Special Feature 1 →数学入門
Special Feature 2 →GPUの入門と導入法

Software Design

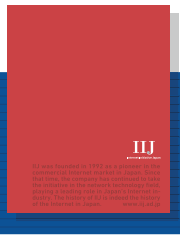
［ソフトウェア デザイン］
OSとネットワーク、
IT環境を支える
エンジニアの総合誌

December 2017

12

2017年12月18日発行
毎月1回18日発行
通巻392号
（発刊326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

特別付録
IIJ謹製
「インターネット便利帳」



さあ始めよう！

ITエンジニアと数学

Special 1 Feature

数式を見て直感的にコーディングしたい！

数学プログラミング入門

Special 2 Feature

優れた演算能力とパフォーマンス

気になるGPU超入門

導入法と基礎から
CUDAを使った事例まで

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。

目次

Contents

Software Design

December, 2017

12

Special Feature 1 → 第1特集 | Page

17

さあ始めよう!

ITエンジニアと数学

数学プログラミング入門

目次

Contents

Software Design
December, 2017
12

Column

Development

OS/Network

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

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

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

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

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

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

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

Software Design plus

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

最新刊！

田中洋一郎 著
A5判・360ページ
定価 2,800円（本体）+税
ISBN 978-4-7741-9332-8

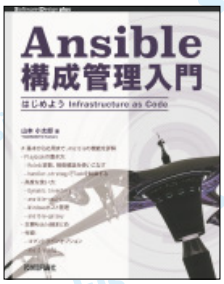
すずきひろのぶ 著
A5判・416ページ
定価 2,600円（本体）+税
ISBN 978-4-7741-9322-9

仲川樽八、森下健 著
B5変形判・368ページ
定価 3,800円（本体）+税
ISBN 978-4-7741-9262-8

香山哲司、小野寺匠 著
A5判・176ページ
定価 2,480円（本体）+税
ISBN 978-4-7741-8815-7

星野武史 著、花井志生 監修
A5判・256ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-8729-7

小川晃通 著
A5判・272ページ
定価 2,280円（本体）+税
ISBN 978-4-7741-8570-5



山本小太郎 著
B5変形判・176ページ
定価 2,480円（本体）+税
ISBN 978-4-7741-8885-0

中井悦司 著
B5変形判・272ページ
定価 2,980円（本体）+税
ISBN 978-4-7741-8426-5

前橋和弥 著
B5変形判・304ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-8188-2

五味弘 著
B5変形判・272ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-8035-9

養成読本編集部 編
B5判・160ページ
定価2,180円（本体）+税
ISBN 978-4-7741-8895-9

養成読本編集部 編
B5判・240ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-8877-5

養成読本編集部 編
B5判・168ページ
定価1,980円（本体）+税
ISBN 978-4-7741-8360-2

養成読本編集部 編
B5判・232ページ
定価2,080円（本体）+税
ISBN 978-4-7741-8385-5

データサイエンティスト養成読本 機械学習入門編
養成読本編集部 編
定価 2,280円+税　ISBN 978-4-7741-7631-4

C#エンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7607-9

Dockerエキスパート養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7441-9

サーバ／インフラエンジニア養成読本 基礎スキル編
福田和宏、中村文則、竹本浩、木本裕紀 著
定価 1,980円+税　ISBN 978-4-7741-7345-0

Laravelエキスパート養成読本
川瀬裕久、古川文生、松尾大、竹澤有貴、小山哲志、新原雅司 著
定価 1,980円+税　ISBN 978-4-7741-7313-9

Pythonエンジニア養成読本
鈴木たかのり、清原弘貴、嶋田健志、池内孝啓、関根裕紀、若山史郎 著
定価 1,980円+税　ISBN 978-4-7741-7320-7

データサイエンティスト養成読本 R活用編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7057-2

Javaエンジニア養成読本
きしだなおき、のざきひろふみ、吉田真也、菊田洋一、渡辺修司、伊賀敏樹 著
定価 1,980円+税　ISBN 978-4-7741-6931-6

JavaScriptエンジニア養成読本
吾郷協、山田順久、竹馬光太郎、智大二郎 著
定価 1,980円+税　ISBN 978-4-7741-6797-8

WordPress プロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6787-9

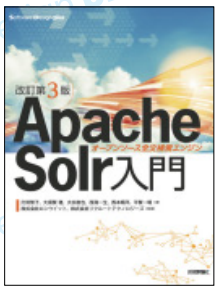
内部構造から学ぶPostgreSQL 設計・運用計画の鉄則
勝俣智成、佐伯昌樹、原田登志 著
定価 3,300円+税　ISBN 978-4-7741-6709-1

サーバ／インフラエンジニア養成読本 ログ収集～可視化編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6983-5

フロントエンドエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6578-3

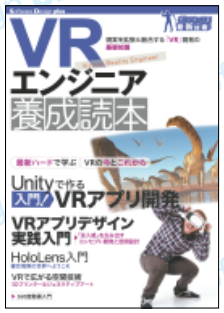
PHPライブラリ&サンプル実践活用［厳選100］
WINGSプロジェクト 著
定価 2,480円+税　ISBN 978-4-7741-6566-0

常田秀明、水津幸太、大島騎頼 著、Bluemix User Group 監修
B5変形判・288ページ
定価 2,800円(本体)+税
ISBN 978-4-7741-9084-6



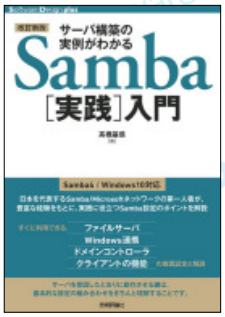
打田智子、大須賀稔、大杉直也、西潟一生、西本順平、平賀一昭 著
B5変形判・392ページ
定価 3,800円(本体)+税
ISBN 978-4-7741-8930-7

養成読本編集部 編
B5判・144ページ
定価1,780円(本体)+税
ISBN 978-4-7741-8865-2



養成読本編集部 編
B5判・112ページ
定価2,180円(本体)+税
ISBN 978-4-7741-8894-2

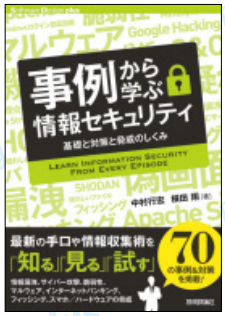
養成読本編集部 編
B5判・192ページ
定価1,980円(本体)+税
ISBN 978-4-7741-8863-8



髙橋基信 著
A5判・256ページ
定価 2,680円(本体)+税
ISBN 978-4-7741-8000-7

高宮安仁、鈴木一哉、松井暢之、村木暢哉、山崎泰宏 著
A5判・352ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-7983-4

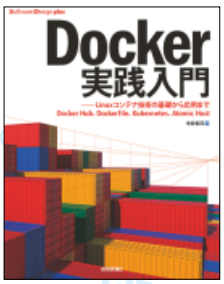
斎藤祐一郎 著
A5判・160ページ
定価 2,280円(本体)+税
ISBN 978-4-7741-7865-3



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

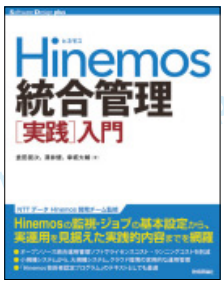
神原健一 著
B5変形判・192ページ
定価 2,580円(本体)+税
ISBN 978-4-7741-7749-6



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

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

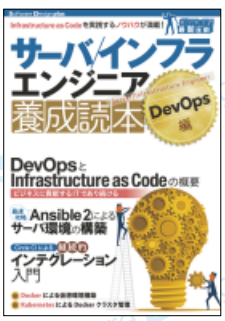
中島雅弘、富永浩之、
國信真吾、花川直己 著
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7369-6



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

養成読本編集部 編
B5判・200ページ
定価2,080円(本体)+税
ISBN 978-4-7741-8034-2

養成読本編集部 編
B5判・112ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7992-6



養成読本編集部 編
B5判・176ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7993-3

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

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

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

第108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。

ES Modules

JavaScriptをモジュール化する「ES Modules」

現在のJavaScriptの仕様は標準化団体のEcma Internationalによって策定されており、ECMAScriptと呼ばれています[注]。このECMAScriptには、ECMAScript 2015（ECMAScript 6とも呼ばれる）から「モジュール」と呼ばれるしくみが追加されました。この仕様はES ModulesやESM、ES6 Modulesなどの名称で呼ばれています（本稿ではES Modulesと表記します）。

ES Modulesとは、簡単に言ってしまえば外部のJavaScriptのスクリプトファイルをimportで取り込んだり、逆にexportで外部向けに公開したりすることができる仕様です。従来のJavaScriptには、外部のファイルを取り込む標準的な方法が用意されていませんでした。

たとえばWebブラウザ向けのJavaScriptの場合、HTMLコードに複数のscriptタグを書き込むことでJavaScriptファイルを分けることは可能でしたが、この方法には次のような問題がありました。

- HTMLファイルとの結合度が強く、JavaScriptファイル単体での独立性が低い
- ファイル読み込みの順序に気をつける必要がある

大規模なアプリケーションの場合、この方法のままではファイルの依存関係などの管理が煩雑になり、現実的ではありません。そこで、適切なモジュール化を実現するためのサードパーティ製のツールが登場しました。これらのツールを使えばJavaScriptのモジュール化は実現できますが、標準仕様ではないため互換性に関する問題を抱えていました。その点ES ModulesはECMAの標準仕様であり、互換性の面では大きな強みがあります。

ES Modulesの使用方法

次のコードは、hello()という関数をexportする例です。

```
export function hello() {...}
```

この関数がhoge.jsに宣言されている場合、これを次のようにimportすることで、同じファイルにある関数と同様に使用できるようになります。

```
import {hello} from "./hoge.js";
```

さて、ECMAScriptの言語仕様ではモジュール機能に関する文法だけが規定されており、実際の挙動については実装側の団体が規格を決定するようになっています。

Webブラウザ側はWHATWGが仕様の策定を行っています。Webブラウザでは、scriptタグでの読み込みの際にtype属性に“module”と指定されている場合だけ、そのファイルをES Modulesと判定します。それ以外の値で読み込んだファイルではimport/exportの宣言がエラーになります。JavaScriptは、従来どおりインラインで記述することもsrc属性で外部ファイルを読み込むこともできます。

```
<script type="module">
<script type="module" src="hoge.js">
```

importで指定するモジュール識別子（fromのあとの部分）はURL形式でなければいけません。拡張子を省略することもできません。

サーバサイドJavaScriptについては、Node.jsが正式サポートを目指して仕様を策定しています。Node.jsではファイル拡張子が「.mjs」か否かでES Modulesであることを判定します。したがって、拡張子が.jsのファイルでimportやexportを使うことはできません。importの際のモジュール識別子はWebブラウザと同様にURLとして扱われますが、拡張子が省略できるという点が異なります。

そのほか、Node.jsの場合にはCommonJS Modulesという既存のモジュール機構があるため、それとES Modulesとの相互互換性という問題を抱えています。これについては制限付きで互換性を保つという方針になっているため、両方のモジュール機構をサポートする場合や、混在させる場合などには注意が必要です。

実装側が完全にES Modulesに対応するまでにはもう少し時間がかかるかと思われますが、次世代のスタンダードとなる見込みは極めて高いので、早い段階で準備を進めておく必要があるでしょう。SD

Standard ECMA-262
http://www.ecma-international.org/publications/standards/Ecma-262.htm

注）ECMAScript仕様は正式には「ECMA-262」という規格番号で標準化されています。

DIGITAL GADGET vol.228

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

箱を開ける楽しみ。新しい購入の形、サブスクリプション

箱に入った商品と新しい購入方法

デジタルガジェット好きなら“Unboxing”と呼ばれる、箱を開ける瞬間のワクワク感を何度も体験していることでしょう。最近では、味気ない茶色の段ボールに入った製品のほうが少なく、コストのかかった美しい化粧箱に入り、梱包の仕方も、取り出し方にも工夫を凝らした箱を多く見るようになってきました（**pic.1**）。

また、箱そのものが役に立つ梱包手段もあります。フィンランドでは子供が生まれたら、国から赤ちゃんを育てるために必要なグッズ一式が段ボール箱で届くそうです（しかも無料で）。そしてその大きくてカラフルな段ボールの空き箱は、そのままベビーベッドとしても使えるサイズなのです（**pic.2**）。赤ちゃんスターターキットとも言えるこのしくみはFINNISH BABY BOX（https://www.finnishbabybox.com/）のほか、いくつかのネットサービスにより、フィンランド以外の国でも購入できるようになっています。

工夫の凝らされた箱の例として、Palm Pixe Plusの箱を知ったときは、たいへん驚きました。携帯電話Palm Pixe Plusが収められた箱は、箱の角が斜面になっており、店頭に陳列されるときも目立ちやすいように、また購入後も箱が捨てられることなく、携帯電話の置き場所として使い続けられるような工夫がなされています（**pic.3**）。

また一方、Amazonの特定商品をボタンを押すだけですぐに購入できるAmazon Dashボタンは簡素な箱でありながら、中身のロゴが見えており、開封せずとも何の商品なのかが一目でわかるように工夫されています。何種類も商品が存在するAmazon Dashボタンの箱に個別の印刷などをせずとも、1種類でまかなえる箱になっているのです（**pic.4**）。

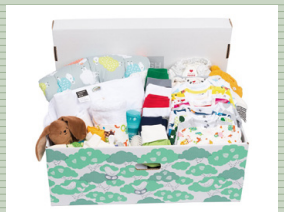
そういった箱を開封する楽しみや便利さを、継続して利用することのできる「サブスクリプション（定期購買）」サービスが、オンラインショッピング周辺で広がってきています。

サブスクリプションで手に入るもの

新聞や雑誌の定期購読や、携帯電話アプリの利用料金などで、ビジネスモデルとしてのサブスクリプションは古くから知られていますが、最近では店舗で購入するようなもの、定期的に購入する日用品、衣服などをサブスクリプション購入できるサービスが広がっています。

今までそのような宅配サービスのなかったタイプの商品をサブスクリプションできるということで注目を浴びはじめたのは、いち早く登場したカミソリのサブスクリプションサービ

pic.1 初代iPhoneの化粧ケース


pic.2 北欧発祥の赤ちゃんボックス「FINNISH BABY BOX」


pic.3 角が斜面になっており、置いた際に斜めに置ける「Palm Pixe Plus」の化粧ケース

pic.4 「Amazon Dashボタン」の箱。箱から開封せずとも、中身を確認できる

※本記事で紹介しているものは
国内未発表・未発売のものを含んでいます。

ス、Dollar Shave Club（http://try.dollarshaveclub.com/welcome/）の評判が広がってからです（**pic.5**）。毎月買わなくてもなんとかなる、買うタイミングを逃しがちな日用品を定期購入するというビジネスモデルは大成功で、Dollar Shave Clubは9,000万ドルの投資を得たうえに、10億ドルで消費財大手の企業に買収されました。

そうして、販売されている商品を購入してモノを所有する風潮から、サービスの利用に対して月ごとに支払う風潮に移りつつあります。最近ではサブスクリプションへの抵抗感が薄れてきたことを示すかのように「サブスク」と略されることもあるようです。

定期購買のしくみは古くからありましたが、インターネットとオンラインショッピングの普及によって、今まで思いもつかなかったような商品が定期購買できるようになってきています。それらは、大きく分けて次のようなカテゴリに分類されます。

- 毎月消費する日常品をお届けするもの
- 毎回何が入っているかわからない、サンプルや選別された商品が届くもの
- 何度でも借りられる、気に入らなければ返品できるレンタル式のもの
- 定期購買することにより、値引きがなされるもの
- 定期購買でしか購入できない、予約制のもの

それぞれについて、いくつか具体例を紹介しましょう。

毎月消費する日常品をお届けするもの

歯ブラシの定期購入
「Good Mouth」（**pic.6**）
https://boka.com/goodmouth

月10ドルの食品サンプル
「Love with Food」（**pic.7**）
https://lovewithfood.com/

日常品をなんでも組み合わせ
「Manpacks」
http://www.manpacks.com/

毎回何が入っているかわからない、サンプルや選別された商品が届くもの

月10ドルの化粧品サンプル
「BIRCHBOX」（**pic.8**）
http://www.birchbox.com/

普段使いできるポーチ入り化粧品「ipsy」
https://www.ipsy.com/

カリフォルニアからの雑貨いろいろ
「SUN-Kissed Box」
http://sunkissedbox.com/

日本の駄菓子
「Freedom Japanese Market」（**pic.9**）
https://www.freedomjapanesemarket.com/

おじいちゃん・おばあちゃんへの
定期プレゼント「Grand Box」
http://www.mygrandbox.com/

何度でも借りられる、気に入らなければ返品できるレンタル式のもの

オシャレな服を1回69ドル
「Bombfell」（**pic.10**）
http://www.bombfell.com/

ネクタイ、カフスなどのアイテムレンタル
「KASHI KARI」
https://kashi-kari.jp/

専門スタイリストがチョイスするビジネススーツ
「suitsbox」
https://www.makuake.com/project/suitsbox/

定期購買することにより、値引きがなされるもの

好みを反映した子供服
「Rockets of Awesome」（**pic.11**）
http://welcome.rocketsofawesome.com/

ビジネスシャツ「Hall Madden」
https://hallmadden.com/

月に1本10.95ドルからのネクタイ
「Tie Society」
http://www.tiesociety.jp/

定期購買でしか購入できない、予約制のもの

カスタムシャンプー
「Function Of Beauty」（**pic.12**）
https://www.functionofbeauty.com/

オリジナルの化粧品を作れる「BITE beauty」
http://www.bitebeauty.com/

レコード購入「Vinyl Me, Please」
http://www.vinylmeplease.com/

pic.5 カミソリの定期購買「Dollar Shave Club」

pic.6 Good Mouth

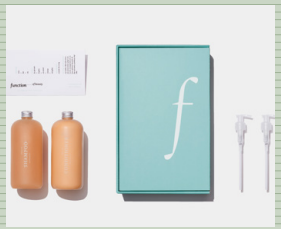
pic.7 Love with Food

pic.8 BIRCHBOX

pic.9 Freedom Japanese Market

pic.10 Bombfell

pic.11 Rockets of Awesome



pic.12 Function Of Beauty

月20ドルでスタイリストがチョイスした服「STITCH FIX」
https://www.stitchfix.com/

子供のコンピュータ学習ボックス「Bitsbox」
https://bitsbox.com/

大人の学習ボックス「New Hobby Box」
http://www.newhobbybox.com/

モノの所有の仕方、モノの買い方が変わるサブスクリプション

サブスクリプションモデルの販売方法になると、1年を通しての財務的な計画が変わってきたり、今まで以上に解約率を減らすことが重要視されてきたりしています。サブスクリプションサービスを始めるための環境を提供してくれるWebサービスCratejoy（https://www.cratejoy.com/）などもあり、サブスクリプションで売れるものは無限に広がってくるでしょう。

サブスクリプションサービスで扱われるようになったものから、いまだに店舗で買うことが主流な商品もあり、モノの買い方、所有の仕方、支払いの仕方がネットの進化とともに大きく変化してきています。

最近ではCDを購入したり音楽ファイルを個別にダウンロード購入したりする層が減り、月額で聴き放題の音楽サービスに加入する層が増えているそうで、「所有」から「利用する権利」に移りつつあるのかもしれません。

その一方で、レコードジャケットの大きな盤面ケースに郷愁を感じたり、DVDやBlu-rayの豪華な紙箱で手元に持っておきたかったりする感情も存在します。また、VHSビデオテープやMDのように、再生装置がそのうち一般的ではなくなってしまう心配がありながら、カセットテープや使い捨てフィルムカメラのように、不便さや制限を楽しむという不思議な傾向もでてきました。

定期購入の仕方がインターネットの普及で手軽になったり、サブスクリプションのように支払いの仕方が変わったりしても、箱を開けるときのワクワク感は、どんなに時代が進化しても変わらないのかもしれませんね。SD

Gadget 1

Brick Phone Case 1973'

https://en.pinkoi.com/product/TLYm8ARp

大昔の携帯電話風ケース

Brick Phone Case 1973'は、携帯電話登場当初、まだまだ現在のような小さくてスマートな携帯電話になる以前の、レンガのように巨大でゴツい形状だったころの携帯電話を模した小物入れケースです。チャックを開けると、ケーブルやバッテリー、デジタルガジェット向けの変換コネクタなど、細々したものを収納できます。完全に冗談グッズではありますが、鞄からこのケースを取り出すと、皆の注目を集めることは間違いなしです。

Gadget 2

おもいでばこ

http://omoidebako.jp/

デジタル写真収納／共有専用デバイス

「おもいでばこ」は、デジタル写真の記録、共有に特化した、世代を越えて平易に使えるデジタルデバイスです。スマートフォンの普及で、写真を撮影したり、SNSなどで共有することは爆発的に増えましたが、写真プリントしてアルバム化したり、大きな画面で大勢で見たりといったことは逆に減ってきた印象があります。「おもいでばこ」はデジタル機器特有の複雑な操作なしで、デジタル写真を保存し、テレビに接続し写真を表示して楽しめるデジタルの箱です。

Gadget 3

クラシック収納箱

http://www.columbuscircle.co.jp/products/?id=1502347558-111814&ca=22

ゲーム機専用箱

（クラシックミニSFC用）クラシック収納箱は、2017年10月に発売された、昔を懐かしむためのスーパーファミコンクラシックミニを収納するための専用の箱です（任天堂のライセンス商品ではありません）。ミニサイズの本体、ケーブル、コントローラをすべて収納し、持ち運びやすいケースとして取り扱えます。純正の専用ケースが用意されているデジタルガジェットも多いですが、デジタルカメラをはじめ、いろいろなケースを活用、工夫していくのもデジタルガジェットの楽しみの1つです。

Gadget 4

通信薬箱（実証実験）

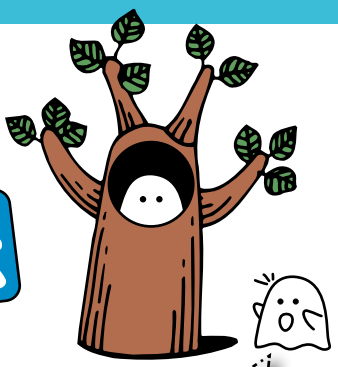
http://www.toppan.co.jp/news/2017/07/newsrelease170710.html

通信機能付き薬箱

凸版印刷とデンソーウェーブが実証実験を進める通信機能付き薬箱は、ICタグを活用した薬の包みとの組み合わせで、薬の飲み忘れ防止、遠隔地からの見守りを実現するためのデジタル機器です。薬の飲み忘れ、分量の間違い、飲み過ぎなど、とくに高齢者向けのソリューションとして期待されます。アラームによる服薬時刻の通知や、実際に箱から薬を取り出したことを検知する機構、それらの情報とクラウド上のサービスとの連携で運用されます。

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

結城浩の再発見の発想法 55



DoS攻撃

DoS攻撃とは

DoS（ドス）**攻撃**とは、セキュリティの用語で、**サービス提供者に対して大量の要求を出し、サービス提供者のリソースを消費し尽くす攻撃**のことです。“DoS”（ドス）は“Denial of Service”（サービス拒否）の略語で、多くの場合“o”は小文字で表記されます。

典型的なDoS攻撃は、1つのWebサーバに対して連続的にアクセスしたり、大量のデータを送りつけたりする行為です。アクセスを繰り返すのは数の攻撃で、大量のデータを送りつけるのは量の攻撃とも言えるでしょう。Webサーバはクライアントからの要求に応えるように作られていますが、応答できるキャパシティは無限ではありません。要求に応えるためにはネットワーク帯域、メモリ、ディスク資源などのリソースが必要だからです。リソースを消費し尽くすと、Webサーバの反応速度が極端に遅くなったり、エラーになったりするでしょう。そうすると、攻撃者以外のユーザはそのWebサーバにアクセスできなくなります。ここではWebサーバが攻撃を受ける「サービス提供者」となっています。

ここでポイントとなるのは、DoS攻撃を行うにあたって、攻撃者は難しい暗号を解いたりする必要はない点です。DoS攻撃は「機密性」への攻撃ではなく「可用性」への攻撃と言えます。攻撃者は、サービス提供者が公開している窓口に通常どおりアクセスしており、ただし、そのアクセスしている数や量が非常に多いだけなのです。

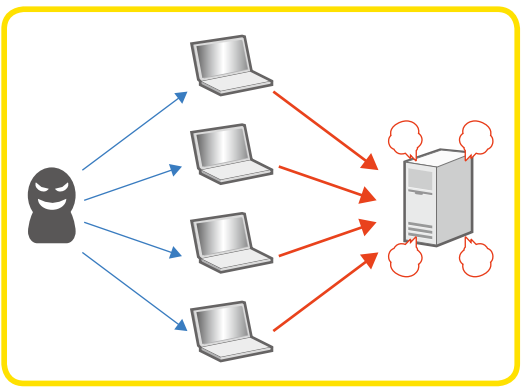
DDoS攻撃

攻撃を行うクライアントがたった1台ならば、サービス提供者の防御は比較的楽です。異常な要求を行ってくるクライアントのみをアクセス拒否してしまえばいいからです。

しかし、DoS攻撃が分散化された**DDoS**（ディードス）**攻撃**を防ぐのは難しくなります。“DDoS”は“Distributed Denial of Service”（分散型DoS）の略で、DoS攻撃が多数のクライアントから行われる攻撃のことです（**図1**）。

攻撃者はDoS攻撃を行うにあたり、攻撃用に多数のクライアントを用意します。そして、攻撃者の命令でそのクライアントがいっせいにDoS攻撃を開始するのです。DDoS攻撃は、店舗に多人数で押し寄せるようなもので、原始的といえば原始的ですが、それだけに防ぐことは

▼図1　DDoS攻撃（分散型）



難しいと言えます。

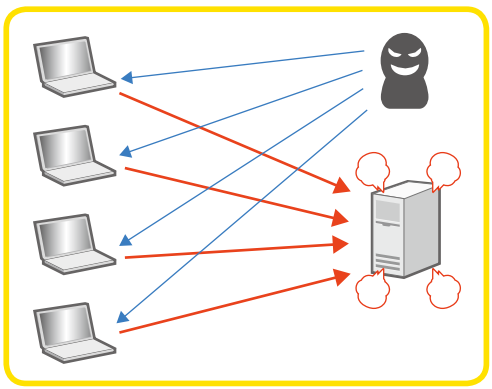
このときに攻撃者は多数のコンピュータが必要になりますが、そのコンピュータを攻撃者が所有している必要はありません。コンピュータウイルスなどを介して前もって世界中のコンピュータに自分のプログラムを違法に配置しておき、攻撃者の命令(あるいは特定の日時)で自動的に「DoS攻撃開始」する方法が考えられるからです。この場合、自分のサービスとは無関係な世界各国のユーザからアクセスされることになり、防御は難しくなるでしょう。

DRDoS攻撃

DDoS攻撃の中で、さらに巧妙な攻撃として**DRDoS(ディーアールドス)攻撃**があります。"DRDoS"は"Distributed Reflective Denial of Service"(分散反射型DoS)の略です。

DRDoS攻撃では、「リクエストに対して自動的に応答を返す」ネットワークプロトコルが悪用されます。まず、攻撃者は、自分が攻撃したいサービス提供者のコンピュータになりすまします。そして、非常に多数のコンピュータ(攻撃対象ではない)に対して「応答を要するリクエスト」を発行します。そのリクエストを受け取った多数のコンピュータは、ネットワークプロトコルに従い、悪意なく無邪気に応答を返します。ただし、その応答を返す相手は、攻撃者ではなく、攻撃者がなりすましたサービス提供者です。

▼図2　DRDoS攻撃(分散反射型)



その結果、無邪気な多数の応答がサービス提供者に襲いかかることになります。あたかも光が鏡で反射するように、攻撃者のリクエストが多数のコンピュータで反射することになるので、分散反射型と呼ばれているのです(**図2**)。

日常生活とDoS攻撃

大量に来る**迷惑メール**は受信者に対するDoS攻撃になります。本来コミュニケーションを取りたい相手からのメールを仕分けすることが難しくなるでしょう。迷惑メールは、受信者のメール処理能力というリソースを消費させていることになります。迷惑メールをフィルタリングする、不用意にメールアドレスを公開しないなどの防御が必要です。

これは犯罪ですが、攻撃対象になりすまし、ネットショップを使って**不愉快な商品を送りつける行為**は反射的なDoS攻撃と言えます。

カスタマーサービスに対して**大人数で電話をかける行為**は、結果的にDDoS攻撃になる場合がありますね。会社にとって「ユーザに電話で応対する」のはコストがかかることですから、各社とも負荷分散をする工夫がなされています。Webサイトに「よくある質問」を掲載する、電話番号は掲載せずにWebフォームだけのサポートにするなどです。

大規模災害の**被災地に物品を送る行為**が問題になる場合があります。古着や千羽鶴などが全国から被災地に送られて、ただでさえ逼迫(ひっぱく)している現地のリソースが消費されてしまうのは、DDoS攻撃と言えます。しかも送っている側は善意のつもりなので対策はやっかいです。十分に注意しないと、報道機関はDRDoS攻撃の「攻撃者」となってしまいますね。

あなたの周りを見回して、サービス提供者のリソースを消費してしまう行為はありませんか。分散している無邪気なユーザへの不用意な指示によって、はからずもDDoS攻撃やDRDoS攻撃が生まれていることはありませんか。ぜひ、考えてみてください。SD

及川卓也の プロダクト開発の道しるべ

品質を高めるプロダクトマネージャーの仕事とは？

第14回 OKR

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

みなさんもOKRという言葉を聞いたことがあるかもしれません。OKRは米国のハイテク企業を中心として用いられている目標の設定と管理手法です。

OKRは必ずしもプロダクトマネージャーにだけ関係するものではありません。むしろ、企業の経営や事業運営により深く関わります。しかし、プロダクトマネージャーによるプロダクトチームの目標設定手法としても有効です。

今回はこのOKRについて解説しましょう。

OKRとは

OKRはObjectives and Key Resultsの略です。日本語で言うと、「目的と主要結果」とでもなるかと思いますが、その言葉のとおり、企業や組織、プロジェクトなどの業績を目的とその結果で管理する手法です。

目標設定とそのレビューはどの組織でも行っているかと思いますが、OKRは目的に対する結果をあらかじめ明確化しておくことにより、目的の達成度を測りやすくしたところが特徴です。従来の目標管理だと、ともすると「できたか・できなかったか」のどちらかでしか成否を判断できなかったり、努力していることを情緒的に評価してしまったりするというあいまいさが生じてしまうことがありました。

OKRは、誰が見ても、その目的への達成度が明確になることを理想とします。のちほど例で具体的に示しますが、できるだけ数値などを主要結果(KR)とすることで、客観的にその成否を判定できます。この客観的な判定が可能というところがOKRのポイントです。判定をする人は自分も含みます。OKRに限らず、自分で立てた目標を判定する段階になって、これは達成できたと言って良いか悩んだ経験がある人もいることでしょう。先ほどの「できたか・できなかったか」のような二者択一の目標設定をしてしまった場合でも、達成できなかった場合に達成度がゼロなのかというと、そうとは言えないことも多いのではないでしょうか。理想的なOKRはそのような場合でもきちんと達成度を判定できるものです。

なぜ、今OKRか

OKRはIntel社内で生まれ、その後、GoogleやFacebookなどのシリコンバレー企業が採用したことにより一般的になりました。日本でもメルカリが採用するなどで、スタートアップを中心とした企業で導入が進んでいます。

Intelが利用を始めたのも、Googleが導入したのもかなり前のことですが、今なぜ注目を集めているのでしょう？

それは、OKRが企業や組織が率先して取り組むべき作業に集中するための良いフレームワークであるからです。企業、とくにスタートアップのような資金も乏しい組織においては、従業員が優先度の低い仕事に労力を割いていては、資金も組織力も規模の異なる大企業に太刀打ちできません。限りある資金を投資すべきところを明確にし、そこに全リソースを投入するには、OKRのように明

確に達成すべき結果を示したものが求められます。

OKRを利用することにより、何を優先すべきかと同時に、何を今はしなくて良いのかも明確になります。また、OKRは社内で共有されますので、社内の議論の核となり、社内コミュニケーションも円滑化されます。「全社一丸となって」という言葉もありますが、まさにOKRはそのためのフレームワークなのです。

OKRの具体例

それでは、具体例を見ましょう。

ここでは、プロ野球チームを例に取ります。興行収入が赤字となってしまったチームの立て直しを命じられたプロ野球チームがあったとします。このチームのOKRを見てみましょう。

このチームの今シーズンの全社OKRの目的(Objective)は「黒字化達成」です。何はともあれ、親会社に頼らずに運営できる企業として自立する必要があります。

この「黒字化達成」という目的は次の3つの主要結果(Key Results)によって判定されます。

1. クライマックスシリーズ出場
2. 観客数増員、前年度比150%
3. グッズ販売の売上、前年比200%

実際にはプロ野球チームの経営はほかにもさまざまな要素があるでしょうが、ここではこの3つに簡略化して説明します。説明上簡略化したとはいえ、実際のOKRもこのくらいのシンプルさが求められます。全社OKRの項目はどんな大企業であったとしても、全社員が覚えられ、常に意識していられるわかりやすさが重要です。

さて、OKRはこの全社OKRから、さらに部署ごとのOKRへと継承・分割されていきます。つまり、1つめの「クライマックスシリーズ出場」という主要結果を今度はチーム管理・選手育成部署がそれを目的とした主要結果を考えていくことになります。また、2つめの「観客数増員、前年度比150%」と3つめの「グッズ販売の売上、前年比200%」は広報や営業部署が主要結果を考えます。

その結果、チーム管理・選手育成部署は次のように主要結果を考えました。これはチーム力の強化によりクライマックスシリーズに出場するという目的を達成するための主要結果です。

- 投手力強化：チーム防御率3点台
- 3割打者を3選手以上
- チーム本塁打120本以上

また、同じように広報部署は観客数150%化のために次の主要結果を立てました。

- ファン感謝イベント企画を10本以上行う
- ホーム球場沿線の車内広告をゴールデンウィークまでに開始

グッズ販売の売上増加についても営業部署が次のように主要結果を立てました。

- 新しいマスコットキャラクタを制作し、そのグッズ販売を開幕までに開始
- 有名衣料品チェーンとのコラボTシャツを制作し、その販売をゴールデンウィークまでに開始

これらの主要結果はさらにその所属部署によって分割・継承されていきます。たとえば、チーム管理・選手育成部署の中の選手採用担当部署では、「投手力強化：チーム防御率3点台」を目的として、次のような主要結果を考えられます。

- ドラフト会議にて、即戦力となる新人の獲得
- 開幕までにトレードによる中継ぎの獲得

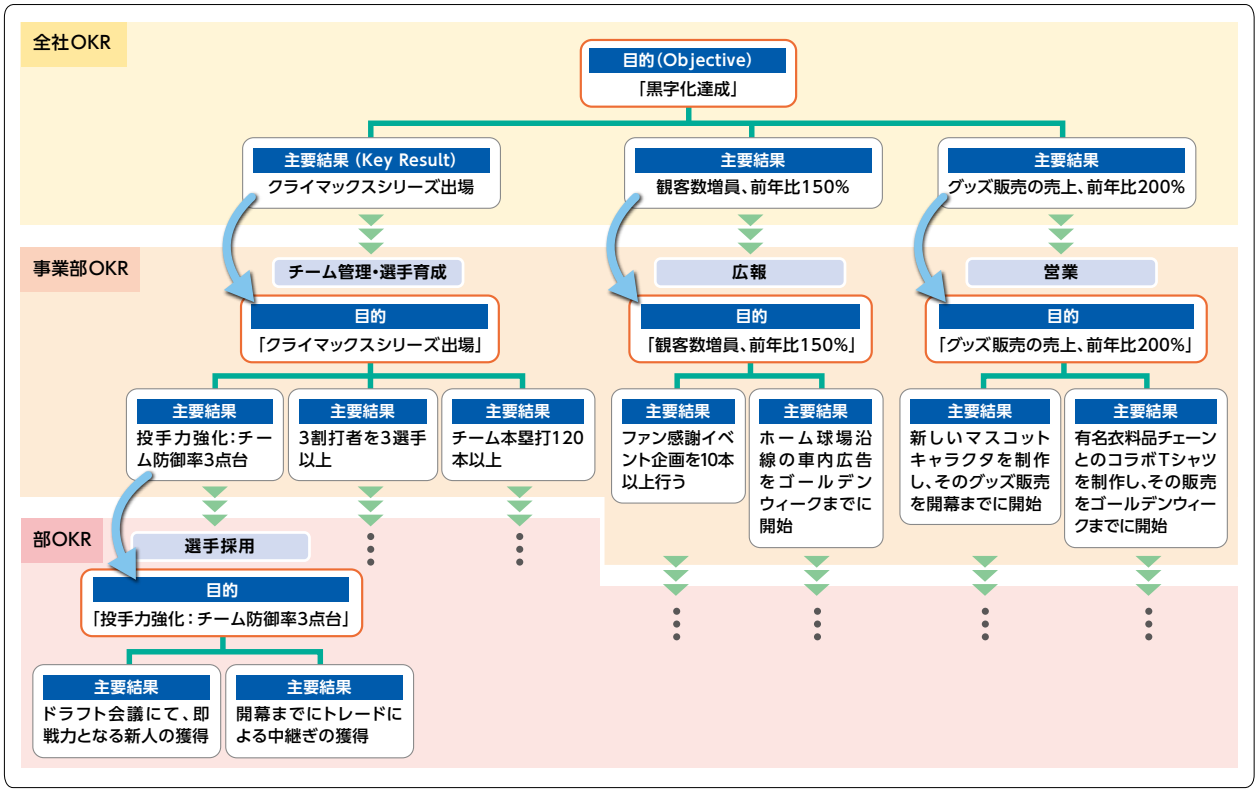
ほかは省略しますが、ここまでで説明したこのプロ野球チームのOKRの全体は**図1**のようになります。

OKRはこのように、上位のOKRの主要結果(KR)部分を下位は目的(Objective)として引き継いでいくことにより、全社で整合性の取れたものとしていきます。

OKRの使い方

では、実際の組織でのOKRの使い方をさら

▼図1　プロ野球チームのOKR全体像



に詳しく見ていきましょう。

OKR作成方法

OKRを作成する場合には、できるだけ上位の部署から作成していきます。全社的にOKRが導入されている場合には、全社OKRが最も上位のOKRにあたります。このように上位のOKRが準備されることにより、先ほどの例で見たように、下位の部署が自分たちの役割に基づき、上位の目的を達成するための貢献を考えることができます。

OKRは年間のOKRを作成した後、それを四半期ごとに分割して実現していく形をとることがあります。年間OKRなしで、四半期ごとのOKRだけを用意するのでも良いですが、筆者は少なくとも全社OKRもしくはそれに相当する最上位のOKRについては年間OKRを用意することを勧めています。年間OKRを時間軸で達成していく方向での分割と、組織軸で粒度を細かくしていく方向での分割の2方向の分割でOKRが組み上げられていくことで、達成への具体的な手段が明確となります(**図2**)。

なお、このように上位組織からOKRを順に作成する方法は、律儀に守ろうとすると、上位組織のOKRが作成されない限り、下位組織はOKRを作成できないということになり、OKR作成までに時間がかかってしまいます。これでは、OKRというフレームワークを用いたプロセスを回すことだけが目標となり本末転倒です。年間の全社OKRが公開されているなど、上位のOKRが予測できる場合には、とりあえずそれを基にOKRの用意を進め、もし予想していないOKRが上位で作成された場合は期中であってもOKRを更新するという割り切った運用を取り入れるなど、臨機応変の対応が現実的なこともあります。

SMART

OKR作成時に心がけるべきことにSMARTという概念があります。これは次の5つの頭文字です。

- **S**pecific：具体的に
- **M**easurable：計測可能に
- **A**chievable：達成可能なもの
- **R**elevant：（目標達成に）関連性が高いもの
- **T**ime-oriented：期限を明確に

判定時に自分も含めた誰もが判定しやすくなるような主要結果を作成する必要がありますが、そのためには、このSMARTの5つの基準を採用するようにします。

このうちのAchivable（達成可能なもの）には1つ補足があります。達成可能なものにするあまり、保守的な主要結果を作ってしまうことは望ましくありません。このAchivableの前には、“Stretch but ……”（「厳しいが」）という言葉が隠されていると考えるべきでしょう。

OKRのレビュー

OKRは四半期の始めに作成され、終わりにレビューを行います。レビューはそれぞれの目的と主要結果に対してスコア[注1]を付与することで行います。スコアは1を最大として0.1から0.9を付与する方法と、100％を最大とするパーセントで付与する方法があります。表現方法の違いだけで、どちらも同じです。

先ほどのプロ野球チームの例で、観客数増員を前年度比150％という目的を持つ広報部の主要結果に対する実際の結果が次のようであったとします。

- ファン感謝イベント企画を10本以上行う
  → 5本実施（スコア：0.5）
- ホーム球場沿線の車内広告をゴールデンウィークまでに開始
  → 4月中に実施（スコア：0.7）

この主要結果はSMARTに従って定義されていたため、スコアリングもさほど難しくなく、関係者も納得できるでしょう。目的「観客数増員、前年比150％」のスコアは、主要結果のスコアが0.5と0.7なので、これを基にスコアリングすることもできますが、観客数増員が前年度比150％に達しているかどうかを見て判定するほうが相応（ふさわ）しいでしょう。ただ、達成できなかった場合であっても、主要結果のスコアも参考にして、スコアは考えます。

◆ ◆ ◆

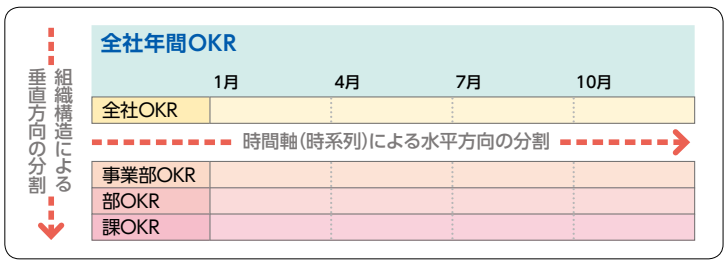
以上がOKRの説明となります。

開発プロジェクトにおいて、このOKRの運用のキーとなるのがプロダクトマネージャーです。エンジニアと事業との橋渡しをするプロダクトマネージャーはプロダクトチームのOKRの設定も行います。

どのようなプロダクトを開発し、どのようにプロダクトを成長させていくかは、プロダクト単独では行えません。同じことはプロダクトの中の各機能についても言えます。企業や組織という大きな単位の戦略との整合性を取りつつ、プロダクトの方向付けを行うために、OKRは有用なフレームワークとなります。SD

注1）グレードと言うこともあります。

▼図2　全社年間OKRから時間軸で四半期ごとのOKRとし、組織階層で粒度を細かくしたOKRを作成する



Profile
ソフトウェアエンジニアとして社会人キャリアをスタートした後、MicrosoftやGoogleでプロダクトマネージャーやエンジニアリングマネージャーを経験。現在はいくつかのスタートアップのプロダクトマネージメントをサポートしている。

第22回 OSC東京／千葉とデブナイト出張

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

年間スケジュールとイベント開催時期

大規模イベントでは、できるだけたくさんの人が参加できる日程を検討する必要があります。**表1**に、月ごとの人の動きやすさを整理してみました。

整理してみると、イベント日程を組みやすいのは6月か10月。ただし、企業協賛は予算が執行可能になるのが5月～6月ぐらいまでずれ込むため、6月はイベント開催にはあまり向いていません。結局、日程に余裕がある10月を中心とした秋にイベント開催の日程が集中することになるわけです。

そこで、できるだけ重複を避けるために日程をずらそうとしますが、2017年はその思惑が逆に一致してしまい、9月にイベントが目白押しになるという事態になってしまったのは皮肉ですね。

▼表1　年間のスケジュール概要

4月	年度が始まったばかりで動きにくい。企業は予算がまだ固まっていないこともある
5月	GWのため、実質最後の1週間ぐらいしか使えない
6月	比較的動きやすい
7月	学生さんは月末ころに前期末試験があるため動けない
8月	夏休みがあるが、逆に狙い目？
9月	上期末なので月後半は動きにくい。週末は運動会などと重複する
10月	イベント開催のピークで日程が重複しまくる
11月	文化祭との重複が多い
12月	忘年会シーズン
1月	年末年始を挟むため、実質最後の1週間ぐらい
2月	学生さんは後期試験や卒論で忙しい。入試関係で学校は貸してもらいにくい
3月	年度末なので月の後半は人の動きが悪くなる

▼写真1　懇親会はもちろん大盛況。土日開催は土曜日が懇親会なので参加者が多めです

今回のOSC東京のイメージは「ゆったり」

イベント過密日程のトップバッターになったためか、いつもに比べると少し出展数やセミナー数が減った感じですが、それでも土日2日間合わせて1,100名とたくさんの人に集まっていただけました（**写真1**）。また、毎回展示スペースが過密でキツキツになることが多いのですが、今回は少し「ゆったり」とレイアウトできたため、結果として出展者と来場者が「ゆったり」と会話できたようです。この「ゆったり感」がよかったようで、出展者のみなさんからは好意的な意見が聞かれました（**写真2**）。一方で、企業出展の方などはやはりイベントとしての集客数のようなわかりやすい客観的な指標がないと協賛しにくくなるわけで、このあたりのバランスが難しいところです。

今後、数値的な指標だけでなく、「出展者・来場者満足度」のような質的な指標も提示していく必要があるかもしれません。

デブナイト出張版を開催

これまでのOSCでは、基本的に招待講演などは行わず、活動の成果を発表したい企業やコミュニティからの自発的な参加申請に基づいてセミナーや展示を行ってきました（**写**

▼写真2　毎度お馴染み終了後のスタッフ集合写真。いろいろな人が片付けを手伝ってくれたので早めに終わりました

▼写真3　OSCドリブンで開発が進められているプラレール半加算器。今回はさらにコンパクトになりました

真3)。ただ、最近のようにテーマが多様化してくると、OSCのようなメタなイベントですべてをカバーすることが難しくなってきます。その現れの1つが、前回レポートした開発者向けのイベントであるODC(Open Developers Conference)でもあったわけです。

そこで今回のOSC東京では、ほんの少しですがオープンソースとは関係ないセミナーを開催してみました。仕事柄、とかく不健康になりがちなエンジニア向けに開催してきた「デブナイト」というイベントの出張版です。これまでウェアラブルデバイスや健康アプリについてのLT中心で開催してきましたが、今回は板橋 里麻さん(株式会社たべかた代表・管理栄養士)をお招きして、健康についてのセミナーを開催しました。たべかたさんに作っていただいたランチボックス(ヘルシータコライス)を食べながら(**写真4**)、エンジニアが知っておきたい「食」の話を聞くという趣向です。かなり多岐に渡ったお話がありましたが、たとえばおやつにはカカオ成分の多いチョコレートや、低GI食品と呼ばれるスナック類が太りにくくていいようです。とくに大豆食品(納豆や豆腐など)が健康にいいとのこと。ぜひ参考にしてみてください。

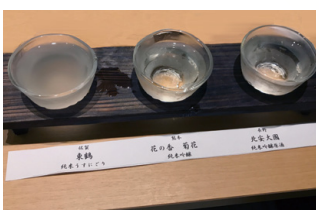
OSC千葉も開催しましたよ

OSC東京の前哨戦ということで、9月2日(土)に千葉工業大学でOSC千葉を開催しました。OSC東京の会場である日野市は千葉方面からみると完全に東京の横断が必要なため、場合によっては2時間以上かかってしまいます。昨年はアンカンファレンス形式でしたが、今回は大部屋1つを15分ごとのセミナー、そして廊下踊り場に展示スペースという形で開催し、100名以上の人に集まってもらえました。こちらも「ゆったり感」は同じで、出展者、参加者同士の会話も弾みました。コミュニティイベントとしてはこれぐらいがちょうどいいのですよね。

懇親会にも50名以上が集まり、大量の揚げ物と格闘するなどのイベントもありつつも盛り上がりました(**写真5**)。2次会も盛り上がり過ぎてうっかり終電を逃すという失態を演じてしまいましたが、楽しい開催となりました(**写真6**)。来年はもう少し規模感を大きくしての開催になりそうです。SD

▼写真4　ヘルシータコライス。お肉の代わりに大豆ミートで作られています

▼写真5　懇親会で出てきた揚げ物盛り合わせ。みんなで頑張りましたが、完食できず



▲写真6　2次会で飲んでいたらしい日本酒飲み比べ。すでに記憶がなく、スマホの写真で気づきました

ツボイのなんでもネットにつなげちまえ道場

第30回

LoRaWANを使ってみる（中編）

Author 坪井 義浩（つぼい よしひろ） @ytsuboi

前回まで

前編ではLoRaWANの概要と、LoRa Arduino開発シールド AL-050を試しに動かしてみたところまで紹介をしました。しかし、単にシールドを動かしてみただけではLoRaWANのことはよくわかりません。そこで今回は、STマイクロエレクトロニクスのB-L072Z-LRWAN1 STM32L0 LoRa Discovery Kit（以降B-L072Z-LRWAN1）（**写真1**）を使い、LoRaWANを実際に体験してみます。

B-L072Z-LRWAN1

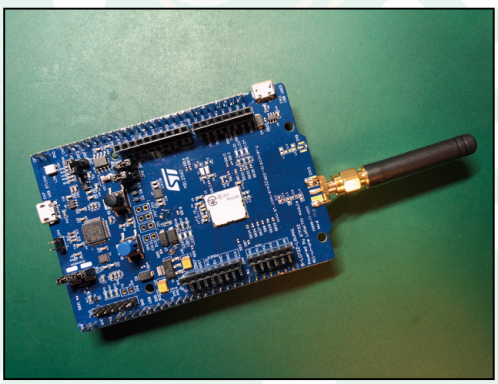
B-L072Z-LRWAN1[注1]は、村田製作所のCMWX1ZZABZ-091というモジュールを採用しており、このモジュールの中にはSTM32L072CZというCortex-M0+のマイコンと、SemtechのSX1276という無線のチップが入っています。Cortex-M0+のマイコンを使い、Mbed enabledとありますので、Mbedでこのボードの開発もできます。

しかし、MbedのB-L072Z-LRWAN1のページ[注2]を見る限り、本稿執筆時点ではLoRaWANを使うためのドライバが提供されていません。また、Mbed用にSemtechが提供しているLoRaWANのライブラリ[注3]は、今のところ日本を含むアジア10ヵ国向けのAS923という仕様をサポートしていません。国内で使うには、この実装が必須です。このような事情で、本連載らしくMbedでの開発を紹介したいところですが、今回はMbedを使わず、MDKというArmのIDEを使って開発をしてみます。

今回、B-L072Z-LRWAN1を使うことにしたのは、B-L072Z-LRWAN1で採用されているモジュール、CMWX1ZZABZ-091が工事設計認証を受けているからです。海外からB-L072Z-LRWAN1を購入した場合などには技適マークが貼付されていませんが、国内のいくつかの販売店から購入した場合、技適マークと工事設計認証番号が刷られたステッカーがモジュールに貼付されています。今回は、SORACOMさんのゲートウェイを使って接続をしたいので、同社のUser ConsoleからB-L072Z-LRWAN1[注4]を購入しました。同社から購入したのは、前回紹介した「デバイスID」のネットワークサーバへの登録や、OTAA（Over the air activation）のための仮の暗号鍵が必要だからです。デバイスIDの

注1） http://www.st.com/ja/evaluation-tools/b-l072z-lrwan1.html

▼写真1　STM32L0 LoRa Discovery Kit



注2） https://os.mbed.com/platforms/ST-Discovery-LRWAN1/

注3） https://os.mbed.com/teams/Semtech/code/LoRaWAN-lib/

注4） https://soracom.jp/products/lora/stm32l0/

登録は、User Consoleで「受け取り確認」ボタンを押すことでできます。また、筆者が本稿執筆時に知り得る限りでは、AS923の仕様のうち日本向けの周波数帯を使うように記述されたコードは、SORACOMが購入者向けに提供しているコードのみです。これもB-L072Z-LRWAN1をSORACOMから購入した理由です。

B-L072Z-LRWAN1のセットアップ

さて、開発を行うツールであるMDKですが、http://www.keil.com/mdk-stからダウンロードしてインストールします。インストールやライセンスキーの使い方はダウロードページに記載されています。また、SORACOMのサポートページ[注5]も参考になるでしょう。SORACOMからB-L072Z-LRWAN1購入時に通知されるサンプルコードの中には、5種類のサンプルコードが入っています。ここでは、LoRaWANのエンドノードのサンプルコードである、「End_Node」を使ってみることにします。

サンプルコードのアーカイブの、¥Lora_soracom_rev1¥Projects¥Multi¥Applications¥LoRa¥End_Node¥MDK-ARM¥B-L072Z-LRWAN1を開くと「Lora.uvprojx」というファイルがあります（**図1**）。このファイルがMDKのプロジェクトファイルですので、これをダブルクリックすることでMDKでサンプルプロジェクトを開くことができます（**図2**）。MDKを起動し、プロジェクトを開いたら、コードをビルドしてみましょう。保存アイコンの下にある「Rebuild」か、その左の「Build」ボタンをクリックすることでビルドできます。ビルドを終えたら、B-L072Z-LRWAN1をパソコンに接続してMDKの「LOAD」というアイコンをクリックすることで、ビルドしたプログラムをB-L072Z-LRWAN1のマイコンに書き込みできます。

書き込みをしたら、TeraTermなど適当なシリアルターミナルでB-L072Z-LRWAN1の仮想シリアルポートを開きます。115,200bps、8bit、パリティ・フロー制御なし、ストップビット1bitです。そこで、B-L072Z-LRWAN1にある黒いリセットボタンを押すとマイコンに書き込んだプログラムが動き始めます。

プログラムが動き始めると、マイコンはLoRaWANで接続すべくJOIN REQUESTを送ります。これが、前回説明したOTAAでの仮の鍵を使った認証と、本番の鍵の発行のプロセスです。正常に交換を終えると、ターミナルに「JOINED」と表示され（**図3**）、その後続いて本番の鍵の値が表示されます。

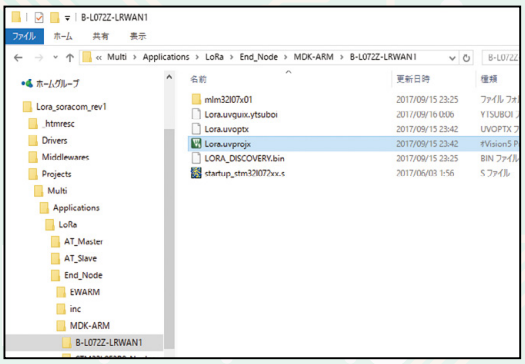
センサの値を送ってみる

End_Nodeは、STマイクロエレクトロニクスのセンサ拡張ボード、X-NUCLEO-IKS01A2[注6]のセンサの値を送信できるように作られていま

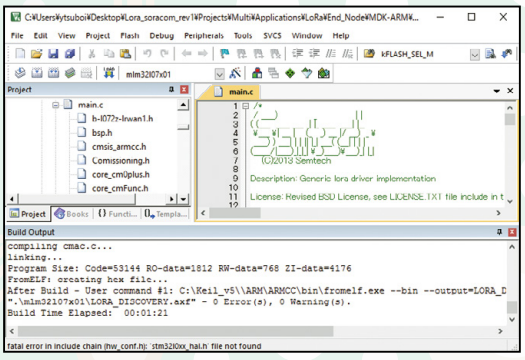
注5）https://dev.soracom.io/jp/start/lora_hw_stm32l0/

注6）http://www.st.com/ja/ecosystems/x-nucleo-iks01a2.html

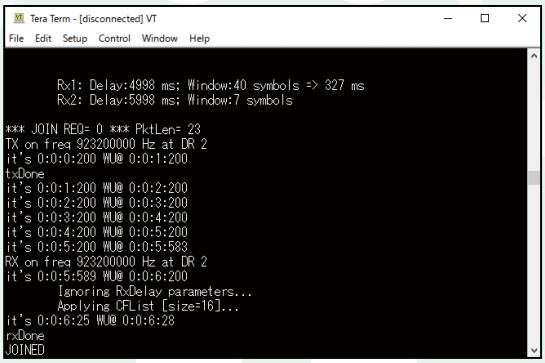
▼図1　プロジェクトファイル



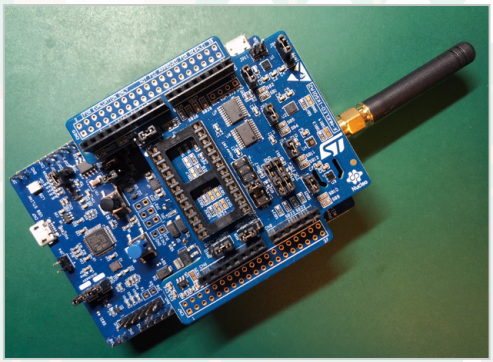
▼図2　MDK



▼図3　TeraTermの画面



▼写真2　センサ拡張ボードを取り付ける



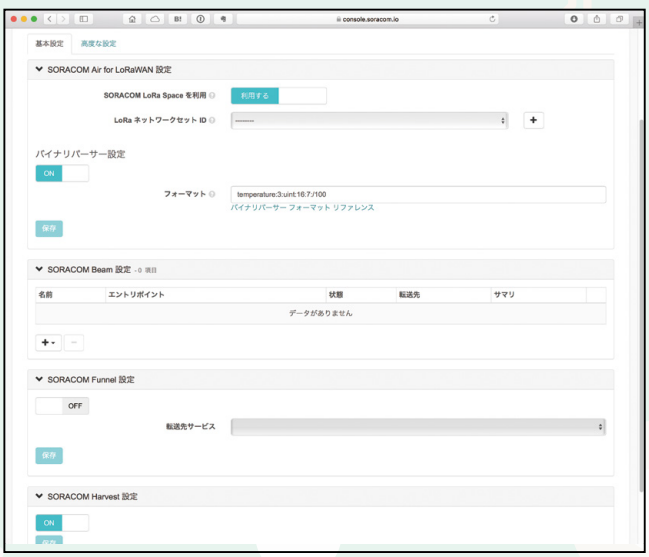
す。せっかくですので、この拡張ボードを使って測った値をLoRaWANで送り、疎通確認をしつつ、ソラコムコンソール(Harvest)上でグラフにしてみたいと思います。

まずは、エンドノードのプログラムを編集し、センサ拡張ボードの値をマイコンが読み込むようにします。MDKの「Project」ペインのmlm32l07x01→Projects/End_Nodeにあるmain.cの左側にある「+」をクリックすると、hw_conf.hというファイルを開くことができます。hw_conf.hの136行目でSENSOR_ENABLEDというマクロがコメントアウトされていますので、アンコメントしてマクロを定義します。そうして再度ビルドを行い、マイコンへの書き込みを行うと、B-L072Z-LRWAN1はセンサから値を読み取り、LoRaWANで値を送るようになりました。

マイコンへの書き込みを終えたら、いったんパソコンとB-L072Z-LRWAN1を接続するUSBケーブルを取り外し、B-L072Z-LRWAN1にセンサ拡張ボードを取り付けます(**写真2**)。

次に、SORACOMのUser Cosoleで値を受信できるようにしましょう。User Cosoleにログインした後メニューボタンをクリックし、「LoRaグループ」を選択して適当な名前でグループを作ります(筆者はここで「Seeed」という名前にしました)(**図4**)。

▼図4　LoRaグループの追加



グループでは「SORACOM LoRa Spaceを利用」と「SORACOM Harvest」をONにするのを忘れないでください。SORACOM Lora Spaceを利用するというのは、共有ゲートウェイを利用するということです。また、SORACOM Harvestというのは、ノードから受け取った値をユーザコンソールから簡単に参照できる機能のことです。もうひとつ、グラフに値をうまく表示するために、「バイナリパーサー」という機能を使いましょう。バイナリパーサーはその名のとおり、受信したバイナリの電文をパース(解釈・変換)する機能です。詳し

くは入力欄の下にリファレンスがありますのでそちらを参照すれば知ることができます。ここでは、End_Nodeから送られてくる値のうち、温度をグラフにできるように「temperature:3:uint:16:7:/100」と記載してください。

センサ拡張ボードを併用すると、B-L072Z-LRWAN1は温度だけでなく、湿度と大気圧も送るのですが、現在のところHarvestのグラフ機能のY軸は1つのスケールしか扱うことができません。大気圧は1,000hPaなど4桁の値でしょうが、温度は20～30℃くらいと値が取り得る範囲が大きく異なりますので、温度だけをグラフに描画することにしました。

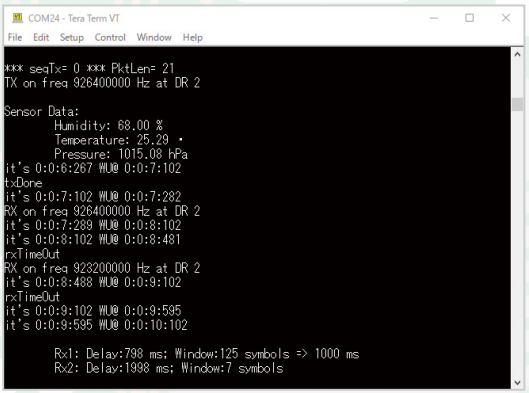
準備が終わったところで、センサ拡張ボードを取り付けたB-L072Z-LRWAN1を動かしてみましょう。先ほどと同様にシリアルターミナルで仮想シリアルポートを開くと、**図5**のように値を送信しているところが確認できます。ここで送っているパケットの長さは21byteで、データレートが2であることがわかります。また、とくに下りのデータを用意していませんので、rxTimeOutと表示されて、そのまま通信が終了しています。

ここで、Webブラウザに戻り、SORACOMのUser Cosoleのメニューから、「LoRaデバイス管理」を選択し、今回使っているB-L072Z-LRWAN1のデバイスIDをクリップボードにコピーします。次に、メニューの一番下にある「データ収集」をクリックし、Harvestの画面に移りましょう。「リソースID」欄にデバイスIDをペーストし、その左の「SIM」と表示されているドロップダウンリストから「LoRaデバイス」を選択しましょう。そこで「検索ボタン」をクリックすると、**図6**のようにグラフが描画されるはずです。

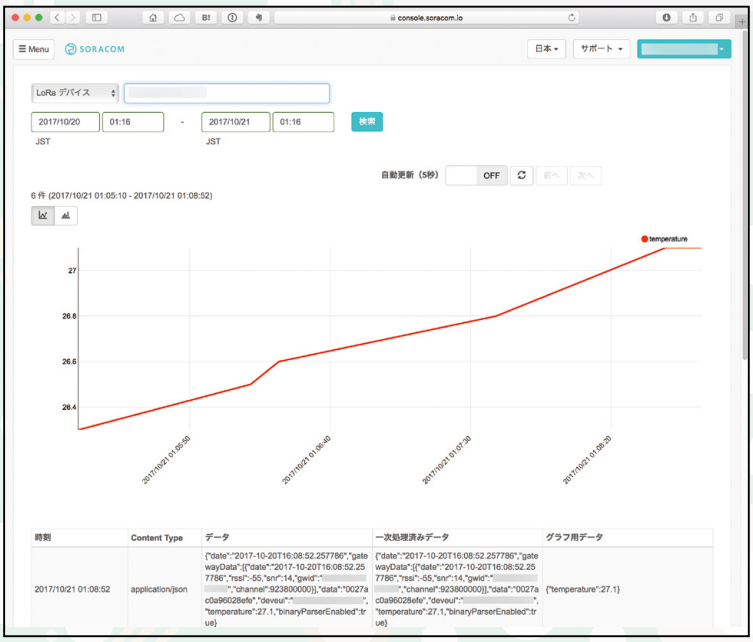
次回

こうして、今回はB-L072Z-LRWAN1を一通り動かしてみました。次回は、ADR（アダプティブ・データレート）を使ってみたり、メッセージタイプを変更したり、下りの通信を試したりしてみましょう。SD

▼図5　センサの値を送信している



▼図6　Harvestでグラフを表示



読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2017年12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 REALFORCE「R2-JP4-BK」

1名

「REALFORCE」はステップスカルプチャー構造と独自のキー荷重特性によって快適なキータッチを実現した、静電容量無接点方式のキーボードブランド。今回新たに、「R2」シリーズが登場しました。丸みを帯びた従来シリーズのデザインから、直線的で洗練されたデザインになっています。「日本語配列・108キーレイアウト・かな表記なし」の標準ブラックモデルをプレゼントします。インターフェースはUSB。

提供元 東プレ　http://www.realforce.co.jp

02 マジックケーブル

「ライトニングケーブル」と「マイクロUSBケーブル」が1つになったケーブル。片側のUSB Type-A端子をPCなどのUSBポートにつなぎ、もう片側の、ライトニング端子とマイクロUSB端子を兼ねた独自設計の端子をiPhoneやスマートフォンと接続します。充電とデータ通信に対応。

提供元 レソナンシアドットワールド
http://resonancia.world

3名

03 pixiv バンカーリング

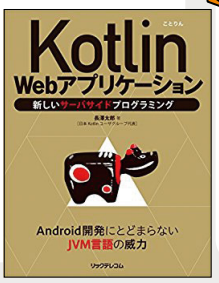
イラストコミュニケーションサービス「pixiv」のサービス開始10周年を記念したイベント「pixiv MEETUP -10th Anniversary-」で配布された、スマホの背面に付けるバンカーリングです。

提供元 ピクシブ
https://www.pixiv.co.jp

1名

04 Kotlin Webアプリケーション

長澤 太郎 著



日本Kotlinユーザグループ代表の著者が執筆。Spark FrameworkやSpring Bootを使ったWebアプリ開発といった、Android開発以外のKotlinのトピックを取り上げています。

提供元 リックテレコム
http://www.ric.co.jp/telecom

2名

05 Haskell 入門

本間 雅洋、類地 孝介、逢坂 時響 著

関数型プログラミング言語Haskellの入門書。型、関数、モナドといった関数型プログラミングの基本概念と、Haskellによる実際のアプリケーション開発の両方を学べる1冊です。

提供元 技術評論社
http://gihyo.jp

2名

06 ソーシャルアプリプラットフォーム構築技法

田中 洋一郎 著

SNSにBOTやAIを活用し、「アプリケーションプラットフォーム」としてビジネス化する際、必要とされる組織、企画、開発技術、運用技術について、広く解説しています。

提供元 技術評論社
http://gihyo.jp

2名

07 IntelliJ IDEA ハンズオン

山本 裕介、今井 勝信 著

IntelliJ IDEAはJetBrains社製の、おもにJavaユーザ向けの統合開発環境。最近流行りのScalaやKotlinでの開発にもピッタリのIDEです。本書は、そんなIntelliJ IDEAの日本初の入門書です

提供元 技術評論社
http://gihyo.jp

2名

第1特集

さあ始めよう!

ITエンジニアと数学

数学プログラミング入門

ITエンジニアにとって、数学を学ぶことにはどのようなメリットがあるでしょうか。数学の知識は、機械学習やブロックチェーンといった高度な技術でも、理論を踏まえたうえで使いこなせるようになることを助けます。それだけではなく、数学的な「思考法」は、ふだんの開発、プログラミングをするうえでの武器の1つになります。

本特集では、数学の勉強法や数学的思考法の身に付け方から、機械学習や物理計算のプログラミング、さらには関数型プログラミングや数式をきれいに表示する方法まで、幅広いトピックでITエンジニアにとっての数学を掘り下げます。昔から数学が苦手だという方も、これを機に数学の世界へ足を踏み入れてください。

第1限目

プログラマ視点の「数学の学び方」

数学とプログラミングの意外な関係?

Author 中井 悦司　グーグル・クラウド・ジャパン合同会社
Twitter @enakai00

プログラミングをするとき、みなさんはどんな考え方で実装を進めていますか? これまでとくに意識をしたことがないかもしれませんが、そこには2つの思考方法が関係していそうです。数学を学ぶときにもこの2つを意識すると、自分の勉強スタイルが見えてくるかもしれません。

IT製品の営業戦略と数学の勉強法?!

本書の読者にはIT業界に関わる方が多いと想像されますが、ITの世界で(とくに営業職の方から!)よく耳にするのが「その製品のユーザ事例はありませんか?」というセリフです。きっと、新しい製品を売り込む際に、「このユーザは、こんなふうに便利に使っています!」と具体例で説明すれば、その製品の価値をわかりやすく伝えられると期待しているに違いありません。

一方、製品の中身をよく理解しているエンジニアは、ここで不思議に思います。「このユーザでうまくいったからと言って、どうして、次のユーザでもうまくいくと言えるのだろう?」「まずは、個々のユーザ環境に依存しない、一般的な機能・しくみを正しく説明するべきなのでは?」——確かに一理ありそうです。すべてのユーザは、それぞれに環境も事情も異なります。特定のユーザで成功したからと言って、手放しに同じことが他のユーザに適用できるとは言えません。

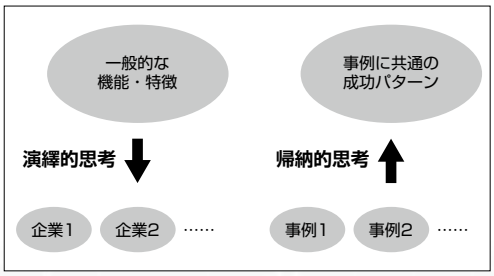
とは言え、一般的な機能説明だけを聞いて、それを自社の環境と照らし合わせて、自社の課題解決にどう役立てられるのか、そこまでの想像力・理解力を自発的に発揮してくれる都合の良い(?)お客様ばかりではありません。まずは、とにかく具体例を並べて、そこから逆に、一般的な成功パターンを理解してもらうというのもひとつの営業戦略なのかもしれません。

……「数学」がテーマの特集で、どうしていきなり営業戦略の話がでてくるのでしょうか?

