにカバーするERPパッケージ製品。プロジェクト管理、販売管理、在庫管理、購買／経費管理、作業実績管理、分析／レポート、ワークフローなどの機能を標準で搭載する。

利用形態としては、ユーザの環境にインストールして利用する一括導入版と、クラウド環境上で必要な機能を月額で利用するSaaS版とが用意されている。

MA-EYESの原価管理機能では、個人事業主に対する発注や支払を管理でき、源泉徴収した債務の一覧の確認や、支払調書の出力を行える。

今回のマイナンバー制度の開始に伴い、源泉所得税申告時に必要な支払調書の様式（ひな形）が変更となった。新様式では法人番号や個人番号などのマイナンバーを記載する欄を新たに設ける必要がある。これに対応するため、MA-EYESの支払調書も自社の法人番号を支払者欄に自動で出力できるように対応した。

CONTACT
㈱ビーブレイクシステムズ URL http://www.bbreak.co.jp

Topic

LPI-Japan、「LPI-Japan OPCELアカデミック認定校」第一号を発表

特定非営利活動法人エルピーアイジャパン（以下LPI-Japan）は1月19日、NECマネジメントパートナー㈱、㈱アドックインターナショナル、アセアン・ラボ㈱の3社を、「LPI-Japan OPCELアカデミック認定校」第一号に認定したことを発表した。

本認定校制度は、LPI-Japanが定めたOpenStackの学習環境基準をクリアした教育機関をOPCEL認定校に指定し、資格の取得を目指す受験者に質の高い教育を提供する制度。各社のOpenStackへの取り組みは次のとおり。

・NEC マネジメントパートナー

「NEC Cloud IaaS」の実現において、OpenStackを活用。また2015年10月には、「NEC Cloud System（OSS構築モデル）」を提供開始した。

・アドックインターナショナル

5G携帯電話サービスのリリースに向け、社内でOpen Stack エンジニアの育成に努めている。

・アセアン・ラボ

コミュニティの立ち上げ段階からOpenStackを支援しており、日本で最初にOpenStack研修事業を立ち上げた。

CONTACT
特定非営利活動法人エルピーアイジャパン URL http://lpi.or.jp

Report

UEI、「enchantMOON Crew Meeting 2016」開催 ～終了宣言とCode name「Discovery」～

1月30日、㈱UEI（旧社名：㈱ユビキタスエンターテイメント）によるenchantMOONユーザとプログラミング教育を考える人々のためのミートアップイベント「enchantMOON Crew Meeting 2016」が開催された。

同社代表取締役社長兼CEOの清水亮氏による講演のほか、ゲストによるライトニングトークや対談が行われ、各講演者の技術領域やプログラミング教育にenchantMOON／MOONBlockを結びつけて未来が語られた。

enchantMOON S-IIが2015年内で販売終了となり次の製品に対する情報が期待されたが、enchantMOONについては事実上の終了宣言が行われた。代わりに新たな製品プロジェクト、コードネーム「Discovery」が2017年にリリースされることが発表され、来場者にモニターへの参加を募った。

ここからは記者の推測に過ぎないが、enchantMOONの後継機種、S-IVBシークエンスと呼ばれていたものの情報をいち早く知りたいと参加した来場者にとって、今回のイベントは肩すかしをくらった感じではなかっただろうか。講演内容がつまらなかった、ということではなく「次のenchantMOONを知りたい」と思っていたニーズと合っていたかという話だ。

VAIOやプレイステーションシリーズのデザインを手がけた後藤禎祐氏による外観デザインやブランド名に対する考え方などは、なかなか聴けない貴重な話だったし、NVIDIAの橋本和幸氏によるVRとペンインターフェース、アスラテックの今井大介氏によるロボットと人工知能、蒲地輝尚氏によるHyperCardとAIプログラミング、PEZY Computingの齊藤元章氏によるスパコンとブレインマシンインターフェースの話などは、いまの技術と未来をつなぐ興味深い視点だったと思う。そうではあるが、enchantMOONとのつながりとなると「こじつけ」感は否めない気がしていた。