実は、ここには、「演繹的思考」と「帰納的思考」の違いが隠されています(**図1**)。演繹的思考というのは、個々の環境に依存せず、すべての場合にあてはまる一般的なルール、すなわち、原理・原則をおさえたうえで、それを個別の場合に適用していくという頭の使い方です。先の例であれば、エンジニアの発想がこれにあたります。エンジニア視点で製品の一般的な機能を説明した場合、その話を聞くお客様も同様の思考方法の持ち主であれば、きっとその説明はうまくいくでしょう。

一方、帰納的思考というのは、少数の具体的な事例を徹底的に調べ上げて、その中から、より広く一般的にあてはまる「成功パターン」を

▼図1　演繹的思考と帰納的思考



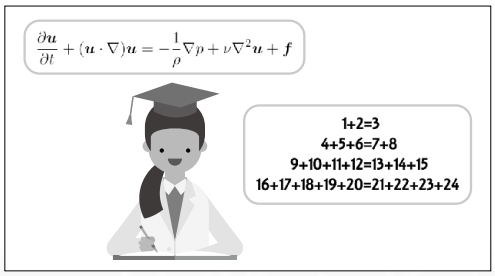
見つけ出そうという発想です。もちろん、いつも正しい結論が得られるとは限りません。限られた例から、本当の意味での原理・原則を見つけ出すには、広く想像力を働かせる必要があり、そして、見つけ出したパターンが正しいかどうかを実際に検証する行動力も求められます。製品説明を受けるお客様がこちらの思考方法の持ち主であれば、ユーザ事例から入るという先ほどの営業戦略もあながち間違いではありません。お客様と一緒に事例を分析して、そのお客様にもあてはまる成功パターンを見つけていく――理屈一辺倒のエンジニアには、なかなかまねのできないスタイルです。

演繹的思考と帰納的思考の組み合わせ

数学を勉強する際にも、この演繹的思考と帰納的思考の切り替えがポイントになります。たとえば、数学者が大発見をする映画の1シーンを想像してみましょう。数学者と思わしき人物が、紙の上に何かを書きなぐっています。さあ、そこに書かれているのは何でしょうか？――映画監督の趣味にもよるでしょうが、2つのパターンがありそうです（**図2**）。

1つは、具体的な数字を書きならべて、そこに何かの法則性を発見しようとする様子です。一見すると、未知の暗号の解読に取り組んでいるようにも思われます。これは、「帰納的思考」と言えるでしょう。具体例の中から、まだ誰も気づいていない、一般的な法則を発見するという発想です。

▼図2　数学者が書いているものは何？



そしてもう1つは、謎めいた文字と記号を書き連ねていき、最後に、たった1つのシンプルで美しい方程式にたどり着くパターンです。これは、「演繹的思考」と言えるでしょう。すでに知られている定理・公式を組み合わせることで、まだ誰も気づいていなかった、新しい定理を導きだそうという発想です。

そして、すでに誰かが発見した定理・公式を学ぶ場合でも、この2つの考え方をうまく切り替える必要があります。筆者の体験を振り返ると、数学の勉強が得意な方は、一般に、演繹的思考が得意のような気がします。まずは、一般的な定理をそのままの形で理解して、それを具体例にあてはめることで、「うんうん。そうだよね。そうなるよね」と悦に入ります。

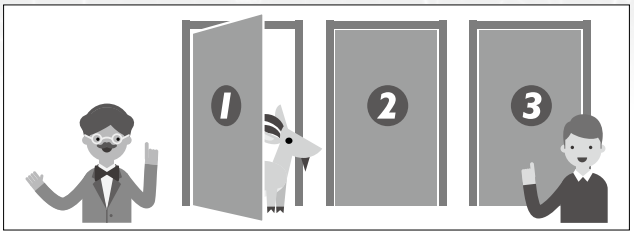
一方、数学はちょっと苦手……という方は、もしかしたら、帰納的思考が得意なのかもしれません。数学の教科書によく見られる、簡単な例を手早く済ませて、すぐに一般論に入るというスタイルではなく、まずは、それなりに複雑な具体例をじっくりと調べ上げて、対象物の性質をよく知ってから一般論へと進めば、きっと理解が深まるに違いありません。

そして、この具体例を調べる際に役立つのが……そう（！）プログラミングです。Mathematicaなど、数学に特化した数式処理ソフトウェアもありますが、Pythonなどの一般的なプログラミング言語の中にも、数学に対応する要素はたくさんあります。あるいは、Jupyter Notebookを使えば、数値計算やグラフの描画も簡単です。紙と鉛筆だけで数学を学んだ時代に比べると、プログラミングを活用して、数学を具体的に楽しむ方法は格段に増えています。このあとは、いくつかの問題を「具体的に実行しながら」味わってみたいと思います。

モンティ・ホール問題で遊ぶ

さて、いきなりですが、確率の有名な問題「モンティ・ホール問題」を考えてみます。モンティ

▼図3　ゲームショーの様子



氏が司会をつとめるゲームショー番組の1シーンがその舞台です（**図3**）。

ゲームの挑戦者の前には、閉まったドアが3枚あります。そのうちの1つは当たりで、賞品の自動車が隠されています。残り2つのドアの後ろには、「はずれ」を意味するヤギがいます。プレーヤが1つのドアを選択して、いよいよドアを開けようかというところ、司会のモンティは、残りのドアの1つを開けて、そこにヤギがいることを見せました。ここで、モンティは、いまなら選択を変えて、もう1つのまだ開いていないドアに変更してもよいと挑戦者に告げるのです。このとき、挑戦者は、選択を変えるべきでしょうか？

これは、内容としては、確率論の問題です。最初に選択したドアが当たりである確率と、変更したドアが当たりである確率、どちらの確率が大きいかを計算すれば答えがでます。「演繹的思考」派のみなさんは、さっそく、紙と鉛筆で計算をはじめているかもしれません。そちらのみなさんにヒントを差し上げると、これは「ベイズの定理」で計算することができます。モンティがドアを開ける前の事前確率に対して、モンティがドアを開けたという事象を観測したあとの事後確率を計算すれば答えが得られます。

……おおっと。ここでそっとページを閉じないでくださいね。もちろん、本稿の主題は、ベイズの定理の解説ではありません[注1]。ここでは、ぜひ、この問題を「帰納的思考」派のみなさんであれば、どのように解くのかを考えてください。先ほど説明したように、帰納的思考の基本は「具体例」です。まずは、実際にこのゲームショーに参加して、選択を変える場合と変えない場合、それぞれを試してみればよいのです。もちろん、数回試しただけではだめです。1万回ぐらい参加して、どちらの場合のほうが当たりになる割合が大きいかを確認すれば、ほぼ確実に答えがわかるでしょう。

……えーっと。すいません。ここでもまた、そっとページを閉じないでください。もちろん、実際にゲームショーに参加するわけではなく、プログラミングを駆使するのです。このショーと同じゲームをアプリケーションとして作成して、何度も何度も繰り返しプレイしてみます。プレーヤとして取るべき行動は決まっていますので、プレーそのものを自動化したシミュレーターを作れば、1万回だろうと、100万回だろうと簡単に実施することができます。

リスト1は、筆者が実際にPythonで作成したサンプルです。それほど複雑なコードではありませんが、念のためにポイントだけ説明しておきます。6行目からはじまる関数gameは、1回だけ、このゲームショーを実施します。引数change_doorには、ドアの選択を変更するかどうかをTrue/Falseで代入します。7行目と8行目では、正解のドアと最初に選択するドアを乱数で決定しています。そして、10行目～13行目では、モンティが開くドアを決定しています。ここでは、未選択で、かつ、正解ではないドアの1つをランダムに選択します。そのあとは、変数change_doorがTrueの場合はドアの選択を変えて、最後に、最終的な選択が正解であれば、得点1を返すという流れです。32行目～38行目は、ドアを変更する場合としない場合、それぞれ、1万回ずつゲームを行って、最終的な勝率を表示します。

注1）そうは言っても、ベイズの定理を用いた計算が気になる方は、筆者のブログを参考にしてください。「モンティ・ホール問題をまじめに計算してみる」
URL http://enakai00.hatenablog.com/entry/20130405/1365118945

▼リスト1　モンティ・ホール問題のシミュレータ

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from random import randint

def game(change_door = False):
  prize = randint(0,2)   # 正解
  choice = randint(0,2)  # 最初の選択

  while True:            # モンティが開くドアの決定
    open_door = randint(0,2)
    if open_door != choice and open_door != prize:
      break

  if change_door:        # 選択を変更する場合
    for new_choice in range(3): # 未選択でまだ空いていないドアを選ぶ
      if new_choice != choice and new_choice != open_door:
        break
    choice = new_choice

  if choice == prize:    # 正解なら1点追加
    return 1
  else:
    return 0

def play(num, change_door):
  point = 0.0
  for _ in range(num):
    point += game(change_door)
  return point

num = 10000

score = play(num, False)
print( u'変更しない場合の勝率\t', score/num)

score = play(num, True)
print( u'変更した場合の勝率\t', score/num)
```

筆者が実行した結果は、次のようになりました。

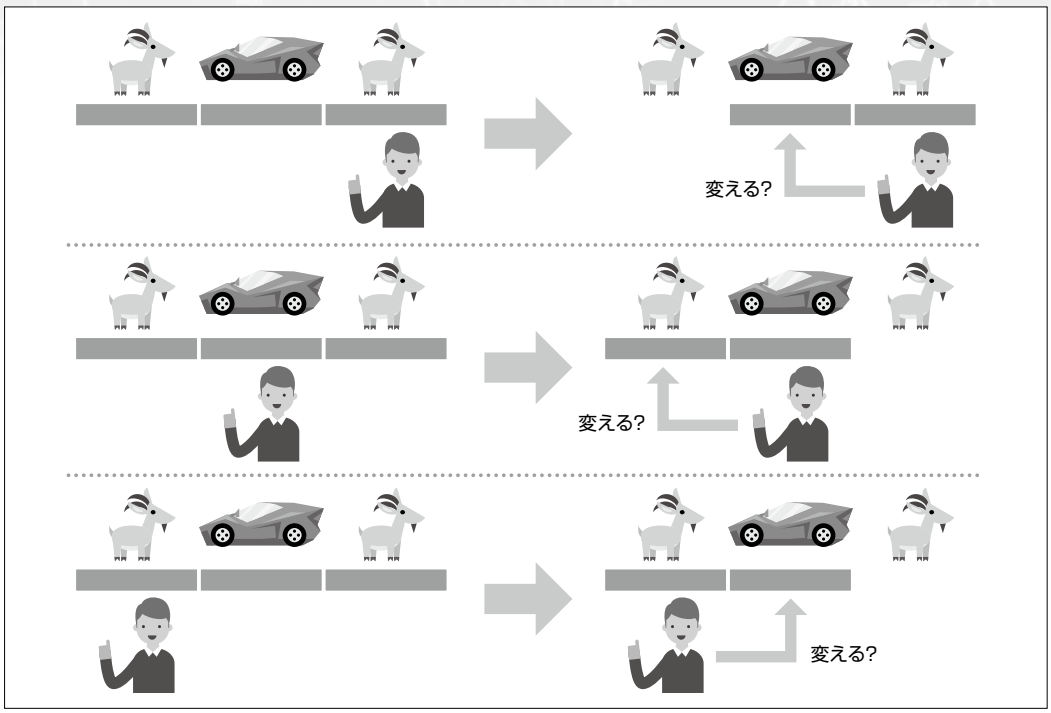
```
変更しない場合の勝率　　0.3358
変更した場合の勝率　　0.6706
```

これで答えは明らかです。ドアを変更しない場合の勝率は約34%、およそ3回に1回当たるということなので、直感とマッチする結果です。そして、ドアを変更した場合の勝率は約67%で、なんと、ほぼ2倍の勝率になっています。ドアを変更するほうが、当たる確率はずっと高くなるのです。

モンティ・ホール問題の真相

モンティ・ホール問題は、ベイズの定理で計算できると言いましたが、ベイズの定理とほぼ同じ内容をぐっとわかりやすく説明すると、**図4**のようになります。真ん中のドアが正解だとした場合、プレーヤが選択するドアには、3つのパターンがあります。何も考えずに、このままドアを開ければ、正解する確率は1/3です。このあと、モンティがほかのドアを開けたとしても、そのまま選択を変えなければ、やはり、正解する確率は1/3のままです。これが、先ほどのシミュレーションによる、勝率34%の真

▼図4　3種類のドアの選択



相です。

一方、ドアの選択を変えた場合、もともとが当たりだった場合は、残念ながら、逆にはずれになってしまいます。ところが、もともとがはずれだった場合は、選択を変えることにより、逆に当たりになります。これにより、正解と不正解の結果が入れ替わり、正解する確率は、もともとの不正解の確率である、2/3になります。これが、ドアを変更すると勝率が2倍に増える原因だったのです。

先ほどのシミュレーション結果を見たみなさんであれば、「なるほど。確かに！」と納得したでしょう。しかしながら、1990年にはじめてこの問題が取り上げられたとき、ある数学者が「ドアを変更すると正解する確率は2/3に増える」という解説をしたところ、その結論に納得がいかない人々からの反論が巻き起こったそうです。反論する人々の中には、なんと多数の数学者も含まれていたのです。結局のところ、別の数学者が自前のパソコンでシミュレーションを実施して、まさに帰納的思考によって、この論争に決着をつけたのです。

ただし、反論した数学者の名誉のために付け加えておくと、この問題は状況設定に少し不備があります。先のシミュレーションでは、「プレーヤが選んだドアにかかわらず、モンティは必ずはずれのドアを1つ開ける」という実装になっています。しかしながら、現実のゲームショーの中では、そのような説明はありません。モンティがドアを開けるかどうかは、あくまで、モンティの自由だと仮定しましょう。仮に、「プレーヤが正解のドアを選んだときだけ、モンティはわざとはずれのドアを開ける」とすれば、当然ながら答えは変わります。

これは、「演繹的思考」の落とし穴と言えるかもしれません。具体例を使わずに議論をしていると、人によって想定する前提条件に違いがあって話がかみあわなくなるのは、数学の世界に限らずよくある話です。具体例を調べることによって、前提条件の不備や見落とし、あるい

は、自分自身の思い込みに気づくことができます。モンティ・ホール問題の場合、その問題をプログラムコードとして実装することにより、暗黙の前提条件が明確になり、無事に人々の誤解が解けたというわけです。

再帰処理は演繹的？ 帰納的？

「数学とプログラミング」というテーマで、まっさきに思いつく例の1つが「再帰処理」です。よくある例ですが、次の階乗計算を考えます。

・1!＝1
・2!＝1×2＝2
・3!＝1×2×3＝6

一般の自然数nに対して、n!を計算する関数fact(n)を実装する場合、素朴にループで実装すると、こうなります。

```
def fact(n):
  result = 1
  for i in range(1,n+1):
    result = result * i
  return result
```

一方、n!を計算するには、「(n-1)!を計算しておいて、それにnを掛ければいいじゃん」と気づいて再帰的に実装したのがこちらです。

```
def fact(n):
  if n == 1:
    return 1
  return fact(n-1) * n
```

では、現実のプログラムでは、どちらのコードを採用するべきでしょうか？　真面目に言うと、答えはどちらでもなくて、Pythonであれば、ライブラリ関数math.factorial(n)を使うのが正解です。上記のコードは、あくまで説明用のサンプルで、nが自然数であるかなど、必要なエラーチェックがなされていませんし、ライブラリ関数のほうが性能的にも最適化されているでしょう。また、上記の2つの例は、実際に行われる計算処理はほとんど同じです。あえて言うなら、再帰処理のほうは、関数をネストして呼び出す分だけメモリを余計に使うというデメリットがあります。シンプルなループで処理できる内容であれば、無理に再帰処理を使わなくてもよいでしょう。

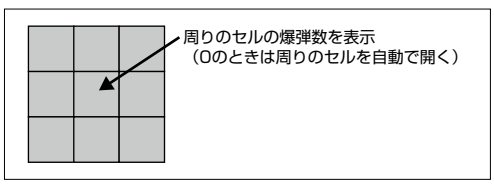
現実のプログラミングで再帰処理が役に立つのは、「繰り返し処理をどう書けばよいのか、もはや自分の頭では追いかけられない」というシーンです。たとえば、みなさんご存じのマインスイーパー[注2]で、「あるセルを開いて、そこに爆弾がなかったとき」の処理を考えてみてください（**図5**）。この場合、

❶周りの8個のセルにある爆弾の数を計算して、開いたセルに表示する
❷爆弾の数が0の場合は、周りの8個のセルも開く

という処理を行う必要があります。ここで気がつくのが、セルを開くという処理の中に、セルを開くという処理❷が入れ子になっている点で

注2）万一マインスイーパーをご存じない方は、Wikipediaを参照ください。
URL https://ja.wikipedia.org/wiki/マインスイーパ

▼図5　マインスイーパーのセルを開く処理



▼リスト2　セルを開く処理のコード例

```
def open_cell(x,y):
  num = mines_around(x,y)  # 周りの爆弾数
  show_num(x,y,num)        # 爆弾数を表示
  if num == 0:             # 0のときは周りのセルを開く
    for dy in [-1,0,1]:
      for dx in [-1,0,1]:
        if dx == 0 and dy == 0: # 自分自身はスキップ
          continue
        open_cell(x+dx,y+dy) # 隣のセルを開く（再帰処理）
```

す。これは、明らかに再帰処理の出番です。**リスト2**は、座標（x,y）のセルを開く処理を実装したコードの例です（境界条件などのチェック処理は省略しています）。これと同等のコードを再帰処理を使わずに、ループだけで実装するのは、なかなかの困難が予想されます。

ただし、**リスト2**のような再帰処理を用いたコードは、見かけはスッキリしていますが、実際にどのように動作するのかは、なかなか一般人の頭では追いかけきれません。筆者の場合、「理屈上はこれでうまくいくはず……」と信じて、まずはコードを実行してみます。だいたいは、細かな境界条件の設定を間違っていて、予想外の動きに驚くのですが、テストと修正を繰り返して、やがて、期待どおりの処理が実現できることになります。

実は、このような再帰処理のテスト／修正の作業をする際に、いつも筆者の頭に浮かぶのは、「演繹的思考」と「帰納的思考」のせめぎ合いです。先ほど「理屈上はこれでうまくいくはず……」と言いましたが、これはまさに演繹的思考です。本気で演繹的思考を突き詰めるのであれば、この再帰処理が正しく動くことを数学的帰納法で証明することもできなくはないでしょう。しかしながら、そこまでの労力をかけるのはたいへんなので、「まずはテストする」という帰納的思考に頭を切り替えます。テスト結果をていねいに観察することで、条件の見落としや勘違いに気づいて、それを修正していくというわけです。

ここで、どのタイミングで演繹的思考と帰納的思考を切り替えるかは、重要なポイントになります。理屈がまったく定まらないまま、やみくもにコードを書いて試してもうまくいきませんし、かと言って、コードを書かずに理屈だけで考えていても、先に進まないこともよくあります。この2つの思考方法を適切に組み合わせられることが、実は、優れたプログラマの資質であり、数学を学習する際のポイントではないでしょうか？

数学とプログラミングの間には、直接的なつながりもたくさんありますが、それよりも、数学の学習を通して、この「理屈と具体例のせめぎ合いの妙」を会得することが、プログラマにとっての数学の大きな価値なのだと思います。

アナグラムファインダー

少し抽象的な話が続いたので、最後に数学がヒントになる例題をもう1つだけ紹介しておきましょう[注3]。「skate」と「stake」、あるいは、「reset」と「steer」のように、文字を並べ替えると一致する単語をアナグラムと言います。いま、1行に1つの英単語が記載された辞書ファイルがあるとして、互いにアナグラムになっている単語をグループ化して、1行に1つのグループを集めたテキストファイルを作成してください。辞書ファイルの例としては、GitHubで公開されている、「words_alpha.txt」を使用します[注4]。

――いま、読者のみなさんは、頭のなかでさまざまな試行錯誤をしていることでしょう。2つの単語を比較して、アナグラムになっているか判定する処理は、なんとか書けるとしても、大量にある単語について、すべてを個別に比較するとなると、これはたいへんです。ここで、数学が得意な方のためにヒントとなるキーワードを差し上げると、「アナグラムは同値関係」と「同値類の代表元（代表ラベル）を活用する」です。これにより、個別に単語を比較する必要がなくなります。さらにヒントを与えると、筆者の作った解答例は、2つのPythonスクリプトから構成されており、次のように、sortコマンドをパイプで挟んで実行します。

```
$ ./map.py < words_alpha.txt | sort | ➥
./reduce.py > output.txt
```

注3）この問題は、「Programming Pearls（2nd Edition）」Jon Bentley（著）より引用させていただきました。

注4）english-words
URL https://github.com/dwyl/english-words

さあ、これは、いったいどういうことでしょうか？　順を追って解説してきましょう。まず、同値関係というのは、かたい言い方をすると、

・A～A
・A～B⇒B～A
・A～BかつB～C⇒A～C

という性質を満たす関係性です。たとえば、「A～B」を「AさんはBさんを知っている」という関係だとします。普通はそんなことはありませんが、仮に、「AさんがBさんを知っていれば、BさんもAさんを知っている」「AさんがBさんを知っていて、BさんがCさんを知っていれば、AさんはCさんを知っている」というルールが成り立つとします。すると、世界のすべての人々は、「お互いに知っている人どうしのグループ」にきれいに分類することができます。このようなグループを「同値類」と呼びます。

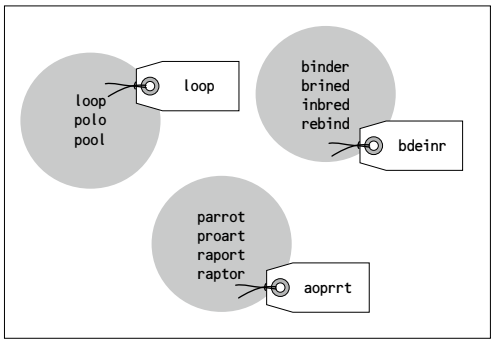
いま、「A～B」を「単語Aの文字を並べかえると単語Bに一致する」という関係だとすると、これは、先ほどの同値関係の条件を満たします。つまり、すべての単語は、お互いにアナグラムとなる単語のグループにきれいに分類することができるのです。そして、このような同値類のグループからは、それぞれの集合を代表する要素を1つ取り出すことができます。これが「代表元」です（**図6**）。

どの要素を取り出してもよいのですが、なるべく代表らしい標準的なものがよいでしょう。お互いにアナグラムになっている単語は、文字の並びをソートすれば、すべて同じ表現になります。たとえば、「loop」「polo」「pool」は、アルファベットの昇順にソートすると、すべて「loop」になります。そこで、それぞれの同値類について、このようにソートして標準化した単語を代表元として選択します。ソートしたものがそのグループに含まれる単語になっていない場合もありますが、ここでは、あくまでその同値類のラベルとして使用するだけなので、そこは気にしないことにします。

この代表ラベルの都合の良い点は、任意の要素から、それをソートするだけで代表ラベルが得られるということです。たとえば、辞書ファイルを1行ずつ読み込みながら、各単語をソートして、代表ラベルを行頭に付加することができます。これが先ほどの「map.py」の処理内容です。そして、この出力をsortコマンドでソートするということは……。そう！　もうおわかりのように、行頭に代表ラベルが付加されているので、これをソートすれば、同じ代表ラベルを持つ行が連続して並ぶのです（**図7**）。

最後は、「reduce.py」によって、この出力を

▼図6　アナグラムの同値類と代表ラベル



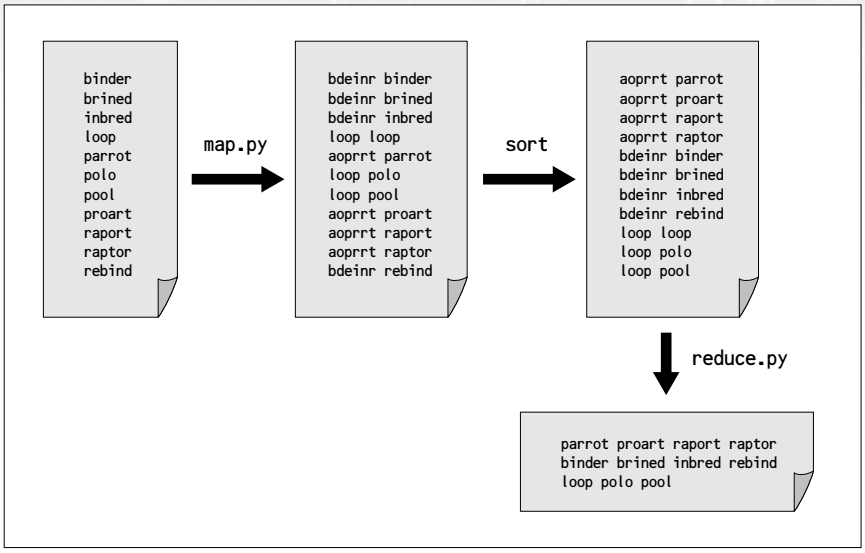
▼リスト3　map.py

```
#!/usr/bin/env python3
import sys

for line in iter(sys.stdin.readline, ''):
  word = line.rstrip()
  canonical = ''.join(sorted(word))
  print (canonical + ' ' + word)
```

▼リスト4　reduce.py

```
#!/usr/bin/env python3
import sys

pre_canonical = ''
result = []
for line in iter(sys.stdin.readline, ''):
  canonical, word = line.rstrip().split(' ')
  if pre_canonical != canonical:
    if pre_canonical != '':
      print (' '.join(result))
    pre_canonical = canonical
    result = []
  result.append(word)
```

▼図7　アナグラムファインダーの動作原理



1行ずつ読み込みながら、代表ラベルが同じものを1行にまとめて再出力すれば完了です。**リスト3**と**リスト4**は、その実装例ですが、手元のMacで実行したところ、約37万行の辞書について約2秒で実行が終わりました。スクリプト名からも想像されるように、sortコマンドをはさんだこの実装は、実は、MapReduce（Map-Shuffle-Reduce）の処理パターンに一致しています。MapReduceの要となるShuffle処理を（性能的にすぐれた）sortコマンドに外出しすることで、全体の性能を向上させることに成功しているのです。

まとめ

本稿では、「演繹的思考」と「帰納的思考」という切り口で、数学の勉強法について考えてみました。わかりやすく言うと、「理屈で考える」ことと「実際に試してみる」ことをうまく組み合わせていきましょう、ということです。モンティ・ホール問題は、プログラミングによって「実際に試してみる」ことで、理屈だけの議論で見落とされていた条件が明確になった例と言えます。マインスイーパーの実装は、「理屈ではうまくいくはず」と当たりをつけたうえで、実際に実行して条件の見落としを見つけていくという、プログラミングの思考過程の好例です。

そして、最後のアナグラムファインダーは、どうでしょうか？　最後の実装例だけを見ると、「なるほど！　こんなシンプルな方法があったか！」と感じるものの、同値関係などの説明は、「そこまで大げさな理屈をつけなくても……」と思うかもしれません。しかしながら、それは、シンプルな答えを見たあとだから言えるのであって、何もヒントがないところからシンプルな答えを見つけるのはたいへんです。同値関係といった、数学の一般的な概念を知っているからこそ、アナグラムに隠された性質を見抜き、驚くほどシンプルな答えを導くことができるのです。

数学を学習する際は、教科書に書かれた一般論をそのまま読むのではなく、まずは、紙と鉛筆を用意して、ぜひ、自分の手で計算をしてください。具体例を徹底的に調べ上げる中で、一般論に隠された、さまざまな事実が浮かび上がることでしょう。この体験こそが、数学を理解するプロセスであり、プログラミングを始めとする現実の問題へと応用する第一歩となるでしょう。SD

第2限目

数式が怖いならコードで理解

機械学習の難解な数式をひもとく

Author 真嘉比 愛（まかび あい） ちゅらデータ株式会社
Twitter @a_macbee URL https://churadata.okinawa

機械学習の関連論文や参考書を読むには数式を読み解く力が求められますが、数式からすぐにはイメージがわかないかもしれません。そんなときは、数式をコード化すれば振る舞いを見ることができます。プログラマならそのほうが近道かもしれません。

データ分析における数学の重要性

GoogleのAlphaGoに端を発し、AIという言葉がメディアで盛んにとりざたされるようになってから、データサイエンス未経験でも機械学習や人工知能を使ってみたいという方が増えてきました。実際に、Azure Machine Learning StudioやGoole Prediction APIといった機械学習サービスを利用すれば、難しい理論を理解せずとも簡単に予測モデルや推薦モデルを作って試すことができます。「とりあえず機械学習を使ってみる」ことは、以前に比べ非常にハードルが低くなりました。

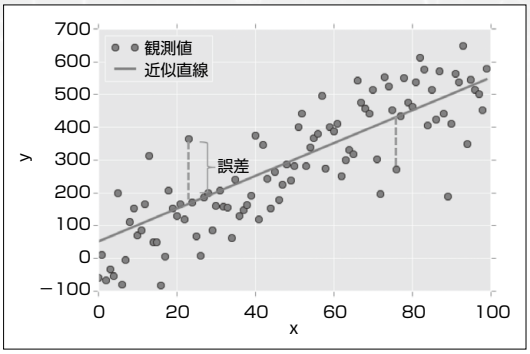
では、とりあえず機械学習を試せるような環境があれば、機械学習の理論やしくみを理解せずとも機械学習を使いこなせるようになるのでしょうか？　いいえ、そうではありません。機械学習を利用したサービスやプロダクトを作っていく過程では、作成した機械学習モデルの改善（＝最適なアルゴリズムに変更したり、パラメータを最適化したり、モデルに投入するデータに適切な前処理を施したり、etc.）が必ずといってよいほど必要になってきます。そしてそのような改善を施すためには、利用している機械学習アルゴリズムについてその振る舞いを理解しておく必要があります。アルゴリズムを理解していれば、モデルの出力結果を正しく解釈し、精度改善に必要なアプローチをとることができます。

数式をイメージすることの難しさ

機械学習を使いこなすためには、その裏にある理論やしくみを理解することが重要であるとわかりました。より具体的に言うと、数式で表現された機械学習アルゴリズムの振る舞いを理解する必要があります。わかりやすくイメージするために、ここでは最急降下法（Gradient descent method）というアルゴリズムを例に見てみましょう。最急降下法というのは、関数の最小値を求める最適化手法の中でも、最も基本的なアルゴリズムになります。どういった場面で使うアルゴリズムなのか、サンプル問題を例に説明します。

まずはアルゴリズム解説によくある数式で説明します。難解に感じられるかもしれませんが、ひとまずこらえて読み進めてください。**図1**に示される点（ここでは便宜的に観測値と呼びます）を近似する直線を求めることを考えます。直感的には、図中の直線で示されるような近似直線がイメージできるはずです。観測値から最適な近似直線を求めるにはどうすれば良いでしょうか？　図中の観測値と近似直線を結ぶ破線を見てください。この破線は、観測値と近似直線のずれ（ここでは便宜的に誤差と呼びます）を

▼図1　サンプル問題のイメージ



表しています。最適な近似直線を引くことができれば、各観測値と近似直線との誤差は小さくなるはずです。この性質を利用して、最適な近似直線を求める問題を「各観測値と近似直線の誤差の総和が最も小さくなるような近似直線を求める」問題として定式化します。

観測値をd_n、近似直線を$f(x_n; w) = x_n \cdot w$（パラメータwは近似直線の傾きを表す）としたとき、各観測値と近似直線の誤差の度合い$E(w)$を下式のとおり表すことができます。

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}(d_n - f(x_n; w))^2 \quad \text{—(式1)}$$

観測値と近似直線の誤差には正負があるため、二乗することで誤差を正の値に統一しています。関数$E(w)$は、機械学習の分野で誤差関数と呼ばれる値です。誤差関数$E(w)$を最小化するパラメータ$\hat{w} = arg_w min\ E(w)$を見つけることで、最適な近似直線$f(x_n; \hat{w}) = x_n \cdot \hat{w}$を求めることができます。

最小化したい誤差関数$E(w)$を定義することができたので、さっそく、最急降下法を利用して誤差関数を最小化するパラメータ$\hat{w}$を探してみましょう。最急降下法のアルゴリズムは次のとおり定義されます。

①乱数などを利用して$w^{(1)}$を適当に決める
②次式を繰り返し計算し、$w^{(2)}$、$w^{(3)}$、……を求める

$$w^{(t+1)} = w^{(t)} - \epsilon \frac{dE(w)}{dw} \quad \text{—(式2)}$$

※εは学習係数とよばれるハイパーパラメータで、wの更新量を決める定数
③wの変化が一定値以下になったら終了する。最終的に得られるwが誤差関数を最小化するパラメータ$\hat{w}$になる

最急降下法は、特定の式を繰り返し適用することでパラメータwの値を更新し、最終的に最適な近似直線を与えるパラメータ$\hat{w}$を求めます。このアルゴリズムではパラメータwの更新式（式2）がキーとなってきます。

しかしこの更新式を見て、具体的にどのような処理が行われるのかイメージできるでしょうか？　学習係数と呼ばれる値と誤差関数をwで微分した値をかけて、それをwからひいた値を次のwとして定義しています。学習係数εとはどのような役割を果たすハイパーパラメータなのでしょうか？　誤差関数をwで微分した値とはどのような性質をもっているのでしょうか？

数学の基礎知識がなければ、式を見ただけでどのような処理が行われているのか直感的に理解するのは困難です。機械学習の理論やしくみを理解しようとするうえで、初学者が最もつまずきやすいポイントの1つが、この数式理解の部分と言えるでしょう。

本記事では、このような数式を理解する一つの手助けとして、数式をプログラムで実装して視覚的かつ直感的に機械学習モデルの振る舞いを理解する方法を紹介します。

数式をプログラムで書いて理解する

最急降下法のアルゴリズムがどういった振る舞いをしているのかを理解するために、アルゴリズムで定義されている数式をプログラムで実装し、実際の挙動を確認してみましょう。ここでは、プログラミング言語としてPythonを利用します。

まずは、サンプル問題を定義します。

```
# Python 3.6.2を利用
# 必要なライブラリのimport
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 99, 100)
 # ↑0から99までの100個の点
f_noisy = np.vectorize(lambda x: x * 10 +
+ np.random.normal(0, 100))
d = f_noisy(x)  # 観測値
```

0から99までの100個の点xについて、$y=10x$の式にノイズを乗せた観測値dを作成しました（ノイズは平均0、標準偏差100の正規分布に従っています）。ここで指定されているパラメータw=10が求めたい真の値になります。xとdをグラフで描画すると、**図1**の点で示されるような観測値を確認することができるはずです。

同様に、近似直線$f(x; w)$と誤差関数$E(w)$についても実装します（**リスト1**）。

ここで、最小化させたい誤差関数$E(w)$がどのような関数なのか確認してみましょう。パラメータwの値を変化させたときの誤差関数$E(w)$の変化について、グラフで描画して確認します。

```
ws = np.linspace(-5, 20, 25)
 # ↑パラメータ w を-5から20の間で変化させる

plt.plot(ws , E(ws), '-')
plt.xlabel('パラメータ w')
plt.ylabel('誤差関数 E(w)')
plt.show()
```

パラメータwの値を変化させたときの誤差関数$E(w)$の変化を**図2**に示します。両者の関係をグラフで描画したことで、さまざまなことがわかりました。まず、今回最小化したい誤差関数$E(w)$はパラメータwに対して下に凸な関数であることがわかります。誤差関数$E(w)$はパラメータwの真の値w=10のあたりで最小値0をとっています。最急降下法を利用して、パラメータwをw=10に近い値に近づけていくことが目標です。

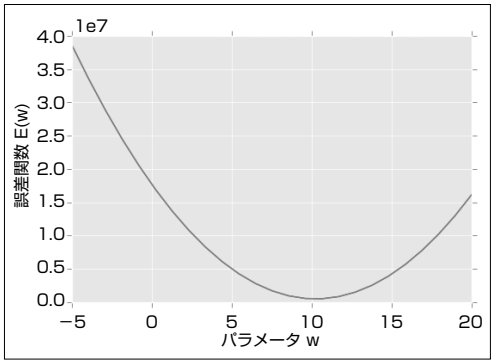
このように、定義した誤差関数について最適化したいパラメータとの関係をグラフで描画して確認することで、最適化のイメージを確認することができます。

近似直線$f(x; w)$と誤差関数$E(w)$を実装できたので、最急降下法のアルゴリズムを実装してみましょう。最急降下法のアルゴリズムを実装するために、誤差関数$E(w)$の微分を次式のとおり求めました。

$$\nabla E(w) = \frac{\mathrm{d}E(w)}{\mathrm{d}w} = \sum_{n=1}^{N} x_n^2 w - \sum_{n=1}^{N} d_n x_n \quad \text{—(式3)}$$

最急降下法のアルゴリズムを関数として実装します（**リスト2**）。そのまま実装すると、収束条件を満たせない場合無限ループに陥ってしまいます。そこで、ここでは最大繰り返し回数iter_maxを定義し、収束条件を満たせなかった場合でも処理が終了するように実装してあります。epsilonに適当な値を渡して、wがどのように変化するのか確認してみましょう。

▼図2　パラメータwの値を変化させたときの誤差関数E(w)の変化



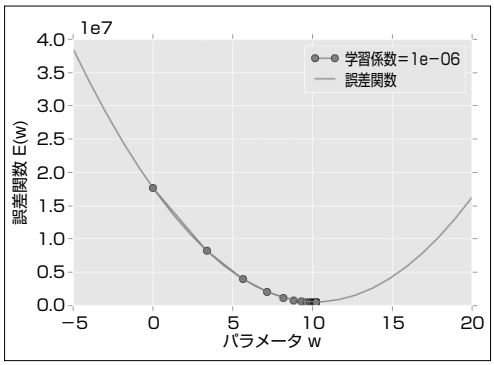
▼リスト1　近似直線と誤差関数の実装

```
f = np.vectorize(lambda x, w: x * w)  # 近似直線
E = np.vectorize(lambda w: 1/2 * sum([(i - j) ** 2 for i, j in zip(d, f(x, w))]))  # 誤差関数
```

▼リスト2　最急降下法の実装

```
def gradient_descent(epsilon, error=1e-8, iter_max=1000):
    """
    esilon: 学習係数
    error: 収束条件
    iter_max: 最大繰り返し回数
    """

    w = np.random.randint(-20, 20)   # 1. w の初期値を決める
    w_history = [w]

    for _ in range(iter_max):
        w_next = w - epsilon * (sum(x * x * w) - sum(d * x))  # 2. wの値を更新する
        if abs(w_next - w) < error:  # 3. wの変化がerror以下になったら処理を終了する
            break
        w = w_next
        w_history.append(w)

    return w, w_history
```

▼図3　最急降下法におけるパラメータwの変化




```
w, w_history = gradient_descent(1e-06)

plt.plot(w_history, E(w_history), 'o-', ↵
label='学習係数=1e-06')
 # ↑最急降下法におけるwの変化
plt.plot(ws, E(ws), label='誤差関数')
 # ↑誤差関数のプロット
plt.legend()
plt.xlabel('パラメータ w')
plt.ylabel('誤差関数 E(w)')
plt.show()
```

図3の点で示されているグラフが、ステップごとのパラメータwの値になります。wは初期値で0をとり、その後更新を繰り返すたびにw=10に向けて値を更新しています。最終的に、$\hat{w} = 10.2176$という最適値に近い値を得ることができました。

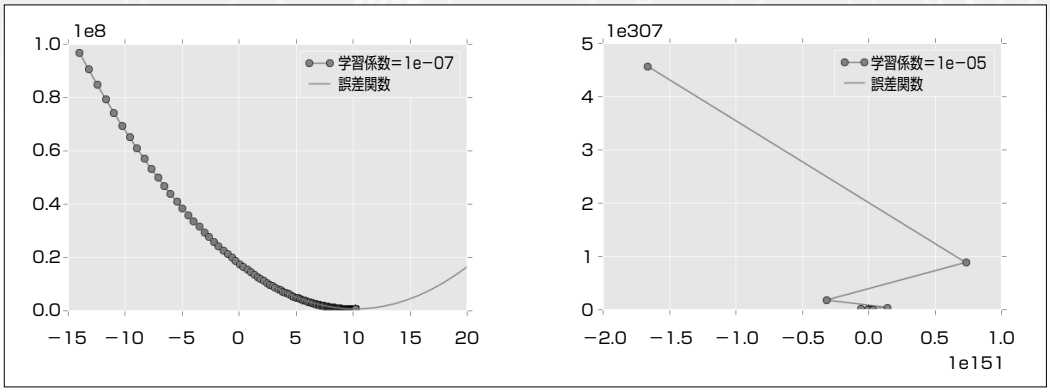
最急降下法では、ステップごとに誤差関数$E(w)$を微分することでその地点における傾きを計算し、誤差が小さくなる方向にパラメータwを更新しています。パラメータwの更新式を見ただけではぴんとこなかった方も、**図3**のパラメータ変化の様子を確認することで、最急降下法がどのようなことを実施するアルゴリズムなのか、直感的に理解できたのではないでしょうか？

さらに、モデルのハイパーパラメータである学習係数εがパラメータwの最適化に対してどのような影響を与えているのかについても確認してみましょう。学習係数εが小さい場合（$\varepsilon = 10^{-7}$）と大きい場合（$\varepsilon = 10^{-5}$）について、誤差関数の値の変化をグラフ化してみます（**図4**）。

学習係数εを10^{-7}とした場合、10^{-6}だったときと比較して、パラメータwの変化幅が小さくなり、時間をかけて緩やかに収束していることがわかります。反面、学習係数εを10^{-5}とした場合は、パラメータwの変化幅が大きすぎてうまく収束できていないことがわかります。この結果から、学習係数εについて次のような性質があることがわかりました。

・学習係数εの値を小さくすればするほど変化幅は小さくなり、より確実にパラメータwの値を収束までもっていくことができるが、

▼図4　学習係数を変化させたときのステップごとの誤差関数値の変化（左：$\varepsilon=10^{-7}$、右：$\varepsilon=10^{-5}$）



反面収束までに必要なステップ数が増え、実行にかかる時間が増大する

・学習係数εの値を大きくすればするほど変化幅は大きくなり、収束するまでのステップを短縮することができるが、あまりにも変化幅を大きくし過ぎるとうまく収束できなくなる

このように、ハイパーパラメータ（今回の場合は学習係数ε）の値を変化させた結果を図示することで、ハイパーパラメータが実行結果にどのような影響を与えるのかを明確化させることができます。

機械学習をとりあえず使ってみるという段階では、作成するモデルのハイパーパラメータがそれぞれどのような意味を持つのかという確認をおろそかにしてしまいがちです。ですが、パラメータの値を少し変更するだけで上記のように結果が大きく変わってしまうこともあり得るため、可能な限りパラメータが与える影響を確認するようにしましょう。数式を見て直感的に理解できない場合でも、上記のように値の変化が与える影響をグラフ化することでパラメータが与える影響を解釈することが可能になります。

まとめ

本記事をとおして、機械学習を利用するうえで、その理論やしくみを理解することの重要性や、そのためにプログラムを使って数式の挙動を確認するテクニックを紹介しました。実際の分析現場でも、直感的に理解しがたいモデルが出てきた場合には、今回紹介したようにパラメータを変化させた場合のモデルの振る舞いを確認して、分析の道筋をたてるといったことを行っています。また、ご自身で数式の入り混じった機械学習の本を読む際にも、説明されている数式をプログラムで実装して動かすということは理解の大きな助けになります（実際に筆者が『データ解析のための統計モデリング入門』[注1]や『パターン認識と機械学習』[注2]などの本を読んで勉強した際には、説明されている数式をプログラムで実装して理解するということを行っていました）。

これまで数学が苦手で機械学習を敬遠されていたという方や、なんとなくライブラリを使ってみていただけという方がいましたら、この機会にぜひ簡単な数式・アルゴリズムから、プログラムで書いて理解するということにチャレンジしていただければと思います。SD

注1）久保拓弥 著、岩波書店 刊、ISBN 978-4000069731

注2）C.M. ビショップ 著、元田浩／栗田多喜夫／樋口知之／松本裕治／村田昇 監訳、丸善出版 刊、上巻：ISBN 978-4621061220、下巻：ISBN 978-4621061244

課外授業 1

ITエンジニアに数学は必要か

パラダイムシフトのたびに起こる議論について

Author 伊勢 幸一（いせ こういち） さくらインターネット株式会社
Twitter @ibucho

IT業界のパラダイムシフトと数学

IT業界で新しいトレンドやテクノロジによって技術のパラダイムシフトが起こるたびに、ITエンジニアには数学が必要かどうかという議論が浮上してきているように感じます。

筆者の周囲で今までに聞きおよぶだけでも、映像の3D化に伴う余弦変換や四元数変換、無理数や超越数の近似値を求める級数展開、P2P/DHTにおける多次元幾何学や位相幾何学、SNSにおけるグラフ理論、そして今は機械学習や深層学習に必要なN次行列計算や分布、確率統計学などがあります。これら数学的根拠をもとに理論が展開され、有効性が示される技術に注目が集まると、プログラマやインフラエンジニアが数学とどう向き合うかについての議論が、どこからともなく繰り返し噴出します。

なぜでしょう？　はたしてITエンジニアに数学は必要なのでしょうか？

数学は（それほど）必要ない？

筆者は工業大学にて一通りの工業数学を履修したとはいえ、数学科の学生のように基本的な代数学、幾何学、解析学、関数論、確率統計などを十分に習得したとは言えず、未知の問題に対して、数学的根拠をもとに解決のための思考を数学的に展開できるほどではありません。文献や書籍に数式が出てきてもアレルギーこそ感じはしませんが、式を見ただけでそれが示す状態や関係をイメージできるほどではなく、その意味を理解するために学生時代の教科書を引っ張り出す始末です。どちらかと言えば、エンジニアの中では数学を苦手としている方に分類されるでしょう。

それでも、曲がりなりにも1980年代末から現在に至るまで約30年近く、このインターネットを取り巻く業界内でITエンジニアとして在り続けることができました。その経歴をもとに言えば「ITエンジニアに数学は（それほど）必要ない」という主張ができないわけではありません。現在のITエンジニアには文系出身の方々も多く、とくに数学や物理学、力学、電気電子工学といった理系分野を苦手としている人も多いでしょう。それでも、IT業界の第一線で活躍している方々が大勢います。

業務と数学

基本的にエンジニアの業務として、自然界にある状態を分析したり、自然現象を制御したりするシステムやプログラムを設計・開発するエンジニアならば、対象に関する自然科学、つまり物理学や化学、数学などとその活用力が求められることは当然です。しかし、現在のIT業界で取り扱う問題要素は、単純な数値やデジタルデータ、テキストデータであり、処理はそれらを整数値化した“群”の操作に限定されるため、整数の加減乗除程度の算術知識があればほとん

どのケースに対応することができます。

また、ネットワークやサーバなどの基盤開発や運用をしているエンジニアであっても、基本的にはメーカーの人が開発製造した機器やソフトウェアを操作したり、他人が開発した言語やフレームワーク、プロトコルをもとにコーディングしたりデバッグしたりする業務がほとんどであり、この段階で自然科学的問題に対処することはほぼないでしょう。

したがって、先人が発案した技術、他人が作った基盤や実装の上で、もしくはそれらを利用して作業する多くのエンジニアにとって、数学を含む自然科学はそれほど必要ではありません。では、その基盤や実装は誰がどうやって作ったのでしょうか。

引き出しとしての数学

数式によって結論への誘導や有効性の根拠が示されている論文、書籍、記事に出会うたび、なぜこの執筆者はこの問題へのアプローチにこの数式が利用できることを思いつくのだろう、と不思議に思います。それに比べて自分は、問題解決のヒントやアイデアに対して数学が活用できないことを痛感します。自分は自然科学知識に乏しいことで、今までにないまったく新しいアイデアや技術、理論、アプローチ方法などを発案することができなかったのだろうと考えてしまいます。

数学に暗くても、ITエンジニアの業務は遂行できます。技術を理解するために必要な数学程度ならば、時間さえかければなんとか追いつくことができるでしょう。しかし、未知の問題に直面したとき、その問題を解くための引き出しとしての数学を持っていないと、経験的に把握している社会科学的知見（要は体験的知識）のみに頼るしかありません。問題解決の道具として数学を持っていたならば、よりエレガントな解法を見つけ出し、よりスマートに問題解決にアプローチすることによって、今とは違ったキャリアを歩めたのかもしれません。

「ITエンジニアと数学」議論をどう見るか

ときどき、業務上のある問題を解かなければならないとき、もしかすると（自分の乏しい知識にある）あの数式を使うと説明できるのではないか？と思い、慌てて学生時代の教科書を引っ張り出したり、書店で数学書を買い求めたりしますが、いつもまず、問題の数式に至るまでの基礎知識を復習しなければなりません。そうしていろいろもがいている間に問題解決の機を逸し、ターニングポイントだったかもしれない重要なチャンスを逃してしまったことが多々あったように思います。結局、エンジニアにとって数学的知識が必要かどうか、その是非を議論することにはあまり意味がないのかもしれません。

しかしなぜ、エンジニアにとって数学は必要かどうかという議論が時折湧き上がるのでしょうか。その理由は、筆者と同じく、基本的に数学を知らなくても業務は遂行できる、が、根本的理論を理解せずして、どこかの誰かが生み出した成果物を利用するだけでなんとか仕事をこなしている自分自身に、不安と自虐と憤りを抱いているから、なのではないでしょうか？　筆者はいまだ、数学をうまく使えない自分自身を、エンジニアとして情けないと常に自嘲しています。

数学に限らず、ある知識や知見がエンジニアにとって必要かどうかを議論してしまうというのは、本能的にそれらが必要であると感じているにもかかわらず、習得する手間、苦労と得られるメリットを秤にかけ、できれば避けて通ってしまいたい根拠が必要だからでしょう。

そうして問題や課題を避けて通ると結局ロクなことにはならないので、必要か不要かを議論するのではなく、とりあえず習得する努力をしたほうがエンジニアとして成長できるのではないでしょうか。

ああ、もっと数学やっておけば良かった。SD

第3限目

数学系エンジニアの思考法

抽象化を心がけていますか?

Author 橘 慎太郎(たちばな しんたろう) 株式会社オルトプラス
Twitter @umekichinano

数式を見るとついひるんでいませんか? 実は案外簡単なことを記号で表現しているだけに過ぎません。エンジニアとして自分の技術で数学を咀嚼(そしゃく)してみるのも解決策の1つです。でも、もっと効果的なのは「抽象化」です。数学は抽象化にあります。これは日常業務でも役立つ考え方です。

Σはこわくない

コーディング思考で理解してみよう

今回の原稿執筆が決まったときに、あるエンジニアの方をふと思い出しました。その人は数学を苦手としている方で、高校時代に習うΣがよくわからないと言っていました。本誌読者の方は覚えている方も多い?——とは思いますが、念のためΣは次のように書くと、a[0]からa[100]まで順番に加えていくという意味になります。

$$\sum_{n=0}^{100} a_n$$

その人に「Σって要するにfor文ですよ。配列をa[0]からa[100]まで順番に加えていくことと同じです」と説明しました。ソースコードで書くと次のとおりです。

・Pythonでのコーディング例

```
answer = 0
for i in range(101):
    answer += i
end
```

・Kotlinでのコーディング例

```
fun main(args: Array<String>){

    var answer: Int = 0

    for( i in 1..100){
        answer += i
    }

    println(answer)
}
```

そうすると彼は、「あー、なんだ、それだけなんだ」と納得してくれました。

ライブラリの挙動で理解してみよう

別の人の例を挙げましょう。あるプロジェクトで機械学習を具体的に利用したいという構想があるのですが、そもそも機械学習の理論がわからない……勉強してもわからない……。どのように自分の事業に活用できるのかわからないという悩みでした。彼には、「scikit-learnというPythonのライブラリがあるから、それをとにかく試してみてください」と教えました。しばらくしてその彼と会ったのですが、「残念ながら自分の思いどおりにはいかなかったけど、あとから本を読み返したら、少し理解できるようになっていた」とのことでした。

数学の考え方にどうなじむか?

筆者の経験から、数学を苦手としている人は

たくさんいることを実感しています。でも、とくにエンジニアの方は「勉強すると良いことがありそうだな……」と考えている方が多いようです。昨今、エンジニアの周りでは機械学習やブロックチェーンなど、数学の必要性が以前よりも高くなっています。もちろん、機械学習もブロックチェーンも数学を知らなくても扱うことができるかもしれません。しかし「なんだかわからないけどうまくいく」ということに気持ち悪さを少なからず感じることがあるはずです。本質的な理解が少しでもできるようになっておくことで、そうした気持ち悪さを打開することができるようになります。

今回は機械学習といった具体的なものについて解説するわけではなく、どのように数学をエンジニアリングに活かしてきたかということと、数学をどのようにして勉強したら良いかといったことについて説明します。

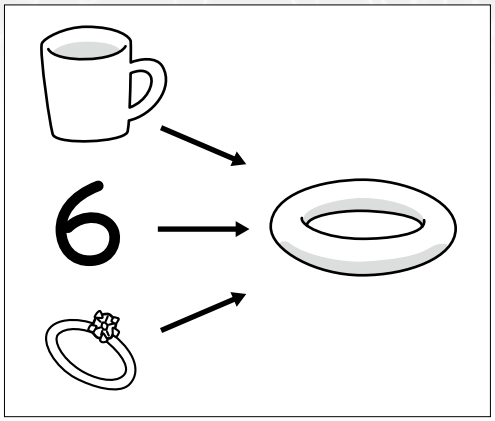
問題解決のための数学的アプローチ

ポアンカレ予想の考え方

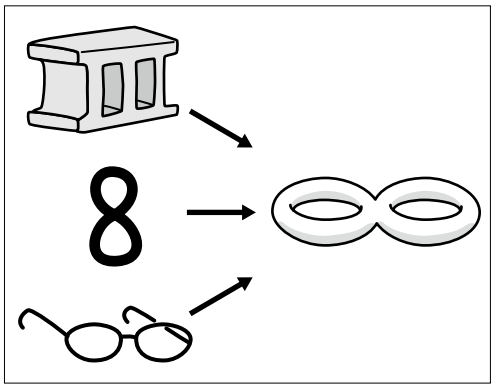
筆者が影響を受けた数学のエピソードを挙げましょう。「ポアンカレ予想」をご存じでしょうか？　1900年頃にフランスの数学者、アンリ・ポアンカレが提唱した「位相幾何学（トポロジー）」と呼ばれる数学の分野の難問です。これは2002年にグレゴリー・ペレリマンによって証明されました。筆者の専門分野ではないのですが、位相幾何学は簡単に言うと、「物体の穴の個数で物の形を決める」というものです。初めて聞くと、奇妙な印象を受けるかもしれません。

図1の場合、指輪とコーヒーカップと6は同じ形です。**図2**のように伊達メガネと8は穴が2つなので同じ形です。このように、「穴の数を変えない限りはどのように形を変形しても良い」という考え方でものを観察していきます。

▼図1　すべて同じ形？（その1）



▼図2　すべて同じ形？（その2）

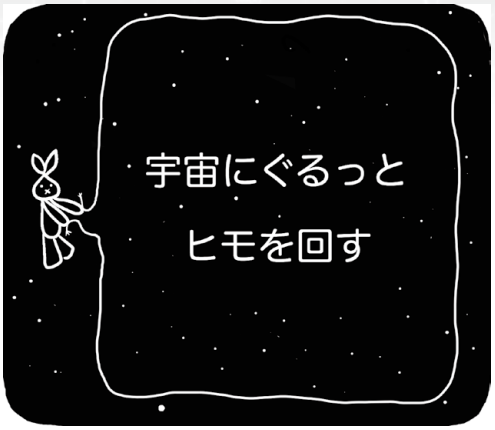


宇宙の形は球型？それともドーナツ型？

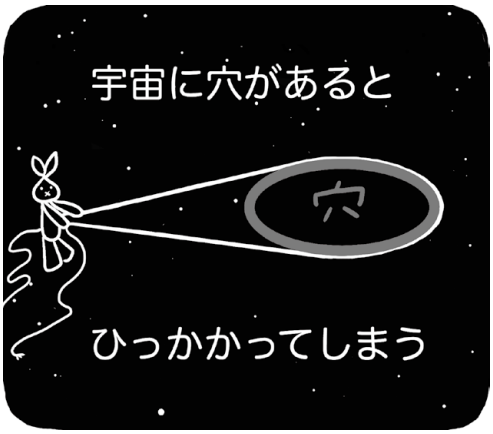
ポアンカレ予想は「単連結な3次元閉多様体は3次元球面に同相である」という、書いている筆者自身も「？」な定理ですが、**図3**、**図4**で少しわかるのではないでしょうか。

「宇宙で紐をぐるっと回して両端をひっぱったら回収できる場合、宇宙は丸い（穴がない）と言える」というのがこの予想の意義です。先ほども述べたとおり、ポアンカレ予想は位相幾何学という分野の問題ですので、世の中の多くの数学者たちは位相幾何学の手法で証明しようと考えました。そのような状態が100年ほど続いたあるとき、ペレリマン氏は位相幾何学を使わずに、物理学の流体力学や熱力学などの知識を使ってポアンカレ予想を証明しました。

▼図3　宇宙全体にひもを回す



▼図4　穴があるとひもはひっかかる



ポアンカレ予想の例はわかりやすいために例に挙げましたが、数学は「幾何学の問題を代数学の手法で解く」といったような、違うアプローチから問題を解決することが重要な学問です。手段に縛られることなく、さまざまな視点から問題を見て、そして過去の人々の知見や自分自身の経験則から焦点を絞っていきます。このものの見方は、エンジニアリングにも応用できる考え方だと思います。

現場で役立つ数学的考え方=抽象化

本誌読者には釈迦に説法ですが、先入観で視野が狭くなってしまうことはないでしょうか。エンジニア業務では、とくに多いのではないかと思います。筆者自身、気をつけているつもりですが、問題に直面したときには真正面からぶつかってしまい、何時間も時間を費やしてしまうこともよくあります。

「指輪とコップと6は同じ形」のように、物事を抽象化してみてみたり形を置き換えてみたりするなど、視点を変えることで解決しやすくなる問題はたくさんあるはずです。

わからなくても次に進んでみること

大学院時代に、ゼミで先生と共著論文を執筆したときの話です。ある証明問題で、n=2のときは成り立つのですが、n=3のときがどうしても証明できないという状況に遭遇しました。こういうときに暗雲立ち込める雰囲気になるのですが、何かいい方法はないかと何日も考え抜きます。しかし結局のところ解決できず、n=2のときだけで書くか、それともこの論文自体をやめるかという決断に迫られました。そのとき何気なく「n=4のときはどうですか？」と筆者が言ってみました。少し試してみると、n=4のときは成立していることがわかりました。ここからはいろいろと端折りますが、これをきっかけにnがすべての自然数（1, 2, 3, ……と続く数）に対して成り立つことを証明できました（そのときの論文：http://www.jams.or.jp/scm/contents/e-2012-4/2012-35.pdf）。「漫画かな？」と思えるくらいよくできた話ですが、本当の話です。

本来であれば、科学や技術は基本的に堅実にひとつひとつ積み重ねていくもので、とくに数学はその最たるものだとも考えています。ですので、理解できない部分でつまずいて立ち止まってしまったり、時間をとられてしまったりする場面というのが多いです。しかし、たとえばコンパイルエラーなどでどうしても次に進めないといった事情があるのであればともかく、「次の一歩に進んでみる」という考え方は功を奏す場面が少なからずあったように感じています。「理解していないと次に進めない気がする」という先入観に縛られて、今持っているモチベーションを失ってしまうのはもったいないことです。

数学をどう勉強するか、何を勉強したらよいか

さて、数学をどう勉強したらよいでしょうか。数学の勉強のコツは、「きちんと手でノートに書いて計算・証明する」ことと「参考書の行間を読む」ことがポイントです。

「きちんと手でノートに書いて計算・証明する」というのは文字どおりで、ノートに書くことで、「あっ、この問題は○○先生の授業でやった問題だ！」というような、直感的な理解をより深めることができます。参考書を眺めているだけでは培えないものです。

一方、「参考書の行間を読む」は慣れるまで少し難しいかと思います。数学の専門書は、ページの都合や著者の意図で多少なりとも説明が省かれていることがあります。これは決して不親切ではなく、逆に「本当に理解しているのか？」が試される場面です。まず、この行間に気づけない場合は理解していない証拠といえます。また、行間に気づいたとしても説明できない場合、これも理解しきれていない証拠といえます。この行間を説明できてこそ、本当に理解したと言えます。

図5は今回の特集の第1章も執筆されている中井悦司さんの名著である、『ITエンジニアのための機械学習理論入門』の計算を確かめている途中に書いた筆者のノートです。本の中では一般化してn行m+1列という行列で解説されていますが、本当にそのようになっているか確かめるために、あえて2行3列の場合を具体的に計算しています。ある程度慣れてきたら、本の中で書かれている証明や計算をなぞるだけでなく、自分なりに計算を確かめてみることで、より理解が深まったり、違った解釈が見えてきたりするときがあります。

図6は筆者が書いたノートです。ちょうど今ブロックチェーンに関する論文を読んでいる途中です。その中で計算が省略されている部分があり、行間を確認するために計算しています。

論文はある程度読者が限られていたり無駄を省いた書き方をしたりしているため、専門書以上に行間が省略されています。違う見方をすれば、論文の著者と読者の試し合いともいえます。著者は読者に間違いがないかを試され、読者は著者にきちんと理解して説明できるかを問われています。

このように本当に理解しているかどうかを手で確かめながら読んでいきます。とても時間のかかる作業ですが、理解度は格段に上がります。

高校時代、公式などを丸暗記して覚えていた方もいるかと思います。もちろん、そのままの形で覚えることはとても重要ですが、どのように使うかという感覚は、実際に計算することで身につきます。証明なども同様で、何度か繰り返して証明することで、他の問題や定理の証明の方向性が見えるようになります。少しの時間でもノートに理解していることを書き写してみてください。

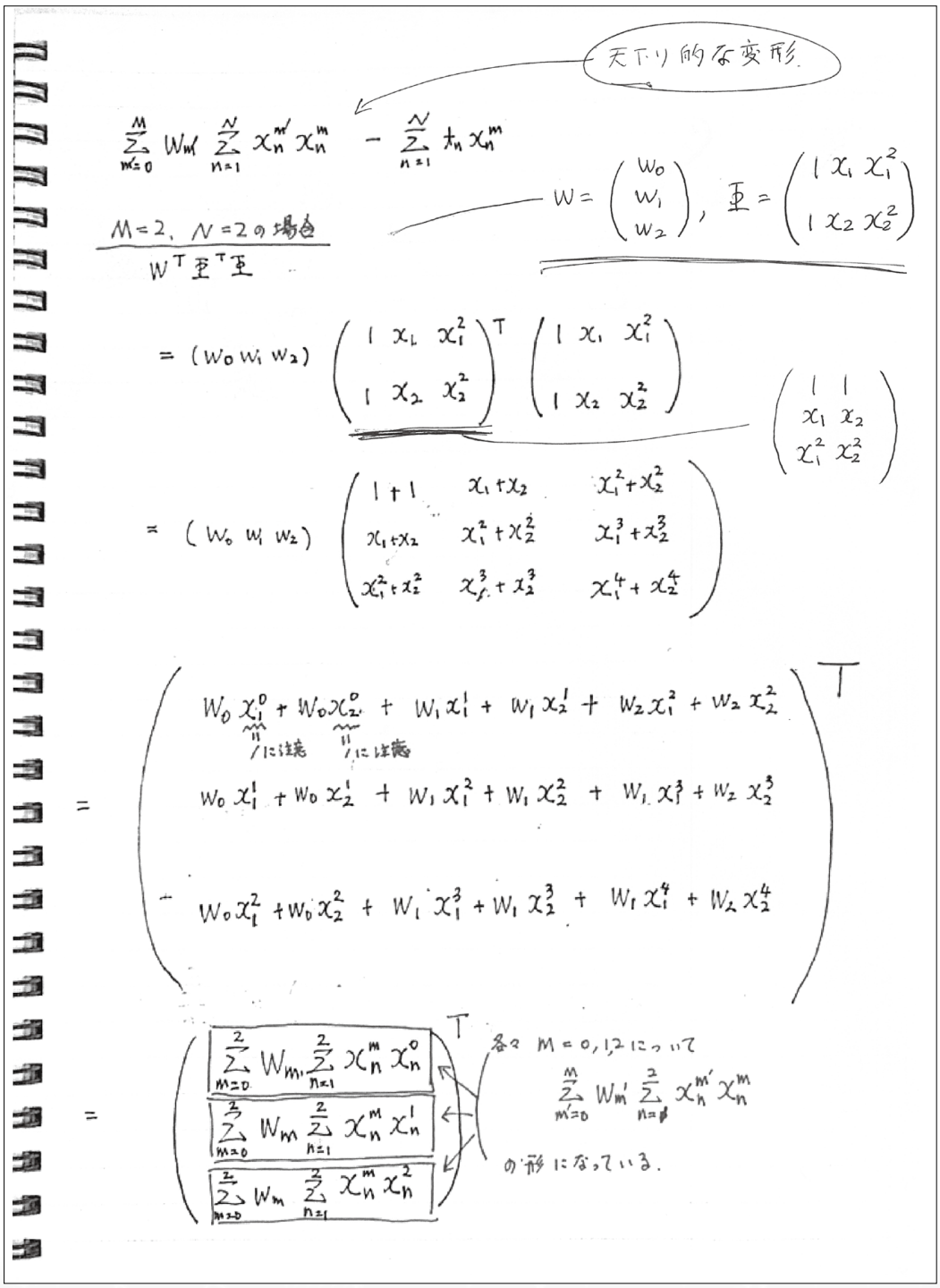
おわりに

数学にはつまずくポイントがたくさんあります。中には本当に難しくて、とても理解できないものもよくあります。しかし、たとえばエンジニアとして初めて機器やライブラリ、もしくはアプリケーションに触るときに「こんな感じかな」と使い始めてみることがあるはずです。同じように、今数学を始めてみることで、過去に苦手意識を持っていた方もそうでない方も、また違う視点やモチベーションで触れるようになっているかもしれません。

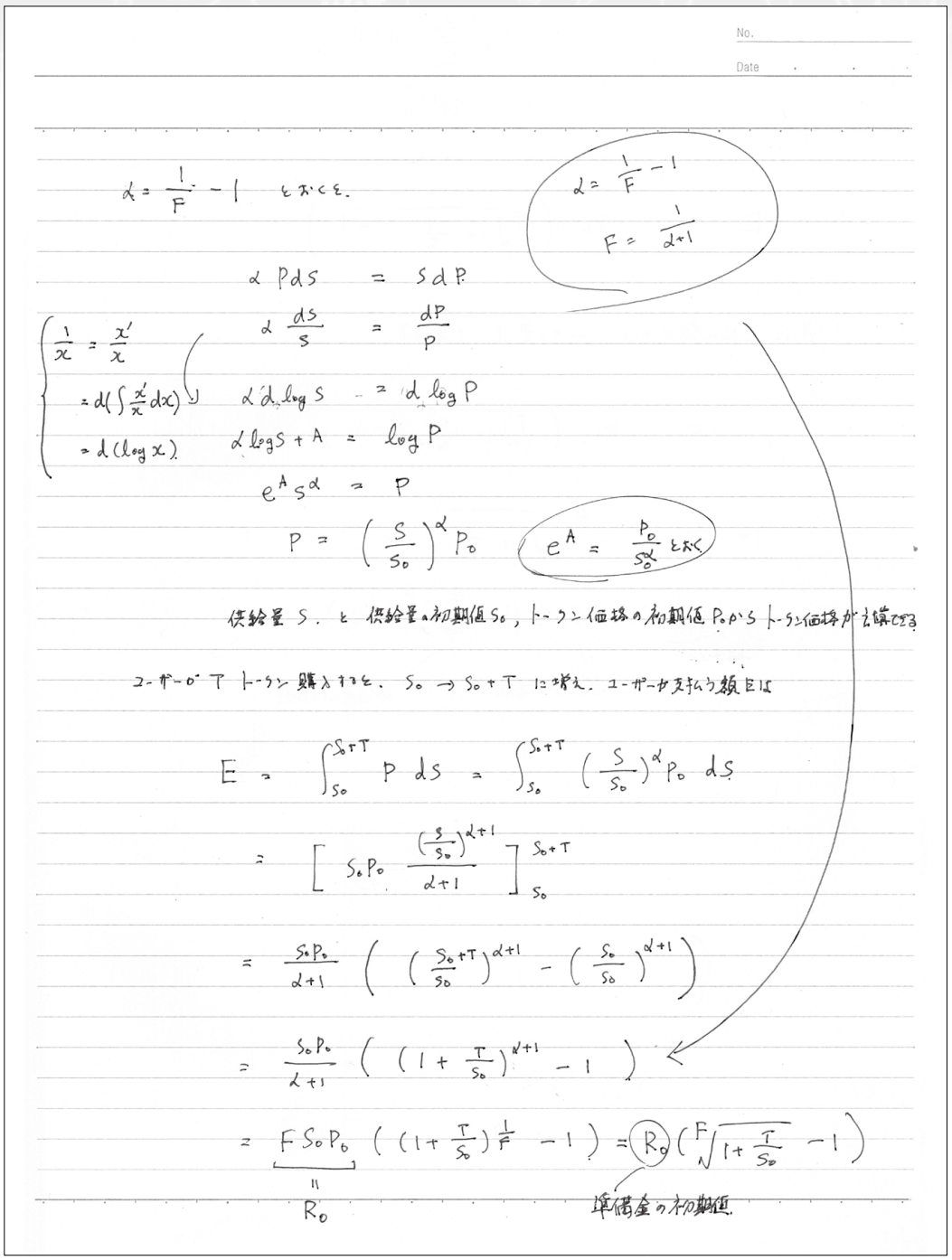
一例を出すと、当たり前に使われている公開鍵認証には代数学や暗号理論といった理論が使われています。また、機械学習の土台には微分積分や線形代数から統計学、確率論などが幅広く使われています。とくに統計学に関してはあらゆる分野でカジュアルに使われています。「少し数学をやってみようかなぁ……」と感じているのであれば、身近なところから始めてみると

いいかもしれません。もし、どうしても数学の始め方がわからない場合は、ご相談に乗りますのでFacebookでメッセージをください。この冬、コタツで数学を始めてみませんか？ SD

▼図5　筆者のノート『ITエンジニアのための機械学習理論入門』の計算過程を確かめている



▼図6　筆者のノート（ブロックチェーンの論文の計算過程を確かめている）



課外授業 2

本を読んで数学と戯れる

「虚数の情緒」と「ゲーデル、エッシャー、バッハ」

Author 吉岡 弘隆（よしおか ひろたか） 楽天株式会社 開発広報

最後に数学に触れたのはいつですか?

現役のプログラマで学生時代、数学に縁がなかった人を想定読者として書いてみます。

文系を専攻したみなさんは、高校時代に触れた数学が最後の最後という人も少なくないと思います。

なぜ数学を学ぶのか

人生において、日常的に数学を必要とする機会はほとんどありません。四則演算は使うとしても、電卓があればとくに困らないので、算数以上の高度な数学を日々使うということはないと言っても過言ではないでしょう。自分は数学の才能がないので、数学を勉強しても無駄だと感じている人も中にはいるでしょう。

『数学の認知科学』[1]という本によると、人間は乳幼児のころから数を認識できるそうです(p.17)。認知科学と神経科学の発展によって、ヒトがどのように数学を理解しているのかというのが明らかになるにつれて、数学を理解する素養は誰にでも備わっていることがわかってきました。

筆者は工学部を卒業したのですが、数学は不得意でした。数学の単位を落としました。数学が得意であれば、相当人生が変わっていたかと思いますが、数学がなくても生きていけます。にもかかわらず、最近数学と仲良くなりたい、和解したいと強く思うようになりました。

数学の専門家ではない自分がどのように数学と戯れているのか

高校や大学の授業をもう一度受ける機会があれば、それが一番のような気がしますが、自分が試みていることは、本を読む、自学するということです。数学は純粋に頭の中にだけ存在しています。理解するのに道具はいりません。数学の本を読みこなせるような認知の力を得たいというのが、自分のモチベーションになっています。

筆者が読んだ書籍を紹介します。正直言って、これらを読んだからといって、日々の仕事に役に立つとか、長年の疑問が氷解するということはいっさいないです。

すぐに役に立つ本をご所望ならば、本屋に行ってそれらしき書棚から自分に合いそうな書籍を何冊か選ぶのが良いでしょう。「すぐに役に立つ本はすぐに役に立たなくなる」という経験則があるので、注意が必要です。

その点、数学はバージョンアップしないので(厳密に言えば、パラダイムシフトはあり得るのですが、ここではそれは問いません)、10年前の書籍も100年前の書籍も、依然として陳腐化していないところがすばらしいです。昔の本でも役に立つのです。

虚数の情緒

『虚数の情緒』[2]は「中学生からの全方位独学

法」と銘打っています。1,000ページを越える大著です。本書では次の「オイラーの等式」が取り上げられています。

$$e^{i\pi} = -1$$

eは自然対数の底（ネイピア数）、iは虚数（iの2乗は-1）、πは円周率とします。いきなり面食らいますね。何が何だかよくわからない。自然対数も虚数も馴染みがないし、高校時代に習ったのかどうかもよく覚えていない。とくに文系のみなさんは、そのように感じると思います。

大丈夫です。本書は高等数学の知識をいっさい仮定しないで、中学生でも理解できるようにひとつひとつ丁寧に説明して、最後には上記のオイラーの等式の理解にまで至ります。時間はかかりますが（筆者は2ヵ月ほどかかりました）、中学生でもわかるように書かれていますので、頑張って読了してみてください。

『ゲーデル、エッシャー、バッハ』

虚数の情緒に比べて、『ゲーデル、エッシャー、バッハ』[3]の難易度は高いと感じました。こちらも700ページを越える大著です。原著は1979年に発行されました。当時第2次AI（人工知能）ブームでした。

正直言って読みこなしたとは言えないのですが、自分なりの理解を記せば、形式システム（コンピュータプログラムなど）とヒトの知性との違いなどを、さまざまな観点から多層的に考察したエッセイです。

形式システムをどんどん精密にしていけば人工知能ができるのではないかという楽観論に対して、ゲーデルの不完全性定理などを引き合いに出しながら、形式システムを越える知性がそこには必要である、ということを主張していると読みました。

1人で読み進めるのはたいへんですので、知人と読書会を開いて、月に1回のペースで読み解いています（執筆現在は同書第1部が終わったあたりです）。読書会によって自分の本の読み方には、よく理解できていないところは無意識にスルーするという特徴があることに気がつきました。

知っていることと理解していることには雲泥の差があります。書かれていることを知ったとしても、誰かに説明できなければ理解しているとは言えません。その差分を意識するのは、読書会のような機会があってこそと思います。

知人たちと、同書を紹介する薄い同人誌を執筆しました（技術書限定の同人誌即売会「技術書典3」（2017年10月22日）で発売）。その原稿を書くことによって、同書をより深く理解できました。

◆ ◆ ◆

『虚数の情緒』や『ゲーデル、エッシャー、バッハ』などの本を読むことによって、自分の認知能力が拡大していく感覚を持ちました。これは、若い人に数学を学ぶことを勧める理由の1つでもあり、難しい本を読むことのメリットです。すぐに役立つ本はたくさん読んでも、そんな感覚を味わうことはほとんどないのが実感です。

数学は語学と同じようなものか

数学を学ぶことによって、自分の認知力を広げたり、深めたりすることができます。数学の本を読むことには時間がかかりますが、メリットが多いと感じています。SD

・参考文献
[1]『数学の認知科学』、G・レイコフ、R・ヌーニェス 著、ISBN 978-4621065044
[2]『虚数の情緒』、吉田武 著、ISBN 978-4486014850
[3]『ゲーデル、エッシャー、バッハ - あるいは不思議の環 20周年記念版』、ダグラス・R・ホフスタッター 著、ISBN 978-4826901253

第4限目

物理と数学、そしてプログラミング

バスケのフリースロー計算で遊んでみよう!

Author 平林 純(ひらばやし じゅん)
Twitter @hirax

数学と物理は非常に近いのはみなさんご存じでしょう。難しい物理計算をコンピュータにさせるのは、至極当然の流れでよく行われています。本稿では、基礎的な物理現象をどのようにプログラミングするのか、Pythonを利用して、その過程をトレースすることで理解を進めてみましょう。

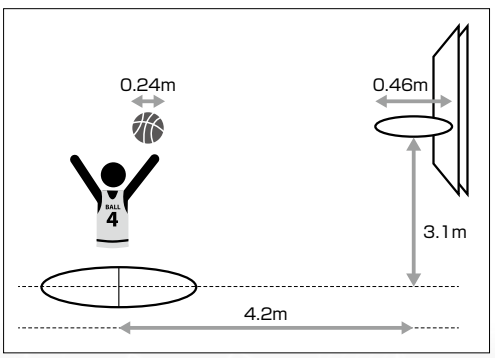
さっぱりわからない?……実は「簡単でおもしろい!」物理計算の世界

「コンピュータによる物理(科学)計算」というと、「難しくて、面白味にかけるもの」「さっぱりわからない」という印象を持っている方も多いかもしれません。けれど、それとはまったく正反対、「簡単なのにおもしろい」のが、物理計算(シミュレーション)プログラミングの世界です。

身の周りにある建築物や工業製品設計、あるいは天気予報などの自然現象予測……今やありとあらゆる用途に物理科学計算が必要不可欠になっています。ゲームでも「物理エンジン」を使わないソフトのほうが珍しい時代です。

本記事では、バスケットボールの「フリースロー」を題材にして(**図1**)、物理計算プログラムを作り、楽しんでみたいと思います。

▼図1 バスケットのフリースローを計算してみよう



方程式をプログラムして計算すれば「すべて」がわかる!

物理計算プログラミングが「簡単なのにおもしろい」理由、それは覚えることは最小限でよくて、それなのにプログラムを動かした途端、「まるでゲームのようにいろいろな現象を再現できて楽しい」からです。

それでは、最小限の覚えることが何かというと、それは「計算したい現象」を説明する方程式です。つまり、福山雅治演じる「ガリレオ」湯川先生が描く「方程式」というわけです。

ちなみに、今回の題材とするバスケのフリースローを計算するために必要な方程式は、「ニュートンの運動方程式」、身の周りのいろいろなことを計算できる古典(ニュートン)力学の基本方程式です。そこで、まずはニュートンの運動方程式がどういうものかを、簡単に納得してみることにしましょう。

連続写真で「位置と速度と加速度の関係」を納得しよう!

ニュートンの運動方程式で使われる「位置・速度・加速度」という言葉を、順を追ってあきらかにしていきましょう。**図2**は、「走る車」を、シャッタースピード1/100秒で連続撮影した場合のイメージ画像です。

最初の時刻①での撮影画像を見ると、車は動

いているようですが、（シャッターが開いていた1/100秒には）車は少ししか動いていないようです。車がいる場所は「位置」と言い、車の位置が「その瞬間にどれだけ動いているか」が「速度」です。時刻①の瞬間は、「車の速度」が遅かったことが写真からわかります。

そして、次の時刻②、つまり時刻①から1/100秒後の1/100秒間を撮影した画像を見ると、時刻①から、車が速度の分だけ移動していることがわかります。また、運転手がアクセルを踏んだのでしょう、車が移動する速度が速くなっています。この（単位時間あたりの）速度の変化量のことを「加速度」と呼びます。時刻②では、速度が増えている（加速している）ので、加速度は「増えた＝プラスだ」ということになります。

最後の時刻③になると、車の位置は速度分だけやはり移動していますが、運転手がブレーキを掛けたのでしょうか、速度は時刻②のときよりも遅く・減っています。つまり、速度の変化量＝加速度は「減った＝マイナス」になっているのです。

ここまでさらっておけば、もうニュートンの運動方程式を納得したのと同じです。

ボールの動きを計算する「ニュートンの運動方程式」

ニュートンの運動方程式は、

速度の変化（加速度）＝（物体に働く）力／物体の重さ
……式（1）

という式です。この式が表していることは、

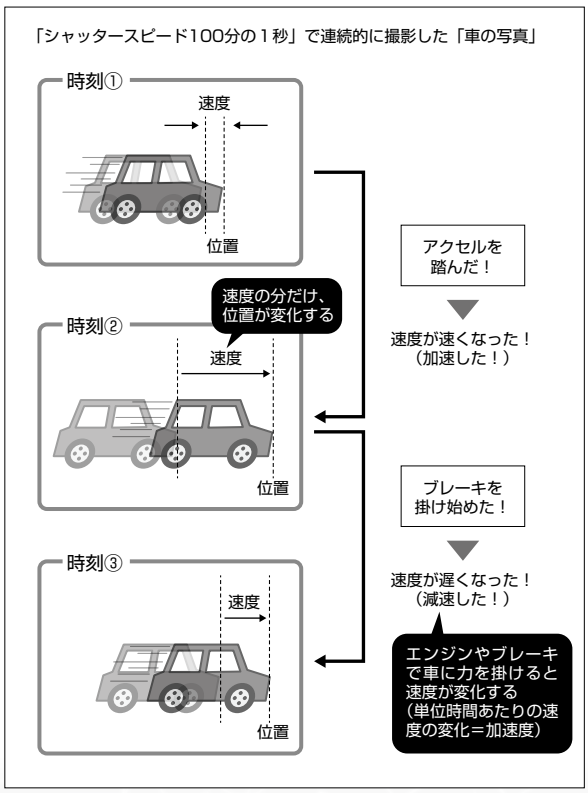
- 「物体に力を掛けると（右辺）、速度が変化する（左辺）」
  （速度の変化量＝加速度は力に比例する）
- 「物体が重いと、同じ力を掛けても速度は変化しづらい」
  （速度の変化量＝加速度は重さに反比例する）

ということです。

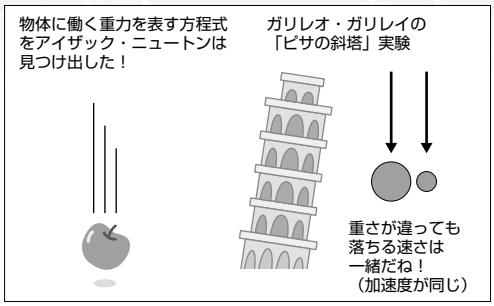
図2の車の例では、アクセルを踏み、車に進む力を掛けると加速する（加速度がプラスで＝速度が増える）し、ブレーキを踏んで車を止める方向＝逆向きの力を掛けると、車は減速する（加速度がマイナスで＝速度は遅くなる）というわけです。……当たり前ですよね。

そして、「同じ力を掛けても、（たとえば人がたくさん乗っていて）重い車だと、車の速度は変化しづらい」ということも「重いものを動かしたり・止めようとしたりしたら、重い分だけ力が必要だ」「そりゃ、そうだろう」と、自然に納得できることと思います。

▼図2　連続写真で納得する「位置・速度・加速度」



▼図3　ニュートンの万有引力の法則とガリレオのピサの斜塔実験



落ちるリンゴと同じ!ボールを下に落とす重力を「ニュートンの万有引力の法則」で計算だ!

フリースローのボールの動きを計算するためには、もう1つ別の式が必要です。それは「飛ぶボールに働く重力」を表す「万有引力の法則」を使った式です（**図3**）。

リンゴが地球に向かい木の枝から落ちるのと同じく、放り投げたバスケのボールも地球に引かれて下に向かって落ちていきます。その「重力」の強さを表すのが、万有引力の式です。その式を簡略化して、地上で働く引力を簡単に表すと、

物体に働く重力＝重力加速度×物体の重さ
……式（2）

のようになります。つまり、「物の重さに比例した力（重力）で、物は地球に引っ張られる」というわけです。「重い物ほど地球に引っ張られる」というのも当たり前ですよね。

そして、式1の「物体に働く力」に式2を代入すると、

速度の変化（加速度）＝重力加速度式
……式（3）

という単純な式になります[注1]。これは、「地球に引っ張られながら飛んでいる物体」の動きを表す方程式で、フリースローシュートで投げられたボールの動きも、この方程式にしたがいます。

そこで、この方程式を細かい時間ステップごとに繰り返し計算して、ボールの動きを刻々と追いかけていけば、フリースローのボール軌道のシミュレーションができることになります。つまり、「ボールの速度を重力加速度で変化させつつ、ボールの位置を速度分だけ（前の位置から）移動させる計算を繰り返せば、ボールがどう動いたかがわかる」というわけです。

それでは、そんな計算処理をするPythonコードを書き、そのしくみを納得しつつ遊んでみることにしましょう！

20行で書く!バスケのフリースロー計算プログラム

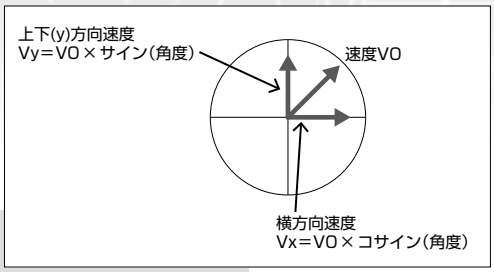
最初は「ボールを投げるだけ」のコード……でも「実におもしろい!」

まず、「ボールを投げるだけ」のプログラムを書いて、その結果を眺めてみることにします。**リスト1**はJupyter Notebook上にPythonでコーディングした「バスケのフリースロー計算」ソースコードリストの叩き台です。三角関数やグラフ表示に必要なライブラリを読み込んだあと（1～2行目）、8～25行目で「ボールを投げる」処理関数throw()を作り、そのthrow()を3回呼ぶ（33～35行目）ことで「ボールを投げる角度違いの3投」の計算を行い、さらに描画関数（27～31行）を使ってボールの軌跡を描画（37～39行目）しています。本体部分だけならば約20行のコードです。

ちなみに、5～6行目の関数interaction()は「ボールが壁や床へぶつかったり、ゴールに入ったりしたときの処理」を行いますが、この時点では「何もしない」コードにしておきます。また、3行目の%matplotlib inlineは、結果表示のグラフをJupyter Notebookに埋め込み表示するためのコマンドなので、本記事中では無視してかまいません。

さて、**リスト1**の核とも言えるthrow()関数

注1）ちなみに、式3から、地表で落下する物体の速度は重さによらず「重力加速度」で決まることがわかります。ガリレオ・ガリレイがピサの斜塔から「違う重さのものを落としたけれど、落ちる速度は同じだった（同時に地上に落ちた）」という伝説も、（空気抵抗を無視すれば）納得できるのではないでしょうか。



▼リスト1　「ボール投げプログラム」とボールを35度・45度・55度で投げるコード

```
 1 from math import sin, cos, radians
 2 from matplotlib import pyplot as plt
 3 %matplotlib inline
 4
 5 def interaction(x,y,vx,vy,ax,ay,dt): # ぶつかる処理など
 6     return (False,x,y,vx,vy) # (後で書くので、今は何もしない)
 7 #ボールを投げて、軌跡=位置(x,y)配列、ゴールしたかを返す
 8 def throw(x,y,v0,angle,repeatNum,dt):#位置,初速度,角度,計算回数,計算ステップ
 9     isSuccess = False # ボールがゴールに入ったか
10     xarray = []; yarray = [] # 水平=x方向と鉛直=y方向の位置格納配列
11     vx = v0*cos( radians(angle) ) # ボールの速度(水平=x方向)
12     vy = v0*sin( radians(angle) ) # ボールの速度(鉛直=y方向)
13     ax = 0.0  # 水平=x方向に働く加速度はゼロ
14     ay = -9.8 # 鉛直=x方向に働く加速度=重力加速度
15     for i in range(repeatNum): # 時間を細かく進めながら、計算していく
16         vx = vx + ax*dt # 水平=x方向加速度で、水平方向速度が変化
17         vy = vy + ay*dt # 鉛直=x方向加速度で、鉛直方向速度が変化
18         x = x + (vx-ax*dt/2)*dt # 水平(x)方向速度で、水平位置が変化
19         y = y + (vy-ay*dt/2)*dt # 鉛直(x)方向速度で、鉛直位置が変化
20         isPassing,x,y,vx,vy = interaction(x,y,vx,vy,ax,ay,dt)
21         if isPassing:
22             isSuccess = True
23         xarray.append(x) # 水平(x)位置を、水平位置を格納する配列に追加
24         yarray.append(y) # 鉛直(x)位置を、鉛直位置を格納する配列に追加
25     return (xarray,yarray,isSuccess)
26    # 描画関数
27 def prepareFigureArea():
28     plt.figure(figsize=(5,5)) # グラフの大きさを設定
29     plt.xlim([0,5]); plt.ylim([0,5]) # 描くXY領域(各0から5まで)を設定
30     plt.xlabel('X - Axis (m)');plt.ylabel('Y - Axis (m)') # 軸説明
31     return
32 # ボールを投げ、軌跡・結果を得る。引数：スタート位置,初速度,角度,計算回数,計算ステップ
33 x1,y1,isSuccess = throw(0,0,6,35,500,0.01)
34 x2,y2,isSuccess2 = throw(0,0,6,45,500,0.01)
35 x3,y3,isSuccess3 = throw(0,0,6,55,500,0.01)
36 # 描画する
37 prepareFigureArea()
38 plt.plot(x1, y1, 'bo-', x2, y2, 'ro-',x3, y3, 'ko-')
39 plt.legend(['35 deg.','45 deg.','55 deg.'])
```

「加速度」は「単位時間あたりの速度変化量」なので、「加速度(=ax)×次の時間までの時間間隔(=dt)」が「速度変化量」になります。「速度の変化量」を「今の時点の速度」に足すと、「次の時間の速度」を計算することができます。それを式にすると、vx+ax*dtとなります。水平(y)成分も同様です。

位置計算も、速度と同じように「次の時刻の位置＝今の位置＋速度×次の時刻までの時間」としたくなります。ただし、この式そのままでは、計算誤差が出ます。なぜかというと「速度は刻々変わっているから」です。そこで、「今の時刻の速度と次の時刻の速度の平均(vx+(vx+ax*dt))/2＝vx+ax*dt/2」を使い、それに時間間隔(dt)をかけることで、「刻々変わる速度の平均×移動する時間＝移動距離」と計算しています。なお、コードでは＋がーと逆になっているのは、vxとvyを(コード行数を少なくするために)先に(上の行で)更新しているためです。

は、ボールの初期位置（x, y）や投げる速度（v0）や角度（angle）、ボールの動きを何回（何ステップ）繰り返して計算するか（repeatNum）、どのくらい細かい時間ごとに計算を行うか（dt）を引数で与えられたうえで、次の手順で計算を行います。

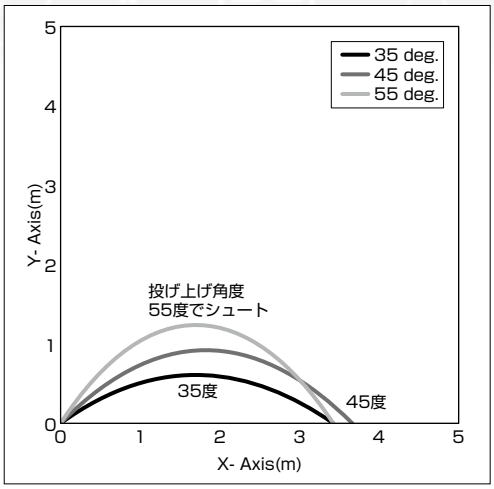
まず、ボールがゴールに入ったかを記録するフラグ変数と刻々のボール位置（x, y）を格納していくための配列を初期化します。そして、ボール位置と速度を、与えられた引数で初期化します。加速度は「重力加速度＝下向き9.8m/sになる」という**式3**の処理をさせます。

さらに、15～24行目の繰り返し文で、「次の瞬間のボール速度は、その瞬間の速度に、加速度分を足した値になる」「次の瞬間のボール位置は、今の位置に速度（と加速度分だけ増えた速度）分を足した場所になる」という処理をして、時々刻々のボール（x, y）位置を格納した配列を返す、という処理[注2]を行います。20～22行では、床・壁・ゴールに対するボール処理を呼んでいますが、今の時点では「何もしない」空処理になっています。

また、prepareFigureArea()はボールの軌跡を描くグラフ描画のための関数です。

このコードを実行すると、ボールを投げる角度違い（35度、45度、55度）の3投を行った結果が表示されます（**図4**）。まだ「バスケのフリースロー」には見えない状態ですが、「なるほど、ボールは角度が低過ぎても高過ぎても遠くに飛ばなくて、45度あたりが一番遠くへ飛ぶのか！」とあらためて気づかされるかもしれません。実は、空気抵抗を無視すると[注3]、45度の角度が一番遠くへモノを飛ばすことができるのですが、そんなこともこのシミュレーションからわかるのです。「実におもしろい」ですよね。

▼図4　ボールを35度、45度、55度で投げた場合の計算結果



床や板を配置して「ぶつかり」「ゴール」処理を書いてみよう!

次は、ゴール・ゴール裏板・床を作ってみます。10cm刻みで50×50、つまり5m四方の領域を、そこに何があるかを表すフラグ値として2次元配列objに格納します。そのうえで、各領域に何があるかが表示されるように、表示関数prepareFigureArea()を書き換えます。

この状態で**リスト2**を実行すると、**図5 (a)**が表示されます。まだ「ぶつかり」処理を書いていませんから、ボールがゴール裏板をすり抜ける「超常現象」が発生します。

そこで、**リスト3**のようにinteraction()関数を書き換えてみます。ボールが移動する次位置（領域）に「何があるか」を調べ、床や板なら、それぞれ上下（y）方向や水平＝左右（x）方向に跳ね返らせるわけです。「跳ね返り」というのは、「ぶつかる相手に対し、速度が反転する」ということです。そこで、「次の移動先が床だったら＝床にぶつかったら、速度を上下に反転させる」「ゴール裏板にぶつかっていたら、速度を左右に反転させる」という処理を入れてみます。

すると、**図5 (b)**のように、「ボールを投げて・

注2) 速度・加速度ともに、「次の時刻の量＝元の量＋（単位時間あたりの）変化量×次の時刻までの時間」という式で計算します。ただし、速度の計算式が少しわかりにくいので、リスト1に説明を書き入れました。

注3) 今回のPythonコードでは空気抵抗を無視していますが、「速度に応じた大きさの抵抗力を受ける（＝進行方向に対してマイナスの加速度を受ける）という処理」を数行追加するだけで、空気抵抗を踏まえた計算を行うこともできます。コードを書き換えてみると、おもしろいと思います。

跳ね返る」ようになります。ただし、このままでは、まだ跳ね返り方が不自然です。

身の周りにあるような物体が、何かにぶつかり「跳ね返る」ときには、向きが変わるだけでなくて、速さが少し遅くなります。その速さが遅くなる度合いを「跳ね返り（反発）係数」と呼び、**リスト3**では跳ね返り係数r=1.0（速度が変わらない）としています。そのため、「跳ね返り過ぎて、自然じゃない」わけです。そこで、rをバスケットボールの公式ルールで決められたボール規格（約0.8）にしてコードを再実行すると、**図5 (c)** のように「バスケのボールらし

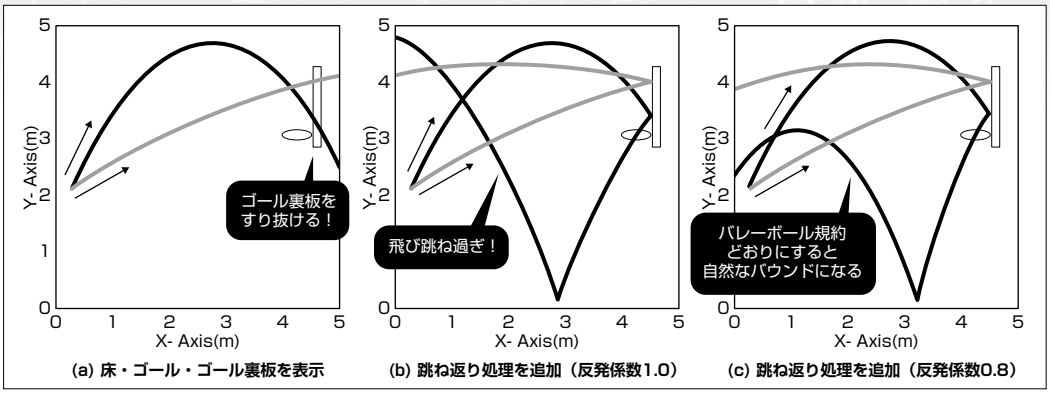
▼リスト2　床・ゴール・ゴール裏板表示を追加する

```
import numpy
nx = 50 # 10cm刻みでX,Yに50分割し、10cm×50=5m
ny = 50 # の領域を確保する
xa = numpy.linspace(0, 5, nx)
ya = numpy.linspace(0, 5, ny)
X, Y = numpy.meshgrid(xa, ya) # x,yの「平面座標メッシュ」を作る
# xyの位置で表される空間に何があるかを、配列として格納しておく
obj = numpy.zeros((ny, nx)) # 最初は「空気=0」で埋める
obj[0:1,0:49] = 1 # 床は、フラグ値1とする
obj[28:42,45:46] = 2 # ゴール裏板は、フラグ値2とする
obj[30:31,42-2:42+2] = 3 #ゴール領域は、フラグ値3とする

# ゴールや壁や床を描くよう、描画関数を変更する
def prepareFigureArea():
    plt.figure(figsize=(5,5))
    plt.xlim([0,5]); plt.ylim([0,5])
    plt.xlabel('X - Axis (m)'); plt.ylabel('Y - Axis (m)')
    plt.contourf(X, Y, obj, alpha=0.2) # フラグ値を表示する
    plt.tick_params()
    return
x,y,isSuccess = throw(0.2,2,8,65,1500,0.01)
x2,y2,isSuccess2 = throw(0.2,2,12,34,1500,0.01)
prepareFigureArea()
plt.plot(x, y, 'bo-', x2, y2, 'ro-')
```

▼リスト3　ボールの跳ね返りやゴール処理を追加する

```
r = 1.0 # 反射する時の「跳ね返り（反発）係数」

def interaction(x,y,vx,vy,ax,ay,dt):
    isPassing = False #ぶつかったか?フラグを「ぶつかっていない」にしておく
    # ボールの位置が計算領域外なら、ぶつかり・ゴール処理はしない
    if y < 0 or 4.9 < y or x < 0.1 or 4.9 < x:
    return (isPassing,x,y,vx,vy)
    # ボールが移動する次位置（領域）に「何があるか」を調べる
    objAtNextPos = obj[round(y*10), round(x*10)]
    if 1 == objAtNextPos or 2 == objAtNextPos: # 床・ゴール板なら
        x = x-(vx-ax*dt/2)*dt # 板や床にめりこんだ状態にならないように
        y = y-(vy-ay*dt/2)*dt # 位置を「前の瞬間」の位置に戻す
    if 1 == objAtNextPos: # 床なら
        vy = -vy*r # 鉛直方向で跳ね返らせる（方向を反転させる）
    elif 2 == objAtNextPos: # ゴール裏板なら
        vx = -vx*r # 水平方向で跳ね返らせる（方向を反転させる）
    elif 3 == objAtNextPos and vy < 0: # ゴールに下向き突入していたら
        isPassing = True # ゴール追加のフラグをTrueとする
    return (isPassing,x,y,vx,vy)
x,y,isSuccess = throw(0.2,2,8,65,1500,0.01)
x2,y2,isSuccess2 = throw(0.2,2,12,34,1500,0.01)
prepareFigureArea()
plt.plot(x, y, 'bo-', x2, y2, 'ro-')
```

▼図5 「床・ゴール・ゴール裏板」を追加し、跳ね返り処理やゴール処理を実装する



(a) 床・ゴール・ゴール裏板を表示　(b) 跳ね返り処理を追加（反発係数1.0）　(c) 跳ね返り処理を追加（反発係数0.8）

▼リスト4 角度と速度を変えてボールを投げてゴールするか確認する

```
r = 0.8
angleRange = numpy.linspace(30, 70, 400) # 角度振り条件
    velocityRange = numpy.linspace(5, 15, 400) # 速度振り条件
Xa,Yv = numpy.meshgrid(angleRange,velocityRange)
areSuccess = [] # 角度・速度の各条件で成功するか格納する2次元配列
for v in velocityRange: # 角度条件ループ
    a = []; for angle in angleRange:# 速度条件ループ
        x,y,isSuccess = throw(0.2,2,v,angle,500,0.01)  # 投げる
        a.append( isSuccess )
    areSuccess.append(a) # 上行と合わせ、成功したかを2次元的に格納
fig = plt.figure(figsize=(5,5)) # 結果を描く
plt.contourf(Xa,Yv,areSuccess,alpha=1.0)
plt.xlabel('Throw Angle(deg)'); plt.ylabel('Throw velocity(m/s)');
```

い動き」が再現されるようになります。

なお、**リスト3**の17～18行目では、「ゴールに向かって下向きにボールが入ったら、ゴールしたことを示すフラグをTrueにする」処理も加えられています。これで、ゴール判定ができるようになりました。

「物理計算エンジン」で、フリースローの最適条件を考えよう

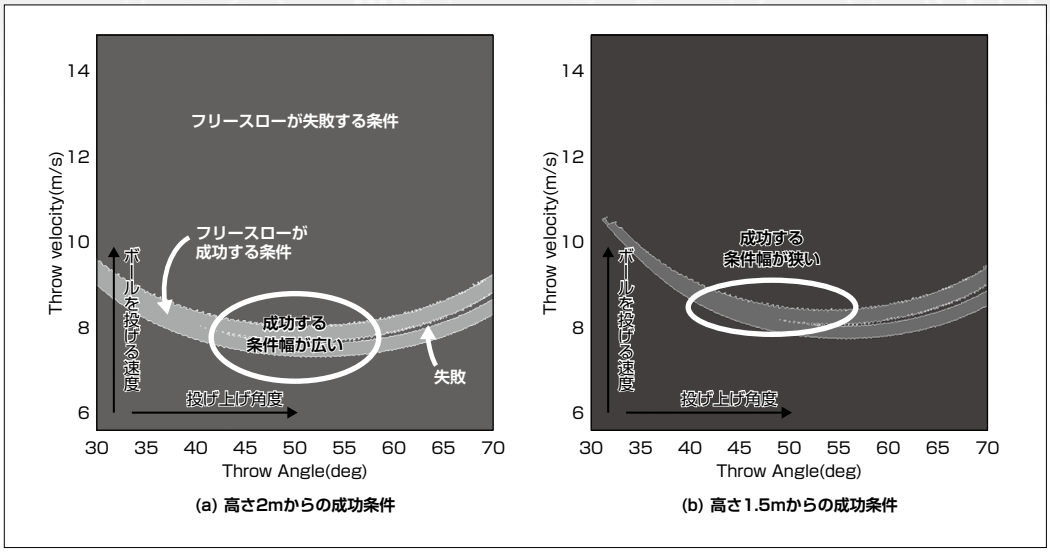
バスケのフリースローで、投げ方に応じた「ボールの動き」「ゴール成功か否か」を計算するコードができたので、この物理計算エンジンを使って「バスケ・フリースローの科学」を簡単に調べてみることにしましょう。

まずは、ボールを投げる角度・速度を変えながら「フリースロー」を行い、「ゴールしたか？」を表示するコード（**リスト4**）を書いてみます。このコードは、「ゴールしたら明るい灰色、シュートに失敗したら暗い灰色」でグラフの角度・速度に応じた個所を表示します。

ボールを放す位置が高い（高さ2m）場合と、ボールを低い位置からシュートする（高さ1.5m）場合の結果を眺めてみると、高い位置から投げたほうが成功する確率が高い（成功する条件幅が広い）ことがわかります（**図6**）。

ちなみに、「ボールを投げる角度 v.s. 速度」のゴール成功マップを眺めると、「ゴールが成功するあたりの真ん中に、ゴールが失敗する谷間」があります（**図7**）。これは、ゴールの付け根にボールが当たってしまうケースです。そういったことは、ゴールが成功したボールのコースだけを描くコード（**リスト5**）を書くとわかります。こうした解析をすることで、「高い投げ上げ角度の場合、普通の速さで直接ゴールに入れるか、速めの速度でゴール裏板でワンバウンドさせてゴールに入れるか」を決めてシュートする必要がある、なんていう戦略を作ること

▼図6　「ボールを投げる角度 v.s. 速度」のゴール成功マップ



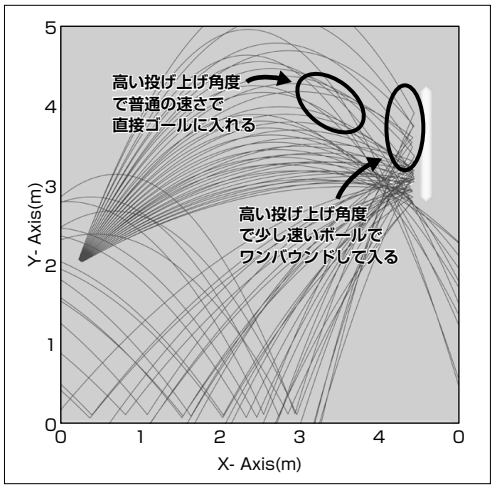
ができるわけです。

おわりに

世界を方程式で表して、何が起きるかを計算するコード……言葉だけ眺めると「さっぱりわからない」難しいものにも思えます。けれど、実は「わりと簡単なのにとても楽しい」ものです。本記事を読んだあなたが（ガリレオ湯川先生の口癖のように）こう感じていただけたら幸いです。

「なるほど、実におもしろい」SD

▼図7　高さ2mからの「成功ゴールのコース」



▼リスト5　角度と速度を変えてボールを投げてゴールしたボールコースを描いてみる

```
angleRange = numpy.linspace(30, 70, 20) # 角度振り条件
velocityRange = numpy.linspace(5, 15, 30) # 速度振り条件
Xa,Yv = numpy.meshgrid(angleRange,velocityRange)
prepareFigureArea()
for v in velocityRange: # 速度ループ
    for angle in angleRange: # 角度ループ
        x,y,isSuccess = throw(0.2,2,v,angle,500,0.01)
        if isSuccess:# 上行でボールを投げて、成功したら描く
            plt.plot(x, y, 'b-')
```

課外授業 3

プロダクトマネージャーと数学

数字力・データ分析力の必要性

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

筆者は現在個人で、エンジニアリング組織作りとプロダクト戦略、そして技術戦略という3つの柱で、スタートアップを中心とした企業にアドバイスを行っています。この中のプロダクト戦略は、プロダクトの成功に対して責任を持つプロダクトマネージャーの仕事そのものです。

筆者は、プロダクトマネージャーに求められる数学は、大枠で物事をとらえるための「数字力」と、プロダクトの戦略を立案・実施するための「データ分析能力」だと考えます。

数字力

プロダクトマネージャーは、プロダクトにおけるさまざまな意思決定を行う必要があります。いくつものプロダクトアイデアの中から成功の確率の高いものを採用する際や、複数の選択肢がある中から最適なものを選ぶ際など、多くの局面において最終的に判断するのはプロダクトマネージャーです。その際に必要となるのが、データを用いた意思決定能力です。

たとえば、筆者の過去のプロジェクトでこんなことがありました。いわゆるクライアント／サーバシステムを検討していたのですが、ある処理をクライアントで行うのかサーバで行うのかを決める必要がありました。サーバのほうが計算能力が高いので、その単体の処理はサーバで行うほうが速いのですが、その処理対象をクライアントからサーバに転送し、さらに結果をクライアントに戻すのに時間がかかります。このときは、その転送時間のほうがサーバで行うことによる処理時間の短縮よりも多くかかることが明らかだったので、その方式の採用は即座に見送りました。

同様の例は、単純な圧縮・解凍というよく使われるテクニックで見ることができます。Webで使われるWeb Fontsには標準Fontフォーマットとして WOFF2[注1]があります。同じ標準フォーマットのWOFF[注2]はzlibを圧縮アルゴリズムとして用いていたのですが、WOFF2はそれに比べて30%ほど圧縮率の高いBrotliという圧縮アルゴリズムを用いているのが特徴です。このWOFF2がWOFFに比べて本当に効果があるかどうかは、**図1**の式が真かどうかにより判断されます。

フォントファイルは事前に圧縮され、Webサーバ上に用意されていますので、圧縮にかかる時間は通常のWebアクセスでは関係ありません。純粋に解凍時間と転送時間のトレードオフになります。

この例は極めて単純なものでしたが、プロダ

注1） URL https://www.w3.org/TR/WOFF2/
注2） URL https://www.w3.org/TR/WOFF/

▼図1　圧縮アルゴリズムの効果を比較

Brotli圧縮されたファイルサイズ／転送速度＋ファイルの解凍時間
＜
zlib圧縮されたファイルサイズ／転送速度＋ファイルの解凍時間

クトマネージャーが必要とする数字力とはこのようなものです。数学というよりも算数でしょうか。四則演算に毛が生えた程度でも十分です。

数字力は基礎的な計算と併せて、本質的な数字の理解が必要です。たとえば、先の例でも、クライアントとサーバのデータを転送する回線の速度などが頭に入っていないと、とっさの判断はできません。厳密な正確さはともかく、少なくとも桁や単位は合っているような数字は頭の引き出しからいつでも出せるようになっていないといけません。

この数字力は、プロダクトマネージャーのもう1つの役割である事業性の判断にも必要となります。インターネット回線が今ほど高速でなかった時代、筆者が関わっていたある会社でのことです。大きなソフトウェア更新をユーザに行ってもらうために、インターネットプロバイダと協力し、CD-ROMをユーザに配布しようと考えたことがありました。これならば、インターネットでのダウンロードが難しいユーザにも更新を適用してもらえると考えたのですが、検討を進めるにしたがって、現実離れしている企画であることが判明しました。1人のユーザに送る配送費をざっくり100円とすると、10万人のユーザに配布するには1千万円かかります。半額の50円だとしても500万円です。プロジェクトにそれだけのコストを投入できないことは明らかでした。この件は本来ならば、アイデア段階で事業性がないことを判断できなければならなかった例です。

KPI（Key Performance Indicator）はプロダクトマネージャーが日々追う数字です。このKPIですが、プロダクトの機能や施策と紐付けるためにも、トップレベルのKPIをブレークダウンし、KPIツリーとして整理する必要があります。

たとえば、売上をトップのKPIとするEコマースサービスの場合、それを構成する下位のKPIは**図2**のように分解されていきます。

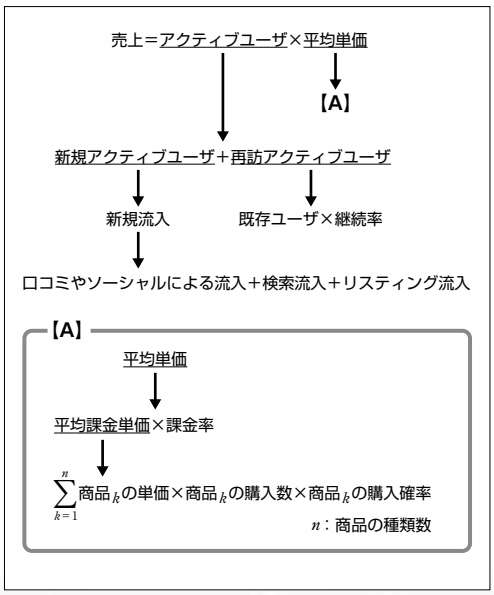
プロダクトマネージャーに必要とされるデータ分析力は、まずこのようにKPIをKPIツリーに分解する能力と、それに加えてKPIを始めとするデータの計測・解析能力になります。

このデータ計測と分析能力が、狭義の意味でのデータ分析能力となりますが、これには基本的な統計学の知識が必要です。2つのデータに相関があるように見えたとしても、きちんと相関係数を算出し、立証する必要があります。カイ二乗検定などで統計データの有意性を確認することなども行います。そのためには、統計学の知識とツールを用いた実務能力が必要です。

◆ ◆ ◆

プロダクトマネージャーには以上のように、数字力とデータ分析力の2つの能力が必要です。機械学習や3次元グラフィックスをばりばりとこなすエンジニアとはまた別の能力ですが、大枠で数字をとらえる力とデータに基づき判断する癖が、プロダクトマネージャーに資質として要求されるものと言えるでしょう。SD

▼図2　KPIツリー



第5限目

数式をきれいに表現するには

ScrapboxとLaTeXで文芸的プログラミング

Author 増井 俊之（ますい としゆき） 慶應義塾大学

プログラムのコードやドキュメントの中で数式を利用すると、プログラムが格段に理解しやすくなることがよくあります。本章では、筆者が開発にかかわっているWikiシステム「Scrapbox」のLaTeX記法を使って、コードや文書の中で数式を利用したり実行したりする方法を紹介します。

はじめに

筆者の連載記事『コロンブス日和』の第16、17回（本誌2017年2、3月号）で、「Scrapbox」[注1]というWikiシステムの紹介をしました。Scrapboxは個人の情報整理やグループの情報共有にたいへん便利なシステムで、テキストだけでなく、数式や図表も簡単に入力したり編集したりできるという特徴を持っています。本特集ではプログラミングにおいて数学をいかに活用できるかが特集されていますが、本章ではScrapboxで数式をきれいに表現して、プログラミングに活用する方法を解説したいと思います。

数式を表す困難

コンピュータのプログラムでは、あちこちで数値演算、論理演算、繰り返しなどの処理を行いますが、これらの処理を普通のプログラミング言語でわかりやすく表現できるとは限りません。たとえば配列aの要素を0から9まで足したいとき、JavaScriptでは、

```
var sum=0;
for(var i=0;i<10;i++)
  sum += a[i]
```

のような記述が必要で、あまりわかりやすいとは言えません。このような計算は数式を使えば、

$$\sum_{i=0}^{9} A_i$$

のように簡潔に表現できるのですが、一般的なプログラミング言語ではこのような表記を利用できません。

次のような二次方程式（$ax^2+bx+c=0$）の解の公式はよく知られていますが、これを計算するためには、この式とまったく異なる見栄えのプログラムを書かなければなりません。

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

普通のプログラミング言語では、このような数式をコメント欄に記述することもできませんから、プログラムの中で数式表現を併用することは簡単ではありません。

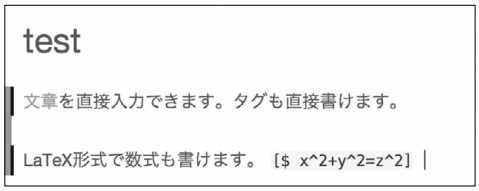
Scrapbox

Scrapboxは、Web上で簡単にメモを書いたり情報共有したりできるWikiシステムです。Scrapboxではテキストと同じページに数式もプログラムも記述できるので、前述のような問題をきれいに解決できます。

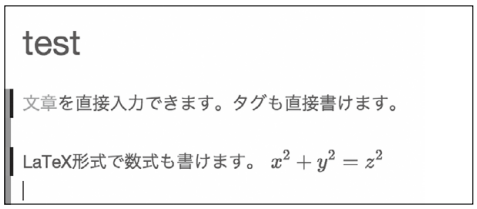
注1）URL http://scrapbox.io

▼図1　Scrapboxの画面

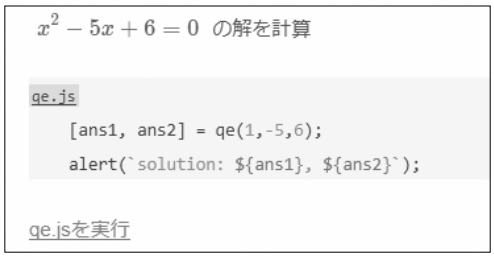
▼図2　LaTeX形式で入力・編集



▼図3　数式として表示される



▼図4　リスト2をScrapboxで見ると



▼図5　実行結果

▼リスト1　二次方程式の解の公式をJavaScriptで実装

```
code:qe.js
 function qe(a,b,c){
   var r = Math.sqrt(b * b - 4 * a * c);
   return [(-b + r) / 2 * a, (-b - r) / 2 * a];
 }
```

Scrapboxの数式編集

Wikipediaなどと異なり、Scrapboxではブラウザ画面上でエディタのように直接文書を編集できます。**[ ]**で単語を囲むことによって、リンクもリアルタイムに生成されます（**図1**）。

Scrapboxでは、「LaTeX形式」で記述した数式を**[$**と**]**で囲むと、数式が整形されて表示されます。編集中は**図2**のようにLaTeXのソースを編集できますが、別の行に移動すると整形された式が**図3**のように表示されます。

数値計算プログラムでの利用

二次方程式を計算する数式とプログラムをScrapboxで記述してみます。さきほど出てきた二次方程式の解の公式を利用します。数式は、Scrapbox上では次のような記述になります。

```
[$ \frac{-b \pm \sqrt{b^2-4ac}}{2a}]
```

公式は**リスト1**のようなJavaScriptコードで計算できます。これをScrapboxに記述すると、1行目の**code:qe.js**によって「qe.js」がソースコードへのリンクになるので、「https://masui.github.io/runp5/?code=https://scrapbox.io/api/code/<ユーザ名>/<ページ名>/qe.js」のようなリンクを利用してコードを実行できます（https://masui.github.io/runp5はJavaScriptの実行環境）。加えて、**リスト2**のように記述すると、**code:qe.js**下の2行が、qe.jsのコードとして追加されます。こうしてできた**図4**の「qe.jsを実行」リンクを押すと、実行結果が表示されます（**図5**）。

このように、Scrapboxの数式記法を利用すると、文芸的プログラミング[注2]的にプログラムを記述できます。

注2）URL https://ja.wikipedia.org/wiki/文芸的プログラミング

▼リスト2　Scrapboxにおける記述

```
[$ x^2 - 5x + 6 = 0] の解を計算

code:qe.js
  [ans1, ans2] = qe(1,-5,6);
  alert(`solution: ${ans1}, ${ans2}`);

[https://masui.github.io/runp5/?code=https://scrapbox.io/api/code/<ユーザ名>/<ページ名>/qe.js qe.jsを実行]
```

数式の身近な利用

数値演算を行うプログラムを書く機会が多くない人でも、文章やプログラムの中でちょっとした式を表現するとき、数式表現ができると便利です。A[1]と書くよりは[$ A_1]→A_1と書くほうがきれいですし、x**2と書くよりは[$ x^2]→x^2と書くほうが、またsqrt(2)や$\sqrt{2}$などと書くよりは[$ \sqrt{2}]→$\sqrt{2}$と書くほうがきれいですから、数式表現を使うとプログラムのドキュメントなどを書くのが楽しくなります。

配列や表の記述

配列や行列は次のようにきれいに書けます。

・[$ a = \{a_1, a_2, a_3 \}]→

$$a = \{a_1, a_2, a_3\}$$

・[$ v = \begin{pmatrix}
a & b \\ c & d \end{pmatrix}]→

$$v = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

論理式の記述

数式表記を利用すると論理式もきれいに書くことができます。たとえば、フィボナッチ関数は次のような数式で定義できます。

・[$ f_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f_{n-2} + f_{n-1} & n > 1 \end{cases}]→

$$f_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f_{n-2} + f_{n-1} & n > 1 \end{cases}$$

これをJavaScriptで記述すると次のようになるでしょう。

```
code:fib.js
 function fib(n){
   if(n == 0) return 0;
   if(n == 1) return 1;
   return fib(n-1) + fib(n-2);
 }
[…(省略)…/fib.js fib.jsを実行]
```

さきほどと同じように、次のように記述するとブラウザ上で実行できるリンクになります。

```
code:fib.js
 alert(fib(10));
```

フィボナッチ関数の場合、数式とプログラムで大きな違いはありませんが、条件などが複雑な場合は数式のほうがきれいに表現できることもあります。たとえば、「条件1と条件2が両方満たされないという条件」を表現したいとき、数式表現だと、

・[$ \overline{\text{条件1}} \cap \overline{\text{条件2}}]→

$$\overline{条件1} \cap \overline{条件2}$$

のように記述できますが、プログラミング言語を利用する場合は、

```
(! condition1) && (! condition2)
```

のような表現が必要になります。

複雑な数式の例

もっと複雑な計算が必要な場合、数式を参照しながらプログラムを書けると便利です。

Wikipediaのベジェ曲線のページ[注3]を参照して、ベジェ曲線をプログラムで描く際のドキュメントについて考えてみましょう。

Wikipediaによれば、ベジェ曲線の定義は**図6**のようになっています。このWikipediaの記述は、Scrapboxで**リスト3**のように記述すると、まったく同じ表示を再現できます。Wikipediaで数式入り文書を書くのはたいへんですが、Scrapboxではエディタの要領で簡単に数式を書いていくことができます。

リスト3の`\mathbf{P}(t)`のような数式表現に馴染みがない人も多いでしょうが、論文作成のために広く使われているLaTeXではお馴染みのものであり、長年に渡り広く利用されているものです。最近はPowerPointでも利用できるようになった[注4]ようですし、数式の記法としては標準的なものだと言えるでしょう。

このように、ScrapboxではWikipediaと同等の数式を記述できることに加え、画像もプログラムのコードも利用できますから、プログラムを文書のように扱うとき、とても有用です。

◆ ◆ ◆

今回はScrapboxを使って、数式をプログラミングで活用する方法を解説しました。普通のプログラミング用テキストエディタと異なり、Scrapboxではプログラムコードも数式も画像も利用できるのでたいへん便利です。

LaTeXの数式表現はDSL（ドメイン固有言語）の1つだと言えます。楽譜やコード譜を描くためのDSLや、データからグラフを描く機能などがあればScrapboxの表現力がさらに増えると思われるので、これからもさまざまな拡張を検討していきたいと思います。SD

注3）URL https://ja.wikipedia.org/wiki/ベジェ曲線
注4）URL http://www.publickey1.jp/blog/17/latexwordpowerpoint.html

▼図6　ベジェ曲線の定義（Wikipediaの一部ページをそのまま引用）

定義 ［編集］

制御点を $\mathbf{B}_0$, $\mathbf{B}_1$, ..., $\mathbf{B}_{N-1}$ とすると、ベジェ曲線は、

$$\mathbf{P}(t) = \sum_{i=0}^{N-1} \mathbf{B}_i J_{(N-1)i}(t)$$

と表現される。ここで、$J_{ni}(t)$ はバーンスタイン基底関数のブレンディング関数である。

$$J_{ni}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

t が 0 から 1 まで変化する時、$\mathbf{B}_0$ と $\mathbf{B}_{N-1}$ を両端とするベジェ曲線が得られる。**一般には両端以外の制御点は通らない。**

▼リスト3　Scrapboxでベジェ曲線の定義を記述

```
制御点を [$ \mathbf{B}_0],[$ \mathbf{B}_1], ...,[$ \mathbf{B}_{N-1}] とすると、ベジェ曲線は、

 [$ \mathbf{P}(t) = \sum_{i=0}^{N-1}\mathbf{B}_i J_{(N-1)i}(t)]

と表現される。ここで、[$ J_{ni}(t)] は[バーンスタイン基底関数]の[ブレンディング関数]である。

 [$ J_{ni}(t) = \begin{pmatrix}n\\i\end{pmatrix} t^i(1-t)^{n-i}]

 [ ] [$ t] が 0 から 1 まで変化するとき、[$ \mathbf{B}_0] と [$ \mathbf{B}_{N-1}] を両端とするベジェ曲線が得られる。[* 一般には両端以外の制御点は通らない。]
```

課外授業 4

数学の勉強法

独習の手がかりとお勧めの書籍紹介

Author 藤原 博文（ふじわら ひろふみ）　株式会社タイムインターメディア

数学で広がる仕事の幅

コンピュータを使ううえで数学の知識は不可欠です。最近は人工知能、ディープラーニング、ビッグデータなどの話題が多く、少し深く知ろうと思うと数式がいっぱい出てきて「これは無理」と思うことも多いかと思います。

筆者は技術系シミュレーションなども行うため、数式を含む計算の解説を読んでプログラムを実装したり、理論どおりの計算結果にならないプログラムのデバッグを頼まれて、数式とプログラムを比較したりすることもあります。

だいたい大学の理工系の学部で習う範囲はわかりますが、それでも知らない技術分野の場合、数式を理解するのに苦労することも珍しくありません。仕事は多様ですが、数式が出てくる仕事も対応しています。

公式の暗記よりも大事なこと

まず、数学は暗記科目ではありません。さまざまな公式を暗記させるような授業や参考書に出くわしたと思いますが、すべて忘れましょう。現実に数学が必要な世界では、公式はいつでも見ればよく、便利に利用するものです。公式をいくら暗記したって、数学ができるようにはなりません。公式の意味、あるいは意図を知り、使いこなすことが重要です。

数学では、さまざまな概念が出てきます。概念は暗記ではなく、理解し、自由にイメージし、利用できるようになることが重要です。

高校時代に三角関数のいろいろな公式を時間をかけて覚えたかもしれませんが、複素数と三角関数の関係を示すオイラーの公式（$e^{i\theta}=\cos\theta + i\sin\theta$）の威力を知ってしまうと、三角関数の公式の暗記がムダと気づいたでしょう（iは虚数）。これは、小学校で算数の範囲に限定することで難解になっていたことが、中学で方程式を習うと機械的に解けるようになることと同じです。数学では、より高度な概念を習得すると、今まで面倒だった計算が簡単かつ統一的に考えられるようになります。

高校までの数学で数学が嫌いになり学習を放棄した人もいると思いますが、コンピュータが高速になり、より高度なことができるようになればなるほど数学の重要性が増しています。数学についての知識の有無で、自分の理解できるプログラムの範囲が限定されます。確率・統計の知識は、大量のデータを高速に扱えるようになり、ますます重要になっています。

数学自習のコツ

数学は、わからないとなったら、何がわからないかもわからなくなります。何度読み直してもわかるようにはなりません。努力だけではどうにもなりません。数学に限りませんが、わからなくなったら、わかる時点まで戻って勉強するのが早道です。確実にわかる、自信が持てる

ところから再出発しましょう。高校レベル、あるいは中学レベルまで戻って勉強し直してみましょう。

数学は理詰めでわかることは当然必要ですが、それだけでは使いこなす、まして好きになるのは無理です。数学の世界をイメージできる、感じるレベルまでいけば、数学が好きになります。数学の点数が良かったら数学が好きというのは本物ではありません。数学の感覚、イメージを楽しむようになってやっと数学の本質がわかると思いますが、最初からそこまで行くのは無理なので、徐々に近づけるようにしましょう。

数学の勉強に必要なのは教科書・参考書と先生・仲間です。プログラミングの勉強会などと同じように、最近は数学の勉強会もいろいろ開催されています。とはいっても、地方であるとか、時間的に無理という場合も多いでしょう。でも、数学の場合、十分な時間を確保できれば自習することもできます。

ノートとボールペンをどれだけ消費するか

数学書は読んだだけで理解するのは難しいものです。紙に数式を書いたり、グラフ、図などを描いてきちんと確認することが重要です。それも、単に書き写すのではなく、途中の計算は本を見ずに自力で計算して、最後に結果が本と一致しているか確認すると、力がつきます。計算に自信がなくなったらカンニングは大いにしてください。

書くためには、紙と筆記具が必要ですが、紙としては5mm方眼ノートをお勧めします。数式を書くときは水平線だけを意識し、グラフを描くときは目盛りとして利用し、表では罫線として、図ではグリッドとして活かせます。B5サイズの5mm方眼ノートは種類も多く、持ち歩きにも便利で、100円ショップでも入手できます。数学書1冊勉強するのに数冊必要になるでしょうが、とても安い投資で、保存しておけば学習記録にもなります。

筆記具は、鉛筆、シャーペン、ボールペン、消えるボールペン、そのほか何でもかまいません。とにかく延々と書く癖を付けましょう。本に書いていることを鵜呑(うの)みにするのではなく、もしかして印刷ミスがあるかもしれないくらいの疑いを持ちながら計算しましょう。数学書には特殊な記号が多く、ミスが見つかることも珍しくありません。しっかり計算すると、ボールペンのインクがなくなります。そのくらい計算すると、実力もついてきます。

プログラミングへの応用は大学数学の復習から

学校の数学は、授業科目として勉強していた場合が多いでしょう。でも、今はプログラミングのために数学が必要、つまり差し迫った理由があって勉強しているのではないでしょうか。過去に数学を勉強しなかったことを悔やんでもしかたがありません。筆者も過去にもっと数学を勉強しておけばよかったと思いますが、プログラミングを始めたころは人工知能やビッグデータなどが今のように隆盛になり数学がここまで重要になると想像できませんでした。

プログラミングといってもいろいろな分野があり、それぞれで使う数学が違います。それでも、ほぼ共通して必要となる数学分野は存在します。

プログラミングの本の多くは、高校の数学範囲までが前提で書かれていることが多いので、高校数学で挫折した場合や、高校数学を忘れている場合は、高校数学の復習から始めてください。高校数学の場合、多数の学習参考書があり、レベルやていねいさもかなり自由に選べます。

もう大学入試はないので、難問、奇問を解ける必要はありません。基本をきちんと理解するだけで十分なので、できるだけていねいに説明している参考書で勉強するのがベストです。学習参考書の例題を自分で解き直し、練習問題を解く程度で、問題集まで解く必要はありません。

高校数学が大丈夫の場合は、高専・大学教養

レベルの数学をマスターしましょう。

大学レベルの数学書となると、高校までと違い、延々と理論的な説明や計算が続いていると感じ、急に難しくなったと思うでしょう。実際にそう感じる学生は多いようですが、大学1、2年で教える数学をきちんと習得していないと、理工系の場合には専門分野の勉強に行き詰まることがあります。文系でも、経済学などでは数学まみれになって、こんなはずではなかったと思うこともあるでしょう。文学、芸術の分野でもコンピュータ利用が進み、数学の利用が増えています。

どんな本を読むべきか

大学レベルの数学書でも、非常にていねいに、まるで受験参考書そっくりの体裁で書かれた本もあります。一般の大学1、2年向けの本格的な教科書に比べると説明範囲が狭かったりしますが、専門課程へ進むために必要となる数学のほぼ全範囲を非常にていねいに説明しているシリーズがあり、理工系の大学の生協で並んでいるものに、マセマの『大学数学キャンパス・ゼミ』があります(**図1**)。

全9巻からなり、理工系の1、2年、一部3年くらいまでの数学をほぼ網羅していて、各巻の関連が矢印で示されています。とくに、最初の微分積分と線形代数は、ディープラーニングはもちろん多くの分野で必須とされる知識です。できれば、複素関数、ベクトル解析、常微分方程式くらいまでは勉強してほしいです。

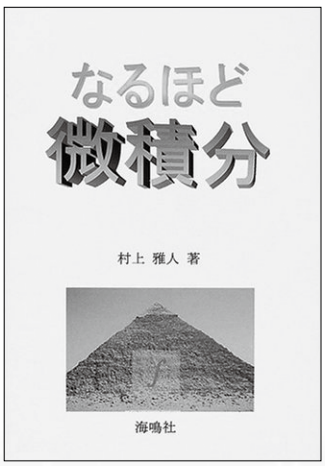
各巻に対応した演習書もありますが、そこまで頑張る必要はないと思います。

前述のシリーズでは、いかにも受験勉強の続きの気がして嫌な気分になる場合、同様の範囲を非常にていねいに説明しているのが海鳴社から出ている『なるほどシリーズ』(村上雅人 著)です(**図2**)。地味な本ですが、省略のない計算が延々と示されており、独習に向きます。マセマでは詳しく扱っていない整数論や、確率統計関係が3冊もあり、最近のニーズに合っています。

いわゆる立派な数学シリーズとして、岩波書店、朝倉書店、共立出版、東京大学出版会などが大学レベルの数学をきちんとカバーする格調高いシリーズを出しています。しかし、多くは

▼図1　基本をおさえる『大学数学キャンパス・ゼミ』
馬場敬之、『微分積分キャンパス・ゼミ』マセマ出版社、2017年
そのほかに線形代数、統計学、複素関数、ベクトル解析、常微分方程式、フーリエ解析、ラプラス変換、偏微分方程式など(http://www.mathema.jp/)

▼図2　独習に向いている『なるほどシリーズ』
村上雅人、『なるほど微積分』海鳴社、2001年
そのほかに線形代数、確率論、統計学、整数論、虚数、複素関数、微分方程式、回帰分析、ベクトル解析、フーリエ解析など(http://www.kaimeisha.com/)

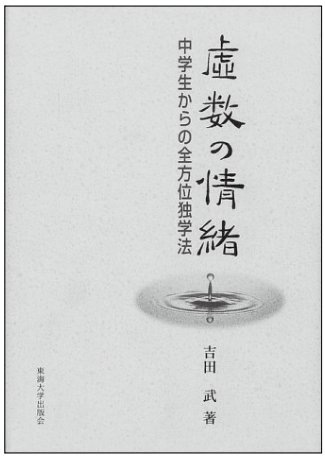


難しいので、やさしいていねいな本が済んでから仕上げとして勉強するにはとても良い本です。

高校数学も無理、延々とシリーズを読破するのはとても無理だけれど、数学的な考え方をしっかり身に付けたい場合に適した数学書もあります。

『虚数の情緒・中学生からの全方位独学法』(吉田武 著) は、中学生レベルから始めて、最後は数学的にも大変と思われている量子力学の説明まで書かれている1,000ページの分厚い本です (**図3**)。虚数についての本というより、数学とは何かを虚数を通して延々と語っています。途中で野球のバットの話がやたらに詳しくなりますが、数学の本質に触れてみたい人にはお勧めの本です。

▼図3　吉田武、『虚数の情緒・中学生からの全方位独学法』東海大学出版部、2000年 (https://www.press.tokai.ac.jp/)



▼図4　トランスナショナルカレッジオブレックス編、『フーリエの冒険』言語交流研究所ヒッポファミリークラブ、2013年 (http://www.lexhippo.gr.jp/)

『フーリエの冒険』は、数学ではなく人間の多言語を子供のころから使おうという小学生から大人までの集まりが、ことば (音声) は波なんだから、波について勉強しようと考え、どんどん進んで行ったら、sin、cosはもちろん、虚数も出てきて、波を分析したらフーリエ解析にたどり着いてしまうという話です (**図4**)。記述は小学生が習得し、別の小学生にもわかるように図を多用して非常にていねいに説明します。高校で習う三角関数は試験のためだけの勉強という感じですが、ちゃんとした目的で三角関数、虚数が使われる例になっていて、数学教育のあるべき姿になっています。

おわりに

コンピュータで利用する数学は、理工系全学科共通の数学とは別に、離散数学、整数論、グラフ理論、集合、論理、代数学などが必要になります。アルゴリズムについて深く知ろうと思うと、このあたりの知識が必要になりますが、大学教養レベルの数学を済ませてからでよいでしょう。レベルがさらに上がってしまうので、今回は説明を省略します。

数学に限らず、勉強はとにかくモチベーションが大切です。数学の勉強に必要な書籍や情報はいっぱいあり、モチベーションがあれば大学レベルくらいは習得が可能になっています。

数学の勉強は、ゆっくりペースが重要です。プログラミング言語の学習のペースではとても無理です。ゆっくり時間をかけて考えたり悩んだりしながらで大丈夫です。1日勉強して1ページも進まないことも珍しくはありません。ゆっくり、でも継続的に勉強すれば、きっと数学ができるようになります。SD

第6限目

関数型プログラミングと数学

両者における「関数」の違いを探る

Author 五味 弘(ごみ ひろし) 沖電気工業株式会社

関数型プログラミングにおける「関数」、それ以外のプログラミングにおける「関数」、そして数学における「関数」は、それぞれ性質が異なります。LispやJavaで数学のモデルを実装しながら、それらの違いについて考察しましょう。

関数型プログラミングの関数は数学の関数と同じ?

関数型プログラミングの入門書[注1]の多くでは、関数型プログラミングにおける関数とは数学の関数と同じである、ということから始まっています。果たして、本当に同じものでしょうか。それよりも何よりも、プログラムの関数とは何者でしょうか。そしてそもそも、数学の関数の正体はなんでしょうか。

関数型プログラミングの世界は数学に彩られています。関数型プログラミングの計算モデルはラムダ計算と帰納関数(再帰関数)から始まり、項書き換え系など、いろいろな数学的な計算モデルがあります。また、これらの計算モデルの基礎となる代数学、さらに群論から圏論などの離散数学の多くの分野が、関数型プログラミングの世界をキラキラときらびやかに、そしてうっとうしいものにしています。

それでは本題に戻って、関数型プログラミングの関数は数学の関数と同じものでしょうか。この答えは「初心者で純粋な方」への答えはもちろんyesで、「中級者以上でひねくれた方」への答えはnoです。これが、理想世界の数学と現実世界のプログラミングのギャップ(萌え)になります。

この、yesともnoともはっきりしない答えの根拠(言いわけ)を知るために、次節から数学の正体を炙(あぶ)り出していきます。

数学の関数と関数型プログラミングの違いとは?

ここではsinやcosのような数学の関数とはどんなものか、それが一般のプログラミングの関数や関数型プログラミングの関数とどこが違い、どこが同じなのかをみていきます(先に**図1**にまとめています)。

参照透過性と副作用

関数とは入力として引数を与えて、出力として値を返すものです。ここまでは数学でも、プログラミングでも同じです。値を返さない関数は関数ではありません(voidという単語は忘れてしまいましょう)。

私たちが算数の時代から習っている関数に、2項演算子の+(プラス)があります。それでは最初にこの+を、小一時間ほど問い詰めていくことにしましょう。ところで1+2は3になります。これは関数+が入力として1と2の引数を受け取り、+関数の中でそれらを足し算して、その結果、出力として3の値を返すということです。これは小学生でもわかるものですが、これが数学の第一歩です。

注1) 「『はじめてのLisp関数型プログラミング』、五味弘 著、ISBN 978-4-7741-8035-9」でも数学の関数と同じと言っています。

Column

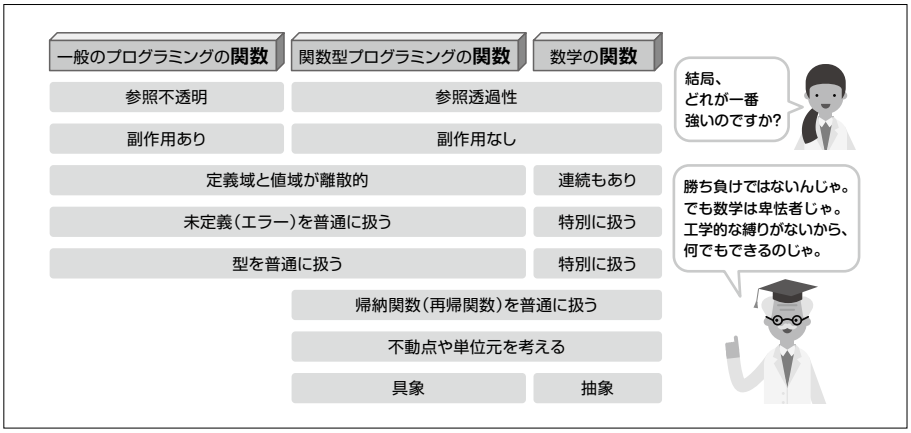
関数型プログラミングを彩る数学のいろいろ

本文で出てきた数学用語を少し紹介したいと思います。

ラムダ計算（lambda calculus）は、3個の計算規則だけで関数の定義と適用を定めたものです。帰納関数（再帰関数、recursive function）も、計算モデルとして原始再帰関数など5個の関数を用意して再帰関数での計算可能性を示しています。関数適用を項の書き換えとしてとらえた項書き換え系（term rewriting system）は、関数の評価戦略などで使われます。群論（group theory）は、コンピュータ的に言えば関数（2項演算子）適用の基礎となるものであり、圏論（category theory）もコンピュータ的に言えば型を持った演算子の基礎となるものです。離散数学（discrete mathematics）は連続でない離散した値（例：整数）を扱う数学の一分野です。ここはコラムですので、わからなくても気にする必要はありません。安心してください。

今回はこれらのキラキラしている数学用語を本章の枕と落ちに使いますが、これらを詳しく解説することはしませんので安心してください。でも知っていると、関数型プログラミングを暗記の世界から応用の世界へ導いてくれます。知っているとお得です。それに知っていれば、周りのプログラマに見栄も張れます（たぶん）。

▼図1　分野における「関数」の違い



ここで気づいてほしいのですが、数学の関数は入力と出力だけの作用だけで、それ以外の作用（これを副作用と言います）がなく、どこからでも・いつでも・誰が関数適用をしても、同じ入力に対しては同じ出力結果を返します。これを参照透過性と言います。これがプログラミングの関数との大きな違いです。プログラミングの関数は代入文などで、副作用を積極的に使っているものが多いのです。「副作用がある関数は関数ではない」と謳う数学者もいるほどです。

一方、関数型プログラミングは関数で副作用を持たないようにプライドをかけて作るために、数学の関数と同じように参照透過性を持ちます。数学の関数と関数型プログラミングの関数は同じと言えます。

ただこの話は、“純粋な”関数型プログラミングの世界での話です。少しでも副作用が入る不純な関数型プログラミングでは、純粋な数学の関数とイコールではありません。ストリーム（無限リストによる実装）を使って入出力を副作用なしにしたと言い繕っている関数型プログラミング言語でも、その言語処理系内部では副

参照透過性と副作用

参照透過性（referential transparency）とは、同じ入力に対して、いつでもどこでも同じ出力を返すことです。参照透過性があると、プログラム置換や並列動作が簡単にできるというメリットが生まれます。参照透過性のためには関数に副作用がないことが必要です。

副作用（side effect）とは、関数の主作用――入力から計算して出力する――以外の作用のことです。たとえば、入力と出力以外の状態を変化させる作用です。入力で与えないグローバル変数やファイルなどの状態を変化させると副作用が生じ、参照は透過的でなくなります。もちろん数学の関数には副作用がありませんので、参照透過になります。副作用があるプログラムの関数と区別するために、プログラム側から見た呼び名としての数学の関数を純粋関数（pure function）と言います。

作用が生じています。たとえばGC（ガベージコレクション）は透過的に動作しているように見えても、それを保証できません。しかし外側から見れば参照透過性を持ちますので、安心してください。

リスト1を見てください。たとえば関数addには副作用がありません。関数addにはグローバル変数などがありませんので、状態を変化させません。いつどこからadd関数を適用しても、同じ入力に対しては同じ出力です。この意味だけにおいては数学の関数と同じです。一方、関数addUpはグローバル変数totalの値（状態）を変化させています。totalの値によって、同じ引数でもaddUpの返す値が異なりますので、参照透過ではありません。入力で与えない変数の値やファイル、データベース、ネットワークなどの入出力などの状態を変化させるプログラムの関数は参照透過ではないのです。

数学のように参照透過性があると、いつどこで関数を実行しても同じ結果を返すので、引数の実行順序を好きなようにできます。たとえば、引数の値が必要になるまで実行を遅延させることができます。**リスト2**の関数condで`cond(condition, add(1, 2), add(3, 4))`を実行するときに、conditionの値により`add(1,2)`か`add(3,4)`のどちらかの実行は不要になります。このとき、実行が必要なconditionだけを実行する戦略を「遅延評価による必須引数評価」戦略と言います。なんか格好いいです。

ここまでは、数学の関数は一般のプログラミングの関数とは異なることを紹介しましたが、関数型プログラミングの関数との違いは紹介していません。次からその違いを見ていくことにしましょう。

値の連続性と範囲

関数の引数の定義域（domain）や値の値域（range）について考えてみます。小学校低学年なら四則演算の定義域と値域は自然数の範囲になります。負数を習えば整数の範囲になります。分数や小数を習えば有理数の範囲になり、平方

▼リスト1　副作用のない関数と副作用のある関数（Java）

```
public static int add(int x, int y){ return x + y; }  ←副作用がない

public static int total = 0;
public static int addUp(int x){ return total = total + x; }  ←副作用がある
```

▼リスト2　遅延評価による必須引数評価戦略にしたほうがいい関数（Java）

```
public static int cond (boolean c, int x, int y){ return c ? x : y; }
```

Haskellのモナドは副作用を救うのか?

プログラミング言語Haskellでは入出力の副作用に対応させるために、無限リストによる実装よりも、命令型言語の動作を普通に実現できるように、“後付け”でモナドを導入しました。モナドにより、静的な型付けの厳密さで副作用のある個所とない個所をきっちり分離できるようになりました(ここがモナドのすごいところです)。

これは、関数プログラミングの枠組みで命令型言語を実現するとどうなるか、という解になっています。しかし結局、関数型プログラミングに堂々と副作用を入れてしまいました(なんたることでしょう)。つまりモナドの採用はある意味、関数型プログラミングの敗北です。数学者は見逃してあげてください。しかし善良なプログラマには、そうだとわからないように理屈を並べて煙にまいています(という噂があります)。噂ですから安心してください。

Common Lispと無限長の整数

Common Lispでは一般的に無限長整数をサポートしていますので、(メモリの許す範囲内で)無限長の整数が扱えます。Javaにも無限長整数としてjava.math.BigIntegerがありますが、四則演算にもいちいちメソッド呼び出しが必要です。さらにCommon Lispには分数型がありますので、分数(循環小数)の範囲では近似値でなく正確です。2割る6は1/3と分数を返します。ここはCommon Lispの自慢できる点です。

▼表1　数学とプログラミングにおける値

	定義域と値域	説明
数学	自然数、整数、分数、無理数、虚数	範囲は無限
Java	整数、離散的な小数(浮動小数点数)	範囲は有限
Lisp	整数、分数、離散的な小数、離散的な複素数	範囲はメモリ限界まで

根などの無理数を習えば実数、虚数を習えば複素数になります。有理数や実数、複素数の段階になると定義域や値域は離散的でなくなり、稠密(ちゅうみつ)になり、連続になります。

一方、デジタルコンピュータはどこまでいっても離散的な値を使います。floatも近似値という離散値になります。数学とプログラミングはまずここが違います(**表1**)。

加えて、コンピュータの整数は数学の無限長の整数とは違い、ある範囲内に制限された整数です。これは整数と呼ぶにはあまりにもインチキで詐欺的な整数です。intさん、あなたのことですよ。longさんでも駄目です。さらに悪いことに、範囲を越えるとエラーになるのはマシなほうで、符号が逆転したりもします。まったく、無責任な整数です。

このように、プログラミングの関数の定義域と値域の範囲は小さいもので、無責任なものです。「コンピュータって、なんて正確でなく、それに小さな世界なんだ!」(数学者からの感想)。

このような違いはコンピュータから見れば、いちゃもんのように思えるでしょう。理想世界の数学でもない限り無限は存在せず、現実世界の工学では有限が当たり前です。離散的なのが悪いのであれば、CDを聞かずにアナログレコードを聴けと言いたくなります。それよりも何よりも、これは関数型プログラミングの責任ではありません(きっぱりと、責任を一般のプログラミングに転嫁)。

未定義のエラー

小学校で習った算数では、四則演算の領域は定義されている範囲で計算していました。つま

りプログラミングでよく起こる範囲外のエラーはありませんでした。x+yは必ず計算でき、値も範囲内に収まっていました。

でも割り算だけはそうではありませんでした。小学校のときにx÷0はやってはいけないと言われていました。つまり0除算を含んだ代数ではなく、0除算を除いた代数でした。一方、プログラミングの世界では0除算も含めたプログラムが要求されます。つまり定義域として第2引数に0を含み、値域としてエラーを含むようになります(**リスト3の(1)**)。

もちろん数学でもこれに対応するために、たとえば「⊥(ボトム)」のような、未定義の値を半順序で定義することになりますが……もちろん小学校では習いません。

しかしプログラミングの世界では「正常系はサルでも作れる。例外処理を作れるのはヒトだ」という言葉があります。実際、プログラミングの多くの時間が例外処理に費やされています。残念ながら関数型プログラミングでもそうです。0除算だけではありません。

たとえば、最初のfact(**リスト3の(2)**)は負数の入力に対しては未定義になります。実際にはスタックを使い潰してスタックオーバーフローのエラーになります。そして例外処理を入れることになり、2番めのfact(**リスト3の(3)**)になります(なお、3番めのfact(**リスト3の(4)**)は階乗計算の定義に反します)。このように、プログラムは段々と例外処理だらけとなっていき、きれいな数学の世界から乖離(かいり)していきます。実に残念です。

半順序

半順序(partial order)とは、任意の2個の要素すべてに順序が定義されていない順序のことを言います。身近なところでは、複素数どうしの順序が挙げられます(一方、任意の2個の要素すべてに順序が定義されているものを全順序と言います)。値が定義されているかどうかを「未定義⊑定義」で示せば、⊥⊑1や、⊥⊑2の順序はありますが、1と2の間では⊑で順序付けできず、半順序になります。

型

プログラミングの世界には型があります。も

▼リスト3　未定義のエラー(Java)