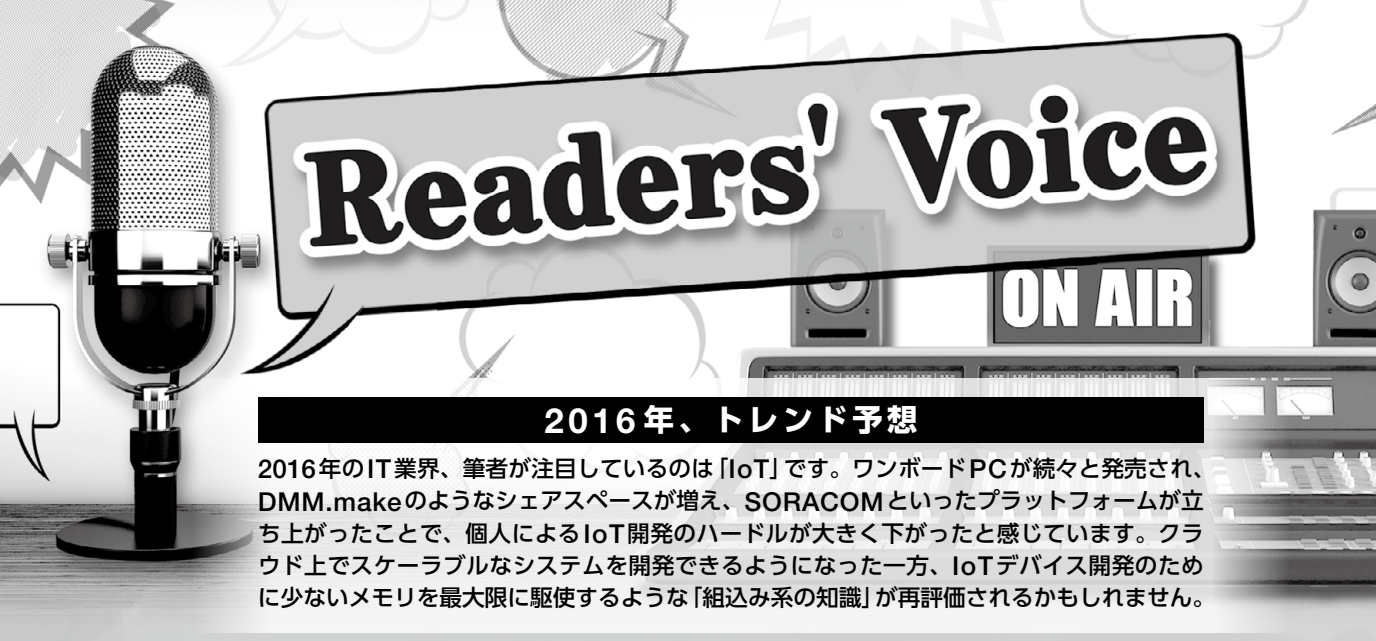
だがしかし。最後の清水氏の話を聞いて、その認識は甘かったのではと思うにいたった。

休日返上で集まったユーザにとって、清水氏から告げられた「enchantMOONという名の製品はもう作りません」という言葉は衝撃だったに違いない。しかし清水氏はもっと先を見ていて、たとえば後藤氏の基調講演は、次に出す製品はVAIOやプレイステーションのようにたくさん売れるものを作るという意思の表れであり、そのために製品名を含めた製品戦略を練り直すということと受け取れる。

また、来場者をいちばんどよめかせたであろう、ペンデバイスならではのチラシ制作を見せたデモ映像では、手書き文字認識はもちろん、文脈から推測した意味補完、太さや配置から推測したフォント選択、類似画像検索といったことが行われていた。これらはまさに講演で話されていたディープラーニングや人工知能（AI）といった領域の技術であり、次に出す製品に組み込まれることを予感させてくれる。

「次のenchantMOON」ではなかったが、enchantMOONに続く同社の挑戦が、2017年に製品となって登場することに期待したい。

CONTACT
㈱UEI URL http://www.uei.co.jp



2016年、トレンド予想

2016年のIT業界、筆者が注目しているのは「IoT」です。ワンボードPCが続々と発売され、DMM.makeのようなシェアスペースが増え、SORACOMといったプラットフォームが立ち上がったことで、個人によるIoT開発のハードルが大きく下がったと感じています。クラウド上でスケーラブルなシステムを開発できるようになった一方、IoTデバイス開発のために少ないメモリを最大限に駆使するような「組込み系の知識」が再評価されるかもしれません。

2016年1月号について、たくさんの声が届きました。

第1特集 はじまっています。ChatOps

SlackやHipChatといったチャットツール上に、Hubotで作ったbotを常駐させ、デプロイをはじめとした作業を肩代わりさせる新しい開発手法「ChatOps」。実際にChatOpsを導入した7つの組織に、導入によって開発がどのように変わったのかを伺いました。

IRCはセキュリティの面で心配だったので、ChatOpsで使われるようなツールはどのくらい保障されているか気になります。 **Tayuさん／千葉県**

社内で使われているツールとしてメール、ハングアウト、Skype、Slackと乱立しているので、どうにかしたいなと考えていました。この特集を参考にして統一したいです。 **binaさん／東京都**

初めて聞く単語が多く、勉強になりました！ **南雲さん／埼玉県**

とてもおもしろい特集でした。弊社でも社内用の簡単なチャットツールは導入されていますが、利用されていないのが実情なので、今回の記事を参考にしたいと思います。 **山本なほさん／神奈川県**

フォローとストックが効率的に行える方法が、いろいろと参考になりました。 **出玉のタマさん／大阪府**

最近、ChatOpsを導入しつつありますが、確かにメールよりもリアルタイムで通知を確認でき、さらにチームで同じツールでチャットをしながら対応できるのでとても便利なのですが、その反面チャットツールがずっと気になってしまい、かなりの割り込みとなってしまうので、どうしたらいいものかと考えている昨今です。 **n0tsさん／東京都**

特集内でも、また読者からのお便りでも「どのように導入するか」以上に「いかにメンバーに継続的に使ってもらうか」が重要な点のようです。チャットルームを細かく分けたり、botにかわいいキャラクターを設定したりと、どの組織でもさまざまな工夫がなされていることがわかりました。

第2特集 Ansibleでサーバ構成管理を省力化

サーバの構成をコードで管理するというInfrastructure as Codeを実現できるツール「Ansible」の特集です。Ansibleを使うメリットから、導入方法、Inventory・Playbookの書き方、Windowsへの対応まで解説しました。

別の構成管理ツールと比較するうえで参考になった。 **米坂さん／神奈川県**

引き継ぎのことなどを考えると、サーバ構築の自動化には興味があるので、とてもおもしろかった。Dockerfileとどちらが良いのか悩む。 **tack41さん／愛知県**

構成管理ツールについてはあまり考えたことがなかったので、自分が使っている部署の構成管理ツールについて興味を持ちだした。 **ももんがさん／静岡県**

Red Hatの買収によりどうなるか、注目しています。 **山下さん／東京都**

Ansibleでサーバ構成管理を自動化、省力化するのは魅力的です。人間が手動で行うと間違う恐れがあるので、自動化できるものは自動化していくべきだと思います。 **永作さん／東京都**

構成管理にAnsibleを使い始めました。Ansibleは敷居は低いのですが、複雑なことをしようとすると混み入ったことをしないといけない印象を受けていたので、本特集が具体的でとても参考

1月号のプレゼント当選者は、次の皆さまです

①7インチHDMIマルチモニター「LCD-7000VH」
川野邦仁様（大阪府）

②世界最小USBハブ「USB2-HUBMC2SS」
尾崎潤二様（東京都）、陰山善行様（神奈川県）

③Acronis True Image Cloud
高橋良司様（愛媛県）、モモンガ様（滋賀県）、牧秀亮様（大阪府）

④『The Art of Computer Programing Volume 3』
渡辺啓太様（埼玉県）、中村泰大様（茨城県）

⑤『Apache Spark入門』
コメット様（兵庫県）、山崎秀峰様（東京都）

⑥『ITエンジニアのための機械学習理論入門』
中村洋様（大阪府）、瓜生聖様（東京都）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

になりました。 犬棟梁さん／埼玉県

手軽さ、シンプルさが売りのAnsible。興味を持った読者も多いようです。同じく構成管理ツールであるChefにはRubyの知識が要るので、言語を選ばないAnsibleを選ぶ人が多いかもしれませんね。

短期連載　クラウド時代のWebサービス負荷試験再入門【2】

ITインフラの中心がオンプレミスからクラウドへ移るに伴って、システムに対する負荷試験も、それに合わせたものを用意する必要があります。本連載ではクラウドに載せたWebサービスの「スケーラブル」を担保するための負荷試験について見ていきます。第2回では、負荷試験を行うためのツールを中心に解説していきました。

動画データ通信のプロトコル開発を担当する自分には、負荷試験は興味津々です。 Whiskyさん／宮崎県

いろいろな負荷をかけるソフトウェアがあると知れたのが良かった。 那須さん／東京都

Webの負荷試験はオンプレミスでもクラウドでも必要なので、参考になった。 ほまれさん／千葉県

ほとんどの場合、テスト機器側、負荷をかける側がボトルネックになり苦労します。それをふまえての説明や分析が良かった。 atachibanaさん／東京都

アプリケーション開発で必要なノウハウな割に、あまり経験する機会がないので、この機に勉強しようと思います。 鐘ヶ江さん／東京都

負荷試験のためのツールについての整理された情報が役に立った、という声がいくつか寄せられました。Webのコンテンツはますますリッチに、スマホの普及によってユーザはますます増えている今、負荷試験の重要性はさらに高まっていますね。