```
// 未定義を含む除算
public static int div(int x, int y){ return x / y; }

// 未定義を含まない除算(1)
public static int div(int x, int y){
  if (y == 0) throw new IllegalArgumentException("0以外の値で割ってください");
  return x / y;
}

// 未定義を含む階乗計算(2)
public static int fact(int n){ return n == 0 ? 1 : n * fact(n - 1); }

// 未定義を含まない階乗計算(3)
public static int fact(int n){
  if (n < 0) throw new IllegalArgumentException("nは0以上にしてください");
  return n == 0 ? 1 : n * fact(n - 1);
}

// 負数に対してインチキな階乗計算(4)
public static int fact(int n){ return n <= 0 ? 1 : n * fact(n - 1); }
```

▼リスト4　型を多用するプログラミング（Java）

```
Person birth(String name, float height, float weight, Sex sex, ...){ …（省略）… }
```

▼リスト5　コンパイラが行う型ディスパッチャをプログラムで表現（Java）

```
public static Object add(Object x, Object y){
  if (x instanceof String) return (String)x + (String)y;
  if (x instanceof Integer && y instanceof Integer) return (Integer)x + (Integer)y;
  …（省略）…
```

ちろん数学にも型の概念を導入した多ソート代数[注2]があり、圏論などもあり、型を扱えます。しかし多ソート代数は一般的でなく、型がない単一ソートの代数が数学だと思われています。もちろん小学校では多ソート代数は習いません。

プログラミングの世界ではこの型を自由自在に使います（**リスト4**）。型をいちいち宣言するのは手間ですが、苦労して型宣言したプログラムは読みやすくなり、何よりもコンパイラが最適化しやすくなります。もちろん実行効率も良くなります。Lispなどの型なし言語もありますが、もちろん内部実装ではきちんと型を持っています。

プログラミングにおける関数も、異なる型に対して多重定義（overload）できます。たとえば、文字列の連結関数は+であり、整数も加算関数も+です。つまり、"abc"+"def"で"abcdef"を返し、さらに1+2で3を返し、さらに"abc"+3のように型を越えて足し算ができます（この場合は文字列の連結関数が呼ばれます）。この点は、プログラミングが数学よりも勝っているように見えます。しかし、このような多重定義は嫌われます。たとえば、言語処理系が実行時に**リスト5**のような仕事をしなければいけません。これは面倒です。

不動点

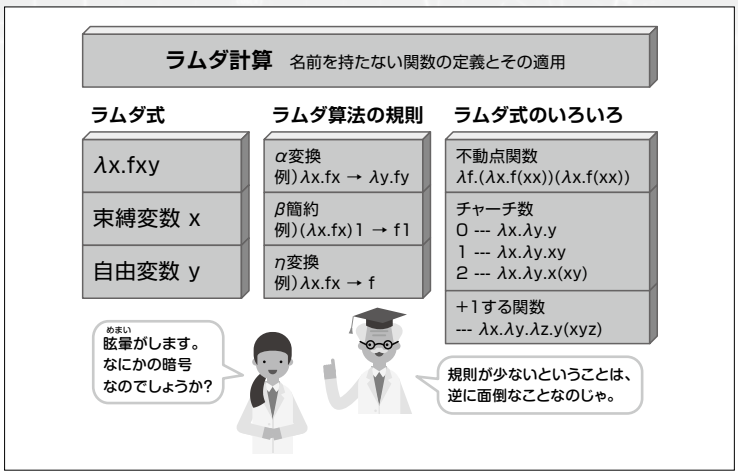
不動点とは`f(x) = x`が成り立つ点のことを言います。不動点は再帰プログラミングや再帰的構造を持つデータ型の意味論を考えるときや、コンパイラの最適化判断に使います。しかし不動点は、一般のプログラミング言語の関数では考えることもしません（きっぱり）。そもそも不動点って何？　おいしいの？　それってプログラミングできるの？という話になります。これは数学者と関数型プログラマ、コンパイラ屋（言語屋）さんぐらいしか気にしないものですが、単位元と同じくらい大事なものです。また不動点は、型のあるプログラミング言語ではプログラムとして表現できませんが、関数型プログラミングの計算モデルであるラムダ計算では不動点関数（不動点オペレータ）もラムダ式で表現できます（これは自慢です）。この自慢のラムダ計算は後述します。

ここまで、数学の関数とはどんなものであったのか、普通のプログラミングの関数との違い、さらに関数型プログラミングの関数との違いを見てきました。それでは次に関数型プログラミングで使う数学分野を見ていくことにしましょう。

最初に紹介したように、関数型プログラミングで使っている数学にはいろいろなモデルがあります。今は群論や圏論などの参考文献が多くありますので、ここでは関数型プログラミングの基礎であるラムダ計算を少し見ていくことにします。圏論よりは簡単ですので安心してください。

注2） 多ソート代数（many-sorted algebra）や混交代数（heterogeneous algebra）では台（プログラミングの型に相当）があり、複数の型を扱うことができます。

▼図2　ラムダ計算



▼リスト6　JavaとLispのラムダ規則

```
// Java[事前準備]
  private static interface Lambda { public Object f(Object x); }
  public static Object f(Object x){ ... }
; Lisp[事前準備]
  (defun f (x) ...) ; Lispの準備は簡単だけど、Javaは準備が面倒……

// Java[α変換]
  Lambda lambda = (x) -> f(x); ⇒ Lambda lambda = (y) -> f(y);
; Lisp  α変換
  (lambda (x) (f x)) ⇒ (lambda (y) (f y))

// Java[β簡約]
  Lambda lambda = (x) -> f(x);
  lambda.f(1); ⇒ f(1)
; Lisp[β簡約]
  ((lambda (x) (f x)) 1) ⇒ (f 1)

// Java[η変換]
  すべてのxで(x) -> f(x);が同じ値を返すとき関数は等価である
  (x) -> f(x) ⇒ f
; Lisp[η変換]
  すべてのxで(lambda (x) (f x))が同じ値を返すとき関数は等価である
  (lambda (x) (f x)) ⇒ f
```

ラムダ計算とは

ラムダ計算とは無名関数（匿名関数）で計算するもので、関数定義と関数適用を3個の規則（α変換とβ簡約、η変換）で表したものです（**図2**）。Lispは、このラムダ計算を基本とする最初のプログラミング言語です。ここはLispの自慢できるところです。Javaにもだいぶ遅れて、このラムダ計算が導入されました。

ラムダ計算は、λx.fxのようなラムダ式で計算します。ラムダ式はλのあとに変数xの束縛（ローカル変数の宣言）を書きます。「.（ドット）」のあとに関数定義をfxのように書きます。このときfは自由変数（グローバル変数で関数名に相当）で、λxで束縛されたxは束縛変数（ローカル変数）になります。α変換はλx.fxをλy.fyのように、束縛変数の名前xをyに自由に変えることができる規則です。β簡約は関数適用の規則で、ラムダ計算の中心になります。(λx.fx)1は束縛変数（ローカル変数）xに1を束縛（代入）して、f1に簡約する規則になります。η変換は関数の等価性（外延性）の規則で、すべてのxに対してλx.fxが同じ値を返すときは、fと等価であるとする規則です。プログラマの方に理解してもらうために、ラムダ規則をJavaやLispで表現してみました（**リスト6**）。

それでは、このラムダ計算は何の役に立つのでしょうか。本来は関数型プログラミングの計算モデルとして理屈を与えて、計算可能性を保証していました。しかしこのラムダ計算は、プログラミングにも大いに役立ちます。実際、ラムダ式の導入がJavaやC#などで行われたのが、その証拠です。

ラムダ計算では関数名を付けずに関数適用を行います。そうなのです。名前をわざわざ付ける必要がありません。プログラマは命名でどん

▼リスト7　JavaとLispでのラムダ式の関数plus

```
// Java[事前準備]
private static interface Binary<T1, T2, R> { R call(T1 x, T2 y); }
// Java[2項演算子plusの実行]
Binary<Integer, Integer, Integer> plus = (x, y) -> x + y; // 関数を値として格納
System.out.println(plus.call(1, 2));

; Lisp[2項演算子plusの実行]
(setf plus (lambda (x y) (+ x y)))   ; 関数を値として格納(Lispは簡単！)
(funcall plus 1 2)
```

▼リスト8　Lispの不動点関数λf.(λx.f(x(x)))(λx.f(x(x)))（ただし内側優先で実行すると無限再帰になります）

```
(lambda (f) ((lambda (x) (funcall f (funcall x x))) (lambda (x) (funcall f (funcall x x)))))
```

なに苦労をしてきたことか。変な名前を付けると情報量がないと怒られ、良い名前を付けたと思ったらコーディング規則に合わずボツにされ……。その苦労から解放されます。どうせ一瞬だけ使う関数にコストを掛けるのは馬鹿げています。文字数もなるべく少なくしてキーボードの打鍵数も少なくしたいものです。これが一番大きい効果です（きっぱり）。

さらに便利なのが（実はこちらが重要だと教科書には書いてありますが）、関数をデータとして扱うという習慣が身に付くことです。**リスト7**の関数plusはラムダ式で表されたデータです。このようにラムダ計算では関数もデータも同等です。どちらもファーストクラスオブジェクト[注3]です。intを扱うのと同じ感覚で関数が扱えます。関数をデータのように気楽に変更でき、人工知能やIoTのプログラミングで求められる動的なプログラミングもお茶の子さいさいです。

不動点関数

ここまでがラムダ計算の実のあるところです。次に、前節でも紹介した不動点関数を見ていきますが、格好付けの意味もありますので、わからなくても大丈夫です。でも知っていると見栄は張れます（たぶん）。それでは不動点関数がラムダ式で表されることを見ていきます。

ラムダ計算の（最小）不動点関数は**F = λf.(λx.f(x(x)))(λx.f(x(x)))**になります。ここから、**F(f)**が**f(Y(f))**になることを示します。

F(f) = λf.(λx.f(x(x)))(λx.f(x(x)))(f)
（fにfを代入）
(λx.f(x(x)))(λx.f(x(x)))
（**f(x(x))**のxに**λx.f(x(x))**を代入）
f((λx.f(x(x)))(λx.f(x(x)))) = f(F(f))

不動点になりました。このようにラムダ計算は不動点も自ら定義できました。この不動点関数をLispで表現してみます（**リスト8**）。

参考までにJavaでの不動点関数の定義も紹介します（**リスト9**）。こちらは遅延評価もしていますので、無限再帰にはなりませんので、安心してください。しかしJavaのジェネリック型は、こうした関数に対して表現力が貧弱ですから、それを使わずにキャスト演算しています。

チャーチ数

また、データ（数値）もチャーチ数と呼ばれるラムダ式で表現できます。たとえば、0はλx.λy.y、1はλx.λy.xy、2=λx.λy.x(xy)になります。さらに1プラスする関数はラムダx.λy.λz.y

注3）ファーストクラスオブジェクト（第一級オブジェクト、first class object）とは、関数の引数にも値にもなり、ほかのどんな場所でも使える万能のオブジェクトです。

▼リスト9　Javaの不動点関数と階乗計算で動作確認（同僚の鈴木寿郎氏作）

```
public class FixedPoint {
    private static interface Nullary { Object call(); }
    private static interface Unary { Object call(Object x); }

    public static void main(String[] args) throws Exception {
        Unary Y = (f) -> { Unary func = (x) -> {
                Object y = ((Unary) x).call(x);
                return ((Unary) f).call(y);
            };

            // 遅延評価の代用として零引数のラムダ式を結果とする関数を作る
            Unary arg = (x) -> (Nullary) () -> {
                Object y = ((Unary) x).call(x);
                return ((Unary) f).call(y);
            };
            return func.call(arg);
        };
        System.out.println("Y = " + Y);

        Unary g = (f) -> (Unary) (i) -> {
            int n = (Integer) i;
            if (n == 0) return 1;
            else {
                // 遅延された評価をここで実行する
                Unary f1 = (Unary) ((Nullary) f).call();
                Object r =  f1.call(n - 1);
                return n * (Integer) r;
            }
        };
        System.out.println("fac(5) = " + ((Unary) Y.call(g)).call(5));
    }
}
```

(xyz) となります。このラムダ式を使うと0+1=1や1+1=2になります（詳細は『はじめてのLisp関数型プログラミング』を参照してください）。不動点関数もデータも関数もラムダ式で定義できるラムダ計算は万能です。万歳！

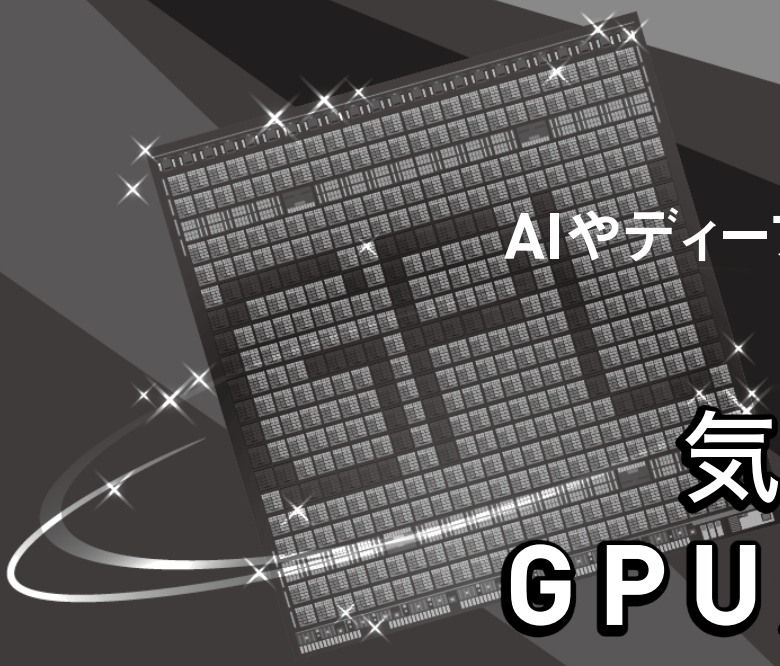
◆ ◆ ◆

関数型プログラミングで使う数学のうち、基礎的なラムダ計算を紹介しましたが、ほかにも帰納関数や群論、圏論などの離散代数や、それらを含む離散数学があります。しかしこれらは、基礎となるラムダ計算を先に学んでおけば、その都度必要になってから学ぶのでも遅くはありません。安心してください。

関数型プログラミングと数学の妖しい関係を見てきました。両者は切っても切れない縁で結ばれています。しかし、まだまだ紹介していない帰納関数、項書き換え系、そして群論から圏論、代数学などがあります。これらはプログラムの意味論を論じたり、コンパイラの最適化で使ったりするだけではありません。もっと実務的な、副作用のないプログラミングの方法や遅延評価をする理論的背景になり、関数型プログラミングを暗記の世界から応用の世界へ導いてくれます（重要なことですから2回めです）。

若いときに数学を学ぶほうが、そのあとでそれを使える時間が多い分、お得です。この意味では、小学校でプログラミングを習うよりも算数をしっかりと身に付け、数学的なプログラミングを身に付けたほうが良いかもしれません。これからも楽しみながら数学とプログラミングをしていきましょう。SD

第2特集



AIやディープラーニングで注目される 気になるGPU超入門

GPU（Graphics Processing Unit）はその名のとおり、画像処理用の装置です。高速でグラフィックス処理を行うGPUは、おもにリアルタイムゲームや、画像処理ツールの高速化に利用されてきました。しかし最近では、機械学習やディープラーニングにも利用されています。

本特集では、GPUの入門的なことから最新のテクノロジまでを解説します。また、GPUの機能を機械学習やディープラーニングに利用できるように、NVIDIAが提供しているGPUコンピューティング環境の「CUDA」について、実際に体験してみましょう。

第1章 GPUとGPUコンピューティング

Author 鶴長 鎮一（つるなが しんいち）
ソフトバンク㈱／サイバー大学 講師

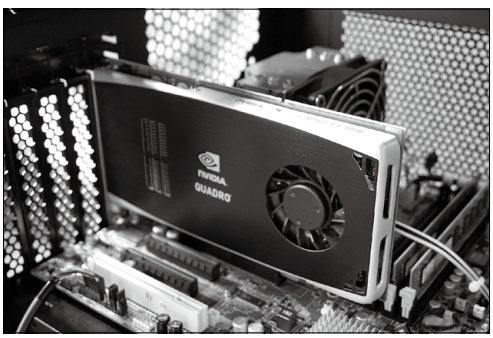
グラフィックス処理に優れたGPU（Graphics Processing Unit）を、科学技術計算のような汎用計算に利用する「GPGPU（General-purpose computing on GPU）」が、機械学習やディープラーニングといった技術とともに注目されるようになっています、中でもNVIDIAが提唱している「GPUコンピューティング」は優れた開発環境により普及が進んでいます。本章ではGPUコンピューティングの利用を前提に、その中核となっているGPUについて解説します。

GPUの利用

GPUはグラフィックス機能を制御するプロセッサです。PCのマザーボード上にある、PCI Expressのような拡張スロットに接続することによって、「グラフィックボード（**写真1**）」として使用できるようになります。GPUとグラフィックボード（通称、グラボ）を同じ意味で使ったり、「ビデオカード」と呼称したりしますが、総称として「GPU」と呼ぶことが多くなっています。

既成のPCを購入すれば、わざわざGPUを増設しなくても標準でGPUが搭載されています。エントリ向けPCを中心に使われているのが、Intelの「Core iシリーズ」や、AMDの「APU」のように、CPUにGPUコアがパッケージ化された「統合型GPU」です。PCI Expressのような、PC内部の拡張スロットに増設する「単体GPU」に比べて、統合型GPUは性能面で劣るものの、消費電力が低く、また低コストで生産できるため、広く使われるようになっています。統合型GPUは、ネット接続やオフィス系ソフトの利用といったライトユースはもちろん、複数のディスプレイに画面を出力する「マルチモニター環境」や、3Dゲームを楽しめる程度の3D描画にも対応しています。最近はWindowsやmacOSのように、GUI（Graphical User Interface）の一部に3D視覚効果を使うようになっていますが、その程度の3D描画なら問題なく処理できます。統合型GPUの性能は年々向上しており、よほど高いグラフィックス処理を必要としない限り、ミドルクラスやハイエンドクラスのPCにも利用が拡大しています。

▼写真1　グラフィックボード



多くのユーザにとって、わざわざGPUを選択して搭載する必要性は少なくなっている一方、単体GPUが求められるケースもいまだに多くあります。ハイクオリティな3Dゲームを楽しむための「ゲーミングPC」や、CADや3Dグラフィックスを利用するための「ワークステーション」では、単体GPUの能力が必要不可欠です。また多くのプロセッサを利用し並列処理を行う「HPC（High-performance computing）」の分野で

も、単体GPUの活用が進んでいます。近年GPUが注目されるようになったのも、HPC分野での活用が成果を上げるようになったことが大きな一因となっています。

単体GPUが必要とされる用途では、PC内部のPCI Expressのような拡張スロットにグラフィックボードを増設します。CPUに内蔵される統合型GPUと区別するため、「外部GPU」と呼んだり、「ディスクリートGPU」と呼んだりします。内部拡張スロットを持たないノートPCや、カードを納めるスペースのないスモールファクターPCでは、「外付けGPUボックス(**写真2**)」を使用することで、外部GPUを増設できます。外付けGPUボックスは、Thunderbolt3ケーブルやUSB 3.1ケーブルでPCと接続します。そのため拡張性の低いPCにも対応できます。

GPUの歴史

現在、グラフィックボードとして市販されているのは、NVIDIAかAMDのチップを搭載したものになります。チップはボードベンダに供給され、ボードベンダがグラフィックボードに仕立てて販売を行っています。複数のベンダから発売されているように見えても、実際にチップを供給しているのは2社になります。Intelはディスクリート向けにはGPUの提供を行っていません。

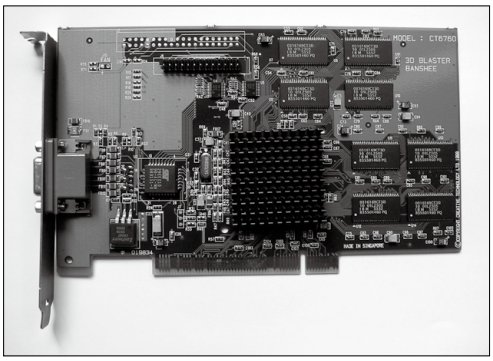
GPUチップの黎明期

現在は数少ないGPUチップベンダですが、1990年代半ばまで遡(さかのぼ)れば、3dfx、S3、Matrox、C&T、ATI(後にAMDに買収)など、多くのベンダがGPU用チップを提供していました。MicrosoftがWindows 95を発売し、OSのGUIを描画するのにGPUが欠かせないものとなっていた時期です。CPUがVRAM(ビデオメモリ)に書き込んだデータを読み取り、ディスプレイに表示できるような信号に変換するのが、それまでのビデオコントローラの役目でしたが、Windows 95が登場するころから、ハードウェアによるアクセラレータ機能を使って、CPUの描画処理能力を補うのが大きな役割となっていました。まだGPUという言葉もなく、2D描画機能に特化した「グラフィックアクセラレータ」や「Windowsアクセラレータ」といったものがコンシューマ向けに数多く販売されていたころです。

そんな中、3Dアクセラレータ機能に対応した、3dfx社の「Voodooシリーズ(**写真3**)」が発売されると、アーケードゲーム機並みの3Dグラフィックスを描画できると一気に注目が集まりました。Voodooシリーズが発売された当初は、2D描画専用アクセラレータと併設するなど、3D描画が一般には特別なものとして見られていました。当然、GUIの視覚効果にも3D描画は使われておらず、またまだフルカラーでSXGA解像度を出力できる2Dアクセラレータが高級とされていた時代です。Voodooシリーズ専用3DグラフィックスAPIの「Glide」に対応したゲー

▼写真2　NVIDIA純正の外付けGPUボックス

▼写真3　1990年代後半にPCゲームの市場で大きなシェアを獲得した3dfx社のVoodooシリーズ(写真はVoodoo Banshee)



ムソフトも数多く販売されていました。今では3Dグラフィックスを描画するためのAPIとして、Microsoft社の「Direct3D」が、Windows OSやXbox向けに提供されており、デファクトスタンダードとして使用されていますが、当時はベンダ各社が独自APIを提供していたため、ゲームソフトを購入する際は、対応しているグラフィックボードかどうかを確認する必要がありました。

GPUチップベンダの統廃合期

こうしたGPUチップベンダが乱立した状況も、2000年に入るころには一変し、ベンダの統廃合が進みます。また3D描画のためのAPIも、ゲームやエンターテイメント向けにはDirect3Dが、ワークステーションや業務向けにはOpenGLが定着します。GPUを選択するうえで、Direct 3DやOpenGLとの互換性の高さが重要な判断基準になったため、独自APIは姿を消すことになりました。各社はDirect3DやOpenGLのバージョンアップに追従するように、より高精細でより高品質なグラフィックスをリアルタイムに再現できるようGPUの性能を高めていきました。その大きな進化がDirectX 7で標準化された「ハードウェアT&L(座標変換＆ライティング)」です。空間に置かれた立体モデルの座標をスクリーン座標に変換(Transformation)する計算や、影を作るライティング(Lighting)計算は、それまではソフトウェアによって行われていましたが、ハードウェアT&Lをサポートすることで、GPU上でより高速に行えるようになりました。

プログラマブルな3D演算機能内蔵へと進化

ただしハードウェアT&Lでは、あらかじめ決められた3D演算しかできませんでした。プログラマが自由にグラフィックス処理をコーディングできるようにしたほうが、3Dグラフィックスの表現力を上げることができます。そうした目的でDirectX 8ではプログラム可能な2種類のハードウェアユニットが導入されました。図形の頂点座標のデータを変換する「バーテックスシェーダ(Vertex Shader)」と、ピクセル単位で陰影処理を行う「ピクセルシェーダ(Pixel Shader)」です。自由にプログラムできることから、「プログラマブルシェーダ(Programmable Shader)」と呼ばれていました。両シェーダはDirect Xのバージョンアップとともに、より高性能に、より汎用性の高いものに進化し、ついにDirectX 10では、バーテックスシェーダやピクセルシェーダなど、複数のシェーダ機能を併せ持った「統合型シェーダ(Unified Shader)」がサポートされるようになりました。

それまで、バーテックスシェーダはフル稼働しているのにピクセルシェーダは休止している状態にあったり、その逆の状態にあったりとGPU全体としては非効率な動きをすることがありました。1つのシェーダがすべてのシェーダ機能を受け持ち、複数のシェーダが同時並列的に稼働できるようにした方が、より効率よくGPUの性能を引き出すことができます。そのために統合型シェーダが考案されました。なおプログラム可能なハードウェアユニットのシェーダを、グラフィックス用途以外にも使うことから「GPUコア」と呼ぶようになっています。

すべてのシェーダ(GPUコア)を無駄なく稼働させることで、CPUのマルチコアを上回る計算能力を引き出すことができるようになり、また統合型シェーダによるハードウェアレベルでの汎用化が実現したことで、GPUをグラフィックス処理以外の「GPGPU」と言われる汎用計算に利用できるようになりました。こうしたGPUの発展により、今ではスーパーコンピュータにも利用されるまでになっています。

GPUを機械学習／ディープラーニングに利用する

グラフィックス処理に特化したGPUは、汎用CPUに比べて回路がシンプルです。そのため、演算コアのシェーダ数を増やす「メニーコア化」

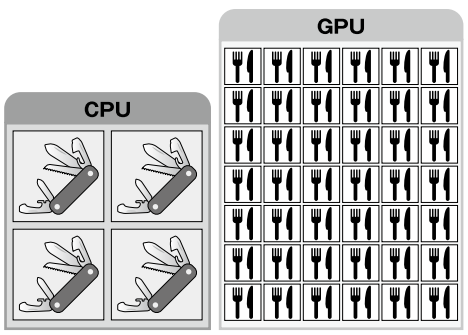
が劇的に進んでいます(**図1**)。5年前のコンシューマ向けGPUでも240個、2017年10月現在市販されているHPC向けのハイスペックなGPUになると、5,120個のGPUコアを搭載しているものもあります。3Dグラフィックス処理では、描画対象を移動させたり回転させたりするのにすべての座標に対して同じ計算をする必要があります。メニーコア化は、こうした描画処理を高速化させる手段として用いられてきましたが、大量のデータに対して同じ演算を並列で行う機械学習やディープラーニングのような分野にも、メニーコアは有効です。また座標計算で使われる行列の積和演算は、ディープラーニングで頻繁に使用されます。

さらにGPUはスケーラビリティに富んでおり、複数のGPUを増設することで、処理能力を向上させることができます。CPUを増設するよりコストを抑えることができ、増設できる数もCPUに比べて多くなります。ディープラーニングの分野では、高価なスーパーコンピュータより、GPUを使った汎用システムのほうが、費用対効果が高くなります。

2012年にGoogleは、YouTubeにアップロードされている動画から、ランダムに取り出した200×200pixelサイズの画像を1,000万枚用意し、これにディープラーニングを適用することで「ネコを認識するAI」を開発したと発表しました。世界中に衝撃を与えたこの研究では、合計2,000CPU、16,000CPUコアを持った1,000台のサーバでシステムを構築し、それを3日間稼働させることで成果が得られていました。ところがその翌年発表された、米スタンフォード大学のAndrew Ng氏とNVIDIAの共同研究では、Googleの構築したシステムに比べて、6.5倍の処理能力を持ったAIネットワークを、たった3台のGPUを使ったサーバで構築できることが示されました。このときに使われたのは、12基のNVIDIA製GPUを搭載した2万ドルのシステムです。その後はGoogleもGPUを採用するようになり、ほかのAI研究者もGPUを使った安価なシステムを使って、より身近な環境で研究するようになりました。

ハードウェアとしての性能や費用対効果の高さに加え、開発環境や実行環境といったソフトウェアの面でも整備が進んでいます。GPUを使ったソフトウェア開発には、エンジニアの育成が欠かせませんが、NVIDIAが提供している「CUDA(クーダ)」を始め、AMDやIntelが開発に加わっている「OpenCL」など、GPGPUのためのフレームワークを使うことで、すぐにGPUを使ったサービスを開発できるようになっています。また「Caffe」や「Chainer」のようなディープラーニングのためのライブラリやフレームワークは、コードを書かなくてもGPUを使った高速な演算処理が利用できます。

▼図1 CPUとGPUのコアのイメージ



CPUコアは複雑な命令も処理できる反面、コア数を増やすのが難しい。GPUコアは単純な命令にしか対応していない分、コア数を増やしやすく、大量の単純な処理を同時並列的に処理できる。

グラフィックボードを増設するには

市販のグラフィックボードでも、機械学習やディープラーニングのフレームワークを実行できます。ゲーミングPC用に売られているものや、自作PC向けのものなら入手しやすく、入門用に利用できます。統合型CPUのようにGPUがオンチップで搭載されていても、増設したGPUに自動的に切り替わります。自動で切り替わらない場合は、手動でBIOS設定を変更することで切り替えます。

スロットの確認

グラフィックボードは、PC内部のマザーボード上のPCI Express拡張スロットに増設します。x16／PCI Express 2.0／PCI Express 3.0と、インターフェースも年代により異なるため、購入前に互換性を確認しておきましょう。またPCケースの大きさや内蔵されているマザーボードの規格も確認しておきましょう。マザーボードの規格にはおもに**表1**のような3種類のフォームファクタが利用されています。メーカー製PCに使用されているのは「ATX」や、ひと回り小さい「Micro-ATX」です。組み込み向けや省スペース型PCに使われるのは「Mini-ITX」です。最低限PCI Express拡張スロットがないとGPUを内蔵できません。なおフォームファクタには上位互換があるため、ATX用ケースにMicro-ATXやMini-ITXのマザーボードを取り付けることはできます。GPUを増設するスペースがないPCケースでは、より大型のPCケースに移植することで、大型のGPUを増設できるようになります。その場合マザーボードのほか、電源ユニットや各配線も移植することになるため大掛かりな作業になります。

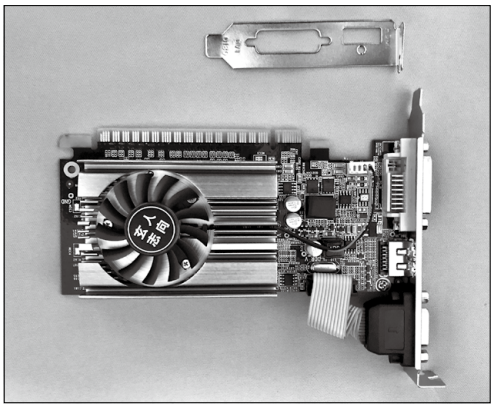
ブラケットサイズの確認

グラフィックボードの大きさは、通常サイズのボードに比べてブラケット（拡張カードをケースに固定するための金属板）が短い「ロープロファイル（**写真4**）」と呼ばれるカードから、奥行きが30cmを超える「フルサイズ（**写真5**）」のものまでいろいろです。パフォーマンスに比例してサイズも大きくなるため、PCケースによっては収納できない場合があります。またカードの厚みもパフォーマンスに比例して大きくなります。グラフィックボードによっては、2スロットを占有するものがあったり、3スロットを占有する大型クーラーを搭載したものもあったりします。その分PCIスロットが使えなくなり、ほかのカードが挿せなくなります。

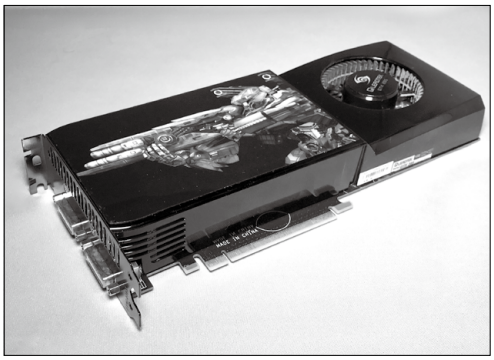
電源容量の確認

加えて、消費電力が高い高性能なグラフィックボードだと、PCI Express拡張スロットからの電源供給だけでは足りず、補助電源が必要に

▼写真4　ロープロファイル（上）とノーマルサイズ（下）のブラケット



▼写真5　奥行きが30cmを超えるフルサイズのグラフィックボード



▼表1　おもに使用されるPCのフォームファクタ

フォームファクタ	サイズ	特徴
ATX	305mm × 244mm	メモリスロットは4〜8基。拡張スロットが最大7基。拡張性が最も高い。ケースも大型化
Micro-ATX	244mm × 244mm	メモリスロットは2〜4基。拡張スロットは最大4基。マイクロタワーやミニタワーのような省スペース型デスクトップパソコンに用いられる
Mini-ITX	170mm × 170mm	とにかく小さな自作PCを作りたい場合に使用

なります(**写真6**)。電源ユニットから6ピンや8ピンの補助電源ケーブルが出ていれば、グラフィックボードに挿入できますが、出ていない場合は「補助電源変換ケーブル」を別途購入することになります。ただし変換ケーブルで補助電源が供給できるようになっても、電源ユニットの容量が足りないと正常に動作しません。PCの電源ユニットには「ATX電源」か、Micro-ATX用に小型化した「SFX電源」です。SFX電源は騒音が抑えられる分、ATX電源より出力が小さくなります。メーカー製PCの場合、SFX電源が使われることが多く、電源容量にも余裕がありません。グラフィックボードに補助電源を供給する場合、必要な電源容量は説明書に記載されています。事前に調べて、PCの電源ユニットに余裕があるか確認しておきましょう。

熱対策

消費電力が大きくなると発熱量も増えます。グラフィックボードを増設する場合、スペースに余裕がないと排熱効率が悪化し、冷却不足から動作が不安定になったり、パフォーマンスが低下したりします。オンボードの冷却ファンに加え、PCケースにも排熱ファンや冷却装置が必要になるケースもあるため、注意が必要です。

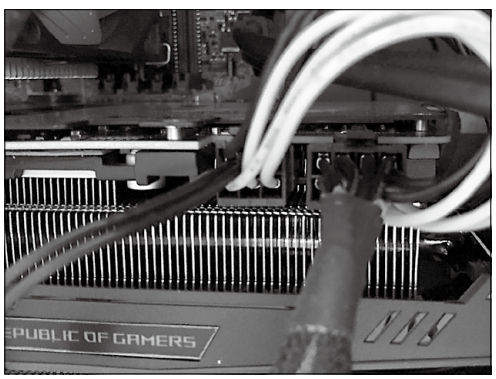
NVIDIAのGPUコンピューティング

GPUが機械学習やディープラーニングのようなAI研究に活用されるようになったのは、NVIDIAの働きが大きく関与しています。スタンフォード大学のAndrew Ng氏と共同で研究を行い、Googleの構築したサーバを凌ぐGPUコンピューティング環境を作り上げたのもNVIDIAでした。NVIDIAはGPUの計算処理能力を活かした開発環境の「CUDA(クーダ)」を2006年に発表し、グラフィックス処理以外のGPUの活用方法を早い段階から模索していました。CUDAは誰でも無料で使えるGPGPUのための開発環境です。第2章で解説しますが、GeForceシリーズのようなコンシュマ向けGPUを使って、機械学習やディープラーニングのためのプラットフォームを個人でも作ることができます。

NVIDIAのGPUによって医薬品の効果を分子レベルのシミュレーションで測ったり、人間の臓器をCTスキャンの映像から立体化したり、銀河レベルのシミュレーションを使って宇宙の原理を解明したりと、GPUを使った汎用計算が「GPUコンピューティング」という言葉とともに普及しました。最近もGPUを使ったAI車載コンピューティングプラットフォームの「NVIDIA DRIVE PX」がトヨタ自動車との協業で話題になりました。今では“GPUコンピューティング”が一般用語として使われるようになっていますが、本来GPUコンピューティングといった場合、NVIDIAのGPUを使ったGPGPUを指します。こうした誤解を与えるほど、AIプラットフォームにおける、NVIDIAの位置づけが大きくなっています。

本章冒頭「GPUの歴史」で、1990年代後半に3dfxが独自3D APIのGlideとともにPCゲームの市場で大きなシェアを獲得したことを説明しました。その後、汎用的なDirect3Dにより3dfx社はシェアを落とすことになりました。CUDAもNVIDIAの独自APIです。OpenCLのような汎用性の高いAPIに取って代わることもあるかもしれませんが、まだまだハードウェアからソフトウェアまでを手がけるNVIDIAの優位性は続きそうです。SD

▼写真6 GPUの補助電源



第2章 CUDAによるGPUコンピューティングの実践

Author 鶴長 鎮一(つるなが しんいち)
ソフトバンク㈱／サイバー大学 講師

GPUを汎用計算に利用する「GPGPU」を身近なPCを使って実践してみましょう。PCにNVIDIAが提供しているGPUコンピューティング環境の「CUDA」をインストールし、機械学習／ディープラーニングフレームワークの「DIGITS」や「Tensor Flow」を実行する方法を解説します。機械学習やディープラーニングをよく知らなくても、簡単な手順で動作させることができます。

CUDAとは

「CUDA(Compute Unified Device Architecture の略。クーダと発音)」はNVIDIAが無償で提供しているGPUコンピューティングのための統合開発環境です。汎用計算(GPGPU)アプリケーションを、C/C++ベースのプログラミング言語を使って作成できます。CUDAはNVIDIA製GPUにしか対応していませんが、ほかの汎用的なGPGPU実行環境に比べてパフォーマンスが高く、使いやすさの面で優れています。2006年に提供を開始して以来、多くのディープラーニングアプリケーションのベースになっています。NVIDIA GeForceシリーズのような低価格なGPUを使えるため、GPUコンピューティングを身近に体験できます。

対応するGPU

CUDAに対応しているGPUは、NVIDIA製に限定されます。Core iシリーズのようなプロセッサ内蔵型GPUはもちろん、ディスクリート型GPUのチップベンダであるAMD(旧ATI)のGPUにも対応していません。

NVIDIA製のGPUとしては、ゲーム向けのGeForce(ジーフォース)シリーズ、クリエイター向けのQuadro(クアドロ)シリーズ、GPGPU向けのTesla(テスラ)シリーズがありますが、どのシリーズもCUDAに対応しています。NVIDIAはグラフィックボードを製造販売していないため、MSI[注1]やELSA[注2]といったボードベンダが販売しているグラフィックボードを使用します。

NVIDIA製のGPUは世代を重ねるごとに演算コア(シェーダ)数が多くなり、**表1**のようにコアクロックも向上しています。演算コアやシェーダの役割や、GPUコンピューティングへの応用については第1章を参照してください。2008年に発売された「GeForce 200シリーズ」では、統合型シェーダにより、GPGPUにも使えるプロセッサへと変化しました。このころから「シェーダ」と呼ばれていた演算コアが「CUDAコア」と呼ばれるようになりました。

Compute Capability

CUDAも年々バージョンアップを重ねています。そのため古いGPUになると最新のCUDAに対応できません。互換性の目安になるのが、

注1) https://jp.msi.com
注2) http://www.elsa-jp.co.jp

「Compute Capability（以降、CC）」と呼ばれるバージョン番号です。CUDA 8.0をインストールするには、最低CCが2.0以上のGPUを使用する必要があります。また最低CCを満たしていたとしても、CUDAアプリによっては実行できないものもあります。たとえば米Googleの機械学習ライブラリ「TensorFlow」をGPUを使って実行する場合、CC 3.0以上のGPUが必要になります。CCによってインストールできるCUDAのバージョンは**表2**のとおりです。

NVIDIA GPUのCCは、「CUDA GPUs[注3]」で確認できます。GeForce 200シリーズのような古いGPUの情報は、「CUDA Legacy GPUs[注4]」で確認します。

CPUの増設

NVIDIAのGPUには、100万円を超える「Quadro GP100」から4,000円の「GeForce GT 710」まで数多くあります。エントリ向けの「GeForce GT 710」でも、最新のCUDAに対応しているため、トレーニングや入門目的なら十分です。古いMacBookやゲーミングPCを使ってCUDAを試す場合、CCが低い古いGPUだと、CUDAのインストールに苦戦することになるため注意してください。

注3） https://developer.nvidia.com/cuda-gpus
注4） https://developer.nvidia.com/cuda-legacy-gpus

なおグラフィックボードは、PC内部のPCI Express拡張スロットに増設します。増設方法や注意点については第1章を参考にしてください。

CUDAのインストール

CUDAをLinuxディストリビューションのUbuntu 16.04 LTSにインストールします。CUDAをインストールできるLinuxディストリビューションは、ほかにFedora/openSUSE/RHEL/CentOS/SLESといったものになります。またWindowsやmacOSにも対応していますが、ほかのアプリケーションとの互換性から、本章ではUbuntu 16.04 LTS（長期サポート版）を使用します。またCUDAも、最新の9.0ではなく8.0を使用します。今回インストールするUbuntuとCUDAのバージョンは次のとおりです。

▼表2　CUDAのインストールに必要なCC

CUDAのバージョン	インストールに必要なCC
5.5	1.0
6.0	1.0
6.5	1.1
7.0	2.0
7.5	2.0
8.0	2.0
9.0	3.0

▼表1　NVIDIA製GPUの比較

GeForce型番	コアクロック	CUDAコア数	ROP数	メモリクロック	メモリ量	TDP
10（10xx）シリーズ　Pascal						
GTX 1080 Ti	1,480MHz	3,584個	88個	10Gbps	11GB	250W
GTX 1080	1,607MHz	2,560個	80個	10Gbps	8GB	180W
900シリーズ Maxwell						
GTX 980 Ti	1,000MHz	2,816個	96個	7Gbps	6GB	250W
GTX 980	1,126MHz	2,048個	64個	7Gbps	4GB	165W
700シリーズ Kepler/Maxwell						
GTX 780 Ti	875MHz	2,880個	48個	7Gbps	3GB	250W
GTX 780	863MHz	2,304個	48個	6Gbps	3GB	250W
200シリーズ						
GTX 295	576MHz	240個	28個	999MHz	1,792MB	289W
GTX 285	648MHz	240個	32個	1,242MHz	1,024MB	183W

OS：Ubuntu 16.04 LTS（日本語Remixも可）
CUDA：8.0（Ubuntu 16.04用deb版）
（※GPUのCCは3.5以上を使用）

Ubuntuのインストール

Ubuntu 16.04 LTSのインストールイメージ注5、または日本語版のインストールイメージ注6をダウンロードします。そしてDVD-RやUSBメモリにライティングしたインストールメディアを使って、PCにUbuntuをインストールします。Ubuntuのインストールは、表示される画面のとおりに行います。特別な操作は不要です。NVIDIA用グラフィックスドライバとして、オープンソース版の「nouveau」がデフォルトでインストールされますが、CUDAをインストールするとNVIDIAのプロプライエタリドライバに置き換わります。

Ubuntuのインストールが完了しUbuntuを起動できたら、aptコマンドでパッケージをアップデートしておきます。Ubuntuでコマンドを実行するには、Ctrl＋Alt＋Tキーでターミナルを起動します。またrootユーザの権限でコマンドを実行する場合、「sudo」をつける必要があります。

```
$ sudo apt update
$ sudo apt upgrade
```

場合によってはこのあと再起動が必要になります。

GPUの確認

増設したグラフィックボードが正常に認識されされているか、次のコマンドを実行しデバイス一覧で確認してください。デバイス一覧にグラフィックボードがないようなら、正しく装着できているかどうか、BIOSで無効化されていないかどうかを確認してください。

```
$ lspci | grep -i nvidia
```

```
01:00.0 VGA compatible controller: NVIDIA Corporation Device 128b (rev a1)
(...省略...)
```

CUDAのインストール

CUDA 8.0をインストールするには、NVIDIAのCUDAダウンロードサイト注7にアクセスし、次のようにプラットフォームやファイル種別を選択し「cuda-repo-ubuntu1604_8.0.61-1_amd64.deb」ファイルをダウンロードします。

- Operating System：Linux
- Architecture：x86_64
- Distribution：Ubuntu
- Version：16.04
- Installer Type：deb[network]

ダウンロードしたあと、次の手順でインストールします。バージョンを指定してCUDA 8.0をインストールするようにします。バージョンをつけないとCUDAの最新版（9.0）がインストールされるので注意してください。

```
$ sudo dpkg -i cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
$ sudo apt update
$ sudo apt install cuda-8-0
```

2.5GBを越えるファイルをダウンロードするため、時間がかかります。

環境変数の設定

CUDAのインストールが完了したら、環境変数を設定し、CUDAのライブラリやコマンドにパスを通します。それには、ホームディレクトリ下にある「.bashrc」ファイルをエディタで開き、**リスト1**のような内容を末尾につけ加えます。

ここまでの作業が完了したら、いったんPCを再起動し、再度ログインしなおしておきます。

動作確認

それではCUDAが正しくインストールできて

注5）http://releases.ubuntu.com/16.04/
注6）https://www.ubuntulinux.jp/News/ubuntu1604-ja-remix
注7）https://developer.nvidia.com/cuda-80-ga2-download-archive

いるかどうか確認しましょう。それには「nvidia-smi」コマンドを実行し、GPUの情報が**図1**のように正しく表示されているか確認します。チップの名称やグラフィックスドライバのバージョンのほか、電力消費量や実行中のCUDAアプリの一覧が表示されます。GPUによっては表示に対応していない項目もあります。

試しにCUDAアプリを実行してみましょう。CUDAをインストールすると「/usr/local/cuda/samples/」ディレクトリ下にサンプルアプリのソースファイルが用意されるため、ビルドして実行できるようになります(**図2**)。

この中で、試しに1_Utilitiesディレクトリ下にある「deviceQuery」をビルドし、実行するには、**図3**のような手順を実行します。deviceQueryを実行すると、より詳細なGPUの情報が表示されます。CUDAのバージョンとGPUのCCが正しく表示されているか確認します。

▼リスト1　「.bashrc」ファイルの末尾に環境変数を追加

```
CUDA_ROOT_DIR="/usr/local/cuda"
export CUDA_ROOT_DIR
PATH="$PATH:${CUDA_ROOT_DIR}/bin"
export PATH
LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:
${CUDA_ROOT_DIR}/lib64/"
export LD_LIBRARY_PATH
```

DIGITS

「DIGITS(Deep Learning GPU Training System)」はNVIDIAが提供している、機械学習／ディープラーニングのためのプラットフォームです。標準でWebインターフェースを備えているため、コマンド操作を知らなくてもGUIで簡単に操作できます。また学習データの作成過程や、モデルの作成過程の可視化にも対応しています。インストールが簡単なため入門向けに

▼図1　nvidia-smiコマンドを実行し、GPUの情報を表示

```
$ nvidia-smi
Sun Oct 22 18:10:52 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.81                 Driver Version: 384.81                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GT 710      Off  | 00000000:01:00.0 N/A |                  N/A |
| 40%   35C    P8    N/A /  N/A |    171MiB /   979MiB |     N/A      Default |
+-------------------------------+----------------------+----------------------+
(...省略...)
```

▼図2　サンプルファイルが用意されている

```
$ cd /usr/local/cuda/samples/
$ ls
0_Simple     2_Graphics  4_Finance      6_Advanced       EULA.txt  bin
1_Utilities  3_Imaging   5_Simulations  7_CUDALibraries  Makefile  common
```

▼図3　deviceQueryをビルドし、実行する

```
$ cd 1_Utilities/deviceQuery
$ sudo make
$ ./deviceQuery
(...省略...)

Device 0: "GeForce GT 710"
  CUDA Driver Version / Runtime Version          9.0 / 8.0    ← CUDAのバージョン
  CUDA Capability Major/Minor version number:    3.5          ← GPUのCC
(...省略...)
```

利用できます。ここで使用するCUDA、Python、DIGITSのバージョンは次のとおりです。

・CUDA：8.0
・Python：2.7（Ubuntu標準）
・DIGITS：5

DIGITSのインストール

DIGITSのサイト[注8]にアクセスし、ユーザ登録を済ませてログインします。簡単なアンケートに回答すると、ダウンロードページにアクセスできるようになります。**図4**のように項目を選択し「nv-deep-learning-repo-ubuntu1604-ga-cuda8.0-digits5.0_1-1_amd64.deb」ファイルをダウンロードし、次の手順でインストールします。

```
$ sudo dpkg -i ダウンロードしたファイル
$ sudo apt update
$ sudo apt install digits
```

1.4GBを越えるファイルをダウンロードするため、インストールには時間を要します。

インストールが終わったら、Webブラウザを立ち上げ「http://localhost:34448」で**図5**のWeb UIにアクセスできるのを確認します。エラー画面が表示される場合は、DIGITSのバックエンドサービスを再起動します。

```
$ sudo systemctl restart digits
```

外部のPCからもアクセスできます。その場合はURLとして「http://サーバアドレス:34448」を指定します。URLさえわかれば、誰でもアクセスできてしまうため、不用意なネットワークでDIGITSを起動しないようにしてください。

DIGITSの利用（MNIST）

「MNIST」データセットを使ってDIGITSの使い方を解説します。MNISTデータセットには**図6**のような28×28pixel 256階調の手書き数字画像が60,000個収められています。MNISTデータセットを学習し、手書き数字を認識できるようなモデルを作成し、実際に手書き数字を認識できるかどうか試してみましょう。

MNISTデータセットのためのディレクトリを作成し、自動ダウンロード用Pythonスクリプトを起動します。

```
$ mkdir ~/MNIST
$ python -m digits.download_data mnist ↵
~/MNIST
```

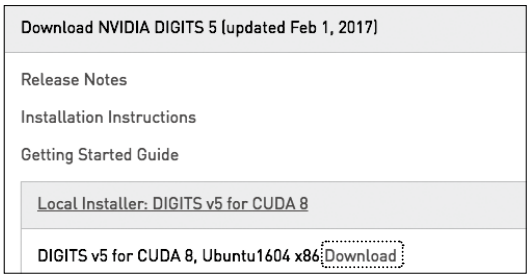
Web UIにアクセスし、画面右上の「Login」をクリックします。画面が切り替わったら、任意のユーザ名を入力し「Submit」ボタンをクリックします。次の手順で作業をすすめます[注9]。

①データセットの読み込み
②学習モデルの作成

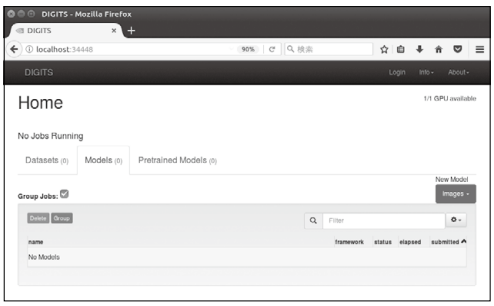
注8）https://developer.nvidia.com/digits

注9）画像つきの詳細な解説は「https://github.com/NVIDIA/DIGITS/blob/master/docs/GettingStarted.md」を参考。

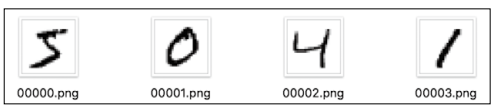
▼図4　ダウンロード時に選択する項目



▼図5　DIGITSのWeb UI



▼図6　MNISTデータセットのトレーニング用画像



③作成したモデルのテスト

①データセットを読み込むには、Web UIで「Datasets」タブに切り替え、「New Dataset」ラベル下の「Images」をクリックし表示されるリストから「Classification」を選択します。「New Image Classification Dataset」画面が表示されたら、次のように入力します。

・Training Images：「~/MNIST/train」をフルパスで指定
・Image Type：Grayscale
・Image size：28 x 28
・Dataset Name：任意の名前（mnist_testなど）

入力が完了したら「Create」ボタンをクリックします。画面が切り替わったら「Job Status」項目が「Done」と表示されるまで待ちます。

②学習モデルを作成するには、「Home」画面に戻り、「Models」タブに切り替え、「New Model」ラベル下の「Images」をクリックし、表示されるリストから「Classification」を選択します。「New Image Classification Dataset」画面が表示されたら、次のように入力します。

・Select Dataset：先の手順で作成したデータセット名をクリック
・LeNet：ラジオボタンをチェック
・Model Name：任意の名前（mnist_modelなど）

入力が完了したら「Create」ボタンをクリックします。

③作成したモデルをテストするには、自分で書いた数字画像を用意するか、MNISTデータセットに含まれる「test」画像を利用します。モデルを作成したあとに表示される画面を「Trained Models」が表示されるまで下にスクロールします。「Upload image」ラベル下の「Browse...」ボタンをクリックし、用意した手書き画像を読み込むか、「~/MNIST/test」にある画像を1つ読み込みます。「Show visualizations and statistics」チェックボックスをオンにしておくと、モデルの各ノードでどのような評価が行われたのかわかるようになります。「Classify One」をクリックし分類テストを開始します。テストが完了すると、**図7**のように、分類結果と予測値が表示されます。

MNIST以外にも、CIFAR-10やCIFAR-100といったデータセットを使って画像認識を試すこともできます。ただしGPUのメモリが少ないと「Out of memory」エラーが発生します。

・CIFAR-10の用意

```
$ python -m digits.download_data ↵
cifar10 ~/○○
```

・CIFAR-100の用意

```
$ python -m digits.download_data ↵
cifar100 ~/○○
```

「TensorFlow（テンソルフロー、またはテンサーフローと発音）」はGoogleが開発している、本格的な機械学習／ディープラーニングのためのプラットフォームです。プログラミング言語「Python」のライブラリとして提供されており、利用する

▼図7　DIGITSによる分類結果

※「Show visualizations and statistics」チェックボックスをオンにしたため、モデルの各ノードでどのような評価が行われたのかがわかる

にはPythonコードの知識が必要になります。

TensorFlowを利用するには、CUDAやPythonのほかに、NVIDIAのディープラーニングライブラリである「cuDNN」が必要になります。多くのソフトが連携するため、バージョンの組み合わせしだいで動作しない場合があります。最新のものを使用せず実績のあるバージョンを使用するようにしてください。ここでは次のバージョンを組み合わせて利用します。

・CUDA：8.0
・Python：2.7（Ubuntu標準）
・cuDNN：6.0
・TensorFlow：1.3.0

cuDNNのインストール

cuDNN（CUDA Deep Neural Network library）をインストールするには、NVIDIAのcuDNNサイト[注10]にアクセスします。ダウンロードの前にユーザ登録を行う必要があります。ログインしダウンロードページにアクセスしたら、**図8**のように項目を選択し、「libcudnn6_6.0.21-1+cuda8.0_amd64.deb」ファイルをダウンロードします。

ファイルをダウンロードできたら、次の手順でインストールします。

```
$ sudo dpkg -i ダウンロードしたファイル
$ sudo apt update
$ sudo apt install cuba-8.0
```

TensorFlowのインストール

TensorFlowはPythonのパッケージ管理コマンドの「pip」を使ってインストールします。Ubuntuではpipコマンドを別途インストールする必要があります。次の手順を実行します。

```
$ sudo apt install python-pip
$ sudo pip install tensorflow-gpu
```

TensorFlowの利用（MNIST）

チュートリアル用のMNISTデータセットを使って、**図9**の手順でTensorFlowを試してみましょう。Ubuntuにgitコマンドをインストールしたのち、チュートリアル用のサンプルコードをGitHubからダウンロードします。ダウンロードが完了したら、サンプルコードを実行できるようディレクトリを移動し、Pythonスクリプトファイルの「fully_connected_feed.py」を実行します。**図9**では評価結果とし

注10）https://developer.nvidia.com/cudnn

▼図8　ダウンロードページで選択する

```
Download cuDNN v6.0 (April 27, 2017), for CUDA 8.0
-->cuDNN v6.0 Runtime Library for Ubuntu16.04 (Deb)
```

▼図9　TensorFlowのインストールと実行手順