短期連載　SMB実装をめぐる冒険【3】

Windows共有フォルダをChromeOSのファイルアプリにマウントするアプリ。その開発には「SMBプロトコル」の理解とJavaScriptでの実装が必要でした。連載第3回では、接続先の「共有リソース」の扱いについて見ていきます。

若干、難易度が高いように感じましたが、その分読み応えがあり勉強になりました。 ＹＹさん／神奈川県

やっぱりパケットキャプチャが必要なんだ……。 **まだ40MBのHDD持っているさん／奈良県**

この連載では情報の少ないSMBプロトコルの中身が紹介されているので、とても勉強になります。 スマイルさん／茨城県

前回に引き続き、開発の苦労に共感するような声が多く寄せられました。今でこそOSSが主流になりつつあるIT業界ですが、プロプライエタリソフトウェアが当たり前だった時代は、連載のような、既存ソフトやプロトコルの解析・再開発が多く行われていたのでしょうか。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
April 2016

2016年4月号
定価（本体1,220円+税）
192ページ

3月18日発売

［第1特集］やればできる!ワンランク上のプログラミング

今すぐ実践できる良いプログラムの書き方
～きれいなコード／モダンなコードが書きたい～
［C、Java、C#、Ruby、JavaScript］

良いプログラムとは、「可読性が良い」「処理効率が良い」「その言語の慣習に沿っている」「セキュリティのことを意識している」などいろいろあります。ですが、そんなコードを書くためにどうすればいいのかは、書籍やマニュアルにはなかなか書かれていないもの。そこで、5つの言語のプロフェッショナルに良いコードを書くためのポイントを伝授してもらいます。スペシャリストは何に注意してコードを書いているのか、その観点を取り入れて、脱初心者、脱レガシープログラマをめざしましょう。

［第2特集］基礎の基礎と使いどころ完全解説

オブジェクトストレージの教科書

［特別企画1］LANケーブリングの次はこれだ!

サーバラッキング+配線の教科書

［特別企画2］日本で運用と開発はコラボできるか?

DevOps座談会

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

SD Staff Room

●3歳の娘に振り回される日々。最近は何か計画を企み、それを押し通す知恵を付けてきて驚愕。あの小さい頭で何かを考えているのだと思うと不思議。お菓子やオモチャを得るために祖父母やママの注意を惹き、パパをだまし、伏線を張ったり、ジャストなタイミングでだだをこねたり……あ、これって女性の原点か!（本）

●医療・健康系の番組が多い。年配者が増え、身体に気を遣うことが多いからと推察するが、健康の分野こそ、もっとIoTや最新センサーを利用した各家庭や自分自身でのモニタリング技術が増えてもいいんじゃないかと思う。体温や鼓動などだけでもビッグデータ化して利用できないものか。（自己計測オタクの幕）

●駅で、ふと目に入ったポスターには「超獣バキシム」が。スタンプラリーらしい。ウルトラマンにあんまり詳しくない私が覚えているほど、子ども心にカッコイイと思った怪獣。懐かしい顔に出会えたような気分になって、会社の最寄り、市ヶ谷駅もチェック。「ヤメタランス」……orz チカラヌケタ（キ）

●4月号の特集「良いプログラムの書き方」は、5種類の言語について5人の著者に書いてもらう予定です。各著者が考える「良いプログラム」の基準に共感したり、言語ごとに機能／書きやすさ／得意とする用途などの違いが見えたりと、いろいろ楽しめます。まるでプログラミング言語の試食会です。（よし）

●最近、する予定も予算もない引っ越しのことをずっと考えています。家賃が抑えめで、映画館のある街に住みたいなと思っているので、立川や亀有など、実際に下見にも行ってきました。どの街も、今住んでいる所と比べると妙にキラキラして見えるんですよね。隣の芝生はなんとやら……でしょうか。（な）

●「VISCUIT（ビスケット）」で初プログラミングを体験した息子。自分で描いた絵を動かしながら作れるので楽しくプログラミングの世界に入り込めたようです。母としてはできあがりを楽しみにしていたのですが、残念ながら栄えある第一号は、保存する前に終了ボタンを押したために幻に。なんてお約束的なの!（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6173

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2016年3月号

発行日
2016年3月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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