```
$ sudo apt install git
$ cd ~
$ git clone --recurse-submodules https://github.com/tensorflow/tensorflow
$ cd tensorflow/tensorflow/examples/tutorials/mnist/
$ python fully_connected_feed.py
(...省略...)
name: GeForce GT 710     ←GPUが使用されていることを確認
major: 3 minor: 5 memoryClockRate (GHz) 0.954
pciBusID 0000:01:00.0
Total memory: 979.81MiB
Free memory: 719.81MiB
(...省略...)
Step 1800: loss = 0.40 (0.004 sec)
Step 1900: loss = 0.40 (0.003 sec)
Training Data Eval:
  Num examples: 55000  Num correct: 49157  Precision @ 1: 0.8938
Validation Data Eval:
  Num examples: 5000  Num correct: 4493  Precision @ 1: 0.8986
Test Data Eval:
  Num examples: 10000  Num correct: 8987  Precision @ 1: 0.8987  ←評価テスト 精度89.87%
```

て、89.87%の精度が得られています。精度は実行するたびに変わります。

ニューラルネットワークの学習過程やデータフローを見るには、「tensorboard」を使って可視化します。ログディレクトリ（/tmp/tensorflow/mnist/）を指定してtensorboardコマンドを実行します（**図10**）。

Webブラウザを起動し、「**http://localhost:6006**」にアクセスすると**図11**のように可視化されたデータが表示されます。外部のPCからもアクセス可能なため、不用意にアクセスされないよう注意してください。

GPU ベンチマークの測定

GPUの性能差がGPUコンピューティングの処理にどの程度影響を与えるのか、2種類のGPUでベンチマークを測定し確認します。使用するのは、ハイスペックモデルのGeForce GTX 1080と、エントリーモデルのGeForce GT 710です（**表3**）。GTX 1080は2017年10月現在、市場価格が7万円程度するのに対し、GT 710は5,000円程度で購入できます。CUDAコア数や、消費電力などの数値も大きく異なります。

Core i7-7700K（4.20GHz）／メモリ32GBのPCに2つのグラフィックボードを挿入し、GPUを切り替えながら以下2つの処理にかかった時間を計測しました。

①DIGITSを使ったMNISTデータセットのモデル作成
②TensorFlowでMNISTチュートリアルを実行

結果は**図12**のとおりです。MNISTのような単純なモデルでも、GPUの性能差が処理速度にあらわれています。さらにメモリやGPUの限界まで使用するような複雑なモデルでは、結果に大きな差が見られることが予想されます。SD

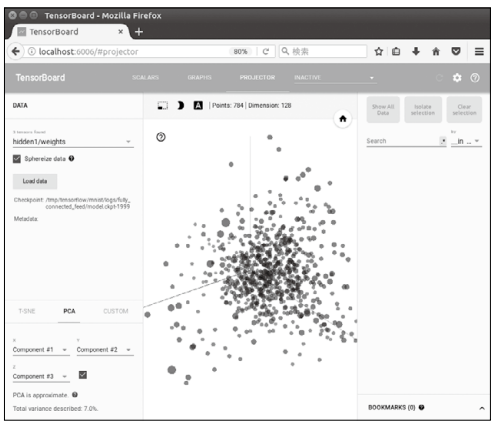
（ベンチマーク協力：緑川耀一さん）

▼図10　tensorboardコマンドを実行しデータフローを可視化する

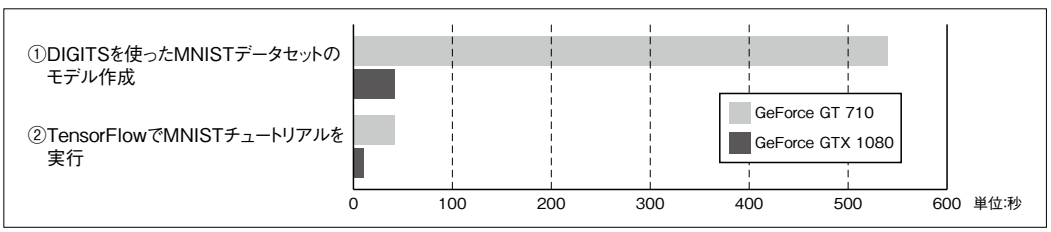
```
$ tensorboard --log=/tmp/tensorflow/mnist/
TensorBoard 0.1.8 at http://gpu02:6006
(Press CTRL+C to quit)
```

※終了するにはCtrl＋Cキーをタイプ

▼図11　tensorboardによって可視化されたニューラルネットワークの学習過程やデータフロー



▼図12　計測結果（グラフが短いほど高速）



▼表3　ベンチマーク測定に使用したGPUのスペック

GPU	GPUアーキテクチャ	CUDAコア数	メモリ容量	消費電力	ベースクロック	Compute Capability
GeForce GTX 1080	Pascal	2,560基	8GB	180W	1,759MHz（オーバークロック）	6.1
GeForce GT 710	Kepler	192基	2GB	19W	954MHz	3.5

第3章 NVIDIAのGPUの歩みから、最新のテクノロジまで

Author 佐々木 邦暢(ささき くにのぶ)
エヌビディア

本章では、グラフィックスアクセラレータとして開発されたGPUの登場からNVIDIAのGPUを取り巻くアーキテクチャの歩み、そしてGPUを使ったHPC(ハイパフォーマンスコンピューティング)からGPU仮想化テクノロジまでを紹介します。

まずは歴史を少し

昨今、「AI」や「ディープラーニング」の文脈でよく取り上げられるGPUは、第1章の解説のように、もともとは3Dグラフィックスを高速に描画するためのアクセラレータとして開発されました。これがどのようにAIやディープラーニングにつながっていくのか、ここではまずNVIDIAのGPUの歴史を少し振り返り、記念碑的なプロダクトを紹介するところから始めたいと思います。

最初の製品:NV1

NVIDIA最初の製品が、1995年に発売されたNV1というグラフィックスチップです。このチップを搭載したビデオカードであるDiamond Multimediaの「EDGE 3D」にはNV1向けに移植された「パンツァードラグーン」や「バーチャファイターリミックス」などの3Dグラフィックスを活用したゲームソフトがバンドルされていました。

「GPU」の登場:GeForce 256

1999年に登場したGeForce 256は、「座標変換(Transpose)とライティング(Lighting)」(略してT&L処理)とレンダリングの機能を1チップに納めた初めての製品で、NVIDIAはこれを「GPU(Graphics Processing Unit)」と名付けました。これが「GPU」という名前と、今に続く「GeForce」製品の誕生です。またこの1999年は、CADなどの高度なグラフィクス用途向けの「Quadro」製品が登場した年でもあります。

プログラマブルGPU:GeForce 3

2001年に登場したGeForce 3は、初の「プログラマブルGPU」でした。これは、バーテックスシェーダやピクセルシェーダの処理を自由にプログラム可能にしたもので、後のGPUコンピューティングにつながる重要な進化です。Microsoftの初代Xboxには、このGeForce 3のカスタム版が搭載されていました。

GPUの特性

さて、ここでGPUはどのような処理に向いているのかをあらためて整理してみます。GPUは前述のとおり3Dグラフィックスの処理を高速化するための装置です。3Dグラフィックスは、三角形のパネルをたくさん組み合わせて表現され、コンピュータはそれを三角形の頂点座標の集合として扱います。3D空間内のオブジェクトを移動・回転させる、視点を変える、あるいは

光の反射を勘案してパネルの色や明るさを決めるといった処理は、膨大な座標の集合に対する数値演算にほかなりません。そしてその計算は、非常にすばやく実行される必要があります。たとえば一般的なPCゲームでも毎秒30回以上、あるいはVRコンテンツでは90回程度、画面の書き換えが必要になってきます。

3Dグラフィクスは年々豪華になり(＝頂点の数が増え)、画面の解像度も高まる一方(＝ピクセル数が増える)ですから、GPUに求められる演算能力もまた年々増大しています。

しかし、これらの処理には「並列に計算しやすい」という特性があります。たとえばT&L処理の“T”である座標変換(Transform)の処理は頂点ごとに独立しており、ある頂点の結果を待たずに別の頂点の計算を同時に行えますし、“L”のライティング(Lighting)処理もまた画面のピクセルごとに独立しているからです。そのため、GPUは並列演算性能を高める方向で進化してきました。CPUが備えるシングルスレッド性能を高めるためのしくみ、たとえばアウトオブオーダー実行や分岐予測といった機能を省き、そのぶん多くの演算器を搭載しているのです。そのため、GPUは同世代のCPUと比較すると10倍以上の理論演算性能を持つことも珍しくありません。

GPUコンピューティングの誕生

さて、2006年ぐらいにまた戻りましょう。プログラマブルシェーダによってGPUの処理に自由度が生まれると、この非常に高い並列演算性能を、「もっとほかの用途に利用できないか」という要求が当然出てきます。それにお応えしてNVIDIAは2006年に「CUDA(Compute Unified Device Architecture：クーダ)」というしくみを発表します。これは、C言語を拡張する形で、GPUを使った並列プログラムの記述を可能にするものです。

また、同時期に発表した「G80」という革新的なGPUチップがGPUコンピューティングの実用化を決定的なものにします。G80は(第1章で説明された)「ユニファイドシェーダ」方式を採用した初めてのGPUで、GeForce 8800(**写真1**)という形で製品化されただけでなく、2007年にはGPUコンピューティングに特化した新しい製品として登場したTeslaシリーズ(**写真2**)にも搭載されたのです。

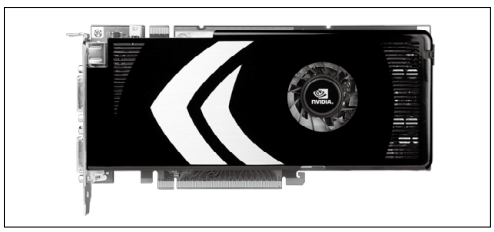
こうして、CUDAとG80、そしてTeslaの登場によって、GPUコンピューティングの時代が本格的に幕を開けました。また、「コンシューマ向けのGeForce」「ワークステーション向けのQuadro」「サーバ向けのTesla」という現在まで続く製品ラインナップが完成したのです。

GPUコンピューティングの進化

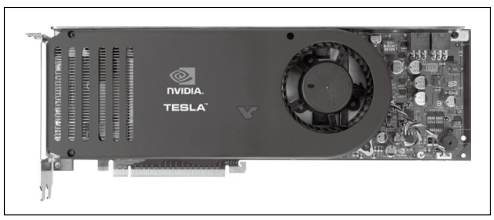
2007年に登場したTeslaですが、早くもその翌年にはHPC(ハイパフォーマンスコンピューティング)の最高峰で実力を示すことになります。東京工業大学のスーパーコンピュータ「TSUBAME」が、Tesla S1070(**写真3**)を170台追加することで「TSUBAME 1.2」へと進化したのです。2008年11月のTOP500ランキング[注1]

注1) 毎年6月と11月に発表される、スーパーコンピュータの性能番付。http://www.top500.org

▼写真1　GeForce 8800



▼写真2　Tesla C870



でTSUBAME 1.2は29位に入り、「世界初のGPUスパコン」となりました。

このあともTeslaの進化とともにGPUスーパーコンピュータが躍進します。いくつか紹介しましょう。

Fermiアーキテクチャ

2009年、NVIDIAは「Fermi」と呼ぶ新たなGPUアーキテクチャを発表しました。この名前は物理学者のエンリコ・フェルミに由来し、これ以降、NVIDIAのGPUアーキテクチャは科学者の名前をいただくようになります。

Fermi世代のTeslaは、倍精度浮動小数点演算の性能が大幅に向上したことが、とくにHPC用途に大きな効果をもたらしました。その結果、2011年11月のTOP500ランキングでは、Tesla M2050(**写真4**)を搭載した中国のTianhe-1Aが首位を獲得。また、新たに構築されたTSUBAME 2.0は、ノードごとに3基のTesla M2050を搭載し、TOP500ランキングで4位という世界トップクラスのスーパーコンピュータとなりました。

Keplerアーキテクチャ

2012年にはKeplerアーキテクチャに基づく製品が投入されます。名前は天文学者のヨハン・ケプラーが由来です。

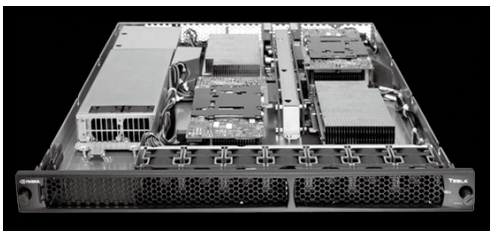
Kepler世代のGPUは、Fermiから電力性能比が大幅に向上したこともあり、多くのスーパーコンピュータに採用されました。本稿執筆時点での最新TOP500ランキングは2017年6月のものですが、なお46のKepler搭載システムが登録されているほどです。その中でも、米オークリッジ国立研究所のTitanはTesla K20x(**写真5**)を搭載して2012年11月のTOP500ランキングで首位を獲得しています。また、東京工業大学が次世代TSUBAMEのプロトタイプとして作成した、油浸冷却方式のTSUBAME-KFCも、Titan同様Tesla K20xを搭載し、電力性能比のランキングであるGreen500で2013年11月、2014年6月の2期連続で世界一位に輝きました。

Maxwellアーキテクチャ

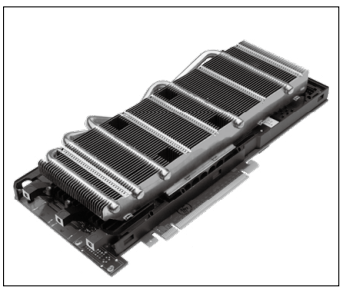
2014年にはMaxwellアーキテクチャが投入されました。Maxwell世代のTesla(**写真6**)は、ディープラーニングなどのワークロードにフォーカスし、単精度演算性能を向上させることを重視しました。そのため、単精度性能はKepler世代の1.4倍ほどになっていますが、倍精度演算性能はKeplerよりも下がっています。

このわりとハッキリした性格付けのため、Maxwellはディープラーニング用途で活用されましたが、倍精度演算性能が必要となるような科学技術計算には向きません。そのため、TOP500リストにランクインするようなスーパーコンピュータにはMaxwell世代のGPUは採

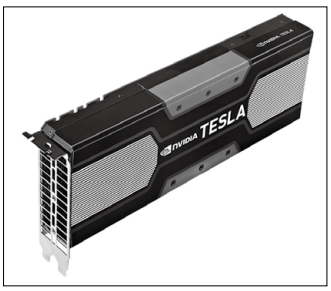
▼写真3　Tesla S1070



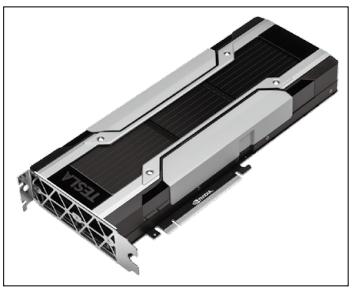
▼写真4　Tesla M2050



▼写真5　Tesla K20



▼写真6　Tesla M40



用されていません。

Pascalアーキテクチャ

2016年の“GPU Technology Conference（GTC）”でNVIDIAは新たな「Pascal」アーキテクチャとそれに基づく「Tesla P100」GPUおよびTesla P100（**写真7**）を8基搭載するサーバである「DGX-1（**写真8**）」を発表しました。Tesla P100は前世代のMaxwellから大幅な進化を遂げています。

・16nmプロセスルールの採用

MaxwellはKeplerと同様の28nmでした。

・積層メモリ「HBM2」を初採用

メモリバンド幅は732GB/sと非常に高速になりました。

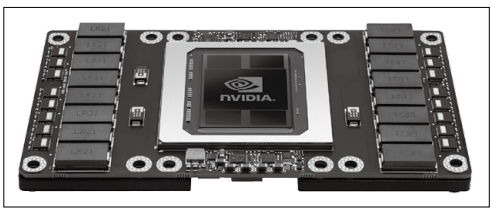
・GPU間接続にNVLinkをサポート

最大160GB/sの帯域幅を持つ新たなインターコネクトです。マルチGPU構成でのスケーラビリティが向上します。

・Unified Memoryの進化

CUDA 8とPascalの組み合わせでは、GPUのメモリサイズに制約されない大きな仮想アドレス空間を扱うことが可能となり、ホスト（CPU）とデバイス（GPU）のメモリが分かれていることを意識しない、シンプルなプログラミングを実現します。

▼写真7　Tesla P100



▼写真8　DGX-1

基本的な演算性能の面でもPascalは進化しており、Tesla P100の理論演算性能は次のとおりです。

・倍精度（FP64）：5.3 TFLOPS
・単精度（FP32）：10.6 TFLOPS
・半精度（FP16）：21.2 TFLOPS

Maxwell世代で「一回休み」となってしまった倍精度浮動小数点演算の性能も、P100ではしっかりと確保されており、HPC用途に対応可能となりました。

また、ディープラーニング用途で重要度の上がってきた半精度（16bit）演算の最適化が図られており、単精度の2倍の性能が出るようになっています。このように、Tesla P100はHPCもディープラーニングもこなせる「オールラウンダー」であり、Pascal世代のフラッグシップ製品です。

また、Tesla製品としては初めて、単体のGPUボードだけではなく、Tesla P100を搭載したサーバをNVIDIAが送り出したこともPascal世代の大きなニュースです。「DGX-1」というこのサーバは、Tesla P100のNVLink版を8基搭載し、4枚のInfiniBand EDRカードも備えてクラスタ構成も可能な大型サーバです。

Tesla P100とスーパーコンピュータ

2017年6月のTOP500/Green500ランキングは賑やかなものとなりました。Kepler以来「久しぶりに倍精度できるやつ」として登場したTesla P100を搭載したスーパーコンピュータが大量にランクインしたのです（**表1**）。その数、実

▼表1　Green500の上位に入った日本のTesla P100搭載システム

順位	システム名	所属
1	TSUBAME 3.0	東京工業大学
2	Kukai	Yahoo! JAPAN
3	AIST AI Cloud	産業技術総合研究所
4	RAIDEN GPU Subsystem	理化学研究所
8	RCF2	国立環境研究所
11	Reedbush-H	東京大学

に25システム。また、Green500の上位をほぼTesla P100搭載システムが独占[注2]しており、これはその電力性能比の高さを端的に表していると言えるでしょう。

日本のシステムが多く上位に入っているのも今回の特徴です。

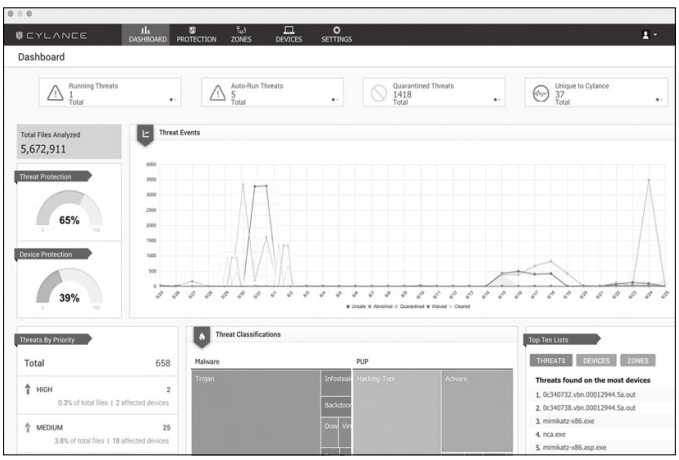
GPUとディープラーニング

さて、ずいぶんHPC(スーパーコンピューティング)の話ばかり書いてきたので、ここで1つ話題を変えましょう。ここ数年、GPUコンピューティングの活用が最も進んだ用途と言えば、やはりディープラーニングでしょう。

注2) 上位16システム中、14システムがTesla P100を搭載。

▼写真9 Horus Technologyの視覚補助デバイス

▼図1 CylancePROTECT



ここでは、まずいくつか活用事例を紹介してから、GPUがどのように役立っているのか、見ていきたいと思います。

写真9はHorus Technology[注3]というイタリアのスタートアップが開発中のデバイスです。「見えないものを聞こえるようにする」と謳われており、視覚障害者の方を補助するものだそうです。ヘッドセットの部分にはステレオカメラと骨伝導スピーカーが内蔵されており、カメラからの入力映像から、物体検出、画像分類を行って、装着者に音として伝えます。当然リアルタイムの画像処理が必要になりますが、有線接続された本体にNVIDIAのJetsonというGPU搭載コンピュータが搭載されており、ここでディープラーニングを使った高精度な画像認識を行います。「GPUを使うことで、CPUに比べ物体認識が48倍速くなった」とのことです。

もう1つ、**図1**はCylancePROTECTというマルウェア対策製品ですが、検査対象ファイルの「有害/無害」を分類するためにディープラーニングを活用しています。2017年5月に猛威を振るったランサムウェア「WannaCry」を、その亜種も含めて正しく検知できたとのこと。

もう1つは自動運転車の事例です。Wepod(**写真10**)というこの6人乗りの小さなバスは、駅から大学までの間という短い決められたルートではありますが、人が操作するためのステアリングもブレーキもない完全な自動運転で往復します。カメラから得た映像の解析に、ディープラーニングが利用されています。

このように、画像の認識を中心にディープラーニングは活用が進んでいますが、現在のような高精度の認識が可能になったのはいつごろからなのでしょうか。

少し調べてみると、2011年ぐらいからGPUを活用したディー

注3) http://horus.tech/?l=en_us

プラーニングの報告が増えてくるようです。たとえばMicrosoft Researchの著者による“Conversational Speech Transcription Using Context-Dependent Deep Neural Networks”[注4]という2011年の論文では、GPUを使ったディープラーニングで音声のテキスト変換（speech-to-text）の精度を高めたことが報告されています。

“We utilize multiple NVidia Tesla GPGPU devices connected to a single host.”

とあるように複数のTeslaを利用し、その結果、

“The speed-up from using GPGPUs is so large”

とのことで、大きな効果が得られたようです。

また、ディープラーニングを一躍有名にした出来事の1つに、2012年のILSVRCでトロント大学のチームが圧勝した事例がありますが、あの論文[注5]には、

“Our network takes between five and six days to train on two GTX 580 3GB GPUs.”

と、ズバリGPUの製品名が入っています。GTX 580（**写真11**）はFermi世代のGeForce上位モデルです。

では、なぜディープラーニングにGPUが有効なのでしょうか？

ディープラーニングの学習処理は、つまるところ行列の積和演算の膨大な繰り返しです。これは、GPUのもともとの仕事である、3Dグラフィックスの座標変換処理などと同じ種類の計算なのです。「あ、俺それ得意だよ、昔からそればっかりやっているのだから！」と、GPUがAI（?）で話せたら言うかもしれませんね。

最新のVoltaアーキテクチャ

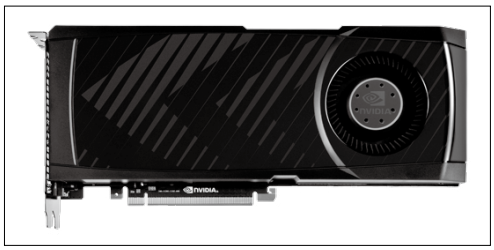
さて、先ほどはG80を搭載した最初のTeslaから、Pascalアーキテクチャの Tesla P100まで年代順に紹介しました。しかし実は、もう1つあるのです。それがGTC 2017で発表された“Volta”アーキテクチャと、その最初の製品である“Tesla V100”（**写真12**）です。

Voltaの話をする前にディープラーニングを持ってきたのにはわけがあります。Tesla V100には、ディープラーニングの学習処理を高速化することに主眼を置いた「新兵器」が搭載されているのです。

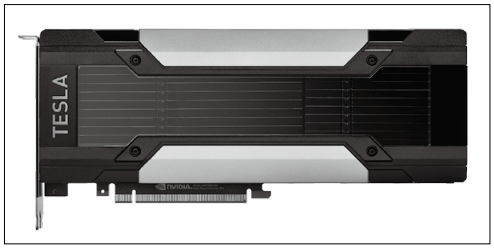
注4）https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CD-DNN-HMM-SWB-Interspeech2011-Pub.pdf
注5）https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

▼写真10　Wepod

▼写真11　ジェフリー・ヒントン先生（トロント大学）も使ったGeForce GTX 580



▼写真12　Tesla V100



「Tensorコア(テンサーコア)」というこの新しい演算器(**図2**)は、「4行4列の半精度(16bit)行列2つの積を取り、半精度または単精度(32bit)の結果に足し込む」という動作を1クロックで行います。その理論演算性能は120 TFLOPSとなり、Tesla P100の単精度性能(10.5 TFLOPS)と比較するならば12倍、半精度性能(21.2 TFLOPS)と比べても6倍という高い性能を持ちます。

このTensorコア、Pascal世代までは存在しなかった新しいハードウェアですから、その効果を引き出すためにはソフトウェア側の対応も必要になってきます。具体的には、Caffe2やCNTK、MXNetといったディープラーニングフレームワークが、Tensorコアに対応する必要があります。そのために、NVIDIAのソフトウェアエンジニアはフレームワークの開発チームと積極的に協業しています。前述の3フレームワークは、今年5月のGTC 2017においてTensorコア対応が発表されました。

また、日本ではPreferred Networks様と連携し、ChainerのTensorコア対応もほぼ完了しています。

さて、このTesla V100は、NVIDIAのDGX-1と、そして新たに登場した「DGX Station(**写真13**)」という水冷のワークステーションに搭載されているほか、2017年中には複数のサーバベンダ様からも搭載機が出荷される予定です。

また、GTC 2017で「EC2にVoltaインスタンスを投入する」と表明くださったAWSは、Tesla V100を最大8基搭載する「P3インスタンス」の提供を始めました。クラウドでもすでにVoltaが利用可能になっています。

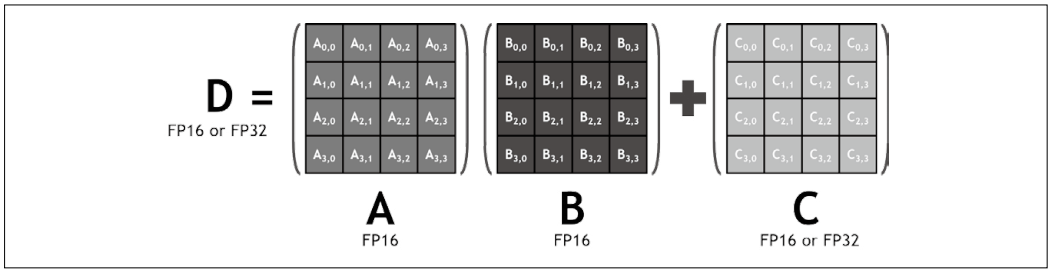
Teslaのもう一つの用途? GPU仮想化

ここまでお伝えしてきたように、Teslaは「サーバ向けGPU」であり、HPCやディープラーニングのようなGPUコンピューティング用途に使われていますが、もう1つ大事な用途があります。それが、GPU仮想化テクノロジを使ったVDI(仮想デスクトップ基盤)用途です。

WordやExcelのようなオフィスソフトウェアをおもに利用するインフォメーションワーカーや、3D CADを活用する設計エンジニアなど、企業内にはグラフィックス要件の異なるユーザに合わせてさまざまな端末が展開されます。これらは台数が増えるほど管理が難しくなり、物理的なコンピュータの設置場所にユーザが縛られるため、ワークスタイルの柔軟性にも欠ける

▼写真13　Tesla V100を4基搭載するDGX Station

▼図2　Tensorコアの行列積和演算



ことになります。

こういったさまざまな端末を仮想化し、Teslaを搭載したサーバに集約することで、どこからでも安全に自分の作業環境にアクセスできるようになります。

GPU仮想化テクノロジによって、物理的には1基のTeslaを論理分割し、複数台のGPUとして利用できますので、ユーザは「まるで自分専用のGPUがあるかのように」グラフィックスアプリケーションを利用できます(**図3**)。

Tesla P100を始めとするPascal世代以降のGPUは、HPCやディープラーニングのみならず、このGPU仮想化用途にも使えるようになりました。

「グラフィックス要件の厳しい端末はVDIできないと思っていた」という方は、ぜひNVIDIAのGPU仮想化テクノロジに触れてみてください。

さて、ここまでGPUコンピューティングの話題を中心にお届けしてきました。NVIDIAの事業領域は広く、これ以外にゲーミングやVR、CADなどの高度なグラフィクス処理、小型機器やロボット、自動車向けのGPUなど、多岐に渡ります。

これら全体の最新情報を一度に吸収できるのが、年1回の技術カンファレンス「GTC Japan」です。今年は12月12～13日の2日間、東京はお台場で開催します[注6]。

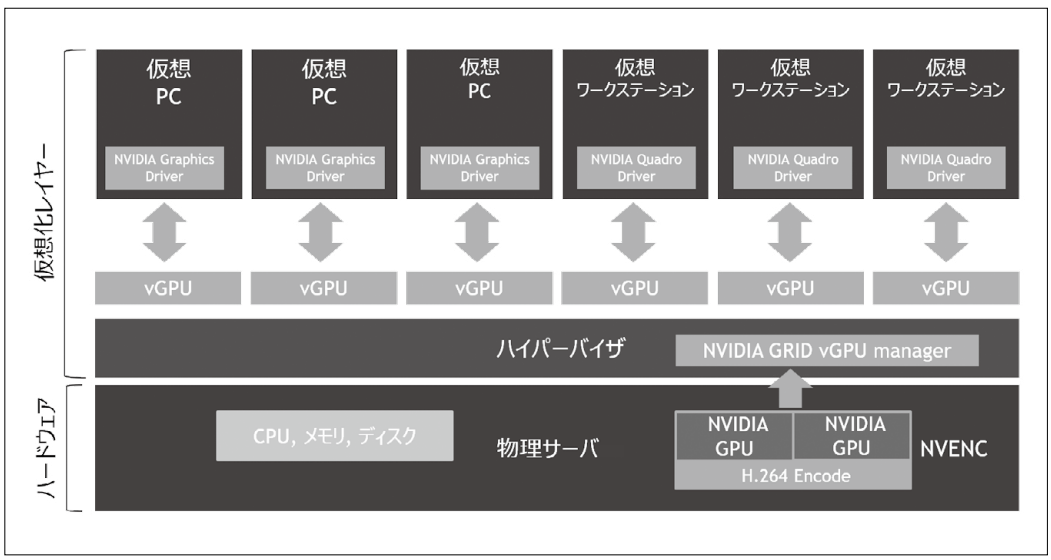
初日にはディープラーニングを実践的に学べる「Deep Learning Institute」のハンズオントレーニングを過去最大の規模で実施するほか、CUDAやOpenACCなどのGPUコンピューティング技術、そして各種ディープラーニングフレームワークに関するたくさんのテクニカルセッションを用意しています。

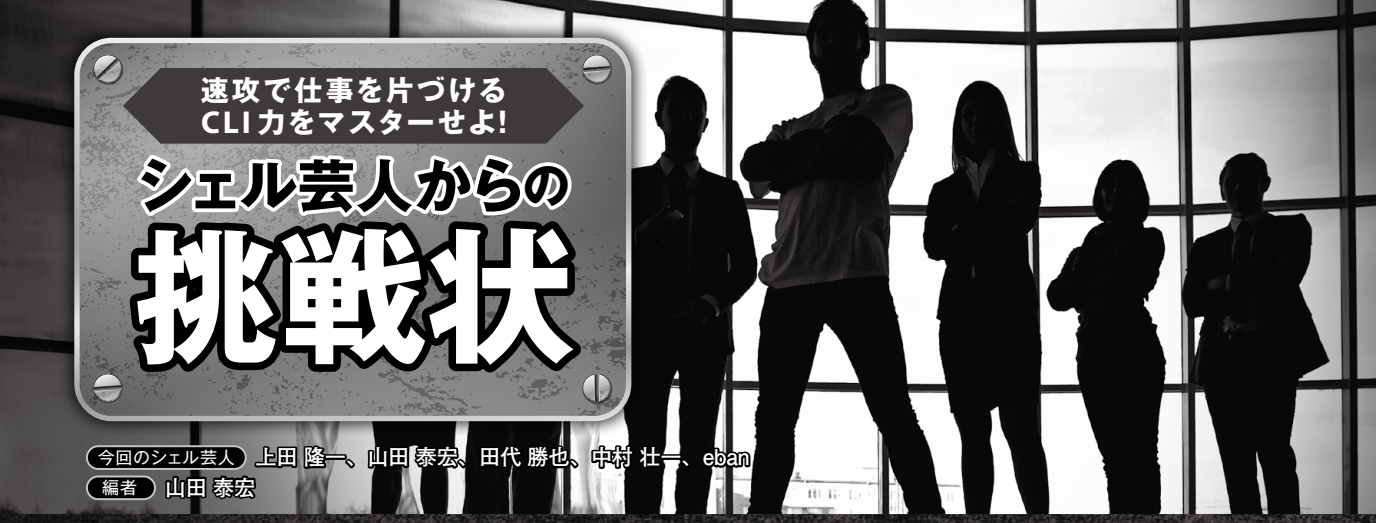
2日目には、CEOのジェンスン・ファンが基調講演に登壇し、数々の発表を行うほか、ディープラーニング、HPC、プロフェッショナルグラフィックス、GPU仮想化、自動車といった種別ごとに専門のトラックを設け、多くのテクニカルセッションをお届けします。

ぜひ、GTC Japan 2017で最新のGPU技術情報に触れてください。SD

注6）https://www.gputechconf.jp/

▼図3　GPU仮想化のしくみ





速攻で仕事を片づける
CLI力をマスターせよ!

シェル芸人からの挑戦状

今回のシェル芸人 上田 隆一、山田 泰宏、田代 勝也、中村 壮一、eban
編者 山田 泰宏

第4回 ネットワーク(その2)

はじめに

今回のテーマは、前回(第3回)に続き「ネットワーク」です。ネットワークに関連する問題を5問出題します。前回を読んでいないといけないということはないので、初めての方もお手元のCLI端末でチャレンジしてみてください。

環境

問題の検証に使ったOSはUbuntu Server 16.04 LTSです。Ubuntuですので、基本的なコマンドはGNU core utilitiesの使用を前提にしています。そのほかのコマンドについては、さまざまなものを試すのでその都度インストールします。解答の解説においてインストールが必要なことを説明します。ただ、aptで簡単にインストールできるものについてはインストールする手順を省略することがあります。シェルは前回に引き続きbashを使います。バージョンは4.3です。

問題1

IPアドレスの追加

初級 ★

出題、解答、解説:上田

ローカルネットワーク環境のLinuxマシンのネットワークデバイス(eth0やwlan0など)に、今使っているIPアドレスとは別に100個以上のIPアドレスを追加してみてください。そのあと、削除してみてください(失敗しても良い環境で試しましょう)。

解答

同じサーバにあるWebサーバやその他サービスを複数提供したいとき、1つのネットワークデバイスにIPアドレスを複数割り振り、アクセスされたIPアドレスでサービスを振り分けることがあります。こういうとき、たとえば既存のネットワークカードにIPアドレスを追加できると便利です。実際それは可能で、コマンドを使う場合は次のように打ちます。

```
$ sudo ip addr add local 192.168.2.150/24 ➥
dev eno1 label eno1:150
```

これは、eno1というネットワークデバイスに追加でIPアドレスを割り振る例です。コマンド実行後、ipコマンドで確認すると、**図1-1**のように既存のアドレス(192.168.2.7)のほかに192.168.2.150が追加されていることがわかります。コマンド中のeno1:150の「150」は、このIPアドレスと結び付けたラベルです。

IPアドレスの割り当てを解除するときは、

```
$ sudo ip addr del local 192.168.2.150/24 dev eno1 label eno1:150
```

と、addをdelに変えます。

さて、これをふまえると、たとえば**図1-2**のような解答を考えることができます。標準入力から割り当てたいIPアドレスの一部の数字をxargsに渡し、-I@で1つずつipコマンドに数字を渡してIPアドレスとラベルを作り、割り当てを実行していくと、一気にIPアドレスが振れます。解除するときも同様です。

この問題は、実用性はあまり考えておらず単なる遊びですが、たとえばIPアドレスを使い切ったあとにDHCPの挙動を見るときなどに使えそうです。テストのとき、このような極端な設定をワンライナーで済ますことは、個人的にはよくあります。

▼図1-1　eno1にIPアドレスが2つ割り振られている

```
$ ip addr | grep eno1
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    inet 192.168.2.7/24 brd 192.168.2.255 scope global dynamic eno1
    inet 192.168.2.150/24 scope global secondary eno1:150
```

▼図1-2　解答

```
↓IPアドレスを割り当て
$ seq 100 200 | sudo xargs -I@ ip addr add local 192.168.2.@/24 dev eno1 label eno1:@
↓確認
$ ip addr | grep 192.168.2
    inet 192.168.2.7/24 brd 192.168.2.255 scope global dynamic eno1
    inet 192.168.2.100/24 scope global secondary eno1:100
    inet 192.168.2.101/24 scope global secondary eno1:101
    (..略..)
    inet 192.168.2.200/24 scope global secondary eno1:200
↓割り当ての解除
$ seq 100 200 | sudo xargs -I@ ip addr del local 192.168.2.@/24 dev eno1 label eno1:@
(このあと、再度確認しておきましょう)
```

問題2

ネットワーク監視

出題：山田　解答：上田　解説：田代

初級 ★

5秒に1回特定のURLにアクセスし、レスポンスのHTTPステータスが200なら“Success”、それ以外なら“Warning”と標準出力に出力するコマンドを考えてください。

解答

CLIで利用できるHTTPクライアントはいくつかありますが、curlコマンドを使った解答例を**図2-1**に示します。

図2-1について説明します。HTTPレスポンスを取得する際は、HEADリクエストを送ることでHTTPレスポンスのヘッダ部のみ受け取れます。**図2-2**は、curlで-Iオプションを使ってHEADリクエストを送り、HTTPレスポンスヘッダを取得した結果です。

HTTPステータスコードの番号はRFC 2616[注1]の6.1.1で**表2-1**のように規定されています。

注1）https://www.w3.org/Protocols/rfc2616/rfc2616.txt

curlの-o ファイル名オプションでHTTPレスポンスをファイルに保存できるのですが、ここではファイル名に/dev/nullを指定して出力を捨てています。curlの-w '%{http_code}'オプションを使うと、ステータスコードの番号のみを表示できます(**図2-3**)。http_code以外にも指定ができます。詳細はman curlでご確認ください。curlは出力先がファイルやパイプの場合に進捗状況などを表示しますが、今回は不要なので-sオプションを指定して非表示にします。

ステータスコードの番号をパイプでawkに渡し、三項演算子を使って200番か否かで表示を切り替えます(**図2-4**)。

あとはwhileループとsleepコマンドを利用して、5秒ごとに実行させて完成です(**図2-1**)。

別解

grepコマンドは指定した文字列の行が見つかると正常終了し、見つからない場合は異常終了します。後半のSuccessとWarningの表示切り替えを、grepの終了ステータスを使って切り替える解答が**図2-5**です。

▼図2-1　解答

```
$ while sleep 5; do curl -Is -o /dev/null -w '%{http_code}' https://www.google.co.jp/ | ↵
awk '{print /200/?"Success":"Warning"}' ;done
Success
Success
(Ctrl+Cで止める)

$ while sleep 5; do curl -Is -o /dev/null -w '%{http_code}' https://www.google.co.jp/missing | ↵
awk '{print /200/?"Success":"Warning"}' ;done
Warning
Warning
(Ctrl+Cで止める)
```

▼図2-2　HTTPレスポンスヘッダを取得する

```
$ curl -I https://www.google.co.jp/
HTTP/1.1 200 OK  ←HTTPステータスの行
(..略..)
Vary: Accept-Encoding
```

▼表2-1　HTTPステータスコードの意味

番号	意味
1xx	Informational - 処理中(負荷が高いときなど)
2xx	Success - 正常処理
3xx	Redirection - 移転通知(リダイレクトなど)
4xx	Client Error - クライアントのリクエストに不備
5xx	Server Error- サーバの処理に不備

▼図2-4　番号が200か否かで表示を切り替える

```
$ 図2-3のコマンド | ↵
awk '{print /200/?"Success":"Warning"}'
Success
```

▼図2-3　HTTPステータス番号のみを表示させる

```
$ curl -Is -o /dev/null -w '%{http_code}' https://www.google.co.jp/
200
$ curl -Is -o /dev/null -w '%{http_code}' https://www.google.co.jp/missing
404
```

▼図2-5　別解(田代)

```
$ while true; do curl --silent --head https://www.google.co.jp/ | head -n 1 | ↵
grep -q '200' && echo Success || echo Warning; sleep 5; done
```

問題3

即席のWebサーバ

出題、解答、解説：中村

初級 ★

あなたは友人にちょっとしたファイルを送信したくなり、即席のWebサーバをたてることにしました。ワンライナーでWebサーバをたててみてください。

解答

Pythonの組み込みモジュールを使えば簡単にWebサーバを立ち上げられます。

```
$ python3 -m http.server 8080
```

これはポート番号8080にWebサーバを立ち上げるワンライナーです。引数を省略すると、ポート番号8000で待ち受けします。また、--bindオプションを付けることで、待ち受けるIPアドレスを指定できます。たとえば、下記を実行すれば、ホスト自身のみがアクセスできるような制限ができます。

```
$ python3 -m http.server 8080 --bind 127.0.0.1
```

これで上記のワンライナーを実行したディレクトリにあるファイルを公開できます。このように即席でWebサーバをたてられると、ちょっとしたファイルのやりとりをするのに便利です。筆者(中村)はよく、社内ネットワーク内でmacOSからWindowsにファイルを転送するときなどに利用します。

別解

即席のWebサーバをたてる方法はいくつもあり、シェル芸人たちからも多くの別解が寄せられました。**図3-1**はnc(Netcat)コマンドを利用した別解です。このワンライナーではFILEというファイルを決め打ちで転送します。HTTPのしくみを勉強する際によく試される方法です。

図3-2はRubyのunライブラリ[注2]とWEBrickを使った別解です。Pythonと同様に起動したディレクトリにあるファイルを公開できます。

細かい説明は割愛しますが、opensslのs_serverサブコマンドを使ってhttpsサーバを起動する方法も紹介します(**図3-3**)。-WWWオプションでカレントディレクトリにあるファイルを公開できます。コマンドの実行には、SSL/TLS通信のための秘密鍵と証明書が事前に必要になります。

注2) ebanさんはunライブラリの作者です。

▼図3-1 ncコマンドを利用した別解(上田)

```
$ while : ; do sed '1iHTTP/1.1 200 OK\nContent-Type: text/plain\n' FILE | nc -l 8080 ; done
```

▼図3-2 RubyのunライブラリとWEBrickを利用した別解(eban)

```
$ ruby -run -e httpd . -p 8000
```

▼図3-3 httpsサーバを起動(田代)

```
↓サーバを起動
$ openssl s_server -WWW -cert server.crt -key server.key
↓別端末：コマンドを起動したフォルダからの相対パスをURLに入力
$ curl -k -s https://localhost:4433/unko
```

問題4

複数IPアドレスが登録されているドメイン

出題、解答、解説：山田

中級 ★★

リスト4-1のようなドメイン名のリストである「domains.txt」というファイルがあります。このリストの中から、複数のIPアドレスが登録されているドメイン名のみを抽出してください。

▼リスト4-1　domains.txt

```
gihyo.jp
github.com
gitlab.com
wikipedia.org
```

解答

digというDNSサーバに問い合わせができるコマンドがあります。これにドメイン名を引数として渡すことで、そのドメインに登録されているIPアドレスを表示できます(**図4-1**)。

図4-1の"ANSWER SECTION"のすぐ下の行にIPアドレスを含んだ行がありますね。これはDNSのAレコード[注3]を表しており、2行続いています。これは1つのドメイン名に対して2つのIPアドレスが登録されていることを意味しています[注4]。

ここからgrepを使ってAレコードの個所を抽出しても良いのですが、digコマンドに+noallと+answerオプションを付けると、"ANSWER SECTION"の内容のみを出力できます。xargsを使ってdomains.txtにあるドメイン名ひとつひとつに対してこのコマンドを実行することで、**図4-2**の結果が得られます。

あとは、複数のIPアドレスを持った行のみを抽出すれば良いですね。まずは、awkでドメイン名の含まれる1番目のフィールドのみを取り出します(**図4-3**)。uniqコマンドは重複した行を取り除くのによく使われますが、-dオプションを与えることで逆に「内容が重複している行」のみを抽出できます。sedで末尾のドットを取り除けば、複数のIPアドレスが紐付(ひもづ)いているドメイン名一覧の完成となります(**図4-4**)。

注3) このAレコードがDNSサーバに登録されていると、そのAレコードに紐付いているドメイン名からIPアドレスを取得できます。

注4) 負荷分散のために特定のドメイン名に複数のIPアドレスが登録されることがよくあります。気になる方は「DNSラウンドロビン」で検索してみましょう。

▼図4-1　digコマンドの実行結果

```
$ dig gihyo.jp
; <<>> DiG 9.8.3-P1 <<>> gihyo.jp

;; QUESTION SECTION:
;gihyo.jp.                IN   A

;; ANSWER SECTION:
gihyo.jp.           300   IN   A   104.20.33.31
gihyo.jp.           300   IN   A   104.20.34.31

(..略..)
```

▼図4-2　各ドメインの"ANSWER SECTION"のみを出力

```
$ cat domains.txt | xargs -n1 dig +noall +answer
gihyo.jp.           300   IN    A   104.20.33.31
gihyo.jp.           300   IN    A   104.20.34.31
github.com.         11    IN    A   192.30.255.113
github.com.         11    IN    A   192.30.255.112
gitlab.com.         300   IN    A   52.167.219.168
wikipedia.org.      555   IN    A   198.35.26.96
```

▼図4-3　ドメイン名のみ抽出

```
$ 図4-2のコマンド | awk '{print $1}'
gihyo.jp.
gihyo.jp.
github.com.
github.com.
gitlab.com.
wikipedia.org.
 ↑登録されているIPアドレスの個数分だけ同じ行が出現する
```

▼図4-4　解答(内容が重複している行のみを抽出)

```
$ cat domains.txt | ↵
xargs -n1 dig +noall +answer | ↵
awk '{print $1}' | uniq -d | sed 's/.$//'
gihyo.jp
github.com
```

問題5

パケットを使ったOSの推定

出題、解答、解説：山田

上級 ★★★

【小問1】

あなたの好きなドメイン(例ではgihyo.jpにします)にpingを送ってみてください。そのドメインとして登録されたホストからレスポンスがあれば、結果が表示されます。その結果には**図5-1**のようにttlという項目が含まれていると思います。このTTLの値を1つだけ出力して、動作を終了するワンライナーを考えてください。たとえばTTLが55であれば、**図5-2**のようになります。

▼図5-1　pingの実行結果

```
$ ping gihyo.jp
PING gihyo.jp (104.20.33.31) 56(84) bytes of data.
64 bytes from 104.20.33.31: icmp_seq=1 ttl=55 time=0.381 ms
64 bytes from 104.20.33.31: icmp_seq=2 ttl=55 time=0.394 ms
(..略..)
```

▼図5-2　解答の実行結果のイメージ

```
$ 解答のワンライナー…
55
```

【小問2】

小問1でpingのパケットが通る通信経路には、宛先のホストだけでなくルータやレイヤ3以上のスイッチ(以下、便宜上ノードと呼びます)が存在します。パケットを送って返ってくるまでに、いくつのノードを経由するのか数えて数字を出力してください(送信元ホスト自身は数から除いてください。また、宛先のホストは数に入れてください)。

解答(小問1)

pingは正しいレスポンスがあるとttl=[数字]という文字列を含んだ結果を出力します(**図5-1**)。ただし、必要な結果は1つだけなので、grepの-mオプションを使います。-mに数字を指定すると、その回数分パターンを含む行を見つけたら、動作が終了するようになります(**図5-3**)。

あとはttl=[数字]というパターンの[数字]の個所のみ取り出せば良いですね。sedやawkを使うのも良いですが、grepだけでも数字は取り出せます。**図5-4**の解答では-oという「マッチした部分のみを出力」するオプションを使いつつ、-Pでperlの正規表現を有効にしています。ttl=\K\d+というパターンが指定されていると思います。perlの正規表現では\Kという表現を与えることで、それより左側のパターンはすべてアンカー[注5]として扱うことができます。これにより-oオプションの出力の対象になりません。\dはperlの正規表現で数字を表すため、\d+で数字の1回以上の繰り返しとなります。\Kより右側のパターンにマッチした個所のみが出力され、解答となります。

▼図5-3　"ttl="というパターンが含まれる行を1行だけ抽出する

```
$ ping gihyo.jp | grep -m 1 'ttl='
64 bytes from 104.20.33.31: icmp_seq=1 ttl=48 time=8.76 ms
```

▼図5-4　解答(小問1)

```
↓grep の-Pと"K"という正規表現を使って数字のみ抽出
$ ping gihyo.jp | grep -m 1 -oP 'ttl=\K\d+'
48
```

解答(小問2)

ホスト間にあるノードの情報を調べるにはtracerouteコマンドがよく使われます。-Iオプションを付けると、pingと同じICMPパケットを送ってくれます(**図5-5**)。

注5) 文字列にはマッチせず、位置にマッチする正規表現です。たとえば文頭(^)や文末($)などの表現もアンカーの一種です。

ここからノードの数を数えます。**図5-5**の結果を見ると、数字で始まる行にIPアドレスが記載されており、1行が1つのノードに対応しているようです。1フィールド目の数字が経由したIPアドレスの個数だけ数が増えています。つまり、最終行の数字を抽出すれば解答になりそうです。awkを使った解答が**図5-6**です。traceroutの結果にはpingの送信元自体のホストのIPアドレスも1番目に含まれています。問題文にあるように、これは数えたくないので数字を1引いたものを出力しています。

出題の意図

実は小問1と2の結果を使って宛先のホストのOSが推定できます。パケットを返すときのTTLの値は、OSごとに初期設定値が存在します。初期設定値としては**表5-1**にあるようなものが知られています[注6]。

pingで得られるレスポンスのTTLの値は、パケットが経由したノードの数の分だけ減ったものです。そのためtracerouteでネットワーク経路上のノードを数えて、その分の数字を足せば、ホストが返すTTLの初期値が得られます。ネットワークの状態によりtracerouteの結果は多少変化するので、おおよそにはなりますが、次の式が成り立ちます。

TTLの初期値＝小問1の結果＋小問2の結果

たとえば、gihyo.jpの場合、**図5-4**のpingの結果と**図5-6**のtracerouteの結果を上記の式に当てはめると48 + 14 = 62となります。結果は64付近なので「Linuxを使っているのではないか？」という推定ができます。

実際にはtracerouteを使わずともpingを特定のホストに投げてTTLが64付近なのか128付近なのかを見るだけでも、あるホストがWindowsなのかそうでないかの判定くらいには十分使えます[注7]。また、より正確にOSを特定したいのであればnmapというコマンドもお勧めです[注8]。

終わりに

2回に渡ってネットワークの問題を出題しましたが、いかがでしたでしょうか？　今回はサービスの死活監視のスクリプトを作ったり、ファイル共有のためのサーバをたてたり、パケットの情報からほかのマシンのOSを推定したり、「何か役立ちそう！」と思わせるような問題が多かったのではないでしょうか。

さて、次回はネットワークからはいったん離れたテーマになる予定です。ぜひ次回もお楽しみに！ SD

注6）あくまでおおよその分類です。たとえばLinuxでもディストリビューションやカーネルのバージョンによっては異なる場合があります。また管理者であればTTLの初期設定値は変更できるので過信は禁物です。

注7）このような一見些細な情報も、トラブルシューティングのための手がかりになることがあります。

注8）https://nmap.org

▼表5-1　OSごとのTTLの初期設定値

OSの種類	TTL
比較的昔のWindows（95、98）など	32
macOS、Linuxなど	64
比較的最近のWindows（7、10）など	128
Solaris、AIX、Ciscoのルータなど	255

▼図5-5　tracerouteの実行結果

```
$ sudo traceroute -I gihyo.jp
traceroute to gihyo.jp (104.20.34.31), 30 hops max, 60 byte packets
 1  192.168.3.1 (192.168.3.1)  2.362 ms  1.803 ms  2.448 ms
 2  * * *
 3  * * *
(..略..)
15  104.20.34.31 (104.20.34.31)  8.745 ms  8.740 ms  8.748 ms
```

▼図5-6　解答（小問2）

```
↓結果をきれいに表示するために標準エラー出力は捨てる
$ sudo traceroute -I gihyo.jp 2>/dev/null | awk 'END{print $1-1}'
14  ←パケットは14個のノードを経由する
```

Kotlin Webアプリケーション

長澤 太郎 著
B5変形判／284ページ
3,000円＋税
リックテレコム
ISBN＝978-4-86594-066-4

Kotlinというと、最近ではAndroidアプリ開発の言語のイメージが強いが、本書にはまえがき部分以外でAndroidの話題は出てこない。JVM言語としてのKotlinの、Javaの資産を使えるという特性を活かし、サーバサイドでの開発を取り上げている。第Ⅰ部ではKotlinの文法について、最新バージョンのKotlin 1.1の新機能にも触れながら解説していき、第Ⅱ部ではSpark Frameworkを、第Ⅲ部ではSpring Bootを使ったWebアプリ開発をレクチャーする。データベースとの接続や、Herokuでのアプリ公開の解説もあり、実用的だ。加えて、Kotlinは簡潔で書きやすいと評判の言語なので、初心者でもコードを追いやすい。モバイル開発からサーバサイド開発までこなせるようになったKotlinの可能性を感じられる1冊だ。

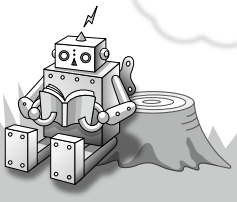
Haskell入門

本間 雅洋、類地 孝介、逢坂 時響 著
B5変形判／432ページ
3,280円＋税
技術評論社
ISBN＝978-4-7741-9237-6

関数型プログラミングは古くからあるパラダイムだが、その特性がもたらす設計上の利点などから、近年さらに評価が高まっている。関数型プログミング言語のうち、最も注目されているのはHaskellだろう。強力な型、I/Oの扱い、遅延評価など、少々癖のある特徴で知られる。難解な言語を想像するかもしれないが、実際には実用的な言語として一線で活躍している。

本書はそんなHaskellの、実践的な言語としての側面に注目した入門書だ。文法基礎から入り、モナドのような概念にも実際的な側面から触れ、最終的にはWebアプリケーションやコマンドラインツールなどを作成する。手を動かしながらHaskellを一歩ずつ学びたい人、実践的に学びたい人にはお勧めだ。

SD BOOK REVIEW



ソーシャルアプリプラットフォーム構築技法

田中 洋一郎 著
A5判／360ページ
2,800円＋税
技術評論社
ISBN＝978-4-7741-9332-8

若者はLINEを使い、年配者はFacebookを使う。SNSは日常の中に染み渡って電話以上に使われているツールになった。SNSを舞台に、このしくみを利用してサービス／アプリケーションを開発するにはどうしたらいいのか、多面的に分析しビジネスに応用する手がかりを示す、それが本書の要諦だ。SNS上のソーシャルグラフ（交流関係）を単に利用するだけではビジネスは広がらない。もっと抜本的に物事を考えねば成功しないのだ。多くのITベンチャーの中で成功している（ユニコーン）企業はこのあたりの理解が深いようだ。本書では述べていないがメルカリの急成長事例も思い当たることが多い。ソーシャル＝アプリ、その器としてのプラットフォーム。これをいかに構築するか、開発者だけでなく、Webに関わるすべての方にお勧めしたい。

IntelliJ IDEAハンズオン

山本 裕介、今井 勝信 著
B5変形判／232ページ
2,680円＋税
技術評論社
ISBN＝978-4-7741-9383-0

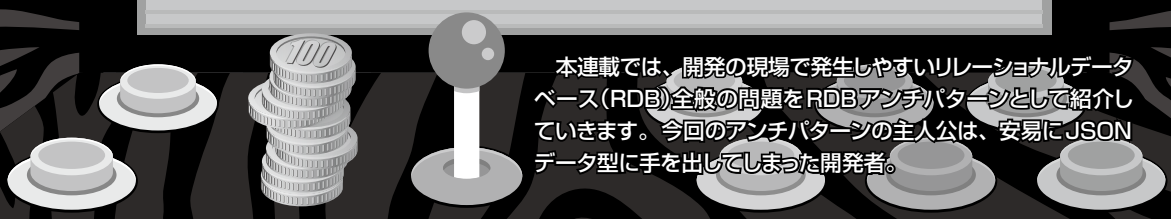
チェコのJetBrains社が開発した統合開発環境「IntelliJ IDEA」の一番の売りは、強力な補完機能だ。いわゆる「おまじない」と言われる長いコードも、IDEが適切に空気を読んで次に書くべき候補を提案してくれるので、少ない操作ですばやく完成させられる。開発において本来人間が行うべき作業――アイデアの実現やアルゴリズムの表現――に集中できるので、開発が楽しくなるIDEとして、最近注目度が高い。本書では、小さなコードを書きながらIntelliJ IDEAの操作感に慣れ、本格的な開発にも対応できる機能の使い方を習得できる。IntelliJ IDEAはまた、注目株の関数型プログラミング言語Scala、Androidアプリ開発の選択肢として新たに加わったKotlinとも相性抜群なので、本書を片手にぜひ新しい言語にも挑戦してほしい。

RDBアンチパターン

PostgreSQLとMySQLの失敗と対策

Author 曽根 壮大（そね たけとも） ㈱はてな Twitter @soudai1025

第8回 JSONの甘い罠



本連載では、開発の現場で発生しやすいリレーショナルデータベース（RDB）全般の問題をRDBアンチパターンとして紹介していきます。今回のアンチパターンの主人公は、安易にJSONデータ型に手を出してしまった開発者。

JSONの甘い罠

前回（本誌2017年11月号）は「隠された状態」と題して、DB設計で行いがちな問題について説明しました。今回は「JSONの甘い罠（わな）」という題名で、最近のMySQLやPostgreSQLなどのRDBMSが対応している「JSON」のアンチパターンについてお話します。

本来、リレーショナルデータモデルはスキーマレスな設計と相性が悪いことは、前回のEAVの話の中でも指摘しました。ですが、データをJSONとしてカラムに保存できれば、簡単にスキーマレスな設計を実現できます。

TEXT型にJSONを入れたり、PHPなどでSerializeした情報を入れたりして、このような設計を実現している例がありますが、これはまた別のアンチパターンであり、多くの問題を持っていました。そこでRDBMS側が出した答えが、JSONデータ型[注1]を用意するというものです。これにより、今までは実現できなかった設計や利便性を得ることができます。しかしJSONデータ型は、残念ながら完全ではありません。今回はJSONデータ型のメリットとデメリット、そしてJSONによって引き起こされる問題について説明します。

注1）URL https://www.postgresql.jp/document/9.6/html/functions-json.html

事の始まり

開発者Aさん：画面に項目が追加されるたびにDBの仕様変更になるのが辛い……。クライアントの要求はどんどん追加されるし、このままだと仕様変更のたびにDBが変更されて開発が間に合わないぞ。とは言えEAVな設計はあとから困るし……。ん？　PostgreSQLにはJSONデータ型ってのがあるのか！　これだとスキーマレスなDB設計を実現できる（**リスト1**）！

運用者Sさん：Aさん、同じ電話番号の人を抽出したいんだけどどうすればいい？

開発者Aさん：えーと……、ちょっと待ってくれる？　SQL調べてくるから。

運用者Sさん：え、また？　最近多くない？　昔はそんなことなかったのに。

開発者Aさん：このDBはJSONデータ型を使ってさ……、MySQLとPostgreSQLでも違うし、普段SQLで問い合わせしないからスッとSQLが書けないんだよね……。

運用者Sさん：それじゃあ「曽根」さんの住所を変更したいんだけど。

開発者Aさん：あー、それもすぐには難しいね。たしか本番はPostgreSQL 9.3だったよね？部分更新がないから。

運用者Sさん：それ、本当に設計として良かったの？

開発者Aさん：開発工数は下がった、と思うよ。

今回のアンチパターン

今回のアンチパターンは、すべての値をJSONとして保存したことです。確かにJSONデータ型を使えば、スキーマレスで変化に柔軟な設計ができます。しかしそれによって、検索の複雑性が上がったり、更新のコストが上がったりします。たとえば、運用者Sさんが求めたような「同じ電話番号の人を抽出する」クエリは、**図1**のようなSQLになります。このSQLはおもに3つの機能を使って目的を達成しています。

1つめはJOINです。JOINは第3回(本誌2017年7月号)でもお話したとおりRDBMSの一般的な機能で、今回CROSS JOINとINNER JOINを使っていますが、ここでは後述するLATERAL句との合わせ技です。MySQLの人には一見しても挙動がイメージしずらく、しかもパフォーマンスにも課題のあるクエリとなっています。

2つめはJSONをサポートする関数や演算子です。jsonb_array_elementsや->>などはPostgreSQL独自のため、多くのユーザは知らないでしょう。これらがネストを深くし、可読性の高いクエリからはほど遠くなっています。

▼図1　JSONデータ型に対しての検索

```
demo=# SELECT
  u1.id AS users1,
  u2.id AS users2,
  u1.properties ->> '名前' AS "p1 name",
  u2.properties ->> '名前' AS "p2 name",
  phone1 ->> 'type' AS "type",
  phone1 ->> 'number' AS "number"
FROM users AS u1
  INNER JOIN users AS u2
    ON (u1.id > u2.id)
  CROSS JOIN LATERAL jsonb_array_elements(u1.properties -> '電話番号') phone1
  INNER JOIN LATERAL jsonb_array_elements(u2.properties -> '電話番号') phone2
    ON (
        phone1->'type' = phone2->'type'
        AND phone1->'number' = phone2->'number'
      );
 users1 | users2 |   p1 name   |  p2 name  |   type   |   number
--------+--------+-------------+-----------+----------+-------------
      2 |      1 | Soudai Sone | 曽根 壮大 | 携帯番号 | 819012345678
(1 row)
```

▼リスト1　JSONデータ型を駆使した設計

```
CREATE TABLE users(
    id serial primary key,
    properties jsonb not null
);

INSERT INTO users(properties) VALUES ('
{
    "名前": "曽根 壮大",
    "住所": [
        {
            "都道府県": "広島県",
            "市町村": "福山市御幸町",
            "郵便番号": "720-0002"
        },
        {
            "都道府県": "東京都",
            "区市町村": "杉並区"
        }
    ], "電話番号": [
        {
            "type": "携帯番号",
            "number": "819012345678"
        }
    ]
}');

INSERT INTO users(properties) VALUES ('
{
  "名前": "Soudai Sone",
  "電話番号": [
        {
            "type": "携帯番号",
            "number": 81912345678
        }
  ]
}');

INSERT INTO users(properties) VALUES ('
{ "名前": "そね　たけとも" }');
```

3つめはLATERAL句[注2]です。この句はSQL標準で規定されており、PostgreSQL以外のRDBMSでもLATERAL句、もしくはAPPLY句としてサポートされていますが、MySQLにはありません。LATERAL句は単体の関数やサブクエリを修飾する形で使われます。付与されたサブクエリなどはほかのFROM句の評価の後に評価されます。そのため、ほかのサブクエリの結果をサブクエリの中から参照できます。今回はLATERAL句で、`INNER JOIN users AS u2 ON (u1.id > u2.id)`の後に評価された関数の結果をJOINしています。

改めて、このクエリを見てみなさんどのように思いましたか？　LATERAL句やJSONのサポートがなく、TEXT型に生のJSONを入れた場合は、この比ではない難易度になります。ただそれでも、十分に難しいSQLと言えるのではないでしょうか。パフォーマンスで言えば、正しく正規化こそしていればINDEXを利用できる場合もあります。

「なんでもJSON」の危険性

JSONデータ型は多くの可能性を秘めていますが、このほかにも次のような問題をはらんでおり、むやみやたらに使うべきではありません。

ORMが使えない

多くのORM(Object-relational mapping)はJSONデータ型をサポートしていません。またそもそも、JSONを取り出すためのSQLを表現することがたいへん難しいです。そのためJSONデータ型に頼ったDB設計をしてしまうと、一時的に開発工数を下げることができたとしても、最終的には開発工数が激増することが多々あります。ですからJSONデータ型を取り出す方法、保存する方法との相性についても検討することが必要です。

データの整合性が保てない

JSONデータ型はEAVの代替案でありながら、次のようなEAVと同じような問題があります。

・必須属性の指定が難しい

たとえば名前を必須属性にしたい場合、PostgreSQLの場合はCHECK制約を利用してJSONのKeyを指定できます。ただ指定できるのは既知の情報だけですので、たとえば電話番号も必須属性にしたい場合はALTERが必要になります。これならば、DBにカラムを増やすほうがシンプルだと思いませんか？

・データの中身を指定できない

日付としてyyyy/mm/ddとyyyy-mm-ddのように、フォーマットが違うデータが生まれる可能性があります。さらにPostgreSQLのJSONデータ型のうち、「jsonb型」[注3]の->を利用している場合、演算子の返却型はJSONオブジェクトですので、"12345678"(文字列)と12345678(数値)は別物として扱われ、型違いでerrorになります。しかもJSONは値の型を指定することはできないため、こういった型の違いを制限することはできません。ですから実は、**図1**には"819012345678"と819012345678を比較しないというバグが潜んでいます。そのためJSONの値を文字列として扱う->>という演算子を使って、**図1**の白枠部分を、次のように文字列に統一する記述に修正する必要があります。

```
ON (
    phone1->'type' = phone2->'type'
    AND phone1->>'number' = phone2->>'number'
  );
```

このように、データ型によって本来は守られていたことが失われてしまいます。

・参照整合性制約を強制できない

JSONの中身はユニークではないですし、正規化されていないため、外部キー制約を使うこ

注2）URL https://www.postgresql.jp/document/9.6/html/queries-table-expressions.html#QUERIES-LATERAL
URL https://lets.postgresql.jp/documents/technical/lateral/1

注3）JSON形式でバイナリデータを格納するデータ型。

とができません。たとえば"都道府県"というKeyに対して、東京都と東京の両方が保存される可能性があります。さらに、値と同じようにKey名も表記揺れの影響を受けます。

◆ ◆ ◆

これらのように、EAVと同じく汎用性を高められる反面、RDBが持っている多くのメリットを失い、RDBの本来の責務であるデータを守ることが難しくなります。JSONデータ型は銀の弾丸ではないのです。

JSONデータ型のメリット

ここまでJSONデータ型のデメリットについてお話しましたが、ではどのような場合に使うのが良いのでしょうか。そこでまず、JSONデータ型のメリットを挙げたいと思います。

(1)JSONそのものに対応している
(2)スキーマレスに値を保存できる

(1)により、カラムの値が本来JSONであるべきもののとき、パースすることなく対応できます。たとえば次のような値は、JSONで扱うことが多いのではないでしょうか。

・Web APIの戻り値
・PHPのComposer(設定ファイルがJSON)

これらのような値はもちろん、正規化してそれぞれの値として保存することもありますが、そのまま扱うことで便利になるケースが多々あります。この場合は、カラムの属性自体がJSONである必要があるので、何も問題ありません。

(2)は、RDBではできない設計のため、実務でJSONデータ型やEAVが必要とされる背景となっています。ですが、この条件が本当に必要なものであるのなら、そもそもRDBMS以外の選択肢も検討する必要があります。確かにJSONデータ型でスキーマレスな設計は可能ですが、パフォーマンスやスケーリングに課題が残ります。そこを無視できるデータサイズの場合は問題ありませんが、大きなシステムの場合は慎重に判断しましょう。

JSONデータ型を使うユースケース

筆者が実際に使った設計を紹介します。

Web APIと戻り値

TwitterのWeb APIと連携するため、**図2**のように、メインで使うテーブル(twitter_account)とサブのテーブル(twitter_account_detail)を分けつつ、履歴も残す設計を行いました。このようにしたのには次の2つの意味があります。

・Twitter側は予告なくAPIを変更するので、戻り値がErrorにならずに保存されているときも対応できるように保存
・必要なデータは正規化してその他が含まれるJSONは別テーブルに分けることで、パフォーマンスを向上させつつ、必要なデータが取得できなくなったときにはErrorで気づける

▼図2 TwitterのWeb APIと連携

twitter_account

id	screen_name	token	account
1	そーだい	hogefuga	soudai1025
2	たけとも	Hogehoge	taketomo1025
:	:		

twitter_account_detail　　JSON型

twitter_account_id	settings	created_at
1	{protected:false,screen_name:そーだい,…}	2017-10-16 1:51:00
1	{protected:false,screen_name:曽根,…}	2017-10-17 13:41:00
1	{protected:false,screen_name:そーだい,…}	2017-10-17 15:11:00
2	{protected:false,screen_name:たけとも,…}	2017-10-17 1:51:00
:	:	:

APIの残りの部分を捨てずに本来のJSONデータ型で保存することで、Twitter側の仕様変更にも柔軟に対応できます(もちろん、これとは別にリリースノートなどをチェックすることも大切です)。

OS情報

OS情報の値自体はJSONではないのですが、スキーマレスに使うために**図3**のような設計で実施しました。OS情報にはディストリビューションによって固有のものがあり、detailカラムをJSONにすることでディストリビューションごとの違いを吸収する設計にしています。リレーショナルデータモデルで設計するのであれば、「OSテーブル」に紐づく「ディストリビューションテーブル」をそれぞれに作ることが正しい設計でしょう。しかしこの場合は、次のような案件だったためJSONデータ型を採用しました。

- レコード数が少なく、数千程度だった
- 一度保存したレコードに対してUPDATEすることはほとんどない
- 取り出すときはJSONをまるごと取得

ただ**図3**の場合、次のような操作はアプリの実装が複雑になるので避けるのが良いでしょう。

- JSONのKeyに対して検索したい
- JSONの特定の値を更新したい
- JSONの中の値に制約を設けたい

これら操作が必要な場合、正規化を行って個別にカラムを指定しましょう。

ユーザが任意で登録するプラグインの情報

自作でCMS(コンテンツ管理システム)を作成した際、プラグインの機構を作る必要がありました。プラグインを登録する際は自由にテーブルを作れる構造にしたのですが、それによって「プラグイン名_setting」のようなテーブルが乱立することになりました。これを防ぐため、次のような値を保存する場所として、JSONデータ型のカラムを用意しました。

▼図3　OSの情報の保存

hosts　　JSON型

host_id	host_name	ip	OS	detail
1	Host1	192.168.0.1	CentOS 6	{…}
2	Host2	192.168.0.2	CentOS 6	{…}
3	Host3	192.168.0.3	Ubuntu 14.04	{…}
4	Host4	192.168.0.4	CentOS 7	{…}
:	:	:		

- プラグインで必要な設定値を保存するカラム
- 「プラグインが依存するプラグイン」がある場合の、プラグイン名とバージョンを記載するカラム

これにより、不必要なテーブルは減り、依存関係を明示することで類似のテーブルを親プラグインにまとめることもできました。とくに設定値を保存するカラムには作成者が任意の値を設定したいため、スキーマレスである必要がありました。JSONデータ型はEAVと違い、1プラグイン・1レコードであれば良いため、複数の設定値が登録されてもパフォーマンス的に有利というメリットもありました。またそのほかにも、次のような理由がありました。

- データが十分に小さいシステムである
- このシステム自体が開発途上でまだまだ変更が入る予定がある

とくに後者の理由から、積極的に正規化するコストよりも、まずプロトタイプとしてリリースしながら修正できるようにするためにJSONデータ型を採用しました。また、使用したWebアプリケーションフレームワークがLaravelで、それに付属するORMの「Eloquent」がJSONデータ型に対応しているのも、大きな採用の理由です。この設計は筆者の中でも非常にうまくいった好例の1つです。

このアンチパターンのポイント

このようにJSONデータ型は、EAVの代替案

として機能が豊富で非常に優秀である反面、EAVと同じような問題を解決できていません。では、EAVからJSONデータ型に置き換えるときは、どのような点に気を付ければいいのでしょうか。次の点が重要と言えるでしょう、

- 正規化してリレーショナルデータモデルで表現することができないか
- JSONに対して頻繁に更新を行いたいか
- 検索条件としてJSON内の属性が固定できない場合

1つでも該当がある場合はJSONデータ型を採用すべきではありませんし、多くのケースのがこれらに該当するので、JSONデータ型を採用すべきではありません。JSONデータ型はRDBMSの機能と引き換えに柔軟性を与える、本当に最後の切り札なのです。

次回のRDBアンチパターン

今回のRDBアンチパターンはいかがでしたでしょうか？　JSONはEAVに対する新しい解でもある反面、まだまだ扱いが難しいのが現状です。使い方を間違えると痛い目を見るアンチパターンですので、これから採用を検討される場合はよく考えて利用しましょう。

ここまでの連載では制約の大切さを説いて来ましたが、それが裏目に出た場合のアンチパターンもあります。次回の「強過ぎる制約」もお楽しみに！ SD

RDB TIPS JSONデータ型のほかの使い道

JSONデータ型には、保存する以外の使い方があります。たとえば**図4**の、「①レコード→JSON」と「②JSON→レコード」のパターンです。ここではデータを活用する段階で、JSONデータ型を利用しています。

①のメリットはアプリ側でループを回すことなく、1レコードとしてテーブルの情報を取り出せることです。たとえば例のように、中国地方の店舗を県別で表示したい場合、各県ごとになったレコードを順に利用すれば良いのでシンプルになります。②のメリットはアプリケーション側から渡されたJSONを、ストアドプロシジャなどを用意しなくても適切に扱えるようになることです。これにより、アプリ側でparseしなかった情報でも適切に分割できます。

とくに①の使い方は、適切に活用すれば実務でもパフォーマンスを向上させられる場面が多くありますし、アプリ側のロジックを簡略化させる効果もあります。機会があればぜひ、利用を検討してみてください。JSONデータ型は薬にも毒にもなることを体験できるでしょう。

▼図4　レコード→JSONとJSON→レコード

shops

pref	shop	…
広島	ほげ	…
広島	ふが	…
岡山	ふー	…
山口	bar	…
:	:	:

①アプリに渡すときにRDBのレコードの形をJSONに

アプリケーションへの戻り値

```
{
"広島": [
    {"shop":"ほげ", "...":"..."},
    {"shop":"ふが", "...":"..."}
  ],
"岡山": [
    {"shop":"ふー", "...":"..."}
  ],
"山口": [
    {"shop":"bar", "...":"..."}
  ]
}
```

shops

pref	shop	…
広島	ほげ	…
広島	ふが	…
岡山	ふー	…
山口	bar	…
:	:	:

②アプリから渡ってきたJSONを、JSONデータ型を利用してSQLでparseしてレコードの形に

presented by
Japan Android Group
http://www.
android-group.jp/

コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

第21回 GoogleのARプラットフォーム ARCore

Androidは世界で出荷される約9割のスマートフォンに搭載される標準OSです※。そのため、多くのAndroidアプリが開発され続けており、そして多くのエンジニアが活躍しています。Androidで広がる新しい技術に魅了されたエンジニアが集うコミュニティもあり、そこでは自分が愉しむための技術を見つけては発信しています。その技術の一幕をここで紹介します。

※Gartner Worldwide Smartphone Sales to End Users by Operating System in 3Q16

高橋 憲一
(たかはし けんいち)
㈱カブク
日本Androidの会
Twitter @ken1_taka

AndroidとAR

AndroidとARは、Androidがリリースされた当初から高い関連がありました。古くはセカイカメラやWikitudeなどの、位置情報＋カメラプレビューとグラフィクスの合成により実現するアプリケーションに始まり、ARToolKitなどのライブラリを使用し、認識したマーカーの上に3Dオブジェクトを表示するタイプ、Vuforiaなどのライブラリを用いた任意画像をマーカーとして認識するタイプなど、いくつかの形のものが進化してきました。Pokémon GOが位置情報に加えて、カメラプレビューの背景の上にポケモンを表示する画面を備えたARゲームとして昨年リリースされたことも記憶に新しいところです。

もう1つ、Googleが2014年から進めてきたTangoもあります[注1]。Tangoは通常の色情報を取得するカメラに加えて、ワイドアングル(魚眼)カメラ、深度取得用のセンサーを端末に搭載することにより周囲の環境を認識して、精度の高いARを実現するものです。深度センサーにより、現実の物体に画面上のオブジェクトが隠れるという表現も可能です。

そして、本記事で取り上げるARCoreは2017年の8月末日にGoogleから発表されました。

ARCoreとは

ARCoreは、GoogleがAndroid端末でAR(Augmented Reality：拡張現実)アプリを実現するためのプラットフォームとして開発したものです。

本記事の執筆時点(2017年10月中旬)では、early preview(早期プレビュー)というステータスになっています。そのため、動作対象の端末の種類もGoogleのPixelおよびPixel XL、SamsungのGalaxy S8と限られています[注2]。10月4日に発表された新しいPixel 2とPixel 2 XLはARに最適化されており、低光量の環境でも安定したトラッキングや、1秒あたり60フレームの性能でARオブジェクトの描画ができるようになっているという話もありました(こういうときに頼りになるはずのGoogle謹製端末であるPixelとPixel 2が日本では発売されていないという状況ですが、日本国内で購入したGalaxy S8でも動作したという報告があるのは救いです)。

Tangoと異なり特別なセンサーを必要としないため、より多くの種類の端末で実行できるこ

注1) ARCoreの発表にともない、Tango SDKは現行でリリースされている端末であるAsusのZenFone ARとLenovoのPhab 2 Proのみをサポートし、GoogleのARの取り組みは特別なハードウェアを必要としない新しいARCoreとして継続していくということになりました。

注2) Galaxy S8+とNote8も近く対応するという発表が10月18日にありました。

とが見込まれます。正式リリースまでには数百万台の端末で利用できるようになるとのことです。

ARCoreのしくみ

ARCoreは周囲の実環境を認識して、端末の位置をトラッキングします。端末のカメラが捉えた映像から特徴点を抽出してその動きを追いかけ、端末に搭載されている加速度とジャイロスコープのセンサーが検出した動きを組み合わせることで実現しています。このしくみはConcurrent Odometry and Mapping(COM)と呼ばれます。Odometryは走行距離計測のことで、カメラからの画像を処理するコンピュータービジョンの技術と、Inertial Measurement Unit(IMU：慣性計測装置)すなわちジャイロや加速度のモーションセンサーを組み合わせることで、自分が空間の中で時間の経過とともにどの方向にどれだけ動いたかを認識することができます。

以上のことから、カメラからの画像をコンピュータビジョンで処理すること、モーションセンサーとカメラからのデータを精密に同期して取得すること、3Dグラフィクスの描画を行うことなどをふまえて考えると、高い精度でのトラッキングには、ある程度以上の高い性能の端末が求められることになるはずです。

ARCoreでできること

モーショントラッキング

先ほど述べたCOMのしくみにより、カメラのpose(位置と向き)を推定します。その位置と向きに沿うようにバーチャルなカメラを設定して3Dオブジェクトを描画し、カメラプレビューからの映像と合成することで、その3Dオブジェクトがあたかもそこに置かれているように見せることができます。

環境認識

特徴点と平面を検出することで現実の周囲の環境を認識し続けます。机やテーブルのような水平の面とその境界を認識して、アプリはその情報をplane(平面)として取得できます。その平面の上に3Dオブジェクトを設置することができます。特徴点を使用するため、たとえば真っ白な机など、表面に模様がない面の場合は正しく認識できないことがあります。

光源推定

カメラ画像から現在見えている環境の照明の状態を検出し、光量の推定値を得ることができます。これにより、表示する3Dオブジェクトの照明を周囲の環境に合わせることで現実感を高めることが可能です。

ユーザインタラクション

画面内に見えている空間への当たり判定を行うことができます。たとえば、端末の画面をタップした位置から線を伸ばしていき、認識済みの平面、もしくは特徴点に当たった情報を取得できます。

図1はGalaxy S8でサンプルアプリを実行したものです。床面を水平面として認識し、その上に3Dオブジェクトを配置して、カメラプレビュー画像と合成表示することができています。

▶図1 サンプルアプリの実行例

ARCoreを使う開発で必要なもの

PCに通常のAndroidアプリの開発環境が用意されていること、およびAndroid端末にはARCoreサービスのapkをインストールする必要があります。開発情報サイト[注3]にあるARCore Serviceのダウンロード元からapkファイル(現時点ではarcore-preview.apkという名前になっています)を入手して、端末をPCに接続してadb installコマンドでインストールします。

SDKはAndroid Studio、Unity、Unreal Engineの3種類の開発環境用のものが用意されているので、それぞれ必要なものを開発情報サイトのリンクから入手します。ここではAndroid StudioとUnityの場合について見ていきます。

Android Studio

標準のAndroidアプリ開発の場合と同様に言語としてJavaを使用します。この場合、3Dグラフィクスの表示にはOpenGL ESを使うことになります。

ARCoreを使うアプリはエミュレータでは動かすことができないのですが、幸運にもARCoreの動作対象端末をお持ちの方は、まずはSDKに含まれるjava_arcore_hello_arという名前のサンプルをAndroid Studioで開いて試してみましょう。

build.gradleのdependenciesを見ると、**リスト1**の★印の2行のようにarcore_clientとobj-0.2.1の2つのライブラリが追加されているのがわかります。arcore_clientはもちろんARCoreを使用するためのものですが、obj-0.2.1のほうはobj形式という3Dモデルを読み込んで表示できるようにするためのものです。プロジェクトにはAndroid robotの3Dモデル[注4]が含まれています。Androidアプリのビルドをしたことがある環境であれば、サンプルのビルドと端末での実行はすぐにできると思います。

Unity

Unityのバージョンは2017.2以降である必要があります。SDKはarcore-unity-sdk-preview.unitypackageというファイル名のパッケージになっており、サンプル実行用のシーンもこの中に含まれてます。

新規プロジェクトを作成し、SDKのパッケージをインポートするとProjectのAssetsの下にGoogleARCoreというフォルダが追加されます。さらにその下のHelloARExample/Sceneの下に、HelloARというシーンがあります。このシーンをビルド対象として追加して、端末で実行できます。ビルドするにあたって必要な設定として、Play SettingsのXR SettingsというARやVRに関する項目があります。**図2**のように、その中のARCore SupportedをONにします。

SDKの構成

Java、Unity、Unreal Engineのそれぞれの場合で細かいAPIの使用方法は異なりますが、機能としては同様のものがそろっています。たとえば、LightEstimateにはgetPixelIntensity()というメソッドがあり、光量の推定結果が0.0から1.0の範囲の値で返ってきます。PlaneHit

注3) https://developers.google.com/ar/develop/java/getting-started

注4) ファイル名はandy.obj ;)

▼リスト1　build.gradleのdependencies設定

```
dependencies {
    compile (name: 'arcore_client', ext: 'aar')   ★
    compile (name: 'obj-0.2.1', ext: 'jar')       ★
    compile 'com.android.support:appcompat-v7:25.0.0'
    compile 'com.android.support:design:25.0.0'
}
```

Resultは衝突した平面、PointHitResultは近接する特徴点を取得します。AR Experiments[注5]で公開されている例の中には水平だけでなく垂直な平面も認識しているものもあり、PlaneHitResultで平面の該当がなければ、PointHitResultで取得した点の場所に垂直な平面を構成するということを行っているようです。

WebでもAR

three.ar.jsとプロトタイプのAR対応ブラウザを使うことにより、WebでもARCoreを使うことができます。特筆すべきこととして、プロトタイプのブラウザはARCore on Android用だけでなく、ARKit on iOS用も提供されており、どちらの場合もサイトからソースコードをダウンロードしてビルドして端末に転送することで使えるようになります。現在はプロトタイプの特別なブラウザを必要としますが、将来ChromeなどのブラウザにARの機能が取り込まれれば、Web用に用意した1つのARコンテンツがAndroidでもiOSでも使えるということが期待できます。Web上で3Dグラフィクスを表示するにはthree.jsというライブラリが使われることが多いのですが、three.ar.jsはそのthree.js用のヘルパーライブラリで、JavaScriptからARを使うことを可能にするものです。

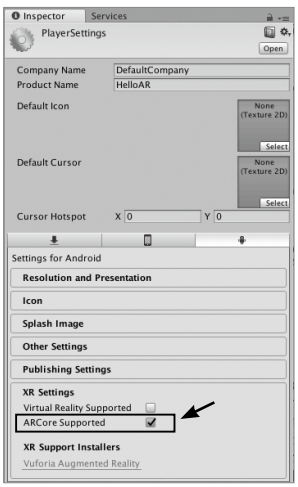
Web用の開発情報ページにはAR対応ブラウザのソースコードのダウンロード元や、いくつかサンプルが掲載されています[注6]。**図3**に見られるように、床面を認識してその上に3Dオブジェクトを配置できます。アプリで実行した際と同様に、端末を持って移動するとそれに合わせて最初に配置した場所に張り付いているように見ることができます。

この時点からWeb用の開発環境を用意し、クロスプラットフォームでの動作を念頭に置いているのはGoogleらしいアプローチと言えるでしょう。Android用は動作対象端末がまだ限られていますが、iOSはARKitが動作する端末なら試すことができます。表示する3Dモデルを別のものに変えるなど、いろいろ実験してみるとおもしろいと思います。SD

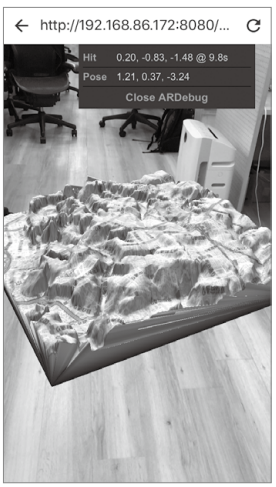
注5）https://experiments.withgoogle.com/ar

注6）https://developers.google.com/ar/develop/web/getting-started

▼図2 ARCore Supportedの設定



▼図3 Webでの実行画面



COLUMN

GDG Devfest Tokyo 2017

2017年10月9日にGDG Devfest Tokyo 2017が開催されました。Googleの技術に関するイベントではありますが、あくまで主体はコミュニティによるものです。実に12ものコミュニティにより運営され、Android、GCP、TensorFlowなど多岐にわたるセッションがありました(筆者もDaydreamのセッションを担当しました)。Android関連のものだけでも、日本Androidの会、Shibuya.apk、droidgirls、ARCore(Tango)Working Group、DroidKaigiと5つものコミュニティが関わっています。とくにARCore(Tango)Working Groupは新しいコミュニティで、本記事で解説したARCoreを使って何かやりたいという人が集まっています。

一歩進んだ使い方のためのイロハ

Vimの細道

第24回

Vimの新機能terminal（前編）

mattn
twitter:@mattn_jp

Vim 8に新しく搭載されたterminalをご存じでしょうか？　Vimの画面内で端末を起動できるので、Vimを終了することなく、別の作業を行えます。そんなterminalについて、前編では基本操作からWindowsでの環境構築方法、筆者の実用例まで紹介します。

今年の一番のニュース

2017年で一番のVimのニュースと言えば、誰が何と言おうと間違いなくterminal（Vimから使える端末エミュレータ）の実装だと思います。Vimは基本的に、TUI（Text User Interface）のプログラムです。GUI版のgvimであってもその構造は変わらず、端末上で起動する「文字をベースにしたテキストエディタ」です。シェルからvimコマンドで起動し、:qというコマンドで終了してシェルに戻ります。Vimを起動中に、端末上で何かほかの操作をしたい場合には、一度Vimを終了するか、:shを実行して新しいシェルを起動する必要がありました。

筆者はGitの履歴を確認するのによくtigというツールを使いますが、Vimで編集しているソースファイルに、先ほどcommitしたばかりの変更に含まれる内容を再度引用したい、といったケースがよく起こります。

今までであれば:shでシェルを起動し、git logを見たり、git showを実行してその結果をgrepしたりしていました。Gitでは、特定のコミットの差分を見るにはハッシュという40文字のIDを指定する必要がありますが、こんな長いIDは覚えられません。シェルコマンドを駆使しながら特定のIDを見つけ出し、そのIDの差分をVimの中に取り込むには、クリップボードを使うなど、そこそこパワーのいる作業が必要です。

もちろんそれ専用のプラグインも存在します。たとえばAlisue（@lambdalisue）さんが作っているgita.vim[注1]を使うと便利です。しかしこういったニーズは、Gitだけに限った話ではありません。そんな皆の思いが少しずつ増えていき、VimのforkであるNeoVimがterminalを実装しました。terminalはNeoVimの特徴的な機能となりました。Vimユーザの間でも、terminalがほしいという意見も出てきていましたが、最終的にはVimの作者であるBram Moolenaar氏自身が、「もしかするとterminalは便利かもしれない」と開発者メーリングリストでアナウンスし、Vim 8にterminalが実装され始めることになりました。

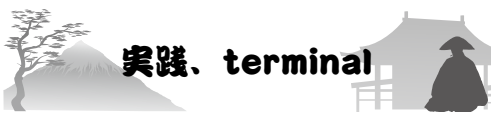
Vimにとっての:terminal

Vimには、「Vimにとって相応しくない設計」を記したドキュメント「design-not」があり、:h design-notで確認できます。その中では、「VimはVimの中でシェルやデバッガを起動することはできない」と書かれていました。これは今から20年以前に書かれた文書です。しかし20年の

注1）URL https://github.com/lambdalisue/vim-gita

歳月が過ぎてVimを取り巻く環境も変わり、Vimにとっての「正しさ」というものが変わってきたのかもしれません。この:terminalの実装に伴い、「design-not」も変更されることになったのです。現在の「design-not」には次のように書かれています（筆者訳）。

> *Vimはシェルやオペレーティングシステムではない。terminalウィンドウが提供され、その中でシェルやデバッガを起動できる。たとえばssh接続もできる。しかしテキストエディタを必要としないケースならば、これはスコープ外の物である（screenやtmuxを代わりに使うべきだ）。*
> *Vimは、Emacsのようにキッチンシンク以外のすべてを備えるようなことはしないが、Vimを使ってキッチンシンクをきれいにすることはできる、という人もいる;-)*



それでは、新しく追加されたterminalを使ってみましょう。

terminalの起動

terminalの起動は簡単です。次のコマンドを実行します（Vim 8.0以上が必要です）。

```
:terminal
```

起動すると画面が分割され、お使いのシェルが起動するはずです。このウィンドウではタイプしたキーがそのまま端末に送り込まれ、一般的な端末エミュレータと同じ操作を行うことができます。端末の中で、さらにVimを起動したり、Emacsを起動したり、tigコマンドを起動することもできます。

terminal上で起動するアプリケーションは、ほかのウィンドウにフォーカスがある状態でも起動し続けるので、たとえば、増え続けるログファイルを次のように監視するのも良いでしょう。

```
$ sudo tail -f /var/log/nginx/access.log
```

また色付きで表示されるので、htopコマンドのようなカラフルなTUIツールを起動するのも便利です。なお:terminalコマンドは引数を取ることができるので、直接コマンドを指定してterminalを起動することもできます。

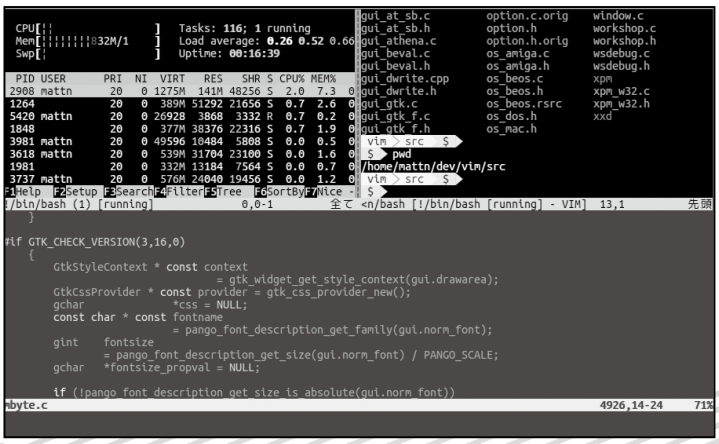
```
:terminal htop
```

複数のterminalを同時に開くことができるので、1つのウィンドウでhtop、もう1つのウィンドウでシェルを起動するといった使い方もできます（**図1**）。別のウィンドウでさらにnginxのログを表示しておくのも良いですね。

terminalの終了

terminal上で起動しているプロセスを Ctrl -w Ctrl -c で終了すると、そのバッファはTerminal-Normal-Modeというモードになります。

▼図1　ソースコードを編集しながら、terminalでhtopとシェルを同時に開く



Terminal-Normal-Modeは、端末上で実行した画面の出力を表示するモードです。このバッファ上で、ユーザは必要なテキストをyankして別のバッファにコピーしたりすることができます。

Terminal-Normal-Modeはスクロールバックしたテキスト(画面スクロールで流れたテキスト)も含まれるので、hjklを使って、過去の出力も含め任意の場所に移動することができます。

Terminal-Normal-Modeのバッファは通常のバッファと同様に、:qや:bwipeoutで閉じることができます。

キーアサイン

terminalはほかのバッファと異なり、ほぼすべてのキーがterminalに送り込まれます。Vimとしての動作を扱うために唯一、Ctrl -wが用意されています。Ctrl -wに続いて特定のキーをタイプすると、端末ウィンドウに対していろいろな操作を行うことができます(**表1**)。

terminalでプロセスが起動中であっても、前述のTerminal-Normal-Modeに入ることができます。Terminal-Normal-Modeに入っている間は、端末の更新は行われません。プロセスが起動している間のTerminal-Normal-Modeでは、バッファの変更が行われるような操作が行われると、Terminal-Normal-Modeを抜け、再び端末との対話モードに戻るようになっています。

余談ですが、Vimの中のterminalでさらにVimを起動し、そのVimの中で再度terminalを起動した場合、そのterminalへCtrl -w :を送信するには、Ctrl -w . :とタイプする必要があります。

UNIX版のVimのterminalは、libvtermというライブラリを使って実装されています。libvtermはXTermのbracket pasteという機能を実装しているため、この機能に対応しているアプリケーションであれば、そのアプリケーションと連携したペーストが行われます。gvimではフルカラーで表示され、また端末版Vimであってもその端末が256色に対応していて、かつtermguicolorsオプションが有効であれば256色で表示されます。

▼表1　terminalでのVim操作

キー	説明
Ctrl -w Ctrl -w	次のウィンドウへ移動
Ctrl -w :	Exコマンドの実行
Ctrl -w .	Ctrl -wをterminalに送信
Ctrl -w N	Terminal-Normal-Modeへ
Ctrl -\ Ctrl -n	Terminal-Normal-Modeへ
Ctrl -w " {reg}	レジスタ{reg}をterminalにペースト
Ctrl -w Ctrl -c	terminalで起動しているプロセスを終了

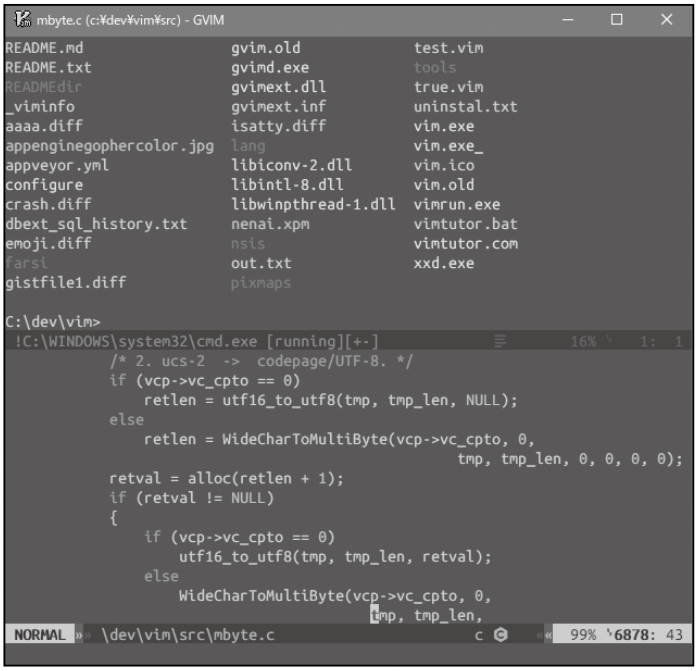
Windowsにおけるterminal

terminalはWindowsもサポートしています。gvim.exeだけでなくvim.exeでも動作します。Windowsでの:terminalはコマンドプロンプトが起動し、通常のコマンドプロンプトと遜色(そんしょく)ない表示や操作ができます。ただし、Windowsで:terminalを使用するためには少しだけ環境構築が必要です。winptyというライブラリを使用するので、該当GitHubのreleaseページ[注2]から必要なファイルをダウンロードする必要があります。このページからwinpty-0.4.3-msvc2015.zip(2017年10月時点)をダウンロードし、解凍してできた次の2つのファイル(x64/bin/winpty.dll・x64/bin/winpty-agent.exe)を、vim.exeもしくはgvim.exeのある場所にコピーしてください。32bit OSの場合は、同zipファイルの中の、ia32ディレクトリの実行モジュールをコピーしてください。

あとはVimを再起動して:terminalを実行すれば、Linuxと同じようにウィンドウが開き、コマンドプロンプトが起動します(**図2**)。gvim.exeで:terminalを起動して、その中でvim.exeを実行するといったこともできます。

注2) URL https://github.com/rprichard/winpty/releases

▼図2　Windows版Vimでのterminal実行



ドが行われ、エラーがあれば赤いメッセージが、またエラーが修正されれば緑のメッセージが表示されます。視覚的にとてもわかりやすく、Vimのterminalでgomonを起動したままにしておくと、ソースファイルを編集して:wで保存するだけで、コンパイルエラーに気付くことができます。

また、GitHubでブログを書く方はJekyll[注4]というツールを使っている方が多いと思いますが、terminalでjekyll serveコマンドを起動しながら、Vimで「_config.yml」やブログ記事を書くと、画面にエラーが表示されます。このほかにも、使い方はいろいろあると思います。

◆ ◆ ◆

今回はVimに新しく追加された機能:terminalを紹介しました。Vimのこれまでのユーザ体験を大きく変える機能だと思います。ぜひ活用してみてください。正直に言うと、筆者もまだ使い始めたばかりで、その可能性を探っているのが現状です。来月号の後編では、terminalを便利にするオプションや設定、terminalと連携するプラグインを紹介したいと思います。SD

筆者のterminal活用法

筆者はGo言語をよく使うので、Vimからterminalを起動し、そこでgomon[注3]というツールを動作させています。

gomonは、ファイルの変更を検知してgo buildやgo testを実行してくれる便利なコマンドです。Goのソースを編集して保存すると、勝手にビル

注3）URL https://github.com/c9s/gomon

注4）URL https://jekyllrb-ja.github.io

Vim豆知識

redrawtimeオプションの一部変更

patch 8.0.1133にて、redrawtimeオプションの扱いが変わりました。これまではmatch()やhlsearchなどで、遅いマッチをタイムアウトさせるためのオプションとして使われてきましたが、本バージョンから、通常のシンタックスハイライトでも適用されるようになりました。これまで小さな値を設定していた方は、突然シンタックスハイライトが消える動作に遭遇するかもしれません。設定を無効にしてデフォルト値(3000)を使うと、問題が解消するはずです。

書いて覚える Swift 入門

第32回 APFSの研究

Author 小飼弾（こがい だん）　twitter @dankogai

最低限文化的なマカーのためのAPFS入門

iOSでは10.3以降、問答無用でHFS+を置き換えたAPFSが、macOS High Sierraでいよいよ Macにもやって来ました。利用にあたってファイルシステムを意識する必要が事実上皆無で、外付けストレージをマウントすることもほとんどないiPhoneやiPadやApple WatchやApple TVと異なり、外付けストレージもNASもごく当然に利用し、Boot CampなどでOSすらmacOS 以外のものをサポートしているMacを利用する以上、ファイルシステムというものをまったく意識しないということは不可能です。今回はSwiftを意識しつつも、Swiftの開発において事実上必須である、macOSの新メインファイルシステムたるAPFSをあらためて見ていきます。

なぜ、

- APFSは事実上問答無用で現在サポートされているApple製品すべてで採用されたのか
- Fusion DriveやHDDを内蔵するMacはその例外となったのか
- 開発者たる我々はどのような点に気を付けたら良いのか

本記事を読めば、これらの点にも納得がいくはずです。

Copy-on-Write＝SwiftとAPFSの共通点

APFSの最大の特長は、それがCopy-on-Write(CoW)なファイルシステムであるということです。公式ページ[注1]にも真っ先にそれが示されています。

CoWとはいったい何なのでしょうか？　実はSwiftもCoWを採用しています。そのことを確認してみましょう。

```
extension String {
    func peek() {
        let type = (UInt, UInt, UInt).self
        let (u0, u1, u2) = unsafeBitCast(self, to:type)
        debugPrint(u0, u1, u2)
    }
}

let orig = "Hello, playground"
var copy = orig
orig.peek()
copy.peek()
copy += " string"
orig.peek()
copy.peek()
```

SwiftのStringは抽象化されていて、その構造を直接見ることは本来できないのですが、ここでは無理やりunsafeBitCastを使って覗いています。最初にpeek()したときにはorigもcopyも同一だったのが、copyに"string"を追記したあとは内容が変わっていることが確認できます。

SwiftのString構造体は常に24bit＝ポイン

注1）URL https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/APFS_Guide/Features/Features.html

タ3つ分。この値が固定であることからも、文字列の内容は構造体そのものではなく構造体の中のポインタが示すアドレスに格納されていることが察せられます。それが最初にcopy = origした時点では変わらず、copyに変更が加えられたあとに変わったというのはどういうことか。内容に変更がなされるときに初めてデータが複製され、その複製されたデータに対して変更が行われたのです。Copy-on-Writeすなわち「複製は書き込み時」というわけです。

別の見方をすると、CoWにおいてはデータの上書きというのは存在せず、必ずコピーされたデータを変更するということになります。たとえ1バイトでも変更されたら、元のデータとは違う領域に変更後のデータが保存されるということです。

これが何を意味するのか。元のデータへのポインタさえ残っていれば、変更前のデータが必ず取り出せるということです。APFSを含め、CoWなファイルシステムにおけるスナップショット(snapshot)とは、このもとのデータへのポインタのことなのです。先ほどのSwiftのコードのorigに相当します。

スナップショットがあると何がうれしいか。まず、Undoが超簡単になります。Swiftの例であれば、もう一度copy = origするだけです。それがファイルシステム全体に対して行えるのです。

実際にやってみましょう。macOSにはv10.5以来Time Machineというバックアップ用インターフェースが用意されています。賢明なる読者のみなさんはすでに使っていらっしゃるはずですが、High Sierraよりこれはスナップショットへのインターフェースともなっています。

まずは念のためにTime Machineを有効にしておきましょう(**図1**)。バックアップ先は用意しなくてもOKです。システム設定の自動バックアップにチェックを入れてもOKですし、Terminalからsudo tmutil enableを実行してもOKです。

これだけで、1時間おきに自動でスナップショットが作られるのですが、tmutil snapshotで手動でスナップショットを取ることもできます。完了は一瞬です。CoWではデータの複製は書き込み時なのですから、「ここがスナップショット」以上の情報は書き込まないのですから当然です。

▼図1　Time Machineを有効にしておく


```
$ tmutil localsnapshot
Created local snapshot with date: 2017-10-17-165216
```

スナップショットのリストは、tmutil listlocalsnapshotsのあとにマウントポイントを指定して見ることができます。

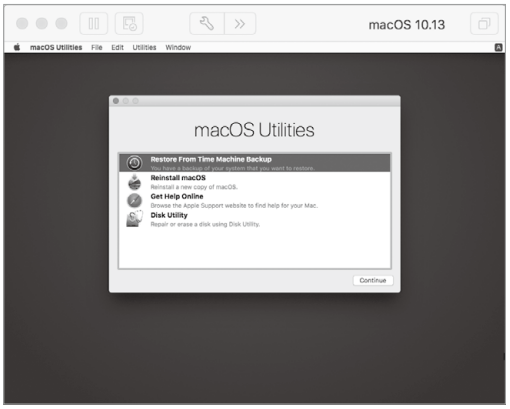
```
% tmutil listlocalsnapshots /
com.apple.TimeMachine.2017-10-18-150407
com.apple.TimeMachine.2017-10-18-160303
com.apple.TimeMachine.2017-10-18-170437
com.apple.TimeMachine.2017-10-18-180253
com.apple.TimeMachine.2017-10-18-190345
com.apple.TimeMachine.2017-10-18-200341
com.apple.TimeMachine.2017-10-18-210231
com.apple.TimeMachine.2017-10-18-220244
com.apple.TimeMachine.2017-10-18-230916
com.apple.TimeMachine.2017-10-19-000459
com.apple.TimeMachine.2017-10-19-010440
com.apple.TimeMachine.2017-10-19-021104
com.apple.TimeMachine.2017-10-19-030708
com.apple.TimeMachine.2017-10-19-040243
com.apple.TimeMachine.2017-10-19-050420
com.apple.TimeMachine.2017-10-19-070312
com.apple.TimeMachine.2017-10-19-080249
com.apple.TimeMachine.2017-10-19-090235
com.apple.TimeMachine.2017-10-19-111504
com.apple.TimeMachine.2017-10-19-122825
com.apple.TimeMachine.2017-10-19-133025
com.apple.TimeMachine.2017-10-19-142826
```

スナップショットが作成されていることを確認したら、Macをリカバリーモードで再起動してみましょう。起動時に⌘Command＋Rです(**図2、図3、図4、図5**)。

あとはここからTime Machineを選択し、バックアップの代わりにスナップショットされたディスクを選択すれば、スナップショットのリストが表示されます。リストアはやはり一瞬で完了します。

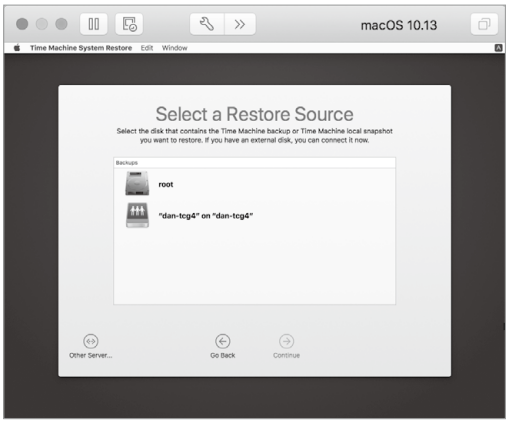
ファイルシステム全体をスナップショットでundoする場合はリカバリーモードでの再起動が必要ですが、特定のファイルだけを元に戻したい場合は普通にTime Machineに入れば、バックアップストレージが接続されていなくてもスナップショット上のファイルを元に戻せます(**図6**)。

▼図2　macOS UtilitiesでTime Machineからリストア

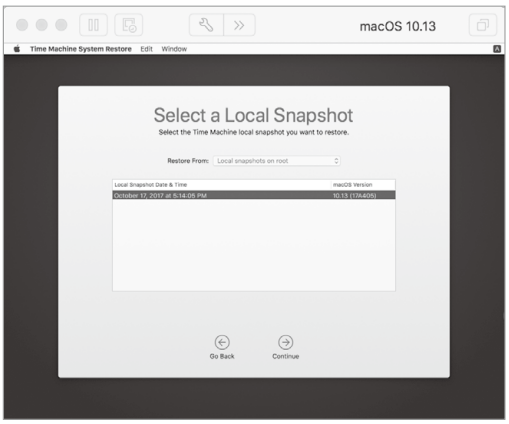


▼図3　リストア開始

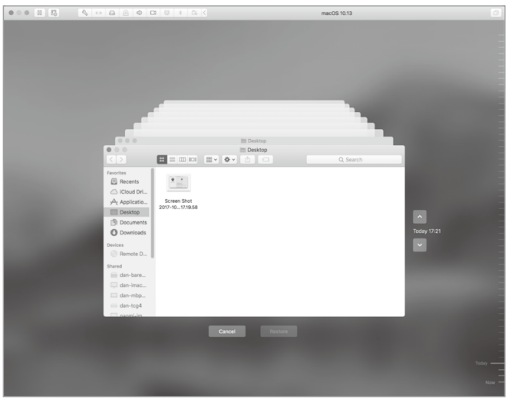
▼図4　リストア元の選択



▼図5　スナップショットを選択



▼図6　バックアップがなくてもローカルのスナップショットでリストア可能



なんだか良いことづくめのように思えるCoWですが、重大な欠点が1つあります。ファイルに上書きをしないということは、その分ストレージを食うということにもなります。実際、APFS上のゴミ箱を空にしても空き容量はすぐには減りません。ゴミ箱の中のファイルがスナップショットにあったら当然そうなります。また、Finderが申告する**空き容量**とdfコマンドの出力結果は食い違います。前者は「削除可能(purgeable)なスナップショットを削除した後の空き容量」を、後者は本当にどこにも使われていない空きブロックの容量をそれぞれ表示します。

macOSの実装は、なるべくスナップショットは残しておき、空き容量が本当に足りなくなって初めて古いスナップショットを消すという仕様になっているようです。常時バックアップ用のSSDが接続されている筆者のiMacでも、ほぼ24時間分のスナップショットが残っています。

ちなみに、これらのスナップショットは`tmutil`コマンドで削除することもできます。`tmutil deletelocalsnapshot YYYY-MM-DD-hhmmss`で日付を指定して1つずつ消すこともできますし、`tmutil thinlocalsnapshots`で「十分」な空き容量が確保されるまで古いスナップショットを消すこともできます。このあたりを手動でできるところも、ほかのApple製品とMacの違いですね。

なお大事なことなのであらためて申し上げると、スナップショットというのはバックアップを置き換えるものではありません。ハードウェアが壊れてしまえばスナップショットも一蓮托生。というわけでバックアップも相変わらず必要なのですが、そのバックアップを取る手法もスナップショットのおかげで変わります。

CoWではないHFS+では、ファイルシステムへの変更を常時監視して、書き込みがあったファイルをマークしてそれをバックアップするという方法をとっていました。しかしAPFSのバックアップでは、スナップショットどうしを比較するという手法になりました。常時監視しなくてもいい分、ずっとシンプルになったのです。

残念ながら、Time Machineのバックアップディスクは現時点ではHFS+なので、ZFSのsend/recvのようにスナップショットの差分そのものをまとめてブロック転送という手法は使えないのですが、近い将来には必ずそうなるのではないかと筆者は考えています。

さよなら パーティション

APFSのもう1つの特長は、物理的なパーティションを不要にしたこと。物理的にストレージをパーティションした場合、それぞれのパーティションの空き容量は物理的に固定されてしまいますが、APFSではまず「コンテナ」(container)があり、そこに論理的なヴォリューム(volume)を作成することで、コンテナの空き容量をすべて活用できます。ZFSをご存じの方は前者がpool、後者がdatasetというと理解しやすいでしょう。

図7と**図8**は起動ヴォリュームと同じコンテナにヴォリュームを追加した例。どちらも総容量が一緒であることが確認できます。

APFSではヴォリュームが論理的な存在です。実はこのことが、HDDやFusion DriveへのAPFS導入が見送られた理由でもあります。APFSというのはファイルシステムというよりメモリ管理システムのようだと感じた読者は鋭い。空き容量を増やすにはスナップショットを削除する必要があるというのはガベージコレクションのようですし、ヴォリュームがコンテナを共有するというのは実メモリをプロセスに仮想アドレスとして割り当てるような感じ。実はそうなのです。SSDというのはRAMに似て、どのブロックにもアクセスする速度は変わりません。だったらより利便性の高いメモリ管理に

▼図7　起動ボリューム

▼図8　ボリュームを追加

似せようとするのはソフトウェアエンジニアの願望といってもいいでしょう。ZFSでは実メモリをどっさり使ってRAMに似ていないHDDでそれを実現しました。SSDを前提とすればそれも不要です。そして今日日のApple製品は、廉価なデスクトップを除けばその前提条件がそろっています。実にAppleらしい割り切りっぷりではありませんか。

「APFS？　何それ知らない」と昔のハードウェアは言った

新開発のファイルシステムの導入はそれ自体がたいへんなものですが、それを標準とするにあたってはもう1つ問題があります。OSをそこからブートするにはどうすれば良いのでしょう？

ここで、新旧2つの起動ストレージを見比べてみましょう。どちらもdiskutil listの表示結果です(**図9**、**図10**)。

High SierraではSierraのころにあったRecovery Partitionが消え、APFSコンテナの中に収納されたことが見てとれます。それはよしとして、Prebootとはいったい何でしょう？

実は、現代のOSというのはmacOSを含め、最初にロードされるプログラムではありません。まずブートローダーという、OSをロードするプログラムが読み込まれ、そのブートローダーがOSをロードするという2段ロケットのようなしくみになっています。BIOS時代、パソコンはそもそもファイルシステムなんて知らず、起動ディスクの最初のセクター(これがMBR[Master Boot Record])を決め打ちで読み込んで、それがブートローダーを読み込んで、最後にやっとOSがロードされるというしくみでした。EFI時代になって、パソコンはFAT32でフォーマットされたEFIパーティションをサポートするようになりましたが、それにしてもハードウェアが知っているのはFAT32だけです。Macのファームウェアはそれに加えてHFS+もサポートしているので、たとえEFIパーティション

▼図9　Sierraの場合

```
/dev/disk0 (external, physical):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *68.7 GB    disk0
   1:                        EFI EFI                     209.7 MB   disk0s1
   2:                  Apple_HFS Macintosh HD            67.9 GB    disk0s2
   3:                 Apple_Boot Recovery HD             650.1 MB   disk0s3
```

▼図10　High Sierraの場合

```
/dev/disk0 (internal, physical):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *68.7 GB    disk0
   1:                        EFI EFI                     209.7 MB   disk0s1
   2:                Apple_APFS Container disk1          68.5 GB    disk0s2

/dev/disk1 (synthesized):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      APFS Container Scheme -                      +68.5 GB    disk1
                                 Physical Store disk0s2
   1:                APFS Volume root                    15.2 GB    disk1s1
   2:                APFS Volume Preboot                 19.9 MB    disk1s2
   3:                APFS Volume Recovery                519.9 MB   disk1s3
   4:                APFS Volume VM                      20.5 KB    disk1s4
```

が空でもブートできたのですが、それにしてもAPFSなんて知らないことは変わりません。

Prebootの役割は、FAT32とHFS+しか知らない現在のMacにAPFSを教えてあげるために存在するのです。

さらに今日においては、「コンピュータ」自体物理的な存在とは限りません。仮想マシンの利用が進んだ結果、物理的なコンピュータのことをわざわざベアメタルと呼ぶようになったぐらいです。Macもこの例外ではなく、仮想MacがあるおかげでTravis CI[注2]でSwiftコードもテストできるのです。

その仮想マシンにおけるAPFS Bootサポートですが、本原稿執筆現在、VMware Fusion 8.5.8以降がOK、VirtualBox 5.1.XがNGという状態でした。前者の場合も、Sierraの仮想マシンをAPFSにアップグレードしようとした場合はPreboot Volumeの作成に失敗するため、アップグレード経由ではHFS+のままです。インストール用DVDイメージを作成したうえで新規インストールする必要があります。おかげで本記事が執筆できたのですが、そこに至るまでずいぶん手間暇がかかりました。同じ苦労を読者に味あわせるのは忍びないので、DVDイメージ作成スクリプト[注3]をGistに置いておきます。App StoreからHigh Sierraのインストーラを(再)ダウンロードした状態でお使いください。"use at your own risk"で念のため。

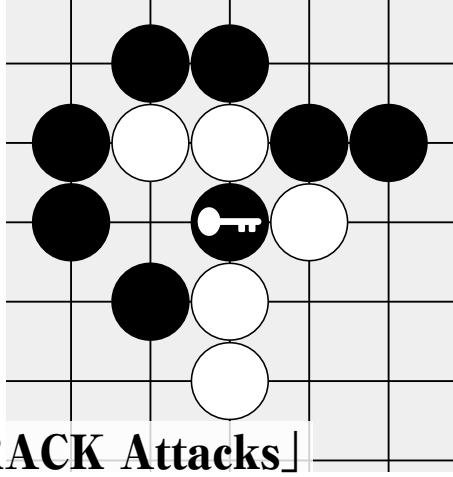
次回予告

どんなにすばらしい建築物も、土台が沈めば建物ごと沈んでしまいます。よって今回はXcodeの土台であるmacOSの新しい土台であるAPFSを検証しました。実際に体験してみて、強引に将来を見据えた、Appleという名を冠にするにふさわしい一品でした。次回はふたたび物を作るための建物であるXcodeとSwiftに焦点をあてます。SD

注2）URL https://travis-ci.org
注3）URL https://gist.github.com/dankogai/b7207666742ae5e3e8522b816a3278b1

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

【第四九回】WPA2の脆弱性への攻撃「KRACK Attacks」

2017年10月、無線LANのプロトコルWPA2（Wi-Fi Protected Access II）に、「第三者に暗号鍵が入手されて通信内容を盗み見られる可能性がある」脆弱性が発見されました。PC、スマートフォンなどあらゆる無線LANクライアントが影響を受けるということで、騒ぎになりました。

KRACK Attacks

2017年10月16日、ルーヴァン・カトリック大学（KU Leuven）のセキュリティ研究者Mathy Vanhoef氏は、WPA2の脆弱性に関するサイトkrackattacks.com（**図1**）を公開しました。同時に、ACM CCS 2017（ACM SIGSAC主催のカンファレンス）で発表される予定のWPA2の脆弱性に関する論文も、同サイト上で公開[注1]されました。

◆**図1　krackattacks.com（https://www.krackattacks.com/）**

脆弱性にキャッチーな名前を付けてニュースバリューを高める最近の流行にものっており、今回は「KRACK Attacks」あるいは「KRACKs」と呼ばれています。また、この問題に関しては、複数の脆弱性が報告されています[注2]。

無線LANのクライアント（子機）として使われているPC、スマートフォン、IP電話、その他の機材などは、ほぼすべてこの影響を受けます。基本的にクライアント側の脆弱性なので無線LANルータなどは問題なく使えますが、たとえば無線LAN中継のために子機として使っているようなケースでは、PCやスマートフォンと同様に影響を受けます。また、無線LANアクセスポイント（以下、アクセスポイント）に802.11r高速ローミングを導入している場合も影響を受けます（コラム「802.11rを導入しているアクセスポイントでの対策」を参照）。

Ubuntuは2017年10月17日にセキュリティアップデートが出ていますし、Windowsは2017年10月10日付けのアップデートで解消されています。本誌が読者の手元に届くころにはサポート期間が切れていない[注3]PCやスマートフォンに関してはすでにアッ

注1）"Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2"（**https://papers.mathyvanhoef.com/ccs2017.pdf**）
注2）10月25日時点では10個の脆弱性が報告されています。詳細は次のサイトを参照。"JVNVU#90609033 Wi-Fi Protected Access II（WPA2）ハンドシェイクにおいてNonceおよびセッション鍵が再利用される問題"（**http://jvn.jp/vu/JVNVU90609033/**）
注3）PCにしろスマートフォンにしろ、サポート切れになった機器もたくさん使われているとは思います。その点に関してはのちほど議論したいと思います。

プデートされていると思います。

WPA2のハンドシェイク時に攻撃

攻撃の種類は大きく次の4つになります。

(1) 4-wayハンドシェイクへの攻撃
(2) PeerKeyハンドシェイクへの攻撃
(3) グループ鍵ハンドシェイクへの攻撃
(4) BBS Translationハンドシェイクへの攻撃

すべて解説するには誌面が足りないので、(1)と(3)に話を絞ります。

WPA2ではハンドシェイクの情報を使い暗号通信を行う際の暗号鍵を生成するのですが、今回の攻撃ではハンドシェイク時に攻撃を行い、攻撃側の意図する鍵を使わせてしまいます。WPA2で暗号化されているつもりでも、攻撃者がこの攻撃を成功させた場合、暗号化の意味がなくなります。

たとえば、アクセスポイントとそこに接続するクライアント機器があるとします。クライアント機器とは、GNU/LinuxやWindowsが入ったPC、Mac、iPhone、Android端末などとします[注4]。

◉802.11rを導入しているアクセスポイントでの対策

802.11r高速ローミングの場合、これをdisable(無効)にしてもアクセスポイントの機能は果たすので、その方法を選択するのも手だとは思います。詳しくはメーカーサイトをチェックしてみてください。ちなみに、Ciscoに関しては次のサイトからたどるのが便利でした。

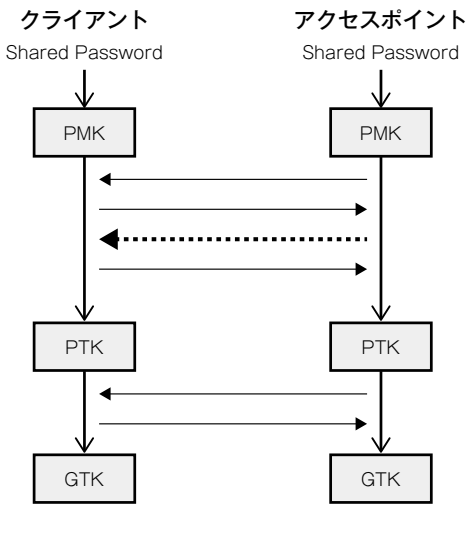
- CRITICAL 802.11R VULNERABILITY DISCLOSED FOR WIRELESS NETWORKS AS PART OF KRACK VULNERABILITIES
  (https://meraki.cisco.com/blog/2017/10/critical-802-11r-vulnerability-disclosed-for-wireless-networks/)

アクセスポイントとクライアントがWPA2で接続を開始する際、まず、データを送るときの暗号通信で使う鍵を作るための情報をやりとりします。その流れは**図2**のとおりです。

PMKの生成から、PTKをクライアントとアクセスポイントの双方で共有するまでに、4回やりとりするので4-wayハンドシェイクと呼ばれます。

PTKを共有してGTKを作り、そのGTKを使い暗号通信を行います。その際に使われる暗号化方式はCCMP(Counter-mode CBC-MAC Protocol)で、暗号プリミティブにはAESが使われます。今回は攻撃者が暗号鍵を入手してしまうという問題なので、暗号の強度などのファクターは関係ありません。

◆**図2　クライアントとアクセスポイント間のハンドシェイクのタイミング**



PTKを作る部分にて4回やりとりするので4-wayハンドシェイクと呼ばれる。4-wayハンドシェイクの脆弱性は3回目のメッセージを送る部分に攻撃を与える

・PMK (Pairwise Master Key)
共有しているパスワード(いわゆる無線LANのパスワード)から生成する鍵

・PTK (Pairwise Transient Key)
4-wayハンドシェイクを終了した時点で双方が持っている中間鍵

・GTK (Group Temporal Key)
実際にWPA2の通信で暗号化する際に使われる鍵

注4) IEEE 802.1Xに従えば、認証要求を出すクライアント(実際にはソフトウェア)側のことをサプリカント(Supplicant)と呼び、認証するアクセスポイント側をオーセンティケータ(Authenticator)と呼びます。ここではWPA2のクライアントとアクセスポイントの関係のみで閉じますので、サプリカントのことをクライアント、オーセンティケータのことをアクセスポイントと呼ぶことにします。

4-wayハンドシェイクへの攻撃

攻撃するための条件は、アクセスポイントとクライアントの間に、中間者攻撃（MitM、Man-in-the-Middle attack）をするための攻撃者の無線LANアクセスポイントが存在することです。

アクセスポイントとクライアントで4-wayハンドシェイクをするのですが、1、2、3回目のメッセージはそのままスルーします。4回目のクライアントからアクセスポイントに送られるメッセージは中間攻撃者がブロックします。これはちょうど無線の状態が悪くデータが届かないのと同じ状況です。アクセスポイントは再度、3回目のメッセージを送ります。このとき、クライアントは以前の3回目のときに用意されていたPTKは破棄し、再度、PTK（と同時にGTKも）を作りなおします。冒頭の論文では、このことをKey Reinstallationと呼んでいます。

この方法で鍵を操作できるので、暗号化されて情報が秘匿されているはずの無線LANの通信を攻撃者が解読できるという状況になってしまいます。

■大きな影響が出ているGNU/LinuxとAndroid

GNU/Linuxが採用している無線LAN認証のパッケージwpa_supplicantは、現状のディストリビューションで使われているであろうバージョンに大きな影響が出ています。

Mathy Vanhoef氏は鍵の不正再送による再設定によって暗号鍵をコントロールしようとしましたが、その過程で、wpa_supplicant v2.4～2.5のバージョン[注5]にはそもそもプログラムにバグがあり、暗号鍵がゼロクリアした状態になることを発見しました。そして、このバージョンのwpa_supplicantはAndroid 6.0.1にも使われており、同じことが起こります。プロトコル以前の問題として、かなりひどいバグだと言えるでしょう。

それ以前のwpa_supplicant v2.3、およびv2.6はKey Reinstallation攻撃が可能です。こちらはv2.4～2.5に比べれば、しかたがない範囲と言えるかもしれません（同じように攻撃可能なシステムが複数存在するので、マシとは言いませんが）。

グループ鍵ハンドシェイクへの攻撃

通信内容を暗号化するときにGTKを使います。GTKはずっと使い続けるわけではなく一定の時間で更新されます。GTK更新の際にアクセスポイント側からクライアント側に新しい鍵のためのデータが送られます。そのとき、そのデータ送付を中間攻撃者がブロックします。さらに中間攻撃者が以前の古いGTKに使われたデータをクライアントに送ると、クライアントは以前のものを使うことになる実装がされているとのことです。なぜそのような実装になったのか本当のところはわかりませんが、論文が示している動作を見ると、とにかく接続を切らずにつなげることを最優先したように見えます。

グループ鍵ハンドシェイクへの攻撃には、Attacking Immediate Key InstallationというすべてのWPA2実装で脆弱性を持つものと、Attacking Delayed Key InstallationというOpenBSDなどには効かないものの2つがあります。この論文には鍵の再送チェックと更新タイミングが示されている表（論文中の表2）があるのですが、この表をじっと眺めていると（組み込み）Linux系、組み込みOEM系、Windows系、OpenBSDに分類できそうな気がします。

この脆弱性の影響はスマートフォンもタブレットもPCも、とにかくWPA2が使えるクライアントすべてです。膨大な数にのぼるでしょう。このようにたいへんインパクトは大きいと言えます。

IoTのWPA2脆弱性対応

IoT向けの小さい組み込みチップの中にWi-Fiの機能が入っているようなものも、最近はよく見かけます。筆者もESP-WROOM-02というWi-Fiモジュールを使って、Wi-Fi経由で動作するIoT的な機材を実験的に作ってみたりしています。すでにこれらのチップ向けにWPA2脆弱性対応のパッチを

注5）リリースされていた期間はおおよそ2015年3月から2016年10月まで。

公開しているベンダもあります[注6]。なので、アップデートすることは可能です。

しかし、いったん何かに組み込まれたチップを回収して、アップデートイメージを書き込み、戻すといったことは現実的には難しいでしょう。

ユーザが無線LANについて考えるべきこと

今回、集中的にハンドシェイクの不備を見つけ、複数の脆弱性を見つけたことは、インパクトが大きかったのは事実です。サポートが切れていないPCやスマートフォンなどであればセキュリティアップデートを適用することで問題を解決できますが、WPA2が暗号化の目的を果たせていなかったことは事実です。また、セキュリティパッチの対応外の機材もあるので、WPA2が暗号の機能を十分に満たしていなくて困るというケースも、多くはないですがゼロではないはずです。

ですがそれは、ユーザが無線LANを使うときの致命的な問題なのでしょうか？

WPA2脆弱性のためにデータが暗号化されずパスワードが盗まれるといった説明をしている報道を見たのですが、通信する際はEnd-to-Endでデータを保護するのが基本です。家庭や職場の無線LANアクセスポイントなら信頼できるので大丈夫かもしれません。しかし街中のアクセスポイントやWi-Fiサービスの接続先に何が存在しているのかはわかりません。ネットワークの途中でパケットモニタリングされていてもユーザは知りようがありません。

そもそも無線LANでWPA2を使うのは、勝手にアクセスポイントが使われないようにユーザ認証で使っているのが主なのではないでしょうか。海外では暗号化していないWi-Fiサービスはたくさんあります。日本国内でもWi2などは暗号化をしていません。街角で提供している無料Wi-Fiも暗号化していないところをよく見かけます。

そんな公衆無線LANには簡単に偽アクセスポイントを作ることができます。また、ユーザは偽アクセスポイントではないと、こと細かに調べて使っているわけではありませんし、いちいちできません。

暗号はEnd-to-Endで使うのが基本

本来はHTTPSのようにEnd-to-Endで暗号化すべきでしょう。昔はSSL証明書を手に入れるために高額なお金を払わなければいけませんでした。しかし、今はLet's Encryptがあります。

もちろん通信すべてがWebサーバとのやりとりというわけではありません。信頼できないネットワーク区間はVPNで守り、信頼できるゲートウェイを経由してインターネットを使うという利用方法もあります。

筆者はコンテンツ類を置くためにVPSを借りています。そんなに高くはなく、そこそこお手ごろな値段のものです。そこにOpenVPNサーバを用意し、スマートフォン、タブレット、ノートPCのいずれにもOpenVPNクライアントを入れています。国内国外の公衆無線LANを使うときは、OpenVPN経由でインターネットに接続させています。これができないときは、よほどのことでない限りその公衆無線LANは使いません。

残念ながら日本では、VPNと言えば企業が使うものという印象があります。個人の価格帯で用意されているものは見つけられませんでした。しかし海外では、スマートフォン、タブレット、PCで使える価格帯が5～10ドルの個人向けVPNサービスを簡単に見つけられます。

技術のある人ならば月数百円のVPSを借りて、そこにOpenVPNのサーバを置き、手元のスマートフォンやタブレットでOpenVPNのクライアントを動かすのはたやすいことでしょう。

しかし、この安全性は望むユーザすべてに提供されるべきものなのです。そのようにして誰でも安全にインターネットを使えるようにサポートすべきなのです。早く日本でも手ごろな価格で個人向けVPNが出てくることを願っています。SD

注6）"Espressif Releases Patches for WiFi Vulnerabilities (CERT VU#228519)"
(http://espressif.com/en/media_overview/news/espressif-releases-patches-wifi-vulnerabilities-cert-vu228519)

SOURCES

レッドハット系ソフトウェア最新解説

第15回

レッドハットが提供する技術サポート

レッドハット製品を購入すると利用できるようになる、さまざまな技術サポートの内容や特典を紹介します。

Author 小島 啓史(こじまひろふみ)
mail : hkojima@redhat.com
レッドハット㈱ テクニカルセールス本部 ソリューションアーキテクト

レッドハットのサポート

レッドハット製品を購入すると、契約期間内で問い合わせ回数無制限の技術サポートを利用する権限が付与されます。これは「とりあえず何か困ったことがあれば、レッドハットに問い合わせができるのでは?」と思っている方が多いと思います。おおむねそのとおりなのですが、サポートの前提条件／対象範囲／内容を知ることで、より高度に活用できるようになります。ここではサポートの基本的な内容からあまり知られていなさそうな内容まで一通り示します。

サポートポリシー

製品サポートの対象範囲[注1]は次になります。

サポート対象内の項目

- インストール
- 使用方法
- 設定
- 診断
- バグの報告／修正
- Red Hat Extras チャンネル

サポート対象外の項目

- 変更されたRPM
- サードパーティのソフトウェア／ドライバおよび未認定のハードウェアおよびハイパーバイザ
- エンタープライズリリースのベースとなるコミュニティプロジェクト
- コード開発
- システムおよびネットワーク設計
- セキュリティールールおよびポリシーの導入と開発
- Red Hat Supplementary/Optionalチャンネル
- テクノロジプレビュー機能
- 依存関係を満たすために組み込まれるパッケージ、またはスタンドアロンアイテムとしてデプロイされる場合に「付随的に組み込まれるもの」

大きくまとめると、製品インストール／設定／利用中に何か問題が起こった場合のワークアラウンドやバグ修正を提供することはサポート範囲に含まれますが、ソフトウェアの改変や設計／開発、そしてレッドハットが提供するいくつかのソフトウェアはサポート範囲外となると

注1) URL https://access.redhat.com/ja/support/offerings/production/soc

いう話です。ちなみに「使用方法／設定」もサポート範囲に含まれるため、レッドハットの製品ドキュメント[注2]や製品のベースとなるコミュニティ版OSSのドキュメントなどに記載されている機能について、「この機能ってレッドハット製品に含まれているのか？　含まれているなら、どうやったら使えるのか？」といった問い合わせもできたりします。

また、テクノロジプレビュー機能については「ユーザの機能テストやフィードバックを歓迎します」という状態のものとなり、何らかの不具合が発生したとしてもワークアラウンドを提供しません。ただし、これらの機能のいくつかは、将来的には通常サポート範囲に含めていくことを目的としています。代表例がRHELのext4ファイルシステムです。ext4はRHEL5.3から登場[注3]しましたが、正式にサポートされるようになったのがRHEL5.6[注4]からです。このように重要な新機能については正式サポートに先取りして製品に搭載することがあります。どの機能がテクノロジプレビューかは各製品ドキュメントのリリースノートに記載されているので、参考にするといいでしょう。

少しわかりにくい点として「チャンネル」という単語があります。これはソフトウェアのグループを意味します。「Extras」チャンネル[注5]はAnsibleやDockerのような比較的新しい技術が用いられたソフトウェア、「Optional」チャンネルはOSSですがサポートされないソフトウェア、「Supplementary」チャンネルは再配布可能なプロプライエタリ・ソフトウェアのグループとなります[注6]。

サポートの利用

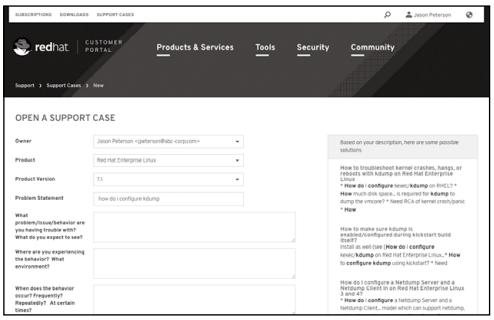
何かサポートに問い合わせたいことがある場合は、電話／チャット／サポートケースのいずれかを利用[注7]してサポートチームにアクセスします。このうちメインで利用されるのはサポートケースとなります。サポートケース上でのやりとりは世界中のレッドハットのサポートチームに共有され、問題解決速度の向上に役立っています。サポートケースは、カスタマーポータル上のサポートケース作成ボタンをクリックすると表示される**図1**のような画面に必要情報を入力して作成できます。ここでやりとりできる内容は製品利用方法や問題発生時の対応・解析のほかにも、機能拡張のリクエスト（RFE[注8]）やアーキテクチャレビューがあります。

RFEは製品の機能拡張対応期間だけ（RHELであれば運用フェーズ1と定義された期間[注9]。リリースからおよそ5年）投げることができます。

▼図1　サポートケースの作成画面イメージ

注2）URL https://access.redhat.com/documentation
注3）URL http://red.ht/2zmxajg
注4）URL http://red.ht/2zB184n
注5）URL https://access.redhat.com/ja/node/3127211
注6）URL https://access.redhat.com/ja/node/1520103
注7）URL https://access.redhat.com/ja/support
注8）URL https://access.redhat.com/ja/solutions/3175221
注9）URL https://access.redhat.com/ja/support/policy/updates/errata#Production_1_Phase

▼図2　サポートケースの画面イメージ



また、アーキテクチャレビューにより製品の構成について導入前にレビューを受けることで、レッドハットのベストプラクティスを適用して問題を予防できるようになります。アーキテクチャレビュー対象となる製品はRHELのクラスタソフトウェア[注10]やOpenStack[注11]などがあり、とくにこの2つについては製品ドキュメントだけでなくナレッジベースに記載されているサポート対象外の構成パターンもありますので(導入後に問題が発生して初めて気づき、慌てて火消し対応したこともありました)、レビューを受けることを推奨しています。こうした対応(RFEとアーキテクチャレビュー対応を除く)はあらかじめ定義されたSLA[注12]に従って実施されるため、少なくともまったく応答がないという状況にはなりません。

なお、サポートケース上ではこれまでのやりとりのほかにも、記載内容に関連すると推測されたナレッジベースのリンク集も自動的に表示されます(**図2**)。これらのナレッジベースは今まで発生した問題に対する解決策やレッドハット製品のベストプラクティスなどが記載されていますが、有効なレッドハットのサポート契約を持っていないと閲覧できないものが多々あります。ですので、レッドハットのナレッジを活用して類似した問題発生の予防もしたいと考える場合にも、サポート契約が役に立つと言えます。

こうしたサポートケースの利用ガイドについてスクリーンショットを添えて解説しているページ[注13]がありますので、参考にしてください。

さらなるサポートの特典

サポートの対応時間は契約の種類によって異なります。レッドハットの平日営業時間のもの(スタンダード)と24時間365日(プレミアム)のものがあります。プレミアムのほうがスタンダードよりも割高なのですが、問題の重大度によっては応答時間が短くなるなどの特典があります。さらに、一部の製品については次のような特典も付いてきます。

特定のマイナーリリースに対してのバグ／脆弱性修正の提供

レッドハットの製品は各メジャーリリース(RHEL6やRHEL7など)に対してライフサイクルが定義されています。たとえばRHELでは10年サポートを謳(うた)っていますが、これはRHEL6.0やRHEL7.0がリリースされてから10年のサポートを提供するという意味です。各メジャーリリースの中にはマイナーリリース(RHEL6.4やRHEL7.2など)があります。通常マイナーリリースに対してバグ／脆弱性修正が提供される期間は、次のマイナーリリースが提供されるまで(たとえばRHEL7.2に対してはRHEL7.3が出るまで提供。半年ほどの間、OSSコミュニティで提供する修正がバックポートされます)となっていますが、RHELのプレミアム契約を持つと特定のマイナーリリースに対してバグ／脆弱性修正が提供される期間が延長(最大2年)されます。これを延長アップデートサポート(EUS[注14])と定義しています。EUSはスタンダード契約を持っていても別途購入することで利用できるようになります。EUSを利用すると、マイナーリリースのアップデートごとに実施が必要

注10) URL https://access.redhat.com/articles/2359891
注11) URL https://access.redhat.com/solutions/1441243
注12) URL https://access.redhat.com/ja/support/offerings/production/sla
注13) URL https://access.redhat.com/ja/node/2435791
注14) URL https://access.redhat.com/ja/node/3055941

となるようなアプリケーションの再検証や再認定の数を削減できるようになります。

なお、EUSを利用していない場合、メジャーリリースのサポート期間中は各マイナーリリースに対しての問い合わせを受け付けますが、問題解決のために最新のマイナーリリースまでアップデートしてくださいと依頼することもあります[注15]ので注意ください。EUSについては執筆時点ではRHELだけですが、JBoss Enterprise Application Platform(レッドハット製品の1つであるJava EEアプリケーションサーバ)でも提供していく予定です。

プロアクティブチケット

深夜にRHELのカーネルをアップデートしてシステム再起動を行う予定があり、不測の事態に備えたいなどの状況に対応するためのサポートサービスとして、プロアクティブチケット[注16]があります。プロアクティブチケットの利用はプレミアム契約を持っている人に限定されます。4日前までにサポートケースを英語で作成しなければいけない(やりとりも英語に限定)という制約はありますが、これを利用するとレッドハットのサポートチームもあらかじめスタンバイ状態になり、迅速な対応ができるようになります。

まとめ

今回紹介した内容は、普段レッドハットのサポート内容を活用されている方でも知らない点があったのではないでしょうか。レッドハット製品でなくOSSコミュニティ版のソフトウェアを使っている方でも、こうしたサポート内容に相当するサービスを自前で用意せざるを得なくなってしまい、しかも技術的に不安があるという場合は、ぜひレッドハットサポート契約の購入を検討ください。SD

注15) URL https://access.redhat.com/ja/node/2710221

注16) URL https://access.redhat.com/ja/solutions/3040601

51

Debian Developer　やまねひでき　henrich@debian.org

DebConf17レポート（中編）

Debian Hot Topics

Debian 9.2リリース

10月7日にDebian 9.2がリリースされました。詳細はリリースアナウンス[注1]を参照ください。リリースの際にbase-filesパッケージの更新を忘れてしまっているため、一部の環境では/etc/debian_versionが9.1のまま、ということがあるようです（stable-updatesリポジトリを有効にしていると対応されたパッケージが適用されますので、気づかない人も多いかもしれません）。今回から、安定版については2ヵ月ごと、旧安定版については4ヵ月ごとのリリースが予定されているので[注2]、次回9.3は12月となります[注3]。

というところで、前回に引き続き今回もDebConf17[注4]の話題をお届けします。

Flatpak関連セッション

Simon McVittieさんの「A Debian maintainer's guide to Flatpak」は、Flatpakの説明セッションでした[注5]。FlatpakとはLinux用のアプリケーションフレームワークで、スマートフォン向けアプリに似た形でアプリケーションを提供します。依存関係にあるライブラリをまるごとアプリに詰め込んで提供するため、ライブラリバージョンの問題が発生しません。また、その際にサンドボックス化された環境で動作させ、必要なアクセス権だけを利用することでセキュリティを保ちます[注6]。特徴として次のものがあります。

- rpmやdebなどのシステムワイドなパッケージではなく、ユーザ単位で管理が行われる
- 「/usr」の小さなコピー（しかし、Flatpakのアプリケーション（app）は実際の/usr以下は参照しない）で、/app以下[注7]にインストールされる（決め打ちで移動したら動かない）
- インストールするのはアプリケーションが対象であり、initなどのシステム層には踏み込まない
- アプリケーションにはライブラリが同梱され、その選択はアプリケーションの作者が行う。また、ライブラリやアプリケーションのバージョンは作者が決める
- 基本的に個々のFlatpakパッケージは独立しているが、org.freedesktop.Platform、org.gnome.Platformのような共通ランタイムライブラリがあり、参照できるようになっている
- 各ランタイムライブラリは「branch」を持ち、appはランタイムライブラリの特定ブランチに依存する（例：org.gnome.gedit//stableがorg.gnome.Platform//3.24に依存）
- サンドボックス化の際、間違ったnamespace

注1）URL http://deb.li/3kjWc
注2）URL http://deb.li/iWz8y
注3）URL http://deb.li/pEN4
注4）例年行われるDebian開発者向け会議。今年はカナダのモントリオールで行われました。
注5）URL https://debconf17.debconf.org/talks/59/
注6）Linux用とあるのは、サンドボックスの技術として「namespace」や「cgroups」などのLinuxカーネル特有の機能を使うからです。*BSDなどでは動作しません。
注7）実際のデータは各ユーザの~/.var/appにインストールされるのですが、動作の際にはコンテナ化されて/appとして参照されます。

の利用があると、そこがまた攻撃対象となるので、「Bubblewrap(bwrap)」という小さなsetuidヘルパーでusernsのサブセットだけが使えるようにしている

これまではISV(独立系ソフトウェアベンダ)がアプリケーションをLinuxに提供しようとしても、ディストリビューションによってシステムの前提が異なっていたので「iOS用」や「Android用」のように「Linux用」という形でアプリケーションを用意できませんでした。Flatpakを使うことで、ディストリビューション起因の問題を考慮せずに済み、配布の問題を解決できるようになります。

また、ユーザとしても「『使う程度には信用している』が、『大事なデータ(GPG(GNU Privacy Guard)の秘密鍵などやシステム)にアクセスを許可するほどには信用していない』サードパーティ製ソフトウェアをインストールしたいときに、サンドボックス化されているFlatpakを使うことでリスクの軽減ができます。

すべてのパッケージがFlatpakで提供/解決できるかというとそうではないため、Debianのようなディストリビューションベンダはシステム部分を担い、更新サイクルが速いデスクトップアプリケーションは最適なバージョンでFlatpakを使う、という形が想定されています。

また、デスクトップアプリケーションでも、

- upstreamがきちんとメンテナンスをしているAppは、upstreamからFlatpakで
- upstreamの開発が停滞し、DebianがパッチなどでメンテナンスをしているAppは、Debianから(FlatpakでもdebパッケージでもI)

という形が良いのでは、とSimonさんは述べています。

Simonさんは、DebianパッケージからFlatpakランタイム・アプリケーションを作る「flatdeb」というプロトタイプを作っているそうです[注8]。

今後は通常の安定版リリース以外でもデスクトップアプリケーションごとにDebianからFlatpakパッケージとして適宜リリースする、という対応が可能になるかもしれませんね。

Flatpakの実例 ——魅力的なEndless OS

実際にFlatpakを使った例として、Debianをベースにしているものの直接aptなどは使わず、パッケージシステムとしてOSTree[注9]とFlatpakを使用している「Endless OS」[注10]というものがあります。DebConf17でも、「Debian meets OSTree and Flatpak, a case study: Endless OS」という発表が行われました[注11]。

Endless OSはおもに東南アジア諸国をターゲットにしたデスクトップ向けディストリビューションです。筆者もVMで動かしてみましたが、起動画面のアニメーションやデスクトップ画面がAndroidのような印象です(**図1、2**)。アプリケーションは、App StoreやGoogle Playス

▼図1　アニメーション表示される起動ロゴ

▼図2　起動直後からYouTubeやFacebookなどを起動するためのアイコンが表示される

注8) URL https://flatpak.debian.net/

注9) システムバイナリをパッケージ単位ではなくGitのようにツリー状にまとめて扱うしくみで、Flatpak内部でも利用されています。Endless OSでは既存のdebianパッケージをもとにしてOSTreeを利用しているようです。
URL http://deb.li/3QaED

注10) URL https://endlessos.com/

注11) URL https://debconf17.debconf.org/talks/41/

トアのように「GUIで一覧からワンクリックインストール」で済むようになっており、スマートフォンの利用が多い昨今のユーザにアピールできるものになっていると感じます。

LSB関連パッケージの削除理由は

Debian 9.0 "Stretch"ではLSB(Linux Standard Base)関連パッケージの大半が削除されました。通常、何かの標準に準拠するのは歓迎されることですが、ここではあえて逆の方向へと進んでいるように見えます。この背景にはどんなことがあったのか？　その詳細な説明が、Didier Raboud さんのセッション「Why I (tried to) killed the LSB」で行われました。

LSBはLinux Foundation配下のプロジェクトで、2001年にLSB 1.0が発表されました(**表1**)。ディストリビューション間での互換性を高め、その仕様に準拠したシステムならばどのシステムでも、同一のバイナリで動作するということを目的としています(そして、その最新の仕様書は1,771ページにもおよびます)。

発表者のDidierさんがDebianのLSB関連パッケージに携わったのは「当時のリリースクリティカルバグを直そうとしたときだけ」とのこと。そして成り行き上メンテナになったあとは、Canonical社がUbuntuに対して独自にLSB関連パッケージに変更を加えていたのをDebianへバックポートするなどしてメンテナンスに努めていました。しかし、作業を進めていくうえでLSB関連パッケージのメンテナンス自体にさまざまな疑問が湧いたようです。

LSBの要素とDebianでの対応

LSBは次の5つから成り立っています。

①FHS(Filesystem Hierarchy Standard)
②Init(System initialization)
③lsb_release
④ABI(Binary compatibility)
⑤RPM(RPM package format)

このうちlsb_releaseコマンドは、多くのディストリビューションで同様にサポートされており問題になりません。問題はほかの項目です。

FHSについては、そもそもDebianはDebianポリシーに準拠しており、Debianポリシーは FHS 2.3という極めて古いバージョンをベースに独自に拡張をしています。つまり、FHSに完全には準拠していません。initシステムについては、LSBはSysVinitを前提としてさまざまな標準を定めていますが、現在ではsystemdが標準となり、意味をなさなくなってきています。バイナリ互換(ABI)についても、このために多数のライブラリが必須となっていますが、後述するようにすでに使われていないバージョンのライブラリも強制されます。RPMについては、そもそもDebianはdebパッケージを利用しています(RPM互換のためにalienが導入されはしましたが)。このように5項目の内4つが達成できないか、対応の意味がないため、「DebianのLSB準拠」はかなり難しいものと言わざるを得ません。

さらに、LSBの歩み自体が遅い点も状況をややこしくします。2012年の時点でLSB 4.1ではライブラリとしてQt3が必須となっています[注12]。しかし、Qt3はこの時点の安定版Debian 7.0 "Wheezy"には含まれていません。なぜならば、Qt3は2005年の時点でQt4に置き換えられており、2007年7月にすでにEOLになっていたからです。Debianが2012年の時点でLSBに準拠しようとすると、upstreamで5年も前にサポートが完全に終了しているライブラリを同梱するという、たいへんありがたくな

▼表1　LSBの歴史

バージョン	リリース時期
LSB 1.0	2001年6月
LSB 2.0	2004年8月
LSB 3.0	2005年7月
LSB 4.0	2008年11月
LSB 5.0	2015年6月

注12) 2015年のLSB5.0でようやくQt3が必須条件から削除されています。

い選択をとらねばなりません。

このように、Debianにしてみると多大な努力を払って「何のために」LSBに準拠するのかが見えないのです。そして、LSBに準拠しようという姿勢を見せていたDebian 3.0 "Woody"のころでも、LSB 1.1はLinuxカーネル2.4を要求していたにもかかわらず、Debian 3.0はカーネル2.2だったために非準拠となっています。以来、一度もLSBに準拠したことはありません。そしてそれが大きな問題になったこともありません。ここで「ならばDebianにとってLSBは不要なのでは？」という結論に行き着きます。

LSBサポートがなくなると困るのか?

一方、「LSB準拠のソフトウェアにとってDebianがサポートをなくすのは問題では？」というと現状でそのようなソフトウェアは少数で、Google Earthとエプソンプリンタドライバの2つしか存在しないとのこと。これなら個々に働きかけて対応をしてもらうのも現実的です。

さらに、現在LSBに準拠しているディストリビューションはごく少数(**表2**)で、Ubuntuは相当古いバージョンでサポートを終了しています。対応するディストリビューションが追加される気配もありませんし、現行のディストリビューションが最新のLSBに準拠しようという姿勢も見られません。

- Debianは一度もLSBに準拠したことがない
- LSB準拠のためには労力がかかるが、そのLSBは現実の状況を反映していない
- LSB準拠のほかのディストリビューションはほぼ存在しない(＝何のための互換性？)

▼表2　LSBバージョンと準拠しているディストリビューション

LSBバージョン	準拠しているディストリビューション
LSB 4.0	Red Hat Enterprise Linux 6.0
	Oracle Linux 6
	Ubuntu 9.04 ※
LSB 4.1	Red Hat Enterprise Linux 7.0
LSB 5.0	なし

※2010年10月にすでにサポートが終了。

以上の条件がそろったので、Didierさんは「果たしてLSBに準拠する必要があるのか？」という問いかけを始めました。そして、

- 3年前のDebConf14から議論を始めたが反応は薄いまま
- 現状でもLSB認定アプリケーション一覧に記載されているのは、6つの企業による8個のアプリケーションのみで、たった1つだけがLSB 4以降の対応
- Debianに、LSBに準拠することを期待している人は誰もいない。「誰か興味ある人はいないか？」と聞いて3ヵ月待ったが反応は0だった

という結果、lsb-baseとlsb-releaseを残しLSB関連パッケージは全削除となったのです[注13]。

LSBを振り返って

「ディストリビューション間に互換性をもたせよう」という意図は良かったのですが、現状では次の点から意義がなくなってきています。

- 「実際に動いているものが標準」であってその逆ではない
- LSBの決定プロセスは遅すぎる(例：Qt3の削除)
- systemdが標準となり、initスクリプトの互換性問題は消えてきている
- ディストリビューション間の差異を吸収するツールとしては「コンテナ」があり、アプリケーションパッケージとしてもFlatpak/Snap/AppImageなどが出てきている

筆者も、「2001年当時にUNIXのプロプライエタリアプリケーションを移植するなどの後ろ盾のためにはLSBのお題目は役に立ったのかもしれないが、現状では役割を終えた」と感じます。今回のLSB関連パッケージの削除は妥当で、影響はほぼないでしょう。SD

注13) Debian 9 "Stretch"ではGoogle Earthとエプソンプリンタドライバのため、lsb-compatパッケージを作って一応の互換性をもたせるようにしてあります。

タイル型ウィンドウマネージャー「bspwm」を使う

Ubuntu Japanese Team
水野 源（みずの はじめ）

今回は、デスクトップ全体にウィンドウを「敷き詰める」ことができる、タイル型ウィンドウマネージャーの「bspwm」を紹介します。

タイル型ウィンドウマネージャーとは?

先日リリースされたUbuntu 17.10では、デフォルトのデスクトップシェルがUnityからGNOME Shellへと変更されました。使い慣れた環境が変わってしまうことに、抵抗を感じている方もおられると思います[注1]。

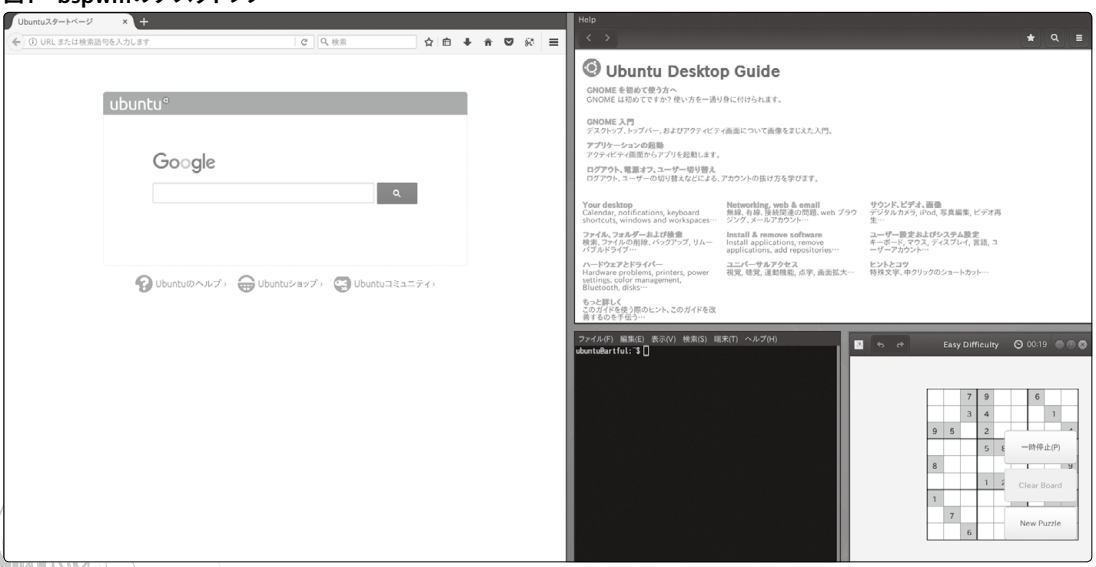
どうせ環境が大きく変化するのであれば、いっそのことまったく別のウィンドウマネージャーを試してみるのはいかがでしょうか？　今回は、CLIコマンドで設定するタイル型ウィンドウマネージャーの「bspwm」（**図1**）を紹介します。

GNOMEやKDE、さらにはWindowsやmacOSなどのデスクトップ環境では、デスクトップ上にウィンドウが「浮かんで」おり、それぞれのウィンドウを「重ねて」配置します。こうしたウィンドウマネージャーは「スタック型」や「コンポジット型」と呼ばれます。

これに対し「タイル型」と呼ばれるタイプのウィンドウマネージャーが存在します。タイル型ウィンドウマネージャーの特徴は、デスクトップ全体にウィ

注1）とはいえ17.10のGNOME環境では、おおむねUnityっぽい見た目を再現することに成功しています。

図1　bspwmのデスクトップ



ンドウを「敷き詰める」ところです[注2]。

開いているウィンドウが1つであればデスクトップ全体に最大化して表示し、ウィンドウが増減すると、デスクトップ全体にフィットするよう、すべてのウィンドウの大きさと位置を自動的に調整して再配置します。デスクトップ上で、これらのウィンドウが「重なる」ことはありません[注3]。

スタック型のウィンドウマネージャでは、ウィンドウを最大化していない限り、デスクトップ上にウィンドウが表示されていない「無駄なエリア」ができてしまいます。またウィンドウが重なっていると、下にあるウィンドウを操作する前に、最前面に移動させたり、上の邪魔なウィンドウを移動させなければならず、手間がかかります。もしも2つのウィンドウを並べて作業したいと思ったら、マウスドラッグでウィンドウサイズを調整し、手でウィンドウを配置する必要があります。

タイル型ウィンドウマネージャーは、こうしたウィンドウ配置、リサイズを自動的に行うため、限られた資源であるデスクトップ面積を無駄なく使えます[注4]。またすべてのウィンドウを常時俯瞰(ふかん)できるため、ウィンドウが行方不明になってしまうこともありません。

これだけ言うと、従来のデスクトップ環境とはまったく違うもののように思えるかもしれません。しかしWindows 7以降で採用されているAero Snap[注5]や、ウィンドウを最大化すると専用の操作スペースが生えてくるmacOSの挙動は、非常に「タイル的」であると言えるでしょう。そう、タイルはすでに、我々の日常に溶け込んでいるのです[注6]。

bspwmのインストールと初期設定

それではUbuntu 17.10にbspwmパッケージをインストールしましょう(**図2**)。

bspwm自身はキーボードの入力を処理しないため、bspwm単体では操作ができません。そこで別途ホットキーデーモンのsxhkdパッケージもインストールします[注7]。今回はプログラムランチャーとしてdmenuを使いたいため、suckless-toolsパッケージも追加でインストールします。さらに壁紙を表示するため、イメージビューアであるfehもインストールします[注8]。

次にbspwmとsxhkdの設定ファイルを用意します[注9]。ホームディレクトリの「.config」以下にbspwmとsxhkdというディレクトリを作成し、/usr/share/doc/bspwm/examples以下にあるbspwmrcとsxhkdrcというサンプル設定ファイルをコピーします(**図3**)。

注2) 一般的にウィンドウと呼ばれるものを、bspwmでは「ノード」と呼称します。本記事ではわかりやすさを優先し、ノードをウィンドウと表記します。

注3) とはいえ、それでは一部のアプリケーションで不都合があるため(例:GIMP)、特定のウィンドウのみを浮かす機能を搭載しているものもあります。

注4) この特性ゆえ、タイル型ウィンドウマネージャーでは壁紙がほとんど意味をなしません。人によっては、これが重要な問題になるかもしれません。

注5) デスクトップの左右にウィンドウが貼りつく自動整列機能のことです。Unityにも同様の機能が存在し、Super+Ctrl+カーソルキーでウィンドウを効率よく配置できます。一般的なキーボードでは、Superキーは⊞キー(Windowsマークキー)に割り当てられています。

注6) そういえば、Windows 1.0はタイル型でしたね。

注7) bspwmパッケージはsxhkdパッケージに依存しているため、明示的に指定しなくてもインストールされます。

注8) 前述のとおり、タイル型ウィンドウマネージャーにおいて壁紙を表示することはほとんど意味がありません。しかしUbuntu 17.10のbspwm 0.9.3では、ウィンドウを閉じたときやワークスペースを移動したとき、最後に描画していたウィンドウの表示が残ってしまう(背景がリフレッシュされてないような)挙動に遭遇しました。壁紙を表示することでこの問題を回避できるため、とりあえずのワークアラウンドとして利用しています。

注9) これを忘れたままbspwmを起動すると、何も表示されず、キー入力すらいっさい受け付けない無敵のデスクトップが起動してしまいます。しかたないのでCtrl+Alt+Fnで仮想コンソールを切り替えてkillするのも、bspwmあるあるです。

図2 bspwmと関連パッケージのインストール

```
$ sudo apt install bspwm sxhkd suckless-tools feh
```

図3 bpswmとsxhkdの設定ファイルの用意

```
$ mkdir -p ~/.config/{bspwm,sxhkd}
$ cp /usr/share/doc/bspwm/examples/bspwmrc ~/.config/bspwm
$ cp /usr/share/doc/bspwm/examples/sxhkdrc ~/.config/sxhkd
```

bspwmの起動時（ログイン時）にはbspwmrcファイルが実行されるのですが、これは単なるシェルスクリプトです。bspwmはbspcというコマンドを実行して各種設定や操作を行うため、設定ファイルイコール、コマンドを列挙したシェルスクリプトになるというわけです。

bspwmrcの先頭で、sxhkdを起動しています。以後はキーボードからのショートカットキー入力をsxhkdが受け取り、キーに対応したbspcコマンドを実行してbspwmを操作するというしくみです。ショートカットキーの組み合わせと実行されるコマンドは、sxhkdrcに記述されています。

bspwmでは、Super + Enterキーで端末を起動します。bspwmの標準ターミナルはurxvtに指定されているのですが、Ubuntuはデフォルトでurxvtがインストールされていないため、このままでは端末を起動できません。そこでsxhkdrcの該当部分をgnome-terminalに書き換えます[注10]（**図4**）。それに加えて筆者は、Unityと同様にCtrl + Alt + tでも端末を起動するよう、キー設定を追加しています（**図5**）。

最後に、bspwmrcの最後にfehコマンドを記述して、壁紙を設定します（**図6**）。表示する画像はお好みのものを指定してください。

以上の設定ファイルのコピーと編集ができたら、一度Ubuntuを再起動します。GDMのログイン画面で「サインイン」ボタン左隣の歯車のアイコンをクリックし、セッション「bspwm」を選択してからサインインしてください（**図7**）。

注10）もちろん設定を書き換えずにurxvt（rxvt-unicodeパッケージ）をインストールしたり、あるいは別の端末を指定してもかまいません。

bspwmの基本的な操作

起動直後のbspwmには、壁紙以外何も表示されていないため、何をしてよいかわからないかもしれません。とりあえずSuper + Enterで端末が起動することと、Super + Alt + Escでbspwmを終了できることを覚えておいてください。これさえ忘れなければ、なんとかなるはずです。

Super + Spaceでプログラムランチャーのdmenu_runが起動します。dmenuは「Dynamic Menu」の略で、標準入力から受け取ったリストをキーボードから検索可能にするGUIインターフェースです。Go言語のpecoコマンドのようなもの、と言えばイメージがつきやすいかもしれません。そのdmenuのラッパースクリプトで、環境変数PATHの中からコマンドの一覧を取得し、dmenuに渡すのがdmenu_run（リ

図4　Super+Enterキーで起動するプログラムを、urxvtからgnome-terminalに変更する

```
super + Return
    urxvt
      ↓
super + Return
    gnome-terminal
```

図5　Ctrl+Alt+tでgnome-terminalを起動するショートカットキー設定を追記する

```
ctrl + alt + t
    gnome-terminal
```

図6　壁紙を設定するコマンドを追記する

```
/usr/bin/feh --bg-scale /usr/share/backgrounds/warty-final-ubuntu.png
```

図7　bspwmセッションを選択してログインする

図8　dmenuからアプリケーションを起動する

```
gnome-sc    gnome-screensaver  gnome-screensaver-command  gnome-screenshot
```

スト1）です。dmenu_runを起動するとデスクトップ上部にバーが表示され、キー入力欄と選択候補の一覧が表示されます（**図8**）。Unityのアプリケーション Lensのように、起動したいプログラム名をキーボードから入力、もしくはカーソルキーで候補を選択して Enter を押してください。

sxhkdのサンプルにある、bspwm用の設定をそのまま利用しているのであれば、**表1**のショートカットが使えます[注11]。複雑で最初はとっつきにくいかもしれませんが、少しずつ慣れていきましょう。

画面ロックの設定

一定時間PCを操作しなかったら画面をロックし、ディスプレイを消灯してほしいですよね。当然ですがbspwmにそのような機能はないため、i3lockとxautolockを使ってこの機能を実現します。まずこの2つのパッケージをインストールします。

```
$ sudo apt install i3lock xautolock
```

bspwmの起動時に、xautolockも同時に起動させたいため、bspwmrcの最後に**図9**のコマンドを記述します。

「-time」オプションはタイムアウトまでの時間を分で表したもので、デフォルトは10分、最小は1分、最大は60分です。ここに指定した時間が経過すると、「-locker」オプションに指定したロックコマンドが実行されます。ロックコマンドの起動後、

リスト1　dmenu_runの内容

```
#!/bin/sh
dmenu_path | dmenu "$@" | ${SHELL:-"/bin/sh"} &
```

注11）これが設定されているショートカットキーのすべてではありません。より詳しくは設定ファイルを読んでみてください。

表1　デフォルトのbspwmのショートカットキー

ショートカットキー	動作
Super ＋ Enter	端末を起動する
Super ＋ Space	dmenu（ランチャー）を起動する
Super ＋ Esc	sxhkdの設定をリロードする
Super ＋ Alt ＋ q	bspwmを終了する
Super ＋ w	アクティブなウィンドウを閉じる
Super ＋ m	ウィンドウレイアウトを切り替える（モノクル／タイル）
Super ＋ y	アクティブなウィンドウを、デスクトップ内で最大のウィンドウと入れ替える
Super ＋ t/T/s/f	アクティブなウィンドウをタイル／疑似タイル／フローティング／フルスクリーン表示に変更する
Super ＋ Ctrl ＋ y	アクティブなウィンドウをスティッキー（すべての仮想デスクトップに表示）にする
Super ＋ c	次のウィンドウへフォーカスを切り替える
Super ＋ Shift ＋ c	前のウィンドウへフォーカスを切り替える
Super ＋ h/j/k/l	隣接したウィンドウへフォーカスを切り替える
Super ＋ Shift ＋ h/j/k/l	アクティブなウィンドウを隣接したウィンドウと入れ替える
Super ＋ /	前／次の仮想デスクトップへ切り替える
Super ＋ 1 ～ 0	指定した仮想デスクトップへ切り替える
Super ＋ Shift ＋ 1 ～ 0	アクティブなウィンドウを指定した仮想デスクトップへ移動させる
Super ＋ ↑/←/→/↓	アクティブなフローティング状態のウィンドウを移動する
Super ＋ Alt ＋ h/j/k/l	アクティブなウィンドウを拡大する
Super ＋ Alt ＋ Shift ＋ h/j/k/l	アクティブなウィンドウを縮小する
Super ＋ Ctrl ＋ h/j/k/l	次に開くウィンドウの位置を（現在アクティブなウィンドウの中で）事前指定する
Super ＋ Ctrl ＋ Space	事前指定したウィンドウの位置をキャンセルする

図9　bspwm起動時にxautolockを起動する設定

```
/usr/bin/xautolock -time 1 -locker '/usr/bin/i3lock -n' -killtime 10 -killer 'xset dpms force suspend' &
```

「-killtime」オプションに指定した時間が経過すると、今度は「-killer」オプションに指定したコマンドが実行されます。つまりこれは、操作せずに1分経過するとi3lockで画面をロックし、そこからさらに10分が経過すると「xset dpms force suspend」コマンドを実行してディスプレイを消灯する、という意味です。なおkilltimeのデフォルトは20分、最小は10分、最大は120分となっています。

PCの前から離れるときなど、1分を待たずに即座にロックしたい場合もあるでしょう。そのためsxhkdに手動ロックのショートカットキーを追加しておくと便利です。sxhkdrcに**リスト2**の設定を追記してから、sxhkdをリロードしてください。Super + Ctrl + Esc キーで画面がロックされるようになります。

自動起動するプログラムの設定

ownCloudのクライアントなど、ログイン時に自動的に起動してほしいアプリケーションがありますよね。UnityやGNOMEでは、gnome-セッション-propertiesで自動起動するアプリケーションを設定していました。しかし言うまでもありませんが、bspwmにそのような機能はありません。そこで、ディレクトリ内にある.desktopファイルを実行してくれるコマンド、「dex」を使います。

```
$ sudo apt install dex
```

dexコマンドに「-a」オプションをつけて実行すると、「/etc/xdg/autostart」と「~/.config/autostart」以下の.desktopファイルを読み込み、順次実行します。特定のディレクトリ内の設定だけを実行したい場合は、サーチパスを指定する「-s」オプションを追加して、次のようにします。

```
$ dex -a -s ~/.config/autostart
```

リスト2　i3lockを起動するショートカットキーの追加

```
# display lock
super + ctrl + Escape
    i3lock
```

desktopファイルの中にはOnlyShowInというエントリが存在することがあります。これは自動起動するデスクトップ環境を限定するためのものですが、とくに指定がない場合、dexはこのエントリを無視します（すべて実行します）。もしも特定の環境向けのプログラムの実行をスキップしたいような場合は、「-e」オプションに現在の環境名を指定するとよいでしょう。たとえば「-e bspwm」のように、予約されていない環境名[注12]を指定すると、OnlyShowInに（bspwm以外の）指定があるプログラムはいっさい実行されなくなります。

```
$ dex -a -e bspwm
```

bspwmrcの末尾に追記しておくとよいでしょう。

カスタマイズの例

デフォルトでは、フォーカスのあたっているウィンドウが非常に視認しづらいでしょう。次のコマンドを実行すると、フォーカスのあるウィンドウのボーダーが太く、オレンジ色になり、視認しやすくなります。

```
（ウィンドウのボーダーを目立たせる設定）
bspc config focused_border_color '#FF8000'
bspc config border_width         4
```

自分好みになるよう、色やボーダー幅を調整してみてください。ちょうどいい色とサイズが決まったら、bspwmrcに追記しておくとよいでしょう。

ウィンドウのフォーカスはショートカットキーでしか変更できませんが、図の設定で、マウスクリックでウィンドウを選択可能になります。

```
（マウスクリックでウィンドウの切り替えを可能にする設定）
bspc config click_to_focus true
```

bspwmの機能についてより詳しく知りたい場合は、bspcコマンドのマニュアルを読んでみてください。SD

注12）https://specifications.freedesktop.org/menu-spec/latest/apb.html

Software Design plus　技術評論社

IBM Bluemix

IBM Bluemix クラウド開発入門

IBMのクラウドサービスであるBluemixを、基本的な導入方法の解説から実際のアプリケーションを作る方法まで紹介します。Bluemixの特徴はさまざまな事例に支えられた豊富なサービス群です。アプリケーションを効率的に開発・運用するためのDevOpsについて工夫が凝らされており、最近話題のAI:拡張知能利用のためのWatsonAPI等も提供されてます。IoTについても各種の対応が施されており、本書ではRaspberry PiとWatsonを組み合わせた事例を紹介しています。スマートなクラウド利用の手引きとしてご活用ください。

常田秀明、水津幸太、大島騎頼 著、
Bluemix User Group 監修
B5変形判／288ページ
定価（本体2,800円+税）
ISBN 978-4-7741-9084-6

大好評発売中！

こんな方におすすめ

- クラウドへのシステム移行を考えている方
- クラウド上のソフトウェア開発を体験してみたい方
- クラウド上でのIoTと拡張知能を組み合わせて新しいサービスを考えたい方

Software Design plus　技術評論社

進化する銀行システム

24時間 365日動かすメインフレームの設計思想

人々の生活や企業活動を支える銀行のオンラインシステム。地震などの大規模災害対策として、銀行の国際競争力を高める手段（振込の24時間化や休日・夜間の即時決済など）として、24時間365日止まらずに稼動し続けることが求められています。本書は、多くの銀行システムで採用されているメインフレームのしくみ、とくに信頼性、可用性、保守性を高める技術をわかりやすく解説します。銀行システムの歴史や特性、メインフレームやそのOS「z/OS」の特徴や機能を学びたい人、システムの障害・災害対策を検討したい人に役立つ1冊です。

星野武史　著、花井志生　監修
A5判／256ページ
定価（本体2,580円+税）
ISBN 978-4-7741-8729-7

大好評発売中！

こんな方におすすめ

- 金融機関システムの開発／運用に携わっているSE
- 金融機関のシステム部門のSE

Unixコマンドライン探検隊
Shellを知ればOSを理解できる

Author 中島 雅弘(なかじま まさひろ) ㈱アーヴァイン・システムズ

sedとawkのスクリプトを使ってみましょう。また、シェルとテキスト処理ツールの連携を、実用的な事例をみながら体験します。

第20回 テキスト処理(その4) ——パスワード管理ツールを作ってみる

sedやawkを使ううえでの注意

sedとawkがとても強力で機敏なテキスト処理ツールであるのは、連載第17回「テキスト処理(その3)」(本誌2017年9月号)で紹介しました。今回はステップアップを目指して、sedとawkはもちろん、テキスト処理ツールをシェルとともに使いこなします。

はじめる前にちょっと注意しておきましょう。これまで何気なく使っていたsedやawkですが、これらにはバージョンや派生があります。各OSで標準的に入っているものは、POSIX準拠の機能に限定されていたり、日本語などのマルチバイト言語のロケールへの対応がきちんとできていなかったりと、本連載で紹介する機能をそのまま使えないことがあります。

本連載で対象にしているsedとawkは、GNUのsedとawkです。コマンド名がgsedだったりgawkだったりすることもあります。GNU sedやgawkは、POSIXよりも多くの機能があり、広く普及しているうえ、ロケール対応も進んでいます。運用・保守上の事情で、POSIX準拠や標準インストールしているものを選択しなければならないなら、できる範囲を見極めて使いこなすようにしてください。

次に示したコマンドと出力はバージョンを確認した場合の例です。日本語文字列をうまく扱えないなど、コマンドの動作がどうもおかしいと感じたら、使っているコマンドのバージョンを確認してみるとよいでしょう。

```
macOS brew環境での例
$ /usr/local/bin/sed --version↵
sed (GNU sed) 4.2.2
...略...
$ /usr/bin/sed --version↵
/usr/bin/sed: illegal option -- -
...略...
$ /usr/local/bin/awk --version↵
GNU Awk 4.1.4, API: 1.1 (GNU MPFR 3.1.6, ➘
GNU MP 6.1.2)
...略...
$ /usr/bin/awk -version↵
awk version 20070501
```

本格的なテキスト処理2

ワンライナーでは少々手に余る複雑なテキスト処理は、スクリプトを使って処理します。そして、Unixらしく小さなツールを組み合わせる手法で、より複雑な処理にも対応できるようにします。

スクリプト——sedを一歩踏み込んでみる

「行頭に〜を挿入して、文字列○○を××に置き換えて、x行目からy行目までの空行は削除する。それをディレクトリ内にあるN個のファイルに対して処理したい」といった場合、対話型エディタで処理するのはとてもたいへんです。sedスクリプトを使えば、こうした定型的処理

を一発で解決できます。

sedスクリプトを記述するうえで基本となるルールは、次の3つです。①スクリプト中のすべてのコマンドは、それぞれの行に対して、記述された順に適応される。②編集コマンドは、行アドレスで制限されなければ、すべての行が対象となる。③入力元のファイルは変更されず、編集結果は標準出力へ。

sedスクリプトの簡単な例として、タブで区切られた表形式のテキストファイルを、Wikiなどで使われている、|で区切った表形式に変換するスクリプトを作ってみましょう。次の例では、行中にタブが含まれていない行は編集の対象にしません。タブを含まない行でそれぞれの表を区切れば、複数の表を含んでいても一括で変換できます。

table.sed：タブで区切られたテーブルをWiki形式のテーブルに変換する

```
/\t/s/^/|/      ←タブを含む行に対して、行頭を|に置換
/\t/s/$/|/      ←タブを含む行に対して、行末を|に置換
/\t/s/\t/|/g    ←タブを含む行に対して、すべてのタブを|に置換
```

たとえば、次のようなタブ区切りテキストを処理してみましょう。

table.tsv(注：区切りはタブを含む)

```
ABC DEF GHI
1stカラム 2ndカラム 次のカラム

name address phone
Irvine 品川区 03-5475-5001
Mike Smith 世田谷区 040-2222-2222
```

sedに-fオプションを付けてスクリプトファイル名を指定します。

実行と結果

```
$ sed -f table.sed table.tsv↵
|ABC|DEF|GHI|
|1stカラム|2ndカラム|次のカラム|

|name|address|phone|
|Irvine|品川区 03-5475-5001|
|Mike Smith|世田谷区 040-2222-2222|
```

GNU sedで動作させています。冒頭で紹介したように、macOSに標準で入っているsed(/usr/bin/sed)では、動作が異なります。

この例は短く単純なスクリプトですので、スクリプトファイルに記述するまでもなく、「その3」で紹介した;でコマンドを並べて次のように記述するか、

```
sed '/\t/s/^/|/; /\t/s/$/|/; ➪
/\t/s/\t/|/g' table.tsv
```

続く文字列がsedスクリプトに追加する命令であることを示す、-eオプションを用いて、次のようにしても良いですね。

```
sed -e '/\t/s/^/|/' -e '/\t/s/$/|/' -e ➪
'/\t/s/\t/|/g' table.tsv
```

sedは、通常処理の対象にならない行もそのまま出力します。出力を抑止したい場合は、-nオプションを指定します。明示的に出力する行はpコマンドを使います。次の例は、タブを含む行に対して置換が生じた行だけを表示するように、前出のスクリプトをほんのちょっと修正しています。

```
$  sed -n '/\t/s/^/|/; /\t/s/$/|/; /\t/ ➪
s/\t/|/gp' table.tsv↵
|ABC|DEF|GHI|
|1stカラム|2ndカラム|次のカラム|
|name|address|phone|
|Irvine|品川区 03-5475-5001|
|Mike Smith|世田谷区 040-2222-2222|
```

sedは、デフォルトでは入力ファイルに対して変更を加えませんが、-iオプションを指定すれば、入力ファイルを変更できます。

STEP UP! シェルとテキスト処理ツールの連携

さまざまなテキスト処理を連携して、テキスト中の任意の位置にある「~~で括(くく)られた」文字列を暗号化・復号するツールを作りましょう。テキスト処理をawkスクリプトで担い、CUIの対話的インターフェースをbashスクリプトで実現します。暗号化したファイルは、Dropboxで管理することもできます。

なお、ここで扱うソースコードenc_dec.awkとssti.shは、誌面の都合ですべてを掲載することができません。サポートページ[注1]からダウンロードしていただけますので、そちらを参照

注1) http://gihyo.jp/magazine/SD/archive/2017/201712/support

ください[注2]。

ソフトウェアの外部仕様はさておき、スクリプトから見ていきましょう。

テキスト中の任意の位置を暗号化するawkスクリプト

まず、テキストの暗号化処理をするawkスクリプトを作成します。あとに作成する対話インターフェースだけでなく、バッチやWebアプリケーションからも呼び出すことを考慮して、awkスクリプト単独でも動作するようにします。

awkスクリプトは次の形式で記述します。

```
パターン1 { アクション1 }
パターン2 { アクション2 }
...
```

パターン(正規表現、式)にマッチする行を読み込むと、続くアクションが実行されます。BEGINとENDは特殊なパターンで、**BEGIN**：データ入力の開始前、**END**：すべてのデータを読み込んだ後、に対応します。

それではさっそく、それぞれのパターンに対して、どのような処理をしているか見てみましょう。

BEGIN▶ファイルの読み込み前の処理(3〜15行目)

オプションに対する処理をします。パスワードが指定されていなければ、ここで入力を促します。modeに"enc"を指定すると暗号化モードで、それ以外は復号モードで動作します。

NR==1 && mode=="enc"▶読み込む各ファイルの1行目で暗号化処理の場合(18〜25行目)

複数のファイルから目的の情報を検索するにあたって、fgrep[注3]を使いますが、デフォルトではすばやくスクリーニングするために1行目を対象にします。データを登録する際に、全角アルファベットと数字を半角に、/を.に置き換える、単純な正規化をtrコマンドで施します。

/~.*~/▶~で挟まれたパターンを見つけたとき(28〜51行目)

~で挟まれた文字列がある行は、挟まれた文字列を暗号化もしくは復号します。

```
037:     act | getline converted
038:     close(act)
```

変数actに入っている文字列で外部コマンドopensslを実行し、処理した文字列をパイプを使ってgetline関数に受け渡します。awkは、

注2) [プログラムに対するコピーライトと免責]
ここで掲載したいずれのスクリプト／プログラムも、学習・実験用を目的としております。オリジナルの著作は著者に帰属しますが、読者が断りなく任意に利用・改変することができます。また、これらを使って生じるいかなる損害について、著者および技術評論社は一切の責任を負いません。

注3) 正規表現を使わないことで、特殊記号などを含んでいても素直に文字列が見つけ出せるようにfgrepを使います。

STEP UP! systemライブラリファンクションについての考察

ソースコード中の5行目、

readを呼び出しているところ
```
005:          "read -sp \"-password:\" ↵
pass; echo ${pass}" | getline pass
```

の記述により、awkスクリプトを単独で実行するときの引数-v pass="xxx"を省略すると、macOSではパスワードの入力プロンプトを表示してパスワードを入力できるようにしていますが、Ubuntuでは次のようなエラーメッセージが出てしまいます。

```
sh: 1: read: Illegal option -s
```

エラーメッセージから、readのオプションに-s(入力を端末に表示しない)を指定できないようです。bashでは-sオプションが使えますが、ほかのシェルでは使えないものがあります。外部コマンドの実行には内部的にsystem(3)ライブラリファンクションが呼び出されます。man 3 systemにあるように、systemはshを実行しているのです。

system関数の呼び出しにより、macOSではbashが動いています。Ubuntuの/bin/shはbashではなく、dashというシェルです。dashのreadには、-sオプションがないので、上記のような動作になります(ここでは、後述のシェルスクリプトからパスワードが必ず渡されてくるので、Ubuntuで動作させるための修正は加えていません。動かすだけなら-sオプションを取ってしまえばよいのですが、入力したパスワードが画面に表示されるので安全ではありません)。

いずれの環境でも、man shでshが何のシェル(bash、dash、その他)を実行しているのか確認できます。

外部コマンドをパイプ[注4]で連結します。コマンドの呼び出しが終了したら、オープンしたファイルをcloseすることが重要です。この例においても、各対象を処理するたびに新しくファイルをオープンしファイルディスクリプタを消費します。ファイルディスクリプタの数は有限ですので、closeをしないといずれファイルがオープンできなくなってしまいます。

!/~.*~/ ▶暗号化不要の行の処理(53～55行目)

~で挟まれた文字列を含まない行は、そのまま出力します。

fmatchとgmatch(59～87行目)

そして、このスクリプトでのミソは、最左最短の連続マッチを実現している関数、fmatchとgmatchです。~XXXX~のように、特定のパターンで括られた文字列を抽出できるようにしました[注5]。awkの正規表現は、最左最長マッチですので、最左最短でパターンを抽出したい場合は、このような工夫が必要です。

このawkスクリプト(enc_dec.awk)は、実行パーミッションを設定すれば次のように単独でも動作します[注6]。

```
$ enc_dec.awk  -v mode="enc" -v
pass="パスワード" 入力ファイル名
$ enc_dec.awk  -v pass="パスワード"
入力ファイル名
```

シェルスクリプトによるユーザインターフェース

スクリプトssti.shを見てください。大まかに3つの機能に集約できます。

❶対象ファイルを検索する部分:関数lookup(147～210行目)

fastlookupモードでは、awkを使って1行目中を検索。全文を検索対象にするlookupモードでは、fgrepを使って検索しています。いずれもアルファベットで大文字小文字は区別していません。

オプションを指定しない全ファイル検索では、見つかったファイル一覧を自前のページャ機能でページングしています。195行目で、次のようにsttyコマンドの結果をawkで解析して、1画面の行数を抜き出しています。awkを使うことで、LinuxとmacOS間におけるstty出力の違いを吸収しています。

```
lines=$( stty -a | awk -F ";" 'NR==1{for
(i=1; i<=NF; i++) print $i}' | awk '/rows
/{ sub("rows","",); sub(";","",); print }' )
```

❷暗号化処理をする部分:関数enc_action、each_dir、encode(47～108行目)

暗号化するときに1行目を正規化するtr[注7]の呼び出しは、enc_dec.awk内の処理に合わせています。

注4) 本誌2017年6月号 第14回参照。

注5) 拙著『AWK実践入門』(技術評論社)に掲載した、最左最短マッチ関数fmatchと連続マッチ関数gmatchを少しハックしています。

注6) 1行目の#!(シェバン:shebang)には、gawkの場所を指定してください。macOSのbrew環境では、/usr/local/bin/awkを指定しています。

注7) sedで置き換えています。

trをsedで代用する

macOSのbrew環境では問題ありませんが、Ubuntu 16.04のtrは、全角文字列をうまく扱えません。これを回避するため、次に示すようにsedの「y」コマンドを使って、trと同じように動作する文字マップを実現しています。yコマンドは、文字を1対1で置き換えるコマンドです。一般には区切り文字に「/」を用いますが、ここでは「/」がディレクトリ記号であり、置き換えの対象にしたいので、区切り文字を「;」としています。連載第17回で解説しましたが、コマンドによってCロケール(POSIXロケール)以外への対応がきちんとできていないことがあります。あるコマンドが使えないケースに備えて、同様な処理をほかのコマンドで置き換えることができるか、普段からシミュレーションしておくと良いでしょう。

ほぼ同じ動作をする、trの文字置き換えとsedでの文字置き換え

```
$ tr 'A-Za-z0-9' 'A-Za-z0-9'"

$ sed 'y;ABCDEFGHIJKLMNO
PQRSTUVWXYZabcdefghi
jklmnopqrstuvwxyz
0123456789/;ABCDEFGHIJKLMNOPQ
RSTUVWXYZabcdefghijklmnopqrstuvwxyz
0123456789.;'"
```

3 閲覧(復号／復号せず)する機能：
関数decode(111〜114行目と179行目)

外部コマンドlessを使ってページング処理をしています。閲覧を終了したら、lessさえ抜けておけば画面上に結果が残らないので、うっかり端末を放置しておいても画面上にドキュメントは表示されていないので安全です。

暗号化と復号処理は、まるっとenc_dec.awkを呼び出しています。

◆ ◆ ◆

さらに、今まできちんと解説していなかった部分や、特徴的な記述部分をピックアップして見ていきましょう。

このスクリプトでは、処理対象のディレクトリをどこにするかを賢く扱っています。

```
dirbase="${WINHOME:-${HOME}}/Dropbox/sstdb"
```

この223行目は、変数WINHOMEが定義してあればそれを採用し、なければHOMEを使うという記述です。

このあと、ディレクトリ($dirbaseの値)の存在確認を[コマンド[注8]の-dオプションでおこない、なければ事前に設定してあるローカルディレクトリ($localdirの値)としています。

コマンド引数のオプションを解析するなら、bashの内部コマンドgetoptsが便利で汎用的です。getoptsの使い方の定跡は、whileループとcase文で処理します。getoptsコマンドは、最初の引数にオプションとして指定する文字を指定します。第1引数中で現れる:は、直前の文字に続いて空白文字でオプションから区切って引数を取れることを意味します。ここでは、dオプションを指定したあとに、ディレクトリ名を指定しています。引数は、OPTARGに保存されています。シェルスクリプト実行時のオプションは、1つずつ読み取られて変数OPTに入ります(231〜244行目)。

getoptsを使っているところ
```
231: while getopts d:lqaceE? OPT
232: do
233:   case $OPT in
234:     d) dirbase="${OPTARG}"; ;;
...略...
240:     E) mode="forceencode"; ;;
241:     \?) usage_exit ;;
242:   esac
243: done
244: shift $((OPTIND - 1))
```

使ってみよう

enc_dec.awkは、sstiと同じディレクトリにあることを前提としています(214行目)ので、/usr/local/bin/にパスが通っているとして、ssti.shを/usr/local/bin/sstinに、enc_dec.awkを/usr/local/bin/enc_dec.awkに配置します。

次の例では、a.txt、b.txt、c.txtという3つのファイルを作成して、~/.sstdb/source/以下に配置しています。Dropboxフォルダを用意していないので、スクリプトが自動で~/.sstdb/ディレクトリを操作の対象とします。暗号化前のファイルは、Dropboxが存在していても、安全のためそこには配置しません。ここに含めるファイルは、暗号化されていませんので、別のユーザが閲覧できないようにしておきます。

実行例 暗号化処理
```
$ ssti -e↵
Dropboxではなく、/home/masa/.sstdbを使います。
mode=encode
----
password: ←パスワードを入力します
a.txt encrypted, and added.
b.txt encrypted, and added.
c.txt encrypted, and added.
```

暗号化されたファイルは、~/Dropbox/sstdb/

注8) testコマンドですね。

注意：コマンド引数上の秘密情報のあつかいについて

マルチユーザ・マルチタスクOSであるUnixは、psコマンドなど別の利用者がどのようなコマンドをどのように実行しているか／したかを把握するすべがあります。この例でもコマンドラインを使ってパスワードを引き渡しています。

この方法では、自分が専有するデスク／ラップトップ・ワークステーションでない場合には、別のユーザに引数の情報を見られるリスクがありますので注意してください。ここでは触れませんが、より安全にコマンド間で秘密情報を連携するには、少し工夫が必要になります。

asstes/があればそこに、なければ~/.sstdb/assets/以下に保存します。Bash on Windowsでは、環境変数WINHOMEに、Windowsユーザのホームディレクトリを設定しておけば、その配下のDropboxディレクトリを対象にします。最初の1行を正規化(全角アルファベットは半角にするなど)してファイルのタイトルにします。また、検索時にはデフォルトでこの文字列が対象になるので、わかりやすい名称を1行目に記述してください。

暗号化処理のたびにパスワードを指定します。また、暗号化後に次のようにすれば、平文のソースファイルを削除できます。

暗号化されていないファイルを一掃する
```
$ ssti -c
```

ソースをアップデートしたら、再度暗号化しましょう。-eオプションは、更新があったファイルだけを再暗号化します。-Eは、強制的に暗号化上書きします。

続いて、"office"を先頭行に含む情報を検索して、内容の表示をさせてみます。sstiに引数を指定しなければ、すべてのファイルが対象です。-aオプションを指定すると、ファイル全体を対象にキーワードを検索します。

見つかったファイルは1つずつ表示され、それぞれに対してどのような処理をするか、s:暗号を解かずに表示、d:復号して表示、e:復号してファイルに保存、r:そのファイルを削除、n:何もせずスキップ、q:プログラムを終了、と指示を待ちます。

"office"を探す
```
$ ssti office↵
mode=fastlookup
----
[fast lookup] office
:: found 2 files ::

"Officeの合言葉" : (Show/Decode/Export/
Remove/Next/Quit)? d↵  ←復号して表示
Officeの合言葉
~~山~~といえば~~川~~
~~目には~~といえば~~歯を~~
...略...

"Office内のサーバ" : (Show/Decode/Export/
Remove/Next/Quit)? s↵  ←復号しないで表示
Office内のサーバ

onigashima01:
user=~~z5/iXARIs7XAUZBo5yfCFw==~~
passwd=~~yzodgd15+ZMWPEoNxwLjxw==~~
...略...
```

いかがでしたか。さらなるプログラムの内部仕様や機能についての詳細は、探検隊らしくソースを読んで確認してみてください。理解にあたっては、スクリプト中のコメントも参考になるはずです。

今回の技術が活躍するところ

sedとawkのスクリプトを記述しました。また、テキスト処理コマンドをbashの技で組み合わせて、テキストを暗号化するツールを作成しました。awkの標準機能にはない、最左最短マッチ関数なども、コマンドラインでの作業やスクリプト記述において実用的に使うことができます。

次回について

次回は、ネットワーク関連のコマンドを探検します。SD

今回の確認コマンド

【manで調べるもの
(括弧内はセクション番号)】
sed(1), awk(1), pipe(2), tr(1), grep(1), openssl(1), system(3), sh(1), ps(1)
【helpで確認】
getopts, case, read
【書籍で確認】
『AWK実践入門』(技術評論社)

Linux カーネル観光ガイド

第68回 ディスクデータの検証用の誤り訂正 ～dm-verityのFEC機能

Text：青田 直大　AOTA Naohiro

10月23日に、Linux 4.14-rc6がリリースされています。rc6は比較的大きくなったので、さらなる修正も多くなると予想されています。したがって、Linusはrc8まで必要になるとみているようです。いずれにせよ、本誌が発行されるころにはちょうどLinux 4.14が出ているのではないかと思います。

今回は、ディスクデータの正当性を検証するdm-verityの、FECという誤り訂正の機能について紹介します。

dm-verity

dm-verityは、verity(真実、真理)という名前のように、デバイスのデータを「真実」かどうか検証する機能を持った読み込み専用のデバイスを提供する機能です。もともと、この機能はChromium OSのディスク向けに開発され、ディスクの改ざんを検出するために使われています。

さっそく、dm-verityのデバイスを作成してみましょう。**/dev/libvirt_lvm/verity-data**は、2GBのデバイスで、Ext4のファイルシステムを作成して、ルートディレクトリの"foo"というファイルに"FOOBAR_OK"を書いています。**/dev/libvirt_lvm/verity-hash**は、Ext4を作成したデバイスのデータのハッシュが保存されるデバイスです。こちらは32MBとしています。

dm-verityを設定するには、まずハッシュデバイスにデータデバイスのハッシュを書く、「フォーマット」を行う必要があります。これには、"cryptsetup"パッケージの"veritysetup"コマンドを使います(**図1**)。"veritysetup format <データデバイス> <ハッシュデバイス>"によって、データデバイスのハッシュが計算され、ハッシュデバイスに書き込まれます。さらに、UUIDや使用するハッシュアルゴリズムやハッシュのsaltなどの情報が出力されます。これらのほとんどは、ハッシュデバイスのヘッダに記録されます。しかしながら、"Root hash"というハッシュデータに対する、「最上位」のハッシュはディスク上の、どこにも記録されません(厳密に言えば、ディスク上のデータから計算することはできますが)。この"Root hash"の値は、trusted bootなど改ざんから保護された部分に保存され、ディスクの最終的な検証に用いられます。

次にフォーマットしたデバイスからdm-verityデバイスを作成します。後述するように、"dmsetup"コマンドで構築することもできますが、引数が多いため"veritysetup create"を使う方が便利です。"veritysetup create <dm-verityデバ

イス名> <データデバイス> <ハッシュデバイス> <Root hash>"というコマンドを実行します。ここではデバイス名として"verity"を使い、先ほどフォーマットしたデバイス、およびそのRoot hashを指定してコマンドを実行すると、システム上にread-onlyのデバイスとして、"/dev/mapper/verity"が認識されます(**図2**)。このデバイスはread-onlyのファイルシステムとしてmountでき、先ほど作っておいたファイルが読み取れることもわかります。

では、データが改ざんされていたらどうなるでしょうか。"veritysetup remove"を使って、一度verityデバイスを削除します。そして、ファイルを書き換えて再度verityデバイスを作ってmountしようとするとどうなるでしょうか。ファイルを書き換えても、ふたたび"veritysetup create"を実行すると、"Veritydevice detected corruption after activation."と、デバイスは追加したものの、データの破損が検出されていることが出力からわかります。

このとき、データを読もうとしてもI/Oエラーで失敗しますし、当然mountもsuperblockが読めない、ということで失敗します。dmesgを見てみると、デバイスverityのデータブロック0が破損しているというログが連続して出力されており、"reached maximum errors"というログも出ています。"…… is corrupted"は、ハッシュが不一致した場合に出るエラーです。そして、このエラーが一定回数(100回)起こると、"reached maximum errors"と出力され、これ以上ログが出なくなります。さらに、"Buffer I/O error ……"とI/Oエラー、Ext4のsuperblockを読めないとのエラーも出ていることが見てとれます。

ここで、"data block 0"以外では読み取れるブロックもあることに注目してください。たとえば、**図3**のように"4096＊2"byte目からのブロックは、データが書き換えられておらず、したがってハッシュが変わっていないのでデータ

▼図1　dm-verity デバイスのフォーマット

```
$ sudo mount /dev/libvirt_lvm/verity-data /mnt/tmp
$ echo FOOBAR_OK | sudo tee /mnt/tmp/foo
FOOBAR_OK
$ sudo umount /dev/libvirt_lvm/verity-data
$ sudo veritysetup format /dev/libvirt_lvm/verity-data /dev/libvirt_lvm/verity-hash
VERITY header information for /dev/libvirt_lvm/verity-hash
UUID:                   b534e080-fe11-44e5-b25e-6de0be6a4f0c
Hash type:              1
Data blocks:            524288
Data block size:        4096
Hash block size:        4096
Hash algorithm:         sha256
Salt:                   5166fa4da700d6c2ece92b6762d628c47d8dd16e8f5b496b43b980e81ebff75e
Root hash:              b5a03dc094d2162f4432904b25474c553191a0acd452294e646218977dc30066
```

▼図2　dm-verity デバイスの認識

```
$ sudo veritysetup create verity /dev/libvirt_lvm/verity-data /dev/libvirt_lvm/verity-hash ↵
b5a03dc094d2162f4432904b25474c553191a0acd452294e646218977dc30066
$ lsblk /dev/mapper/verity
NAME   MAJ:MIN RM SIZE RO TYPE  MOUNTPOINT
verity 254:16   0   2G  1 crypt
$ sudo mount /dev/mapper/verity /mnt/tmp
mount: /mnt/tmp: WARNING: device write-protected, mounted read-only.
$ mount|grep verity
/dev/mapper/verity on /mnt/tmp type ext4 (ro,relatime,data=ordered)
$ cat /mnt/tmp/foo
FOOBAR_OK
```

を読み取ることができています。すなわち、今回のようにmount/umountでsuperblock(の最後のmount時間など)が書き換えられず、あるファイル中の一部データだけが化けたような場合であれば、そのファイル以外はまったく問題なく読み込みができるということになります。

dm-verityの構造

では、dm-verityのハッシュデバイスの構造を見ていきましょう。ハッシュを使って、あるデータの改ざんを確認するには、最も単純にはデータ全体のハッシュをとるのがよいでしょう。この場合、最初にデータ全体を読んでハッシュを確認すればデータ改ざんをチェックできます。しかし、ファイルシステムにこの方法を適用するとどうなるでしょうか。一番最初、すなわちmountのタイミングでディスク全体を読み取ってハッシュを計算することになります。これでは、ファイルシステムのmount、ひいてはシステムの起動に多大な時間がかかることになります。さらには、ファイルシステムの中にはシステムの利用中、まったく読み取られないブロックもありえます。こうしたブロックは本来は検証する必要がなかったはずなのに、ハッシュの考慮に入れてしまうという無駄が発生します。

するとディスクをある程度の大きさのブロックに分割し、そのブロックごとにハッシュをとるのがよいように思われます。しかし、これではブロックの数だけのハッシュが必要となります。たとえば、先ほどの2GBのファイルシステムで、ブロックサイズを4KB、使うハッシュをsha256とすると16MB分のハッシュが生成されます。これは、ディスクが大きくなればなるほどにサイズが大きくなり、trusted bootなど改ざんされない環境にいつでも置けるというわけにはいきません。

ここでこの16MBを、もう一度sha256のハッシュをとれば、32byteで「データブロックのハッシュ列」を検証できるというのがポイントです。このように、データブロックのハッシュを取る、

▼図3　データが改ざんされた場合

```
(一度、デバイスを削除)
$ sudo veritysetup remove verity
(改ざん済のデバイスを再度読み込み)
$ sudo veritysetup create verity /dev/libvirt_lvm/verity-data /dev/libvirt_lvm/verity-hash ⏎
b5a03dc094d2162f4432904b25474c553191a0acd452294e646218977dc30066
Verity device detected corruption after activation.
$ sudo hexdump -C /dev/mapper/verity
hexdump: /dev/mapper/verity: Input/output error
$ sudo mount /dev/mapper/verity /mnt/tmp
mount: /mnt/tmp: can't read superblock on /dev/mapper/verity.
$ dmesg
(...中略...)
[2351296.941343] device-mapper: verity: 254:14: data block 0 is corrupted
[2351296.942940] device-mapper: verity: 254:14: data block 0 is corrupted
[2351296.944353] device-mapper: verity: 254:14: reached maximum errors
[2351323.322495] buffer_io_error: 91 callbacks suppressed
[2351323.324230] Buffer I/O error on dev dm-16, logical block 0, async page read
[2351323.327280] EXT4-fs (dm-16): unable to read superblock
[2351323.329855] EXT4-fs (dm-16): unable to read superblock
[2351323.332422] FAT-fs (dm-16): unable to read boot sector
[2351328.704935] Buffer I/O error on dev dm-16, logical block 0, async page read
$ sudo hexdump -C -s $(( 4096 * 2 )) /dev/mapper/verity|head
00002000  02 80 00 00 02 80 01 00  02 80 02 00 02 80 03 00  |................|
00002010  02 80 04 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
(...後略...)
```

そのハッシュ列をブロックに分割してハッシュを取る、さらにそのハッシュ列のハッシュを取る……と階層的に「ハッシュのツリー」を構築していきます。すると、一番上のハッシュ、すなわち“Root hash”からそれぞれのデータブロックのハッシュまでの正当性を検証できます。

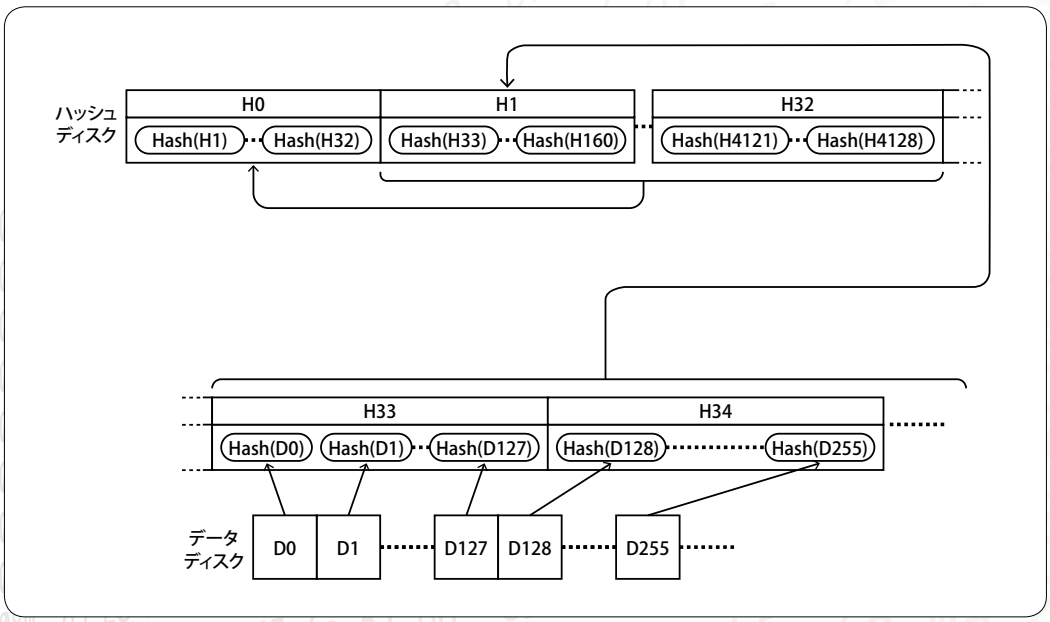
では、先ほどの2GBのディスクにおいて、どのようにハッシュが計算され、どのようにハッシュデバイスにハッシュが書き込まれるのかを見ていきましょう。“veritysetup format”のコマンド出力にあるとおり、データブロックのサイズ＝ハッシュブロックのサイズ＝4KBです。すると、データブロックの数は2GB/4KB＝524,288個となります。sha256のハッシュは、1つあたり32byteで、1つのハッシュブロック(4KB)に128個のハッシュが入ります。すなわち、データブロックのハッシュは、524,288/128＝4,096個のハッシュブロックに書かれることになります。このデータブロックのハッシュが書かれるハッシュブロックを、「レベル0のハッシュブロック」と呼びます。これら4,096個のハッシュブロックのハッシュは4096/128＝32個のハッシュブロックに入ります。このレベル0のハッシュブロックを「レベル1のハッシュブロック」と呼びます。このようにレベルが増えるごとに、そのレベルのハッシュブロックの数が減っていきます。実際、レベル1のハッシュブロック32個のハッシュは1つのレベル2のハッシュブロックに収まります。

これらハッシュブロックは、**図4**のように先頭のヘッダの入るブロックに続けて、レベルの高いハッシュブロックからレベルの低いハッシュブロックへの順に並べて保存されています。なおヘッダには、UUID・使用するハッシュの種類・ブロックのサイズ・saltなどが記録されています。

ここまでで、ハッシュデバイスの構造がわかりました。ここでRoot hashを計算してみましょう。Root hashはsaltと最もレベルの高いハッシュブロックとから計算されます。

まず、“veritysetup dump”コマンドを使ってヘッダ上の情報を見てみましょう(**図5**)。ここで、ハッシュタイプ、使用するハッシュのアルゴリズム、saltがわかります。次にsaltおよび最もレベルの高いハッシュブロックをダンプしておきます。Hash type＝1のdm-verityでは、salt

▼図4　ハッシュツリーの構造



▼図5　Root hashの計算

```
（パラメータの確認）
$ sudo veritysetup dump /dev/libvirt_lvm/verity-hash
VERITY header information for /dev/libvirt_lvm/verity-hash
UUID:                   b534e080-fe11-44e5-b25e-6de0be6a4f0c
Hash type:              1
Data blocks:            524288
Data block size:        4096
Hash block size:        4096
Hash algorithm:         sha256
Salt:                   5166fa4da700d6c2ece92b6762d628c47d8dd16e8f5b496b43b980e81ebff75e
（saltのダンプ）
$ echo 5166fa4da700d6c2ece92b6762d628c47d8dd16e8f5b496b43b980e81ebff75e |xxd -r -l 32 -c 32 -ps - > salt
（root blockのダンプ）
$ dd if=/dev/libvirt_lvm/verity-hash of=root-block bs=4096 skip=1 count=1
（Root hashの計算）
$ cat salt root-block| sha256sum
b5a03dc094d2162f4432904b25474c553191a0acd452294e646218977dc30066  -
```

はブロックデータの前につけられるので、そのようにcatコマンドで連結して、sha256のハッシュを計算します。すると得られたハッシュはたしかに**veritysetup**コマンドで出力されていた"Root hash"と一致しています。

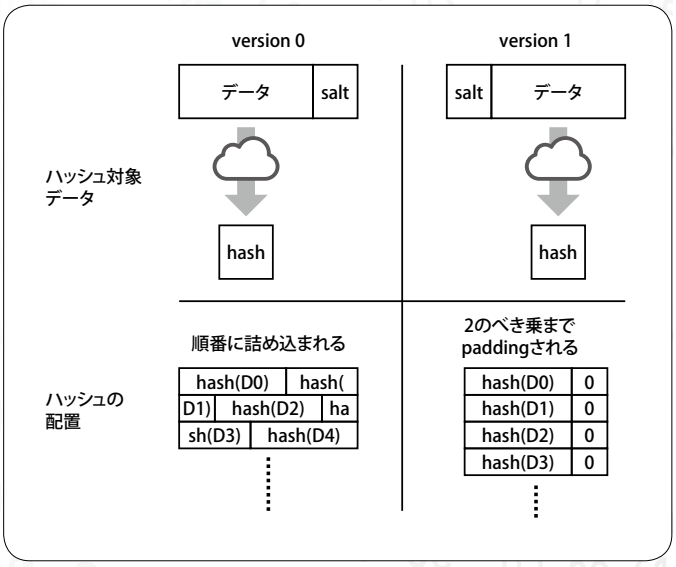
dm-verityのバージョン

先ほど「Hash type = 1のdm-verityでは」と言いましたが、実はdm-verityには一応2つのバージョンが存在しています(**図6**)。Hash type = 0がオリジナルのフォーマットで、Chromium OS向けに使われていたものです。一方、Hash type = 1が現行のバージョンです。

これら2つのバージョンには、2つの違いがあります。1つはsaltの位置です。バージョン0では、saltをハッシュ対象のデータの後ろにつけるのに対して、バージョン1では前述したようにsaltをデータの前につけます。

もう1つはハッシュブロック内のハッシュの配置方法です。バージョン0では単純にリニアに並べていましたが、バージョン1では2のべき乗にアライメントされて配置されます。今回使っているsha256の場合、どちらでも変わりはありませんが、ほかのハッシュのサイズが2のべき乗でない場合には配置が変わってきます。またべき乗単位で配置することで演算の効率化をはかっている面もあります。

▼図6　dm-verityのバージョンの違い



エラー訂正

Linux 4.5からは、このdm-

verity に Forward Error Correction(FEC：前方誤り訂正)の機能が追加されています。dm-verityのFECは、リード・ソロモン符号を用いて、データ・ハッシュの破損を修復します。

dm-vertiy では、231から253byte分のデータまたはハッシュを1つのパリティブロックとして、そこに“255-<パリティブロックサイズ>”byteのパリティをつけます。すなわち、2から23byte分のパリティがつくことになります。dm-verityでは、最大でこのパリティサイズまでのbyte数のエラーを修復できます。一般のリード・ソロモン符号ではデータ列のどこが壊れているのかが不明であるため、パリティ数の半分までのエラーまでしか修復できませんが、dm-verityの場合、ハッシュの検証によって壊れている場所がわかっているのでパリティの数だけのエラーに対応できるのです。

パリティの配置

以後は、パリティが2byteの設定での話を進めます。このとき、253byteのデータごとに、パリティ2byteが生成されることになります。ここでどのようにこの253byteを配置するのかという問題があります。単純な方法として、ディスク上で連続した253byteを使うことが考えられます。しかし、一般にディスクのエラーはセクタごとなど、連続した領域に発生します。すると、ディスク障害でデータがまったく復元できなくなってしまいます。

そこで、dm-verityはパリティ対象の253byteを、ディスク全体にバラバラに、なるべくそれぞれのbyteが離れるように配置します。すなわち、データブロックとハッシュブロックの空間を合わせて253分割し、その分割の1単位ごとに1byteを割り当てていきます。

具体的に、先ほどの2GBのディスクでの例を見ていきましょう。この場合、データブロックが524,288個で、ハッシュブロックが1 + 32 + 4,096 = 4,129個でした。これらを253分割すると、(524,288 + 4,129)/253 = 2088.60…… と2,089ブロックずつの領域に分割されます。すなわち、データブロック0の0byte目は、データブロック2,089の0byte目、ブロック4,178の0byte目……と組み合わされパリティが計算されます。同様にブロック0の1byte目は同じブロックの組み合わせの1byte目と合わせてパリティが計算される、といった具合いになります。dm-verityでは、パリティの数だけのデータバイトが失われても復旧ができるので、2,089 * 2ブロック、おおよそ16MB分のデータが連続して失われても残りの部分からデータを復旧できます。

FECの使い方

最後に、FECを実際に使ってみましょう(**図7**)。FECを使うには、cryptsetupの2.0.0-rc0が必要です。まず、先ほどとほぼ同様にフォーマットを行います。ここで“--fec-device=”引数でパリティを保存するデバイスを指定します。また、ここでは先ほどと違ってハッシュデバイスの部分をファイルに変えています。どうやら現状のバージョンでは、ハッシュデバイスが使用するハッシュ領域よりも大きい場合に、パリティ計算がおかしくなるバグがあるようです。この問題を、ハッシュデバイスをファイルにすることで回避しています。

“veritysetup format”が終わるとハッシュとパリティが書き込まれています。ここでディスクの先頭256KBをゼロで埋めてしまいましょう。dm-verityがうまくデータを復帰させなければ、とうていmountすらできません。

そして、先ほどとはコマンドの名前(と引数の順序)が変わり、“veritysetup open”でデバイスの認識を行います。ここでRoot hashを間違えて指定すると、ずっとエラー訂正をし続けてデバイスが外せなくなってしまうことがありますので気をつけてください。このタイミングで先頭部分にI/Oが来るので、修復が始まります。

dm-verityで訂正されたデータは書きもどされ

ないことに注意してください。たとえば、/dev/libvirt_lvm/verity-dataを直接mountしようとしても、先頭がゼロうめされてしまっているのでmountできません。したがって、dm-verityのディスクを作りなおしたり、あるいはキャッシュが消えたりすると、そのたびにI/Oが発行されパリティが読み込まれ、ふたたびデータが復旧されることになります。この復旧には大量のI/Oと計算が必要であるため、かなりの時間がかかり、多くのデータがうしなわれた状況でこのデバイスを使うのはあまり実用的ではないかもしれません。それでも、2GBあたり16MBという程度のオーバーヘッドで万一の破損を透過的に解決できる、というのはなかなかおもしろい機能ではないでしょうか。SD

▼図7　FECのテスト

```
$ sudo ~/cryptsetup-2.0.0-rc0/src/veritysetup format /dev/libvirt_lvm/verity-data verity-hash --fec-device=/dev/libvirt_lvm/verity-fec
VERITY header information for verity-hash
UUID:                   38f4d2aa-098d-4d2b-88bb-f3c558f5fc1b
Hash type:              1
Data blocks:            524288
Data block size:        4096
Hash block size:        4096
Hash algorithm:         sha256
Salt:                   46d6b600dd73f275ce34a723700093726a8d651296fbd7578bc90fb0c88b63af
Root hash:              04d66d050d9caf969fa4097bbb00cba5c4d4e9c68459b1d6eca0fc7fb0150469
$ sudo dd if=/dev/zero of=/dev/libvirt_lvm/verity-data bs=256K count=1
1+0 records in
1+0 records out
262144 bytes (262 kB, 256 KiB) copied, 0.00147542 s, 178 MB/s
$ veritysetup open /dev/libvirt_lvm/verity-data verity verity-hash 04d66d050d9caf969fa4097bbb00cba5c4d4e9c68459b1d6eca0fc7fb0150469 --fec-device=/dev/libvirt_lvm/verity-fec
veritysetup status verity
/dev/mapper/verity is active.
  type:        VERITY
  status:      verified
  hash type:   1
  data block:  4096
  hash block:  4096
  hash name:   sha256
  salt:        46d6b600dd73f275ce34a723700093726a8d651296fbd7578bc90fb0c88b63af
  data device: /dev/mapper/libvirt_lvm-verity--data
  size:        4194304 sectors
  mode:        readonly
  hash device: /dev/loop5
  hash loop:   /home/naota/verity/verity-hash
  hash offset: 8 sectors
  FEC device:  /dev/mapper/libvirt_lvm-verity--fec
  FEC offset:  0 sectors
  FEC roots:   2
$ mount /dev/mapper/verity /mnt/tmp
mount: /mnt/tmp: WARNING: device write-protected, mounted read-only.
$ dmesg
...
[10123.206004] device-mapper: verity-fec: 254:13: FEC 253952: corrected 14 errors
[10125.328194] device-mapper: verity-fec: 254:13: FEC 258048: corrected 14 errors
[10127.538251] device-mapper: verity-fec: 254:13: FEC 0: corrected 104 errors
[10129.705071] device-mapper: verity-fec: 254:13: FEC 4096: corrected 200 errors
[10129.724687] EXT4-fs (dm-17): mounted filesystem with ordered data mode. Opts: (null)
```

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。

2017年11月号

第1特集
データ分析に効く
SQL50本ノック

第2特集
なぜ、コンピュータは割り算が下手なのか!?

一般記事
・エンジニア任せにしないデータ分析の基盤作り
・ホワイトボックススイッチって何?

定価（本体1,220円+税）

2017年10月号

第1特集
Gitのキホン
GitHub、Bitbucket、GitLab、GitBucket

第2特集
脆弱性スキャナVuls入門
システムのセキュリティチェックをもっと楽に

一般記事
・ネットワークエンジニア　その技術の極め方
・身近なメールで使われている技術をネットワークコマンドで体験してみよう

定価（本体1,220円+税）

2017年9月号

第1特集
Web技術【超】入門
いま一度振り返るWebのしくみと開発方法

第2特集
tmux&Byobu
開発効率アップのターミナル改造術

一般記事
・認証を支える技術
・Ejectコマンドで遊んでみませんか?

定価（本体1,220円+税）

2017年8月号

第1特集
私も機械学習エンジニアになりたい!
先端Web企業の取り組み方は?

第2特集
エンジニアのための うけるプレゼン・すべるプレゼン
——あなたの思いを伝える技術、教えます

一般記事
・「Mastodon」旋風からわかるSNSの未来

定価（本体1,220円+税）

2017年7月号

第1特集
もっとbashを使いこなしませんか?
理論&応用でシェル力の幅を広げる

第2特集
データの抽出・加工に強くなる!
MySQL［SELECT文］集中講座

一般記事
・ハッシュ関数を使いこなしていますか?（後編）
・Jamesのセキュリティレッスン【10】
・Windows Server 2016で構築する最新ファイサーバ（後編）

定価（本体1,220円+税）

2017年6月号

第1特集
Vim、Emacs、Atom、Visual Studio Code
あなたのプログラミングを加速させるエディタ

第2特集
多用途に使いこなせ、コードが読みやすく保守しやすい
今すぐはじめるPython

一般記事
・ハッシュ関数を使いこなしていますか?（前編）
・Windows Server 2016で構築する最新ファイサーバ（前編）

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
	千代田区	書泉ブックタワー	03-5296-0051	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	丸善　丸の内本店	03-5288-8881	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	中央区	八重洲ブックセンター本店	03-3281-1811		広島市中区	丸善　広島店	082-504-6210
	渋谷区	MARUZEN＆ジュンク堂書店　渋谷店	03-5456-2111	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322

※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）、「雑誌オンライン.com」（http://www.zasshi-online.com/）、「Gihyo Digital Publishing」（https://gihyo.jp/dp）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。

December 2017

NO.74

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

複数のプログラミング言語を学ぼう

今回は、8月に開催したLLイベントについて報告します。

Learn Languages 2017 in ODC

■Learn Languages 2017 in ODC
【日時】2017年8月20日（日）12:30～18:00
【会場】日本工学院専門学校 蒲田キャンパス

■はじめに

毎年夏の恒例行事となっているLLイベントを今年も開催しました（**写真1**）。昨年まではLLをLight weight Languageの略として使っていましたが、今年からLearn Languages（複数の言語を学ぶ）の略に変更しました。また、今年は新たにスタートしたオープンデベロッパーズカンファレンス（ODC）の中で開催しました。参加者は106人でした。以下、実施したセッションについて報告します。

■基調講演「ハッカーになるためには何の言語を勉強したらいいですか？」

出演：竹迫良範（SECCON実行委員長）

今年はSECCON実行委員長である竹迫さん（**写真2**）に基調講演をお願いしました。ハッカーの起源やプログラミングの歴史などの話に続いて、ハッカーを育成するにはコンピュータやネットワークのしくみを総合的かつ実践的に学べる場所が必要で、プログラミング言語も一般的なものだけでなく機械語やビジュアル言語など幅広く知っておく必要があるという話がありました。後半は竹迫さんがこれまでに開発してきたプログラムの数々が紹介され、それを通じて竹迫さんの幅広い知識の一端を垣間見ることができました。

■関数型言語のすすめ　出演：山下伸夫（Haskell）、水島宏太（Scala）、大原常徳（Elixir）　司会：鈴木嘉平

タイトルどおり関数型言語について勉強するセッションです。前半は、Haskell、Scala、Elixirについて個別に説明がありました。Haskellは純粋関数型プログラミング言語、Scalaはオブジェクト指向と関数型が融合した言語、Elixirはネットワークサーバ構築に特化した非純粋な関数型言語と、一口に関数型言語と言ってもそれぞれに特徴があることがわかりました。後半は、四則演算の構文解析を行うプログラムを3つの言語で書いてもらい、違いを見比べました。プログラムを書くときの考え方も含めて解説していただくことにより、関数型言語への理解を深めることができました。

写真1　会場の様子（撮影：小山哲志）

写真2　基調講演を行う竹迫氏（撮影：米田真治）

■プログラム言語鑑定団

出演：鹿野桂一郎、高橋征義、高野光弘、佐々木健

司会：小山哲志

司会から相談内容が紹介され、その相談に対してどのプログラミング言語を学ぶのが良いかを鑑定人たちが回答するというセッションです（**写真3**）。相談としては次のものが用意されました。

- Web制作会社でHTMLやCSSを学んだが、Web技術だけでは将来が不安
- ゲームクリエイターになりたい
- AIや機械学習を使いこなしたい
- SIerでJavaしか触っていないので、ほかの仕事もできるようになりたい
- 学習がはかどりそうなおもしろいプログラミング言語を勉強したい
- インフラの自動化をしたい

回答は鑑定人によってさまざまでしたが、それぞれにきちんとその言語を推す理由があり、見ていてとても興味深いセッションでした。

■ライトニングトーク　司会：法林浩之

イベントの最後にライトニングトークを行いました。プログラミング言語に関する話題であれば何でもOKという条件で発表者を募り、8名の方が発表しました。演題と発表者は次のとおりです。

- 従来言語で理解する次世代言語の概念（高野光弘）
- 同じ処理を複数の言語で書いてみよう（増井敏克）
- おいら的テスティングフレームワーク比較 Java C# PHP JavaScript（sengoku）
- PHPにおけるメタプログラミングの温床（うさみけんた）
- 関係記述型言語 ラダーのすすめ（土井康正）
- now going with go（@nasa9084）
- Klassic言語の宣伝（水島宏太）
- RubyKaigi 2017のご紹介＋α（@takatayoshitake）

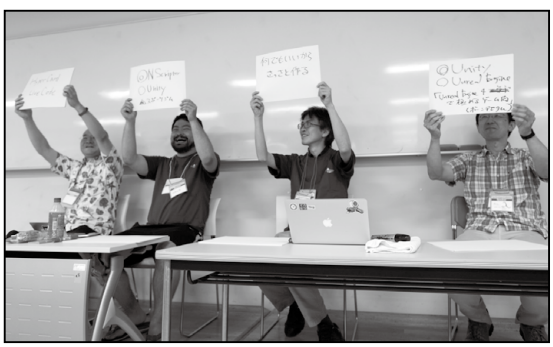


写真3 「プログラム言語鑑定団」の鑑定人のみなさん（左から佐々木氏、高野氏、高橋氏、鹿野氏）（撮影：米田真治）

■書籍の販売と展示

イベントの主旨は変わっても、プログラミング関連書籍を紹介する活動は継続することになり、今年もIT系出版各社の協力を得て書籍の販売と展示を行いました。協力いただいた出版社を紹介します。

アスキードワンゴ、インプレス、エスアイビー・アクセス、オーム社、オライリー・ジャパン、達人出版会、マイナビ出版、USP出版

■終わりに

LLイベントが始まった当初は、このイベントで扱う言語の大半はビジネスの現場で使われているとは言えない状況でした。しかし15年が経過し、RubyやPythonなどLLと呼ばれる言語たちも普通に仕事で使われるようになりました。これはLLイベントの活動の成果だと考えています。

しかし、当初の目標を達成した今でも、いろいろな言語を知っている人はまだまだ少ないようで、そこを啓蒙するのがLLイベントの次なるテーマになりそうです。

これからもプログラマのみなさんに有用な情報を提供し、プログラミングは楽しいと感じてもらえるようなイベントを続けていきたいと思います。資料などはLLイベントのWebサイト[注1]に掲載していますので、そちらもご覧ください。SD

注1） URL https://ll.jus.or.jp/2017/

あなたのスキルは社会に役立つ

2011年3月11日の東日本大震災発生の直後にHack For Japanは発足しました。今後発生しうる災害に対して過去の経験を活かすためにも、エンジニアがつながり続けるためのコミュニティとして継続しています。防災や減災、被災地の活性化や人材育成など、「エンジニアができる社会貢献」をテーマにした記事をお届けします。

第72回
過去の災害で開発されたシステムやアプリは次の備えになっているのか

●Hack For Japanスタッフ　鎌田 篤慎（かまた しげのり）　Twitter @4niruddha

2011年の東日本大震災のあの日から6年が経ち、また、その6年の間にもさまざまな災害が発生し、多くの被災者が出ました。そうした被災者の方々に向けて、我々Hack For Japanを含むさまざまなボランティア団体や開発者がシステムを開発し、提供する試みは続いています。そうしたボランティア活動を実施した各団体が集まり、互いの知見を共有する「ITx災害」というコミュニティも誕生し、過去の本連載でも紹介しました。その中で出会ったメンバーによって、災害被害を減らす目的で立ち上げられた「減災インフォ[注1]」というサイトは、被害が甚大だった熊本地震やそのほかの災害の減災につながるよう随時、情報発信を行っています。

この「減災インフォ」を運営するメンバーが、去る2017年7月にYahoo! JAPANのコワーキングスペースLODGEで「災害時のIT・情報支援のこれまで、これから。[注2]」を開催しました。過去の災害発生時に提供されたシステムやボランティア活動にはどのようなものがあり、何ができて何ができなかったのかを振り返るイベントです。

イベントでは、我々Hack For Japanが過去に開催したようなハッカソンなどから生み出された被災者向けシステムを取り上げ、効果をあげられたものとそうでなかったもの、継続されているものとそうでないものなどを定量的に分析した発表を中心に、これまでの活動を振り返り、その知見をもとにこれからの災害対策を考えたものとなりました。今回の連載では我々のような活動に取り組む者にとって、非常に有意義だったこのイベントをレポートします。

注1　https://www.gensaiinfo.com/
注2　https://gensai.connpass.com/event/57867/

全員でこれまでを振り返る

「災害時のIT・情報支援のこれまで、これから。」の開催に先立って、設立2周年を迎えた減災インフォのメンバーである小和田香さんから防災、減災に関する振り返りの挨拶が行われました（**写真1**）。

20年以上前の阪神・淡路大震災から振り返ってみれば、インターネットが普及し、個人の取り組みが世の中に発信でき、防災や減災につながる活動はしやすくなりました。一方で、行政から配信される情報はいまだにPDFなどによる情報公開で、データ活用がされにくい点はまだ解決されているとは言えない状況にあります。また、災害発生時に影響を受ける被災地区の住民の状況はというと、東日本大震災が発生した2011年時点では日本のスマートフォンの普及率は29％だったのに対し、先の熊本地震が発生した時点では72％にまで増えていました。テクノロジの発展により、年々、災害時に必要な情報を入手する手段が行き渡っています。小和田さんはそうした外部環境の変化の中で、これまでの活動

▼写真1　開催の挨拶をする小和田さん

に対してKPT[注3]を実施し、そこで挙がった防災、減災にあたる活動の中で良かった点、悪かった点を紹介してくれました。

良かった点としては、ITの活用で支援のノウハウの蓄積、改善が進んでいる点、「ITx災害」のようなコミュニティの誕生により、復興支援や防災、減災に取り組むコミュニティの横のつながりが生まれた点を挙げられました。一方で悪かった点は、地域間の横の連携が取りづらい点や、ネットなどにあふれる情報の取捨選択をどのようにしていくべきかの課題が残っている点が挙げられました。

減災インフォに参加しているメンバーは本業を持っている方々が多数であり、小和田さんは組織を大きくしていきたいという気持ちはないとのこと。災害のたびに繰り返される「もっとできたことがあったはず」という悔しい気持ちをなくすため、オールジャパンで痛みを減らすエコシステムを作ることができたのならば、情報による後方支援を行う減災インフォのような団体は解散しても良いはずという強い思いを抱えて、この2年間の取り組みを紹介されました。

近年、災害が激震化していく中、活動の振り返りをテーマとした本イベントは、過去の取り組みに対する振り返りの挨拶からスタートしました。

定量的に災害支援活動を振り返る

基調講演は、東日本大震災発生以降の支援系の活動を定量的に調査されていた国立保健医療科学院の特命上席主任研究官である奥村貴史さんからの「災害とハッカー[注4]」と題した講演です。災害発生時、支援系ボランティア活動の流れで提供された被災者やボランティア向けのシステムに対し、それらが今どういった状況にあるのかを、時間軸を加味した形でまとめた論文を中心にお話いただけました。

まず、東日本大震災のときに何が起きたかというと、被災地での被害のほかに、津波による福島第一原発の事故に伴う停電などにより、比較的に被害が少なかった首都圏も含む全国の社会生活に大きな影響がありました。災害が頻発する日本ではもともと、防災に対する意識は世界の国々と比較しても非常に高いものがあり、災害が発生した場合を想定したさまざまなシステムが用意されていました。システム開発をされている読者のみなさんも想像できるかと思いますが、こうしたシステムは事前の想定をもとに開発されるもので、想定を超えるような事態が発生した場合は動作しないといったことが起こり得ます。実際に東日本大震災が発生した際、防災や減災に向けて備えたシステムが動かないといった事例が散見されたことを紹介いただきました。

そうした想定されなかった未曾有の災害により動かないシステムの隙間を埋める形で、我々Hack For Japanを含むさまざまな開発者がその場のニーズに合わせて必要なシステムを開発していきました。例を挙げると、被災地から離れた人たちが自分の家族、知人などの生存、居場所を確認するために提供されたGoogleによるGoogle Person Finderや、避難所における救援物資を適切に配分、マッチングするためのシステム、地割れなどで通行できない道路を共有するといった、交通事情をマップ上で可視化するサービスなどです。

当時は被災地ではない地域でも、計画停電に伴う節電や福島第一原発による放射線などに気を使う場面が数多く見られました。被災地に向けた支援系のシステムのほか、そうした場面に対してもさまざまな開発者や企業が活動を行い、たとえばYahoo! JAPANからは、電力会社が供給可能な電力と実際に利用される電力を照らし、国民が日常生活を維持しつつ節電に配慮することを促す消費電力予測や、計画停電マップ、福島第一原発の影響を観測するための放射線マップなどが数多く誕生しました。

このようにハッカーが課題に応じて次々とシステムやアプリケーションを開発し、提供していった東日本大震災において、網羅性を考慮しながら、それらシステムやアプリケーションの件数を記録すると169件にもおよびます。今回の発表もそうした記録から傾向や課題を分析したものとなっています。

注3　Keep、Problem、Tryの略で、振り返りのためのフレームワーク。
注4　https://www.gensaiinfo.com/blog/2017/0708/7967

東日本大震災とそれ以降の災害

2011年の東日本大震災以降もさまざまな災害が発生しましたが、近年で最も注目を集めたのは2016年の熊本地震でした。被害の規模で見ると、熊本地震では49名の方が亡くなられました。東日本大震災とその規模は比較になりませんが、局所的な災害としては住居やシンボルである熊本城が一部倒壊するなど、先の震災を思い起こす規模の災害でした。

東日本大震災と比較すると小規模だった熊本地震ですが、先の東日本大震災での活動と同様の活動が数多く行われたことがわかります。そこで奥村さんは2011年当時にさまざまなニーズに応える形で開発されたはずのシステムやアプリケーションが、震災の状況としては類似する点もあったであろう5年後に生じた熊本地震において、どうなっていたのかを統計的に調査しました。

まず、2011年時点に開発された被災者向けのシステムやアプリケーションの件数が累計169件であるのに対し、2016年では累計で58件でした。これらのシステムやアプリケーションがどういった種類のものか、誰によって開発されたものかなどの軸で見てみると、さまざまなことが見えてきました。

東日本大震災と熊本地震の違い

東日本大震災と熊本地震、それぞれの災害に提供されたシステムやアプリケーションの種別を見てみると、電力情報系と放射線・放射能情報系のものが2016年の熊本地震ではほとんど開発・提供されることはありませんでした(**図1**)。これは考えてみると当たり前ですが、熊本地震ではそうした原子力発電所に関連する被害が発生しなかったためです。

提供されたシステムが誰の手によって開発されたものなのかを見ていくと、大きく分けて政府によるもの、企業によるもの、団体によるもの、個人の手によるものといった形で母体ごとに分けられます(**図2**)。この軸で分けて見てみると、個人の手によって開発されたシステムやアプリケーションが思いのほか多いことがわかりました。一方、東日本大震災と熊本地震の間に発生していた災害に目を向けると、その間に提供されたシステムやアプリケーションは少数ではあるものの、政府や企業、団体によるものはあったのに対し、個人によって提供されたものはありませんでした。世の中の注目が集まる非常に大きな災害が発生した場合はモチベーションに影響するのか、個人による貢献が非常に増えるという結果が見て取れます。

東日本大震災で開発されたものは今動くのか？

こうして比較すると、必然と2011年当時に被災者向けに開発されたシステムが、似た状況も多く見られた熊本地震で動いていたのかどうかが気になります。そこで、奥村さんを含む調査チームは過去の2011年に提供されたシステムやアプリケーションが現在はどうなっているのかも調査されています。

まず、2011年に開発されたシステムで「使用可能だったものと使用不可だったもの」に分けます。さ

▼図1　災害時に提供されたシステムの種別

サービス種別毎のシステム数

2016年には、影響が小さかった

ID	分類名	~2011	2011	2012-2015	2016	合計
1	電力情報系	0	50	0	0	50
2	放射線・放射能情報系	0	14	0	1	15
3	支援物資情報系	0	11	0	3	14
4	総合情報	0	11	2	8	21
5	医療系情報	0	11	0	4	15
6	スマートフォンアプリ	1	13	4	0	18
7	被災者情報系	0	9	2	1	12
8	Twitter系ツール	1	8	0	0	9
9	統合GIS	2	7	2	9	20
10	道路情報系GIS	0	6	2	2	10
11	その他情報提供系	1	8	0	14	23
12	ペット情報系	0	3	0	0	3
13	避難所情報系GIS	0	3	1	3	7
14	運行情報系	1	3	0	0	4
15	避難者受け入れ支援ツール	0	3	0	5	8
16	行政系システム	0	2	0	3	5
17	ボランティア情報系	0	3	1	5	9
18	電波時計調整ツール	0	2	0	0	2
19	首都圏情報系	0	2	0	0	2
Total		6	169	14	58	247

FacebookやGoogle mapのような一般化したサービス上に構築されたシステムが、数の上での主体に

▼図2　捕捉したシステムやサービスの提供者

	2011年以前	2011年	2012年～2015年	2016年	合計
政府	1	5	4	6	**16**
企業	2	30	6	12	**50**
団体	2	40	3	7	**52**
個人	1	41	0	15	**57**
不明	0	53	1	18	**72**
合計	**6**	**169**	**14**	**58**	**247**

２０１６年の段階ではアクセスできず、状況不明

個人が予想外に貢献している

ただし、個人が活動するのは大きな災害が起きた年に限られる

らに、「使用可能だったものでも、現在も継続して活動中のものと活動が停止されたもの」、「使用不可だったものでも、活動の停止などが宣言されたような管理下に置かれているものと放置されているもの」といった形で2つの大分類と4つの小分類で分け、それぞれ先ほどのシステムやアプリケーションを提供している母体ごとに見てみるとおもしろいことが見えてきます。

図3を見てみると、東日本大震災から5年たった今でも使えるシステムやアプリケーションは39%で、およそ6割のシステムやアプリケーションは5年後には利用できなかったということがわかってきます。さらに個人が開発したものは5年後も活動しているケースは稀(まれ)で、団体によって提供されているものと比較すると5年後の活動率は5倍もの差があることがわかります。このことから、個人がボランティアでモチベーションを維持し続けるのは困難な様子がうかがえます。

災害時のシステム開発はどうすべきか？

こうした調査をもとに、災害時に緊急に提供する必要があるシステムやアプリケーションの開発について、奥村さんらが推奨するプラクティスが提案されました。

- 第三者が提供する高速に開発するためのミドルウェア上で開発を行う
- アプリケーションの配布に関しては独自システムや独自サイトからではなく、安定したプラットフォームを用いる
- モチベーションが高い開発の初期に、提供するシステムやアプリケーションの“ここまでやる”という水準を決め、出口戦略を用意する

▼図3　東日本大震災で開発されたシステムの5年後

	使用可能		使用不可		
	活動中	活動停止	管理下	放置	合計
政府	2	2	0	1	5
企業	4	1	6	19	30
団体	23	7	3	7	40
個人	7	15	7	12	41
不明	1	4	3	45	53
合計	37	29	19	84	169
	22%	17%	11%	50%	
	39%		61%		

	活動中	放置	活動中 / 放置
団体	58%	18%	3.2
個人	17%	29%	0.6

団体によるプロダクトは、個人のものよりも5年後の活動率が5倍！

これらに配慮することで、すばやく被災者に向けてシステムを提供し、安定供給も可能となり、中途半端な状態で頓挫するのを防ぐことに近づきます。

逆に推奨されないプラクティスとして挙げられたのは、開発されたシステムやアプリケーションの生存率の観点から個人での開発を避けることが挙げられ、できる限りコミュニティを作るか、組織に所属した取り組みにしていく必要がありそうです。

ほかにもシステムの普及を決定づけるプラットフォームのシェアやユーザ需要などにも配慮することを挙げられました。これはどういうことかというと、東日本大震災の際、Google Person Finderの利用者は67万人にもおよびました。これを受けて携帯キャリア各社はそれぞれの安否確認サービスをGoogle Person Finderに統合しましたが、昨年の熊本地震においてはGoogle Person Finderの利用者はわずか1,200人でした。災害規模の違いはあれ、この利用状況の落ち込みの背景にはLINEやTwitterに代表されるSNSの普及が挙げられ、被災者とつながりのある人たちがそれぞれのSNSで安否確認を行った結果、実装されたシステムが利用されなかったことにつながっていることがうかがえます。こうした背景も配慮することで、被災地向けに開発するシステムやアプリケーションの利用につながるとのことでした。

まとめ

```
while (Japan.recovering)
    we.hack();
}
```

今回の災害インフォ主宰のイベントにおいて、過去、我々Hack For Japanが取り組んだことでうまくいったものと、いかなかったものの肌感と一致する話がたくさん挙げられました。こうして振り返ることで、次なる災害時により有効な支援が可能になると思います。ぜひ、読者のみなさんにも参考にしていただければと思います。SD

温故知新
ITむかしばなし

第72回

PC-9821 ～PC-98シリーズの起死回生はなるか～

速水 祐（はやみ ゆう） http://zob.club/ twitter @yyhayami
資料協力：小高 輝真（こだか てるまさ）

はじめに

1990年代初め、PC-9800シリーズはPC/AT互換機とWindowsに翻弄されていました。その中、起死回生を図って日本電気から21世紀の未来を見据えた改良を施したPC-9821シリーズが登場しましたが、最後の機種でさえ21世紀を迎えられませんでした。今回はこのお話です。

PC-9821とは

1980年代後半、日本電気製パソコンPC-9800シリーズは、国内の圧倒的なシェアを誇り、ほかのパソコンを寄せ付けずに快進撃を続けていました。1990年代に入ると、そこにPC/AT互換機とWindowsの波が押し寄せます。そして、1993年3月にPC-9821 Ap/As/Ae（以下、A-Mate）を発表します（**図1**）。前年に発売されていたCD-ROMを搭載したマルチメディアパソコンPC-9821[注1]のメインストリームとしてA-Mateは登場するのです。

注1） マルチメディア向けの98MULTiのCPUを高速なものに換え、Windowsを快適に動かすためのさまざまな改良を施したPC-9821シリーズ。

PC-98 vs. PC/AT互換機

1991年にWindows 3.0の日本語版が登場します。それを迎え入れるPC-9800マシンは、i80386 DX 20MHzでしかなかったのです。PC/AT互換機ではi486DXの25MHzや倍クロックの50MHzのマシンが登場しようとしていました。ユーザは、Windowsの快適とは言わずとも通常の操作ができるマシンを期待していたのですが、1992年初頭に登場したマシンPC-9801FA[注2]は、i486SX 16MHzでしかなかったのです。ユーザの落胆は大きく、ここでPC/AT互換機に移ったユーザも多かったようです。

この状況を打開するために、日本電気が打った策は、単なるCPUの強化だけに留まらず、Windowsを快適に動作させるマシンの開発だったのです。Windowsにおける高度なグラフィックとサウンド機能を実現するため、これらの機能を強化したA-Mateが生まれます。

最上位機種のPC-9821Apは、当時最高速のi486DX2 66MHzを搭載していました。

注2） 前面に補助デバイスを簡単に装着できるファイルスロットが搭載され、A-Mateシリーズに受け継がれた。

グラフィック機能は、従来のPC98の解像度640ドット×400ライン16色／4,096色中、2画面のグラフィック機能に追加して、解像度640ドット×480ライン256色／1677万色中とVGAと同じ解像度であり、PEGC[注3]と後に呼ばれる高機能なグラフィックを操作する拡張グラフアーキテクチャを有していました。サウンド機能は、6音同時出力できるFM音源部とPCM音源部を持っていました。さらに、より高解像度で高速なグラフィックボードがオプションでセットできるローカルバスを備えていました。これらの3つの機能を有することが、初期のPC-9821の必要条件と思われていました。

PC-9821の拡張機能

・グラフィック機能

A-Mateの拡張グラフアーキテクチャには、パックトピクセルモードとプレーンモードの2つのモードがあり、V-RAMは512KBを有しており、その一部

注3） PC-9821版Windows3.1のSYSTEMにグラフィックドライバPEGCV8.DRVが存在するため、PEGCと呼ぶことになったと思われる。

である32KBをメインメモリ空間に割り当てることでV-RAMへのアクセスを行います。

パックトピクセルモードは、1ドットを1byte(8bit)で表現し、直接V-RAMの特定アドレスに1byteデータを書き込むことで256色を表示できますが、大きなサイズのV-RAMへのアクセスとなるためCPUによる書き込みだけでは、描画スピードは遅くなってしまいます[注4]。

それに対して、プレーンモードは、PEGC機能により8画面のグラフィックプレーンへの同時書き込みができるだけでなく、ドット単位の移動やV-RAM上のデータとのAND/ORなどの論理演算を同時に行いながらのリード／ライトが可能でした。このPEGC機能は、従来のPC-9801の持っていたEGC機能の拡張版であり、Windowsを使わなくともMS-DOS上でこの機能を駆使すれば高度なグラフィック処理を実現できたのです。

・サウンド機能

サウンド機能におけるFM音源部は、PC-8801のサウンドボードIIに使われていた従来のYAMAHA YM2608(OPNA)というFM音源ICにより実現されています。OPNAは、4オペレータのFM6音、リズム6音、SSG3音が同時に利用できます。また、それまでのFM音源ボードであるPC-9801-26Kに搭載されているYM2203(OPN)の上位互換製品であり、従来のOPNを使用したアプリケーションからも音声を出力できます。PCM音源部は、最高44.1KHzのステレオ16bitで当時としては最高に近い性能のものでした。

・ローカルバス

PC/AT互換機のVESAローカルバス(VLバス)と同様なCPUバスと直結する32bitの外部バスがローカルバスとしてA-Mateで備えられました。ここにPC-9821A-E01(S3 86C928搭載、ビデオメモリ1MB)などのウィンドウアクセラレータボードを装着することでより高解像度で高速なWindowsの描画機能を実現できたのです。

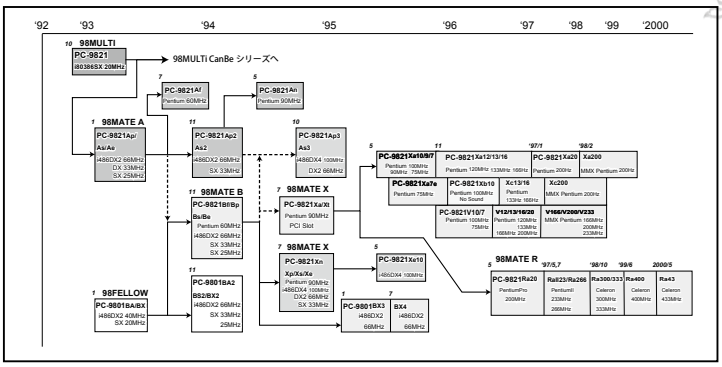
PC-9821の進展と混乱

性能的にはPC/AT互換機に追いついたA-Mateでしたが、価格ではPC/AT互換機にまったく追いつきません(Asで448,000円)。

1993年11月には、PC-9821 Bf/Bp/Bs/BeというB-Mateが登場します。B-Mateは、PC-9821と名乗っていますが内部ローカルバスにウィンドウアクセラレータを標準搭載し、PEGCやサウンドの機能はありません。ただ、価格ではBs(i486DX 33MHz)で298,000円とPC/AT互換機と競えるものになっていました。

1994年7月には、Windows 95の登場を見据えて、PEGCの機能を落とし、サウンド機能はWindows Sound Systemに準拠したPCM音源だけとなったPC-9821 Xn/Xp/Xs/Xe(X-Mate)が登場します。X-Mateは、機能を削減して値段を下げながらさまざまなマシンが1998年まで発売されることになります。1996年5月にSocket 8によりPentium Proを搭載したPC-982Ra20(R-Mate)が登場しますが、当時はWindows 95が普及してPC-9800専用のアプリケーションの使用頻度も下がってきており、PC-9800の互換性は問われなくなってきていました。1997年5月にはSlot 1にPentium IIを搭載したRaII23が発売されこのあとのR-Mateは、同じマザーボードが使われ続けます。そして、新たなPC-9821は登場せず、21世紀に登場するPC-9821は存在しなかったのです。SD

▼図1　PC-9821シリーズの年表



注4) パックトピクセルモードでもPEGCの同時書き込み機能は動作するようで、32bitデータを書き込むと32ドットのV-RAM(32byte)へ同時にアクセスできるようである。

うまくいく チーム開発のツール戦略

第11回 Atlassian製品を最適に運用するためのポイント

Author リックソフト㈱ 大野 智之

はじめに

近年、オンプレミス環境で運用していたシステムを更改と合わせてクラウドへ移行し、新たなシステムを検討する際はクラウドの採用を第一に検討する「クラウドファースト」の流れを実感しています。

MM総研の調査結果[注1]によると、40.5%の企業が「新規システムの構築方法として原則的にプライベートクラウドやパブリッククラウドを利用する」、37.6%の企業が「新規システム構築時にクラウドの活用を検討する」と回答していることからもおわかりいただけると思います。

リックソフトでは「RickCloud」というブランドでエンタープライズ向けクラウドサービスを展開しています。2014年5月のサービス開始以来、多くの法人様にご利用いただいています。

Jira(ジラ)やConfluence(コンフルエンス)などを含むAtlassian製品を重要な業務プロセス基盤や開発基盤として活用する一方で、サービスの継続性が重要になり、サーバ管理者は対策に苦慮されているのではないでしょうか。

そのようなサーバ管理者の方々へ、当社のクラウドサービスの運用方法を少しだけ紹介させていただきたいと思います。

Atlassian製品を安定運用するためのポイント

Atlassian製品を運用するポイントは、大きく分けて4つあります。

1. 安定稼働できるプラットフォームの選定
2. セキュリティ対策
3. 運用監視(問題の早期発見)
4. データ保全(バックアップ)

サーバ管理者の方々は「何を当たり前のことを」と思うかもしれませんが、長年の経験と実績に基づいた内容を少しだけお話させていただきます。

プラットフォームの選定

1つめの「安定稼働できるプラットフォーム選定」です。

Atlassian製品は、OSはWindows、Linuxのどちらでも動作し、データベース(RDBMS)もOracle、PostgreSQL、MySQL、SQL Serverなどに対応しています。

インストールの際は、Atlassian Webサイトの「Supported Platforms」(サポートプラット

▼表1 安定稼働するプラットフォームの構成

OS	次のうちいずれか※ ・CentOS 6 ・CentOS 7 ・Red Hat Enterprise Linux 6 ・Red Hat Enterprise Linux 7
Java	Oracle Java 8
Webサーバ(リバースプロキシ)	Nginx 1.12(Nginx公式サイトで配布されているRPMパッケージ)
RDBMS	PostgreSQL 9.2、9.3、9.4のいずれか(アプリケーションの対応状況によりバージョンを選択)
アプリケーション	サーバ版(tar.gzアーカイブをダウンロードし、/opt以下へ展開)

※配布パッケージの安定性とEOL(End Of Life)の長さより、CentOS／Red Hat Enterprise Linux(RHEL)を採用しています。

注1) URL https://www.m2ri.jp/news/detail.html?id=65

フォーム)[注2]で、Atlassian製品を動作させるうえで必要となるOS、データベース、Javaとそれらのバージョンを確認します。

サポートプラットフォームとこれまでの弊社における構築・運用実績などを参考に、安定稼働する構成を検討しますと**表1**のとおりになります[注3]。

なお、当社においては、それぞれのソフトウェアのバージョン(マイナーバージョン)は、社内検証結果やWeb上で公開されている不具合情報、脆弱性情報などを総合的に判断して採用しています。これまでの実績として、おおむね1～2世代前のバージョンを採用しています。

このプラットフォームに対して、当社のサービスではAmazon Web Services(AWS)を採用しました。その理由は、ご存じのとおりAWSはクラウド業界のリーダー的存在で、世界のクラウド市場で34%のシェアを占めているからです[注4]。また、AWSは幅広い分野で利用されており、とくに最近では三菱東京UFJ銀行が勘定系システムをAWSへ移行するというニュース[注5]が記憶に新しいと思います。

AWSインスタンスモデルの目安

AWS上で運用するために必要なスペックの目安については、**表2**を目安にしてください。この目安は、利用用途や利用アドオンによって大きく変動します。パフォーマンスの状況を見ながらサーバスペックを増強してください。

AWSではなくオンプレミスの場合は、当社のホームページ(ブログ)にサイジング情報が掲載されていますのでご参照ください。

▼表2　AWS上で運用するための必要なスペックの目安

ユーザ数の目安	インスタンスタイプ	vCPU	Mem(GB)	ストレージ(EBS)
10ユーザ	t2.medium	2	4	ルートボリューム(/dev/xvda1) ・EBSタイプ：汎用SSD(gp2) ・容量：30GB ・ファイルシステム：xfs データボリューム(/dev/xvdb)※ ・EBSタイプ：汎用SSD(gp2) ・容量：100GB ・ファイルシステム：ext4
11ユーザ～25ユーザ	c4.large	2	3.75	
26ユーザ～50ユーザ	c4.large	2	3.75	
51ユーザ～100ユーザ	c4.large	2	3.75	
101ユーザ～250ユーザ	c4.large	2	3.75	
251ユーザ～500ユーザ	c4.xlarge	4	7.5	
501ユーザ～1,000ユーザ	c4.2xlarge	8	15	
1,001ユーザ～2,000ユーザ	c4.4xlarge	16	30	
2,001ユーザ～4,000ユーザ	c4.8xlarge	36	60	
4,001ユーザ以上	Data Centerライセンスを使った構成			

※データボリュームは、Amazon Elastic Block Store(EBS)のElastic Volumesに対応できるよう、fdiskでパーティショニングを行わず、/dev/xvdbに対して直接フォーマット(mkfs.ext4)します。

アプリケーションに対するインスタンス数

Atlassian製品を運用する際は、次の理由により、1つのアプリケーションに対して1つのインスタンスで動作させることを推奨します。

- アプリケーションの共倒れを防げる(インスタンス障害や特定のアプリケーションの不具合による他アプリケーションへの影響を回避)
- メンテナンス時の不必要な停止を回避できる(メンテナンス時の停止対象インスタンスが、必要なインスタンスのみに限定)
- アプリケーションの不具合時の対象を特定しやすい(不具合が生じても障害範囲は特定のインスタンスに限定)
- パッケージの依存関係の問題を回避できる

注2) JIRAの例：https://confluence.atlassian.com/adminjiraserver072/supported-platforms-828787550.html

注3) 対応しているバージョンについては、対象ソフトウェアのサポートプラットフォーム情報をご確認いただくか、当社にお問い合わせください。

注4) URL https://www.srgresearch.com/articles/leading-cloud-providers-continue-run-away-market

注5) URL http://itpro.nikkeibp.co.jp/atcl/column/14/346926/022400847/

セキュリティ対策

続いて「セキュリティ対策」についてです。まず、AWSなどのパブリッククラウド上でAtlassian製品を利用するには、次の方法が挙げられます。

- プライベートネットワークから利用する（Direct ConnectやVPNなどのプライベート接続経由でアプリケーションを利用）
- パブリックネットワークから利用する（インターネット経由でアプリケーションを利用）

これからパブリッククラウド上でAtlassian製品をお使いいただく場合、要求されるセキュリティによりますが、パブリックネットワークでの利用をお勧めします。

パブリックネットワークで運用するメリットとして、次の点が挙げられます。

- インターネット接続可能な環境があればアプリケーションを利用できる（専用線やVPNなど、特別なネットワークや設備を必要としない）
- アプリケーションの利用拠点の増減に対応しやすく、利用開始／利用終了が短時間で行える（セキュリティグループまたはリバースプロキシ上での、接続許可IPアドレスの追加／削除のみでアクセス制限が可能）
- 企業によっては、社内LANやプライベートネットワークのセキュリティポリシーを受けない（たとえば、協力会社にも同じアプリケーションを提供しやすい）

なお、パブリックネットワークでの運用に関しては、サーバへの不正侵入や、アプリケーションの改ざん、盗聴の可能性など、セキュリティ面のリスクを心配される方もいると思います。パブリックネットワークで運用する以上は確かにこのようなリスクが多少なりともあるかもしれませんが、通信経路の暗号化やアクセス制限を適切に行い、IDS／IPSで不正なトラフィックを検知／遮断することで、それらのリスクを可能な限り回避できます。また、脆弱性への適切な対応や、各種設定の定期的な見なおしなど、適切なメンテナンスを継続的に行うことで、よりセキュリティを高められます。これらを行うことで、オンプレミスやプライベートネットワークと比べて遜色のないセキュリティを確保できます。

運用監視（問題の早期発見）

続いては「運用監視」についてです。当社では統合監視ソフトウェア（ZABBIX）によるアプリケーション環境や運用基盤のリソースを監視しています。ログイン画面のレスポンス、プロセス監視などのサービス監視や、致命的なエラーのためのログ監視のほか、CPU使用率、ディスク使用率、メモリ使用率、ネットワーク使用率などを監視しています。

代表的な監視項目をまとめると**表3**の内容となります。

New RelicやDatadogなども組み合わせて利用する場合もあります。

このような監視状況をモニタで常に表示し、異常があったときにすぐに気づけるようにしています。

データ保全（バックアップ）

続いては「データ保全（バックアップ）」についてです。

Atlassian製品に限りませんが、AWSでアプリケーションを運用する際は、AWSの機能を最大限に活用することで運用を大幅に効率化できます。

バックアップに関しては、AWSにはEBSスナップショットというボリューム単位でバックアップする機能があります。EBSスナップショットを利用することでインスタンスを停止

することなく、効率的かつ安全、低コストでバックアップができます。また、EBSスナップショットは世代管理に対応していますので、これも活用するとよいでしょう。

併せて、万が一に備え、AWSの機能に依存しない汎用的な手順によるバックアップの方法も用意しておくとよいでしょう。

EBSスナップショットと合わせて、ファイルベースのバックアップ(データベースのDUMP取得とrsyncによる別ボリュームへのバックアップ)も行うとデータの保全性をより高められます。たとえば、AWS側のサービス障害や実行スクリプトの不具合などによってEBSスナップショットが正常に行われていなかった場合、ファイルベースのバックアップが保険になります。

なお、当社の場合は、EBSスナップショットとファイルベースのバックアップの両方を行い、EBSスナップショットは3世代のバックアップを取得しています。

その他にも、弊社では行っていませんが、DRBDを用いてボリュームごとにレプリケーションする方法もあります。たとえば、AWSと西日本にリージョンの存在する他IaaS間でレプリケーションを行うと、災害対策(DR)として有効です。

最後に

このポイントをふまえたうえで、当社のクラウドサービスは、日本では数少ない「ISO27001(ISMS)とISO27017(ISMSクラウドセキュリティ認証)」を取得できました。

ISO27017とは、その適用範囲内に含まれるクラウドサービスの提供もしくは利用に関して、ISO/IEC 27017:2015のガイドラインに規定されるクラウドサービスの情報セキュリティ管理を満たしている組織へ与えられる認証です。JIS Q 27001:2014(ISO/IEC 27001:2013)認証を前提としており、取得した組織には認証ロゴが付与されます。

利用するユーザや格納するデータが増加し、企業の業務システムのサービス継続が重要になってきたときには、この記事を参考に堅牢なサーバインフラで運用することをお勧めします。

不安や心配がある場合は、当社の「RickCloud」サービスに丸ごとお任せいただければと思います。SD

※本連載の過去記事は技術評論社のWebサイト「gihyo.jp」でもご覧になれます。

http://gihyo.jp/ad/01/atlassian

▼表3　代表的な運用監視項目

監視項目	通知
ログイン画面のレスポンス	・ログイン画面のURLへアクセスした際に、40x系や50x系のHTTPステータスコードがレスポンスとして返った場合 ・ログイン画面のURLへアクセスした際に、URLに接続できない場合
プロセス／サービス監視	・アプリケーションのJava Management Extensions (JMX)へ接続できない場合
DBMS(PostgreSQL)監視	・DBへ接続できない場合 ・スロークエリ数が閾値(いきち)を超えた場合 ・DBコネクション数が閾値を超えた場合
MTA(Postfix)監視	・SMTPへ接続できない場合 ・メールキューに滞留するメッセージ数が閾値を超えた場合
CPU使用率	・CPU使用率が一定時間内に継続して閾値を超えた場合(アプリケーションのCPU使用率が高い場合、ディスクI/Oやネットワークl/0の高負荷によってCPU使用率が高くなった場合など)
ディスク使用率	・ディスク使用率が閾値を超えた場合(ディスクの空き容量が少なくなった場合)
メモリ使用率	・インスタンスの割当メモリの使用率が閾値を超えた場合(メモリの空きが少なくなった場合) ・JAVAヒープの使用率が閾値を超えた場合(JAVAヒープの空きが少なくなった場合) ・SWAPの使用率が閾値を超えた場合(SWAPが少なくなった場合)
ネットワーク使用率	・ネットワーク使用率が一定時間内に継続して閾値を超えた場合(高トラフィックの状態が継続する場合など)

NETGEAR ReadyNAS徹底運用

～大切なデータの保護に耐えうるか実力を試す！～

第4回 SSH接続でReadyNASの構造をチェック

今回から3回にわたってReadyNASの中身を読み解いていきます。今回はNASとして動作するためにどのような構造になっているか基本的な中身を追っていきます。

Author 後藤 大地（ごとう だいち）

検証機 NETGEAR ReadyNAS 628X

検証に使ったプロダクトは引き続きNETGEAR ReadyNAS 628Xです。8つの3.5インチSATA HDDスロットと2つの10GbE、2つの1GbEを備えています。10TB HDDを8スロットそろえた場合で、RAID0構成なら73TBほど、RAID6なら55TBほどのストレージとして利用できます。検証機には「Seagate ST6000NM0024 - 6.00 TB/5.46TiB SATA HDD」が8機装備済みです（**写真1**）。

ReadyNASはNASとして開発されたプロダクトですが、NAS以外の用途にも利用できる汎用性も持ち合わせています。いくつかの機能を有効にして利用効率の高いサーバとして運用するというのもおもしろい使い方です。次回以降ではそうしたNASを超える使い方をする方法まで取り上げます。

▼写真1 8つのスロットすべてにHDDを装備済み

ReadyNASが提供している基本的な機能はこれまでの連載をご覧ください :)

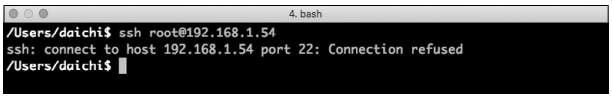
ReadyNASにsshでログイン

まず、セットアップしたReadyNASにpingしてみましょう。本体フロントのLEDパネルに表示されるIPアドレスに対してpingを実行してみると、問題なく通りますね。

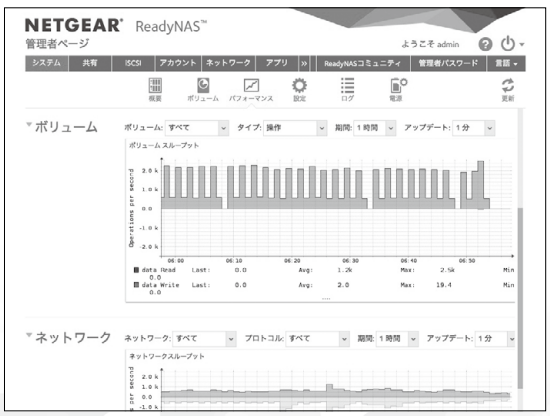
今度はsshでログインしようとしてみますが、**図1**のように22番ポートが空いていないとして蹴られます。

ReadyNASの操作はWebブラウザから可能です。このあたりはこれまでの連載をご覧いただきたいのですが、フロントのLEDパネルに表示されるIPアドレスにHTTPでアクセスすると、**図2**のようにダッシュボード的なインターフェー

▼図1 ReadyNASはデフォルトではsshは通らない



▼図2 ReadyNAS管理者ページ



スである管理者ページにアクセスできます。

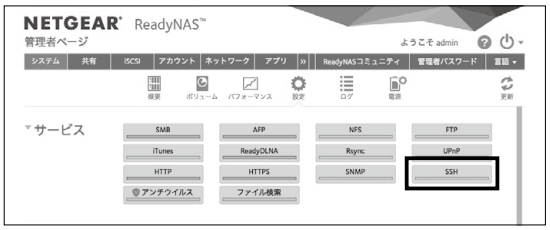
［システム］→［設定］→［サービス］から利用するサービスを管理できますが、sshは最初は無効になっています(**図3**)。SSHをクリックするとこのサービスを有効にできます。

有効にすると、「警告：NETGEARはroot SSHアクセスを有効にした場合のサポートはしません。」という警告が出ます。rootでログインして調べるような緩い設定は、あくまでも検証段階のときだけにしてください。

sshサービスを有効にすれば、**図4**のようにsshでログインできることを確認できます。パスワードは管理者ページのパスワードと同じで、アカウント名がadminではなくrootになっています[注1]。

ここまで来てしまえば、ほとんどのLinuxサーバ管理者にとってあとはもう中身を調べたい放題です。さっそく中身を見ていきましょう。

▼**図3**　ReadyNAS管理者ページ：システム→設定→サービス



▼**図4**　sshでReadyNASへのログインを確認

```
/Users/daichi$ ssh root@192.168.1.54
The authenticity of host '192.168.1.54 (192.168.1.54)' can't be established.
ECDSA key fingerprint is SHA256:x9yqa03WnwRC3bWhisdL6URtCIbm7dSyOdRr/qSN41I.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.54' (ECDSA) to the list of known hosts.
root@192.168.1.54's password:

Welcome to ReadyNASOS 6.8.1

Last login: Tue Oct 10 03:08:24 2017 from 192.168.1.106
root@nas-43-6E-44:~#
```

中から見守るReadyNAS

検証に利用したReadyNAS 628Xのおもなスペックは**表1**のとおりです。まずOSからどのように見えているのかを確認しましょう。

まずOSを調べてみます。次のようにして/etc/の設定ファイルを調べたり、unameコマンドを使ったりすると、ReadyNAS OSのバージョンが6.8.1であることや、中身のベースはDebian GNU/Linux 8で、カーネルは4.4.88のNASサーバであることがわかります。

```
# cat /etc/issue
# cat /etc/os-release
# uname -a
# cat /proc/version
```

次のようにしてプロセッサとメモリを調べると、スペックどおりIntel Xeon D-1521(2.40GHz)が見えていることがわかります。スレッド(論理コア)も8個見つかります。メモリも仕様書どおりに8GBです。

```
# cat /proc/cpuinfo
# cat /proc/meminfo
```

ifconfigコマンドを使えば、ネットワークインターフェースは4つ認識されており、検証環境での出力だとeth0とeth1が10GbE、eth2とeth3が1GbEになっていることがわかります。動画データを保存するようなケースだと10GbEネットワークが必要になるので、このようにポートとして10GbEが用意されているのは重要なポイントです。

ソフトウェアRAIDを調べてみましょう。管理者ページから調べると8スロットのRAID6で構築されていることがわかります(**図5**)。

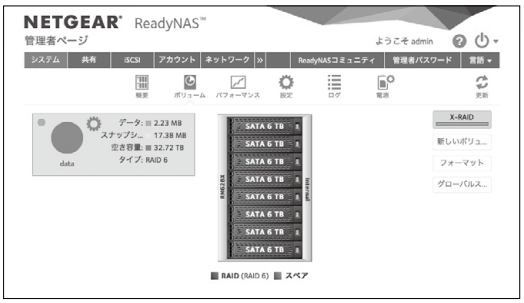
構成から考えるとmdadmを使うでしょうから/proc/mdstatを調べます(**図6**)。/proc/mdstatを見れば、ディスクとしてはsdaから

▼**表1**　ReadyNAS 628Xのおもなスペック

項目	内容
CPU	Intel Xeon D1521 Quad Core Hyper Thread 2.4GHz
メモリ	8GB DDR4 ECC
LAN	10GbE×2／1GbE×2
HDD	Seagate 6TB(ST6000NM0024)×8 SATA

注1) 運用するときにはrootでのアクセスは無効にしておきましょう。

▼図5　管理者ページで8スロットのRAID6構成であることがわかる



sdhまでの8台が認識され、それぞれsda1、sda2、sda3のように3つのパーティションが区切られていること、それらを使ってRAID1のmd0と、RAID6のmd1とmd127が構築されていることを確認できます。

`dmesg | grep 'sd[a-z]'`とコマンドを入力して認識されているディスクを調べると、Seagateのディスクが6.00TB/5.46TiBで認識されていることもわかります。論理ブロックは512バイトで物理は4KBです。当然ですがスペックどおりの認識です。

dfでマウント状況をチェックするとRAID1のmd0にはベースシステムが配置され、RAID6のmd127にはストレージとしてメインの領域が割り当てられていることが確認できます(**図7**)。md0がシステム、md1がスワップ、md127がストレージという割り当てです。

スワップを確認すると、スワップとしてRAID6で構築されたmd1が使われていることがわかります(**図8**)。

mountコマンドでマウント先とファイルシステムおよびマウント時のオプションを確認すると、md0およびmd127の双方がBtrfsになっていることもわかります。

先ほどのdfの出力で/devがudevであることもわかりました。つまり必要なデバイスしか/devには生えてきません。試しに次のコマンドのようにして/dev/の下を見ると、sdaからsdhまでが存在することを確認できます。8台のSATA HDDが認識されていることがわかります。

```
# ls -l /dev/ | grep 'sd[a-z]'
```

このように、ReadyNASの構造はmdadmを使ったシンプルなソフトウェアRAID構造になっていることがわかります。構成を変えた場合に切り分け方がどのように変わるのか気にな

▼図6　/proc/mdstatで基本的なソフトウェアRAID構成を確認

```
root@nas-43-6E-44:~# cat /proc/mdstat
Personalities : [raid0] [raid1] [raid10] [raid6] [raid5] [raid4]
md127 : active raid6 sda3[0] sdh3[7] sdg3[6] sdf3[5] sde3[4] sdd3[3] sdc3[2] sdb3[1]
      35134036992 blocks super 1.2 level 6, 64k chunk, algorithm 2 [8/8] [UUUUUUUU]

md1 : active raid6 sda2[0] sdh2[7] sdg2[6] sdf2[5] sde2[4] sdd2[3] sdc2[2] sdb2[1]
      3142272 blocks super 1.2 level 6, 64k chunk, algorithm 2 [8/8] [UUUUUUUU]

md0 : active raid1 sda1[0] sdh1[7] sdg1[6] sdf1[5] sde1[4] sdd1[3] sdc1[2] sdb1[1]
      4190208 blocks super 1.2 [8/8] [UUUUUUUU]

unused devices: <none>
root@nas-43-6E-44:~#
```

▼図7　md0とmd127の割り当て先をdfで確認

```
root@nas-43-6E-44:~# df
Filesystem       1K-blocks    Used   Available Use% Mounted on
udev                 10240       4       10236   1% /dev
/dev/md0           4190208  476796     3313636  13% /
tmpfs              4028928       0     4028928   0% /dev/shm
tmpfs              4028928    6204     4022724   1% /run
tmpfs              2014464    1004     2013460   1% /run/lock
tmpfs              4028928       0     4028928   0% /sys/fs/cgroup
/dev/md127     35134036992   20140 35131920724   1% /data
/dev/md127     35134036992   20140 35131920724   1% /home
/dev/md127     35134036992   20140 35131920724   1% /apps
root@nas-43-6E-44:~#
```

▼図8　スワップにはmd1が使われている

```
root@nas-43-6E-44:~# cat /proc/swaps
Filename  Type       Size    Used    Priority
/dev/md1  partition  3142268 0       -1
root@nas-43-6E-44:~#
```

るところですので、そのあたりは次回に取り上げようと思います。

サービスもシンプル

今度は動作しているデーモンを調べてみましょう。**ps x | grep -v '\['**で出力されたプロセスは次のようになります。NASとして機能するために必要になるデーモンが一通り動作していることを確認できます。HTTPサーバとしてはApacheが使われており、管理用にはsystemdが動作していることも確認できます。

```
/sbin/init
/lib/systemd/systemd-journald
/lib/systemd/systemd-udevd
/usr/sbin/mdcsrepaird -v
/usr/sbin/cron -f
/usr/sbin/connmand -n -r
/usr/sbin/raidard -S
/sbin/mdadm -F -s -y -p /frontview/bin/md_event
/usr/sbin/wsdd2
/lib/systemd/systemd-logind
/sbin/agetty --noclear tty1 linux
/sbin/agetty --keep-baud 115200 38400 9600 ➥
ttyS0 vt220
/usr/sbin/nmbd
/usr/sbin/minidlnad -S
/usr/sbin/netatalk
/usr/sbin/smbd
/usr/sbin/afpd -d -F /etc/netatalk/afp.conf
/usr/sbin/cnid_metad -d -F /etc/netatalk/afp.conf
/usr/sbin/apache2 -k start
/usr/sbin/readynasd -v 3 -t
apache_log
/usr/sbin/apache2 -k start
/frontview/bin/fvbackup-q
/usr/sbin/minissdpd
ps x
sshd: root@pts/0
-bash
```

ps x | grep '\['でカーネルスレッドを見てみると、Intel 10GbEドライバが動作していることがわかりますし、RAIDおよびBtrfs関連の機能が動作していることもわかります。

これらサービスの制御もシンプルです。systemdのしくみで管理されているようで、**systemctl list-unit-files -t service**コマンドを使えば用意されているサービスがわかります。

実際、最初のsshを有効にしたのでsshのサービスがenabledになっていることも先のコマンド出力で確認できます。デフォルトの状態だとこの部分はdisabledになっていますので、systemdで管理されていることがわかります。管理者ページからサービスの有効無効を変更した場合にどのようにサービスが変わるかも興味深いところです。このあたりも次回以降に調べていきましょう。

シンプルで優しいNETGEAR ReadyNAS

エンタープライズ向けからコンシューマ向けまで、最近のNASプロダクトの中身の多くはLinuxや*BSDです。機能はNASに特化していますが、中身は通常のUNIX系サーバとあまり変わりがありません。プロダクトによってはカスタムカーネルを使っているものもありますし、エンタープライズ向けになるとハイパーバイザーのレイヤが入っていて仮想化機能を持っていたり、ファイルシステムやサービスが機能拡張されたものだったりします。そういった違いはありますが、基本的な構成はいわゆるUNIX系サーバがわかっていれば理解できるものです。

ReadyNASの内部はNASプロダクトとしてはシンプルな構造になっており、いわゆるLinuxベースのソフトウェアRAIDサーバということになります。Linuxサーバの管理に携わっているなら、ReadyNASにログインして行う作業はほかのLinuxサーバとあまり違いを感じないのではないかと思います。次回はNASとしての構成やサービスの制御が実際にOS側でどのように見えるのかを追っていきます。SD

Hardware

NECプラットフォームズ、Wi-Fiホームルータ「Aterm WG1800HP3」発売

NECプラットフォームズ㈱は10月3日、IEEE802.11ac対応のWi-Fiホームルータ「Aterm WG1800HP3」を発売した。価格はオープン。

本製品は、3本のアンテナを利用する3ストリーム対応により、最大1,300Mbps(5GHz帯)・450Mbps(2.4GHz)の高速通信が可能となっている(ともに理論値)。そのほかの特徴は次のとおり。

・「ビームフォーミング」機能
特定の通信相手に的を絞って電波を送る技術。本機能に対応したスマートフォンやWi-Fi端末(子機)を自動で検出し、電波を集中的に照射するので、つながりやすさと実効速度の向上が見込める

・「見えて安心ネット」機能
専用アプリ「Atermスマートリモコン」を使い、ルータに接続している端末を一覧表示して管理できる。それぞれの端末に対して接続の許可/拒否や接続制限など、詳細な設定も行える

・「Wi-Fi設定引越し」機能
買い替え前のルータから、ボタン1つで設定を完了できるWPS機能を使って、各端末の設定情報を引き継ぐことができる。他社製ルータでも、WPS対応であれば使用可能

▲Aterm WG1800HP3

CONTACT
NECプラットフォームズ㈱ URL https://www.necplatforms.co.jp

Software

Mozilla、パフォーマンスの量子的飛躍「Firefox Quantum」公開

Webブラウザ「Firefox」の最新版である、「Firefox Quantum」が10月31日発表された(本記事執筆時点ではベータ版で、11月14日に公開予定)。リリースに合わせ、Mozillaのジェフ・グリフィス氏(コアブラウザ担当プロダクト管理ディレクター)とシンディ・シャン氏(コアブラウザ担当シニアプロダクトマネージャー)も来日し、記者発表会で新機能の概要説明と動作デモンストレーションを行った。

今回のリリースは、あらゆるユーザに対してパフォーマンスを向上させることを主眼としており、その特徴を挙げると、「2倍の高速化」「新しいエンジンとFluid UI」「トラッキング保護」の3つになる。

まず高速化については、AppleやGoogleが使用しているベンチマークツール(SPEEDOMETER2.0)を使用し、レンダリングとJavaScript実行エンジンのテストを行っている。ブラウザのアーキテクチャについてWebページのロード時に処理を少なくすることで、メモリの消費を抑える工夫を実施した。その結果、従来よりも2倍の速度を実現した。今後6～12ヵ月の間にさらに高速化を図る予定とのこと。

また、ルック&フィールの向上も実現した。この新しいUI(プロジェクト名Photon)では、同時にたくさんのタブを開くことができ、Web上での情報編集作業に便利になる機能も追加している。さらに、ユーザのプライバシーのためのトラッキング保護機能も、従来ではプライベートブラウジングモードから別のウィンドウを開かねばならなかったが、ノーマルモードでも可能になった。これは広告ネットワークからの保護機能だが、同時にパフォーマンスの向上も実現した。iPhone/Android版についても同様の機能を提供する予定だ。

▲ジェフ・グリフィス氏

CONTACT
Mozilla URL https://www.mozilla.org/ja/

Software

Adobe Systems、新アプリケーションを含む次世代Creative Cloudを発表

Adobe Systemsは10月19日、クリエイティビティに関する世界最大規模のカンファレンスである「Adobe MAX」において、新製品「Adobe XD CC」、「Adobe Dimension CC」、「Adobe Chratacter Animator CC」と、新たなオンライン連携可能な「Adobe Lightroom CC」を発表した。それぞれの新製品の特徴は以下のとおり（※見出し後のカッコ内はAdobeのアイコン名）。

Adobe XD CC（Xd）

モバイルアプリやWebサイトなどのデザインおよびプロトタイピングのためのオールインワンソリューション。UI/UXデザインのワークフローを一新。デザイン、プロトタイプ、共有のすべてに対応する。

Adobe Dimension CC（Dn）

3Dレンダリングアプリケーション。2Dの作業と同じ使いやすさとシンプルさで、3Dのパワーと柔軟性が提供される。これによって、写真のようにリアルな3D画像を素早く作成し、簡単な操作で繰り返し作業を行えるため、ブランディングデザインやパッケージデザインを簡便に作成できる。

Adobe Character Animator CC（Ch）

顔認識で2Dアニメーションを自動生成できる、2Dアニメーションツール。Photoshop CCやIllustrator CCで作成した静止画像に、生き生きとした動きを与えられる。「Adobe Sensei」と呼ばれる人工知能（AI）と機械学習の統合フレームワークが利用でき、口の動きと音声の正確なマッチングを実現するリップシンク機能が使える。

Adobe Photoshop Lightroom CC（Lr）

以前のPhotoshop Lightroom CCは、「Photoshop Lightroom Classic CC」と名称変更（と機能強化／最適化）された。そして新たに、オンライン連携可能なクラウドベースのアプリケーションと、場所を問わず写真の編集、整理、保管、共有を可能にするサービスを一体化した製品がPhotoshop Lightroom CCである。

Photoshop Lightroom CCは、直観的なユーザーインターフェース、フル解像度で画像編集ができ、変更はすべてのモバイル、デスクトップ、Webに自動的に同期される。写真の整理が容易で、自動的に検索用のキーワードが付与される。また、ソーシャルメディアでの写真の共有も簡単に行える。画像データは、Adobeのクラウド上で管理され、検索用のキーワードなどの画像認識にはAdobe Senseiが利用できる。

また、同社Creative Cloud製品の数々の機能アップデートを発表した。代表的なものは以下のとおり。

Adobe Photoshop CC（Ps）

写真のサポート機能、アドビ ラーニングやチュートリアルの大幅な改善と、全体的な性能向上が図られた。

Adobe Illustrator CC（Ai）

複数のワークフロー、ユーザエクスペリエンス、およびパフォーマンスが向上された。

Adobe InDesign CC（Id）

効率的な作業を実現する機能が追加され、複数ページのレイアウト作成がさらに容易になった。

Adobe Premiere Pro CC（Pr）

高度なコラボレーション機能、最新の360度VRの没入型ワークフローとレスポンシブモーショングラフィックス コントロールが加わった。

Adobe After Effects CC（Ae）

データ駆動型グラフィックスと高品質なVRおよび3D動画制作を簡素化することで、モーショングラフィックスの制作作業が効率化された。また、GPUパフォーマンスが強化され、作業をより迅速に行える。

Adobe Typekit（Tk）

新たに追加されたTypekitのビジュアル検索で、フォントが含まれる写真から類似フォントを探せる。

各デスクトップアプリのアップデートは、Creative Cloudメンバーは既に無償でダウンロードできる。各プランは、それぞれ月あたり税別で、Creative Cloudのすべてのアプリケーションが使えるコンプリートプランが4,980円、単体のアプリケーションだけの単体プランが2,180円、新しいLightroom CCと1TBのクラウドストレージが使えるプランが980円、新しいLightroom CCとLightroom Clasic CCとPhotoshop CCに20GBのクラウドストレージが使えるフォトプランが980円、同プランを1TBのクラウドストレージにしたものが1,980円となっている。詳しくは同社サイトで確認のこと。

CONTACT
アドビ システムズ㈱ URL https://www.adobe.com/jp/

Report

エンタープライズからエンドユーザまで取り込む求心力、「Dell EMC Forum 2017 Tokyo」基調講演レポート

10月26日、ザ・プリンスパークタワー東京（東京都港区）にて、Tokyo Dell EMC Forum 2017が開催された。

会場来場者数2,085名と昨年よりも約500名ほど多く、会場規模もそれに合わせ大きなものになった。昨年はDellとEMCの統合にどのような顧客メリットがあるのか、それに「デジタルトランスフォーメーションの必要性」と関連付けてのプレゼンテーションが多かったが、今年はより一段階進めて、「デジタルトランスフォーメーションの実現（Realize）」がテーマである。

デル㈱代表取締役社長平出智行氏とEMCジャパン㈱代表取締役社長大塚俊彦氏の開会の挨拶から始まった基調講演をレポートする。

○デジタルトランスフォーメーションの実現

米国Dell EMC サービスおよびIT担当プレジデント ハワード・エライアス氏による戦略発表は、デジタルトランスフォーメーションの実現（Realize）をするために何が必要なのか、自社の製品群と絡めてプレゼンテーションするものだった。IT導入によりUberやAirBnBのような企業がここまで成長するとは誰も予想できなかったことを述べ、ファミリー企業の力を集結し顧客システムのデジタルトランスフォーメーションの現実化への道筋を示した。たとえば自律運転型の自動車では、運転中に路上に鹿が飛び出てきた場合、クラウドにデータを送って分析するのではなく、リアルタイムでエッジ側で分析する。これを最終的にクラウドで分析し、機械学習を経てアルゴリズムを作り、それをフィードバックして好循環を生み出す。これを日→時間→分→秒でアップデートして提供するのがデジタルトランスフォーメーションの一例であると説明した。またIoTから生まれるデータ量は桁違いに大きいことにも触れ、これまでは石油や石炭が産業を支えていたが、ITが普及した現在はデータこそがビジネスの燃料であり、産業を生み出す力になるという変化が起きていることを示した。

○デジタルトランスフォーメーションによる新価値の創造

ゲストスピーカーとして招かれたコニカミノルタ㈱代表取締役兼CEO山名昌衛氏は、同社のWorkplace Hubをデジタルトランスフォーメーションの例として挙げた。これは1m四方程度の大きさの機器が、IoTのサーバとなるもの。接続されたカメラから動画情報を取り込み、会議室の使用状況を確認したり、工場の現場レイアウトでの人間の動きを分析したりできる。そしてエッジからクラウドにつながり、ディープラーニングを応用した意思決定支援を行う。これが生産性の向上に大きくつながるという。IoTの時代では、顧客メリットが何であるのか、価値をどう突き詰めていくのかが問題であり、製造業の強みを生かしアジャイルに対応するスピード感が大事であると示した。そしてまだまだデジタルトランスフォーメーションを実施する要素がたくさんあると説明した。

○働き方改革の未来～100人100通りの働き方

同じくゲストスピーカーのサイボウズ㈱代表取締役社長青野慶久氏は、同社での働き方改革を実践してきた経験を語った。同社はもともとは高い離職率に悩むいわゆるブラック企業だったという。そこで人事の方針を「100人いれば、100通りの人事制度があってよい」と変更したことで離職率は大きく改善し、売り上高も上昇した。その他に社員の市場価値から給与を決めることや、オフィスの再設計による環境改善、さまざまな施策をしてきたが、最も大事なことは風土改善であったという。率先垂範（そっせんすいはん）であるとして、自ら育児休業や保育園へ子供を迎えに行くための16時退社を実行、社内の雰囲気を変えた。さらに社員をチームで働く体制にした。その結果、以前より大きな価値を生みだせるようになったという。最後に、働き方改革が世間では経営者にとって残業削減としか理解されないことに対し、情報発信をしつづける重要性を示した。

○ひとり情シスの薄い本

会場で目を引いたのは「ひとり情シス」（非売品）の小冊子である。展示会場入り口近くで配布された、デル㈱執行役員 広域営業統括本部長清水博氏によるもの。「この調査結果による、たった一人だけの情報システム要員が全体の14%を占めており、しかも専任担当者が一人もいない企業が13%に上がっており、双方を合わせると27%を占めました。約3分の1の会社が、情報システム要員が1人未満というものでした（同書より引用）」という衝撃的なアンケート結果をもとに制作された。実際問題、ITの現場は「ひとり情シス」に支えられていることが多く、ここにスポットライトをあてる試みは非常に価値があると感じた。

▲ひとり情シス（同イベント内「ひとり情シス大会議」配布本）

CONTACT
Dell EMC Forum URL http://www.dellemc.com/jp/f1

12月12日、「Ruby biz Grand prix」表彰式開催

「Ruby biz Grand prix」はプログラミング言語Rubyを使った製品・サービスのビジネス事例を表彰する、応募制のコンテスト。2015年度の大賞はトレジャーデータ㈱の「Treasure Data Service/Fluentd/embulk」と㈱ユビレジの「ユビレジ」、2016年度の大賞は㈱Misocaの「Misoca」とラクスル㈱の「ハコベル」だった。今年2017年度の授賞式は、12月12日に帝国ホテル（東京都千代田）にて開かれる。当日は関係者以外の聴講はできないが、本誌2月号で表彰式のレポートを行いたい。2017年度のエントリ企業は次の29社。

㈱IBJ／アクトインディ㈱／㈱あしたのチーム／アルクテラス㈱／合同会社esa／㈱キャスター／㈱キューブス／キラメックス㈱／㈱クレオフーガ／Crevo㈱／コインチェック㈱／JapanTaxi㈱／㈱SKIYAKI／㈱SKIYAKI／㈱SmartHR／ソニーネットワークコミュニケーションズ㈱／テモナ㈱／㈱トラスト&グロース／㈱トレタ／㈱パソナテック／ピクシブ㈱／㈱Fablic／㈱フロムスクラッチ／㈱まちづくり三鷹／㈱マンションマーケット／㈱ミニマル・テクノロジーズ／㈱ラクーン／Repro㈱／㈱ワコムアイティ

CONTACT
Ruby biz Grand prix URL http://rubybiz.jp

アドバンスソフトウェア、郵便番号・住所検索用開発ツール「Yubin7 for Java」を発売

アドバンスソフトウェア㈱は10月31日、郵便番号・住所検索のための開発ツール「Yubin7 for Java」を発売した。

Yubin7 for Javaは、Java EE、Java SE環境で住所・郵便番号の高速、高精度な検索を可能とする開発ツール。Javaコンポーネントとして提供される。専用の郵便番号辞書は、日本郵便から月に一度公開される郵便番号マスタを元に作成され、常に最新のデータで検索を行える。一般的な郵便番号・住所データの検索に加え大口事業所向け個別郵便番号の検索や新旧住所変換にも対応している。また数字、漢数字表記やビルの表記、カ、ガ、ケ、ヶなどの住所表記のゆらぎに対応しているほか、住所データからカスタマバーコードイメージを生成するバーコードコンポーネントも付属している。

価格は基本パッケージが34,000円、配布ライセンスは、クライアントアプリケーションの場合1ライセンス20,000円、サーバアプリケーションの場合1ライセンス年間28,000円（すべて税別）。対応OSはWindows 7（SP1以上）～、Windows Server 2008（SP2以上）～、Red Hat Enterprise Linux 6.x/7.x、CentOS、Ubuntu。対応Java環境は32bit/64bit VM JDK 7.0/8.0。

CONTACT
アドバンスソフトウェア㈱ URL http://www.adv.co.jp

BlueMeme、「OutSystems Rich Grid Component Edition」を提供開始

㈱BlueMemeは9月7日、同社が提供する高速開発基盤「OutSystems」に、グレープシティ㈱が提供するJavaScriptライブラリ「SpreadJS」および「Wijmo」の機能を組み込んだ製品「OutSystems Rich Grid Component Edition」の販売を開始した。

OutSystemsは、ポルトガルのOutSystems社が開発を行うモデル駆動型の開発基盤。ソースコードを極力書かずにWebアプリケーションを開発できるのが特徴だ。今回発表の製品は、WebアプリケーションにExcelライクなUIを実現する「SpreadJS」とHTML5対応のリッチUIを実現する「Wijmo」の機能をラッピングし、OutSystemsの開発部品として提供する。これにより、Excelのようにコピー&ペーストや計算式の取り込みができるWebアプリケーションや、高度なグラフ表示を高速に読み込めるUI画面をOutSystemsでコードを書かずに簡単に実現できる。

大量データの表示やダッシュボードによる経営管理、Visual Basicで構築した社内システムのグリッドインターフェースをそのまま使う場合などの利用を想定している。

CONTACT
㈱BlueMeme URL http://www.bluememe.jp
グレープシティ㈱ URL http://www.grapecity.com/jp

Service

ネオジャパン、「desknet's NEO電子ペーパー設備予約表示システム」を販売開始

㈱ネオジャパンは、大日本印刷㈱と協業し、グループウェア「desknet's NEO」と大日本印刷が提供する「DNP電子ペーパー会議室予約表示システム」とを連携させた「desknet's NEO電子ペーパー設備予約表示システム」を開発した。11月27日から販売を開始する。

従来より、会議室などの共用設備の利用状況の把握と稼働率の向上は、企業の管理部門の課題の1つである。本システムは電子ペーパー端末とグループウェアを連携させ、次のことを実現することでこの課題を解消する。

- desknet's NEOで登録した会議室の予約状況を、会議室の前に設置した電子ペーパー端末上に表示
- 会議室の入退出時に電子ペーパー端末の使用開始・終了ボタンを押すことで利用状況を更新。同時にdesknet's NEOの設備予約画面にも利用状況を反映
- 予約時間になっても未使用状態が続く場合、自動でdesknet's NEO上の予約をキャンセルすることも可能

電子ペーパー端末を使うことで配線・電源工事が不要になり低コストで導入できるのが特徴。会議室以外に社用車や学校の教室の管理などへの利用も見込んでいる。

CONTACT
㈱ネオジャパン URL https://desknets.com

Report

初の日本開催、「openSUSE.Asia Summit 2017」レポート

10月21日、22日、電気通信大学（東京都調布市）において「openSUSE.Asia Summit 2017」が開催された。

openSUSEはサーバ、デスクトップなど多目的な用途に使えるLinuxディストリビューションで、SUSE社のSUSE Linux Enterpriseのベースにもなっている。

openSUSE.Asia Summitはこれまでに北京、台湾、ジャカルタで行われており、4回目となる今回は初の日本での開催で、2日間で延べ200人が来場した。

オープニング

オープニングにて、壇上に立ったのは本イベントの委員長を務める武山文信氏。来場者への感謝の言葉を述べたあと、過去に行われたopenSUSE.Asia Summitを振り返りつつ、今回の東京でのイベントの3つの目的「①日本でopenSUSEに注目を集めさせ、アジアのコミュニティに熱狂をもたらす」「②アジアのコミュニティメンバーの活動範囲を全世界に拡大する」「③openSUSE.Asia Summitを真のアジア／グローバルなサミットにする」について語った。あわせて、武山氏自身も所属している日本のopenSUSEコミュニティについて紹介が行われた。

◀オープニングでは2016年の同イベントのリーダーを務めたEstu Fardani氏（左）からopenSUSE.Asia Summitの思い出を記録した冊子が武山氏（右）に手渡された

基調講演

オープニングに続き、Richard Brown氏の基調講演が行われた。これは「A Reintroduction」と題した発表で、openSUSEや、openSUSEプロジェクトのことをあらためて紹介するもの。プロジェクトにおけるルールや運営方法、SUSE社とプロジェクトの関係、openSUSE内の2つのディストリビューション「Leap」と「Tumbleweed」の違い（Leapは安定したシステムを希望する人向けに定期的なリリースを行うもの。Tumbleweedはより新しい機能を希望する人向けにローリングリリースを行うもの）などについて語られた。

◀基調講演では、openSUSEプロジェクトのチェアマンであるRichard Brown氏が発表を行った

CONTACT
openSUSE.Asia Summit 2017
URL https://events.opensuse.org/conference/summitasia17

ひみつのLinux通信

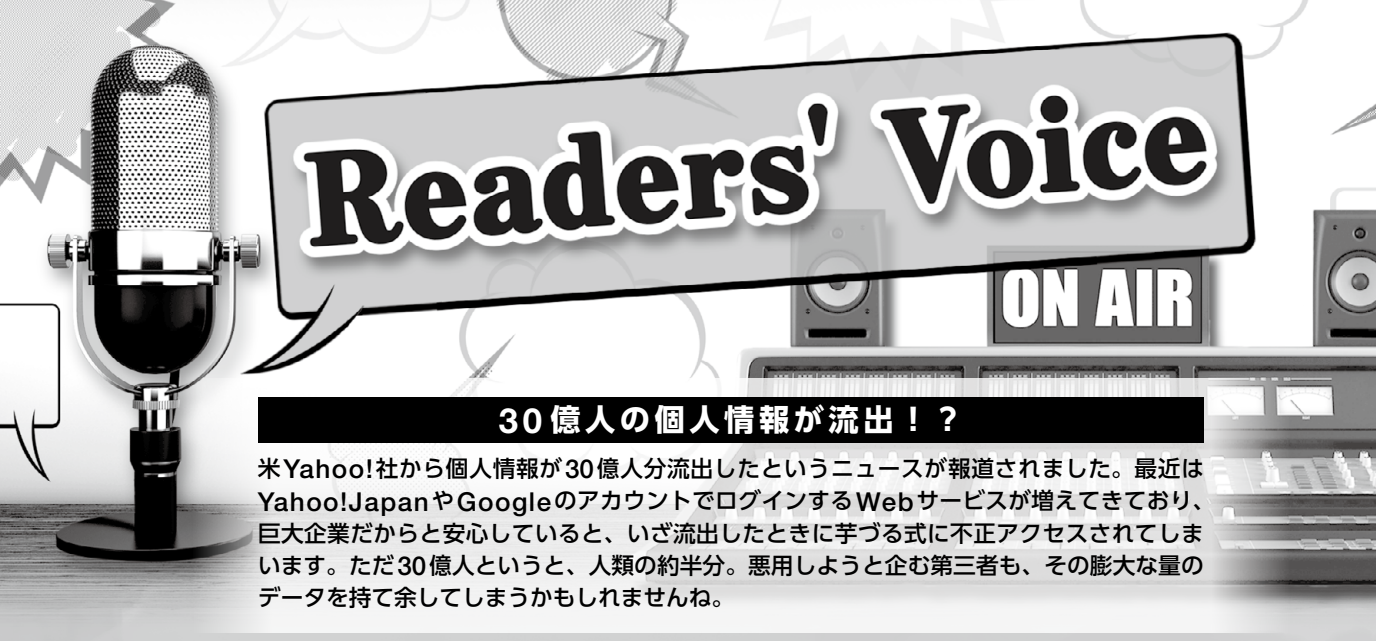
作)くつなりょうすけ
@ryosuke927

第46回 ほしいデーモン

残業が多くて、実店舗に買いに行けない担当編集はWeb通販ばかり。ええ、ウチの編集部はそんな人ばかりですよッ！ 働き方改革って何でしょね？

to be Continued

すっかり浸透した感があるsystemdですが、その機能はinitだけにとどまらず、crondやntpdなども持っています。そのうちsystemdってOSになりそうですね。「d」と言えばデーモン、常時稼働して必要なときに通知やメッセージの発行を行ってほしいものが日常生活には溢れています。某通販サイトなどは定期的に消耗品を送るようにできたりしてデーモン化してるなー、と感じます。ボタン1つで商品が来ちゃうとか、もう依存しちゃいますよね。うん、すでに依存してますね。便利って、すごいですよね。筆者の環境ではそろそろ「花粉症デーモン」が起動します。コレ動かないほうがいいんですが、メンテナンスしてないわりにはちゃんと動くんですよねぇ(涙)。



30億人の個人情報が流出！？

米Yahoo!社から個人情報が30億人分流出したというニュースが報道されました。最近はYahoo!JapanやGoogleのアカウントでログインするWebサービスが増えてきており、巨大企業だからと安心していると、いざ流出したときに芋づる式に不正アクセスされてしまいます。ただ30億人というと、人類の約半分。悪用しようと企む第三者も、その膨大な量のデータを持て余してしまうかもしれませんね。

2017年10月号について、たくさんの声が届きました。

第1特集 Gitのキホン

GitHub/Bitbucket/GitLab/GitBucketという4つのGitリポジトリサービスの比較から、基本操作、内部構造、Vim・AtomからのGit操作、さらには導入事例まで、Gitを使いこなすうえでの基本知識を紹介しました。

エディタからGitを使う解説が非常にためになりました。普段からVimを使っているので参考にさせていただきました。
村橋さん／北海道

Git、今の仕事先で普及させたい。
うたさん／大阪府

Gitには一度触れ、難しかったので諦めていたが、今回の記事を読んでだいぶわかるようになった。
nopazoさん／神奈川県

GitLabというものがあることを知れてよかったです。Bitbucketと併せて、GitHub以外の選択肢として試してみようと思いました。
えぽきしさん／神奈川県

Gitの特集は、たいへん役立ちました。今まで何回か挑戦しようとしていましたが、なかなか踏み切れなくて……。Gitサービスの選び方もたいへん参考になります。ありがとうございます。
山根さん／京都府

広く普及しているGitですが、まだまだ手が出ないといった方も多いはず。特集では基礎を中心に幅広い話題を扱ったので、そんな初心者の方にも役立てたようです。とくにGitのリポジトリサービス比較記事が助かったという声が多かったです。

第2特集 脆弱性スキャナVuls入門

Vulsは、OSのパッケージやソフトウェアに含まれる脆弱性を検知・可視化してくれる脆弱性スキャナです。そんなVulsについて、開発の発端、使い方、運用への組み込み方を、開発者たち自らが解説した特集です。

世の中を良くするためのソフトウェアと、それを作ることを認めてくれる会社は、すばらしいと思った。
卯月さん／埼玉県

Vulsは初めて知りましたが、OpenVASなどの脆弱性スキャンツールについて以前から興味があったので、使用してみたいと思います。
野島さん／東京都

Vulsの機能もすごいですが、とくに第4章のVuls誕生秘話に感動しました。
psiさん／東京都

普段脆弱性スキャナなどのセキュリティツールは使っていないが、今後Vulsだけでなく、NessusやOpenVASなどの類似ツールについても特集があると良いなと感じた。
ひよこ大佐さん／東京都

脆弱性の管理に課題を感じている現場が多いようで、興味を持っていた、初めて知って使いたいと思った、という声が寄せられました。開発者の熱い思いが綴られた第4章に胸を打たれたという読者の方も多かったです。

一般記事　素材集サービスを効率化した自動画像管理システムに学ぶ

注目のアーキテクチャ「サーバーレス」を実際の業務で採用したピクスタに、Amazon Lambda、API Gateway、Serverless Frameworkを使った実際の開発の流れを解説してもらいました。

画像管理が適当になっていることが気

10月号のプレゼント当選者は、次の皆さまです

①ネットワークストレージサーバ「F2-220」
E・R様（東京都）

②薄型モバイルバッテリー「世界のトランプ」
山添淳一様（東京都）、大山晃平様（東京都）

③『ねこ手帳 2018』
卯木輝彦様（千葉県）、與喜好様（大阪府）

④『TensorFlow機械学習クックブック』
組田隆亮様（東京都）、井上和幸様（愛知県）

⑤『暗号技術のすべて』
鈴木涼太様（東京都）、横山達男様（東京都）

⑥『確かな力が身につくC#「超」入門』
長富和人様（大阪府）、川澄良雄様（埼玉県）

⑦『Mackerelサーバ監視［実践］入門』
葉山明寛様（東京都）、TATSU様（福岡県）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

になっていたので、タイムリーな内容でした。 yuipapaさん／東京都

サーバーレスというのがトレンドなのでしょうか。これは新技術を勉強せねば。 Tayuさん／千葉県

サーバーレスがもっと手軽に、一般的になったらって考えると、その分野で仕事してる人は職なしになるのかなと、実際にはそんなことにはならないでしょうけど。 サバ鼻炎さん／東京都

よく聞くようになったサーバーレスという言葉ですが、読者の間ではまだまだ認知度は低いようです。記事では、実際の業務に適用した事例を読むことができたので、実態に即した利便性を感じられたのではないでしょうか。

一般記事　ネットワークエンジニアその技術の極め方

クラウドが普及した現在、ネットワークエンジニアに求められるスキルとは何なのか、どうすれば一歩先を行くネットワークエンジニアにはなれるのかについて、その道の熟練者がアドバイスしました。

実機の手配がたいへんですね。GNS3のようなエミュレーション環境が整備されることを期待したいです。 tack41さん／愛知県

実験環境の紹介がおもしろかった。 たくちゃむさん／群馬県

あらためてエンジニアに必要なものを意識。 北野さん／大阪府

なんちゃってネットワーク管理者な自分にとって、スキルアップは必須課題であり、とても役に立った。 くたくたさん／佐賀県

Solarisを自宅に置いていた先輩を思い出しました。 けどスマホは嫌だよさん／奈良県

自身のスキルアップに悩んでいるエンジニアさんが多いようですが、記事では実際に手を動かして学ぶというところにそのチャンスがあると書いていましたね。自宅にネットワーク環境を構築するのは、ハードルが高いぶん、得られるものも大きそうです。

一般記事　身近なメールで使われている技術をネットワークコマンドで体験してみよう

ネットワークコマンドTelnetを使ったメールサーバへの送信リクエストを通じて、メール技術やクライアント／サーバのしくみについて勉強しました。

昔、会社でメール障害が起きたときに、コマンドでメールサーバにアクセスしていろいろ調べたことを思い出しました。 にわとりさん／東京都

理解できているだろうと思っていましたが、記事を読んでハッとするところがあり、参考になりました。 吉良さん／東京都

まずは体験したくなりました。 パルさん／和歌山県

普段使っている便利なサービスも、基本的なコマンドが駆使されて開発されていることがよくわかりました。便利に使える反面、内部で起きていることには疎くなりがちですので、手元で試せる本記事は受けが良かったです。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
January 2018

2018年1月号
定価（本体1,220円+税）
184ページ

12月18日発売

［第1特集］恒例・シェル芸で年越し
使えるシェルスクリプトの書き方
手作業の自動化を極めて今年はラクチンな年に!

［第2特集］
文字コード・トラブルシューティング［Web編］
なぜ文字化けするのか？　なぜ絵文字がうまくでないのか?

［特別企画］
・SaaS版もリリース!　脆弱性スキャナVulsの来襲!
・ARKitとUnityで作るiPhone ARアプリ集中特講

※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。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2017年10月号
●P.75　第2特集第2章「Vuls導入チュートリアル」、図1
［誤］apptitude changelog　　［正］aptitude changelog
●P.163　Linuxカーネル観光ガイド　P.165
図6に間違いがありました。詳細は以下のサポートページをご覧ください。
http://gihyo.jp/magazine/SD/archive/2017/201710/support

SD Staff Room

●娘が古いiPadを使っている。OSが更新されないので新しいアプリがうまく動かないときがある。メモリのクリア方法を何気なくやったら、それを覚えてしまった。壁紙の変更も自分でやる。お気に入りの画像をダウンロードしてカスタマイズしている。ますますインドア派になる5歳児。大丈夫かな。（本）

●iPad miniの新製品を待ち望んでいたのですが、なかなか発売されず、大きさからしてもう少し小さくても良いかなと思い、iPhone Xを注文しました。というもっともらしくない言い訳で、家人の追求を逃れる所存であります。（最近Android端末何台持っているか聞かれ、その多さに即答できなかった幕）

●Nintendo Switch発送のお知らせがさっき届きました!　注文予約が取れたあと、はやる気持ちを抑え……きれずに追加コントローラやBreath of the Wildなどを先行購入。子どもとコントローラをカチャカチャやってイメトレ済み!　あ、でもまだ「風のタクト」終わってないや。（キ）

●自己投資にお金はケチらない。そう思っていても、高い価格の本などを買うときには迷いが生じます。そこで、小遣いが支給されたらその一部ですぐに図書カードを買うことにしています。その額はもう本に使うしかないわけです。お金の用途を限定しても意外と不便はなく無駄遣いも減りました。（よし）

●クロスバイク走行記＜第3弾＞。今回は荒川サイクリングロードを熊谷手前まで往復100km!　信号機や車の通りがほぼないので、快適に走れました。驚いたのが、首都圏から40～50km走っただけで風景がかなり本格的な田園地帯に変わったこと。東京の一極集中を身をもって経験できました。（な）

●もうすぐクリスマスの季節がやってきますね。大人も子どももわくわく。街はイルミネーションで彩られ、出歩くのも楽しい。そんな12月の初めに、『祈りの光』をテーマにした企画展にちびぬいで参加することになりました。窓の灯りのように温かく、きらきらと心が輝くようなものをお届けできますように!（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2017年12月号

発行日
2017年11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
松井竜馬
大橋　涼
目良直子

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
法人営業課（広告）
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。